A Bunched Approach to Directed HoTT

Louise Leclerc

Samuel Mimram

September 17, 2026

Outline

1. Homotopy type theory
2. Higher categories
3. Simplicial type theory
4. The problem of hom types
5. The bunched solution
6. From theory to application

Presentation

(Homotopy) Type Theory:

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)
- Constructive flavor ($\neq$ Classical Set Theory)

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)
- Constructive flavor ($\neq$ Classical Set Theory)
- Types behaves as spaces.

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)
- Constructive flavor ($\neq$ Classical Set Theory)
- Types behaves as spaces.
- Synthetic tool for higher algebra, homotopy theory...

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)
- Constructive flavor ($\neq$ Classical Set Theory)
- Types behaves as spaces.
- Synthetic tool for higher algebra, homotopy theory...
- Suitable for formalizing mathematics / homotopy theory

Presentation

(Homotopy) Type Theory:

- Mathematical language (like Set Theory)
- Constructive flavor ($\neq$ Classical Set Theory)
- Types behaves as spaces.
- Synthetic tool for higher algebra, homotopy theory...
- Suitable for formalizing mathematics / homotopy theory
- **Goal:** replace spaces by *directed spaces*

Intuitionistic Type Theory

MLTT:

Intuitionistic Type Theory

MLTT:

- MARTIN-LÖF, 1972.

Intuitionistic Type Theory

MLTT:

- MARTIN-LÖF, 1972.
- conversely to Set Theory, no need for first order logic.

Intuitionistic Type Theory

MLTT:

- MARTIN-LÖF, 1972.
- conversely to Set Theory, no need for first order logic.
- “basic types” and “constructors” as building blocks.

Basic Types

- **empty** type: $\bot$ or $\mathbb{0}$.
  no native element or constructor.

Basic Types

- **empty** type: $\bot$ or $\mathbb{0}$.
  no native element or constructor.
- **unit** type: $\top$ or $\mathbb{1}$.
  one native element, $*$.
  $* : \mathbb{1}$.

Basic Types

- **empty** type: $\bot$ or $\mathbb{0}$.
  no native element or constructor.
- **unit** type: $\top$ or $\mathbb{1}$.
  one native element, $*$.
  $* : \mathbb{1}$.
- **booleans** type, Bool or $\mathbb{2}$.
  two native elements of this type, false and true or 0 and 1.
  $0 : \mathbb{2}$ or $1 : \mathbb{2}$.

Basic Types

- **empty** type: $\bot$ or $\mathbb{0}$.
  no native element or constructor.
- **unit** type: $\top$ or $\mathbb{1}$.
  one native element, $*$.
  $* : \mathbb{1}$.
- **booleans** type, Bool or $\mathbb{2}$.
  two native elements of this type, false and true or 0 and 1.
  $0 : \mathbb{2}$ or $1 : \mathbb{2}$.
- **natural numbers** type, $\mathbb{N}$.
  $0 : \mathbb{N}$, $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$.
  $n : \mathbb{N} \rightsquigarrow \mathsf{succ}(n) : \mathbb{N}$.
  eg., $\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(0))) : \mathbb{N}$.

Type operations

- Arrow type: $A \to B$

Type operations

- Arrow type: $A \to B$
- Cartesian product type: $A \times B$

Type operations

- Arrow type: $A \to B$
- Cartesian product type: $A \times B$
- Sum type: $A + B$

Hierarchy of Universes

- Tarski (or Grothendieck) Universes. $\mathcal{U}$ is the "type of types"
- $\mathcal{U}$ closed under $\times$, $+$ and $\to$.
- Having $\mathcal{U} : \mathcal{U}$ leads to paradoxes

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$$

recursors

How to map *out* of a given type ?

recursors

How to map *out* of a given type ?

- $\mathbb{0}$:
  for any A, $\mathsf{rec}_{\mathbb{0},A} : \mathbb{0} \to A$

recursors

How to map *out* of a given type ?

- $\mathbb{0}$:
  for any A, $\mathsf{rec}_{\mathbb{0},A} : \mathbb{0} \to A$
- $\mathbb{1}$:
  for any A and $a : A$, $\mathsf{rec}_{\mathbb{1},A,a} : \mathbb{1} \to A$
  $\mathsf{rec}_{\mathbb{1},A,a}(*) \equiv a$.

recursors

How to map *out* of a given type ?

- $\mathbb{0}$:
  for any A, $\mathsf{rec}_{\mathbb{0},A} : \mathbb{0} \to A$
- $\mathbb{1}$:
  for any A and $a : A$, $\mathsf{rec}_{\mathbb{1},A,a} : \mathbb{1} \to A$
  $\mathsf{rec}_{\mathbb{1},A,a}(*) \equiv a$.
- $\mathbb{2}$:
  for any A, and $a_0 : A$, $a_1 : A$, $\mathsf{rec}_{\mathbb{2},A,a_0,a_1} : \mathbb{2} \to A$
  $\mathsf{rec}_{\mathbb{1},A,a_0,a_1}(0) \equiv a_0$ and $\mathsf{rec}_{\mathbb{1},A,a_0,a_1}(1) \cong a_1$.

recursors

- $\mathbb{N}$:
  for any A, $a_0 : A$ and $f : \mathbb{N} \to A \to A$, $\varphi :\equiv \mathsf{rec}_{\mathbb{N},A,a,f} : \mathbb{N} \to A$.
  $\varphi(0) \equiv a_0$ and $\varphi(\mathsf{succ}(n)) \equiv f(n)(\varphi(n))$.

recursors

- $\mathbb{N}$:
  for any A, $a_0 : A$ and $f : \mathbb{N} \to A \to A$, $\varphi :\equiv \mathsf{rec}_{\mathbb{N},A,a,f} : \mathbb{N} \to A$.
  $\varphi(0) \equiv a_0$ and $\varphi(\mathsf{succ}(n)) \equiv f(n)(\varphi(n))$.
- $A \times B$:
  for any types A, B, and $f : A \to B \to C$, $\mathsf{rec}_{A\times B,C,f} : A \times B \to C$
  $\mathsf{rec}_{A\times B,C,f}(a, b) \equiv f(a)(b)$.

recursors

- $\mathbb{N}$:
  for any A, $a_0 : A$ and $f : \mathbb{N} \to A \to A$, $\varphi :\equiv \mathsf{rec}_{\mathbb{N},A,a,f} : \mathbb{N} \to A$.
  $\varphi(0) \equiv a_0$ and $\varphi(\mathsf{succ}(n)) \equiv f(n)(\varphi(n))$.
- $A \times B$:
  for any types A, B, and $f : A \to B \to C$, $\mathsf{rec}_{A\times B,C,f} : A \times B \to C$
  $\mathsf{rec}_{A\times B,C,f}(a, b) \equiv f(a)(b)$.
- $A + B$:
  for any types A, B, $f : A \to C$ and $g : B \to C$, $\mathsf{rec}_{A+B,C,f,g} : A + B \to C$
  $\mathsf{rec}_{A+B,C,f,g}(\mathsf{inl}(a)) \equiv f(a)$ and $\mathsf{rec}_{A+B,C,f,g}(\mathsf{inr}(b)) \equiv g(b)$.

