CaTT contexts are finite computads

Thibaut Benjamin, Ioannis Markakis, Chiara Sarti

MFPS – June 21, 2024

ω-categories: why?

- Identity types in MLTT

ω-categories: why?

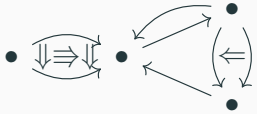
- Identity types in MLTT
- Proof relevant rewriting

ω-categories: why?

- Identity types in MLTT
- Proof relevant rewriting
- Directed homotopy and concurrent programming

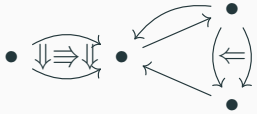
ω-categories: what?

- A globular set



ω-categories: what?

- A globular set

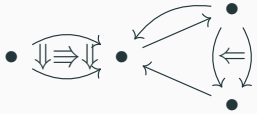


- With compositions operations

  1 for 1-cells, 2 (horizontal/vertical) for 2-cells, 3 for 3-cells ...

ω-categories: what?

- A globular set

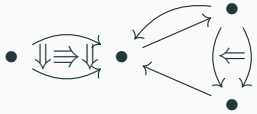


- With compositions operations
  1 for 1-cells, 2 (horizontal/vertical) for 2-cells, 3 for 3-cells ...

- Satisfying various axioms...
  associativity, unitality, exchange rules,...

ω-categories: what?

- A globular set



- With compositions operations
  1 for 1-cells, 2 (horizontal/vertical) for 2-cells, 3 for 3-cells ...

- Satisfying various axioms...
  associativity, unitality, exchange rules,...

- Weakly!
  Up to a higher cell: witnessing an equivalence

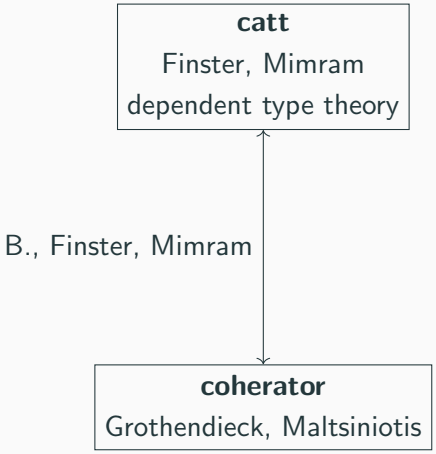
ω-categories: how?

catt
Finster, Mimram
dependent type theory

computads
description by DFMRV [1]
inductively defined

[1] Dean, Finster, Markakis, Reutter, Vicary

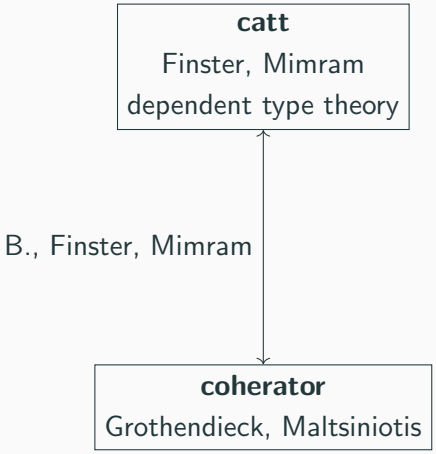
ω-categories: how?

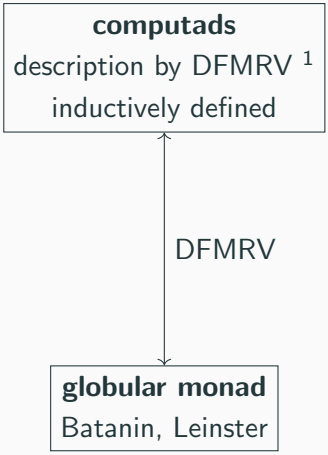


computads
description by DFMRV [1]
inductively defined

[1] Dean, Finster, Markakis, Reutter, Vicary

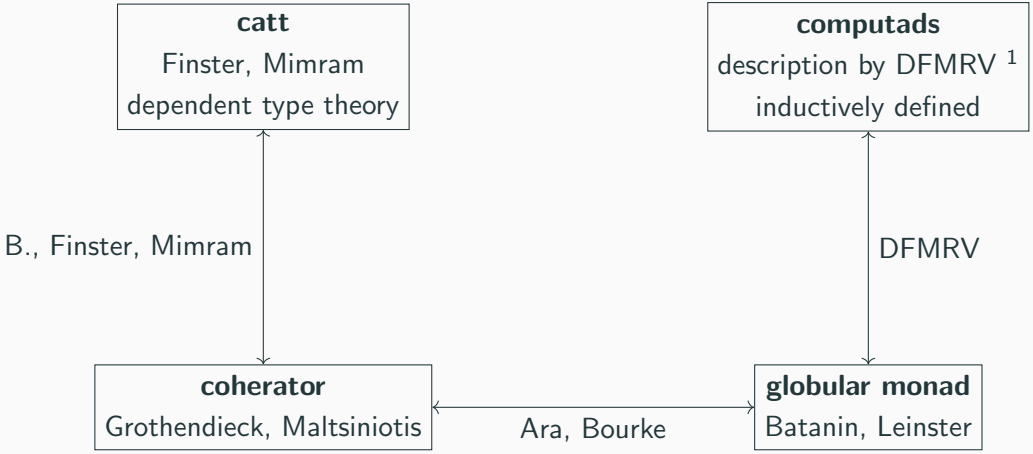
ω-categories: how?





