A Bunched Approach to Directed HoTT

Louise Leclerc

Samuel Mimram

September 17, 2026

Outline

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.
- Statements have the form $\Gamma \vdash a : A$ or $\Gamma \vdash a = b : A$ where $\Gamma = (x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n)$.

Intuitionistic Type Theory

MLTT:

- MARTIN-LÖF, 1972.
- Statements have the form $\Gamma \vdash a : A$ or $\Gamma \vdash a = b : A$ where $\Gamma = (x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n)$.
- Basic types: $\mathbb{0}$, $\mathbb{1}$, Bool, $\mathbb{N}$, ...

Intuitionistic Type Theory

MLTT:

- MARTIN-LÖF, 1972.
- Statements have the form $\Gamma \vdash a : A$ or $\Gamma \vdash a = b : A$ where $\Gamma = (x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n)$.
- Basic types: $\mathbb{0}$, $\mathbb{1}$, Bool, $\mathbb{N}$, ...
- Universes $\mathcal{U}$ closed under $\to$, $\times$, $+$, ...

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:

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

Curry–Howard

Correspondance between types and logical propositions.

- $\to$ is the implication arrow.
- $\times$ is the conjunction.
- $+$ is a constructive disjunction.
- $\mathbb{0}$ is false, $\mathbb{1}$ is true.

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

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

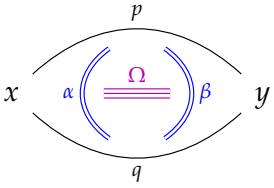
- **concatenation**: $p : x = y, q : y = z \rightsquigarrow p \cdot q : x = z$
- **inverse**: $p : x = y \rightsquigarrow p^{-1} : y = x$
- **unit**: $\mathsf{refl}_x \cdot p = p \quad \text{and} \quad p \cdot \mathsf{refl}_y = p$
- **associativity**: $p \cdot (q \cdot r) = (p \cdot q) \cdot r$
- **inversion**: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

Theorem

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

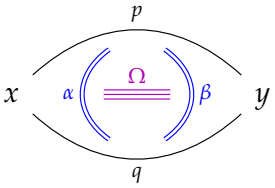
∞-groupoids

Spaces up to homotopy



∞-groupoids

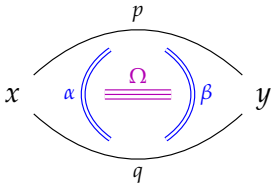
Spaces up to homotopy



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

∞-groupoids

Spaces up to homotopy



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

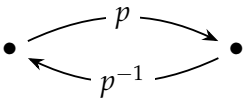
Formalisation of homotopy theory
(Feudenthal, long exact sequences, sphere homotopy groups...).

Non-directedness of HoTT

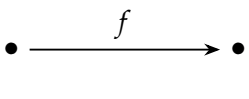
- Paths $x = y$ are always *invertible*.

Non-directedness of HoTT

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



$x = y$



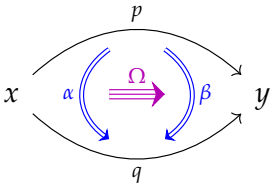
$\mathsf{hom}_A(x, y)$

∞-categories

"Directed" spaces up to homotopy

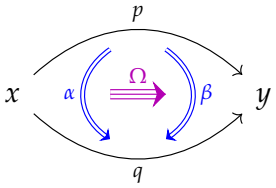
∞-categories

"Directed" spaces up to homotopy



∞-categories

"Directed" spaces up to homotopy

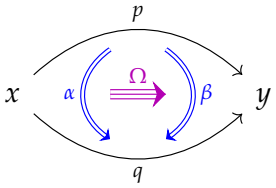


$x, y : A$ $\qquad$ $p, q : \mathsf{hom}_A(x, y)$ $\qquad$ $\alpha, \beta : \mathsf{hom}_{\mathsf{hom}_A(x,y)}(p, q)$

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

∞-categories

"Directed" spaces up to homotopy



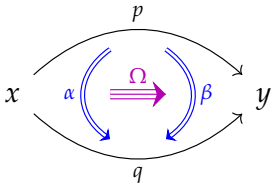
$$x, y : A \qquad p, q : \mathsf{hom}_A(x, y) \qquad \alpha, \beta : \mathsf{hom}_{\mathsf{hom}_A(x,y)}(p, q)$$

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

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.

Simplicial type theory

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

Simplicial type theory

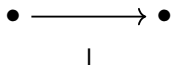
- Riehl, Shulman (2017); Gratzer, Weinberger, Buchholtz…
- Postulate a formal interval $(\mathsf{I}, 0, 1, \leq)$.