More type formers

- Π-types:
  $A : \mathcal{U}, B : A \to \mathcal{U} \quad \rightsquigarrow \quad \prod_{a:A} B(a) : \mathcal{U}.$
  $f : \prod_{a:A} B(a), x : A \quad \rightsquigarrow \quad f(x) : B(x).$

More type formers

- Π-types:
  $A : \mathcal{U}, B : A \to \mathcal{U} \quad \rightsquigarrow \quad \prod_{a:A} B(a) : \mathcal{U}.$
  $f : \prod_{a:A} B(a), x : A \quad \rightsquigarrow \quad f(x) : B(x).$

- Σ-types:
  $A : \mathcal{U}, B : A \to \mathcal{U} \quad \rightsquigarrow \quad \sum_{a:A} B(a) : \mathcal{U}.$
  $x : A, u : B(a) \quad \rightsquigarrow \quad (x, u) : \sum_{a:A} B(a).$

More type formers

- Π-types:
  $A : \mathcal{U}, B : A \to \mathcal{U} \quad \rightsquigarrow \quad \prod_{a:A} B(a) : \mathcal{U}.$
  $f : \prod_{a:A} B(a), x : A \quad \rightsquigarrow \quad f(x) : B(x).$

- Σ-types:
  $A : \mathcal{U}, B : A \to \mathcal{U} \quad \rightsquigarrow \quad \sum_{a:A} B(a) : \mathcal{U}.$
  $x : A, u : B(a) \quad \rightsquigarrow \quad (x, u) : \sum_{a:A} B(a).$

- Identity-types:
  $A : \mathcal{U}, x, y : A \quad \rightsquigarrow \quad x =_A y : \mathcal{U}.$
  $x : A \quad \rightsquigarrow \quad \mathsf{refl}_x : x =_A x.$

Curry–Howard

Correspondance between types and logical propositions.

Curry–Howard

Correspondance between types and logical propositions.

- $\to$ is the implication arrow.

Curry–Howard

Correspondance between types and logical propositions.

- $\to$ is the implication arrow.
- $\times$ is the conjunction.

Curry–Howard

Correspondance between types and logical propositions.

- $\to$ is the implication arrow.
- $\times$ is the conjunction.
- $+$ is a constructive disjunction.

Curry–Howard

Correspondance between types and logical propositions.

- $\to$ is the implication arrow.
- $\times$ is the conjunction.
- $+$ is a constructive disjunction.

$$\text{Logic} \hookrightarrow \text{Algebra}$$

The identity type

- In ordinary maths, $=$ is a proposition. Now it becomes a *type*:

$$A : \mathcal{U},\ x, y : A \quad \rightsquigarrow \quad x =_A y : \mathcal{U}$$

The identity type

- In ordinary maths, $=$ is a proposition. Now it becomes a *type*:

$$A : \mathcal{U},\ x, y : A \quad \rightsquigarrow \quad x =_A y : \mathcal{U}$$

- Canonical proof: $\mathsf{refl}_x : x =_A x$

The identity type

- In ordinary maths, $=$ is a proposition. Now it becomes a *type*:

$$A : \mathcal{U},\ x, y : A \quad \rightsquigarrow \quad x =_A y : \mathcal{U}$$

- Canonical proof: $\mathsf{refl}_x : x =_A x$
- **induction principle**: to prove something for every $p : x = y$, it suffices to treat the case $y \equiv x,\ p \equiv \mathsf{refl}_x$.

The identity type

- In ordinary maths, $=$ is a proposition. Now it becomes a *type*:

$$A : \mathcal{U},\ x, y : A \quad \rightsquigarrow \quad x =_A y : \mathcal{U}$$

- Canonical proof: $\mathsf{refl}_x : x =_A x$
- **induction principle**: to prove something for every $p : x = y$, it suffices to treat the case $y \equiv x,\ p \equiv \mathsf{refl}_x$.
- Think of $x = y$ as the type of paths from x to y.

Groupoid structure on types

- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$

Groupoid structure on types

- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$

Groupoid structure on types

- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$
- **unit**: $\mathsf{refl}_x \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_y = p$

Groupoid structure on types

- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$
- **unit**: $\mathsf{refl}_x \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_y = p$
- **associativity**: $p \cdot (q \cdot r) = (p \cdot q) \cdot r$

Groupoid structure on types

- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$
- **unit**: $\mathsf{refl}_x \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_y = p$
- **associativity**: $p \cdot (q \cdot r) = (p \cdot q) \cdot r$
- **inversion**: $p \cdot p^{-1} = \mathsf{refl}_x \quad \text{and} \quad p^{-1} \cdot p = \mathsf{refl}_y$

Groupoid structure on types

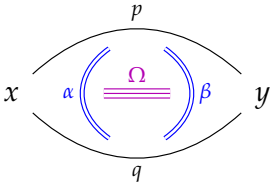
- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$
- **unit**: $\mathsf{refl}_x \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_y = p$
- **associativity**: $p \cdot (q \cdot r) = (p \cdot q) \cdot r$
- **inversion**: $p \cdot p^{-1} = \mathsf{refl}_x \quad \text{and} \quad p^{-1} \cdot p = \mathsf{refl}_y$

Theorem

Every type is equipped with a canonical structure of ∞-groupoid.

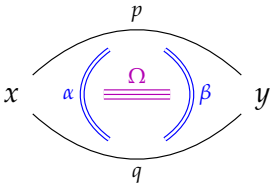
∞-groupoids

Spaces up to homotopy



∞-groupoids

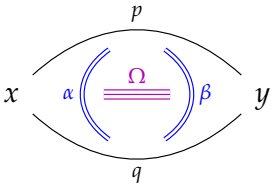
Spaces up to homotopy



$x, y : A$ $\quad p, q : x =_A y$ $\quad \alpha, \beta : p =_{x =_A y} q$ $\quad \Omega : \alpha =_{p =_{x =_A y} q} \beta$

∞-groupoids

Spaces up to homotopy



$x, y : A$ $\quad p, q : x =_A y$ $\quad \alpha, \beta : p =_{x =_A y} q$ $\quad \Omega : \alpha =_{p =_{x =_A y} q} \beta$

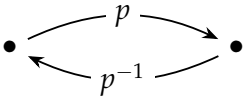
Formalisation of homotopy theory
(Feudenthal, long exact sequences, sphere homotopy groups...).

Non-directedness of HoTT

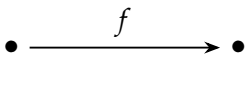
- Paths $x = y$ are always *invertible*.

Non-directedness of HoTT

- Paths $x = y$ are always *invertible*.



