

# **Higher-order Arities, Signatures and Equations via Modules**

Ambroise Lafont

joint work with  
Benedikt Ahrens, André Hirschowitz, Marco Maggesi

Work submitted to FSCD 2019

# Keywords associated with syntax

Recursion

Substitution

Model

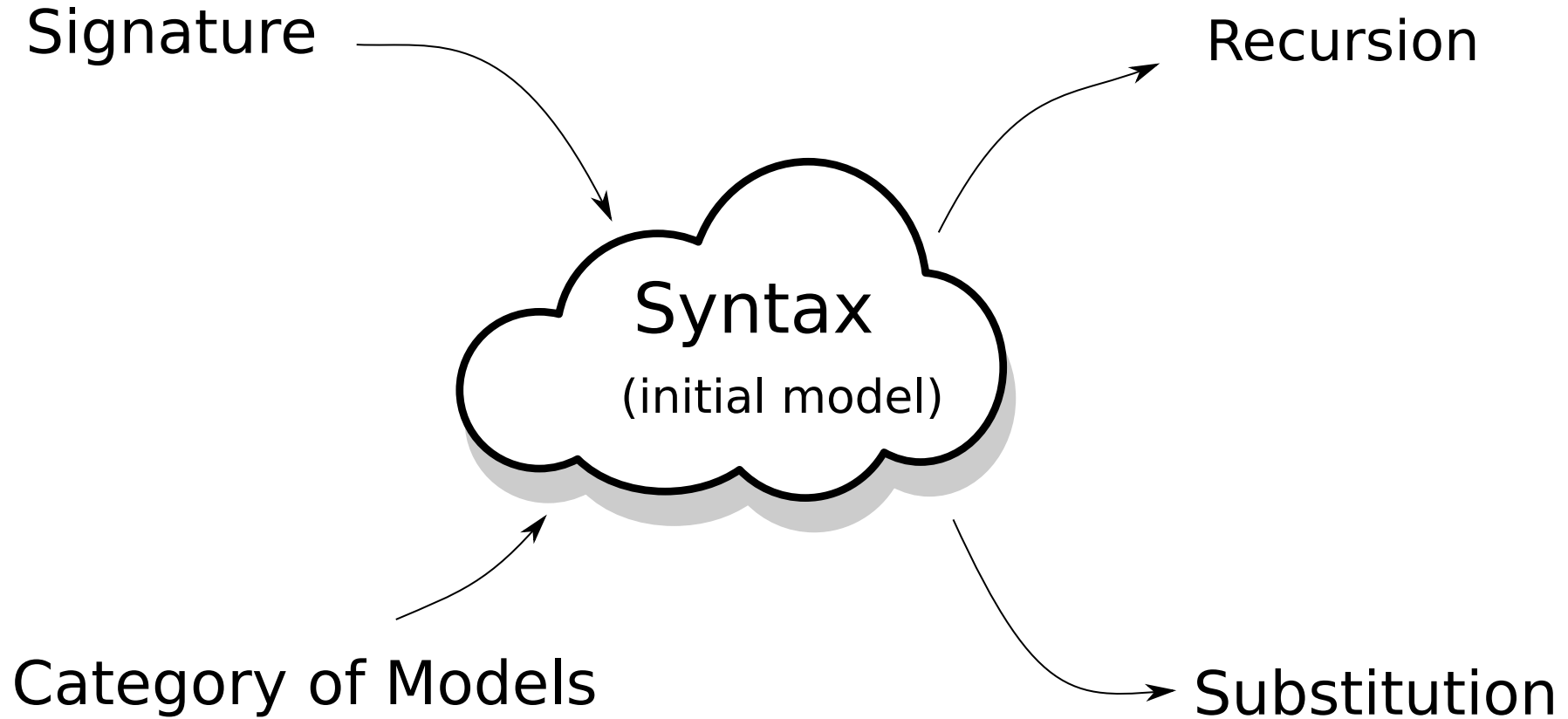
**Syntax**

Operation/Construction

Arity/Signature

**This talk:** give a mathematical framework which *catch 'em all*

# What is a syntax?



**generates a syntax** = existence of the initial model

# Our target: LCD

## Syntax of the *differentiable $\lambda$ -calculus*:

Simple terms  $s, t \in \Lambda$

$s, t ::=$	$x$	(variable)
	$\lambda x. t$	(modulo $\alpha$ -renaming of $x$ )
	$s t$	
	$Ds \cdot t$	
	$s + t$	(modulo associativity and commutativity)
	$0$	(neutral element for $+$ )

# Our target: LCD

## Syntax of the *differentiable $\lambda$ -calculus*:

Simple terms  $s, t \in \Lambda$

$s, t ::=$	$x$	(variable)
	$\lambda x. t$	(modulo $\alpha$ -renaming of $x$ )
	$s t$	
	$Ds \cdot t$	
	$s + t$	(modulo associativity and commutativity)
	$0$	(neutral element for $+$ )

subject to the following equation:

$$D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t$$

# Our target: LCD

## Syntax of the *differentiable* $\lambda$ -calculus:

Simple terms  $s, t \in \Lambda$

$s, t ::=$	$x$	(variable)
	$\lambda x. t$	(modulo $\alpha$ -renaming of $x$ )
	$s t$	
	$Ds \cdot t$	
	$s + t$	(modulo associativity and commutativity)
	$0$	(neutral element for $+$ )

subject to the following equation:

$$D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t$$

and linearity of constructors with respect to  $+$ :

$$\lambda x. (s + t) = \lambda x. s + \lambda x. t \quad \dots$$

# Overview

**Topic:** specification and construction of untyped syntaxes with variables and a well-behaved substitution (e.g. lambda calculus).

## Our work:

1. general notion of **1-signature** based on **monads** and **modules**.
  - *Caveat:* Not all of them do **generate a syntax**
  - special case: classical **algebraic 1-signatures** generate a syntax
2. notion of **2-signature**: a pair of a 1-signature and a set of equations.
  - special case: **algebraic 2-signatures** generate a syntax

# Some examples covered by our result

## Operations:

- Commutative binary operation

$$m : T \times T \rightarrow T \quad \text{s.t.} \quad m(t, u) = m(u, t)$$

- Fixed point operation



# Some examples covered by our result

## Operations:

- Commutative binary operation

$$m : T \times T \rightarrow T \quad \text{s.t.} \quad m(t, u) = m(u, t)$$

- Fixed point operation

## More extensive examples (set of operations with equations):

- $\lambda$ -calculus modulo  $\beta\eta$
- differential  $\lambda$ -calculus

# Table of contents

- 1. Review: Binding signatures and their models**
2. 1-Signatures and models based on monads and modules
3. Equations
4. Recursion

# Table of contents

## **1. Review: Binding signatures and their models**

- Categorical formulation of term languages
- Initial semantics for binding signatures

