Towards Higher Universal Algebra in Type Theory

HoTT Electronic Seminar Talks

Eric Finster

December 6, 2018

Voevodsky's Vision for Univalent Mathematics

Voevodsky's Vision for Univalent Mathematics

h-level 0 The Mathematics of Cantor

- Sets and structured sets

Voevodsky's Vision for Univalent Mathematics

h-level 0 The Mathematics of Cantor

- Sets and structured sets

h-level 1 The Mathematics of Grothendieck

- Groupoids and structured groupoids

Voevodsky's Vision for Univalent Mathematics

h-level 0 The Mathematics of Cantor

- Sets and structured sets

h-level 1 The Mathematics of Grothendieck

- Groupoids and structured groupoids
- In particular the theory of *categories*

Voevodsky's Vision for Univalent Mathematics

h-level 0 The Mathematics of Cantor

- Sets and structured sets

h-level 1 The Mathematics of Grothendieck

- Groupoids and structured groupoids
- In particular the theory of *categories*

h-level ∞ "Higher" Mathematics

- The study of structured *homotopy types*

Voevodsky's Vision for Univalent Mathematics

h-level 0 The Mathematics of Cantor

- Sets and structured sets

h-level 1 The Mathematics of Grothendieck

- Groupoids and structured groupoids
- In particular the theory of *categories*

h-level ∞ "Higher" Mathematics

- The study of structured *homotopy types*

Problem

How can we describe structures on homotopy types without recourse to a "strict" equality?

The Current State of Affairs

- Solutions in some special cases are known:

The Current State of Affairs

- Solutions in some special cases are known:

  Voevodsky Contractibility, equivalences, ...

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case
- There are many other kinds of higher structures:

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case
- There are many other kinds of higher structures:
  - E_n-spaces, ring spectra, homotopy Lie algebras, ...

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case
- There are many other kinds of higher structures:
  - E_n-spaces, ring spectra, homotopy Lie algebras, ...
  - (∞, n)-categories, ∞-double categories, ...

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case
- There are many other kinds of higher structures:
  - E_n-spaces, ring spectra, homotopy Lie algebras, ...
  - (∞, n)-categories, ∞-double categories, ...
  - Even if these can be reduced to simplicial methods, will this be an efficient way to describe them?

The Current State of Affairs

- Solutions in some special cases are known:

Voevodsky	Contractibility, equivalences, ...
Shulman	∞-idempotents
Rijke	∞-equivalence relations

- Long standing approach to the problem:
  - Construct some notion of *semi-simplicial type*
  - Use this to internalize the theory of $(\infty, 1)$-categories
  - Reduce other coherence problems to this case
- There are many other kinds of higher structures:
  - E_n-spaces, ring spectra, homotopy Lie algebras, ...
  - (∞, n)-categories, ∞-double categories, ...
  - Even if these can be reduced to simplicial methods, will this be an efficient way to describe them?
  - Can we describe a natural class of higher structures *directly*?

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
    1. $(\infty, 1)$-operad

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
    1. $(\infty, 1)$-operad
    2. $(\infty, 1)$-category

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
  1. $(\infty, 1)$-operad
  2. $(\infty, 1)$-category
  3. ∞-groupoid

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
  1. $(\infty, 1)$-operad
  2. $(\infty, 1)$-category
  3. ∞-groupoid
- There is a corresponding elementary defintion of an *algebra*

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
  1. $(\infty, 1)$-operad
  2. $(\infty, 1)$-category
  3. ∞-groupoid
- There is a corresponding elementary defintion of an *algebra*
- Special cases of this definition are
  1. A_∞-types, E_∞-types, etc

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
  1. $(\infty, 1)$-operad
  2. $(\infty, 1)$-category
  3. ∞-groupoid
- There is a corresponding elementary defintion of an *algebra*
- Special cases of this definition are
  1. A_∞-types, E_∞-types, etc
  2. Type-valued diagrams on $(\infty, 1)$-categories

In this talk ...

- Adapt Baez and Dolan's operadic method of describing coherent algebraic objects to type theory
- Give an elementary definition of *cartesian polynomial monad*
- Special cases of this definition are
  1. $(\infty, 1)$-operad
  2. $(\infty, 1)$-category
  3. ∞-groupoid
- There is a corresponding elementary defintion of an *algebra*
- Special cases of this definition are
  1. A_∞-types, E_∞-types, etc
  2. Type-valued diagrams on $(\infty, 1)$-categories
  3. Corollary: simplicial types are definable in MLTT with coinduction.

Formalization

Where are we in terms of formalization?

Formalization

Where are we in terms of formalization?

- The formalization of the definition of monad given here is complete.

  `https://github.com/ericfinster/higher-alg`

Formalization

Where are we in terms of formalization?

- The formalization of the definition of monad given here is complete.

  `https://github.com/ericfinster/higher-alg`
- Hence so are any of the definitions which are special cases: ∞-operad, ∞-category, ∞-groupoid, ...

Formalization

Where are we in terms of formalization?

- The formalization of the definition of monad given here is complete.

  `https://github.com/ericfinster/higher-alg`
- Hence so are any of the definitions which are special cases: ∞-operad, ∞-category, ∞-groupoid, ...
- The definition of algebra relies on a construction which is not yet completely formalized (though it is sketched ...)

Formalization

Where are we in terms of formalization?

- The formalization of the definition of monad given here is complete.

  `https://github.com/ericfinster/higher-alg`
- Hence so are any of the definitions which are special cases: ∞-operad, ∞-category, ∞-groupoid, ...
- The definition of algebra relies on a construction which is not yet completely formalized (though it is sketched ...)
- Hence the complete definition of simplicial type is not yet finished.

Formalization

Where are we in terms of formalization?

- The formalization of the definition of monad given here is complete.

  `https://github.com/ericfinster/higher-alg`
- Hence so are any of the definitions which are special cases: ∞-operad, ∞-category, ∞-groupoid, ...
- The definition of algebra relies on a construction which is not yet completely formalized (though it is sketched ...)
- Hence the complete definition of simplicial type is not yet finished.
- The "on paper" definition of algebra, however, is completely transparent. I do not expect any difficulties in finishing it other than the fact that it is somewhat long.

Polynomials as Multi-sorted Signatures

Definition

Fix a type I of *sorts*. A *polynomial* over I is the data of

Polynomials as Multi-sorted Signatures

Definition

Fix a type I of *sorts*. A *polynomial* over I is the data of

1. A family of *operations*

$$\mathrm{Op} : I \to \mathit{Type}$$

Polynomials as Multi-sorted Signatures

Definition

Fix a type I of *sorts*. A *polynomial* over I is the data of

1. A family of *operations*
$$\mathrm{Op} : I \to \mathit{Type}$$
2. For each operation, a family of sorted *parameters*
$$\mathrm{Param} : \{i : I\}(f : \mathrm{Op}\, i) \to I \to \mathit{Type}$$

Polynomials as Multi-sorted Signatures

Definition

Fix a type I of *sorts*. A *polynomial* over I is the data of

1. A family of *operations*
$$\mathrm{Op} : I \to \mathit{Type}$$
2. For each operation, a family of sorted *parameters*
$$\mathrm{Param} : \{i : I\}(f : \mathrm{Op}\, i) \to I \to \mathit{Type}$$

- For $i : I$, an element $f : \mathrm{Op}\, i$ represents an operation whose *output* sort is i.

Polynomials as Multi-sorted Signatures

Definition

Fix a type I of *sorts*. A *polynomial* over I is the data of

1. A family of *operations*
$$\mathrm{Op} : I \to \mathit{Type}$$
2. For each operation, a family of sorted *parameters*
$$\mathrm{Param} : \{i : I\}(f : \mathrm{Op}\, i) \to I \to \mathit{Type}$$

- For $i : I$, an element $f : \mathrm{Op}\, i$ represents an operation whose *output* sort is i.
- For $f : \mathrm{Op}\, i$ and $j : I$, an element $p : \mathrm{Param}\, f\, j$ represents an *input* parameter of sort j.