$x = y$



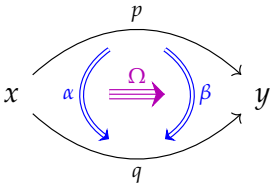
$\mathsf{hom}_A(x, y)$

∞-categories

"Directed" spaces up to homotopy

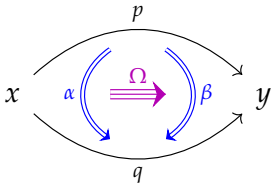
∞-categories

"Directed" spaces up to homotopy



∞-categories

"Directed" spaces up to homotopy

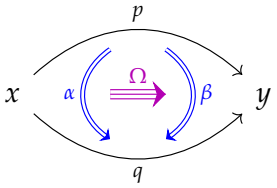


$x, y : A$ $\qquad$ $p, q : \mathsf{hom}_A(x, y)$ $\qquad$ $\alpha, \beta : \mathsf{hom}_{\mathsf{hom}_A(x,y)}(p, q)$

$$\Omega : \mathsf{hom}_{\mathsf{hom}_{\mathsf{hom}_A(x,y)}(p,q)}(\alpha, \beta)$$

∞-categories

"Directed" spaces up to homotopy



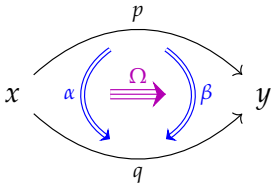
$$x, y : A \qquad p, q : \mathsf{hom}_A(x, y) \qquad \alpha, \beta : \mathsf{hom}_{\mathsf{hom}_A(x,y)}(p, q)$$

$$\Omega : \mathsf{hom}_{\mathsf{hom}_{\mathsf{hom}_A(x,y)}(p,q)}(\alpha, \beta)$$

Rewriting, concurrency,

∞-categories

"Directed" spaces up to homotopy



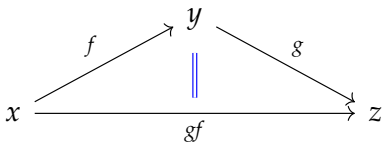
$$x, y : A \qquad p, q : \mathsf{hom}_A(x, y) \qquad \alpha, \beta : \mathsf{hom}_{\mathsf{hom}_A(x,y)}(p, q)$$

$$\Omega : \mathsf{hom}_{\mathsf{hom}_{\mathsf{hom}_A(x,y)}(p,q)}(\alpha, \beta)$$

Rewriting, concurrency,
synthetic manipulation of higher categories.

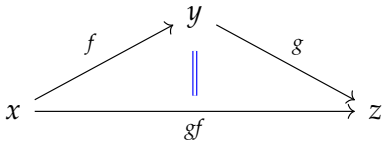
Homotopy coherence

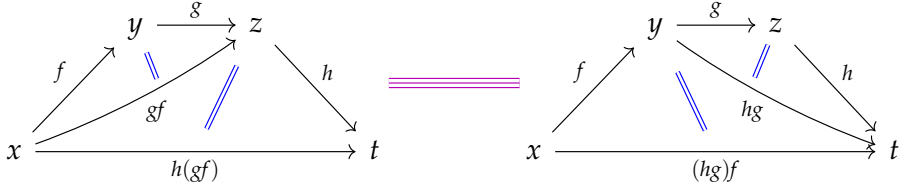
ordinary categories



Homotopy coherence

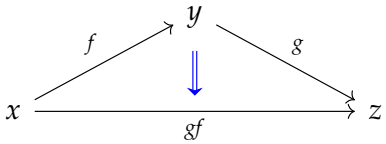
ordinary categories





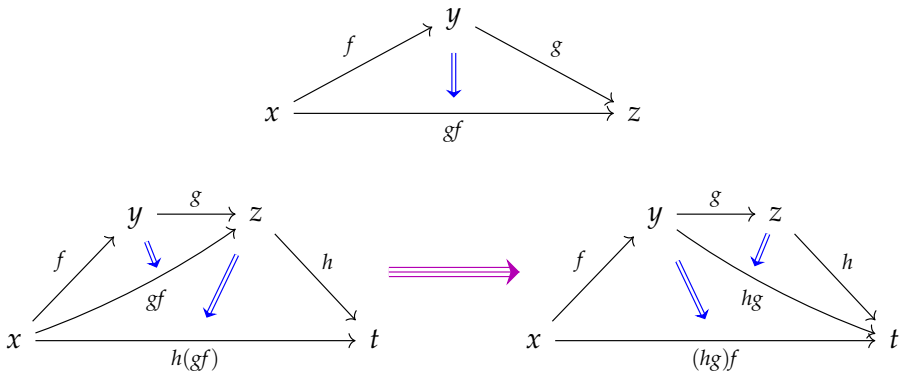
Homotopy coherence

higher categories



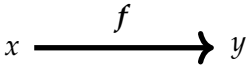
Homotopy coherence

higher categories



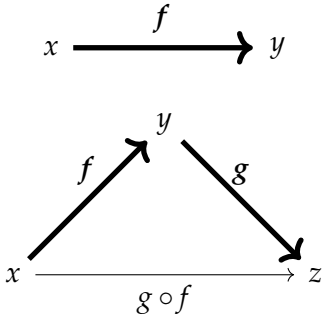
The nerve construction

Idea: categories $\rightsquigarrow$ simplicial set



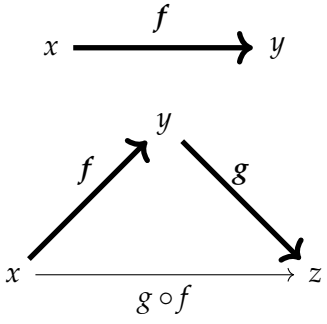
The nerve construction

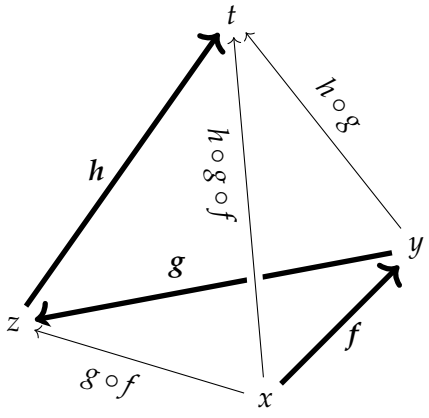
Idea: categories $\rightsquigarrow$ simplicial set



The nerve construction

Idea: categories $\rightsquigarrow$ simplicial set





Simplicial spaces

- *Space* of objects X_0
-

Simplicial spaces

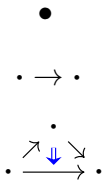
- *Space* of objects X_0
- *Space* of morphisms X_1

$\bullet$

$\bullet \to \bullet$

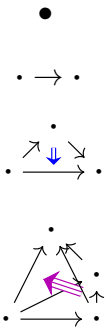
Simplicial spaces

- *Space* of objects X_0
- *Space* of morphisms X_1
- *Space* of composition witness X_2



Simplicial spaces

- *Space* of objects X_0
- *Space* of morphisms X_1
- *Space* of composition witness X_2
- *Space* of associativity witness X_3

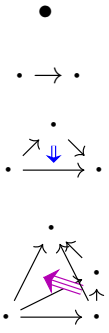


Simplicial spaces

- *Space* of objects X_0
- *Space* of morphisms X_1
- *Space* of composition witness X_2
- *Space* of associativity witness X_3

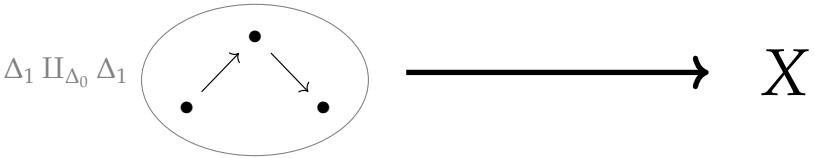
$\vdots$

Space valued presheaf $X : \Delta^{\mathsf{op}} \longrightarrow \mathcal{S}$.



