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

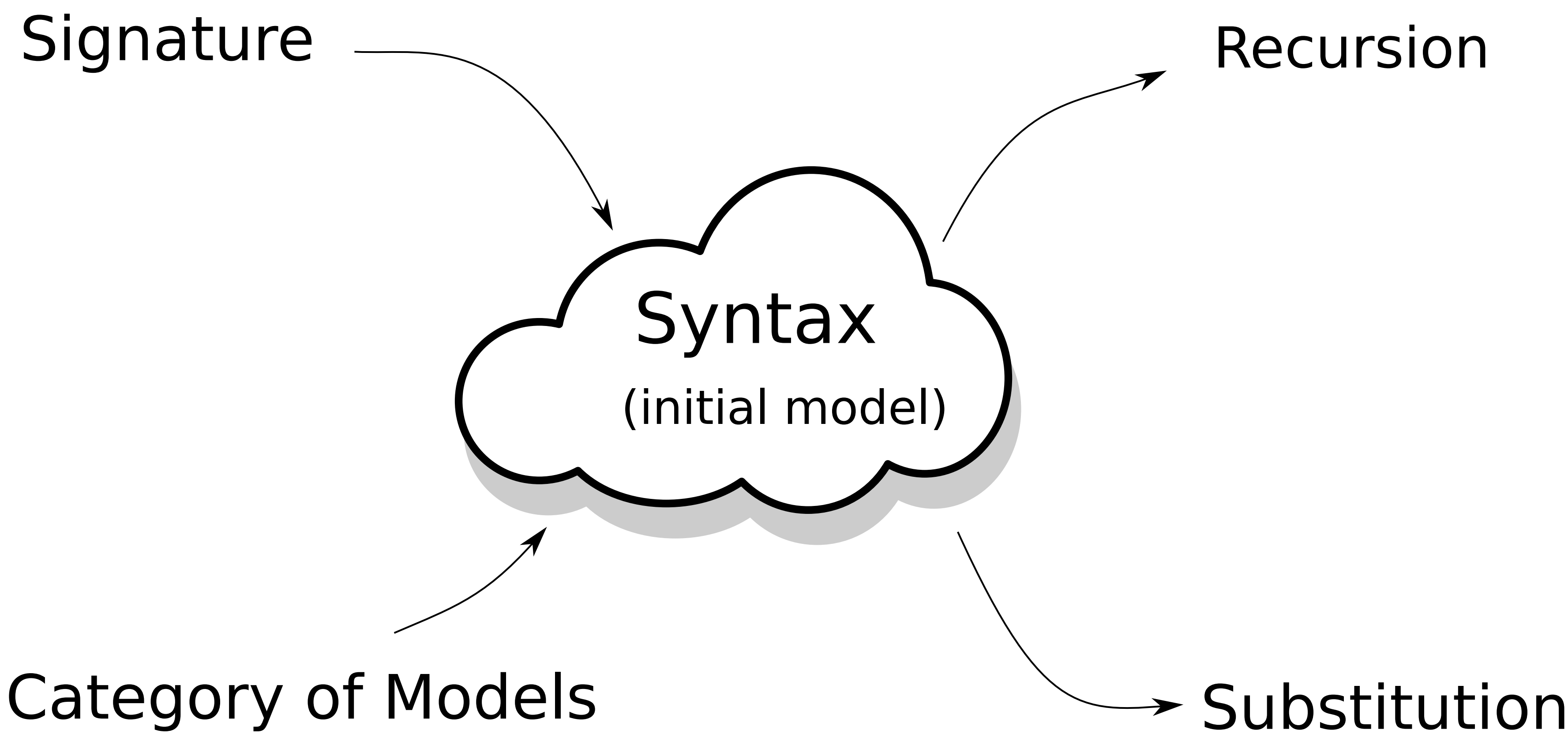
Syntax

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 λ-calculus*:

Simple terms $\mathrm{s},\mathrm{t} \in \Lambda$

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \text{(variable)} \\
 & \mid & \lambda\mathrm{x}.\mathrm{t} & \text{(modulo } \alpha\text{-renaming of x)} \\
 & \mid & \mathrm{s}\ \mathrm{t} & \\
 & \mid & \mathrm{D}\mathrm{s} \cdot \mathrm{t} & \\
 & \mid & \mathrm{s} + \mathrm{t} & \text{(modulo associativity and commutativity)} \\
 & \mid & 0 & \text{(neutral element for } +\text{)}
\end{array}
$$

Our target: LCD

Syntax of the *differentiable λ-calculus*:

Simple terms $\mathrm{s},\mathrm{t} \in \Lambda$

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \text{(variable)} \\
& \mid & \lambda \mathrm{x}.\mathrm{t} & \text{(modulo } \alpha\text{-renaming of x)} \\
& \mid & \mathrm{s}\ \mathrm{t} & \\
& \mid & \mathrm{D}\mathrm{s} \cdot \mathrm{t} & \\
& \mid & \mathrm{s} + \mathrm{t} & \text{(modulo associativity and commutativity)} \\
& \mid & 0 & \text{(neutral element for } +\text{)}
\end{array}
$$

subject to the following equation:

$$\mathrm{D}(\mathrm{D}\mathrm{s} \cdot \mathrm{t}) \cdot \mathrm{u} = \mathrm{D}(\mathrm{D}\mathrm{s} \cdot \mathrm{u}) \cdot \mathrm{t}$$

Our target: LCD

Syntax of the *differentiable λ-calculus*:

Simple terms $\mathrm{s},\mathrm{t} \in \Lambda$

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \text{(variable)} \\
& | & \lambda \mathrm{x}.\mathrm{t} & \text{(modulo } \alpha\text{-renaming of x)} \\
& | & \mathrm{s}\ \mathrm{t} & \\
& | & \mathrm{D}\mathrm{s} \cdot \mathrm{t} & \\
& | & \mathrm{s} + \mathrm{t} & \text{(modulo associativity and commutativity)} \\
& | & 0 & \text{(neutral element for } +\text{)}
\end{array}
$$

subject to the following equation:

$$\mathrm{D}(\mathrm{D}\mathrm{s} \cdot \mathrm{t}) \cdot \mathrm{u} = \mathrm{D}(\mathrm{D}\mathrm{s} \cdot \mathrm{u}) \cdot \mathrm{t}$$

and (bi)linearity of constructors with respect to +:

$$\lambda \mathrm{x}.(\mathrm{s}+\mathrm{t}) = \lambda \mathrm{x}.\mathrm{s} + \lambda \mathrm{x}.\mathrm{t} \qquad \ldots$$

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

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

- Fixed point operation

Some examples covered by our result

Operations:

- Commutative binary operation

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

- Fixed point operation

More extensive examples (set of operations with equations):

- λ-calculus modulo βη

- differential λ-calculus

Table of contents

Table of contents

Categorical formulation of a term language

Example: syntax with a binary operation ★, a constant 0, and variables

$$\begin{aligned} \mathrm{expr} ::= {} & x && \textit{(variable)} \\ \mid {} & t_1 \star t_2 && \textit{(binary operation)} \\ \mid {} & 0 && \textit{(constant)} \end{aligned}$$

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

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

For example:

$$B(\emptyset) = \{0, 0 \star 0, \dots\}$$

$$B(\{x, y\}) = \{0, 0 \star 0, \dots, x, y, x \star y, \dots\}$$

Categorical formulation of a term language

Then we have:

$$\begin{aligned} \bigstar : \; & B \times B \xrightarrow{\cdot} B \\ 0 : \; & \quad 1 \quad \xrightarrow{\cdot} B \\ \mathrm{var} : \; & \quad \mathrm{Id}_{\mathrm{Set}} \xrightarrow{\cdot} B \end{aligned}$$

Putting all together:

$$B \times B + 1 + \mathrm{Id}_{\mathrm{Set}} \xrightarrow{\cdot} B$$

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

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, ★:

$$\Big(\underbrace{()}_{0}, \underbrace{(0, 0)}_{\star}\Big)$$

Lambda calculus:

$$\Big(\underbrace{(0\ ,\ 0)}_{app}\ ,\ \underbrace{(1)}_{abs}\Big)$$

Initial semantics for binding signatures

Reminder

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

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

More generally, any binding signature gives rise to an endofunctor Σ.