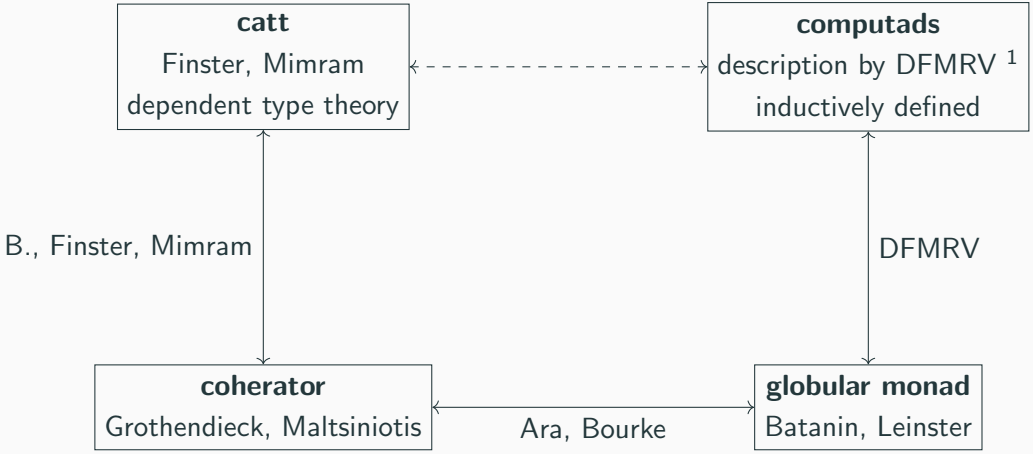
[1] Dean, Finster, Markakis, Reutter, Vicary

ω-categories: how?



[1] Dean, Finster, Markakis, Reutter, Vicary

ω-categories: how?



[1] Dean, Finster, Markakis, Reutter, Vicary

Presentation of CaTT

Types and globular sets

CaTT has 2 type constructors:

$$\frac{\Gamma \vdash}{\Gamma \vdash \star} \qquad \frac{\Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u \to_A v}$$

Types and globular sets

CaTT has 2 type constructors:

$$\frac{\Gamma \vdash}{\Gamma \vdash \star} \qquad\qquad \frac{\Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u \to_A v}$$

- **Idea**: The arrow types represent abstract directed equality.

Types and globular sets

CaTT has 2 type constructors:

$$\frac{\Gamma \vdash}{\Gamma \vdash \star} \qquad\qquad \frac{\Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u \to_A v}$$

- **Idea**: The arrow types represent abstract directed equality.
- **Structure**: Contexts using these types are globular sets

$$(x : \star)(y : \star)(f : x \to y)(g : x \to x)$$

Pasting schemes

- **Idea**: Pasting schemes are diagrams which can be composed in an essentially unique way

Pasting schemes

- **Idea**: Pasting schemes are diagrams which can be composed in an essentially unique way

- **Construction**: freely glue cells onto free spaces

$$\frac{}{(x : \star) \vdash_{\mathsf{ps}} x : \star}(\textsc{start})$$

Pasting schemes

- **Idea**: Pasting schemes are diagrams which can be composed in an essentially unique way

- **Construction**: freely glue cells onto free spaces

$$\frac{}{(x : \star) \vdash_{\mathsf{ps}} x : \star}(\textsc{start}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : A}{\Gamma, y : A, f : x \to_A y \vdash_{\mathsf{ps}} f : x \to_A y}(\textsc{ext})$$

Pasting schemes

- **Idea**: Pasting schemes are diagrams which can be composed in an essentially unique way

- **Construction**: freely glue cells onto free spaces

$$\frac{}{(x : \star) \vdash_{\mathsf{ps}} x : \star}(\textsc{start}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : A}{\Gamma, y : A, f : x \to_A y \vdash_{\mathsf{ps}} f : x \to_A y}(\textsc{ext})$$

$$\frac{\Gamma \vdash_{\mathsf{ps}} f : x \to_A y}{\Gamma \vdash_{\mathsf{ps}} y : A}(\textsc{drop})$$

Pasting schemes

- **Idea**: Pasting schemes are diagrams which can be composed in an essentially unique way

- **Construction**: freely glue cells onto free spaces

$$\frac{}{(x : \star) \vdash_{\mathsf{ps}} x : \star}(\textsc{start}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : A}{\Gamma, y : A, f : x \to_A y \vdash_{\mathsf{ps}} f : x \to_A y}(\textsc{ext})$$

$$\frac{\Gamma \vdash_{\mathsf{ps}} f : x \to_A y}{\Gamma \vdash_{\mathsf{ps}} y : A}(\textsc{drop}) \qquad \frac{\Gamma \vdash_{\mathsf{ps}} x : \star}{\Gamma \vdash_{\mathsf{ps}}}(\textsc{done})$$