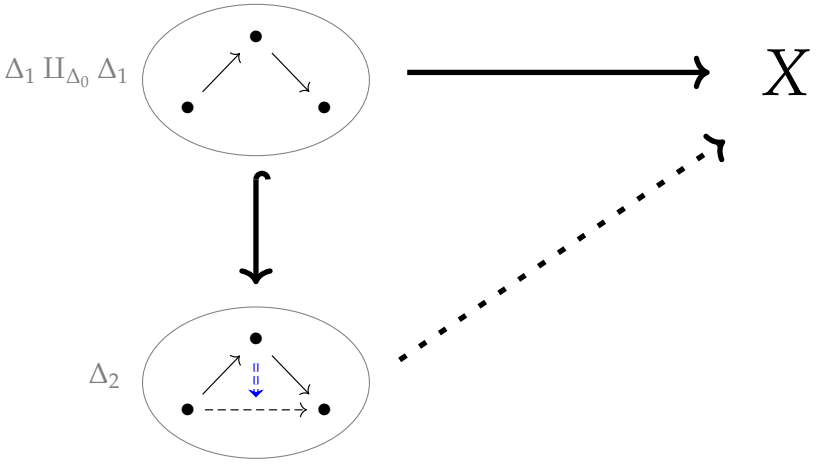
Segal spaces

Idea: encode composability as lifting properties.



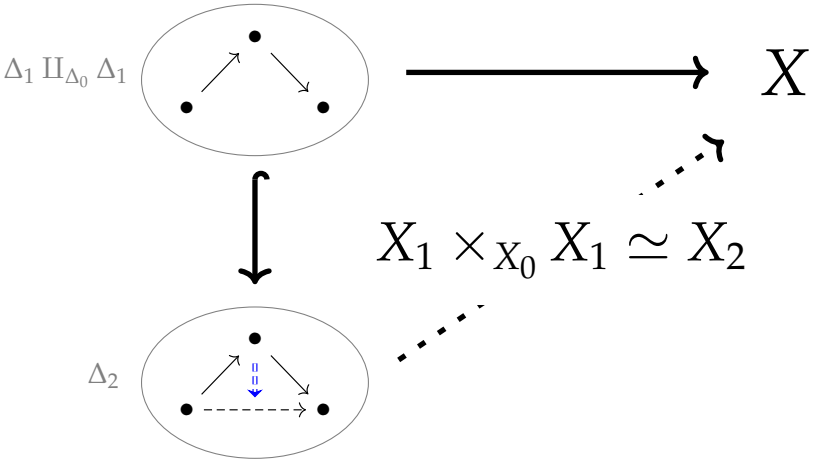
Segal spaces

Idea: encode composability as lifting properties.



Segal spaces

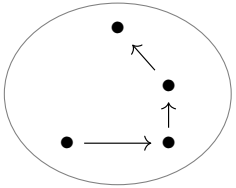
Idea: encode composability as lifting properties.

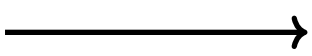


Segal spaces (2)

Idea: encode composability as lifting properties.

$\Delta_1 \amalg_{\Delta_0} \Delta_1 \amalg_{\Delta_0} \Delta_1$

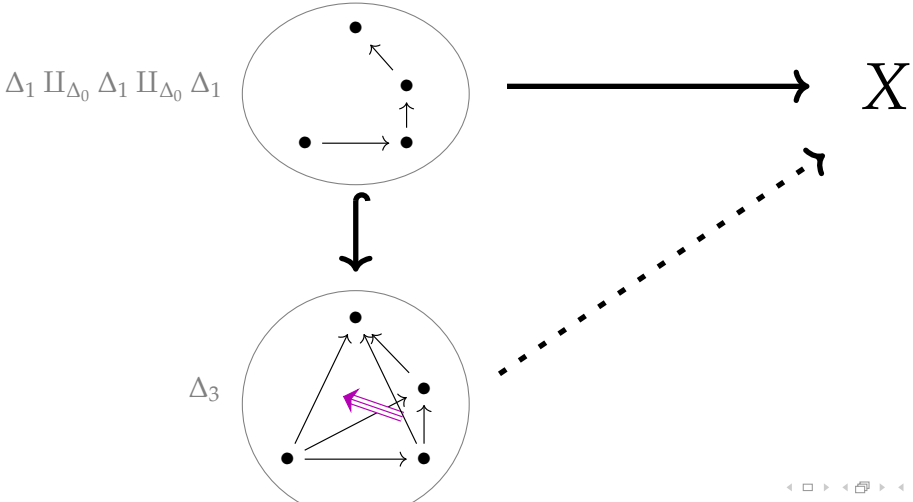




X

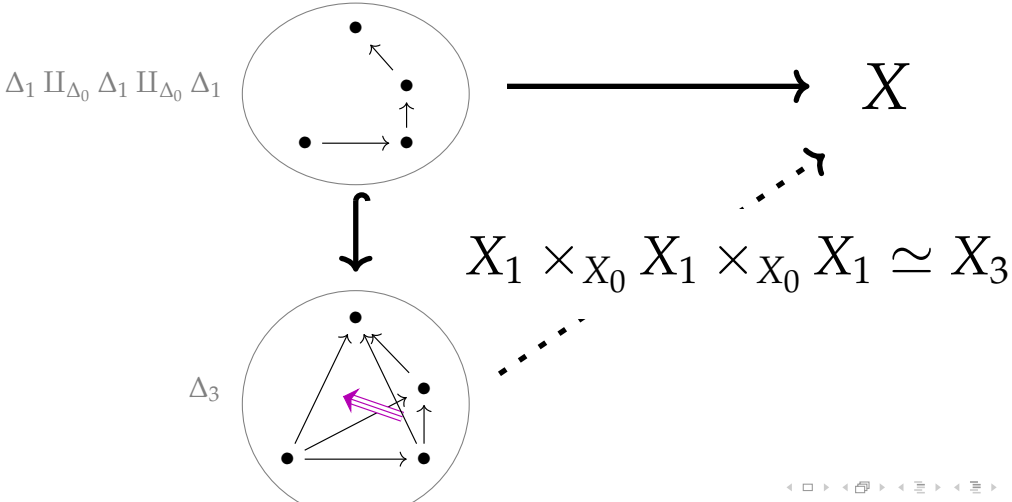
Segal spaces (2)

Idea: encode composability as lifting properties.



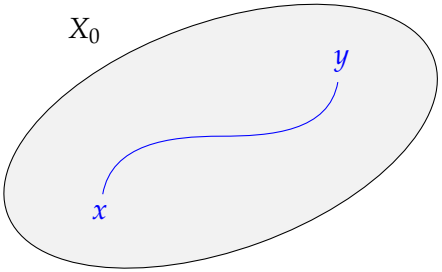
Segal spaces (2)

Idea: encode composability as lifting properties.



