Homotopy Type Theory in Lean

Floris van Doorn

Carnegie Mellon University
github.com/leanprover/lean2

11 July 2017

Synthetic Homotopy Theory

In homotopy type theory, the types are interpreted as *spaces*.

This leads to a new program, *synthetic homotopy theory*:

> To study types in type theory as spaces in homotopy theory.

This method for homotopy theory is

- more general than the classical method;
- constructive;
- easier to formally verify in a proof assistant

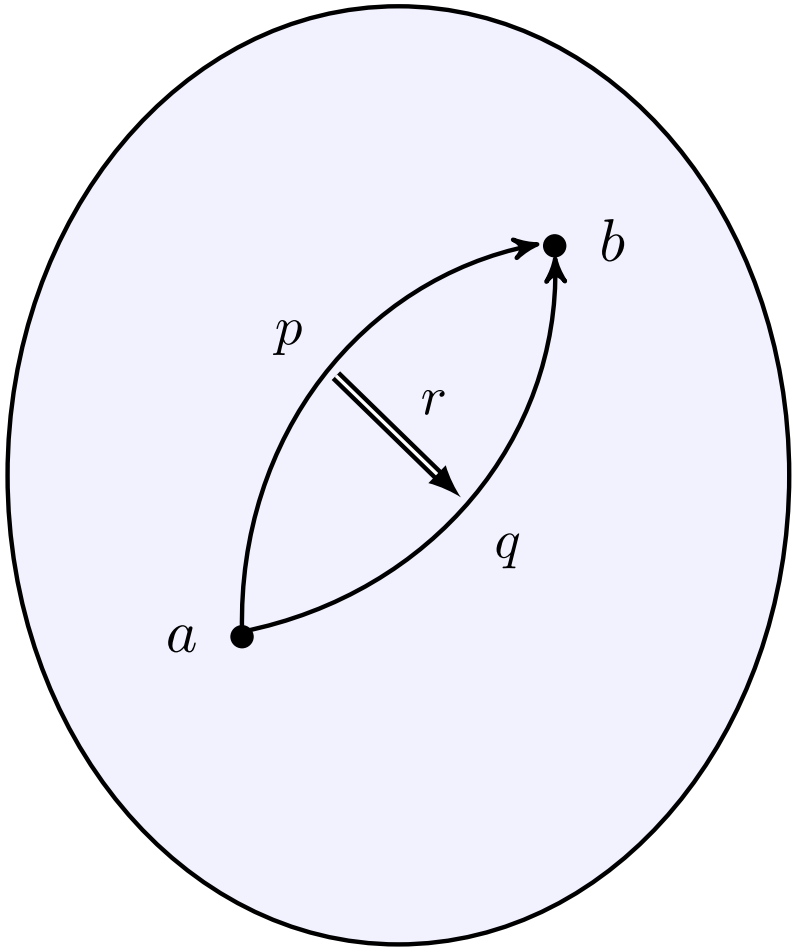
We use the *Univalence axiom* [Voevodsky] and *higher inductive types* [Shulman, Lumsdaine].

Types as spaces

A type A can have

- points $a, b : A$
- paths $p, q : a = b$
- paths between paths $r : p = q$

$\vdots$



Identity Type

The identity type is central in homotopy type theory.

It corresponds to *equality* in logic and to the *path space* in homotopy theory.

Different ways to think about the identity type:

- **Type theory**: The identity type $a =_A (-)$ is generated by reflexivity: $\mathsf{refl}_a : a =_A a$.
- **Logic**: Equality is the least (free) reflexive relation.
- **Homotopy theory**: The path space with one point fixed is contractible.

(This does not mean every proof of equality is reflexivity)

Path Induction

This is made precise by *path induction*:

- If $C : \Pi(x : A),\ a = x \to \mathrm{Type}$,
- to prove/construct an element of $\Pi(x : A).\Pi(p : a = x),\ C(x, p)$
- it is sufficient to prove/construct an element of $C(a, \mathsf{refl}_a)$

What this means is that if we have a path $p : a = x$ where the right endpoint is a variable, we may assume that p is reflexivity and that x is a.

Path Induction

Example If A is a type with points x, y and z. If $p : x = y$ and $q : y = z$, we have a *concatenation* $p \cdot q : x = z$.

Path Induction

Example If A is a type with points x, y and z. If $p : x = y$ and $q : y = z$, we have a *concatenation* $p \cdot q : x = z$.

Proof Since the right endpoint of q is a variable, we may assume q is reflexivity and that y is z. Then we need to construct $p \cdot \mathsf{refl}_y : x = y$, which we define as $p \cdot \mathsf{refl}_y := p$.

Path Induction

Example If A is a type with points x, y and z. If $p : x = y$ and $q : y = z$, we have a *concatenation* $p \cdot q : x = z$.