Representations of Operations

- We can think of our polynomial as a collection of *typed operation symbols*, which we might denote, for example, by

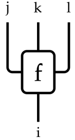
$$f(j, k, l) : i$$

Representations of Operations

- We can think of our polynomial as a collection of *typed operation symbols*, which we might denote, for example, by

$$f(j,k,l) : i$$

- We can depict such an operation graphically as a corolla:

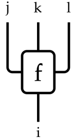


Representations of Operations

- We can think of our polynomial as a collection of *typed operation symbols*, which we might denote, for example, by

$$f(j, k, l) : i$$

- We can depict such an operation graphically as a corolla:



- However, we specifically allow for higher homotopy both in the operations and the parameters

Trees

A polynomial P : Poly I generates an associated type of *trees*.

Trees

A polynomial $P : \mathrm{Poly}\, I$ generates an associated type of *trees*.

Definition

The inductive family $\mathrm{Tr}\ P : I \to \mathit{Type}$ has constructors:

$$
\begin{aligned}
\mathrm{lf} &: (i : I) \to \mathrm{Tr}\ P\ i \\
\mathrm{nd} &: \{i : I\} \to (f : \mathrm{Op}\, P i) \\
&\to (\phi : (j : J)(p : \mathrm{Param}\, f\, j) \to \mathrm{Tr}\, P\, j) \\
&\to \mathrm{Tr}\, P\, i
\end{aligned}
$$

Trees

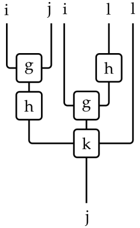
A polynomial P : Poly I generates an associated type of *trees*.

Definition

The inductive family Tr $P : I \to \mathit{Type}$ has constructors:

$$\begin{aligned}
\mathrm{lf} &: (i : I) \to \mathrm{Tr}\ P\ i \\
\mathrm{nd} &: \{i : I\} \to (f : \mathrm{Op}\ P i) \\
&\to (\phi : (j : J)(p : \mathrm{Param}\ f\ j) \to \mathrm{Tr}\ P\ j) \\
&\to \mathrm{Tr}\ P\ i
\end{aligned}$$

We can represent trees both *geometrically*



Trees

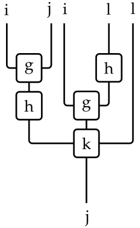
A polynomial P : Poly I generates an associated type of *trees*.

Definition

The inductive family Tr $P : I \to \mathit{Type}$ has constructors:

$$\begin{aligned} \mathrm{lf} : (i : I) &\to \mathrm{Tr}\ P\ i \\ \mathrm{nd} : \{i : I\} &\to (f : \mathrm{Op}\ P i) \\ &\to (\phi : (j : J)(p : \mathrm{Param}\ f\ j) \to \mathrm{Tr}\ P\ j) \\ &\to \mathrm{Tr}\ P\ i \end{aligned}$$

We can represent trees both *geometrically* and *algebraically*



$$k(h(g(i, j), g(i, h(l)), l) : j$$

Leaves and Nodes

For a tree $w : \mathrm{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves and Nodes

For a tree $w : \mathrm{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathrm{Leaf} : \{i : I\}(w : \mathrm{Tr}\ i) \to I \to \mathit{Type}$$
$$\mathrm{Leaf}\,(\mathrm{lf}\ i)\, j :=$$

Leaves and Nodes

For a tree $w : \mathrm{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathrm{Leaf} : \{i : I\}(w : \mathrm{Tr}\ i) \to I \to \mathit{Type}$$
$$\mathrm{Leaf}\,(\mathrm{lf}\, i)\, j := i = j$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$
\begin{aligned}
&\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type} \\
&\mathsf{Leaf}\,(\mathsf{lf}\ i)\, j := i = j \\
&\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j :=
\end{aligned}
$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\begin{aligned}
&\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type} \\
&\mathsf{Leaf}\,(\mathsf{lf}\, i)\, j := i = j \\
&\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j := \sum_{k:I} \sum_{p:\mathsf{Param}\ f\ k}
\end{aligned}$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\begin{aligned}
&\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type} \\
&\mathsf{Leaf}\,(\mathsf{lf}\ i)\, j := i = j \\
&\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j := \sum_{k:I} \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\, k\, p)\, j
\end{aligned}$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$
\begin{aligned}
&\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type} \\
&\mathsf{Leaf}\,(\mathsf{lf}\ i)\, j := i = j \\
&\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j := \sum_{k:I} \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\ k\ p)\, j
\end{aligned}
$$

Nodes

$$
\begin{aligned}
&\mathsf{Node} : \{i : I\}(w : \mathsf{Tr}\ i)(j : I) \to \mathsf{Op}\ j \to \mathit{Type} \\
&\mathsf{Node}\,(\mathsf{lf}\ i)\, j\, g :=
\end{aligned}
$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type}$$

$$\mathsf{Leaf}\,(\mathsf{lf}\ i)\, j := i = j$$

$$\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j := \sum_{k:I}\ \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\ k\ p)\, j$$

Nodes

$$\mathsf{Node} : \{i : I\}(w : \mathsf{Tr}\ i)(j : I) \to \mathsf{Op}\ j \to \mathit{Type}$$

$$\mathsf{Node}\,(\mathsf{lf}\ i)\, j\, g := \perp$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type}$$

$$\mathsf{Leaf}\,(\mathsf{lf}\ i)\,j := i = j$$

$$\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\,j := \sum_{k:I}\ \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\ k\ p)\,j$$

Nodes

$$\mathsf{Node} : \{i : I\}(w : \mathsf{Tr}\ i)(j : I) \to \mathsf{Op}\ j \to \mathit{Type}$$

$$\mathsf{Node}\,(\mathsf{lf}\ i)\,j\,g := \bot$$

$$\mathsf{Node}\,(\mathsf{nd}(f, \phi))\,j\,g :=$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type}$$

$$\mathsf{Leaf}\,(\mathsf{lf}\ i)\, j := i = j$$

$$\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\, j := \sum_{k:I} \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\, k\, p)\, j$$

Nodes

$$\mathsf{Node} : \{i : I\}(w : \mathsf{Tr}\ i)(j : I) \to \mathsf{Op}\ j \to \mathit{Type}$$

$$\mathsf{Node}\,(\mathsf{lf}\ i)\, j\, g := \bot$$

$$\mathsf{Node}\,(\mathsf{nd}(f, \phi))\, j\, g := (i, f) = (j, g)$$

Leaves and Nodes

For a tree $w : \mathsf{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathsf{Leaf} : \{i : I\}(w : \mathsf{Tr}\ i) \to I \to \mathit{Type}$$
$$\mathsf{Leaf}\,(\mathsf{lf}\ i)\,j := i = j$$
$$\mathsf{Leaf}\,(\mathsf{nd}(f, \phi))\,j := \sum_{k:I}\ \sum_{p:\mathsf{Param}\ f\ k} \mathsf{Leaf}\,(\phi\ k\ p)\,j$$

Nodes

$$\mathsf{Node} : \{i : I\}(w : \mathsf{Tr}\ i)(j : I) \to \mathsf{Op}\ j \to \mathit{Type}$$
$$\mathsf{Node}\,(\mathsf{lf}\ i)\,j\,g := \bot$$
$$\mathsf{Node}\,(\mathsf{nd}(f, \phi))\,j\,g := (i, f) = (j, g) \sqcup \sum_{k:I}\ \sum_{p:\mathsf{Param}\ f\ k}$$

Leaves and Nodes

For a tree $w : \mathrm{Tr}\, P\, i$, we will need its *type of leaves* and *type of nodes*.