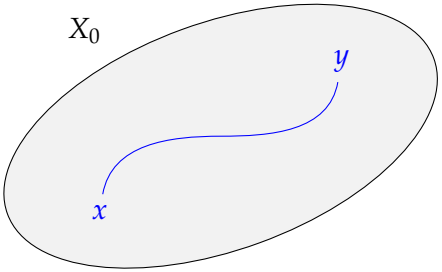
Complete segal spaces

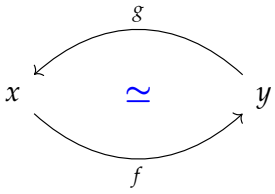
Two coexisting kinds of identifications



Complete segal spaces

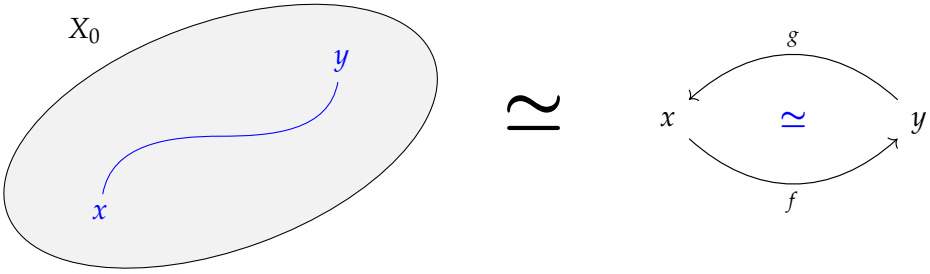
Two coexisting kinds of identifications





Complete segal spaces

Two coexisting kinds of identifications



Simplicial type theory

- Riehl, Shulman (2017); Gratzer, Weinberger, Buchholtz…

Simplicial type theory

- Riehl, Shulman (2017); Gratzer, Weinberger, Buchholtz…
- Postulate simplices as part of the syntax:

$$[\![\mathcal{U}]\!] = \mathbf{Psh}(\mathbf{\Delta})$$

Simplicial type theory

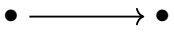
- Riehl, Shulman (2017); Gratzer, Weinberger, Buchholtz…
- Postulate simplices as part of the syntax:

$$[\![\mathcal{U}]\!] = \mathbf{Psh}(\mathbf{\Delta})$$

- Goal: $(\infty, 1)$-categories = Complete Segal spaces.

The interval

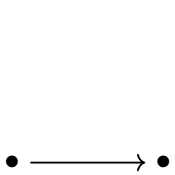
We add to the syntax a collection of shapes



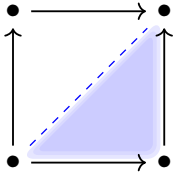
2

The interval

We add to the syntax a collection of shapes



2

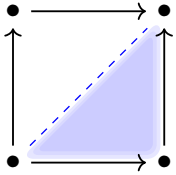


$$\Delta^2 :\equiv \{\langle t_1, t_2\rangle : 2^2 \mid t_2 \leq t_1\}$$

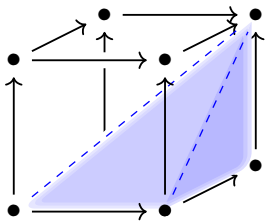
The interval

We add to the syntax a collection of shapes

2



$\Delta^2 :\equiv \{\langle t_1, t_2\rangle : 2^2 \mid t_2 \leq t_1\}$



Δ^3

Simplicial types

S shape, and X type:

$$S \to X \quad \text{is a type}$$

Simplicial types

S shape, and X type:

$$S \to X \quad \text{is a type}$$

$$\mathsf{hom}_X(x, y) = \sum_{f:2 \to X} (f(0) = x) \times (f(1) = y)$$

Simplicial types

S shape, and X type:

$$S \to X \quad \text{is a type}$$

$$\mathsf{hom}_X(x, y) = \sum_{f:2 \to X} (f(0) = x) \times (f(1) = y)$$

$$\forall n, \quad X_n :\equiv \Delta_n \to X \quad \text{is a type}$$

Simplicial types

S shape, and X type:

$$S \to X \quad \text{is a type}$$

$$\mathsf{hom}_X(x, y) = \sum_{f:2\to X} (f(0) = x) \times (f(1) = y)$$

$$\forall n, \quad X_n :\equiv \Delta_n \to X \quad \text{is a type}$$

So each type give rise to a *simplicial type !*

Segal types

When does (X_n) satisfies the Segal condition ?

Segal types

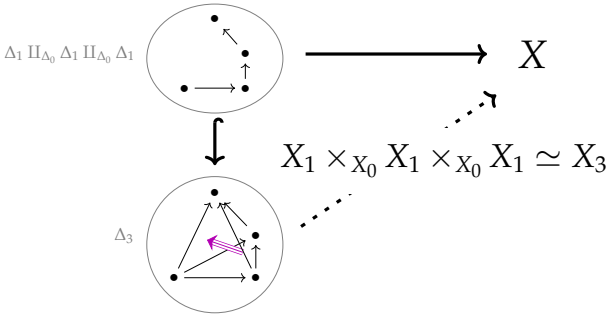
When does (X_n) satisfies the Segal condition ?

$$(\Delta^n \to X) \xrightarrow{\sim} (\Delta_1 \amalg_{\Delta_0} \amalg \cdots \amalg_{\Delta_0} \Delta_1 \to X)$$

Segal types

When does (X_n) satisfies the Segal condition ?

$$(\Delta^n \to X) \xrightarrow{\sim} (\Delta_1 \amalg_{\Delta_0} \amalg \cdots \amalg_{\Delta_0} \Delta_1 \to X)$$



Complete types

By horn filling: a Segal type A admits composites $g \circ f$.

Complete types

By horn filling: a Segal type A admits composites $g \circ f$.

$$\mathsf{isiso}(f) :\equiv \left(\sum_{g:\mathsf{hom}_A(y,x)} g \circ f = \mathsf{id}_x \right) \times \left(\sum_{h:\mathsf{hom}_A(y,x)} f \circ h = \mathsf{id}_y \right)$$

Complete types

By horn filling: a Segal type A admits composites $g \circ f$.

$$\mathsf{isiso}(f) :\equiv \left(\sum_{g:\mathsf{hom}_A(y,x)} g \circ f = \mathsf{id}_x \right) \times \left(\sum_{h:\mathsf{hom}_A(y,x)} f \circ h = \mathsf{id}_y \right)$$

$$x \simeq_A y :\equiv \sum_{f:\mathsf{hom}_A(x,y)} \mathsf{isiso}(f)$$

Complete types

By horn filling: a Segal type A admits composites $g \circ f$.

$$\mathsf{isiso}(f) :\equiv \left(\sum_{g:\mathsf{hom}_A(y,x)} g \circ f = \mathsf{id}_x\right) \times \left(\sum_{h:\mathsf{hom}_A(y,x)} f \circ h = \mathsf{id}_y\right)$$

$$x \simeq_A y :\equiv \sum_{f:\mathsf{hom}_A(x,y)} \mathsf{isiso}(f)$$

$$\mathsf{idtoiso} : \prod_{x,y:A} x =_A y \to x \cong_A y$$

Complete types

By horn filling: a Segal type A admits composites $g \circ f$.

$$\mathsf{isiso}(f) :\equiv \left(\sum_{g:\mathsf{hom}_A(y,x)} g \circ f = \mathsf{id}_x \right) \times \left(\sum_{h:\mathsf{hom}_A(y,x)} f \circ h = \mathsf{id}_y \right)$$

$$x \simeq_A y :\equiv \sum_{f:\mathsf{hom}_A(x,y)} \mathsf{isiso}(f)$$

$$\mathsf{idtoiso} : \prod_{x,y:A} x =_A y \to x \cong_A y$$

Reminiscent to Rezk-completes/univalent categories in HoTT.