Proof Since the right endpoint of q is a variable, we may assume q is reflexivity and that y is z. Then we need to construct $p \cdot \mathsf{refl}_y : x = y$, which we define as $p \cdot \mathsf{refl}_y := p$.

```
variables {A : Type} {w x y z : A}
definition concat (p : x = y) (q : y = z) : x = z :=
by induction q; exact p
```

Path Induction

Example If A is a type with points x, y and z. If $p : x = y$ and $q : y = z$, we have a *concatenation* $p \cdot q : x = z$.

Proof Since the right endpoint of q is a variable, we may assume q is reflexivity and that y is z. Then we need to construct $p \cdot \mathsf{refl}_y : x = y$, which we define as $p \cdot \mathsf{refl}_y := p$.

```
variables {A : Type} {w x y z : A}
definition concat (p : x = y) (q : y = z) : x = z :=
by induction q; exact p

definition con.assoc (p : w = x) (q : x = y) (r : y = z) :
  (p · q) · r = p · (q · r) :=
by induction r; reflexivity
```

Path Induction

Example If A is a type with points x, y and z. If $p : x = y$ and $q : y = z$, we have a *concatenation* $p \cdot q : x = z$.

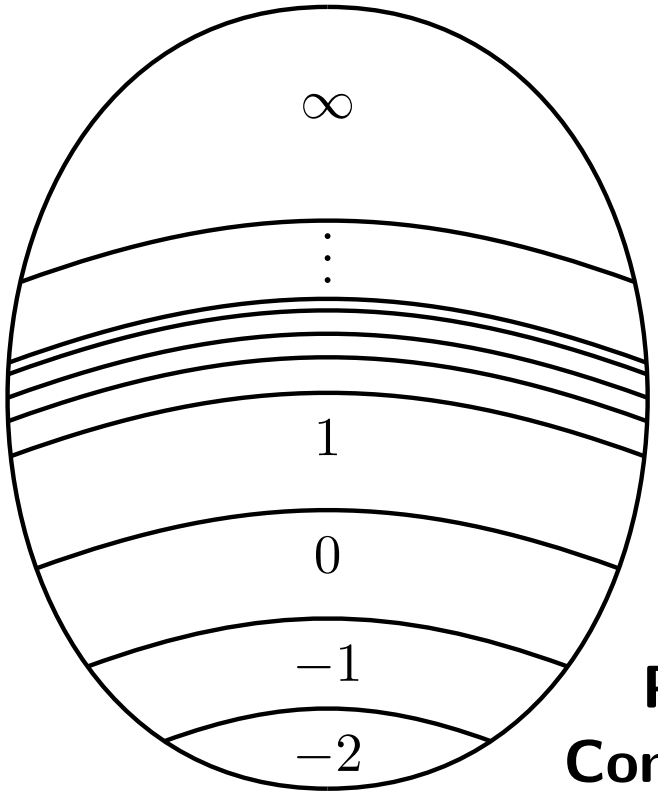
Proof Since the right endpoint of q is a variable, we may assume q is reflexivity and that y is z. Then we need to construct $p \cdot \mathsf{refl}_y : x = y$, which we define as $p \cdot \mathsf{refl}_y := p$.

```
variables {A : Type} {w x y z : A}
definition concat (p : x = y) (q : y = z) : x = z :=
by induction q; exact p

definition con.assoc (p : w = x) (q : x = y) (r : y = z) :
  (p · q) · r = p · (q · r) :=
by induction r; reflexivity
```

After induction on `r`, the goal is `(p · q) · refl y = p · (q · refl y)`

Truncated Types (Voevodsky)



$(n+1)$-**Type**: all paths n-types

1-Type: all paths are sets

Set: satisfies UIP / axiom K

Proposition: as at most one point

Contractible: has exactly one point

Truncation

Given A, we can form the *n-truncation* $\|A\|_n$.

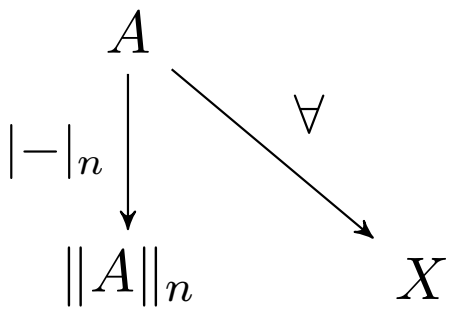
$\|A\|_n$ is the "best approximation" of A which is n-truncated.

$$\begin{array}{ccc} A & & \\ & \searrow^{|-|_n} & \\ & & \|A\|_n \end{array}$$

Truncation

Given A, we can form the *n-truncation* $\|A\|_n$.