Definition

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

Classical Theorem

The initial $(\Sigma + \mathrm{Id}_{\mathrm{Set}})$-algebra of a binding signature Σ 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

The Big Picture of 1-signatures and models

Binding signatures $\hookrightarrow$ Our 1-signatures

A **1-signature** Σ is a functorial assignment:

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

A **model of Σ** is a pair:

$$(R, \quad \rho : \Sigma(R) \to R)$$

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** Σ is a functorial assignment:

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

monad

A **model of Σ** is a pair:

$$(R, \quad \rho : \Sigma(R) \to R)$$

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** Σ is a functorial assignment:

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

monad

module over R

A **model of $\boldsymbol{\Sigma}$** is a pair:

$$(R, \quad \rho : \Sigma(R) \to R)$$

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** Σ is a functorial assignment:

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

monad

module over R

A **model of $\boldsymbol{\Sigma}$** is a pair:

$$(R, \quad \rho : \Sigma(R) \to R)$$

monad

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** Σ is a functorial assignment:

$$\underset{\text{monad}}{R} \mapsto \underbrace{\Sigma(R)}_{\text{module over } R}$$

A **model of Σ** is a pair:

$$(\underset{\text{monad}}{R}, \quad \underset{\text{module morphism}}{\rho} : \Sigma(R) \to R)$$

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, ★ and variables taken in X
- Variables induce a natural transformation $\mathrm{var} : \mathrm{Id}_{\mathrm{Set}} \to B$

Substitution:

$$\mathrm{bind} : B(X) \to (X \to B(Y)) \to B(Y)$$

+ laws

A triple (B, var, bind) is called a **monad**.

monad morphism = mapping preserving var and bind.

Monads

1. $\mathrm{B} : \mathrm{Set} \to \mathrm{Set}$

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

2. A collection of functions $(\mathrm{var}_\mathrm{X} : X \to B(X))_X$

   Variables are expressions

3. For each function $\mathrm{u} : \mathrm{X} \to \mathrm{B(Y)}$, a function $\mathrm{bind_u} : \mathrm{B(X)} \to \mathrm{B(Y)}$

   Parallel substitution

   Notation: $\mathbf{bind_u(t) = t[x \mapsto u(x)]}$

4. Monadic laws:

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

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

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

Preview: Operations are module morphisms

$\star$ 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 $\boldsymbol{B}$**

$\star$ commutes with substitution $\rightsquigarrow$ $\star : B \times B \to B$ is a **module morphism**

Modules VS Monads

Monad

1. $\mathrm{B} : \mathrm{Set} \to \mathrm{Set}$

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

2. A collection of functions $(\mathrm{var}_{\mathrm{X}} : X \to B(X))_X$

   Variables are expressions

3. For each function $\mathrm{u} : \mathrm{X} \to \mathrm{B(Y)}$, a function $\mathrm{bind}_{\mathrm{u}} : \mathrm{B(X)} \to \mathrm{B(Y)}$

   Parallel substitution

   Notation: $\mathbf{bind_u(t)} = \mathbf{t[x \mapsto u(x)]^B}$

4. Substitution laws:

$$\mathrm{var(y)[x \mapsto u(x)]^B = u(y)}$$
$$\mathrm{t[x \mapsto var(x)]^B = t}$$
$$\mathrm{t[x \mapsto f(x)]^B[y \mapsto g(y)]^B = t[x \mapsto f(x)[y \mapsto g(y)]^B\,]^B}$$

Modules VS Monads

~~Monad~~ **Module over a monad** B (e.g. $\mathrm{B} \times \mathrm{B}, \mathbf{2}, \ldots$)

