Introducing *homotopy.io*: a proof assistant for geometrical higher category theory

Jamie Vicary
Department of Computer Science, University of Cambridge

With Nathan Corbyn, Lukas Heidemann, Nick Hu, David Reutter and Calin Tataru

Topological Quantum Field Theory Club
Instituto Superior Técnico, Portugal
4 May 2022

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

However, working with these structures poses many difficulties.

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

However, working with these structures poses many difficulties.

- How can we formally define the structure we are working with (higher proof, manifold, 2-group, knot, etc)?

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

However, working with these structures poses many difficulties.

- How can we formally define the structure we are working with (higher proof, manifold, 2-group, knot, etc)?
- How can we communicate it to collaborators and readers, and learn more about it ourselves?

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

However, working with these structures poses many difficulties.

- How can we formally define the structure we are working with (higher proof, manifold, 2-group, knot, etc)?
- How can we communicate it to collaborators and readers, and learn more about it ourselves?
- How can we modify it, discover its properties, and prove theorems about it?

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

However, working with these structures poses many difficulties.

- How can we formally define the structure we are working with (higher proof, manifold, 2-group, knot, etc)?
- How can we communicate it to collaborators and readers, and learn more about it ourselves?
- How can we modify it, discover its properties, and prove theorems about it?
- How can we use computers to help us achieve these goals, not only theoretically, but practically?

Working with higher categories

Higher categories are now important in many areas of mathematics and computer science, including homotopy theory, quantum field theory, type theory, and representation theory.

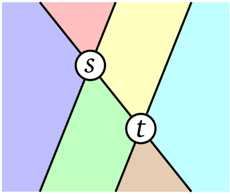
However, working with these structures poses many difficulties.

- How can we formally define the structure we are working with (higher proof, manifold, 2-group, knot, etc)?
- How can we communicate it to collaborators and readers, and learn more about it ourselves?
- How can we modify it, discover its properties, and prove theorems about it?
- How can we use computers to help us achieve these goals, not only theoretically, but practically?

This is the mathematics of the 21st century.
We are only starting to glimpse what is possible.

Graphical calculus

It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



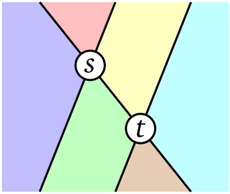
Graphical calculus

It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary ‘commutative diagrams’.

$$\begin{tikzcd}
 & B \arrow[dr] & & \\
A \arrow[ur, bend left] \arrow[dr, bend right] & & C \arrow[dr, bend left] & \\
 & D \arrow[uu, Rightarrow, "s"'] \arrow[ur] \arrow[dr, bend right] & & E \\
 & & f \arrow[uu, Rightarrow, "t"] \arrow[ur, bend right] &
\end{tikzcd}$$

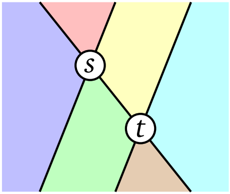
Graphical calculus

It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



Graphical calculus

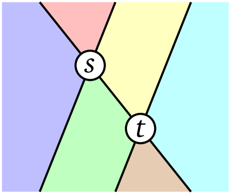
It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



- Idea of formal graphical calculus goes back to Penrose in 1971.

Graphical calculus

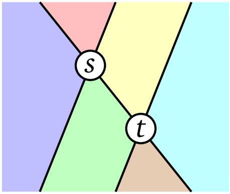
It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



- Idea of formal graphical calculus goes back to Penrose in 1971.
- Strict associativity and unitality comes "built in".

Graphical calculus

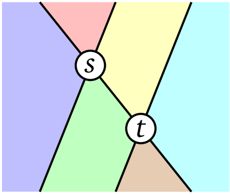
It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



- Idea of formal graphical calculus goes back to Penrose in 1971.
- Strict associativity and unitality comes "built in".
- Joyal and Street developed these ideas for monoidal categories with braiding and symmetry in 1991.

Graphical calculus

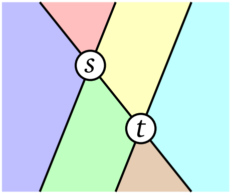
It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



- Idea of formal graphical calculus goes back to Penrose in 1971.
- Strict associativity and unitality comes "built in".
- Joyal and Street developed these ideas for monoidal categories with braiding and symmetry in 1991.
- Extension to Gray categories (special case of $n = 3$) given by Barrett, Meusburger and Schaumann in 2012.

Graphical calculus

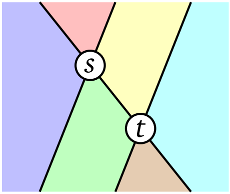
It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary 'commutative diagrams'.



- Idea of formal graphical calculus goes back to Penrose in 1971.
- Strict associativity and unitality comes "built in".
- Joyal and Street developed these ideas for monoidal categories with braiding and symmetry in 1991.
- Extension to Gray categories (special case of $n = 3$) given by Barrett, Meusburger and Schaumann in 2012.
- In higher dimensions, no formal theory has been developed.

Graphical calculus

It is conjectured that n-categories have an n-dimensional graphical calculus, which is the dual of the ordinary ‘commutative diagrams’.



- Idea of formal graphical calculus goes back to Penrose in 1971.
- Strict associativity and unitality comes “built in”.
- Joyal and Street developed these ideas for monoidal categories with braiding and symmetry in 1991.
- Extension to Gray categories (special case of $n = 3$) given by Barrett, Meusburger and Schaumann in 2012.
- In higher dimensions, no formal theory has been developed.
- Nonetheless, regularly used as an informal language.

Homotopy

The graphical calculus suggests *homotopy* as a basic mechanism with which to manipulate terms and execute computations.

Homotopy

The graphical calculus suggests *homotopy* as a basic mechanism with which to manipulate terms and execute computations.