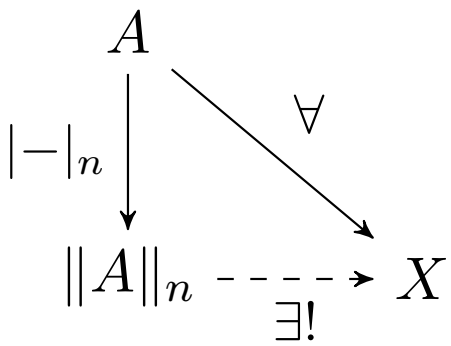
$\|A\|_n$ is the "best approximation" of A which is n-truncated.



Truncation

Given A, we can form the *n-truncation* $\|A\|_n$.

$\|A\|_n$ is the "best approximation" of A which is n-truncated.



Fundamental group

Given a pointed type (A, a_0). The type $\Omega A := (a_0 = a_0, \mathsf{refl}_{a_0})$ has

- multiplication given by concatenation
- inverses given by path inverses
- identity given by reflexivity

These operations satisfy the group laws.

It is not quite a group since it has higher structure. For example, there might be multiple proofs that $(p \cdot q) \cdot r = p \cdot (q \cdot r)$.

Taking the set-truncation "kills off" this higher structure. Therefore, $\pi_1(A) := \|\Omega A\|_0$ forms a group, the *fundamental group*.

The higher homotopy groups are $\pi_n(A) := \|\Omega^n A\|_0$.

Fundamental group

Traditionally, the definition of the fundamental group of (A, a_0) consists of continuous functions $f : [0, 1] \to A$ such that $f(0) = f(1) = a_0$ modulo homotopy.

In HoTT, we don't need to define real numbers, the interval, continuous functions or homotopic of paths to define the fundamental group.

Fundamental group

In HoTT, the fundamental group can be formalized in less than 100 lines of code.

Contrast with [Bohua Zhan, ITP 2017], using powerful automation:

> *Starting from the axioms of set theory, we formalized the definition of the fundamental group, as well as many other results in set theory, group theory, point-set topology, and real analysis. The entire development contains over 13,000 lines of theory files and 3,500 lines of ML code, taking the author about 5 months to complete.*

Fundamental group

Traditionally, if you want to define an operation on the fundamental group, you have to prove that the operation does the same on homotopic paths.

In HoTT all functions automatically respect homotopic paths, since they are equal.

Higher Inductive Types

In Type Theory there are *inductive types*, in which you specify its points.

Examples. $\mathbb{N}$ is generated by 0 and succ
$A + B$ is generated by either $a : A$ or $b : B$
$a =_A (-)$ is generated by $\mathrm{refl}_a : a =_A a$

In homotopy theory we can build cell complexes inductively.

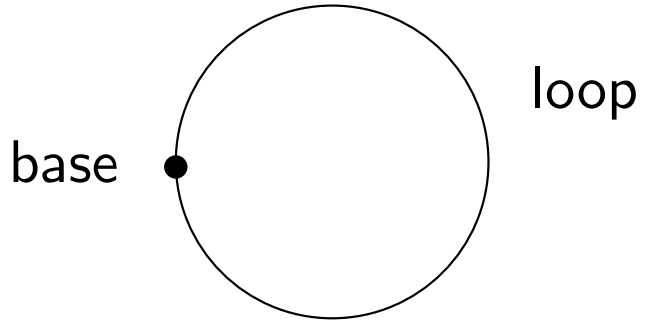
In HoTT we can combine these into *higher inductive types* [Shulman, Lumsdaine, 2012].

The circle

Example. The circle $\mathbb{S}^1$

$\texttt{HIT}\ \mathbb{S}^1 :=$

- base : $\mathbb{S}^1$
- loop : base = base



Recursion Principle. To define $f : \mathbb{S}^1 \to A$ we need to define $f(\text{base}) : A$ and a path $f(\text{base}) = f(\text{base})$ which is the path showing that f respects loop.

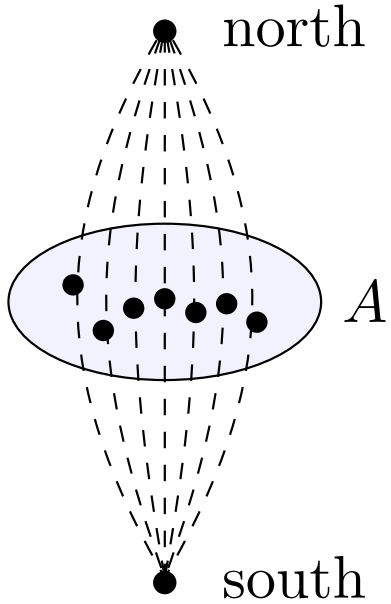
Using univalence, we can prove $\pi_1(\mathbb{S}^1) = \mathbb{Z}$.