Pasting schemes : Example

$$\frac{}{(x : \star) \vdash_{\mathsf{ps}} x : \star}\ (\textsc{start})$$

x

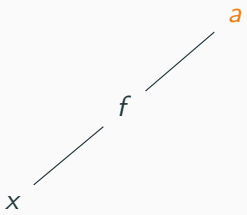
x

Pasting schemes : Example

$$\frac{}{(x : \star)\ (y : \star)(f : x \to y) \vdash_{\mathsf{ps}} f : x \to y}\ (\textsc{start})\ (\textsc{ext})$$

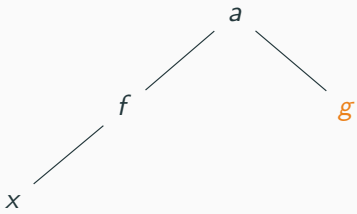
Pasting schemes : Example

$$\frac{}{(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g) \vdash_{\mathsf{ps}} a : f \to g}\ (\textsc{start})\ (\textsc{ext})\ (\textsc{ext})$$

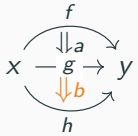


Pasting schemes : Example

$$\frac{}{(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g) \vdash_{\mathsf{ps}} g : x \to y}\ (\textsc{start})\ (\textsc{ext})\ (\textsc{ext})\ (\textsc{drop})$$

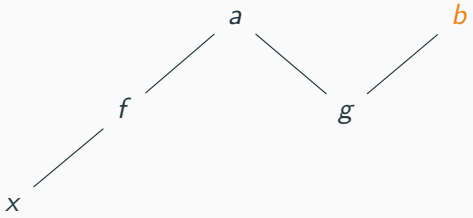


Pasting schemes : Example

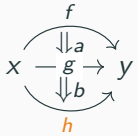


(START) (EXT) (EXT) (DROP) (EXT)

$$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$$
$$(h : x \to y)(b : g \to h) \vdash_{\mathsf{ps}} b : g \to h$$

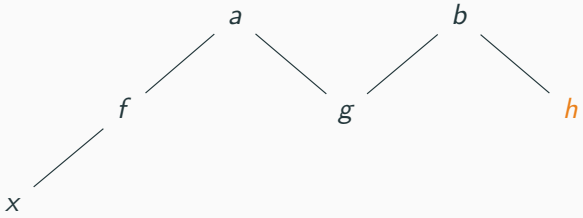


Pasting schemes : Example

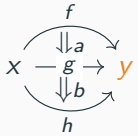


(START) (EXT) (EXT) (DROP) (EXT) (DROP)

$$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$$
$$(h : x \to y)(b : g \to h) \vdash_{\mathsf{ps}} h : x \to y$$

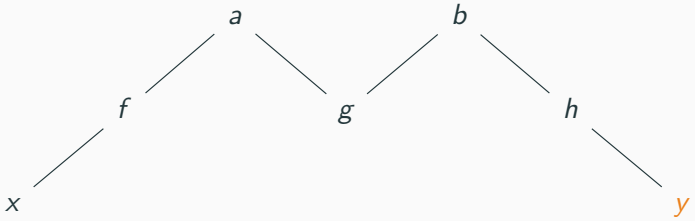


Pasting schemes : Example

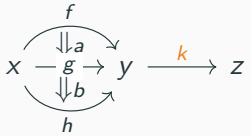


(START) (EXT) (EXT) (DROP) (EXT) (DROP)
(DROP)

$$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$$
$$(h : x \to y)(b : g \to h) \vdash_{\mathsf{ps}} y : \star$$

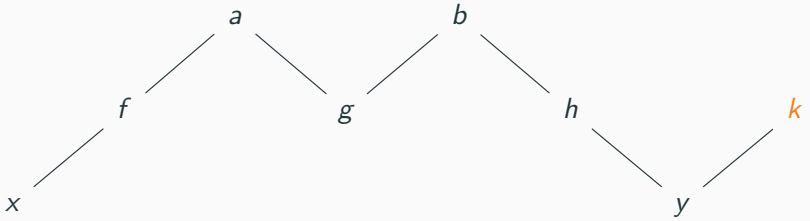


Pasting schemes : Example

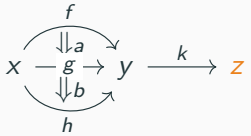


(START) (EXT) (EXT) (DROP) (EXT) (DROP)
(DROP) (EXT)

$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$
$(h : x \to y)(b : g \to h)\ (z : \star)(k : y \to z) \vdash_{\mathsf{ps}} k : y \to z$