## 2. 1-Signatures and models based on monads and modules

## 3. Equations

## 4. Recursion

# Categorical formulation of a term language

**Example:** syntax with a binary operation  $\star$ , a constant 0, and variables

$$\begin{array}{ll} \text{expr} ::= x & (\text{variable}) \\ | t_1 \star t_2 & (\text{binary operation}) \\ | 0 & (\text{constant}) \end{array}$$

The syntax can be considered as the endofunctor  $B$  (on Set):

$$B : X \mapsto \{\text{expressions over } X\}$$

For example:

$$\begin{aligned} B(\emptyset) &= \{0, 0 \star 0, \dots\} \\ B(\{x, y\}) &= \{0, 0 \star 0, \dots, x, y, x \star y, \dots\} \end{aligned}$$

# Categorical formulation of a term language

Then we have:

$$\star : B \times B \rightrightarrows B$$

$$0 : 1 \rightrightarrows B$$

$$\text{var} : \text{Id}_{\text{Set}} \rightrightarrows B$$

Putting all together:

$$B \times B + 1 + \text{Id}_{\text{Set}} \rightrightarrows B$$

i.e.  $B$  is an algebra for the endofunctor  $F \mapsto F \times F + 1 + \text{Id}_{\text{Set}}$  on the category  $\text{End}_{\text{Set}}$ .

Actually,  $B$  can be **characterized** as the initial algebra.

# Binding Signatures

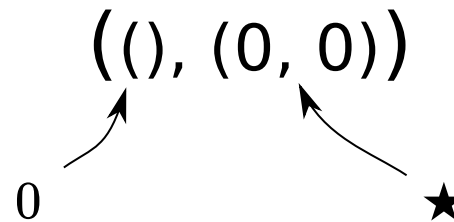
## Definition

**Binding signature** = a family of lists of natural numbers.

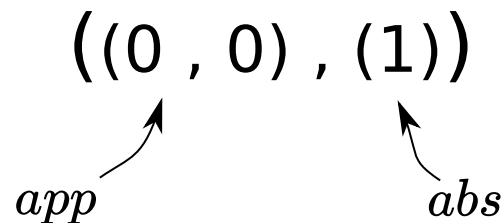
Each list specifies one operation in the syntax:

- length of the list = number of arguments of the operation
- natural number in the list = number of bound variables in the corresponding argument

**Syntax with 0, ★:**



**Lambda calculus:**



# Initial semantics for binding signatures

## Reminder

The syntax  $(0, \star)$  is the initial algebra for the endofunctor:

$$F \mapsto F \times F + 1 + \text{Id}_{\text{Set}}$$

More generally, any binding signature gives rise to an endofunctor  $\Sigma$ .

### Definition

**Model** =  $(\Sigma + \text{Id}_{\text{Set}})$ -algebra

### Classical Theorem

The initial  $(\Sigma + \text{Id}_{\text{Set}})$ -algebra of a binding signature  $\Sigma$  always exists.

**Question:** Does this initial algebra come with a well-behaved substitution?

**Answer:** Yes: see e.g. [Fiore, Plotkin, Turi 1999], [Ghani & Uustalu 2003]

# Table of contents

1. Review: Binding signatures and their models

## **2. 1-Signatures and models based on monads and modules**

- Our take on substitution
- Our take on 1-signatures, models and syntax
- Our take on binding 1-signatures

3. Equations

4. Recursion



# The Big Picture of 1-signatures and models

Binding signatures  $\hookrightarrow$  Our 1-signatures

A **1-signature**  $\Sigma$  is a functorial assignment:

$$R \mapsto \Sigma(R)$$

A **model of**  $\Sigma$  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad  $:=$  endofunctor with substitution

module over a monad  $:=$  endofunctor with substitution

module morphism  $:=$  natural transformation preserving substitution

# The Big Picture of 1-signatures and models

Binding signatures  $\hookrightarrow$  Our 1-signatures

A **1-signature**  $\Sigma$  is a functorial assignment:

$$\text{monad} \nearrow R \mapsto \Sigma(R)$$

A **model of**  $\Sigma$  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

# The Big Picture of 1-signatures and models

Binding signatures  $\hookrightarrow$  Our 1-signatures

A **1-signature**  $\Sigma$  is a functorial assignment:

$$R \mapsto \Sigma(R)$$

monad  $\nearrow$   $\nwarrow$  module over  $R$

A **model of**  $\Sigma$  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

# The Big Picture of 1-signatures and models

Binding signatures  $\hookrightarrow$  Our 1-signatures

A **1-signature**  $\Sigma$  is a functorial assignment:

$$R \mapsto \Sigma(R)$$

monad  $\nearrow$   $\nwarrow$  module over  $R$

A **model of**  $\Sigma$  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad  $\nearrow$

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

# The Big Picture of 1-signatures and models

Binding signatures  $\hookrightarrow$  Our 1-signatures

A **1-signature**  $\Sigma$  is a functorial assignment:

$$R \mapsto \Sigma(R)$$

monad  $\nearrow$   $\nwarrow$  module over  $R$

A **model of**  $\Sigma$  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad  $\nearrow$   $\nwarrow$  module morphism

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

# Substitution and monads

## Reminder:

- $B(X)$  = expressions built out of 0,  $\star$  and variables taken in  $X$
- Variables induce a natural transformation  $\text{var} : \text{Id}_{\text{Set}} \rightarrow B$

## Substitution:

$$\text{bind} : B(X) \rightarrow (X \rightarrow B(Y)) \rightarrow B(Y)$$

+ laws

A triple  $(B, \text{var}, \text{bind})$  is called a **monad**.

**monad morphism** = mapping preserving  $\text{var}$  and  $\text{bind}$ .

# Monads

1.  $B : \text{Set} \rightarrow \text{Set}$

$B(X)$  = expressions built out of  $0$ ,  $\star$  and variables taken in  $X$

2. A collection of functions  $(\text{var}_X : X \rightarrow B(X))_X$

*Variables are expressions*

3. For each function  $u : X \rightarrow B(Y)$ , a function  $\text{bind}_u : B(X) \rightarrow B(Y)$

*Parallel substitution*

**Notation:**  $\text{bind}_u(t) = t[x \mapsto u(x)]$

4. Monadic laws:

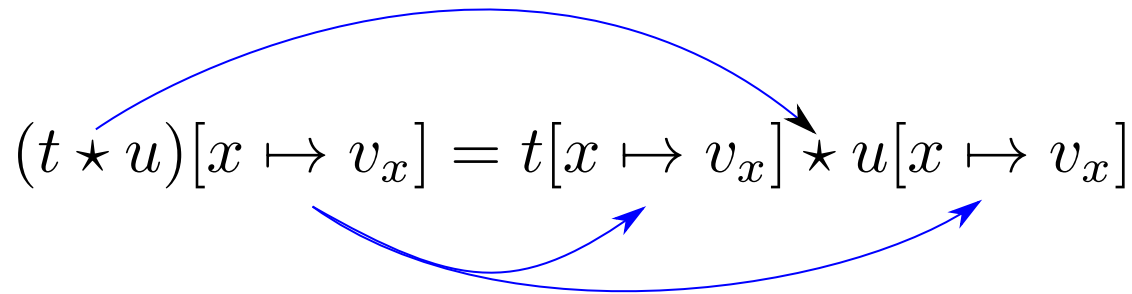
$$\text{var}(y)[x \mapsto u(x)] = u(y)$$