Simplicial type theory

- Riehl, Shulman (2017); Gratzer, Weinberger, Buchholtz…
- Postulate a formal interval $(\mathsf{I}, 0, 1, \leq)$.
- Goal: define $(\infty, 1)$-categories internally.

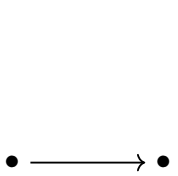
The interval

We add to the syntax an interval $(\mathsf{I}, 0, 1, \leq)$

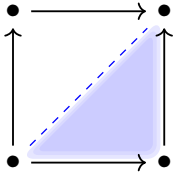


The interval

We add to the syntax an interval $(\mathsf{I}, 0, 1, \leq)$



I

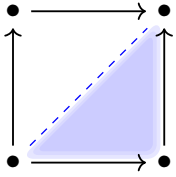


$\Delta^2 :\equiv \{\langle t_1, t_2\rangle : \mathsf{I}^2 \mid t_2 \leq t_1\}$

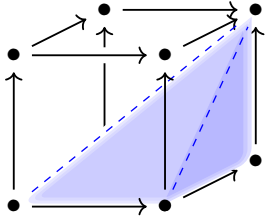
The interval

We add to the syntax an interval $(\mathsf{I}, 0, 1, \leq)$

I



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



Δ^3

Simplicial types

$$X \text{ type}$$

$$\mathsf{I} \to X \text{ type}$$

Simplicial types

$$X \text{ type}$$

$$\mathsf{I} \to X \text{ type}$$

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

Simplicial types

$$X \text{ type}$$

$$\mathsf{I} \to X \text{ type}$$

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

but also

$$\Delta^2 \to X \text{ type}$$

$$\Delta^3 \to X \text{ type}$$

$$\vdots$$

Simplicial types

$$X \text{ type}$$

$$\mathsf{I} \to X \text{ type}$$

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

but also

$$\Delta^2 \to X \text{ type}$$

$$\Delta^3 \to X \text{ type}$$

$$\vdots$$

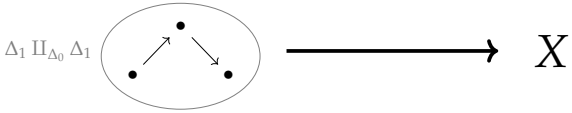
So each type give rise to a *simplicial type*.

Categories (1)

When does X behaves as a category ?

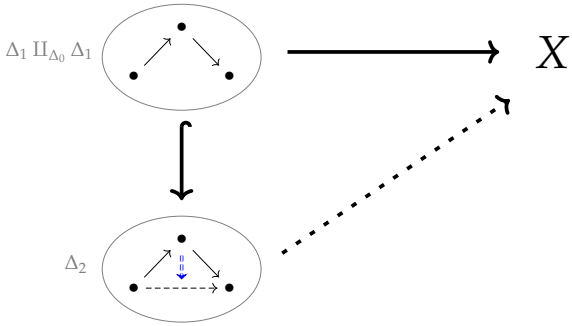
Categories (1)

When does X behaves as a category ?



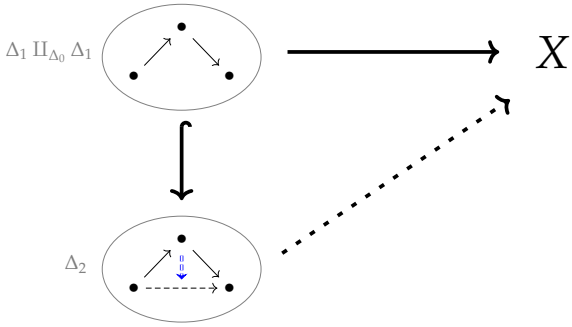
Categories (1)

When does X behaves as a category ?



Categories (1)

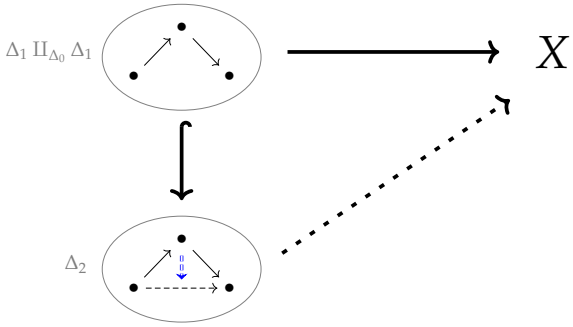
When does X behaves as a category ?



$$(\Delta^2 \to X) \xrightarrow{\sim} (\mathsf{I} +_{\mathbb{1}} \mathsf{I} \to X)$$

Categories (1)

When does X behaves as a category ?