Leaves

$$\mathrm{Leaf} : \{i : I\}(w : \mathrm{Tr}\ i) \to I \to \mathit{Type}$$
$$\mathrm{Leaf}\,(\mathrm{lf}\ i)\,j := i = j$$
$$\mathrm{Leaf}\,(\mathrm{nd}(f,\phi))\,j := \sum_{k:I}\ \sum_{p:\mathrm{Param}\ f\ k} \mathrm{Leaf}\,(\phi\ k\ p)\,j$$

Nodes

$$\mathrm{Node} : \{i : I\}(w : \mathrm{Tr}\ i)(j : I) \to \mathrm{Op}\ j \to \mathit{Type}$$
$$\mathrm{Node}\,(\mathrm{lf}\ i)\,j\,g := \bot$$
$$\mathrm{Node}\,(\mathrm{nd}(f,\phi))\,j\,g := (i,f) = (j,g) \sqcup \sum_{k:I}\ \sum_{p:\mathrm{Param}\ f\ k} \mathrm{Node}\,(\phi\ k\ p)\,j\,g$$

Frames

Definition

Let $P : \mathrm{Poly}\, I$ be a polynomial $w : \mathrm{Tr}\, P\, i$ a tree and $f : \mathrm{Op}\, P\, i$ an operation. A *frame* from w to f is a family of equivalences

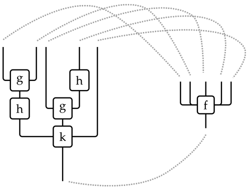
$$(j : I) \to \mathrm{Leaf}\, w\, j \simeq \mathrm{Param}\, P\, f\, j$$

Frames

Definition

Let $P : \mathrm{Poly}\, I$ be a polynomial $w : \mathrm{Tr}\, P\, i$ a tree and $f : \mathrm{Op}\, P\, i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \mathrm{Leaf}\, w\, j \simeq \mathrm{Param}\, P\, f\, j$$

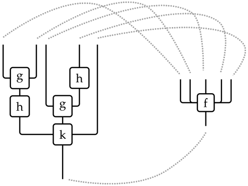


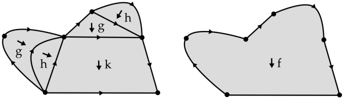
Frames

Definition

Let P : Poly I be a polynomial w : Tr $P\,i$ a tree and f : Op $P\,i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \mathrm{Leaf}\ w\ j \simeq \mathrm{Param}\ P\ f\ j$$



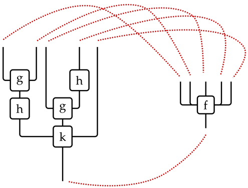


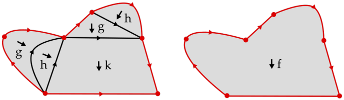
Frames

Definition

Let P : Poly I be a polynomial w : Tr $P\ i$ a tree and f : Op $P\ i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \text{Leaf}\ w\ j \simeq \text{Param}\ P\ f\ j$$



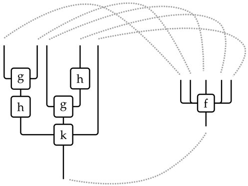


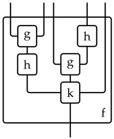
Frames

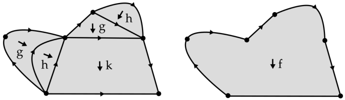
Definition

Let P : Poly I be a polynomial w : Tr $P\,i$ a tree and f : Op $P\,i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \mathrm{Leaf}\; w\; j \simeq \mathrm{Param}\; P\; f\; j$$





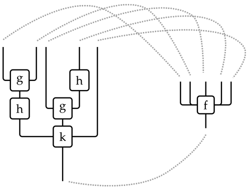


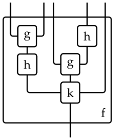
Frames

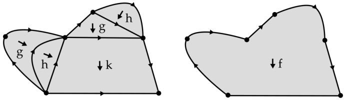
Definition

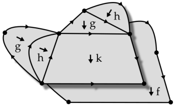
Let P : Poly I be a polynomial w : Tr $P\ i$ a tree and f : Op $P\ i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \text{Leaf}\ w\ j \simeq \text{Param}\ P\ f\ j$$







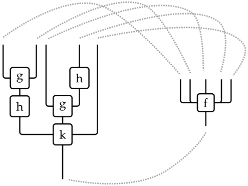


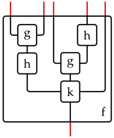
Frames

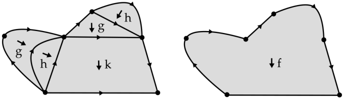
Definition

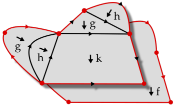
Let P : Poly I be a polynomial w : Tr $P\ i$ a tree and f : Op $P\ i$ an operation. A *frame* from w to f is a family of equivalences

$$(j : I) \to \text{Leaf}\ w\ j \simeq \text{Param}\ P\ f\ j$$









Polynomial Relations

Definition

A *polynomial relation* for P is a type family

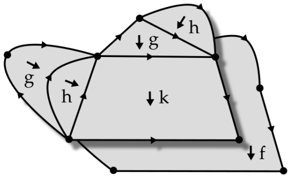
$$R : \{i : I\}(f : \mathrm{Op}\ i)(w : \mathrm{Tr}\ i)(\alpha : \mathrm{Frame}\ w\ f) \to \mathit{Type}$$

Polynomial Relations

Definition

A *polynomial relation* for P is a type family

$$R : \{i : I\}(f : \mathrm{Op}\ i)(w : \mathrm{Tr}\ i)(\alpha : \mathrm{Frame}\ w\, f) \to \mathit{Type}$$

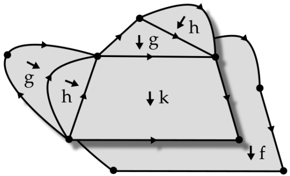


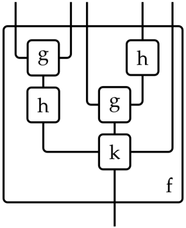
Polynomial Relations

Definition

A *polynomial relation* for P is a type family

$$R : \{i : I\}(f : \mathrm{Op}\ i)(w : \mathrm{Tr}\ i)(\alpha : \mathrm{Frame}\ w\ f) \to \mathit{Type}$$



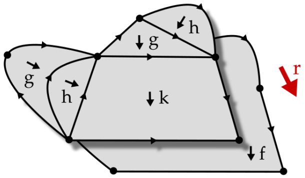


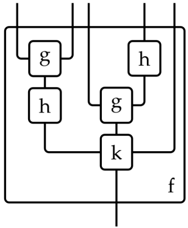
Polynomial Relations

Definition

A *polynomial relation* for P is a type family

$$R : \{i : I\}(f : \mathrm{Op}\ i)(w : \mathrm{Tr}\ i)(\alpha : \mathrm{Frame}\ w\ f) \to \mathit{Type}$$



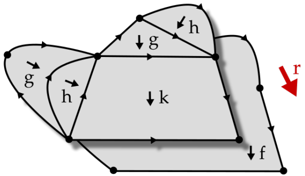


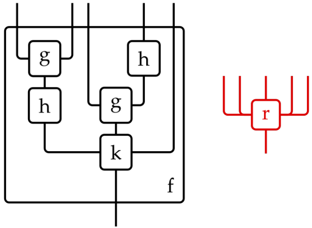
Polynomial Relations

Definition

A *polynomial relation* for P is a type family

$$R : \{i : I\}(f : \mathrm{Op}\ i)(w : \mathrm{Tr}\ i)(\alpha : \mathrm{Frame}\ w\, f) \to \mathit{Type}$$