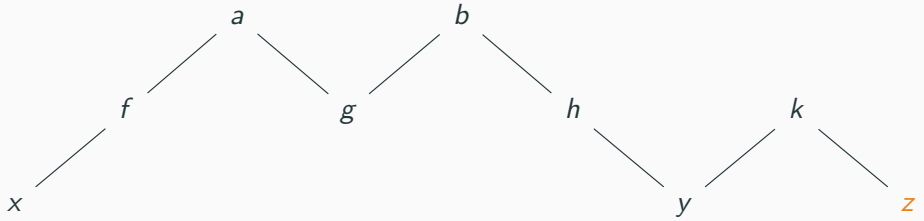


Pasting schemes : Example

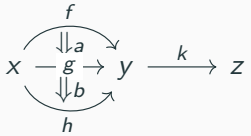


(START) (EXT) (EXT) (DROP) (EXT) (DROP)
(DROP) (EXT) (DROP)

$$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$$
$$(h : x \to y)(b : g \to h)\ (z : \star)(k : y \to z) \vdash_{\mathsf{ps}} z : \star$$

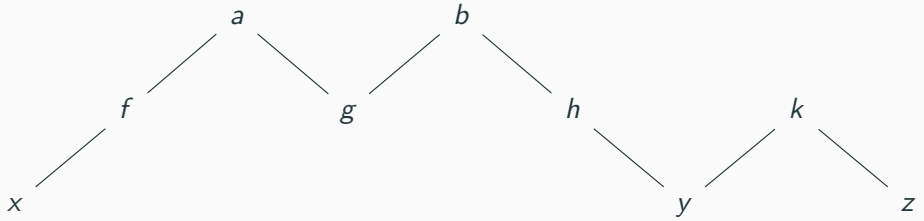


Pasting schemes : Example



(START) (EXT) (EXT) (DROP) (EXT) (DROP)
(DROP) (EXT) (DROP) (DONE)

$(x : \star)\ (y : \star)(f : x \to y)\ (g : x \to y)(a : f \to g)$
$(h : x \to y)(b : g \to h)\ (z : \star)(k : y \to z) \vdash_{\mathsf{ps}}$



Term constructors: compositions and coherences

- **Idea**: Force every pasting scheme to have an essentially unique composite

Term constructors: compositions and coherences

- **Idea**: Force every pasting scheme to have an essentially unique composite

- **Existence**

$$\frac{\Gamma \vdash_{\mathsf{ps}} \qquad \partial^- \Gamma \vdash u : A \qquad \partial^+ \Gamma \vdash v : A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{comp}_{\Gamma,u,v}[\gamma] : u[\gamma] \to v[\gamma]} \quad \begin{cases} \mathsf{Var}(\partial^- \Gamma) = \mathsf{Var}(u) \cup \mathsf{Var}(A) \\ \mathsf{Var}(\partial^+ \Gamma) = \mathsf{Var}(v) \cup \mathsf{Var}(A) \end{cases}$$

Term constructors: compositions and coherences

- **Idea**: Force every pasting scheme to have an essentially unique composite

- **Existence**

$$\frac{\Gamma \vdash_{\mathsf{ps}} \qquad \partial^- \Gamma \vdash u : A \qquad \partial^+ \Gamma \vdash v : A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{comp}_{\Gamma,u,v}[\gamma] : u[\gamma] \to v[\gamma]} \quad \begin{cases} \mathsf{Var}(\partial^- \Gamma) = \mathsf{Var}(u) \cup \mathsf{Var}(A) \\ \mathsf{Var}(\partial^+ \Gamma) = \mathsf{Var}(v) \cup \mathsf{Var}(A) \end{cases}$$

- **Essential uniqueness**

$$\frac{\Gamma \vdash_{\mathsf{ps}} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A \qquad \Delta \vdash \gamma : \Gamma}{\Delta \vdash \mathsf{coh}_{\Gamma,u,v}[\gamma] : u[\gamma] \to v[\gamma]} \quad \begin{cases} \mathsf{Var}(\Gamma) = \mathsf{Var}(u) \cup \mathsf{Var}(A) \\ \mathsf{Var}(\Gamma) = \mathsf{Var}(v) \cup \mathsf{Var}(A) \end{cases}$$

Implementation of CaTT

- A type checker for CaTT is available[2].

[2] https://github.com/thibautbenjamin/catt

Implementation of CaTT

- A type checker for CaTT is available[2].

- Proof-assistant for coherences in ω-categories.

```
coh comp (x : *) (y : *) (f: x -> y) (z : *) (g: y -> z) : x -> z
coh id (x : *) : x -> x
coh unitl (x : *) (y : *) (f : x -> y) : comp (id x) f -> f
```

[2] https://github.com/thibautbenjamin/catt

Implementation of CaTT

- A type checker for CaTT is available[2].
- Proof-assistant for coherences in ω-categories.

```
coh comp (x : *) (y : *) (f: x -> y) (z : *) (g: y -> z) : x -> z
coh id (x : *) : x -> x
coh unitl (x : *) (y : *) (f : x -> y) : comp (id x) f -> f
```

- Includes proof automation and term synthesis.

[2] https://github.com/thibautbenjamin/catt

Presentation of computads

General Idea

Computads translate a the idea of freeness for higher categories.