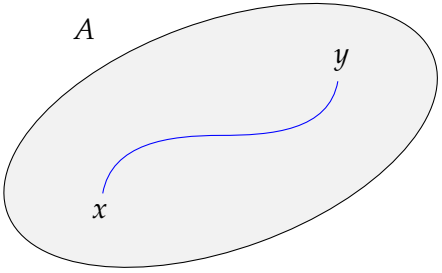


$$(\Delta^2 \to X) \xrightarrow{\sim} (\mathsf{I} +_{\mathbb{1}} \mathsf{I} \to X)$$

$$\sum_{x,y,z:X} \mathsf{hom}_X(x,y) \times \mathsf{hom}_X(y,z) \xrightarrow{\circ} \mathsf{hom}_X(x,z)$$

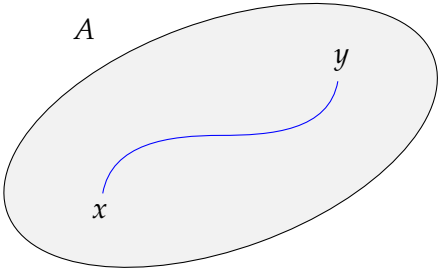
Categories (2)

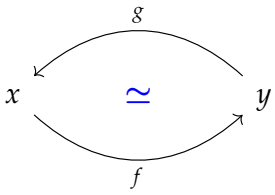
A type A as before admits composites $g \circ f$.



Categories (2)

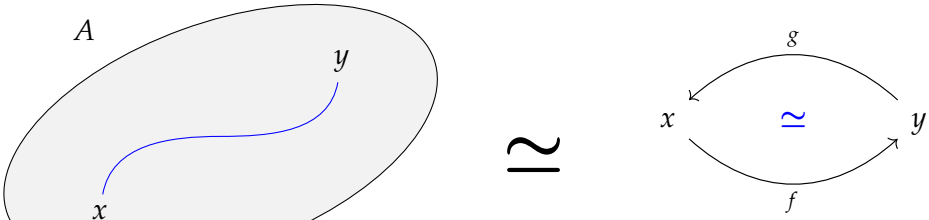
A type A as before admits composites $g \circ f$.





Categories (2)

A type A as before admits composites $g \circ f$.



$$(x =_A y) \simeq \mathsf{iso}_A(x, y)$$

Hom Types in STT

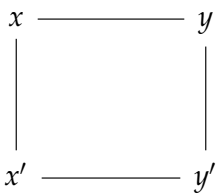
Type of 1-cells $= I \to A$.

$$x \longrightarrow y$$

Hom Types in STT

Type of 1-cells $= I \to A$.

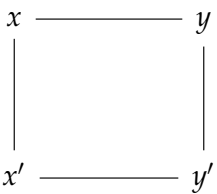
Type of 2-cells $= I \to (I \to A)$.



Hom Types in STT

Type of 1-cells $= I \to A$.

Type of 2-cells $= I \to (I \to A)$.



Cells mediating between 1-cells should be 2-cells.

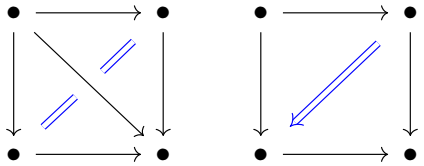
The problem of hom-types

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.

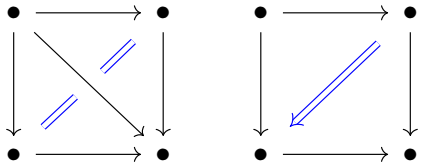


$\mathsf{I} \times \mathsf{I} \quad \neq \quad \mathsf{I} \otimes \mathsf{I}$

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.



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

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

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

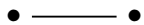
Replacing $\to$

- 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: $\Gamma \otimes \Delta$

$\Gamma \otimes \Delta$



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

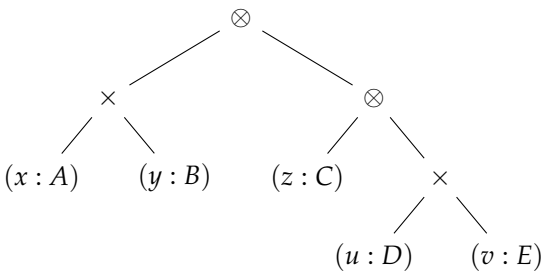
$$\Gamma, \Delta ::= \diamond \mid x : A \mid \Gamma \otimes \Delta \mid \Gamma, \Delta$$

Judgments live in a bunch Γ, which has a *tree* structure.