$$t[x \mapsto \text{var}(x)] = t$$

$$t[x \mapsto f(x)][y \mapsto g(y)] = t[x \mapsto f(x)[y \mapsto g(y)]]$$

# Preview: Operations are module morphisms

## ★ commutes with substitution

$$(t \star u)[x \mapsto v_x] = t[x \mapsto v_x] \star u[x \mapsto v_x]$$


## Categorical formulation

$B \times B$  supports  $B$ -substitution  $\rightsquigarrow$   $B \times B$  is a **module over**  $B$

★ commutes with substitution  $\rightsquigarrow$  ★ :  $B \times B \rightarrow B$  is a **module morphism**



# Modules VS Monads

## Monad

1.  $B : \text{Set} \rightarrow \text{Set}$

$B(X)$  = expressions built out of  $0$ ,  $\star$  and variables taken in  $X$

2. A collection of functions  $(\text{var}_X : X \rightarrow B(X))_X$

*Variables are expressions*

3. For each function  $u : X \rightarrow B(Y)$ , a function  $\text{bind}_u : B(X) \rightarrow B(Y)$

*Parallel substitution*

**Notation:**  $\text{bind}_u(t) = t[x \mapsto u(x)]^B$

4. Substitution laws:

$$\text{var}(y)[x \mapsto u(x)]^B = u(y)$$

$$t[x \mapsto \text{var}(x)]^B = t$$

$$t[x \mapsto f(x)]^B[y \mapsto g(y)]^B = t[x \mapsto f(x)[y \mapsto g(y)]^B]^B$$

# Modules VS Monads

~~Monad~~ **Module over a monad**  $B$  (e.g.  $B \times B, 2, \dots$ )

1.  $M : \text{Set} \rightarrow \text{Set}$

$M(X) = \text{expressions taking variables in } X$

~~2. A collection of functions  $(\text{var}_X : X \rightarrow M(X))_X$~~

3. For each function  $u : X \rightarrow B(Y)$ , a function  $\text{bind}_u : M(X) \rightarrow M(Y)$

*Parallel substitution*

**Notation:**  $\text{bind}_u(t) = t[x \mapsto u(x)]^M$

4. Substitution laws:

$$\text{var}(y)[x \mapsto u(x)] = u(y)$$

$$t[x \mapsto \text{var}(x)]^M = t$$

$$t[x \mapsto f(x)]^M[y \mapsto g(y)]^M = t[x \mapsto f(x)[y \mapsto g(y)]^B]^M$$

# Building blocks for binding signatures

Essential constructions of **modules over a monad  $R$** :

- $R$  itself
- $M \times N$  for any modules  $M$  and  $N$  (in particular,  $R \times R$ )
- The **derivative of a module  $M$**  is the module  $M'$  defined by  $M'(X) = M(X + \{\diamond\})$ .

The derivative is used to model an operation binding a variable  
(Cf next slide).

# Syntactic operations are module morphisms

**module morphism** = maps commuting with substitution.

$$id_M : M \rightarrow M$$

$$0 : 1 \rightarrow B$$

$$\star : B \times B \rightarrow B$$

$$app : \Lambda \times \Lambda \rightarrow \Lambda$$

$$abs : \Lambda' \rightarrow \Lambda$$

# The Big Picture again

A **1-signature**  $\Sigma$  is a functorial assignment:

$$R \mapsto \Sigma(R)$$

monad  $\quad$  module over  $R$

A **model of  $\Sigma$**  is a pair:

$$(R, \rho : \Sigma(R) \rightarrow R)$$

monad  $\quad$  module morphism

A **model morphism**  $m : (R, \rho) \rightarrow (S, \sigma)$  is a monad morphism commuting with the module morphism:

$$\begin{array}{ccc} \Sigma(R) & \xrightarrow{\rho} & R \\ \Sigma(m) \downarrow & & \downarrow m \\ \Sigma(S) & \xrightarrow{\sigma} & S \end{array}$$

# Syntax

## Definition

Given a 1-signature  $\Sigma$ , its **syntax** is an initial object in its category of models.

**Question:** Does the syntax exist for every 1-signature?

**Answer:** No.

**Counter-example:** the 1-signature  $R \mapsto \mathcal{P} \circ R$



powerset endofunctor on Set

# Examples of 1-signatures generating syntax

- **(0,★) language:**

Signature:  $R \mapsto 1 + R \times R$

Model:  $(R, \quad 0 : 1 \rightarrow R, \quad \star : R \times R \rightarrow R)$

Syntax:  $(B, \quad 0 : 1 \rightarrow B, \quad \star : B \times B \rightarrow B)$

- **lambda calculus:**

Signature:  $R \mapsto R' + R \times R$

Model:  $(R, \quad abs : R' \rightarrow R, \quad app : R \times R \rightarrow R)$

Syntax:  $(\Lambda, \quad abs : \Lambda' \rightarrow \Lambda, \quad app : \Lambda \times \Lambda \rightarrow \Lambda)$

Can we generalize this pattern?

# Initial semantics for algebraic 1-signatures

Theorem [Hirschowitz & Maggesi 2007]

Syntax exists for any **algebraic 1-signature**, i.e. 1-signature built out of derivatives, products, coproducts, and the trivial 1-signature  $R \mapsto R$ .

**Algebraic 1-signatures** correspond to binding signatures through the embedding:

Binding signatures  $\hookrightarrow$  Our 1-signatures

**Question:** Can we enforce some equations in the syntax ?

For example: lambda calculus modulo beta and eta.