The suspension

Example. The suspension ΣA

HIT $\Sigma A :=$

- $\text{north}, \text{south} : \Sigma A$
- $\text{merid} : A \to (\text{north} = \text{south})$



Remark. $\mathbb{S}^1 \simeq \Sigma \mathbf{2}$

Definition. The n-sphere is defined by $\mathbb{S}^{n+1} := \Sigma \mathbb{S}^n$ and $\mathbb{S}^0 := \mathbf{2}$

HoTT in proof assistants

There are various proof assistants supporting HoTT

- Coq (UniMath and Coq-HoTT)
- Agda
- Lean
- cubicaltt
- RedPRL

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.

Lean has a standard and HoTT instantiation.
The HoTT instantiation doesn't work in the newest version.
The HoTT library is developed in the stable version Lean 2.
Leo will tell you more about the new features in Lean 3 tomorrow.

The Lean Theorem Prover

Notable features:

- implements dependent type theory
- written in C++, with multi-core support
- small, trusted kernel and multiple independent type checkers
- 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 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)

Lean's kernel

The kernel is very small for a dependent type theory.

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

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

Lean's elaborator

Lean has an elaborator which figures out all the information left implicit by the user.

- 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 this are compiled down to recursors:

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

definition zip : 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)
```

The HoTT library

The Lean 2 HoTT library is a large library of basics in HoTT and specialized advanced formalizations.

In the HoTT library, we add

- *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
```

The HoTT library

The HoTT library (∼33k LOC) contains

- A good library with the basics of homotopy type theory:
  Path algebra, equivalences, truncation levels, consequences of the univalence axiom, higher inductive types, pointed types.
- A category theory library
- A large library for synthetic homotopy theory

There is a separate repository[1] (∼11k LOC) for active formalization projects, which will be moved to the main library if they become (mostly) stable.

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

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

HITs in Lean

In Lean, the quotient and truncations are primitive HITs.

We define other HITs in terms of quotients and the truncation.

Not all HITs can be reduced to quotients. [Lumsdaine, Shulman, 2017]

It is not even clear what we mean by "all HITs."

However, many HITs used in practice are constructible from quotients.

- the homotopy pushout, which gives the suspension, the circle and many other HITs;
- HITs with 2-constructors, such as the torus and Eilenberg-MacLane spaces $K(G, 1)$.
- the propositional truncation [vD];
- (not formalized) n-truncations [Rijke]

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 the smash product and pointed maps.
- cohomology theory
- the Serre spectral sequence (almost!)
- WIP: Spectrification, homology theory,

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 the smash product and pointed maps.
- cohomology theory
- the Serre spectral sequence (almost!)
- WIP: Spectrification, homology theory,

Long exact sequence of homotopy groups

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

$$\vdots$$

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

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

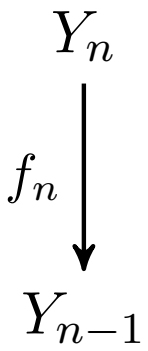
$$\pi_1(F) \xrightarrow[\pi_1(p_1)]{} \pi_1(X) \xrightarrow{\pi_1(f)} \pi_1(Y)$$

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

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

Here F is the fiber of f, i.e. $F := \Sigma(x : X),\ f(x) = y_0$.
$p_1 : F \to X$ is the first projection.