The ‘strictest’ definitions of higher category do not allow these manipulations, and are known to be insufficiently general.

Homotopy

The graphical calculus suggests *homotopy* as a basic mechanism with which to manipulate terms and execute computations.

The 'strictest' definitions of higher category do not allow these manipulations, and are known to be insufficiently general.

At the opposite end of the spectrum, the 'weakest' definitions allow not only these manipulations, but far more besides.

Homotopy

The graphical calculus suggests *homotopy* as a basic mechanism with which to manipulate terms and execute computations.

The ‘strictest’ definitions of higher category do not allow these manipulations, and are known to be insufficiently general.

At the opposite end of the spectrum, the ‘weakest’ definitions allow not only these manipulations, but far more besides.

A model of higher categories is *semistrict* if it is as strict as possible, while still allowing arbitrary homotopy. However, yields long proofs!

Homotopy

The graphical calculus suggests *homotopy* as a basic mechanism with which to manipulate terms and execute computations.

The 'strictest' definitions of higher category do not allow these manipulations, and are known to be insufficiently general.

At the opposite end of the spectrum, the 'weakest' definitions allow not only these manipulations, but far more besides.

A model of higher categories is *semistrict* if it is as strict as possible, while still allowing arbitrary homotopy. However, yields long proofs!

For proof construction, better to be *semiweak*: as weak as possible, except strictly associative and unital. Yields unique composites.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.
- Terms have an immediate geometrical representation.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.
- Terms have an immediate geometrical representation.
- All interaction is by direct manipulation ("point and click").

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.
- Terms have an immediate geometrical representation.
- All interaction is by direct manipulation ("point and click").
- Composition is strictly associative and unital.

The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.
- Terms have an immediate geometrical representation.
- All interaction is by direct manipulation ("point and click").
- Composition is strictly associative and unital.
- All the weak structure is in *homotopies* of composites.

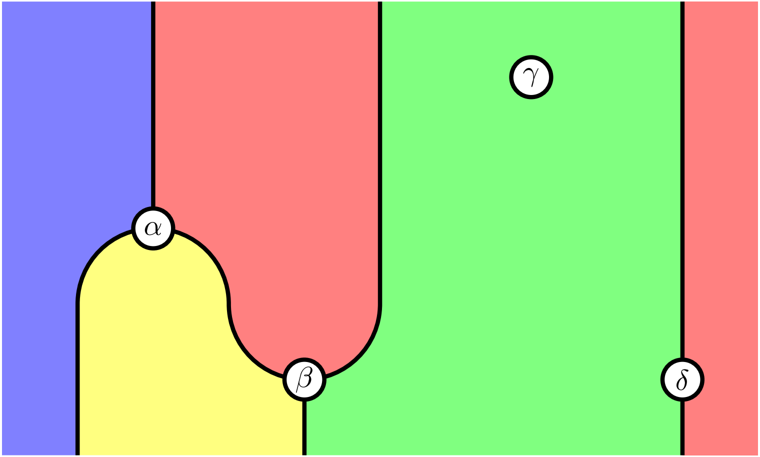
The proof assistant *homotopy.io*

We introduce a proof assistant for semiweak higher category theory, based on an underlying theory called *associative n-categories*.

- The proof assistant lets you build terms in a finitely-presented associative n-category.
- Internal encoding via *zigzags*, a simple combinatorial structure.
- Terms have an immediate geometrical representation.
- All interaction is by direct manipulation ("point and click").
- Composition is strictly associative and unital.
- All the weak structure is in *homotopies* of composites.
- High-level methods assist construction of complex homotopies.

Monotone functions

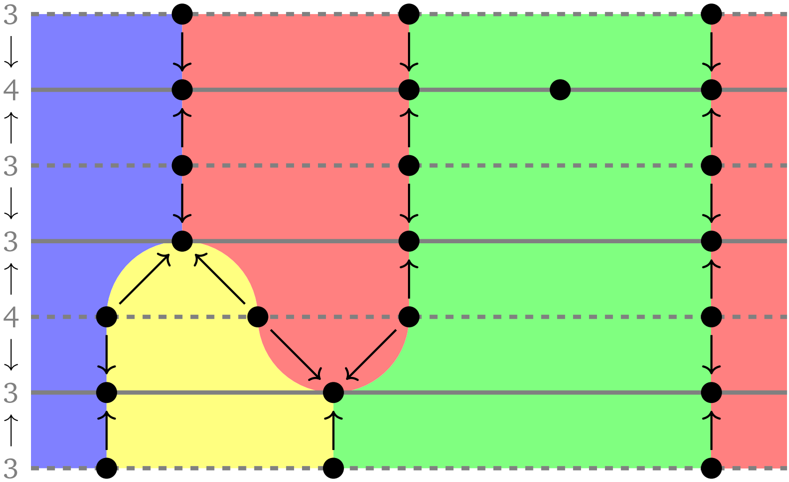
Consider the following string diagram in a bicategory.



Monotone functions

Consider the following string diagram in a bicategory.

It gives rise to an alternating sequence of monotone functions.



Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

- an object X is a sequence of cospans in $\mathcal{C}$;

X $$R_0 \longrightarrow S_0 \longleftarrow R_1 \longrightarrow S_1 \longleftarrow R_2 \longrightarrow S_2 \longleftarrow R_3$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:

$$\begin{array}{ll} X & \qquad\quad R_0 \longrightarrow S_0 \longleftarrow R_1 \longrightarrow S_1 \longleftarrow R_2 \longrightarrow S_2 \longleftarrow R_3 \\ {\scriptstyle f}\big\downarrow & \\ X' & R'_0 \longrightarrow S'_0 \longleftarrow R'_1 \longrightarrow S'_1 \longleftarrow R'_2 \longrightarrow S'_2 \longleftarrow R'_3 \longrightarrow S'_3 \longleftarrow R'_4 \end{array}$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags $Z_{\mathcal{C}}$* as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,

$$\begin{tikzcd}
X \arrow[d, "f"'] & & R_0 \arrow[r] & S_0 \arrow[dl, dashed] & R_1 \arrow[l] \arrow[r] & S_1 \arrow[dr, dashed] & R_2 \arrow[l] \arrow[r] & S_2 \arrow[dl, dashed] & R_3 \arrow[l] & \\
X' & R'_0 \arrow[r] & S'_0 & R'_1 \arrow[l] \arrow[r] & S'_1 & R'_2 \arrow[l] \arrow[r] & S'_2 & R'_3 \arrow[l] \arrow[r] & S'_3 & R'_4 \arrow[l]
\end{tikzcd}$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,

$$
\begin{tikzcd}
X \arrow[d, "f"'] & & R_0 \arrow[r] & S_0 \arrow[dl] & R_1 \arrow[l] \arrow[r] & S_1 \arrow[dr] & R_2 \arrow[l] \arrow[r] & S_2 \arrow[dl] & R_3 \arrow[l] & \\
X' & R'_0 \arrow[r] & S'_0 & R'_1 \arrow[l] \arrow[r] & S'_1 & R'_2 \arrow[l] \arrow[r] & S'_2 & R'_3 \arrow[l] \arrow[r] & S'_3 & R'_4 \arrow[l]
\end{tikzcd}
$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags $Z_{\mathcal{C}}$* as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,
  - with identities between regular levels,

$$
\begin{tikzcd}
X \arrow[d, "f"'] & & R_0 \arrow[r] \arrow[dl, equal] & S_0 \arrow[dl] & R_1 \arrow[l] \arrow[r] \arrow[dl, equal] \arrow[dr, equal] & S_1 \arrow[dr] & R_2 \arrow[l] \arrow[r] & S_2 \arrow[dl] & R_3 \arrow[l] \arrow[dl, equal] \arrow[dr, equal] & \\
X' & R'_0 \arrow[r] & S'_0 & R'_1 \arrow[l] \arrow[r] & S'_1 & R'_2 \arrow[l] \arrow[r] & S'_2 & R'_3 \arrow[l] \arrow[r] & S'_3 & R'_4 \arrow[l]
\end{tikzcd}
$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags $Z_{\mathcal{C}}$* as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,
  - with identities between regular levels,
  - such that all squares commute.

$$\begin{tikzcd}
X \arrow[d, "f"'] & & R_0 \arrow[r] \arrow[dl, equal] \arrow[d, phantom, "\checkmark"] & S_0 \arrow[dl] \arrow[d, phantom, "\checkmark"] & R_1 \arrow[l] \arrow[r] \arrow[dl, equal] \arrow[dr, equal] \arrow[d, phantom, "\checkmark"] & S_1 \arrow[dr] \arrow[d, phantom, "\checkmark"] & R_2 \arrow[l] \arrow[r] \arrow[d, phantom, "\checkmark"] & S_2 \arrow[dl] \arrow[d, phantom, "\checkmark"] & R_3 \arrow[l] \arrow[dl, equal] \arrow[dr, equal] \arrow[d, phantom, "\checkmark"] & \\
X' & R'_0 \arrow[r] & S'_0 & R'_1 \arrow[l] \arrow[r] & S'_1 & R'_2 \arrow[l] \arrow[r] & S'_2 & R'_3 \arrow[l] \arrow[r] & S'_3 & R'_4 \arrow[l]
\end{tikzcd}$$

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,
  - with identities between regular levels,
  - such that all squares commute.

$$
\begin{tikzcd}
X \arrow[d, "f"'] & & R_0 \arrow[r] \arrow[dl, equal] \arrow[d, phantom, "\checkmark"] & S_0 \arrow[dl] \arrow[d, phantom, "\checkmark"] & R_1 \arrow[l] \arrow[r] \arrow[dl, equal] \arrow[dr, equal] \arrow[d, phantom, "\checkmark"] & S_1 \arrow[dr] \arrow[d, phantom, "\checkmark"] & R_2 \arrow[l] \arrow[r] \arrow[d, phantom, "\checkmark"] & S_2 \arrow[dl] \arrow[d, phantom, "\checkmark"] & R_3 \arrow[l] \arrow[dl, equal] \arrow[dr, equal] \arrow[d, phantom, "\checkmark"] & \\
X' & R'_0 \arrow[r] & S'_0 & R'_1 \arrow[l] \arrow[r] & S'_1 & R'_2 \arrow[l] \arrow[r] & S'_2 & R'_3 \arrow[l] \arrow[r] & S'_3 & R'_4 \arrow[l]
\end{tikzcd}
$$

Definition. Write Δ for the category of nonempty finite total orders and monotone functions, Δ_+ when including the empty set, and $\Delta_=$ for the subcategory preserving max & min. Clearly $\Delta_= \hookrightarrow \Delta \hookrightarrow \Delta_+$.

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags* $Z_{\mathcal{C}}$ as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,
  - with identities between regular levels,
  - such that all squares commute.

$$\begin{tikzcd}
X \arrow[dd, "f"'] & & & R_0 \arrow[rr] \arrow[ddll, equal] & & S_0 \arrow[ddll] & & R_1 \arrow[ll] \arrow[rr] \arrow[ddll, equal] \arrow[ddrr, equal] & & S_1 \arrow[ddrr] & & R_2 \arrow[ll] \arrow[rr] & & S_2 \arrow[ddll] & & R_3 \arrow[ll] \arrow[ddll, equal] \arrow[ddrr, equal] & & \\
& & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \\
X' & R'_0 \arrow[rr] & & S'_0 & & R'_1 \arrow[ll] \arrow[rr] & & S'_1 & & R'_2 \arrow[ll] \arrow[rr] & & S'_2 & & R'_3 \arrow[ll] \arrow[rr] & & S'_3 & & R'_4 \arrow[ll]
\end{tikzcd}$$

Definition. Write Δ for the category of nonempty finite total orders and monotone functions, Δ_+ when including the empty set, and $\Delta_=$ for the subcategory preserving max & min. Clearly $\Delta_= \hookrightarrow \Delta \hookrightarrow \Delta_+$.