Bunches

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

Judgments live in a bunch Γ, which has a *tree* structure.



$$\vdash \quad y \otimes (u, v) : B \bigotimes (D \times E)$$

Rules for $\multimap$ and $\rightharpoonup$

$$(\rightharpoonup\text{-Intro})\ \frac{\Gamma,\ (x : A) \vdash b : B}{\Gamma \vdash \lambda x.\, b : A \rightharpoonup B}$$

Rules for $\multimap$ and $\multimap$

$$(\to\text{-Intro})\ \frac{\Gamma,\ (x : A) \vdash b : B}{\Gamma \vdash \lambda x.\, b : A \to B} \qquad (\to\text{-Elim})\ \frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash a : A}{\Gamma,\ \Delta \vdash f\ |a\rangle : B}$$

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

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

$$(\to\text{-Intro})\ \frac{\Gamma,\ (x : A) \vdash b : B}{\Gamma \vdash \lambda x.\, b : A \to B} \qquad (\to\text{-Elim})\ \frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash a : A}{\Gamma,\ \Delta \vdash f\ |a\rangle : B}$$

$$(\multimap\text{-Intro})\ \frac{\Gamma \otimes (x : A) \vdash b : B}{\Gamma \vdash \varrho x.\, b : A \multimap B} \qquad (\multimap\text{-Elim})\ \frac{\Gamma \vdash f : A \multimap B \qquad \Delta \vdash a : A}{\Gamma \otimes \Delta \vdash f\ |a\rangle : B}$$

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

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

$$(\to\text{-Intro})\ \frac{\Gamma,\ (x : A) \vdash b : B}{\Gamma \vdash \lambda x.\, b : A \to B} \qquad (\multimap\text{-Elim})\ \frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash a : A}{\Gamma,\ \Delta \vdash f\ |a\rangle : B}$$

$$(\multimap\text{-Intro})\ \frac{\Gamma \otimes (x : A) \vdash b : B}{\Gamma \vdash \varrho x.\, b : A \multimap B} \qquad (\multimap\text{-Elim})\ \frac{\Gamma \vdash f : A \multimap B \qquad \Delta \vdash a : A}{\Gamma \otimes \Delta \vdash f\ |a\rangle : B}$$

$$(\mathrel{-\!\!\circ}\text{-Intro})\ \frac{(x : A) \otimes \Gamma \vdash b : B}{\Gamma \vdash \partial x.\, b : A \mathrel{-\!\!\circ} B} \qquad (\mathrel{-\!\!\circ}\text{-Elim})\ \frac{\Gamma \vdash a : A \qquad \Delta \vdash f : A \mathrel{-\!\!\circ} B}{\Gamma \otimes \Delta \vdash \langle a |\, f : B}$$

$$(\otimes\text{-Intro})\ \frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma \otimes \Delta \vdash a \otimes b : A \otimes B}$$

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 $\otimes \dashv (\rightarrowtail, \rightarrowtail)$ adjunctions

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

$$(X \otimes Y \rightarrowtail Z) \simeq (X \rightarrowtail Y \rightarrowtail Z) \qquad (X \otimes Y \rightarrowtail Z) \simeq (Y \rightarrowtail X \rightarrowtail Z)$$

witnessed by currying

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

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

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

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

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

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

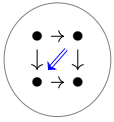
$Y \otimes -$ is left adjoint to $Y \rightharpoonup -$

hom-types

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

hom-types

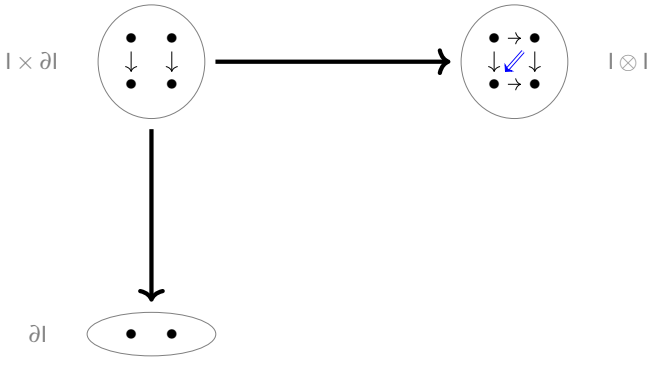
$$\mathsf{hom}_A(x, y) :\equiv \sum_{f : \mathsf{I} \to A} (f\ |0\rangle = x) \times (f\ |1\rangle = y)$$