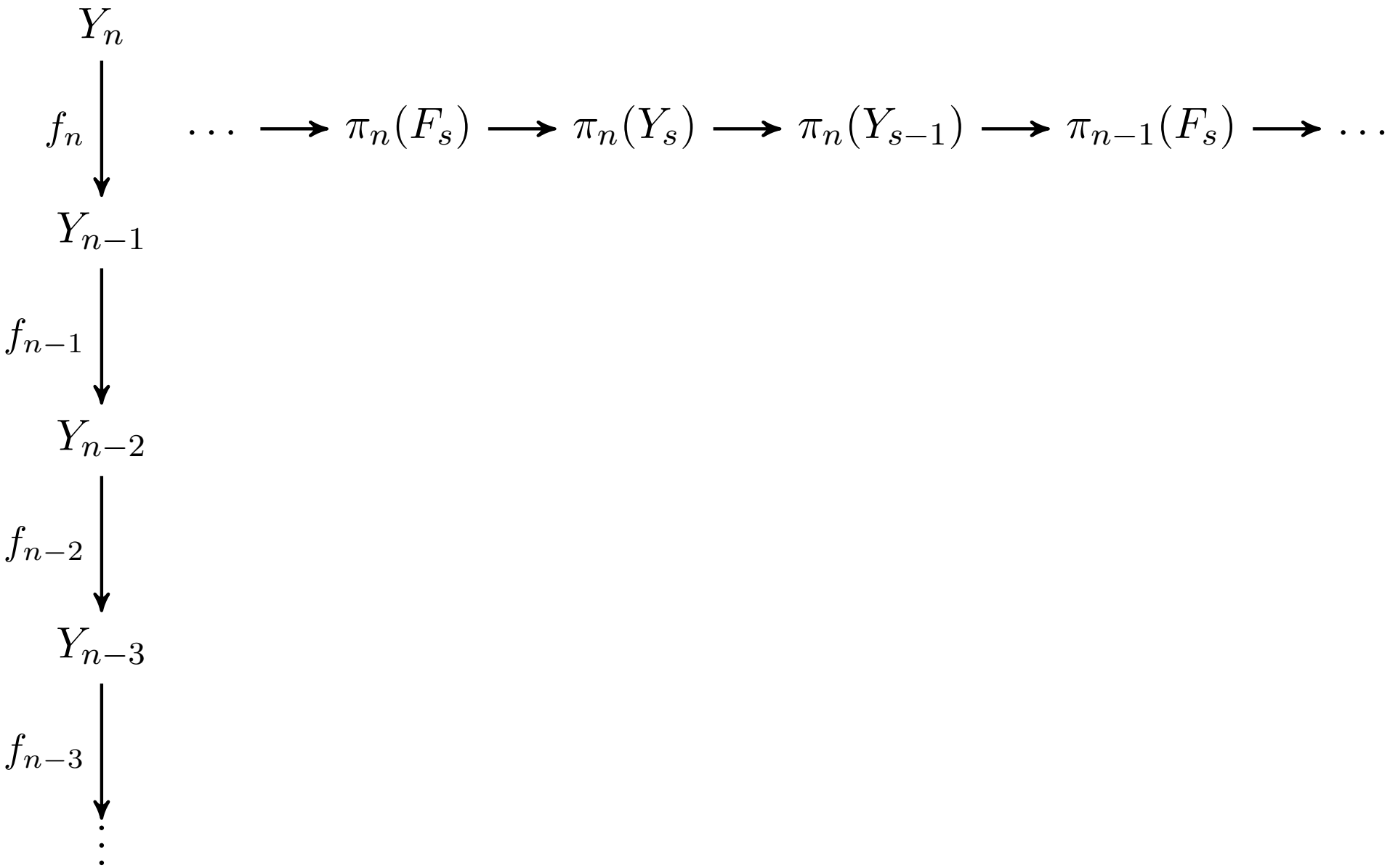
Towards Spectral Sequences



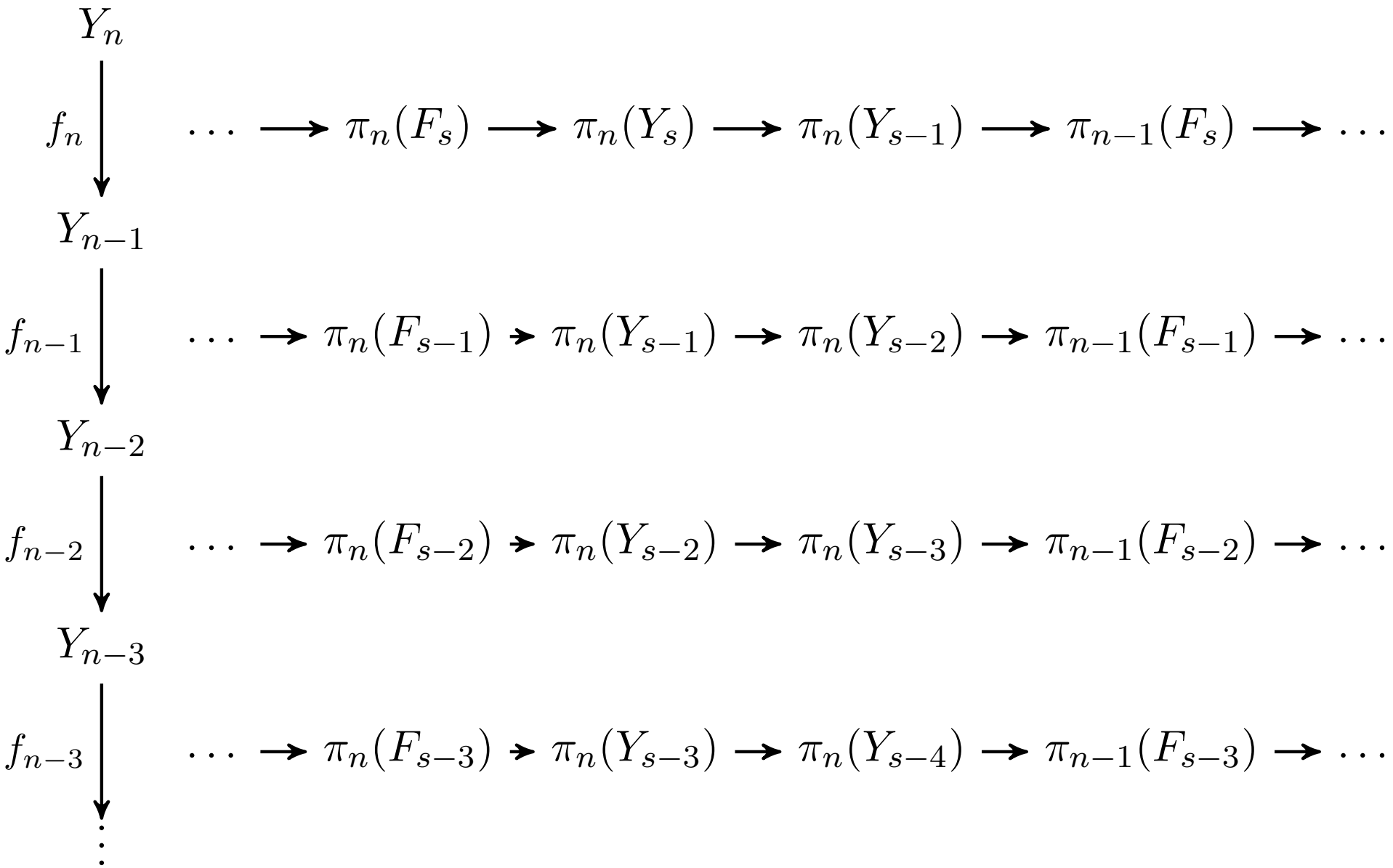
Towards Spectral Sequences

$$\begin{array}{c} Y_n \\ {\scriptstyle f_n}\big\downarrow \\ Y_{n-1} \end{array} \qquad \cdots \longrightarrow \pi_n(F_s) \longrightarrow \pi_n(Y_s) \longrightarrow \pi_n(Y_{s-1}) \longrightarrow \pi_{n-1}(F_s) \longrightarrow \cdots$$