Lemma. There is an equivalence of categories $\Delta_+ \simeq (\Delta_=)^{\mathrm{op}}$.

Zigzags

Definition. For a category $\mathcal{C}$, we define its *category of zigzags $Z_{\mathcal{C}}$* as:

- an object X is a sequence of cospans in $\mathcal{C}$;
- a morphism $f : X \to X'$ is:
  - a monotone function between singular levels,
  - built from morphisms of $\mathcal{C}$,
  - with identities between regular levels,
  - such that all squares commute.

$$\begin{tikzcd}
X \arrow[dd, "f"'] & & & R_0 \arrow[rr] \arrow[ddll, equal] & & S_0 \arrow[ddll] & & R_1 \arrow[ll] \arrow[rr] \arrow[ddll, equal] \arrow[ddrr, equal] & & S_1 \arrow[ddrr] & & R_2 \arrow[ll] \arrow[rr] & & S_2 \arrow[ddll] & & R_3 \arrow[ll] \arrow[ddll, equal] \arrow[ddrr, equal] & & \\
& & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \checkmark & & \\
X' & R'_0 \arrow[rr] & & S'_0 & & R'_1 \arrow[ll] \arrow[rr] & & S'_1 & & R'_2 \arrow[ll] \arrow[rr] & & S'_2 & & R'_3 \arrow[ll] \arrow[rr] & & S'_3 & & R'_4 \arrow[ll]
\end{tikzcd}$$

Definition. Write Δ for the category of nonempty finite total orders and monotone functions, Δ_+ when including the empty set, and $\Delta_=$ for the subcategory preserving max & min. Clearly $\Delta_= \hookrightarrow \Delta \hookrightarrow \Delta_+$.

Lemma. There is an equivalence of categories $\Delta_+ \simeq (\Delta_=)^{\mathrm{op}}$.

Thus we obtain $S : Z_{\mathcal{C}} \to \Delta_+$ and $R : Z_{\mathcal{C}}^{\mathrm{op}} \to \Delta_=$.

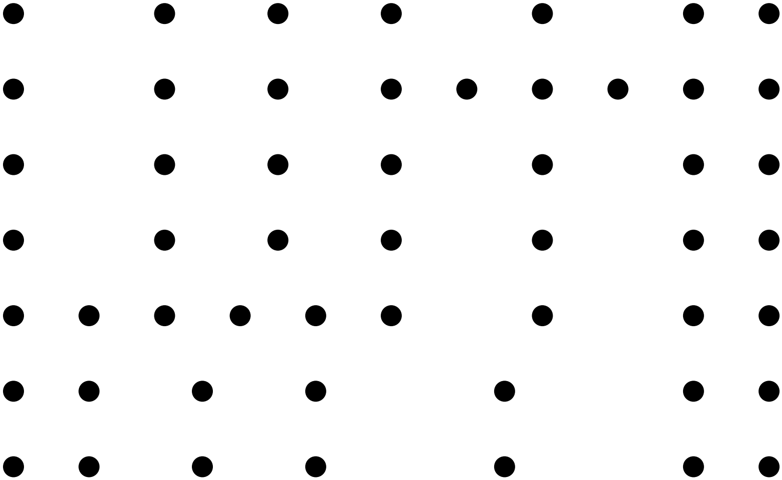
Diagrams from zigzags

Definition. An *untyped n-diagram* is an object of $Z_1^n := Z_{Z_{\cdots Z_1}}$.

Diagrams from zigzags

Definition. An *untyped n-diagram* is an object of $Z_1^n := Z_{Z_{\cdots Z_1}}$.

Here are 53 examples of 0-diagrams, all objects of 1:



Diagrams from zigzags

Definition. An *untyped n-diagram* is an object of $Z_1^n := Z_{Z_{\cdots Z_1}}$.

Here are 7 examples of 1-diagrams, all objects of $Z_1 = \Delta_+$:

$$\bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet$$

$$\bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet$$

$$\bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet$$

$$\bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet$$

$$\bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet \leftarrow \bullet \rightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \rightarrow \bullet$$

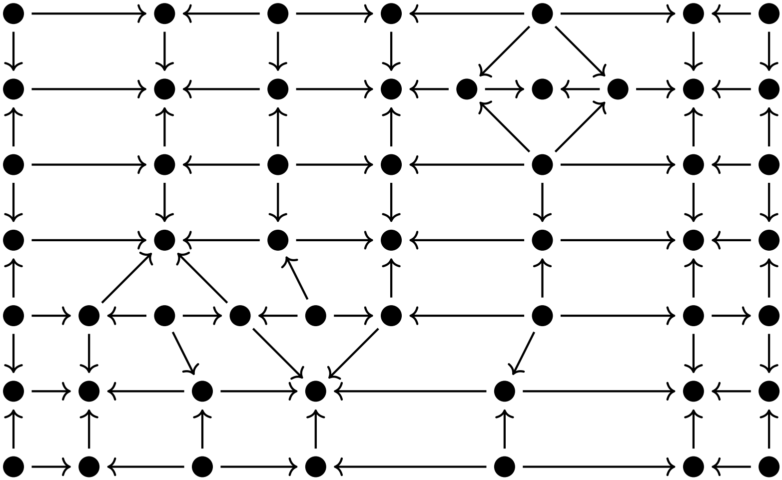
$$\bullet \rightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet$$

$$\bullet \rightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \longleftarrow \bullet \longrightarrow \bullet \leftarrow \bullet$$