1. $\mathrm{M} : \mathrm{Set} \to \mathrm{Set}$

   $M(X)$ = expressions taking variables in X

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

3. For each function $\mathrm{u} : \mathrm{X} \to \mathrm{B}(\mathrm{Y})$, a function $\mathrm{bind}_\mathrm{u} : \mathrm{M}(\mathrm{X}) \to \mathrm{M}(\mathrm{Y})$

   Parallel substitution

   Notation: $\mathbf{bind_u(t)} = \mathbf{t[x \mapsto u(x)]^M}$

4. Substitution laws:

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

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

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

Building blocks for binding signatures

Essential constructions of **modules over a monad $\boldsymbol{R}$**:

- R itself
- M x N for any modules M and N (in particular, R x R)
- The **derivative of a module $\boldsymbol{M}$** 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.

$$id_M : M \to M$$

$$0 : 1 \to B$$

$$\star : B \times B \to B$$

$$app : \Lambda \times \Lambda \to \Lambda$$

$$abs : \Lambda' \to \Lambda$$

The Big Picture again

A **1-signature** Σ is a functorial assignment:

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

monad module over R

A **model of Σ** is a pair:

$$(R, \quad \rho : \Sigma(R) \to R)$$

monad module morphism

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

$$\begin{tikzcd} \Sigma(R) \arrow[r, "\rho"] \arrow[d, "\Sigma(m)"'] & R \arrow[d, "m"] \\ \Sigma(S) \arrow[r, "\sigma"'] & S \end{tikzcd}$$

Syntax

Definition

Given a 1-signature Σ, 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 $R \mapsto \mathscr{P} \circ R$

powerset endofunctor on $\mathbb{S}\mathrm{et}$

Examples of 1-signatures generating syntax

- **(0,★) language**:

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

  Model: $(R\ ,\quad 0 : 1 \to R\ ,\quad \bigstar : R \times R \to R)$

  Syntax: $(B\ ,\quad 0 : 1 \to B\ ,\quad \bigstar : B \times B \to B)$

- **lambda calculus**:

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

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

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

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:

$$\text{Binding signatures} \hookrightarrow \text{Our 1-signatures}$$

Question: Can we enforce some equations in the syntax ?
For example: lambda calculus modulo beta and eta.

Table of contents

Example: a commutative binary operation

Specification of a binary operation

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

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

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: $(R\ ,\ + : R \times R \to R)$ 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: $(R\,,\ + : R \times R \to R)$ 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:

$$\begin{tikzcd} \mathrm{R} \times \mathrm{R} \arrow[rr, bend left, "+"] \arrow[r, bend right, "\text{swap}"'] & \mathrm{R} \times \mathrm{R} \arrow[r, bend right, "+"'] & \mathrm{R} \end{tikzcd}$$

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. λ-calculus modulo $\beta\eta$).

Equations

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

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

model of Σ

parallel pair of module morphisms over R

A **2-signature** is a pair

$$(\Sigma, \mathrm{E})$$

1-signature

set of Σ-equations

***model* of a 2-signature $(\boldsymbol{\Sigma}, \mathbf{E})$**:

- a model R of Σ
- s.t. ∀ (A $\rightrightarrows$ B) $\in$ E, the two morphisms $A(R) \rightrightarrows \mathrm{B}(\mathrm{R})$ are equal

Algebraic 2-signatures

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

1. A "preserves pointwise epimorphisms"

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

2. B is of the form R ↦ R'$^{\cdots}$' (e.g. R ↦ R)

Algebraic 2-signature:

$$(\Sigma, \mathrm{E})$$

algebraic 1-signature

set of **elementary**

Σ-equations

Theorem

Syntax exists for any algebraic 2-signature

Example: λ-calculus modulo βη

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

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

model of $\boldsymbol{\Sigma_{LC}}$ = monad R with module morphisms:

$$\mathrm{app} : R \times R \to R \qquad \mathrm{abs} : R' \to R$$

β-equation: $(\lambda x.t)\ u = \underbrace{t[x \mapsto u]}_{\sigma_R(t,u)}$

η-equation: $t = \lambda x.(t\ x)$

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

Example: λ-calculus modulo βη

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

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

model of Σ_{LC} = monad R with module morphisms:

$$\mathrm{app} : \mathrm{R} \times \mathrm{R} \to \mathrm{R} \qquad \mathrm{abs} : \mathrm{R}' \to \mathrm{R}$$