The Slice of a Polynomial by a Relation

Definition

Let $P : \mathrm{Poly}\, I$ and let R be a relation on P. The *slice of P by R*, denoted $P//R$, is the polynomial with sorts $\Sigma\, I\, \mathrm{Op}$ defined as follows:

$$\mathrm{Op}(P//M)\,(i, f) := \sum_{(w : \mathrm{Tr}\, P\, i)} \sum_{(\alpha : \mathrm{Frame}\, w\, f)} R\, f\, w\, \alpha$$

$$\mathrm{Param}(P//M)(w, \alpha, r)(j, g) := \mathrm{Node}\, w\, g$$

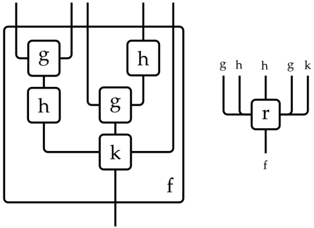
The Slice of a Polynomial by a Relation

Definition

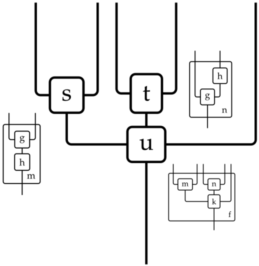
Let $P : \mathrm{Poly}\ I$ and let R be a relation on P. The *slice of P by R*, denoted $P//R$, is the polynomial with sorts $\Sigma\ I\ \mathrm{Op}$ defined as follows:

$$\mathrm{Op}(P//M)\,(i,f) := \sum_{(w:\mathrm{Tr}\,P\,i)} \sum_{(\alpha:\mathrm{Frame}\,w\,f)} R\,f\,w\,\alpha$$

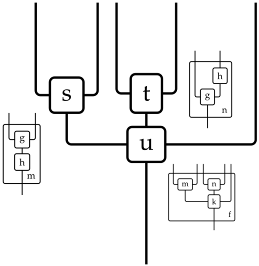
$$\mathrm{Param}(P//M)(w,\alpha,r)(j,g) := \mathrm{Node}\,w\,g$$

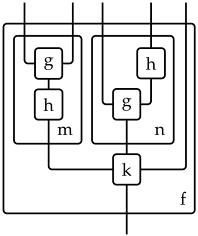


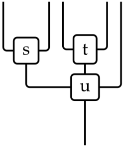
Trees in the Slice Polynomial



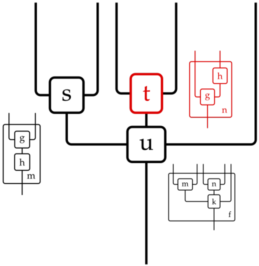
Trees in the Slice Polynomial

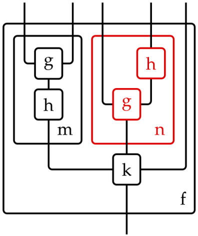


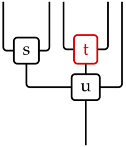




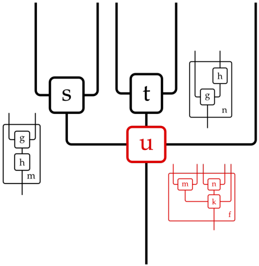
Trees in the Slice Polynomial

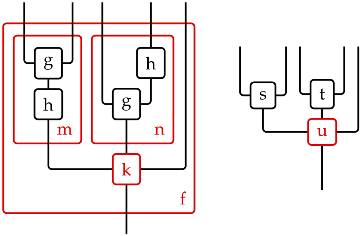




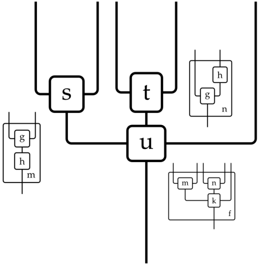


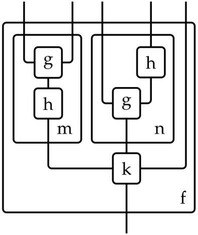
Trees in the Slice Polynomial

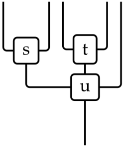




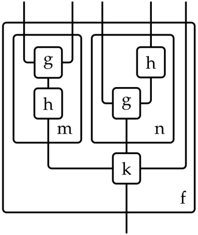
Trees in the Slice Polynomial

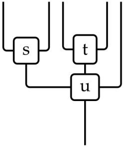




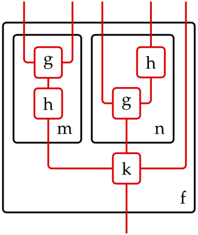


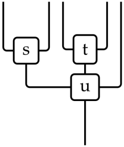
Flattening



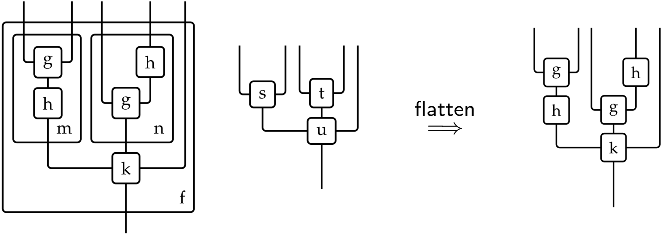


Flattening



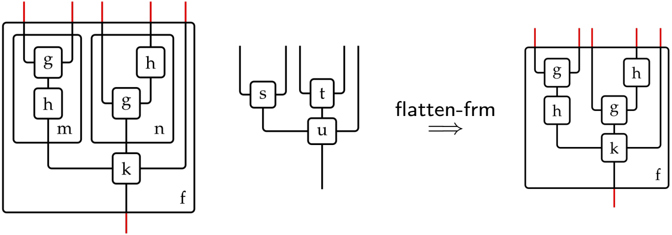


Flattening



$$\mathsf{flatten} : \{i : I\}\{f : \mathsf{Op}\, i\} \to \mathsf{Tr}(P//R)\,(i, f) \to \mathsf{Tr}\, P\, i$$

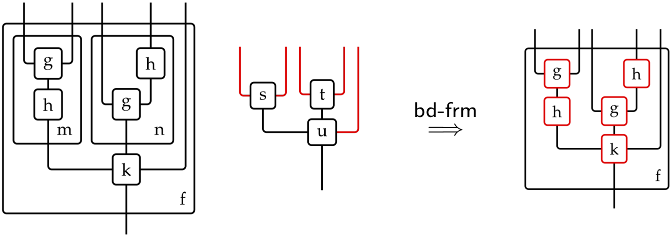
Flattening



$$\mathsf{flatten} : \{i : I\}\{f : \mathsf{Op}\, i\} \to \mathsf{Tr}(P//R)\,(i, f) \to \mathsf{Tr}\, P\, i$$

$$\mathsf{flatten\text{-}frm} : \{i : I\}\{f : \mathsf{Op}\, i\}(pd : \mathsf{Tr}(P//R)\,(i, f))$$
$$\to \mathsf{Frame}(\mathsf{flatten}\, pd)\, f$$

Flattening



flatten : $\{i : I\}\{f : \mathsf{Op}\, i\} \to \mathsf{Tr}(P//R)\,(i, f) \to \mathsf{Tr}\, P\, i$

flatten-frm : $\{i : I\}\{f : \mathsf{Op}\, i\}(pd : \mathsf{Tr}(P//R)\,(i, f))$
$\quad \to \mathsf{Frame}(\mathsf{flatten}\, pd)\, f$

bd-frm : $\{i : I\}\{f : \mathsf{Op}\, i\}(pd : \mathsf{Tr}(P//R)\,(i, f))$
$\quad \to (j : I)(g : \mathsf{Op}\, j) \to \mathsf{Leaf}(P//R)\, pd\, g \simeq \mathsf{Node}\, P\, (\mathsf{flatten}\, pd) g$

Polynomial Magmas

Polynomials serve as our notion of higher signature. Following ideas from the categorical approach to universal algebra, we are going to encode the *relations* or *axioms* of our structure using a *monadic multiplication* on P.