Diagrams from zigzags

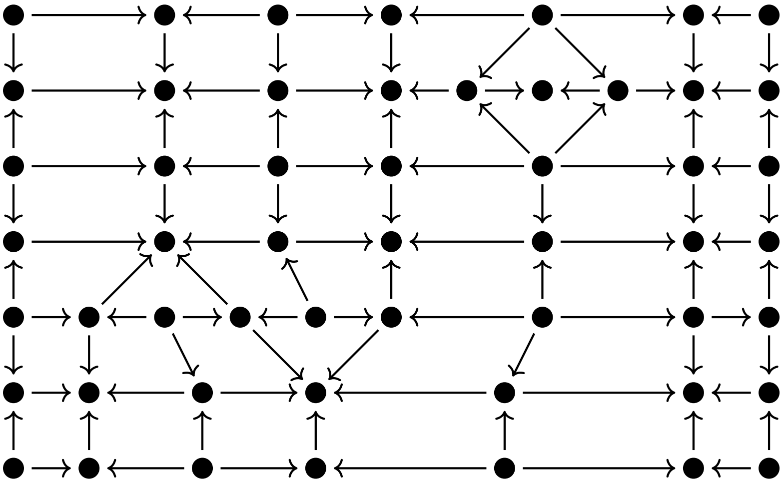
Definition. An *untyped n-diagram* is an object of $Z_1^n := Z_{Z_{\cdots Z_1}}$.

Here is 1 example of a 2-diagram, an object of Z_{Δ_+}:



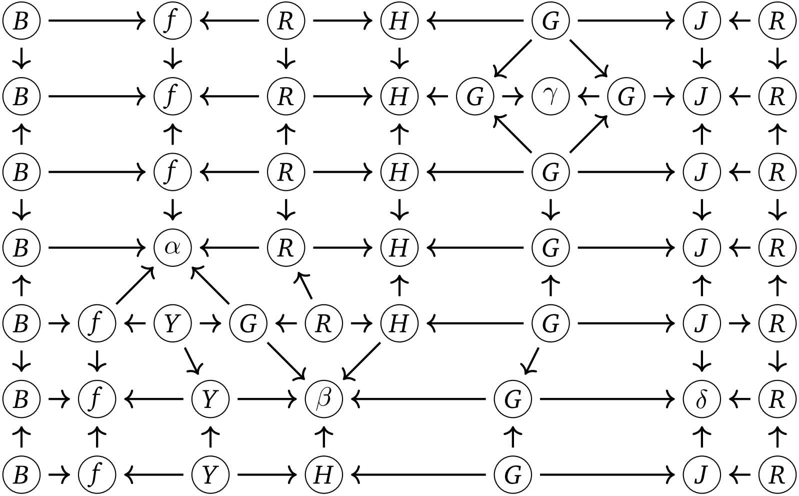
Types

These diagrams are *untyped*.



Types

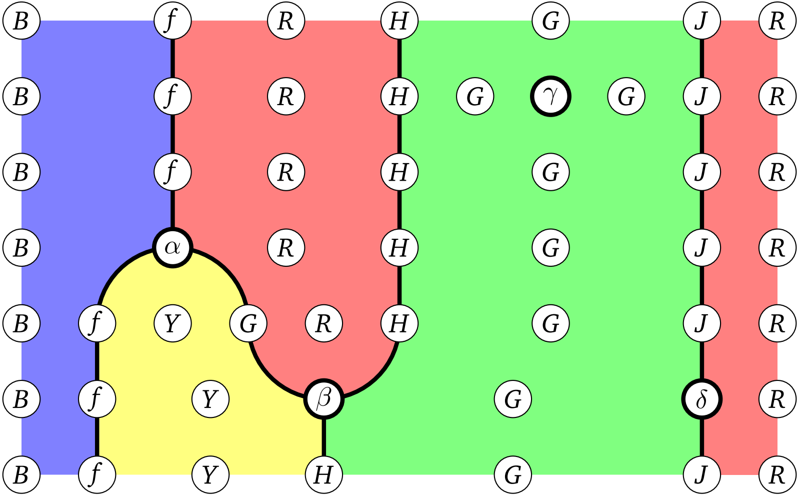
These diagrams are *untyped*. To add types, we decorate with labels.



Types

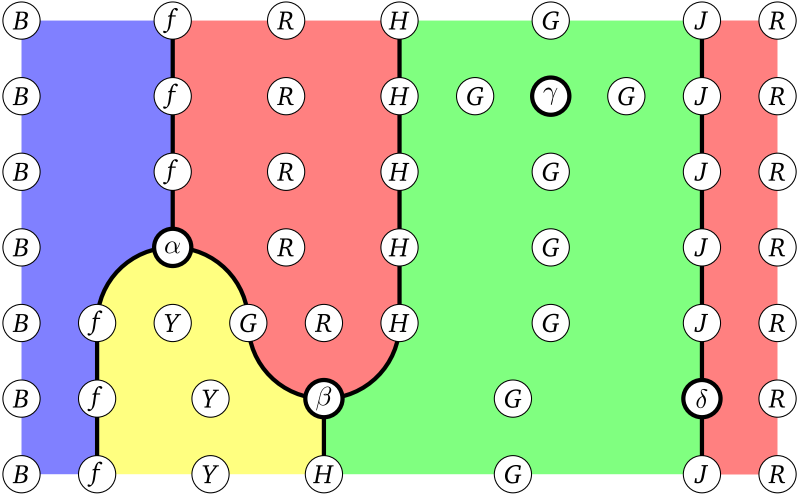
These diagrams are *untyped*. To add types, we decorate with labels.

The standard graphical calculus is just a prettier version of this.



