Higher Universal Algebra in Type Theory

Eric Finster

Joint with M. Sozeau, Y. Guiraud, A. Allioux

November 8, 2018
Univalent Type Theory

- Interpret types as *homotopy types*
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:

$$\text{Id}_U(X, Y) \simeq \text{Equiv}(X, Y)$$
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:
  \[
  \text{Id}_U(X, Y) \overset{\sim}{\longrightarrow} \text{Equiv}(X, Y)
  \]
- Synthetic homotopy theory:
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:
  \[
  \text{Id}_U(X, Y) \simeq \text{Equiv}(X, Y)
  \]
- Synthetic homotopy theory:
  - Eilenberg-MacLane spaces
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:
  \[
  \text{Id}_U(X, Y) \overset{\simeq}{\longrightarrow} \text{Equiv}(X, Y)
  \]

- Synthetic homotopy theory:
  - Eilenberg-MacLane spaces
  - Blakers-Massey Theorem
Univalent Type Theory

- Interpret types as homotopy types
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:
  $$\text{Id}_U(X, Y) \overset{\simeq}{\longrightarrow} \text{Equiv}(X, Y)$$

- Synthetic homotopy theory:
  - Eilenberg-MacLane spaces
  - Blakers-Massey Theorem
  - Calculations of Homotopy Groups
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:

  $\text{Id}_U(X, Y) \xrightarrow{\sim} \text{Equiv}(X, Y)$

- Synthetic homotopy theory:
  - Eilenberg-MacLane spaces
  - Blakers-Massey Theorem
  - Calculations of Homotopy Groups
  - Steenrod Operations
Univalent Type Theory

- Interpret types as *homotopy types*
- Interpret elements of $\text{Id}_X$ as paths, paths of paths, etc.
- Univalence axiom:
  \[ \text{Id}_U(X, Y) \overset{\simeq}{\longrightarrow} \text{Equiv}(X, Y) \]

- Synthetic homotopy theory:
  - Eilenberg-MacLane spaces
  - Blakers-Massey Theorem
  - Calculations of Homotopy Groups
  - Steenrod Operations
  - Serre Spectral Sequence
H-Level and the Stratification of Types

Definition of contractibility

Definition of h-level

Remark: There are types which are not of any finite h-level, i.e., this filtration is not exhaustive
H-Level and the Stratification of Types

Definition of contractibility

\[ \text{is-contr} := \sum_{x: X} \prod_{y: X} (x = x \ y) \]

Remark: \((\text{is-contr}_X) \iff (X \simeq \top)\)

Definition of h-level

\[ \text{is-of-level} (-2)_X := \text{is-contr}_X \]
\[ \text{is-of-level} (S \ n)_X := \prod_{x, y: X} \text{is-of-level} (S \ n)(x = x \ y) \]

Remark: There are types which are not of any finite h-level, i.e., this filtration is not exhaustive.
H-Level and the Stratification of Types

Definition of contractibility

\[ \text{is-contr} := \sum_{x:X} \prod_{y:X} (x =_{X} y) \]

Remark: \((\text{is-contr } X) \iff (X \simeq \top)\)

Definition of h-level

\[ \text{is-of-level } (-2) X := \text{is-contr } X \]
\[ \text{is-of-level } (S n) X := \prod_{x,y:X} \text{is-of-level } n (x =_{X} y) \]

Remark: There are types which are not of any finite h-level, i.e., this filtration is not exhaustive.
H-Level and the Stratification of Types

- Definition of contractibility

\[
is\text{-contr} := \sum_{x:X} \prod_{y:X} (x =_X y)
\]

- Remark: \((\text{is-contr } X) \iff (X \simeq \top)\)

- Definition of h-level
H-Level and the Stratification of Types

- Definition of contractibility

\[
is-\text{contr} := \sum_{x:X} \prod_{y:X} (x =_X y)
\]

- Remark: \((is-\text{contr } X) \iff (X \simeq \top)\)

- Definition of h-level

\[
is-\text{of-level } (-2) X := is-\text{contr } X
\]
\[
is-\text{of-level } (S \, n) X := \prod_{x,y:X} is-\text{of-level } n (x =_X y)
\]

- Remark: There are types which are not of any finite h-level, i.e., this filtration is not exhaustive.
H-Level and the Stratification of Types