Polynomial Magmas

Polynomials serve as our notion of higher signature. Following ideas from the categorical approach to universal algebra, we are going to encode the *relations* or *axioms* of our structure using a *monadic multiplication* on P.

Definition

Let P be a polynomial with sorts in I. A *polynomial magma* M over P is

1. A function $\mu : \{i : I\} \to \mathsf{Tr}\, P\, i \to \mathsf{Op}\, P\, i$
2. A function $\mu_{frm} : \{i : I\}(w : \mathsf{Tr}\, P\, i) \to \mathsf{Frame}\, w\, (\mu\, w)$

Polynomial Magmas

Polynomials serve as our notion of higher signature. Following ideas from the categorical approach to universal algebra, we are going to encode the *relations* or *axioms* of our structure using a *monadic multiplication* on P.

Definition

Let P be a polynomial with sorts in I. A *polynomial magma* M over P is

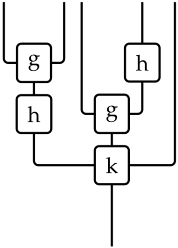
1. A function $\mu : \{i : I\} \to \mathrm{Tr}\, P\, i \to \mathrm{Op}\, P\, i$
2. A function $\mu_{frm} : \{i : I\}(w : \mathrm{Tr}\, P\, i) \to \mathrm{Frame}\, w\, (\mu\, w)$

Notice that a magma M determines a polynomial relation on P by using the identity type:

$$\mathrm{MgmRel} : \mathrm{PolyMagma}\, P \to \mathrm{PolyRel}\, P$$
$$\mathrm{MgmRel}\, M\, f\, w\, \alpha := (\mu\, w, \mu_{frm}\, w) = (f, \alpha)$$

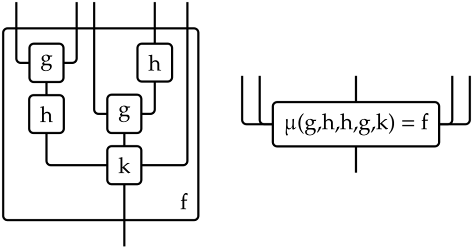
Polynomial Magmas (cont'd)

Using the graphical notation we have developed, we can "picture" the multiplication μ as follows:



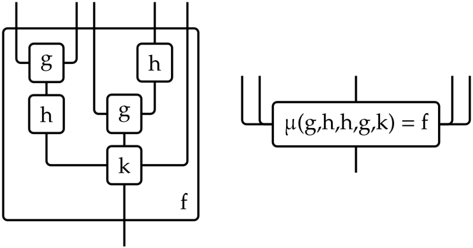
Polynomial Magmas (cont'd)

Using the graphical notation we have developed, we can "picture" the multiplication μ as follows:



Polynomial Magmas (cont'd)

Using the graphical notation we have developed, we can "picture" the multiplication μ as follows:

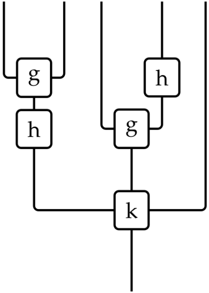


In algebraic notation, this corresponds to the relation

$$k(h(g(x,y)), g(u,h(v)), w) = f(x,y,u,v,w)$$

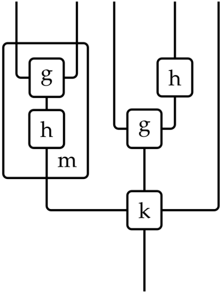
Coherent Relations

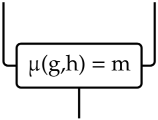
Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:



Coherent Relations

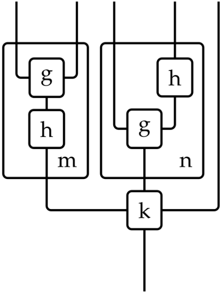
Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:

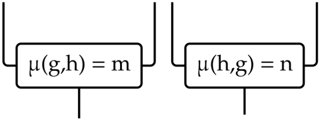




Coherent Relations

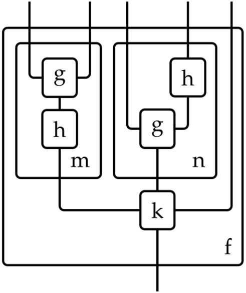
Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:

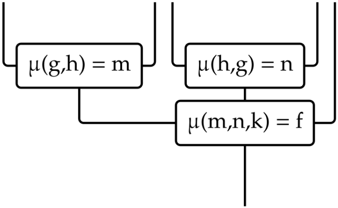




Coherent Relations

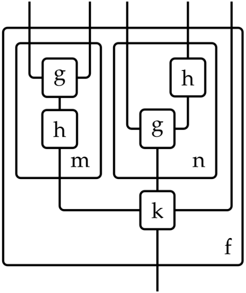
Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:

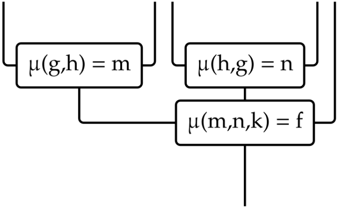




Coherent Relations

Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:



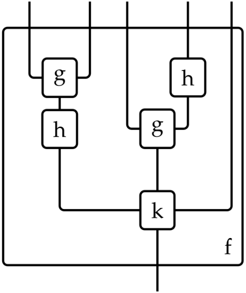


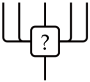
But: without further structure, there is simply no reason that this sequence of multiplications gives rise to the "obvious" relation

$$\mu(g, h, h, g, k) = f$$

Coherent Relations

Furthermore, we can now interpret a pasting diagram $pd : \mathrm{Tr}(P//M)\,(i, f)$ as a sequence of multiplications applied to subterms of flatten pd:





But: without further structure, there is simply no reason that this sequence of multiplications gives rise to the "obvious" relation

$$\mu(g, h, h, g, k) = f$$

Subdivision Invariance

Definition

Let P be a polynomial and R a relation on P. We say that R is *subdivision invariant* if we are given a function.

$$\Psi : \{i : I\}\{f : \mathrm{Op}\, P\, i\}(pd : \mathrm{Tr}(P//R)\,(i, f)) \\ \to R\, f\,(\mathrm{flatten}\, pd)\,(\text{flatten-frm}\, pd)$$

Subdivision Invariance

Definition

Let P be a polynomial and R a relation on P. We say that R is *subdivision invariant* if we are given a function.

$$\begin{aligned}\Psi :& \{i : I\}\{f : \mathrm{Op}\, P\, i\}(pd : \mathrm{Tr}(P//R)\,(i,f)) \\ &\to R\, f\,(\mathrm{flatten}\, pd)\,(\text{flatten-frm}\, pd)\end{aligned}$$

We write SubInvar for the associated predicate on polynomial relations.

$$\begin{aligned}\mathrm{SubInvar} &: \mathrm{PolyRel}\, P \to \mathit{Type} \\ \mathrm{SubInvar}\ R &:= \{i : I\}\{f : \mathrm{Op}\, P\, i\}(pd : \mathrm{Tr}(P//R)\,(i,f)) \\ &\quad \to R\, f\,(\mathrm{flatten}\, pd)\,(\text{flatten-frm}\, pd)\end{aligned}$$