Types

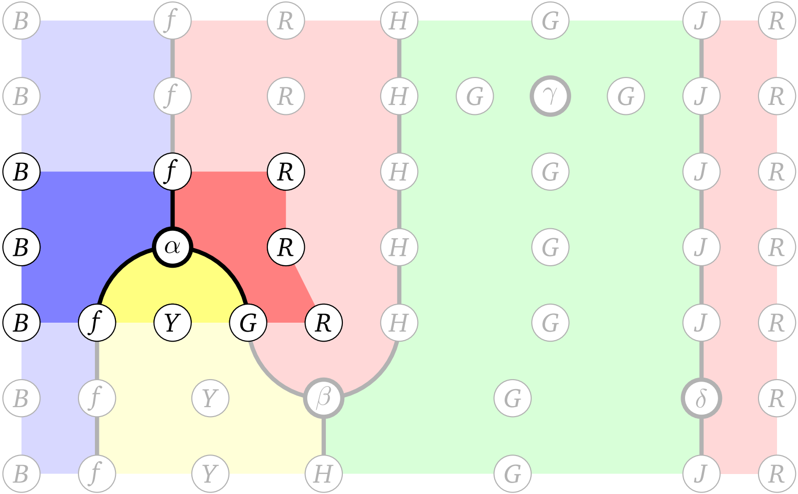
These diagrams are *untyped*. To add types, we decorate with labels. The standard graphical calculus is just a prettier version of this.



Type checking verifies that label neighbourhoods match definition.

Types

These diagrams are *untyped*. To add types, we decorate with labels. The standard graphical calculus is just a prettier version of this.

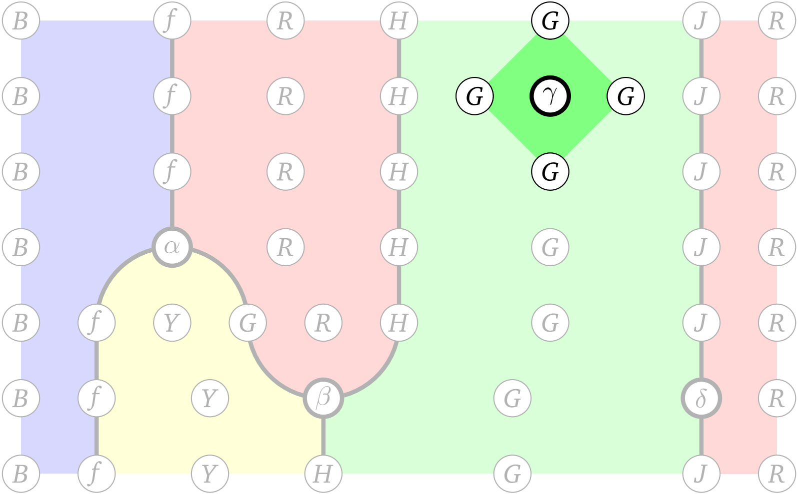


Type checking verifies that label neighbourhoods match definition.

Types

These diagrams are *untyped*. To add types, we decorate with labels.

The standard graphical calculus is just a prettier version of this.



Type checking verifies that label neighbourhoods match definition.

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

$\boxed{\text{HoTT}}$

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

homotopy.io

HoTT

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)

homotopy.io | $\mathrm{Catt}_{\mathrm{sua}}$ | Catt | HoTT

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)

homotopy.io	$\mathrm{Catt}_{\mathrm{sua}}$	Catt	HoTT

The goal is to understand the relationships between these, with conversation algorithms to move proofs between models.

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)

$$\begin{tikzcd} \boxed{\text{homotopy.io}} & \boxed{\mathrm{Catt}_{\mathrm{sua}}} \arrow[r, "\checkmark"] & \boxed{\mathrm{Catt}} \arrow[r, "\checkmark\!\sim"] & \boxed{\mathrm{HoTT}} \end{tikzcd}$$

The goal is to understand the relationships between these, with conversation algorithms to move proofs between models.

$\checkmark$ = done

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)

$$\begin{tikzcd} \boxed{\text{homotopy.io}} & \boxed{\mathrm{Catt}_{\mathrm{sua}}} \arrow[r, shift left, "\checkmark"] & \boxed{\mathrm{Catt}} \arrow[l, shift left, "\cdots"] \arrow[r, "\checkmark\sim"] & \boxed{\mathrm{HoTT}} \end{tikzcd}$$

The goal is to understand the relationships between these, with conversation algorithms to move proofs between models.

$\checkmark$ = done $\quad\quad$ $\cdots$ = in progress

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)

$$
\begin{tikzcd}
\boxed{\text{homotopy.io}} \arrow[r, shift left, "?"] & \boxed{\mathrm{Catt}_{\mathrm{sua}}} \arrow[l, shift left, "?"] \arrow[r, shift left, "\checkmark"] & \boxed{\mathrm{Catt}} \arrow[l, shift left, "\cdots"] \arrow[r, shift left, "\checkmark\sim"] & \boxed{\mathrm{HoTT}} \arrow[l, shift left, "?"]
\end{tikzcd}
$$

The goal is to understand the relationships between these, with conversation algorithms to move proofs between models.

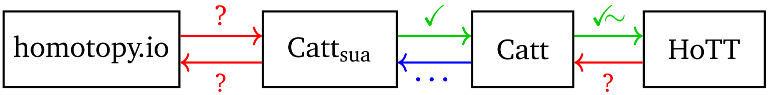
$\checkmark$ = done $\cdots$ = in progress ? = too hard (for now!)

Relationship to type theory

In Martin-Löf type theory, coherences for path types (such as associators, unitors) must be inserted *by hand* where necessary.

The *homotopy.io* project is a geometrical system where some of this structure is trivialized.

Separate project on type theories for semistrict higher category theory (with Eric Finster, David Reutter, Alex Rice.)



The goal is to understand the relationships between these, with conversation algorithms to move proofs between models.

✓ = done $\qquad \cdots$ = in progress $\qquad$? = too hard (for now!)