Hom Types in STT

In STT:

$$\mathsf{hom}_A(x,y) = \sum_{f : \mathsf{I} \to A} (f\,0 = x) \times (f\,1 = y)$$

$$x \text{ ———— } y$$

Hom Types in STT

In STT:

$$\mathsf{hom}_A(x,y) = \sum_{f : \mathsf{I} \to A} (f\,0 = x) \times (f\,1 = y)$$

$$x \longrightarrow y$$

Then:

$$\mathsf{hom}_{\mathsf{hom}_A(x,y)}(f,g) = \sum_{H : \mathsf{I}^2 \to A} \cdots$$

$$\begin{array}{ccc} x & \xrightarrow{\quad f \quad} & y \\ \| & & \| \\ x & \xrightarrow{\quad g \quad} & y \end{array}$$

The problem of hom-types

In I^2: x, y invertible $\Rightarrow (x, y)$ too.

The problem of hom-types

In I^2: x, y invertible $\Rightarrow$ (x, y) too.

I^2 is 1-categorical: hom of a hom can only be a *groupoid* again.

The problem of hom-types

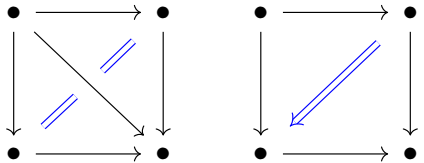
In I^2: x, y invertible $\Rightarrow$ (x, y) too.

I^2 is 1-categorical: hom of a hom can only be a *groupoid* again.

The problem of hom-types

In I^2: x, y invertible $\Rightarrow$ (x, y) too.

I^2 is 1-categorical: hom of a hom can only be a *groupoid* again.

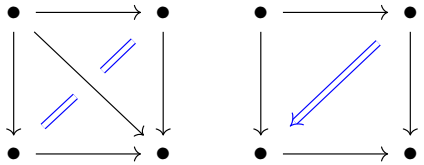


The problem of hom-types

In I^2: x, y invertible $\Rightarrow$ (x, y) too.

I^2 is 1-categorical: hom of a hom can only be a *groupoid* again.

$$\mathsf{hom}_A(x, y) \neq \textstyle\sum_{f : \mathsf{I} \to A} (f\,0 = x) \times (f\,1 = y)$$



$$\mathsf{I} \times \mathsf{I} \qquad \neq \qquad \mathsf{I} \otimes \mathsf{I}$$

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

- Extend contexts in two ways (bunched logic à la Pym–O'Hearn).

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

- Extend contexts in two ways (bunched logic à la Pym–O'Hearn).
- Usual context extension: Γ, Δ

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

- Extend contexts in two ways (bunched logic à la Pym–O'Hearn).
- Usual context extension: Γ, Δ
- Directed, semicartesian: $\Gamma \otimes \Delta$

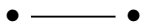
Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

- Extend contexts in two ways (bunched logic à la Pym–O'Hearn).
- Usual context extension: Γ, Δ
- Directed, semicartesian: $\Gamma \otimes \Delta$

$\Gamma \otimes \Delta$



Γ, Δ

Replacing $\to$

- Idea: replacing $(- \times \mathsf{I}) \dashv (\mathsf{I} \to -)$ by $(- \otimes \mathsf{I}) \dashv (\mathsf{I} \multimap -)$

$$\mathsf{I} \multimap (\mathsf{I} \multimap A) \simeq (\mathsf{I} \otimes \mathsf{I} \multimap A)$$

- Extend contexts in two ways (bunched logic à la Pym–O'Hearn).
- Usual context extension: Γ, Δ
- Directed, semicartesian: $\Gamma \otimes \Delta$

$\Gamma \otimes \Delta$ $\qquad$ Γ, Δ

$$\Gamma \otimes \Delta \neq \Delta \otimes \Gamma$$

Interpreted as the Crans-Gray tensor product.

Bunches

$$\Gamma, \Delta ::= \diamond \mid x : A \mid \Gamma \otimes \Delta \mid \Gamma, \Delta$$

Bunches

$$\Gamma, \Delta ::= \diamond \mid x : A \mid \Gamma \otimes \Delta \mid \Gamma, \Delta$$

Judgments live in a split context $\underline{\Lambda} \mid \Gamma$: a *crisp* part, and a *bunch*.

$$\underline{\Lambda} \mid \Gamma \vdash a : A \qquad\qquad \underline{\Lambda} \mid \Gamma \vdash a \equiv b : A$$

Bunches

$$\Gamma, \Delta ::= \diamond \mid x : A \mid \Gamma \otimes \Delta \mid \Gamma, \Delta$$

Judgments live in a split context $\underline{\Lambda} \mid \Gamma$: a *crisp* part, and a *bunch*.

$$\underline{\Lambda} \mid \Gamma \vdash a : A \qquad\qquad \underline{\Lambda} \mid \Gamma \vdash a \equiv b : A$$

crisp asumptions does not care about functoriality.

Rules for $\multimap$ and $\mathrel{-\!\!\circ}$

$$(\multimap\text{-Intro})\ \frac{\Gamma \otimes (x : A) \vdash b : B}{\Gamma \vdash \varrho x.\, b : A \multimap B} \qquad (\mathrel{-\!\!\circ}\text{-Intro})\ \frac{(x : A) \otimes \Gamma \vdash b : B}{\partial x.\, b : A \mathrel{-\!\!\circ} B}$$

$$(\multimap\text{-Elim})\ \frac{\Gamma \vdash f : A \multimap B \qquad \Delta \vdash a : A}{\Gamma \otimes \Delta \vdash f\ |a\rangle : B} \qquad (\mathrel{-\!\!\circ}\text{-Elim})\ \frac{\Gamma \vdash a : A \qquad \Delta \vdash f : A \mathrel{-\!\!\circ} B}{\Gamma \otimes \Delta \vdash \langle a|\, f : B}$$

The $\otimes \dashv (\rightharpoonup, \rightharpoondown)$ adjunctions

For crisp types $X, Y, Z :: \mathcal{U}$

$$(X \otimes Y \rightharpoondown Z) \simeq (X \rightharpoondown Y \rightharpoondown Z) \qquad (X \otimes Y \rightharpoonup Z) \simeq (Y \rightharpoonup X \rightharpoonup Z)$$

The $\otimes \dashv (\rightharpoonup, \rightharpoondown)$ adjunctions

For crisp types $X, Y, Z :: \mathcal{U}$

$$(X \otimes Y \rightharpoondown Z) \simeq (X \rightharpoondown Y \rightharpoondown Z) \qquad (X \otimes Y \rightharpoonup Z) \simeq (Y \rightharpoonup X \rightharpoonup Z)$$

witnessed by currying

$$f \longmapsto \varrho x.\, \varrho y.f\,|\,x \otimes y\rangle \qquad f \longmapsto \partial x.\, \partial y.\, \langle x \otimes y\,|\,f$$