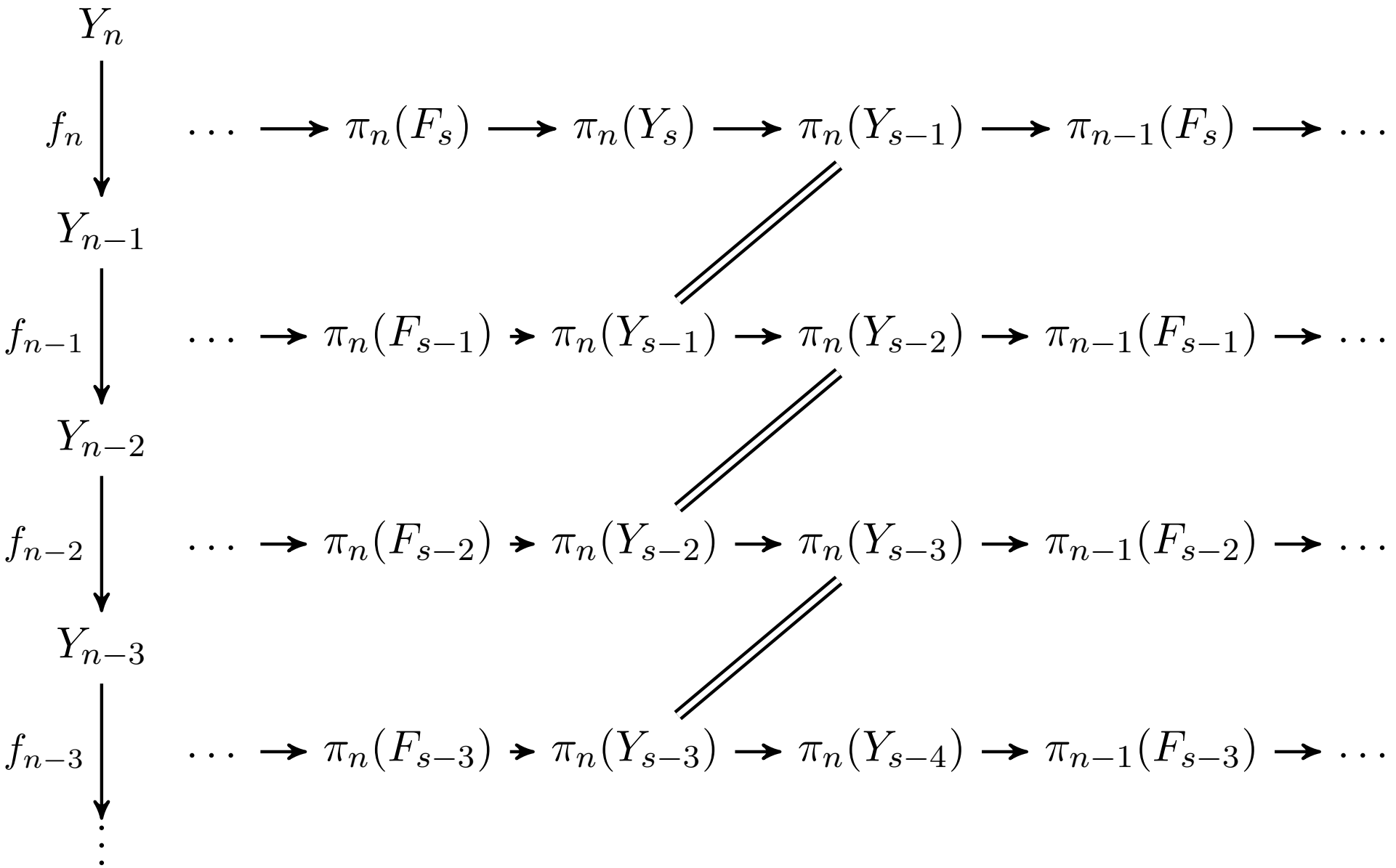
Towards Spectral Sequences



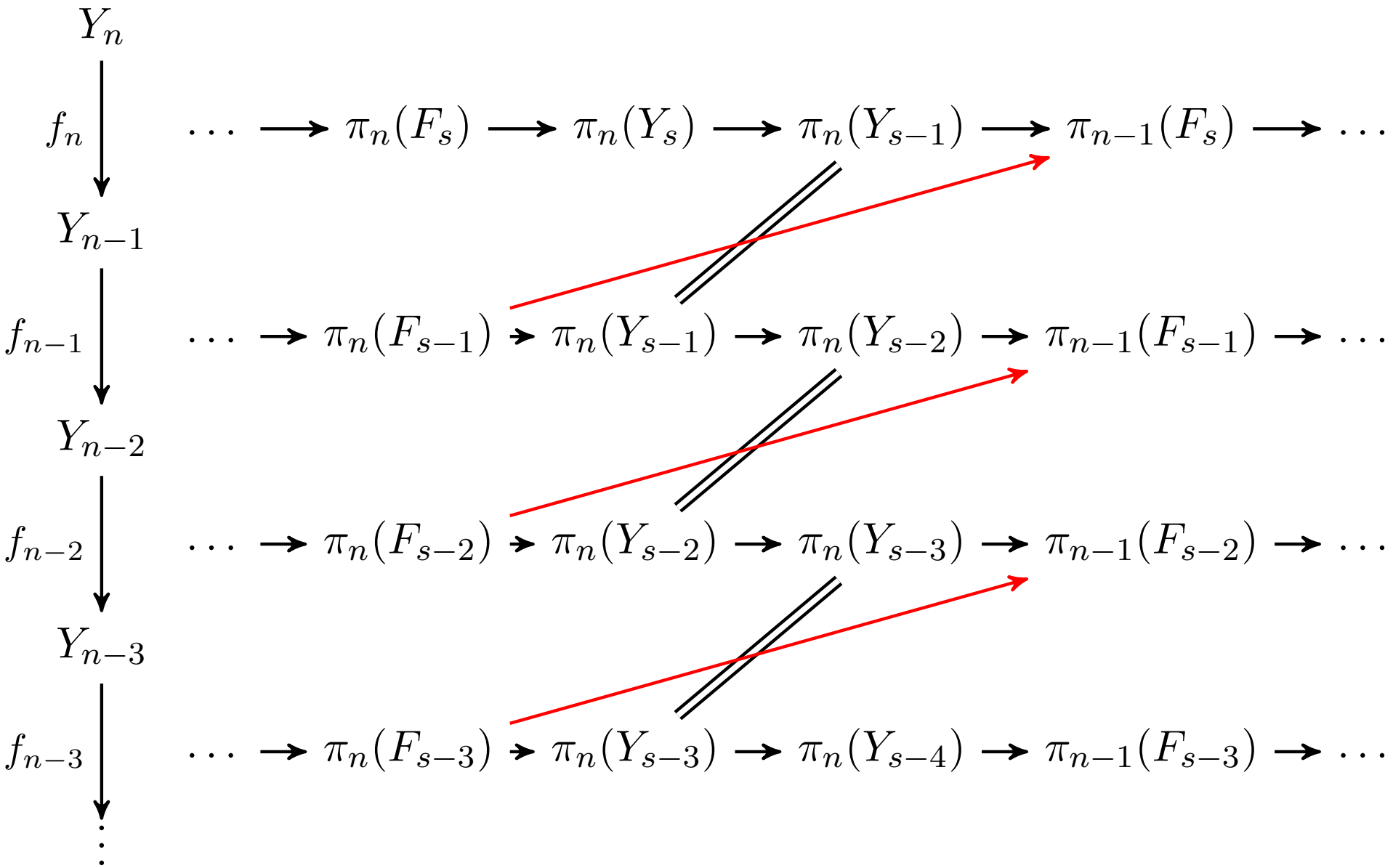
Towards Spectral Sequences



Towards Spectral Sequences



Towards Spectral Sequences



Serre Spectral Sequence

Theorem

Given a type X, a family $F : X \to \mathrm{Type}$ and an abelian group G. Then

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

This means that the cohomology group $H^{p+q}(\Sigma_{x:X}, F(x); G)$ is "built up from" $H^p(X, \lambda x. H^q(F(x); G))$ in some technical way.