General Idea

Computads translate a the idea of freeness for higher categories.

X_0 0-generators

General Idea

Computads translate a the idea of freeness for higher categories.

X_1	1-generators
X_0	0-generators

General Idea

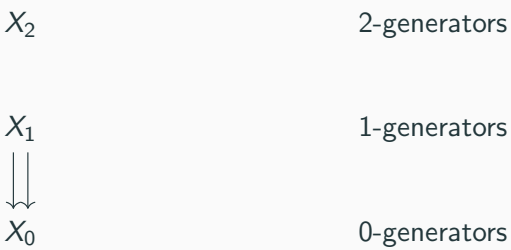
Computads translate a the idea of freeness for higher categories.

1-generators

0-generators

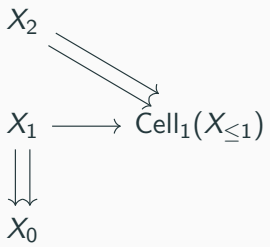
General Idea

Computads translate a the idea of freeness for higher categories.



General Idea

Computads translate a the idea of freeness for higher categories.



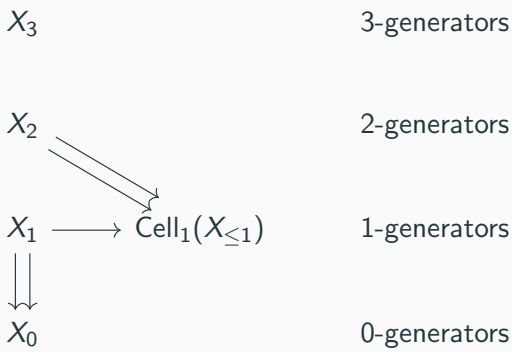
2-generators

1-generators

0-generators

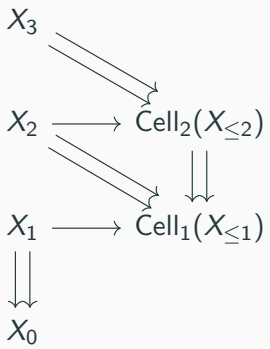
General Idea

Computads translate a the idea of freeness for higher categories.



General Idea

Computads translate a the idea of freeness for higher categories.



3-generators

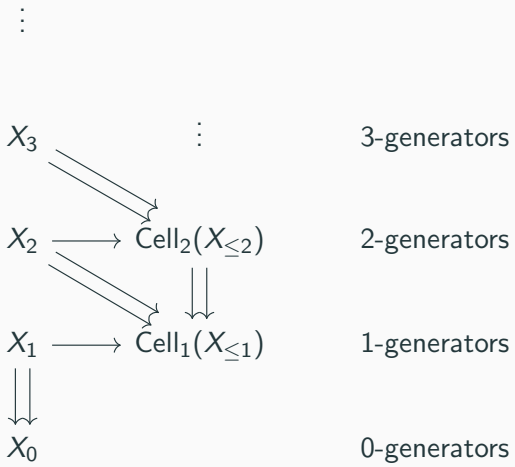
2-generators

1-generators

0-generators

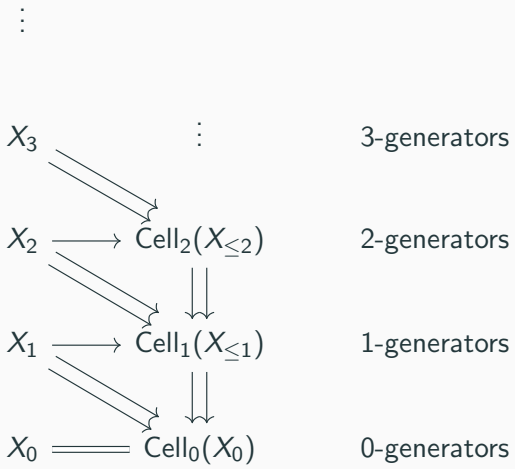
General Idea

Computads translate a the idea of freeness for higher categories.



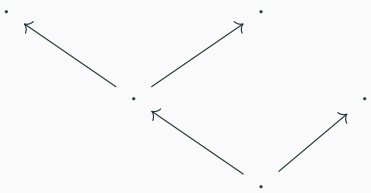
General Idea

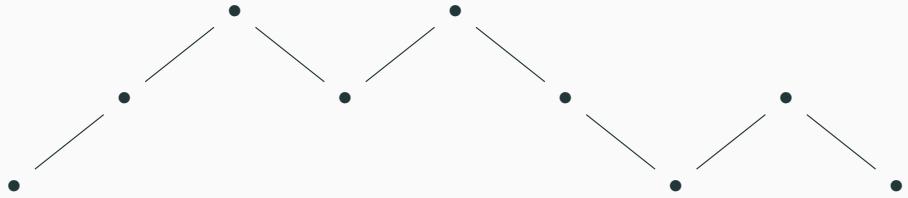
Computads translate a the idea of freeness for higher categories.