- Definition of contractibility
  \[
  \text{is-contr} := \sum_{x:X} \prod_{y:X} (x = x \ y)
  \]

- Remark: \((\text{is-contr} \ X) \iff (X \simeq \top)\)

- Definition of h-level
  \[
  \text{is-of-level} (-2) \ X := \text{is-contr} \ X
  \]
  \[
  \text{is-of-level} (S \ n) \ X := \prod_{x,y:X} \text{is-of-level} \ n (x = x \ y)
  \]

- Remark: There are types which are not of any finite \(h\)-level, i.e., this filtration is not exhaustive
Voevodsky’s Vision

- The Mathematics of Cantor:
  - Sets and structured sets ($h$-level 0)
Voevodsky’s Vision

- The Mathematics of Cantor:
  - Sets and structured sets ($h$-level 0)
- Then Mathematics of Grothendieck
  - The mathematics of categories and structured categories ($h$-level 1)
Voevodsky’s Vision

- The Mathematics of Cantor:
  - Sets and structured sets ($h$-level 0)
- Then Mathematics of Grothendieck
  - The mathematics of categories and structured categories ($h$-level 1)
- 21st Century Mathematics
  - The mathematics of structures on types of higher $h$-level
Definition
A category consists of the data ...
Category Theory for Types?

Definition
A category consists of the data ...

1. Objects:
   
   \[ Ob : Type \]
Definition
A category consists of the data ...

1. Objects:
   \[ \text{Ob} : \text{Type} \]

2. Morphisms:
   \[ \text{Hom} : \text{Ob} \rightarrow \text{Ob} \rightarrow \text{Type} \]

Category Theory for Types?
Definition
A category consists of the data ...

1. Objects:
   \( Ob : Type \)

2. Morphisms:
   \( Hom : Ob \to Ob \to Type \)

3. Identity:
   \( id : (x : Ob) \to Hom x x \)
Category Theory for Types?

Definition
A category consists of the data ... 
1. Objects: 
   \( Ob : Type \)

2. Morphisms: 
   \( Hom : Ob \rightarrow Ob \rightarrow Type \)

3. Identity: 
   \( id : (x : Ob) \rightarrow Hom x x \)

4. Composition: 
   \( \circ : (x, y, z : Ob)(f : Hom y z)(g : Hom x y) \rightarrow Hom x z \)
Laws for Categories

Definition (Cont’d)

... satisfying the laws
Laws for Categories

Definition (Cont’d)

... satisfying the laws

5. Unit Laws:

\[
\text{unit-l : } (xy : Ob)(f : Hom x y) \rightarrow f = id_x \circ f
\]

\[
\text{unit-r : } (xy : Ob)(f : Hom x y) \rightarrow id_y \circ f = f
\]
Laws for Categories

Definition (Cont’d)

... satisfying the laws

5. Unit Laws:

\[\text{unit-l : (xy : Ob)(f : Hom x y) \rightarrow f = id_x \circ f}\]

\[\text{unit-r : (xy : Ob)(f : Hom x y) \rightarrow id_y \circ f = f}\]

6. Associative Law:

\[\text{assoc : (xyzw : Ob)(f : Hom z w)(g : Hom y z)(h : Hom x y) \rightarrow ((f \circ g) \circ h) = (f \circ (g \circ h))}\]
Slice Category?

For $Z \in C$, we would like to define the slice category $C/Z$. 
Slice Category?

For $Z \in C$, we would like to define the slice category $C/Z$.

1. Objects:

   $f : X \to Z$
Slice Category?

For \( Z \in C \), we would like to define the slice category \( C/Z \).

1. Objects:

\[ f : X \to Z \]

2. Morphisms:

\[
\begin{array}{ccc}
X & \xrightarrow{h} & Y \\
\downarrow{f} & & \downarrow{g} \\
Z & & \\
\end{array}
\]
Slice Category?

For $Z \in C$, we would like to define the slice category $C/Z$.

1. Objects:

   $$f : X \rightarrow Z$$

2. Morphisms:

   $X \xrightarrow{h} Y$
   $\downarrow{f} \downarrow{g}$
   $Z$

But we immediately run into a problem:
Slice Category?

For $Z \in C$, we would like to define the slice category $C/Z$.