# Table of contents

1. Review: Binding 1-signatures and their models
2. 1-Signatures and models based on monads and modules
- 3. Equations**
4. Recursion

# Example: a commutative binary operation

## Specification of a binary operation

1-Signature:  $R \mapsto R \times R$

Model:  $(R, + : R \times R \rightarrow R)$

**What is an appropriate notion of model for a commutative binary operation ?**

# Example: a commutative binary operation

## Specification of a **commutative** binary operation

1-Signature:  $R \mapsto R \times R$

Model:  $(R, + : R \times R \rightarrow R)$  s.t.  $t + u = u + t$  (1)

**What is an appropriate notion of model for a commutative binary operation ?**

**Answer:** a monad equipped with a **commutative** binary operation

# Example: a commutative binary operation

## Specification of a **commutative** binary operation

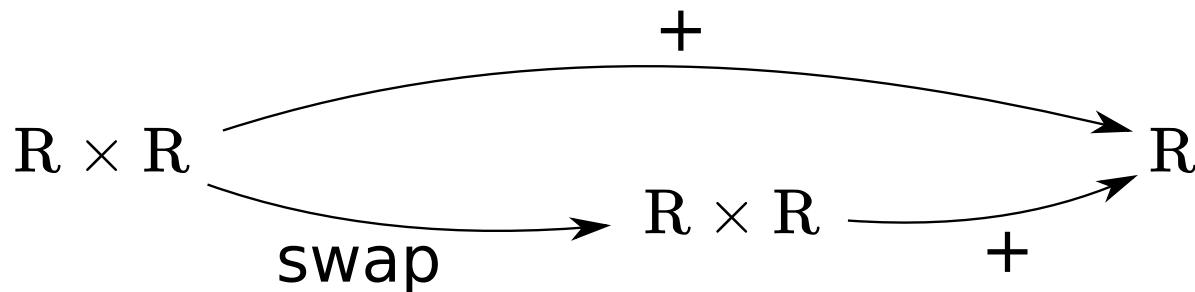
1-Signature:  $R \mapsto R \times R$

Model:  $(R, + : R \times R \rightarrow R)$  s.t.  $t + u = u + t$  (1)

**What is an appropriate notion of model for a commutative binary operation ?**

**Answer:** a monad equipped with a **commutative** binary operation

Equation (1) states an equality between  $R$ -module morphisms:



# Review: Signatures with equations

- [Fiore-Hur 2010]: existence of an initial model for an inductively defined (with a specific syntax) set of possible equations.
- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures generate a syntax (e.g. a binary commutative operation).

# Review: Signatures with equations

- [Fiore-Hur 2010]: existence of an initial model for an inductively defined (with a specific syntax) set of possible equations.

Our framework: alternative approach where monads and modules are the central notions.

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures generate a syntax (e.g. a binary commutative operation).

# Review: Signatures with equations

- [Fiore-Hur 2010]: existence of an initial model for an inductively defined (with a specific syntax) set of possible equations.

Our framework: alternative approach where monads and modules are the central notions.

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures generate a syntax (e.g. a binary commutative operation).

This work: more general equations (e.g.  $\lambda$ -calculus modulo  $\beta\eta$ ).

# Equations

Given a 1-signature  $\Sigma$ , a  **$\Sigma$ -equation**  $A \rightrightarrows B$  is a functorial assignment

$$R \mapsto \left( A(R) \rightrightarrows B(R) \right)$$

model of  $\Sigma$  parallel pair of module morphisms over  $R$

A **2-signature** is a pair

$$(\Sigma, E)$$

1-signature set of  $\Sigma$ -equations

**model of a 2-signature**  $(\Sigma, E)$ :

- a model  $R$  of  $\Sigma$
- s.t.  $\forall (A \rightrightarrows B) \in E$ , the two morphisms  $A(R) \rightrightarrows B(R)$  are equal



# Algebraic 2-signatures

Given a 1-signature  $\Sigma$ , a  $\Sigma$ -equation  $A \Rightarrow B$  is **elementary** if:

1.  $A$  "preserves pointwise epimorphisms"

(e.g., any "algebraic 1-signature")

2.  $B$  is of the form  $R \mapsto R'$  (e.g.  $R \mapsto R$ )

**Algebraic** 2-signature:

$(\Sigma, E)$

**algebraic** 1-signature  $\nearrow$   $\nwarrow$  set of **elementary**  
 $\Sigma$ -equations

Theorem

Syntax exists for any algebraic 2-signature

# Example: $\lambda$ -calculus modulo $\beta\eta$

The algebraic 2-signature  $(\Sigma_{\text{LC}\beta\eta}, E_{\text{LC}\beta\eta})$  of  $\lambda$ -calculus modulo  $\beta\eta$ :

$$\Sigma_{\text{LC}\beta\eta}(\mathbf{R}) := \Sigma_{\text{LC}}(\mathbf{R}) = \mathbf{R} \times \mathbf{R} + \mathbf{R}'$$

**model of  $\Sigma_{\text{LC}}$**  = monad  $\mathbf{R}$  with module morphisms:

$$\text{app} : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R} \qquad \text{abs} : \mathbf{R}' \rightarrow \mathbf{R}$$

**$\beta$ -equation:**  $(\lambda x.t) u = t[\underbrace{x \mapsto u}_{\sigma_{\mathbf{R}}(t,u)}]$

**$\eta$ -equation:**  $t = \lambda x.(t x)$

$$E_{\text{LC}\beta\eta} = \{ \beta\text{-equation}, \eta\text{-equation} \}$$

# Example: $\lambda$ -calculus modulo $\beta\eta$

The algebraic 2-signature  $(\Sigma_{\text{LC}\beta\eta}, E_{\text{LC}\beta\eta})$  of  $\lambda$ -calculus modulo  $\beta\eta$ :

$$\Sigma_{\text{LC}\beta\eta}(\mathbf{R}) := \Sigma_{\text{LC}}(\mathbf{R}) = \mathbf{R} \times \mathbf{R} + \mathbf{R}'$$

**model of  $\Sigma_{\text{LC}}$**  = monad  $\mathbf{R}$  with module morphisms:

$$\text{app} : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R} \quad \text{abs} : \mathbf{R}' \rightarrow \mathbf{R}$$

**$\beta$ -equation:**  $(\lambda x.t) u = t[\underbrace{x \mapsto u}_{\sigma_{\mathbf{R}}(t,u)}]$

**$\eta$ -equation:**  $t = \lambda x.(t x)$

$$\begin{array}{ccccc}
 & \sigma_{\mathbf{R}} & & & \\
 \mathbf{R}' \times \mathbf{R} & \xrightarrow{\quad} & \mathbf{R} & & \\
 \text{abs} \times \mathbf{R} \searrow & & \nearrow \text{app} & & \\
 & \mathbf{R} \times \mathbf{R} & & & 
 \end{array}$$