The $\otimes \dashv (\rightarrow, \multimap)$ adjunctions

For crisp types $X, Y, Z :: \mathcal{U}$

$$(X \otimes Y \multimap Z) \simeq (X \multimap Y \multimap Z) \qquad (X \otimes Y \rightarrow Z) \simeq (Y \rightarrow X \rightarrow Z)$$

witnessed by currying

$$f \longmapsto \varrho x.\, \varrho y.f\, |x \otimes y\rangle \qquad f \longmapsto \partial x.\, \partial y.\, \langle x \otimes y |\, f$$

$$\varrho(x \otimes y).f\, |x\rangle\, |y\rangle \longmapsto f \qquad \partial(x \otimes y).\langle x|\, \langle y|\, f \longmapsto f$$

The $\otimes \dashv (\rightharpoonup, \rightharpoondown)$ adjunctions

For crisp types $X, Y, Z :: \mathcal{U}$

$$(X \otimes Y \rightharpoondown Z) \simeq (X \rightharpoondown Y \rightharpoondown Z) \qquad (X \otimes Y \rightharpoonup Z) \simeq (Y \rightharpoonup X \rightharpoonup Z)$$

witnessed by currying

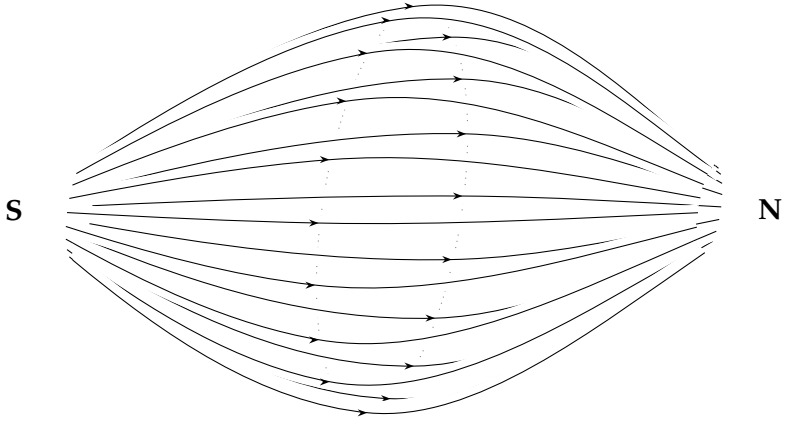
$$f \longmapsto \varrho x.\, \varrho y.f\, |x \otimes y\rangle \qquad f \longmapsto \partial x.\, \partial y.\, \langle x \otimes y|\, f$$

$$\varrho(x \otimes y).f\, |x\rangle\, |y\rangle \longmapsfrom f \qquad \partial(x \otimes y).\langle x|\, \langle y|\, f \longmapsfrom f$$

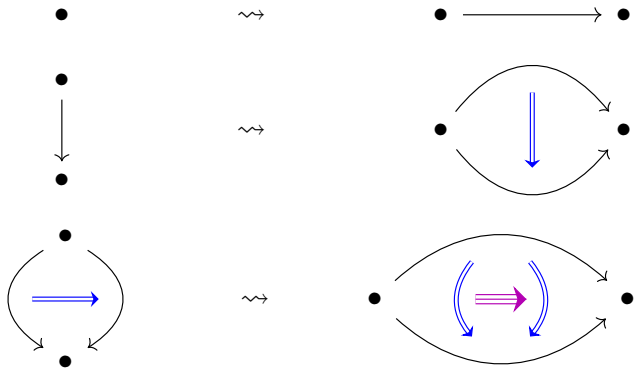
$- \otimes Y$ is left adjoint to $Y \rightharpoondown -$

$Y \otimes -$ is left adjoint to $Y \rightharpoonup -$

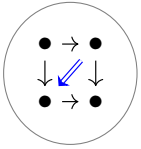
Suspension



Suspension – Examples

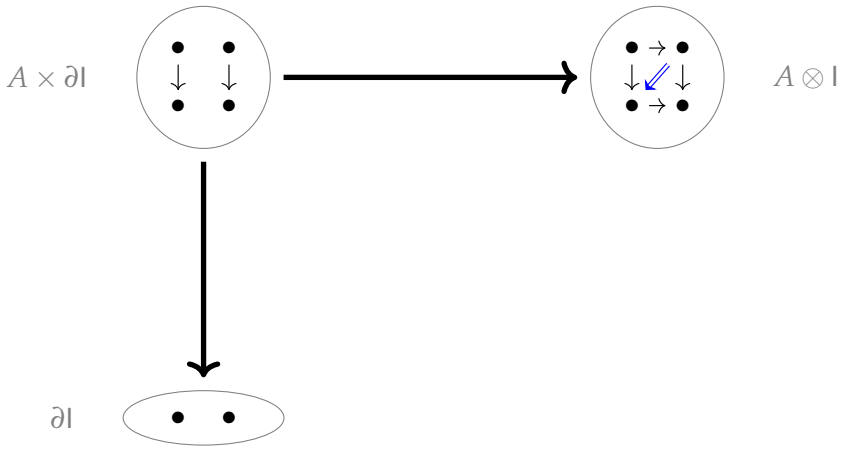


Suspension from $\otimes$

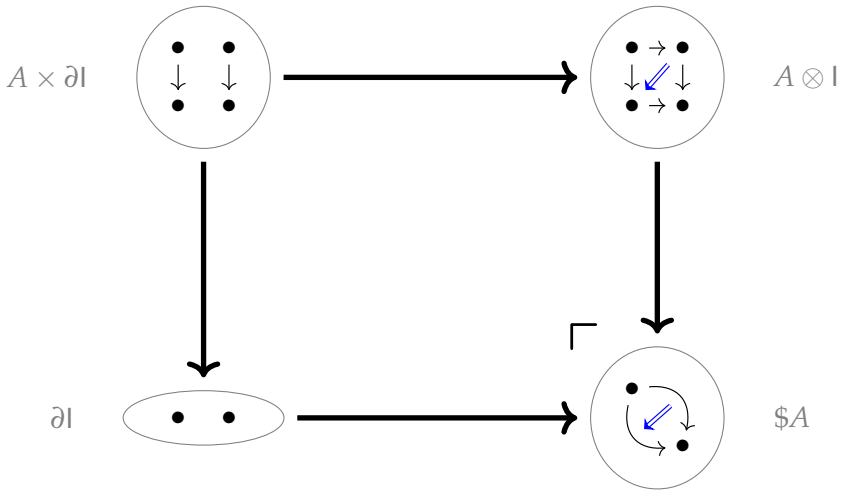


$A \otimes \mathsf{I}$

Suspension from $\otimes$



Suspension from $\otimes$



Suspension ⊣ hom

suspension $\$A$, as a pushout:

$$\begin{array}{ccc} A \times \partial \mathsf{I} & \longrightarrow & A \otimes \mathsf{I} \\ \downarrow & \ulcorner & \downarrow \\ \partial \mathsf{I} & \longrightarrow & \$A \end{array}$$

Suspension ⊣ hom

suspension $\$A$, as a pushout:

$$\begin{array}{ccc} A \times \partial \mathsf{I} & \longrightarrow & A \otimes \mathsf{I} \\ \downarrow & \ulcorner & \downarrow \\ \partial \mathsf{I} & \longrightarrow & \$A \end{array}$$

$$\mathsf{hom}_A(x, y) :\equiv \sum_{f : \mathsf{I} \to A} (f\,|0\rangle = x) \times (f\,|1\rangle = y)$$

Suspension ⊣ hom

suspension $\$A$, as a pushout:

$$\begin{array}{ccc} A \times \partial \mathsf{I} & \longrightarrow & A \otimes \mathsf{I} \\ \downarrow & \ulcorner & \downarrow \\ \partial \mathsf{I} & \longrightarrow & \$A \end{array}$$

$$\mathsf{hom}_A(x, y) \;:\equiv\; \sum_{f : \mathsf{I} \to A} (f\,|0\rangle = x) \times (f\,|1\rangle = y)$$

Theorem

$$(\,\$X \to Y \text{ sending } 0, 1 \text{ to } y_0, y_1\,) \;\simeq\; (X \to \mathsf{hom}_Y(y_0, y_1))$$

`batt`: a type-checker for BaTT

/\^._.^/\

- Ongoing Ocaml implementation by S. Mimram: `github.com/smimram/batt`

`batt`: a type-checker for BaTT

/\^._.^/\

- Ongoing Ocaml implementation by S. Mimram: `github.com/smimram/batt`
- Lexer/parser, Type-checker, Metavariables, Unification.