Thanks for listening!

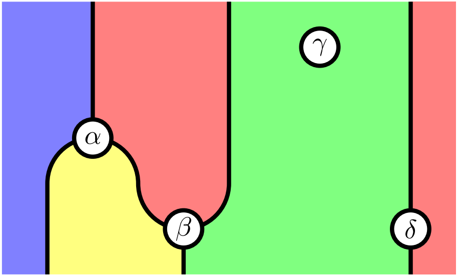
Type checking

Definition. Given an untyped n-diagram D, its *volume* $|D|$ is defined to be 1 if $n = 0$, or else we define $|D| = \sum_i |S_i|$, the sum of the volumes of the singular levels.

Type checking

Definition. Given an untyped n-diagram D, its *volume* $|D|$ is defined to be 1 if $n = 0$, or else we define $|D| = \sum_i |S_i|$, the sum of the volumes of the singular levels.

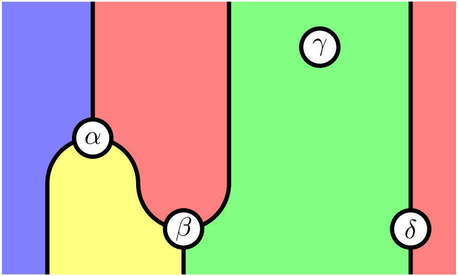
This 2-diagram has $|D| = 10$, meaning it has 10 "pieces":



Type checking

Definition. Given an untyped n-diagram D, its *volume* $|D|$ is defined to be 1 if $n = 0$, or else we define $|D| = \sum_i |S_i|$, the sum of the volumes of the singular levels.

This 2-diagram has $|D| = 10$, meaning it has 10 "pieces":



Essentially, type checking verifies that these "pieces" are either identities, or elements of the signature.

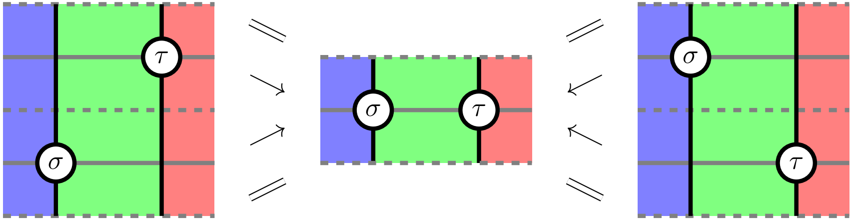
Type checking

But there's a complication: these pieces will in general need "normalizing" before type checking can occur.

Type checking

But there's a complication: these pieces will in general need "normalizing" before type checking can occur.

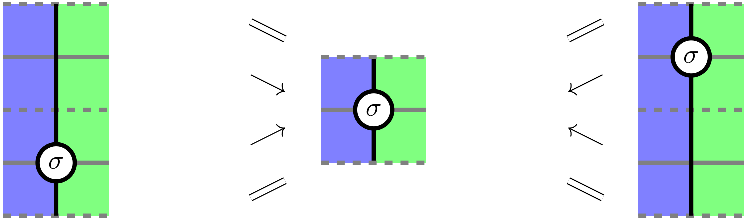
Consider this 3-diagram, an interchanger with volume 2:



Type checking

But there's a complication: these pieces will in general need "normalizing" before type checking can occur.

Consider this 3-diagram, an interchanger with volume 2:



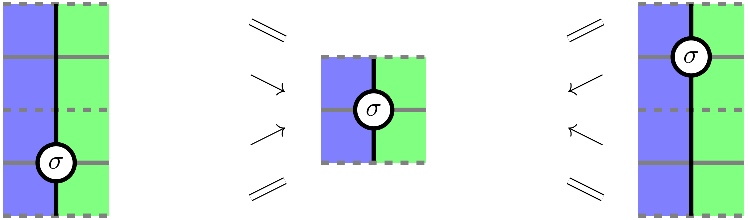
Here is the first piece. Note it has volume 1 as required.

It contains "unnecessary" identities on the left and right. As a result, it is not itself an identity.

Type checking

But there's a complication: these pieces will in general need "normalizing" before type checking can occur.

Consider this 3-diagram, an interchanger with volume 2:



Here is the first piece. Note it has volume 1 as required.

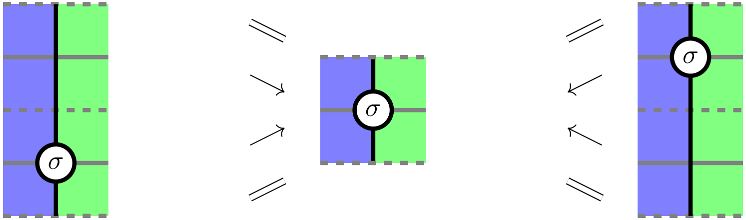
It contains "unnecessary" identities on the left and right. As a result, it is not itself an identity.

We must *normalize* this before it can be checked against the signature.

Type checking

But there's a complication: these pieces will in general need "normalizing" before type checking can occur.

Consider this 3-diagram, an interchanger with volume 2:



Here is the first piece. Note it has volume 1 as required.

It contains "unnecessary" identities on the left and right. As a result, it is not itself an identity.

We must *normalize* this before it can be checked against the signature.

In this way, homotopies come "built in".