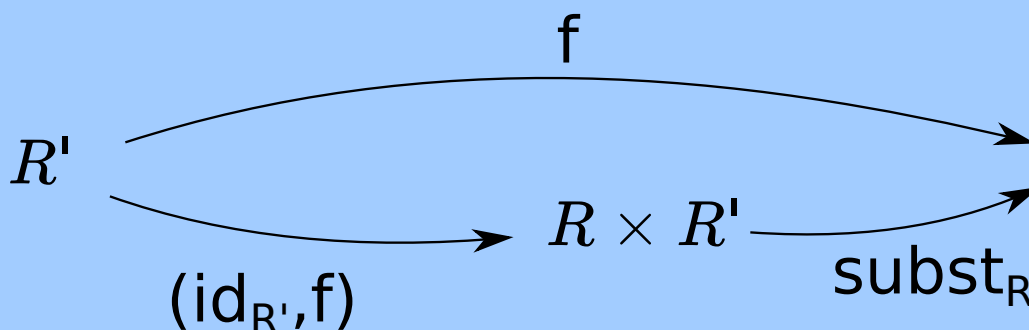
$$\begin{array}{ccccc}
 & \text{id}_{\mathbf{R}} & & & \\
 \mathbf{R} & \xrightarrow{\quad} & \mathbf{R} & & \\
 \downarrow \text{!}_1 & & \nearrow \text{abs} & & \\
 & \mathbf{R}' & & & 
 \end{array}$$

$$E_{\text{LC}\beta\eta} = \{ \beta\text{-equation}, \eta\text{-equation} \}$$

# Example: fixpoint operator

Definition [AHLM CSL 2018]

A **fixpoint operator** in a monad  $R$  is a module morphism  $f : R' \rightarrow R$  s.t.

(1)  commutes.

The algebraic 2-signature  $(\Sigma_{\text{fix}}, E_{\text{fix}})$  of a fixpoint operator:

$$\Sigma_{\text{fix}}(R) := R' \quad E_{\text{fix}} = \{ (1) \}$$

Proposition [AHLM CSL 2018]

**Fixpoint operators** in  $LC_{\beta\eta}$  are in one to one correspondance with fixpoint combinators (i.e.  $\lambda$ -terms  $Y$  s.t.  $t(Yt) = Yt$  for any  $t$ ).

# Combining algebraic 2-signatures

Algebraic 2-signatures can be combined:

fixpoint operator

$\lambda$ -calculus modulo  $\beta\eta$

$(\Sigma_{\text{fix}}, E_{\text{fix}})$

+

$(\Sigma_{\text{LC}\beta\eta}, E_{\text{LC}\beta\eta})$

=

$(\Sigma_{\text{fix}} + \Sigma_{\text{LC}\beta\eta}, E_{\text{fix}} \cup E_{\text{LC}\beta\eta})$

$\lambda$ -calculus modulo  $\beta\eta$  with an explicit fixpoint operator

# Example: free monoid

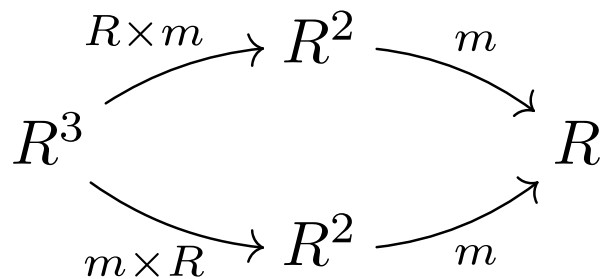
An algebraic 2-signature  $(\Sigma_{\text{mon}}, E_{\text{mon}})$  for the free monoid monad  $X \mapsto \coprod_n X^n$

$$\Sigma_{\text{mon}}(\mathbf{R}) := 1 + \mathbf{R} \times \mathbf{R}$$

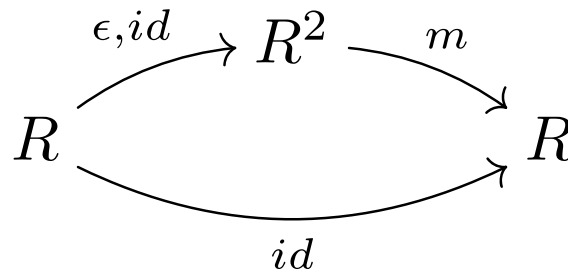
**model of  $\Sigma$**  = monad  $\mathbf{R}$  with module morphisms:

$$\varepsilon : 1 \rightarrow \mathbf{R} \quad m : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$$

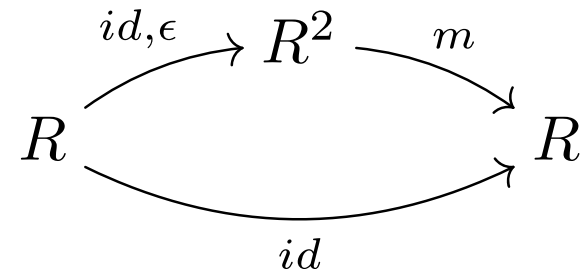
3 elementary  $\Sigma$ -equations:



associativity



left unit



right unit

# Our target: LCD

## Syntax of the *differentiable* $\lambda$ -calculus:

Simple terms  $s, t \in \Lambda$

$$\begin{array}{l|l} s, t ::= & x \\ & \lambda x. t \\ & s \ t \\ & Ds \cdot t \\ & s + t \\ & 0 \end{array} \quad \left. \begin{array}{l} \} \\ \} \end{array} \right\} \begin{array}{l} \lambda\text{-calculus} \\ \text{free commutative monoid} \end{array}$$

subject to the following equation:

$$D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t$$

and linearity of constructors with respect to  $+$ :

$$\lambda x. (s + t) = \lambda x. s + \lambda x. t \quad \dots$$

# Algebraic 1-signature for LCD

## Syntax of the *differentiable* $\lambda$ -calculus:

Simple terms  $s, t \in \Lambda$

Corresponding 1-signature

$s, t ::=$	$x$	$\left. \begin{array}{l} \\ \\ \end{array} \right\}$	$\Sigma_{\text{LC}}(\mathbf{R}) = \mathbf{R}' + \mathbf{R} \times \mathbf{R}$
	$\lambda x. t$		
	$s \ t$		
	$\mathbf{D}s \cdot t$		$\mathbf{R} \mapsto \mathbf{R} \times \mathbf{R}$
	$s + t$	$\left. \begin{array}{l} \\ \end{array} \right\}$	$\Sigma_{\text{mon}}(\mathbf{R}) = 1 + \mathbf{R} \times \mathbf{R}$
	$0$		



# Algebraic 1-signature for LCD

## Syntax of the *differentiable* $\lambda$ -calculus:

Simple terms  $s, t \in \Lambda$

Corresponding 1-signature

$s, t ::=$	$x$	}	$\Sigma_{\text{LC}}(\mathbf{R}) = \mathbf{R}' + \mathbf{R} \times \mathbf{R}$
	$\lambda x. t$		
	$s \ t$		
	$\mathbf{D}s \cdot t$		$\mathbf{R} \mapsto \mathbf{R} \times \mathbf{R}$
	$s + t$	}	$\Sigma_{\text{mon}}(\mathbf{R}) = 1 + \mathbf{R} \times \mathbf{R}$
	$0$		