`batt`: a type-checker for BaTT

/\^._.^/\

- Ongoing Ocaml implementation by S. Mimram: `github.com/smimram/batt`
- Lexer/parser, Type-checker, Metavariables, Unification.
- Small module system, Holes à la Agda, Readable error messages !

```
13) x-11)) (((?13927 x-0) x-1) x-2)))) (λ(x10⊗x11).(Bool_ind(((J(((J(refl) ((?21559 x-2) x-10)) (((postulate54 (λ(x12⊗x13).(x-
2 ⊗ (Bool_ind(postulate36,postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ false)))) ((?21576 x-2) x-10)) ((λ(x12,x13).((λ(x14
x15).x-14) x-13)) x-2)),((J(((J(refl) ((?21450 x-2) x-10)) (((postulate54 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postulate36,postulate3
) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ true)))) ((?21458 x-2) x-10)) ((λ(x12,x13).((λ(x14,x15).x-15) x-13)) x-2))) x-11))) fals
) ≡ ((J((x-9 (((postulate53 (λ(x10⊗x11).(x-10 ⊗ (Bool_ind(postulate36,postulate37) x-11)))) (λ(x10⊗x11).x-11)) false))) ((?209
3 x-2) x-9)) ((λ(x10,x11).((λ(x12,x13).x-12) x-11)) x-2))).(((((((postulate100 (λ(x11⊗x12).(x-11 ⊗ (Bool_ind(postulate36,postu
ate37) x-12)))) (λ(x11⊗x12).x-12)) ((λ(x11,x12).x-11) (((?13927 x-0) x-1) x-2))) (λ(x11⊗x12).(((λ(x13,x14).x-13) x-2) (((postu
ate52 (λ(x13⊗x14).(x-13 ⊗ (Bool_ind(postulate36,postulate37) x-14)))) (λ(x13⊗x14).x-14)) (x-11 ⊗ x-12)))))) Bool_ind(((λ(x11,x1
).((λ(x13,x14).x-13) x-12)) (((?13927 x-0) x-1) x-2)),((λ(x11,x12).((λ(x13,x14).x-14) x-12)) (((?13927 x-0) x-1) x-2)))) (λ(x1
⊗x12).(Bool_ind(((J(((J(refl) ((?21189 x-2) x-11)) (((postulate54 (λ(x13⊗x14).(x-13 ⊗ (Bool_ind(postulate36,postulate37) x-14)
)) (λ(x13⊗x14).x-14)) (x-11 ⊗ false)))) ((?21206 x-2) x-11)) ((λ(x13,x14).((λ(x15,x16).x-15) x-14)) x-2)),((J(((J(refl) ((?210
0 x-2) x-11)) (((postulate54 (λ(x13⊗x14).(x-13 ⊗ (Bool_ind(postulate36,postulate37) x-14)))) (λ(x13⊗x14).x-14)) (x-11 ⊗ true))
) ((?21088 x-2) x-11)) ((λ(x13,x14).((λ(x15,x16).x-16) x-14)) x-2))) x-12))) true) ≡ ((J((x-9 (((postulate53 (λ(x11⊗x12).(x-11
⊗ (Bool_ind(postulate36,postulate37) x-12)))) (λ(x11⊗x12).x-12)) true))) (((?20967 x-2) x-9) x-10)) ((λ(x11,x12).((λ(x13,x14).
-14) x-12)) x-2))))
HOLE in file ./hom.batt line 84 characters 1-2 : (Σ(x9 : (x9 : (?13995 x-2)) → (((((((postulate98 (λ(x10⊗x11).(x-10 ⊗ (Bool_in
(postulate36,postulate37) x-11)))) (λ(x10⊗x11).x-11)) ((λ(x10,x11).x-10) (((?13927 x-0) x-1) x-2))) (λ(x10⊗x11).(((λ(x12,x13).
-12) x-2) (((postulate52 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postulate36,postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ x-11)))))
ool_ind(((λ(x10,x11).((λ(x12,x13).x-12) x-11)) (((?13927 x-0) x-1) x-2)),((λ(x10,x11).((λ(x12,x13).x-13) x-11)) (((?13927 x-0)
x-1) x-2)))) (λ(x10⊗x11).(Bool_ind(((J(((J(refl) ((?23045 x-2) x-10)) (((postulate54 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postulate36
postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ false)))) ((?23062 x-2) x-10)) ((λ(x12,x13).((λ(x14,x15).x-14) x-13)) x-2)),(
J(((J(refl) ((?22936 x-2) x-10)) (((postulate54 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postulate36,postulate37) x-13)))) (λ(x12⊗x13).x-
3)) (x-10 ⊗ true)))) ((?22944 x-2) x-10)) ((λ(x12,x13).((λ(x14,x15).x-15) x-13)) x-2))) x-11))) x-9) ≡ (((λ(x10,x11).x-10) x-2
x-9)).(Σ(x10 : ((((((((postulate100 (λ(x10⊗x11).(x-10 ⊗ (Bool_ind(postulate36,postulate37) x-11)))) (λ(x10⊗x11).x-11)) ((λ(x10
x11).x-10) (((?13927 x-0) x-1) x-2))) (λ(x10⊗x11).(((λ(x12,x13).x-12) x-2) (((postulate52 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postul
te36,postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ x-11))))) Bool_ind(((λ(x10,x11).((λ(x12,x13).x-12) x-11)) (((?13927 x-0)
x-1) x-2)),((λ(x10,x11).((λ(x12,x13).x-13) x-11)) (((?13927 x-0) x-1) x-2)))) (λ(x10⊗x11).(Bool_ind(((J(((J(refl) ((?22676 x-2
x-10)) (((postulate54 (λ(x12⊗x13).(x-12 ⊗ (Bool_ind(postulate36,postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ false)))) ((
22693 x-2) x-10)) ((λ(x12,x13).((λ(x14,x15).x-14) x-13)) x-2)),((J(((J(refl) ((?22567 x-2) x-10)) (((postulate54 (λ(x12⊗x13).(
-12 ⊗ (Bool_ind(postulate36,postulate37) x-13)))) (λ(x12⊗x13).x-13)) (x-10 ⊗ true)))) ((?22575 x-2) x-10)) ((λ(x12,x13).((λ(x1
```

Examples

smimram.github.io/batt/

The end

Thank you !

Questions ?