The Slice Magma

Observation

Let P be a polynomial and R a relation on P. Given a witness Ψ that R is subdivision invariant, the slice polynomial $P//R$ admits a magma structure given by

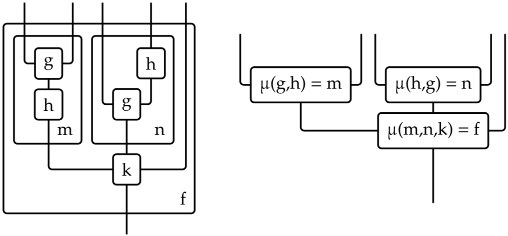
$$\mu(\mathsf{SlcMgm}\, R)\, pd := ((\mathsf{flatten}\, pd, \mathsf{flatten\text{-}frm}\, pd), \Psi\, pd)$$
$$\mu_{frm}(\mathsf{SlcMgm}\, R)\, pd := \mathsf{bd\text{-}frm}\, pd$$

The Slice Magma

Observation

Let P be a polynomial and R a relation on P. Given a witness Ψ that R is subdivision invariant, the slice polynomial $P//R$ admits a magma structure given by

$$\mu(\mathsf{SlcMgm}\ R)\ pd := ((\mathsf{flatten}\ pd, \mathsf{flatten\text{-}frm}\ pd), \Psi\ pd)$$
$$\mu_{frm}(\mathsf{SlcMgm}\ R)\ pd := \mathsf{bd\text{-}frm}\ pd$$

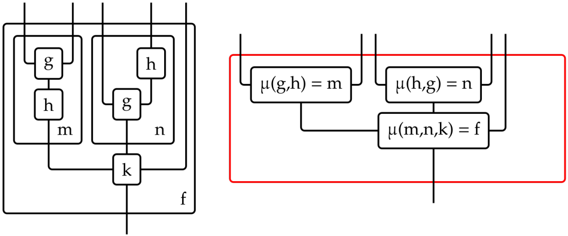


The Slice Magma

Observation

Let P be a polynomial and R a relation on P. Given a witness Ψ that R is subdivision invariant, the slice polynomial $P//R$ admits a magma structure given by

$$\mu(\mathsf{SlcMgm}\ R)\ pd := ((\mathsf{flatten}\ pd, \mathsf{flatten\text{-}frm}\ pd), \Psi\ pd)$$
$$\mu_{frm}(\mathsf{SlcMgm}\ R)\ pd := \mathsf{bd\text{-}frm}\ pd$$

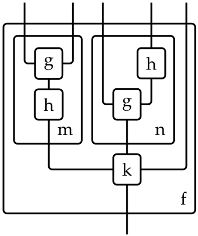


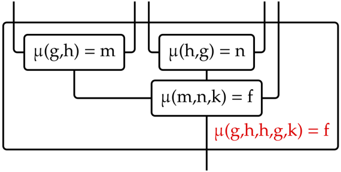
The Slice Magma

Observation

Let P be a polynomial and R a relation on P. Given a witness Ψ that R is subdivision invariant, the slice polynomial $P//R$ admits a magma structure given by

$$\mu(\mathsf{SlcMgm}\,R)\,pd := ((\mathsf{flatten}\,pd, \mathsf{flatten\text{-}frm}\,pd), \Psi\,pd)$$
$$\mu_{frm}(\mathsf{SlcMgm}\,R)\,pd := \mathsf{bd\text{-}frm}\,pd$$



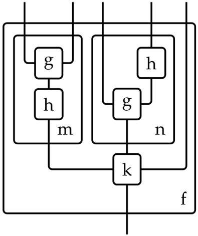


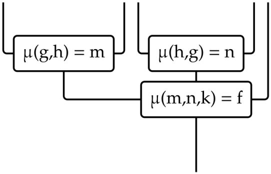
Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.

Example: Associativity

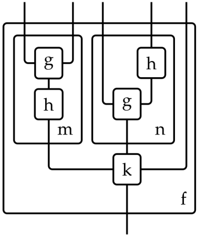
Let us see why, if a magma is subdivision invariant, then it is associative.

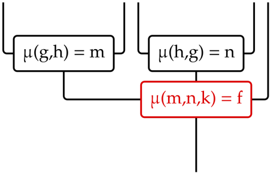




Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.

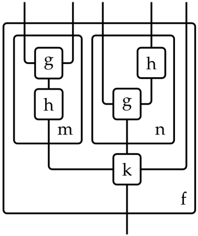


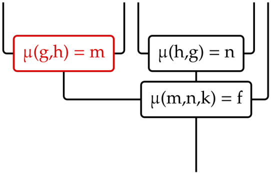


$$\mu(m, n, k) = f$$

Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.

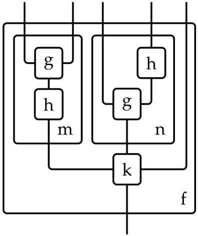


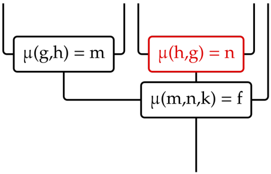


$$\mu(\mu(g,h),n,k) = f$$

Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.

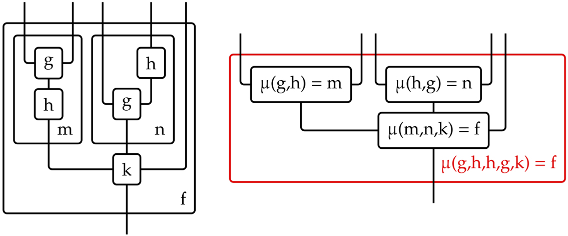




$$\mu(\mu(g,h),\mu(h,g),k) = f$$

Example: Associativity

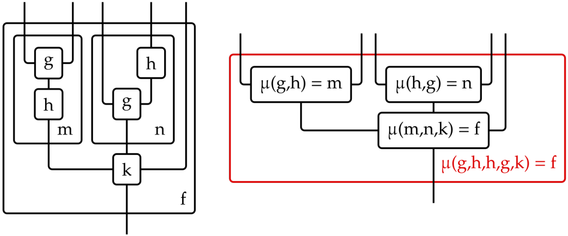
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = f$$

Example: Associativity

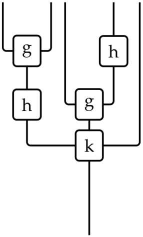
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k)=\mu(g,h,h,g,k)$$

Example: Associativity

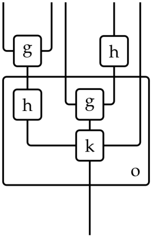
Let us see why, if a magma is subdivision invariant, then it is associative.

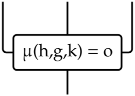


$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$

Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.