$\mathsf{I} \otimes \mathsf{I}$

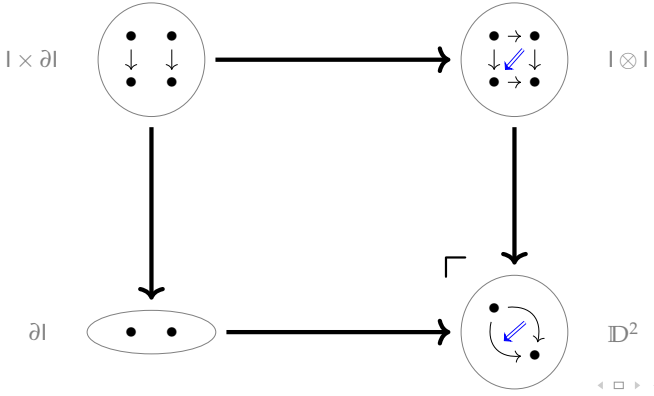
hom-types

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



hom-types

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

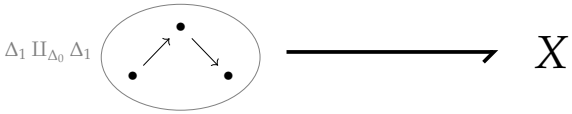


Categories

When does X behaves as a category ?

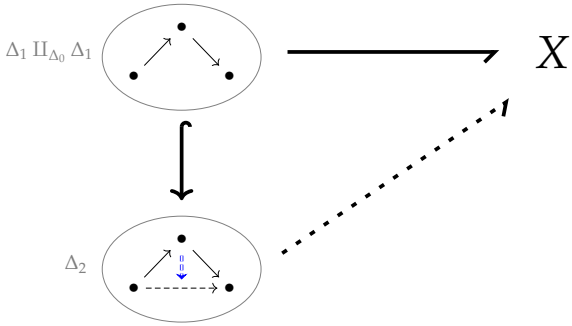
Categories

When does X behaves as a category ?



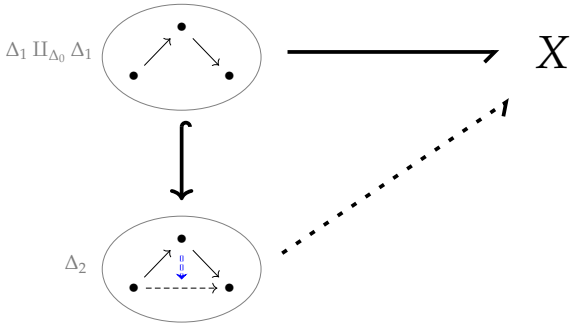
Categories

When does X behaves as a category ?



Categories

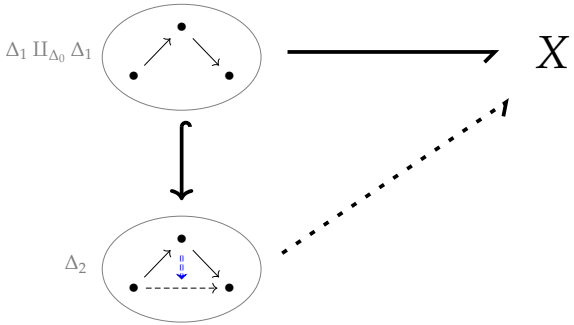
When does X behaves as a category ?



$$(\Delta^2 \rightharpoonup X) \xrightarrow{\sim} (\mathsf{I} +_{\mathbb{1}} \mathsf{I} \rightharpoonup X) \simeq \sum_{x,y,z:X} \mathsf{hom}_X(x, y) \times \mathsf{hom}_X(y, z)$$

Categories

When does X behaves as a category ?



$$(\Delta^2 \rightharpoonup X) \xrightarrow{\sim} (\mathsf{I} +_{\mathbb{1}} \mathsf{I} \rightharpoonup X) \simeq \sum_{x,y,z:X} \mathsf{hom}_X(x, y) \times \mathsf{hom}_X(y, z)$$

$$\sum_{x,y,z:X} \mathsf{hom}_X(x, y) \times \mathsf{hom}_X(y, z) \xrightarrow{\circ} \mathsf{hom}_X(x, z)$$

`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 ?