---

Resulting algebraic 1-signature:

$$\Sigma_{\text{LCD}}(\mathbf{R}) = \Sigma_{\text{LC}}(\mathbf{R}) + \mathbf{R} \times \mathbf{R} + \Sigma_{\text{mon}}(\mathbf{R})$$

# Elementary equations for LCD

## Commutative monoidal structure:

$$\begin{array}{lcl} & s + t = t + s & \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R} \\ E_{\text{mon}} \left\{ \begin{array}{l} s + (t + u) = (s + t) + u \\ 0 + t = t \\ t + 0 = t \end{array} \right. & & \begin{array}{l} \mathbf{R} \times \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R} \\ \mathbf{R} \Rightarrow \mathbf{R} \\ \mathbf{R} \Rightarrow \mathbf{R} \end{array} \end{array}$$

## Differentiation:

$$D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t \qquad \mathbf{R} \times \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R}$$

## Linearity:

$$\lambda x. (s + t) = \lambda x. s + \lambda x. t \qquad \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R}$$

$$D(s + t) \cdot u = Ds \cdot u + Dt \cdot u \qquad \mathbf{R} \times \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R}$$

...


# Table of contents

1. Review: Binding signatures and their models
2. 1-Signatures and models based on monads and modules
3. Equations
- 4. Recursion**

# Principle of recursion

Recursion on the syntax  $\simeq$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$f : R \rightarrow S$   
  
initial model of a 2-signature  $(\Sigma, E)$

# Principle of recursion

Recursion on the syntax  $\simeq$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$$f : R \rightarrow S$$

initial model of a 2-signature  $(\Sigma, E)$



1. Give a module morphism  $s : \Sigma(S) \rightarrow S$

# Principle of recursion

Recursion on the syntax  $\approx$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$$f : R \rightarrow S$$

initial model of a 2-signature  $(\Sigma, E)$



1. Give a module morphism  $s : \Sigma(S) \rightarrow S$   
 $\Rightarrow$  induces a  $\Sigma$ -model  $(S, s)$

# Principle of recursion

Recursion on the syntax  $\simeq$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$$f : R \rightarrow S$$

initial model of a 2-signature  $(\Sigma, E)$



1. Give a module morphism  $s : \Sigma(S) \rightarrow S$   
 $\Rightarrow$  induces a  $\Sigma$ -model  $(S, s)$
2. Show that all the equations in  $E$  are satisfied for this model

# Principle of recursion

Recursion on the syntax  $\simeq$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$$f : R \rightarrow S$$

initial model of a 2-signature  $(\Sigma, E)$




1. Give a module morphism  $s : \Sigma(S) \rightarrow S$   
 $\Rightarrow$  induces a  $\Sigma$ -model  $(S, s)$
2. Show that all the equations in  $E$  are satisfied for this model  
 $\Rightarrow$  induces a model of  $(\Sigma, E)$



# Principle of recursion

Recursion on the syntax  $\approx$  Initiality in the category of models

**Recipe for constructing "by recursion" a monad morphism:**

$f : R \rightarrow S$   
  
initial model of a 2-signature  $(\Sigma, E)$

1. Give a module morphism  $s : \Sigma(S) \rightarrow S$   
 $\Rightarrow$  induces a  $\Sigma$ -model  $(S, s)$
2. Show that all the equations in  $E$  are satisfied for this model  
 $\Rightarrow$  induces a model of  $(\Sigma, E)$

Initiality of  $R \Rightarrow$  model morphism  $R \rightarrow S \Rightarrow$  monad morphism  $R \rightarrow S$

# Example: Computing the set of free variables

$LC$  = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

$\mathcal{P}$  = power set monad

**Definition of a (monad) morphism**  $fv : LC \rightarrow \mathcal{P}$  **s.t.**

$$fv(\text{app}(t, u)) = fv(t) \cup fv(u)$$

$$fv(\text{abs}(t)) = fv(t) \setminus \{\diamond\}$$

# Example: Computing the set of free variables

$LC$  = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

$\mathcal{P}$  = power set monad

**Definition of a (monad) morphism**  $fv : LC \rightarrow \mathcal{P}$  **s.t.**

$$fv(\text{app}(t, u)) = fv(t) \cup fv(u)$$

$$fv(\text{abs}(t)) = fv(t) \setminus \{\diamond\}$$

$\Rightarrow$  make  $\mathcal{P}$  a model of  $\Sigma_{LC}$ :

$$\cup : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$$

$$-\setminus\{\diamond\} : \mathcal{P}' \rightarrow \mathcal{P}$$

# Example: Computing the set of free variables

$LC$  = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

$\mathcal{P}$  = power set monad

**Definition of a (monad) morphism**  $fv : LC \rightarrow \mathcal{P}$  **s.t.**

$$fv(\text{app}(t, u)) = fv(t) \cup fv(u)$$

$$fv(\text{abs}(t)) = fv(t) \setminus \{\diamond\}$$

$\Rightarrow$  make  $\mathcal{P}$  a model of  $\Sigma_{LC}$ :

$$\cup : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$$

$$-\setminus\{\diamond\} : \mathcal{P}' \rightarrow \mathcal{P}$$

Initiality of  $LC \Rightarrow$   $fv : LC \rightarrow \mathcal{P}$  satisfying the above equations (as a model morphism).

# Example: Translating $\lambda$ -calculus with fixpoint

$LC_{\beta\eta\text{fix}}$  = initial model of  $(\Sigma, E) = (\Sigma_{LC\beta\eta} + \Sigma_{\text{fix}}, E_{LC\beta\eta} \cup E_{\text{fix}})$

*$\lambda$ -calculus modulo  $\beta\eta$  with a fixpoint operator  $\text{fix} : LC_{\beta\eta\text{fix}}' \rightarrow LC_{\beta\eta\text{fix}}$*

$LC_{\beta\eta}$  = initial model of  $(\Sigma_{LC\beta\eta}, E_{LC\beta\eta})$

*$\lambda$ -calculus modulo  $\beta\eta$*

monad morphism

**Definition of a translation**  $f : LC_{\beta\eta\text{fix}} \rightarrow LC_{\beta\eta}$  **s.t.**

$$f(u) = "u[ \text{fix}(t) \mapsto \text{app}(\text{Y}, \text{abs}(t)) ]"$$

a chosen fixpoint combinator

# Example: Translating $\lambda$ -calculus with fixpoint

$LC_{\beta\eta\text{fix}}$  = initial model of  $(\Sigma, E) = (\Sigma_{LC\beta\eta} + \Sigma_{\text{fix}}, E_{LC\beta\eta} \cup E_{\text{fix}})$

*$\lambda$ -calculus modulo  $\beta\eta$  with a fixpoint operator  $\text{fix} : LC_{\beta\eta\text{fix}}' \rightarrow LC_{\beta\eta\text{fix}}$*