Batanin trees and pasting schemes

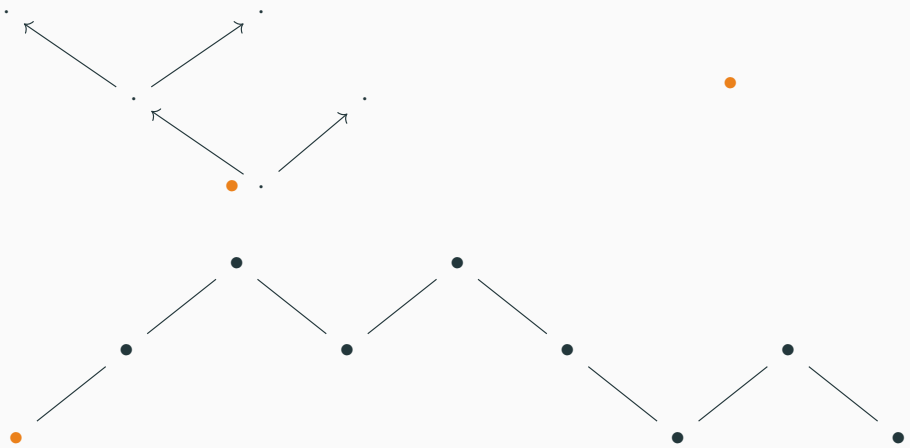
B :=





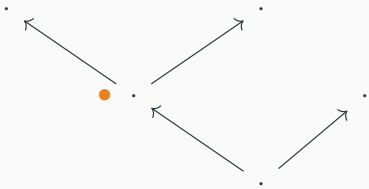
Batanin trees and pasting schemes

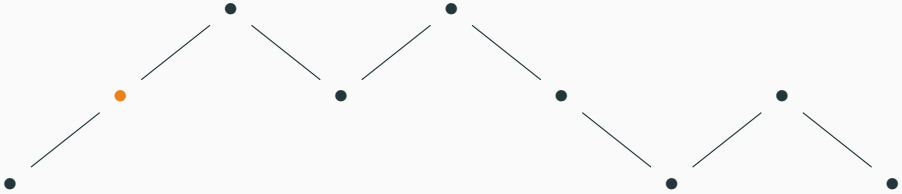
B :=



Batanin trees and pasting schemes

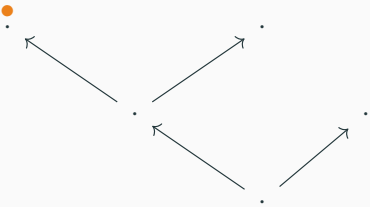
B :=

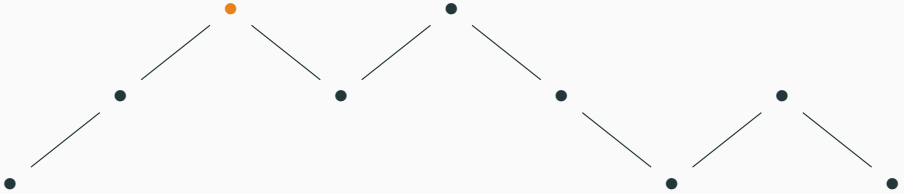




Batanin trees and pasting schemes

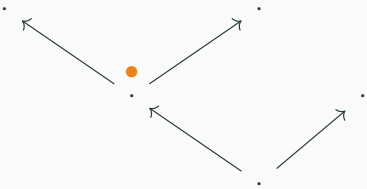
B :=

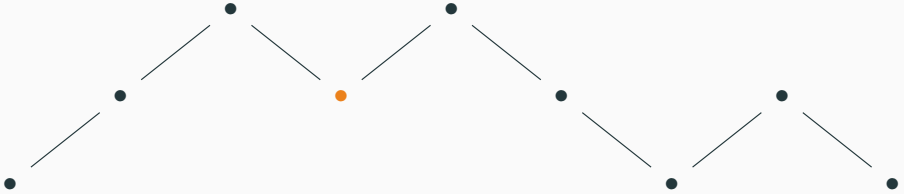




Batanin trees and pasting schemes

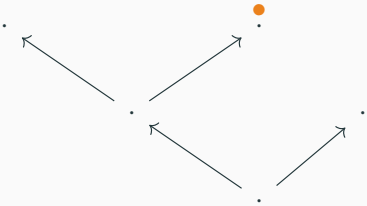
B :=

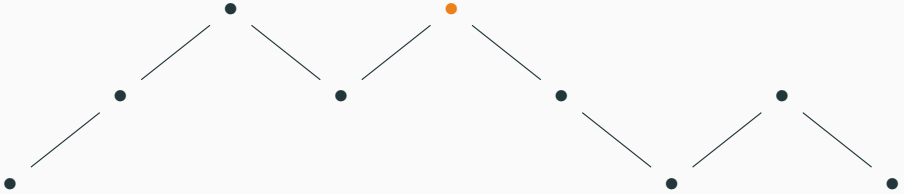




Batanin trees and pasting schemes

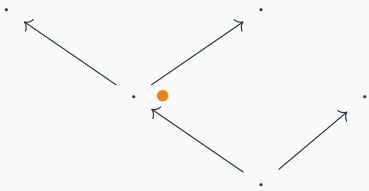
B :=

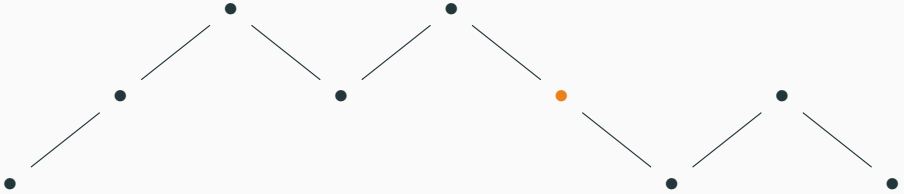




Batanin trees and pasting schemes

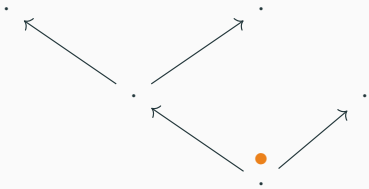
B :=