High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

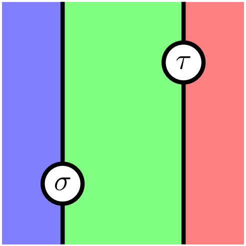
Consider the problem of constructing the homotopy that *contracts* this composite:

High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

Consider the problem of constructing the homotopy that *contracts* this composite:



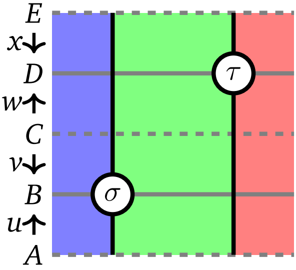
High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

Consider the problem of constructing the homotopy that *contracts* this composite:

$$\begin{tikzcd} & B & & D & \\ A \arrow[ur, "u"] & & C \arrow[ul, "v"'] \arrow[ur, "w"] & & E \arrow[ul, "x"'] \end{tikzcd}$$



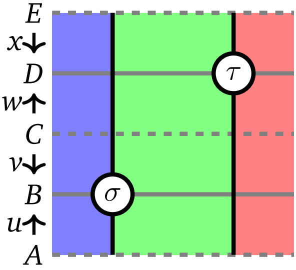
High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

Consider the problem of constructing the homotopy that *contracts* this composite:

$$\begin{tikzcd}
 & & F & & \\
 & B \arrow[ur, "y"] & \arrow[u, phantom, "\lrcorner", rotate=-45] & D \arrow[ul, "z"'] & \\
A \arrow[ur, "u"] & & C \arrow[ul, "v"'] \arrow[ur, "w"] & & E \arrow[ul, "x"']
\end{tikzcd}$$



We build the contraction as a pushout of cospans in Z_1^n, and the homotopy itself as an associated zigzag map.

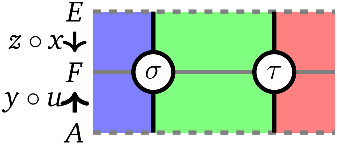
High-level methods

Zigzags are too unwieldy for direct construction by hand.

High-level methods are needed to build nontrivial homotopies.

Consider the problem of constructing the homotopy that *contracts* this composite:

$$\begin{tikzcd}
 & & F & & \\
 & B \arrow[ur, "y"] & & D \arrow[ul, "z"'] & \\
A \arrow[ur, "u"] & & C \arrow[ul, "v"'] \arrow[ur, "w"] & & E \arrow[ul, "x"']
\end{tikzcd}$$



We build the contraction as a pushout of cospans in Z_1^n, and the homotopy itself as an associated zigzag map.

Colimit algorithm

(With David Reutter.)

Theorem. For a category **C**, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

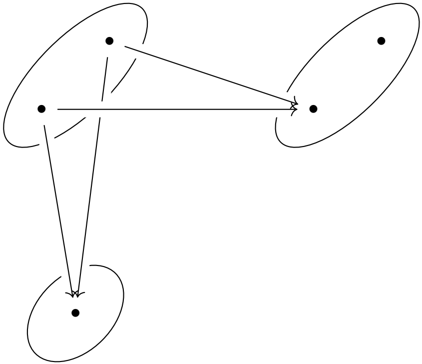
$$\begin{tikzcd} Z \arrow[r, "f"] \arrow[d, "g"'] & Z' \\ Z'' & \end{tikzcd}$$

Colimit algorithm

(With David Reutter.)

Theorem. For a category $\mathbf{C}$, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

(1) Project to Δ.



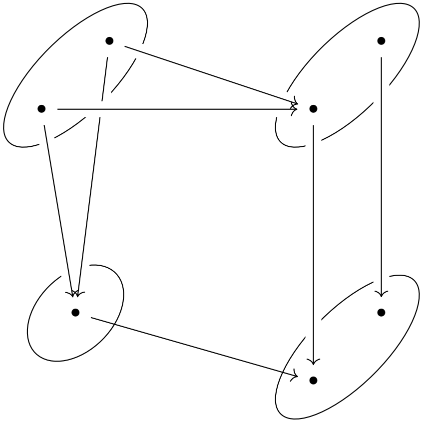
Colimit algorithm

(With David Reutter.)

Theorem. For a category **C**, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

(1) Project to Δ.

(2) Take colimit there, or fail.



Colimit algorithm

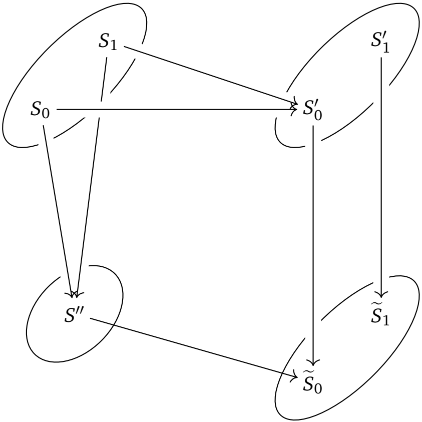
(With David Reutter.)

Theorem. For a category **C**, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

(1) Project to Δ.

(2) Take colimit there, or fail.

(3) Label with colimits of underlying **C**-morphisms, or fail.



Colimit algorithm

(With David Reutter.)

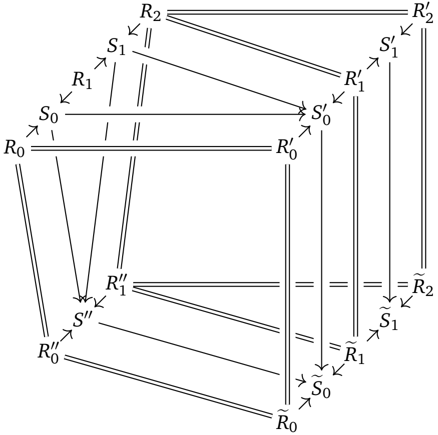
Theorem. For a category **C**, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

(1) Project to Δ.

(2) Take colimit there, or fail.

(3) Label with colimits of underlying **C**-morphisms, or fail.

(4) Commutativity conditions automatically satisfied.



Colimit algorithm

(With David Reutter.)

Theorem. For a category **C**, the following correctly constructs colimits in $Z_{\mathbf{C}}$, or correctly fails:

(1) Project to Δ.

(2) Take colimit there, or fail.

(3) Label with colimits of underlying **C**-morphisms, or fail.

(4) Commutativity conditions automatically satisfied.

(5) Type check the result.