1. Objects:

$$f : X \rightarrow Z$$

2. Morphisms:

$$\begin{align*}
X & \xrightarrow{h} Y \\
& \searrow \downarrow f \\
Z & \quad \quad \quad \downarrow g
\end{align*}$$

But we immediately run into a problem:

- To define the composition in $C/Z$, we must use that composition in $C$ is associative.
Slice Category?

For $Z \in C$, we would like to define the slice category $C/Z$.

1. Objects:

   $$f : X \to Z$$

2. Morphisms:

   $$\begin{array}{ccc}
   X & \xrightarrow{h} & Y \\
   f \downarrow & \quad & \downarrow g \\
   Z & &
   \end{array}$$

But we immediately run into a problem:

- To define the composition in $C/Z$, we must use that composition in $C$ is associative.
- We cannot show that the composition in $C/Z$ is itself associative without more axioms!
Coherence Conditions

A sufficient condition for composition in $\mathbb{C}/\mathbb{Z}$ to be associative is the well known pentagon identity:

We can amend the definition of category to included this new law (which lives in a doubly iterated identity type) ...
A Vicious Circle Starts

... But!

Then we will need to prove the pentagon axiom for $C/Z$
A Vicious Circle Starts

... But!

Then we will need to prove the pentagon axiom for $C/Z$

And we quickly find that in order to do so, we need another axiom:
The classic “coherence problem”

Now it becomes clear that this completely elementary construction is not well defined unless we add \textit{infinitely many} laws to our definition of category.

\[
\begin{array}{c|c}
C & C/Z \\
\hline
\text{assoc} & \circ \\
\text{assoc }-2 & \text{assoc} \\
\text{assoc }-3 & \text{assoc }-2 \\
\text{assoc }-4 & \text{assoc }-3 \\
\vdots & \vdots \\
\end{array}
\]
The classic “coherence problem”

Now it becomes clear that this completely elementary construction is not well defined unless we add *infinitely many* laws to our definition of category

\[
\begin{array}{c|c}
C & C/Z \\
\hline
\text{assoc} & \circ \\
\text{assoc} - 2 & \text{assoc} \\
\text{assoc} - 3 & \text{assoc} - 2 \\
\text{assoc} - 4 & \text{assoc} - 3 \\
\vdots & \vdots \\
\end{array}
\]

▶ We can avoid this problem by supposing that our types are h-sets. (or, for example, that they have decidable equality)
The classic “coherence problem”

Now it becomes clear that this completely elementary construction is not well defined unless we add \textit{infinitely many} laws to our definition of category

\[
\begin{array}{c|c}
C & C/Z \\
\hline
\text{assoc} & \circ \\
\text{assoc} - 2 & \text{assoc} \\
\text{assoc} - 3 & \text{assoc} - 2 \\
\text{assoc} - 4 & \text{assoc} - 3 \\
\vdots & \vdots \\
\end{array}
\]

▶ We can avoid this problem by supposing that our types are h-sets. (or, for example, that they have decidable equality)

▶ But what is the correct notion of category for a general type?
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:

  1. Monoids
  2. Abelian Groups
  3. Commutative Rings
  4. Modules
  5. Lie Algebras
  6. Operads
  7. $n$-categories
  8. etc...

A number of these "higher structures" already exist (and are useful) in modern algebraic topology and algebraic geometry.

We need a general theory of algebra on types.
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:
  
  1. Monoids
Categories are not the only structure we would like to generalize to arbitrary types:

1. Monoids
2. Abelian Groups
Categories are not the only structure we would like to generalize to arbitrary types:

1. Monoids
2. Abelian Groups
3. Commutative Rings
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:
  1. Monoids
  2. Abelian Groups
  3. Commutative Rings
  4. Modules
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:
  1. Monoids
  2. Abelian Groups
  3. Commutative Rings
  4. Modules
  5. Lie Algebras
Higher Universal Algebra

Categories are not the only structure we would like to generalize to arbitrary types:

1. Monoids
2. Abelian Groups
3. Commutative Rings
4. Modules
5. Lie Algebras
6. Operads
Higher Universal Algebra

Categories are not the only structure we would like to generalize to arbitrary types:

1. Monoids
2. Abelian Groups
3. Commutative Rings
4. Modules
5. Lie Algebras
6. Operads
7. \(n\)-categories
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:
  1. Monoids
  2. Abelian Groups
  3. Commutative Rings
  4. Modules
  5. Lie Algebras
  6. Operads
  7. $n$-categories
  8. etc ...
Higher Universal Algebra

Categories are not the only structure we would like to generalize to arbitrary types:

1. Monoids
2. Abelian Groups
3. Commutative Rings
4. Modules
5. Lie Algebras
6. Operads
7. $n$-categories
8. etc ...

A number of these “higher structures” already exist (and are useful) in modern algebraic topology and algebraic geometry.
Higher Universal Algebra

- Categories are not the only structure we would like to generalize to arbitrary types:
  1. Monoids
  2. Abelian Groups
  3. Commutative Rings
  4. Modules
  5. Lie Algebras
  6. Operads
  7. $n$-categories
  8. etc ...

- A number of these “higher structures” already exist (and are useful) in modern algebraic topology and algebraic geometry

- We need a *general* theory of algebra on types
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 $\text{Op} : I \to \text{Type}$
2. For each operation, a family of sorted parameters $\text{Param} : (i : I)(f : \text{Op} i) \to i \to \text{Type}$

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

$$\text{Op} : I \rightarrow \text{Type}$$
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

   \[ \text{Op} : I \to \text{Type} \]

2. For each operation, a family of sorted parameters

   \[ \text{Param} : (i : I)(f : \text{Op } i) \to i \to \text{Type} \]
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*

   \[
   \text{Op} : I \rightarrow \text{Type}
   \]

2. For each operation, a family of sorted *parameters*

   \[
   \text{Param} : (i : I)(f : \text{Op } i) \rightarrow I \rightarrow \text{Type}
   \]

   For \( i : I \), an element \( f : \text{Op } i \) 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

$$\text{Op} : I \rightarrow \text{Type}$$

2. For each operation, a family of sorted parameters

$$\text{Param} : (i : I)(f : \text{Op } i) \rightarrow I \rightarrow \text{Type}$$

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

It is helpful to think of our signature as having "elements" consisting of the operations.

We can think of them as typed symbols:

\[ f(i_0, i_1, i_2) : i \]

Or we can depict them graphically:
Trees

Associated to any polynomial $P : \text{Poly} \ l$ is its $W$-Type, that is, the type of “well-typed terms” generated by the signature.

Definition
Let $P : \text{Poly} \ l$ be a polynomial. Define

$$W \ P : l \rightarrow \text{Type}$$

$$\text{lf} : (i : l) \rightarrow W \ P \ i$$

$$\text{nd} : (i : l) \rightarrow (f : \text{Op} \ Pi)$$

$$\rightarrow (\delta : (j : J)(p : \text{Param} \ f \ j) \rightarrow W \ P \ j)$$

$$\rightarrow W \ P \ i$$
Representations of Trees

- We can represent elements of $WP$ as terms

$$k(g(f(u, v)), h(x, y), z) : w$$
Representations of Trees

- We can represent elements of $W P$ as terms

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

- Or graphically as actual trees:
Leaves and Nodes

It is not hard to define the “type of leaves” and “type of nodes” of a given tree. These have types:

Leaf : \( \{ i : I \}(w : W P i)(j : I) \rightarrow Type \)

Node : \( \{ i : I \}(w : W P i)\{ j : I \}(g : Op P j) \rightarrow Type \)

Leaf \( w x = \{ \bullet, \bullet \} \)

Node \( w f = \{ \bullet, \bullet \} \)
Adding Relations to our Signature

- The next step is to add some axioms/relations to our structure.

1. Relate some term with a single operation
2. Preserve the number of variables

Good:
$$f(g(x, y), z) = k(x, y, z)$$

Bad:
$$f(g(x, y), z) = h(x, k(y))$$

The reason for these restrictions is we can then encode such relations as a multiplication operator on the signature itself.
Adding Relations to our Signature

- The next step is to add some axioms/relations to our structure.
- In order to single out a “tractable” class of structures we will consider relations which:

  1. Relate some term with a single operation.
  2. Preserve the number of variables.

  Good:
  \[ f(g(x, y), z) = k(x, y, z) \]

  Bad:
  \[ f(g(x, y), z) = h(x, k(y)) \]

The reason for these restrictions is we can then encode such relations as a multiplication operator on the signature itself.
Adding Relations to our Signature

- The next step is to add some axioms/relations to our structure
- In order to single out a “tractable” class of structures we will consider relations which
  1. Relate some term with a single operation

Good:
\[ f(g(x, y)), z) = k(x, y, z) \]

Bad:
\[ f(g(x, y)), z) = h(x, k(y)) \]
The next step is to add some axioms/relations to our structure. In order to single out a “tractable” class of structures we will consider relations which:

1. Relate some term with a single operation
2. Preserve the number of variables
The next step is to add some axioms/relations to our structure.