$LC_{\beta\eta}$  = initial model of  $(\Sigma_{LC\beta\eta}, E_{LC\beta\eta})$

*$\lambda$ -calculus modulo  $\beta\eta$*

monad morphism

**Definition of a translation**  $f : LC_{\beta\eta\text{fix}} \rightarrow LC_{\beta\eta}$  **s.t.**

$$f(u) = "u[ \text{fix}(t) \mapsto \text{app}(\text{Y}, \text{abs}(t)) ]"$$

a chosen fixpoint combinator

$\Rightarrow$  make  $LC_{\beta\eta}$  a model of  $(\Sigma, E)$ :

$$\text{app} : LC_{\beta\eta} \times LC_{\beta\eta} \rightarrow LC_{\beta\eta}$$

$$\text{abs} : LC_{\beta\eta}' \rightarrow LC_{\beta\eta}$$

$$\hat{Y} : LC_{\beta\eta}' \rightarrow LC_{\beta\eta}$$

$$t \mapsto \text{app}(\text{Y}, \text{abs}(t))$$

# Example: Translating $\lambda$ -calculus with fixpoint

$LC_{\beta\eta\text{fix}}$  = initial model of  $(\Sigma, E) = (\Sigma_{LC\beta\eta} + \Sigma_{\text{fix}}, E_{LC\beta\eta} \cup E_{\text{fix}})$

*$\lambda$ -calculus modulo  $\beta\eta$  with a fixpoint operator  $\text{fix} : LC_{\beta\eta\text{fix}}' \rightarrow LC_{\beta\eta\text{fix}}$*

$LC_{\beta\eta}$  = initial model of  $(\Sigma_{LC\beta\eta}, E_{LC\beta\eta})$

*$\lambda$ -calculus modulo  $\beta\eta$*

monad morphism

**Definition of a translation**  $f : LC_{\beta\eta\text{fix}} \rightarrow LC_{\beta\eta}$  **s.t.**

$$f(u) = "u[ \text{fix}(t) \mapsto \text{app}(Y, \text{abs}(t)) ]"$$

a chosen fixpoint combinator

$\Rightarrow$  make  $LC_{\beta\eta}$  a model of  $(\Sigma, E)$ :

$$\text{app} : LC_{\beta\eta} \times LC_{\beta\eta} \rightarrow LC_{\beta\eta}$$

$$\text{abs} : LC_{\beta\eta}' \rightarrow LC_{\beta\eta}$$

$$\hat{Y} : LC_{\beta\eta}' \rightarrow LC_{\beta\eta}$$

$$t \mapsto \text{app}(Y, \text{abs}(t))$$

Initiality of  $LC_{\beta\eta\text{fix}} \Rightarrow f : LC_{\beta\eta\text{fix}} \rightarrow LC_{\beta\eta}$

# Example: Computing the size of a term

LC = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

**Definition of a (monad) morphism  $s : LC \rightarrow \mathbb{N}$  s.t.**

$$s(\text{app}(t, u)) = 1 + s(t) + s(u)$$

$$s(\text{abs}(t)) = 1 + s(t)$$



# Example: Computing the size of a term

LC = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

**Definition of a ~~(monad)~~ morphism  $s : LC \rightarrow \mathbb{N}$  s.t.**

$$s(\text{app}(t, u)) = 1 + s(t) + s(u)$$

$$s(\text{abs}(t)) = 1 + s(t)$$



$\mathbb{N}$  is not a monad !

# Example: Computing the size of a term

LC = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

**Definition of a ~~(monad)~~ morphism  $s : LC \rightarrow \mathbb{N}$  s.t.**

$$s(\text{app}(t, u)) = 1 + s(t) + s(u)$$

$$s(\text{abs}(t)) = 1 + s(t)$$



$\mathbb{N}$  is not a monad !

**Solution:** replace  $\mathbb{N}$  with the continuation monad  $C(X) = \mathbb{N}^{\mathbb{N}^X}$

Then, give the relevant ([CSL AHLM 2010]) morphism  $\Sigma_{LC}(C) \rightarrow C$

Initiality of LC  $\Rightarrow f : LC \rightarrow C$

# Example: Computing the size of a term

LC = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

**Definition of a ~~(monad)~~ morphism  $s : LC \rightarrow \mathbb{N}$  s.t.**

$$s(\text{app}(t, u)) = 1 + s(t) + s(u)$$

$$s(\text{abs}(t)) = 1 + s(t)$$



**$\mathbb{N}$  is not a monad !**

**Solution:** replace  $\mathbb{N}$  with the continuation monad  $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

Then, give the relevant ([CSL AHLM 2010]) morphism  $\Sigma_{LC}(C) \rightarrow C$

Initiality of LC  $\Rightarrow f : LC \rightarrow C$  affects an arbitrary size to each variable

**Intuition:** uncurrying  $f_X : LC(X) \rightarrow \mathbb{N}^{(\mathbb{N}^X)}$  yields  $g : LC(X) \times \mathbb{N}^X \rightarrow \mathbb{N}$

# Example: Computing the size of a term

LC = initial model of  $(\Sigma_{LC}, \emptyset)$

$$\Sigma_{LC}(R) = R \times R + R'$$

**Definition of a ~~(monad)~~ morphism  $s : LC \rightarrow \mathbb{N}$  s.t.**

$$s(\text{app}(t, u)) = 1 + s(t) + s(u)$$

$$s(\text{abs}(t)) = 1 + s(t)$$



**$\mathbb{N}$  is not a monad !**

**Solution:** replace  $\mathbb{N}$  with the continuation monad  $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

Then, give the relevant ([CSL AHLM 2010]) morphism  $\Sigma_{LC}(C) \rightarrow C$

Initiality of LC  $\Rightarrow f : LC \rightarrow C$  affects an arbitrary size to each variable

**Intuition:** uncurrying  $f_x : LC(X) \rightarrow \mathbb{N}^{(\mathbb{N}^X)}$  yields  $g : LC(X) \times \mathbb{N}^X \rightarrow \mathbb{N}$

$$s(t) = g(t, (x \mapsto 0))$$

# Conclusion

## **Summary of the talk:**

- presented a notion of 1-signature and models
- defined a 2-signature as a 1-signature and a set of equations
- identified a class of 2-signatures that generate a syntax

The main theorem has been formalized in Coq using the UniMath library.

## **Future work:**

- add the notion of reductions;
- extend our framework to simply typed syntaxes.

# Conclusion

## **Summary of the talk:**

- presented a notion of 1-signature and models
- defined a 2-signature as a 1-signature and a set of equations
- identified a class of 2-signatures that generate a syntax

The main theorem has been formalized in Coq using the UniMath library.

## **Future work:**

- add the notion of reductions;
- extend our framework to simply typed syntaxes.