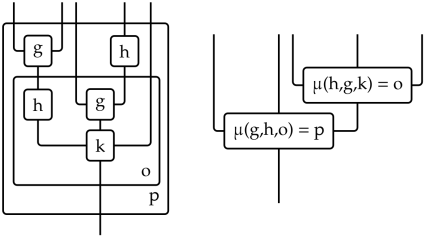




$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$

Example: Associativity

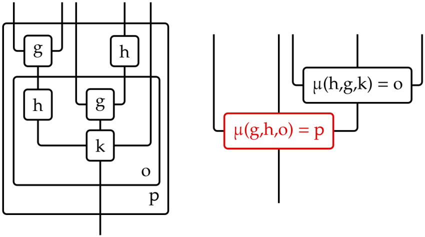
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$

Example: Associativity

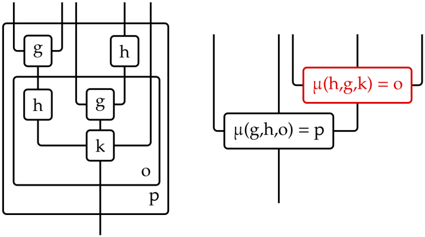
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$
$$\mu(g,h,o) = p$$

Example: Associativity

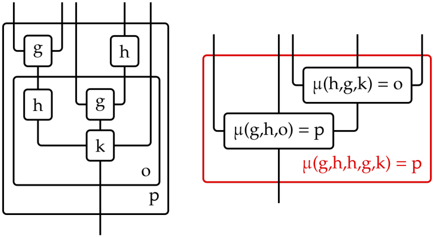
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$
$$\mu(g,h,\mu(h,g,k)) = p$$

Example: Associativity

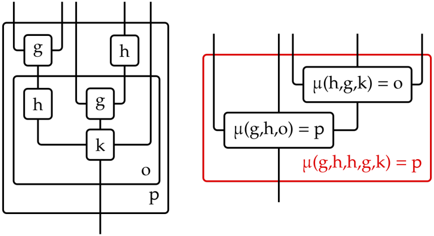
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$
$$\mu(g,h,\mu(h,g,k)) = p$$

Example: Associativity

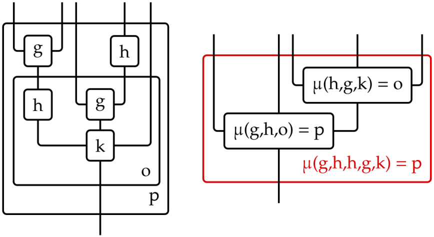
Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$
$$\mu(g,h,\mu(h,g,k)) = \mu(g,h,h,g,k)$$

Example: Associativity

Let us see why, if a magma is subdivision invariant, then it is associative.



$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,h,g,k)$$
$$\mu(g,h,\mu(h,g,k)) = \mu(g,h,h,g,k)$$

Hence

$$\mu(\mu(g,h),\mu(h,g),k) = \mu(g,h,\mu(h,g,k))$$

Polynomial Monads

Let P be a polynomial and M a magma on P.

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof $\Psi : \mathsf{SubInvar}\ M$

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof $\Psi : \mathsf{SubInvar}\ M$
2. Coninductively, a coherence structure on $\mathsf{SlcMgm}\ M\,\Psi$

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof Ψ : SubInvar M
2. Coninductively, a coherence structure on SlcMgm $M\,\Psi$

Definition

A *polynomial monad* consists of

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof Ψ : SubInvar M
2. Coinductively, a coherence structure on SlcMgm $M\,\Psi$

Definition

A *polynomial monad* consists of

1. A polynomial P : Poly I

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof Ψ : SubInvar M
2. Coinductively, a coherence structure on SlcMgm $M\,\Psi$

Definition

A *polynomial monad* consists of

1. A polynomial P : Poly I
2. A magma M : PolyMagma P

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof Ψ : SubInvar M
2. Coinductively, a coherence structure on SlcMgm $M\,\Psi$

Definition

A *polynomial monad* consists of

1. A polynomial P : Poly I
2. A magma M : PolyMagma P
3. A coherence structure C for M

Polynomial Monads

Let P be a polynomial and M a magma on P.

Definition

A *coherence structure* for M consists of

1. A proof Ψ : SubInvar M
2. Conindunctively, a coherence structure on SlcMgm $M\,\Psi$

Definition

A *polynomial monad* consists of

1. A polynomial P : Poly I
2. A magma M : PolyMagma P
3. A coherence structure C for M
4. A proof that M is *univalent*

Univalence for Monads

- For an operation $f : \mathrm{Op}\, i$ we define

Univalence for Monads

- For an operation $f : \mathsf{Op}\, i$ we define

$$\mathsf{Arity}\, f := \sum_{j:I} \mathsf{Param}\, f\, j$$

Univalence for Monads

- For an operation $f : \mathsf{Op}\, i$ we define

$$\begin{aligned} \mathsf{Arity}\, f &:= \sum_{j:I} \mathsf{Param}\, f\, j \\ \text{is-unary}\, f &:= \text{is-contr}(\mathsf{Arity}\, f) \end{aligned}$$

Univalence for Monads

- For an operation $f : \mathsf{Op}\, i$ we define

$$\mathsf{Arity}\, f := \sum_{j:I} \mathsf{Param}\, f\, j \qquad \mathsf{UnaryOp}\, M := \sum_{i:I} \sum_{f:\mathsf{Op}\, i} \text{is-unary}\, f$$

$$\text{is-unary}\, f := \text{is-contr}(\mathsf{Arity}\, f)$$

Univalence for Monads

- For an operation $f : \mathsf{Op}\ i$ we define

$$\mathsf{Arity}\ f := \sum_{j:I} \mathsf{Param}\ f\ j \qquad \mathsf{UnaryOp}\ M := \sum_{i:I} \sum_{f:\mathsf{Op}\ i} \text{is-unary}\ f$$

$$\text{is-unary}\ f := \text{is-contr}(\mathsf{Arity}\ f) \qquad \mathsf{id}\ i := \mu(\mathsf{lf}\ i)$$

Univalence for Monads

- For an operation $f : \mathsf{Op}\, i$ we define

$$\mathsf{Arity}\, f := \sum_{j:I} \mathsf{Param}\, f\, j \qquad \mathsf{UnaryOp}\, M := \sum_{i:I} \sum_{f:\mathsf{Op}\, i} \mathsf{is\text{-}unary}\, f$$

$$\mathsf{is\text{-}unary}\, f := \mathsf{is\text{-}contr}(\mathsf{Arity}\, f) \qquad \mathsf{id}\, i := \mu(\mathsf{lf}\, i)$$

- One can easily check (using μ_{frm}) that $\mathsf{id}\, i$ is unary.

Univalence for Monads

- For an operation $f : \mathsf{Op}\, i$ we define

$$\mathsf{Arity}\, f := \sum_{j:I} \mathsf{Param}\, f\, j \qquad \mathsf{UnaryOp}\, M := \sum_{i:I} \sum_{f:\mathsf{Op}\, i} \text{is-unary}\, f$$

$$\text{is-unary}\, f := \text{is-contr}(\mathsf{Arity}\, f) \qquad \mathsf{id}\, i := \mu(\mathsf{lf}\, i)$$

- One can easily check (using μ_{frm}) that $\mathsf{id}\, i$ is unary.
- We can think of a unary operation $f : \mathsf{Op}\, i$ as a "morphism"

$$f : j \to i$$

where j is the sort of its unique parameter.

Univalence for Monads

- For an operation $f : \mathsf{Op}\ i$ we define

$$\mathsf{Arity}\ f := \sum_{j:I} \mathsf{Param}\ f\ j \qquad \mathsf{UnaryOp}\ M := \sum_{i:I} \sum_{f:\mathsf{Op}\ i} \text{is-unary}\ f$$

$$\text{is-unary}\ f := \text{is-contr}(\mathsf{Arity}\ f) \qquad \mathsf{id}\ i := \mu(\mathsf{lf}\ i)$$

- One can easily check (using μ_{frm}) that $\mathsf{id}\ i$ is unary.
- We can think of a unary operation $f : \mathsf{Op}\ i$ as a "morphism"

$$f : j \to i$$

  where j is the sort of its unique parameter.
- The multiplication μ can now be used to define a composition operation

$$\_ \circ \_ : \mathsf{UnaryOp} \times \mathsf{UnaryOp} \to \mathsf{UnaryOp}$$

Univalence for Monads (cont'd)

Definition

Let M be a polynomial monad. A unary operation $f : j \to i$ is said to be an *isomorphism* if satisfies the bi-inverse property:

$$\text{is-iso}\, f := \sum_{g : i \to j} \sum_{h : i \to j} (f \circ g = \text{id}\, i) \times (h \circ f = \text{id}\, j)$$

Write $\text{Iso}\, M$ for the space of isomorphisms in M.