In order to single out a “tractable” class of structures we will consider relations which:

1. Relate some term with a single operation
2. Preserve the number of variables

Good: \( f(g(x, y), z) = k(x, y, z) \)

Bad: \( f(g(x, y), z) = h(x, k(y)) \)
Adding Relations to our Signature

- The next step is to add some axioms/relations to our structure.
- In order to single out a “tractable” class of structures we will consider relations which
  1. Relate some term with a single operation
  2. Preserve the number of variables

  Good: \( f(g(x, y), z) = k(x, y, z) \)
  Bad: \( f(g(x, y), z) = h(x, k(y)) \)

- The reason for these restrictions is we can then encode such relations as a multiplication operator on the signature itself.
Frames and Magmas

Definition
Let $P : \text{Poly} \ I$ be a polynomial $w : W \ P \ i$ a tree and $f : \text{Op} \ P \ i$ and operation. A frame from $w$ to $f$ is

$$(j : I) \rightarrow \text{Leaf} \ w \ j \simeq \text{Param} \ P \ f \ j$$
Frames and Magmas

Definition
Let $P : \text{Poly} \ 1$ be a polynomial $w : WP i$ a tree and $f : \text{Op} \ 1$ and operation. A \textit{frame} from $w$ to $f$ is

$$(j : 1) \rightarrow \text{Leaf} \ w \ j \simeq \text{Param} \ P \ f \ j$$

Definition
Let $P : \text{Poly} \ 1$ be a polynomial. A \textit{polynomial magma} $M$ over $P$ is

1. A function $\mu : (i : 1) \rightarrow WP i \rightarrow \text{Op} \ 1$
2. A function $\mu_f : (i : 1)(w : WP i) \rightarrow \text{Frame} \ P \ w \ (\mu \ w)$
Visualization of Relations

\[ \mu(f, g, h, k) = r(u, v, x, y, z) \]
Visualization of Relations

\[ k(g(f(u, v)), h(x, y), z) = r(u, v, x, y, z) \]
Visualization of Relations

\[ k(g(f(u, v)), h(x, y), z) = r(u, v, x, y, z) \]

\[ \mu(f, g, h, k) = r \]
Visualization of Relations

\[ k(g(f(u, v)), h(x, y), z) = r(u, v, x, y, z) \]

\[ \mu(f, g, h, k) = r \]
Visualization of Relations

\[ k(g(f(u, v)), h(x, y), z) = r(u, v, x, y, z) \]

\[ \mu(f, g, h, k) = r \]
The Slice of a Polynomial

Given a polynomial $P : \text{Poly} I$ and a magma $M : \text{PolyMagma} P$, we can define a new polynomial $P // M : \text{Poly}(\Sigma I \text{ Op})$ as follows:

$$\text{Op}(P // M)(i, f) := \sum_{w : W \ P i} \mu w = f$$

$$\text{Param}(P // M)(i, f)(w, e)(j, g) := \text{Node} w g$$
Visualizing Iterated Compositions

A tree in the slice polynomial $P//M$ can be visualized as representing a sequence of applications of the multiplication $\mu$. 