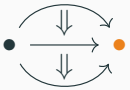


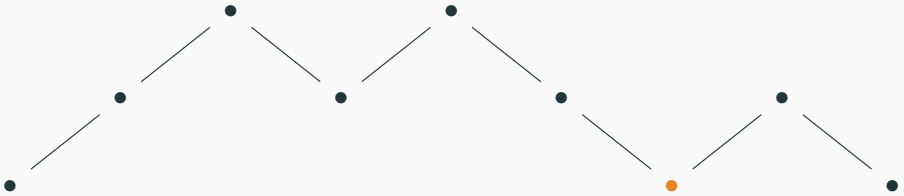


Batanin trees and pasting schemes

B :=

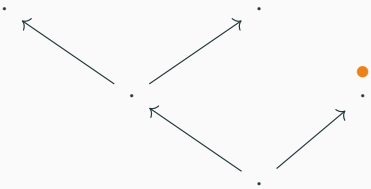


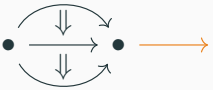


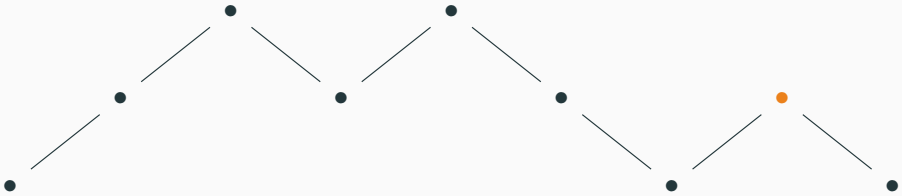


Batanin trees and pasting schemes

B :=

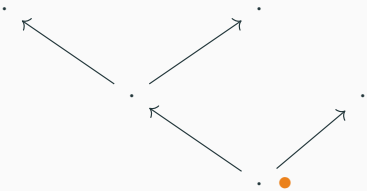


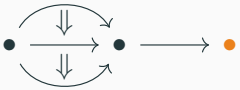


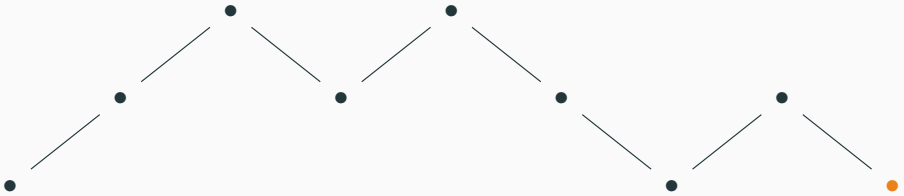


Batanin trees and pasting schemes

B :=

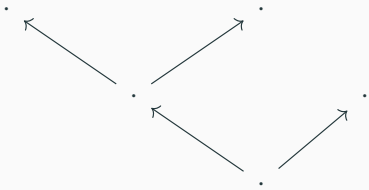




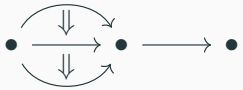


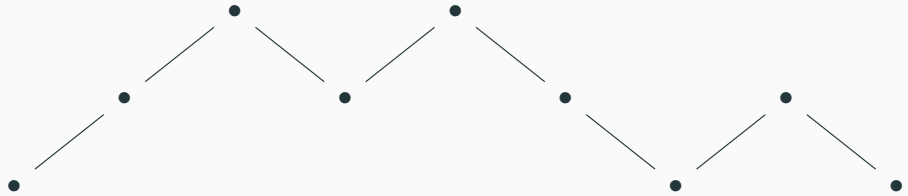
Batanin trees and pasting schemes

B :=



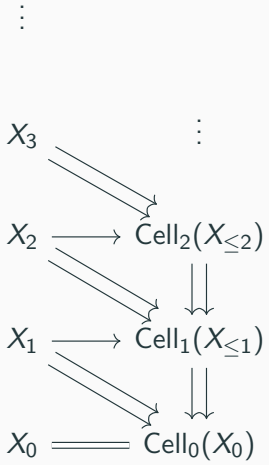
Pos(B) =





Description of $\mathsf{Cell}_n(X_{\leq n})$

The sets $\mathsf{Cell}_n(X_{\leq n})$ and the source and target maps are given by:

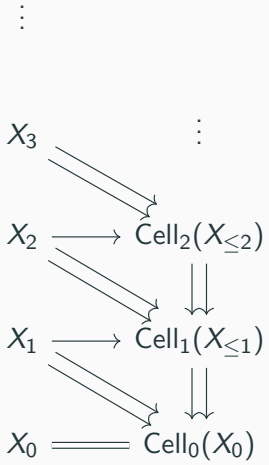


- $\mathrm{var}\, x$ where $x \in X_n$

$$s(\mathrm{var}\, x) = s(x) \qquad t(\mathrm{var}\, x) = t(x)$$

Description of $\mathsf{Cell}_n(X_{\leq n})$

The sets $\mathsf{Cell}_n(X_{\leq n})$ and the source and target maps are given by:



- $\mathrm{var}\, x$ where $x \in X_n$

$$s(\mathrm{var}\, x) = s(x) \qquad t(\mathrm{var}\, x) = t(x)$$

- $\mathrm{coh}(B, (u, v), \gamma)$, where B is a Batanin tree, $u, v \in \mathsf{Cell}_{n-1}(\mathrm{Pos}\, B)$ satisfying conditions similar to `CaTT`, and $\gamma : \mathsf{Cell}(\mathrm{Pos}(B)) \to X$

$$s(\mathrm{coh}(B, (u, v), \gamma)) = \gamma(u)$$
$$t(\mathrm{coh}(B, (u, v), \gamma)) = \gamma(v)$$

The category Comp of computads

- The category Comp
  - **objects:** computads
  - **morphisms:** $f : X \to Y$ = collection of maps $f_n : X_n \to \mathrm{Cell}_n(Y)$ respecting source and target

The category Comp of computads

- The category Comp
  - **objects:** computads
  - **morphisms:** $f : X \to Y$ = collection of maps $f_n : X_n \to \mathrm{Cell}_n(Y)$ respecting source and target

morphisms $\simeq$ substitutions

The category Comp of computads

- The category Comp
  - **objects:** computads
  - **morphisms:** $f : X \to Y$ = collection of maps $f_n : X_n \to \mathrm{Cell}_n(Y)$ respecting source and target

  morphisms $\simeq$ substitutions

- CwF structure on $\mathrm{Comp}^{\mathrm{op}}$:
  - **types:** pair of parallel cells in $\mathrm{Cell}(X)$
  - **terms:** cells in $\mathrm{Cell}(X)$ with prescribed source/target
  - **extension:** adding new generator

From computads to ω-categories

- The descriptions of computads comes with an associated monads on globular sets

From computads to ω-categories

- The descriptions of computads comes with an associated monads on globular sets
- Algebras for this monad are weak ω-categories

From computads to ω-categories

- The descriptions of computads comes with an associated monads on globular sets
- Algebras for this monad are weak ω-categories
- Every ω-category is weakly equivalent to a computad

From computads to ω-categories

- The descriptions of computads comes with an associated monads on globular sets
- Algebras for this monad are weak ω-categories
- Every ω-category is weakly equivalent to a computad
- Computads are the cofibrant objects.

Comparison result

Summary and comparison

CaTT	description of computads
Finite	Infinite
Ordered	Unordered
Structural induction	Induction on dimension
Pasting schemes	Batanin trees
Named variables	Unnamed elements

Summary and comparison

CaTT	description of computads
Finite	Infinite
Ordered	Unordered
Structural induction	Induction on dimension
Pasting schemes	Batanin trees
Named variables	Unnamed elements

- Name of the elements: irrelevant with de Bruijn levels.

Summary and comparison

CaTT	description of computads
Finite	Infinite
Ordered	Unordered
Structural induction	Induction on dimension
Pasting schemes	Batanin trees
Named variables	Unnamed elements

- Name of the elements: irrelevant with de Bruijn levels.
- Pasting schemes and Batanin trees equivalent.

Contribution

Theorem (B., Markakis, Sarti)
There exists a fully faithful morphism of CwF from $\mathcal{S}_{\texttt{CaTT}}$ *to* $\mathsf{Comp}^{\mathsf{op}}$. *Its essential image is made of finite computads.*

`CaTT` **contexts are finite computads**

Conclusion

- Weak ω-categories admit many different definitions, using various tools: monads, coherator, dependent type theory, inductive description of computads.

Conclusion

- Weak ω-categories admit many different definitions, using various tools: monads, coherator, dependent type theory, inductive description of computads.

- Network of connections, that we completed by showing that `CaTT` defines finite computads.

Conclusion

- Weak ω-categories admit many different definitions, using various tools: monads, coherator, dependent type theory, inductive description of computads.
- Network of connections, that we completed by showing that `CaTT` defines finite computads.
- Makes an explicit connection between algebras for the monad and models of the dependent type theory.