Univalence for Monads (cont'd)

Definition

Let M be a polynomial monad. A unary operation $f : j \to i$ is said to be an *isomorphism* if satisfies the bi-inverse property:

$$\text{is-iso}\, f := \sum_{g : i \to j} \sum_{h : i \to j} (f \circ g = \text{id}\, i) \times (h \circ f = \text{id}\, j)$$

Write $\text{Iso}\, M$ for the space of isomorphisms in M.

It is routine to check that for $i : I$, the operation $\text{id}\, i$ is an isomorphism in this sense. Hence we have

$$\begin{aligned} &\text{id-to-iso} : \{ij : I\} \to i = j \to \text{Iso}\, M \\ &\text{id-to-iso}\{i\}\, \text{idp} = \text{id}\, i \end{aligned}$$

Univalence for Monads (cont'd)

Definition

Let M be a polynomial monad. A unary operation $f : j \to i$ is said to be an *isomorphism* if satisfies the bi-inverse property:

$$\text{is-iso}\, f := \sum_{g:i \to j} \sum_{h:i \to j} (f \circ g = \text{id}\, i) \times (h \circ f = \text{id}\, j)$$

Write $\text{Iso}\, M$ for the space of isomorphisms in M.

It is routine to check that for $i : I$, the operation $\text{id}\, i$ is an isomorphism in this sense. Hence we have

$$\begin{aligned} &\text{id-to-iso} : \{ij : I\} \to i = j \to \text{Iso}\, M \\ &\text{id-to-iso}\{i\}\, \text{idp} = \text{id}\, i \end{aligned}$$

Definition

M is said to be *univalent* if the above map is an equivalence.

Special Cases of Monads

- For a type $X : \mathit{Type}$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \text{Fin}\ n\|_{-1}$$

Special Cases of Monads

- For a type $X : \mathit{Type}$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \| X \simeq \mathsf{Fin}\; n \|_{-1}$$

- Let M be a polynomial monad. We define

Special Cases of Monads

- For a type $X : Type$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \text{Fin}\, n\|_{-1}$$

- Let M be a polynomial monad. We define

$$\text{is-}\infty\text{-operad}\, M := \{i : I\}(f : \text{Op}\, i) \to \text{is-finite}(\text{Arity}\, f)$$

Special Cases of Monads

- For a type $X : \mathit{Type}$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \mathsf{Fin}\ n\|_{-1}$$

- Let M be a polynomial monad. We define

$$\begin{aligned}\text{is-}\infty\text{-operad}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-finite}(\mathsf{Arity}\, f) \\ \text{is-}\infty\text{-category}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-unary}\, f\end{aligned}$$

Special Cases of Monads

- For a type $X : Type$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \mathrm{Fin}\, n\|_{-1}$$

- Let M be a polynomial monad. We define

$$\begin{aligned}
\text{is-}\infty\text{-operad}\, M &:= \{i : I\}(f : \mathrm{Op}\, i) \to \text{is-finite}(\mathrm{Arity}\, f) \\
\text{is-}\infty\text{-category}\, M &:= \{i : I\}(f : \mathrm{Op}\, i) \to \text{is-unary}\, f \\
\text{is-}\infty\text{-groupoid}\, M &:= \text{is-}\infty\text{-category}\, M \times (f : \mathrm{Op}\, i) \to \text{is-iso}\, f
\end{aligned}$$

Special Cases of Monads

- For a type $X : \mathit{Type}$ let
$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \| X \simeq \text{Fin}\; n \|_{-1}$$
- Let M be a polynomial monad. We define
$$\begin{aligned} \text{is-}\infty\text{-operad}\, M &:= \{i : I\}(f : \text{Op}\; i) \to \text{is-finite}(\text{Arity}\; f) \\ \text{is-}\infty\text{-category}\, M &:= \{i : I\}(f : \text{Op}\; i) \to \text{is-unary}\; f \\ \text{is-}\infty\text{-groupoid}\, M &:= \text{is-}\infty\text{-category}\, M \times (f : \text{Op}\; i) \to \text{is-iso}\; f \end{aligned}$$
- More special cases are possible:

Special Cases of Monads

- For a type $X : \mathit{Type}$ let

$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \mathsf{Fin}\; n\|_{-1}$$

- Let M be a polynomial monad. We define

$$\begin{aligned} \text{is-}\infty\text{-operad}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-finite}(\mathsf{Arity}\, f) \\ \text{is-}\infty\text{-category}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-unary}\, f \\ \text{is-}\infty\text{-groupoid}\, M &:= \text{is-}\infty\text{-category}\, M \times (f : \mathsf{Op}\, i) \to \text{is-iso}\, f \end{aligned}$$

- More special cases are possible:
  - A *symmetric monoidal ∞-category* is an ∞-operad with enough "universal" operations.

Special Cases of Monads

- For a type $X : \mathit{Type}$ let
$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \| X \simeq \mathsf{Fin}\; n \|_{-1}$$
- Let M be a polynomial monad. We define
$$\begin{aligned} \text{is-}\infty\text{-operad}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-finite}(\mathsf{Arity}\, f) \\ \text{is-}\infty\text{-category}\, M &:= \{i : I\}(f : \mathsf{Op}\, i) \to \text{is-unary}\, f \\ \text{is-}\infty\text{-groupoid}\, M &:= \text{is-}\infty\text{-category}\, M \times (f : \mathsf{Op}\, i) \to \text{is-iso}\, f \end{aligned}$$
- More special cases are possible:
  - A *symmetric monoidal ∞-category* is an ∞-operad with enough "universal" operations.
  - An A_∞-type is an ∞-category for which the type I is *connected*

Special Cases of Monads

- For a type $X : Type$ let
$$\text{is-finite}\, X := \sum_{n:\mathbb{N}} \|X \simeq \text{Fin}\ n\|_{-1}$$
- Let M be a polynomial monad. We define
$$\text{is-}\infty\text{-operad}\, M := \{i : I\}(f : \text{Op}\, i) \to \text{is-finite}(\text{Arity}\, f)$$
$$\text{is-}\infty\text{-category}\, M := \{i : I\}(f : \text{Op}\, i) \to \text{is-unary}\, f$$
$$\text{is-}\infty\text{-groupoid}\, M := \text{is-}\infty\text{-category}\, M \times (f : \text{Op}\, i) \to \text{is-iso}\, f$$
- More special cases are possible:
  - A *symmetric monoidal ∞-category* is an ∞-operad with enough "universal" operations.
  - An A_∞-type is an ∞-category for which the type I is *connected*
  - etc ...

Future Directions

- Finish the definition of simplicial type

Future Directions

- Finish the definition of simplicial type
- Conjecture:

$$\infty\text{-groupoid} \simeq \mathit{Type}$$

Future Directions

- Finish the definition of simplicial type
- Conjecture:

$$\infty\text{-groupoid} \simeq \mathit{Type}$$

- Loop spaces are grouplike A_∞-types?

Future Directions

- Finish the definition of simplicial type
- Conjecture:

$$\infty\text{-groupoid} \simeq \mathit{Type}$$

- Loop spaces are grouplike A_∞-types?
- Initial algebras and HIT's

Future Directions

- Finish the definition of simplicial type
- Conjecture:

$$\infty\text{-groupoid} \simeq \mathit{Type}$$

- Loop spaces are grouplike A_∞-types?
- Initial algebras and HIT's
- Develop higher category theory

Future Directions

- Finish the definition of simplicial type
- Conjecture:

$$\infty\text{-groupoid} \simeq \mathit{Type}$$

- Loop spaces are grouplike A_∞-types?
- Initial algebras and HIT's
- Develop higher category theory

Thanks!