![Diagram of a tree structure](image)
Visualizing Iterated Compositions

A tree in the slice polynomial $P//M$ can be visualized as representing a sequence of applications of the multiplication $\mu$. 
A tree in the slice polynomial $P//M$ can be visualized as representing a sequence of applications of the multiplication $\mu$. 
Visualizing Iterated Compositions

A tree in the slice polynomial $P//M$ can be visualized as representing a sequence of applications of the multiplication $\mu$. 
Flattening

There is a function:

\[
\text{flatten} : (i : I)(f : \text{Op } P \ i) \rightarrow W(P//M)(i, f) \rightarrow WPi
\]

which, given a pasting diagram extracts its “boundary”:
There is a function:

\[
\text{flatten} : (i : I)(f : \text{Op} \ P \ i) \to W(P//M)(i, f) \to WPi
\]

which, given a pasting diagram extracts its “boundary”:
Invariance by Subdivision

A coherence witness $\Psi$ for $M$ is proof that the multiplication $\mu$ is invariant under all subdivisions.

$$\Psi : (i : I)(f : \text{Op } P i)(pd : W(P//M)(i, f))$$

$$\rightarrow \mu(\text{flatten } pd) = f$$
Invariance by Subdivision

A coherence witness $\Psi$ for $M$ is proof that the multiplication $\mu$ is invariant under all subdivisions.

$$
\Psi : (i : I)(f : \mathrm{Op} P i)(pd : W(P/\!/M)(i, f)) \rightarrow \mu(\text{flatten } pd) = f
$$

It turns out that, given a coherence witness $\Psi$, we can define a magma structure on $P/\!/M$ which we will write $M_\Psi$. 
Invariance by Subdivision

A coherence witness $\Psi$ for $M$ is proof that the multiplication $\mu$ is invariant under all subdivisions.

$$\Psi : (i : I)(f : \text{Op} P i)(pd : W(P//M)(i, f)) \rightarrow \mu(\text{flatten pd}) = f$$

It turns out that, given a coherence witness $\Psi$, we can define a magma structure on $P//M$ which we will write $M_\Psi$. 
Invariance by Subdivision

A coherence witness $\Psi$ for $M$ is proof that the multiplication $\mu$ is invariant under all subdivisions.

$$\Psi : (i : I)(f : \text{Op } P i)(pd : W(P//M)(i, f)) \rightarrow \mu(\text{flatten } pd) = f$$

It turns out that, given a coherence witness $\Psi$, we can define a magma structure on $P//M$ which we will write $M_\Psi$. 
Invariance by Subdivision

A coherence witness $\Psi$ for $M$ is proof that the multiplication $\mu$ is invariant under all subdivisions.

$$\Psi : (i : I)(f : \text{Op} P i)(pd : W(P//M)(i, f)) \rightarrow \mu(\text{flatten } pd) = f$$

It turns out that, given a coherence witness $\Psi$, we can define a magma structure on $P//M$ which we will write $M_\Psi$. 
Invariance and Associativity
Invariance and Associativity

\[ \mu(\mu(f, g), \mu(h), k) = r \]
\[ \mu(f, g, h, k) = r \]
Invariance and Associativity
Invariance and Associativity
Invariance and Associativity
Invariance and Associativity
Invariance and Associativity
Invariance and Associativity

\[ \mu(\mu(f), \mu(g, h, k)) = p \]
\[ \mu(f, g, h, k) = p \]
Invariance and Associativity

\[ p = \mu(f, g, h, k) = r \]

\[ \mu(\mu(f, g), \mu(h), k) = \mu(\mu(f), \mu(g, h, k)) \]
Polynomial Monads

The advantage of this formulation is that we can now define a coherent structure as one for which one can find an infinite sequence of such extensions.

Definition
A coherence structure for $M$ consists of

1. A coherence witness $\Psi$ for $M$
2. Coinductively, a coherence structure for $M_\Psi$
Polynomial Monads

The advantage of this formulation is that we can now define a coherent structure as one for which one can find an infinite sequence of such extensions.

Definition
A coherence structure for $M$ consists of
1. A coherence witness $\Psi$ for $M$
2. Coinductively, a coherence structure for $M_\Psi$

Definition
A *polynomial monad* is consists of
1. A polynomial $P : \text{Poly} I$
2. A magma $M : \text{PolyMagma} P$
3. A coherence structure $C$ for $M$
Applications and future work

- A special case of this definition gives a complete definition of category structure on a type

Remains to explore examples and use these techniques to prove coherence theorems.
Applications and future work

- A special case of this definition gives a complete definition of category structure on a type
- More generally, this definition gives us an internalization of higher operads
Applications and future work

- A special case of this definition gives a complete definition of category structure on a type.
- More generally, this definition gives us an internalization of higher operads.
- Remains to explore examples and use these techniques to prove coherence theorems.
Thank you!