Thank you!

# Garbage

Syntax of the <i>differentiable</i> $\lambda$ -calculus		Module M over a monad B
	Monad morphism $B: \mathbf{Set} \rightarrow \mathbf{Set} \quad B \rightarrow C$	B-Module morphism $M: \mathbf{Set} \rightarrow \mathbf{Set} \quad M \rightarrow N$
Simple terms	$s, t \in \Lambda$ $(m_X: B(X) \rightarrow C(X))_X$	$(m_X: M(X) \rightarrow N(X))_X$
Variables	Corresponding 1-signature	
Variables	Review: Binding signatures and their models	
	$s, t ::= x$ $m(\text{var}^B(x)) = \text{var}^C(x)$	
	$\lambda x. t$ $R \mapsto R'$	
Substitution	$\forall f: X \rightarrow B(Y)$ $m(t[x \mapsto f(x)]^B)$	$\forall f: X \rightarrow B(Y)$ $m(t[x \mapsto f(x)]^M) =$
Substitution	$m(t)[x \mapsto m(f(x))]^C$ $R \mapsto R \times R$	$m(t)[x \mapsto f(x)]^N =$
3. Equations		
Substitution		
4. Recursion laws		
5. (Bonus) differential $\lambda$ -calculus		

# Garbage

Syntax of the <i>differentiable</i> $\lambda$ -calculus		Module M over a monad B
	Monad morphism $B \rightarrow C$ $B : \text{Set} \rightarrow \text{Set}$	B-Module morphism $M \rightarrow N$ $M : \text{Set} \rightarrow \text{Set}$
Simple terms	$s, t \in \Lambda$ $(m_X : B(X) \rightarrow C(X))_X$	$(m_X : M(X) \rightarrow N(X))_X$
Variables	$\lambda x. t$	
Substitution	$m(t[x \mapsto f(x)]^B)$	$m(t[x \mapsto f(x)]^M) =$
Equations	$m(t)[x \mapsto m(f(x))]^C$	$m(t)[x \mapsto f(x)]^N$
Substitution laws		
5. (Bonus) differential $\lambda$ -calculus		



# Garbage

Syntax of the <i>differentiable</i> $\lambda$ -calculus		Module M over a monad B
	Monad morphism $B : \text{Set} \rightarrow \text{Set} \quad B \rightarrow C$	B-Module morphism $M : \text{Set} \rightarrow \text{Set} \quad M \rightarrow N$
Simple terms	$s, t \in \Lambda$ $(m_X : B(X) \rightarrow C(X))_X$ $(\text{var}_X : X \rightarrow B(X))_X$	$(m_X : M(X) \rightarrow N(X))_X$
Variables	1. Review: Binding signatures and their models	
	$m(\text{var}^B(x)) = \text{var}^C(x)$ $\forall \lambda x. X \rightarrow B(Y), \quad R \mapsto R'$	$\forall u : X \rightarrow B(Y),$
Substitution	2. 1-Signatures and models based on monads and modules	
	$\forall f : X \rightarrow B(Y), \text{bind}_u : B(X) \rightarrow B(Y)$ $m(t[x \mapsto f(x)])^B [R \mapsto u(x)]^B$	$\forall f : X \rightarrow B(Y), \text{bind}_u : M(X) \rightarrow M(Y)$ $m(t[x \mapsto f(x)])^M [u(x)]^M$
3. Equations	$m(t)[x \mapsto m(f(R))]^C [R \times R]$	$m(t)[x \mapsto f(x)]^N$
Substitution		
4. Recursion laws		
5. (Bonus) differential $\lambda$ -calculus		

# Garbage

Syntax of the <b>differentiable <math>\lambda</math>-calculus</b>	Module M over a monad B
	$\text{Monad morphism } B \rightarrow C$ $B : \text{Set} \rightarrow \text{Set}$ $B\text{-Module morphism } M \rightarrow N$ $M : \text{Set} \rightarrow \text{Set}$
Simple terms $s, t \in \Lambda$	Corresponding 1-signature $(m_X : B(X) \rightarrow C(X))_X$ $(m_X : M(X) \rightarrow N(X))_X$
Variables $s, t ::= \lambda x. t$	Review: Binding signatures and their models $m(\text{var}^B(x)) = \text{var}^C(x)$ $\forall \lambda x. X \rightarrow B(Y), R \mapsto R'$ $\forall u : X \rightarrow B(Y),$
Substitution	$\forall f : X \rightarrow B(Y)$ $\text{bind}_u : B(X) \rightarrow B(Y)$ $m(t[x \mapsto f(x)]^B) = m(t)[x \mapsto u(x)]^B$ $m(t[x \mapsto f(x)]^M) = m(t)[x \mapsto u(x)]^M$
3. Equations	$m(t)[x \mapsto m(f(R))]^C = m(t)[x \mapsto f(x)]^N$ $\text{var}(y)[x \mapsto u(x)]^B = u(y)$
Substitution	
4. Recursion laws	$t[x \mapsto \text{var}(x)]^B = t$ $t[x \mapsto f(x)]^B[y \mapsto g(y)]^B =$ $t[x \mapsto f(x)][y \mapsto g(y)]^B$
5. (Bonus) differential $\lambda$ -calculus	

# Garbage

Syntax of the <b>differentiable <math>\lambda</math>-calculus</b>		Module M over a monad B
	Monad morphism $m : B \rightarrow C$	B-Module morphism $M \rightarrow N$
Simple terms	$s, t \in \Lambda$ $(m_X : B(X) \rightarrow C(X))_X$	$(m_X : M(X) \rightarrow N(X))_X$
Variables	$\lambda x. t$	$\lambda x. t$
Substitution	$m(t[x \mapsto f(x)])^B = m(t)[x \mapsto m(f(x))]^C$	$m(t[x \mapsto f(x)])^M = m(t)[x \mapsto m(f(x))]^N$
Equations	$m(t)[x \mapsto m(f(x))]^C = m(t)[x \mapsto f(x)]^B$	$m(t)[x \mapsto f(x)]^N = m(t)[x \mapsto m(f(x))]^M$
Recursion laws	$t[x \mapsto f(x)]^B[y \mapsto g(y)]^B = t[x \mapsto f(x)]^B[y \mapsto g(y)]^B$	$t[x \mapsto f(x)]^M[y \mapsto g(y)]^M = t[x \mapsto f(x)]^M[y \mapsto g(y)]^M$
<b>5. (Bonus) differential <math>\lambda</math>-calculus</b>		

# Copie de Algebraic 1-signature for LCD

# Copie de Algebraic 1-signature for LCD

# Module morphism VS monad morphism

# Modules VS Monads

# Modules VS Monads



# Modules VS Monads

# Modules VS Monads

# Modules VS Monads

# Table of contents