(slightly modified) formulation in Lean: (proof $97+\%$ done)

```
variables {X : Type} (F : X → Type) (G : AbGroup)
definition serre_convergence :
  (λn s, Hˆ-(n-s)[X; λx, Hˆ-s[F x; G]]) ⟹[g]
  (λn, Hˆ-n[Σ(x : X), F x; G])
```

Homotopy groups of spheres

	$\mathbb{S}^0$	$\mathbb{S}^1$	$\mathbb{S}^2$	$\mathbb{S}^3$	$\mathbb{S}^4$	$\mathbb{S}^5$	$\mathbb{S}^6$	$\mathbb{S}^7$	$\mathbb{S}^8$
π_1	0	$\mathbb{Z}$	0	0	0	0	0	0	0
π_2	0	0	$\mathbb{Z}$	0	0	0	0	0	0
π_3	0	0	$\mathbb{Z}$	$\mathbb{Z}$	0	0	0	0	0
π_4	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	0
π_5	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0
π_6	0	0	$\mathbb{Z}_{12}$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0
π_7	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0
π_8	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$
π_9	0	0	$\mathbb{Z}_3$	$\mathbb{Z}_3$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$
π_{10}	0	0	$\mathbb{Z}_{15}$	$\mathbb{Z}_{15}$	$\mathbb{Z}_{24}\times\mathbb{Z}_3$	$\mathbb{Z}_2$	0	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$
π_{11}	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	$\mathbb{Z}_{24}$
π_{12}	0	0	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	0	0
π_{13}	0	0	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_2$	0

Homotopy groups of spheres

	$\mathbb{S}^0$	$\mathbb{S}^1$	$\mathbb{S}^2$	$\mathbb{S}^3$	$\mathbb{S}^4$	$\mathbb{S}^5$	$\mathbb{S}^6$	$\mathbb{S}^7$	$\mathbb{S}^8$
π_1	0	$\mathbb{Z}$	0	0	0	0	0	0	0
π_2	0	0	$\mathbb{Z}$	0	0	0	0	0	0
π_3	0	0	$\mathbb{Z}$	$\mathbb{Z}$	0	0	0	0	0
π_4	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	0
π_5	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0
π_6	0	0	$\mathbb{Z}_{12}$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0
π_7	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0
π_8	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$
π_9	0	0	$\mathbb{Z}_3$	$\mathbb{Z}_3$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$
π_{10}	0	0	$\mathbb{Z}_{15}$	$\mathbb{Z}_{15}$	$\mathbb{Z}_{24}\times\mathbb{Z}_3$	$\mathbb{Z}_2$	0	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$
π_{11}	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	$\mathbb{Z}_{24}$
π_{12}	0	0	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	0	0
π_{13}	0	0	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_2$	0

Homotopy groups of spheres

	$\mathbb{S}^0$	$\mathbb{S}^1$	$\mathbb{S}^2$	$\mathbb{S}^3$	$\mathbb{S}^4$	$\mathbb{S}^5$	$\mathbb{S}^6$	$\mathbb{S}^7$	$\mathbb{S}^8$
π_1	0	$\mathbb{Z}$	0	0	0	0	0	0	0
π_2	0	0	$\mathbb{Z}$	0	0	0	0	0	0
π_3	0	0	$\mathbb{Z}$	$\mathbb{Z}$	0	0	0	0	0
π_4	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	0
π_5	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0
π_6	0	0	$\mathbb{Z}_{12}$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0
π_7	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}\times\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0
π_8	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$
π_9	0	0	$\mathbb{Z}_3$	$\mathbb{Z}_3$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$
π_{10}	0	0	$\mathbb{Z}_{15}$	$\mathbb{Z}_{15}$	$\mathbb{Z}_{24}\times\mathbb{Z}_3$	$\mathbb{Z}_2$	0	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$
π_{11}	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	$\mathbb{Z}_{24}$
π_{12}	0	0	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	0	0
π_{13}	0	0	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_{12}\times\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_2$	0

Automation in HoTT

Automation is not used very much in Lean-HoTT (or other proof assistants).

The only automation we use is type-class inference, for example to infer that a type X is n-truncated.

It would be interesting to explore automation in HoTT.

Complications:

- HoTT is *proof-relevant*; the given proof matters in general.
- When using dependent types, computation matters a lot.

There are also many cases where the proof doesn't matter. One option is to focus on these cases.

Conclusions

- HoTT is a convenient language for homotopy theory
  - It is more general than traditional homotopy theory
  - The homotopy theoretic notions are primitives in type theory
  - It gives novel ways of reasoning
  - It is constructive (but not anti-classical)
  - It is possible to verify formally in practice
- Lean is a good proof assistant for HoTT
  - We have formalized significant results in homotopy theory