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

$$\begin{tikzcd}
R' \times R \arrow[rr, bend left, "\sigma_{\mathrm{R}}"] \arrow[r, bend right, "\mathrm{abs} \times R"'] & R \times R \arrow[r, bend right, "\mathrm{app}"'] & R
\end{tikzcd}
\qquad
\begin{tikzcd}
R \arrow[rr, bend left, "\mathrm{id}_{\mathrm{R}}"] \arrow[r, bend right, "\mathrm{R}\iota_1"'] & R' \arrow[r, bend right, "\mathrm{abs}"'] & R
\end{tikzcd}$$

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

Example: fixpoint operator

Definition [AHLM CSL 2018]

A **fixpoint operator** in a monad R is a module morphism $\mathrm{f} : \mathrm{R}' \to \mathrm{R}$ s.t.

$$(1) \qquad \begin{tikzcd} R' \arrow[rr, bend left, "\mathrm{f}"] \arrow[r, "(\mathrm{id}_{\mathrm{R}'}{,}\mathrm{f})"'] & R' \times R \arrow[r, "\sigma_{\mathrm{R}}"'] & R \end{tikzcd} \qquad \text{commutes.}$$

The algebraic 2-signature $(\Sigma_{\mathrm{fix}}, \mathrm{E}_{\mathrm{fix}})$ of a fixpoint operator:

$$\Sigma_{\mathrm{fix}}\,(\mathrm{R}) := \mathrm{R}' \qquad\qquad \mathrm{E}_{\mathrm{fix}} = \{\ (1)\ \}$$

Proposition [AHLM CSL 2018]

Fixpoint operators in $\mathrm{LC}_{\beta\eta}$ are in one to one correspondance with fixpoint combinators (i.e. λ-terms Y s.t. $t\,(Y\,t) = Y\,t$ for any t).

Combining algebraic 2-signatures

Algebraic 2-signatures can be combined:

fixpoint operator

λ-calculus modulo βη

$$(\Sigma_{\mathrm{fix}}, \mathrm{E}_{\mathrm{fix}}) \quad + \quad (\Sigma_{\mathrm{LC\beta\eta}}, \mathrm{E}_{\mathrm{LC\beta\eta}})$$

$$=$$

$$(\Sigma_{\mathrm{fix}} + \Sigma_{\mathrm{LC\beta\eta}} \quad , \quad \mathrm{E}_{\mathrm{fix}} \cup \mathrm{E}_{\mathrm{LC\beta\eta}})$$

λ-calculus modulo βη with an explicit fixpoint operator

Example: free monoid

An algebraic 2-signature $(\Sigma_{\mathrm{mon}}, \mathrm{E}_{\mathrm{mon}})$ for the free monoid monad $\mathrm{X} \mapsto \coprod_{\mathrm{n}} \mathrm{X}^{\mathrm{n}}$

$$\boldsymbol{\Sigma}_{\mathrm{mon}}(\mathrm{R}) := 1 + \mathrm{R} \times \mathrm{R}$$

model of Σ = monad R with module morphisms:

$$\varepsilon : 1 \to \mathrm{R} \qquad \mathrm{m} : \mathrm{R} \times \mathrm{R} \to \mathrm{R}$$

3 elementary Σ-equations:

$$\begin{tikzcd} & R^2 \arrow[dr, bend left, "m"] & \\ R^3 \arrow[ur, bend left, "R\times m"] \arrow[dr, bend right, "m\times R"'] & & R \\ & R^2 \arrow[ur, bend right, "m"'] & \end{tikzcd} \qquad \begin{tikzcd} & R^2 \arrow[dr, bend left, "m"] & \\ R \arrow[ur, bend left, "\epsilon, id"] \arrow[rr, bend right, "id"'] & & R \end{tikzcd} \qquad \begin{tikzcd} & R^2 \arrow[dr, bend left, "m"] & \\ R \arrow[ur, bend left, "id, \epsilon"] \arrow[rr, bend right, "id"'] & & R \end{tikzcd}$$

associativity — left unit — right unit

Our target: LCD

Syntax of the *differentiable λ-calculus*:

Simple terms $\mathrm{s,t} \in \Lambda$

$$\left.\begin{array}{rcl} s,t & ::= & \mathrm{x} \\ & | & \lambda \mathrm{x.t} \\ & | & \mathrm{s\ t} \end{array}\right\} \text{λ-calculus}$$

$$\begin{array}{rcl} & | & \mathrm{Ds} \cdot \mathrm{t} \end{array}$$

$$\left.\begin{array}{rcl} & | & \mathrm{s} + \mathrm{t} \\ & | & 0 \end{array}\right\} \text{free commutative monoid}$$

subject to the following equation:

$$\mathrm{D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t}$$

and (bi)linearity of constructors with respect to +:

$$\lambda \mathrm{x.(s{+}t)} = \lambda \mathrm{x.s} + \lambda \mathrm{x.t} \qquad \ldots$$

Algebraic 1-signature for LCD

Syntax of the *differentiable λ-calculus*:

Simple terms $\mathrm{s,t} \in \Lambda$ Corresponding 1-signature

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \\
& | & \lambda \mathrm{x.t} & \Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R}' + \mathrm{R} \times \mathrm{R} \\
& | & \mathrm{s\ t} & \\
& | & \mathrm{Ds} \cdot \mathrm{t} & \mathrm{R} \mapsto \mathrm{R} \times \mathrm{R} \\
& | & \mathrm{s} + \mathrm{t} & \\
& | & 0 & \Sigma_{\mathrm{mon}}(\mathrm{R}) = 1 + \mathrm{R} \times \mathrm{R}
\end{array}
$$

Algebraic 1-signature for LCD

Syntax of the *differentiable λ-calculus*:

Simple terms $\mathrm{s,t} \in \Lambda$

Corresponding 1-signature

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \\
& | & \lambda \mathrm{x.t} & \Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R}' + \mathrm{R} \times \mathrm{R} \\
& | & \mathrm{s\ t} & \\
& | & \mathrm{Ds} \cdot \mathrm{t} & \mathrm{R} \mapsto \mathrm{R} \times \mathrm{R} \\
& | & \mathrm{s} + \mathrm{t} & \\
& | & 0 & \Sigma_{\mathrm{mon}}(\mathrm{R}) = 1 + \mathrm{R} \times \mathrm{R}
\end{array}
$$

Resulting algebraic 1-signature: $\Sigma_{\mathrm{LCD}}(\mathrm{R}) = \Sigma_{\mathrm{LC}}(\mathrm{R}) + \mathrm{R} \times \mathrm{R} + \Sigma_{\mathrm{mon}}(\mathrm{R})$

Elementary equations for LCD

Commutative monoidal structure:

$$s + t = t + s \qquad \mathrm{R}\times\mathrm{R} \rightrightarrows \mathrm{R}$$

$$\mathrm{E_{mon}} \left\{ \begin{array}{ll} s + (t + u) = (s + t) + u & \mathrm{R}\times\mathrm{R}\times\mathrm{R} \rightrightarrows \mathrm{R} \\ 0 + t = t & \mathrm{R} \rightrightarrows \mathrm{R} \\ t + 0 = t & \mathrm{R} \rightrightarrows \mathrm{R} \end{array} \right.$$

Differentiation:

$$D(Ds \cdot t) \cdot u = D(Ds \cdot u) \cdot t \qquad \mathrm{R}\times\mathrm{R}\times\mathrm{R} \rightrightarrows \mathrm{R}$$

Linearity:

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

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

$$Ds{\cdot}(t{+}u) = Ds{\cdot}t + Ds{\cdot}u \qquad \mathrm{R}\times\mathrm{R}\times\mathrm{R} \rightrightarrows \mathrm{R}$$

...

Table of contents

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

1. Give a module morphism $\mathrm{s} : \Sigma(\mathrm{S}) \to \mathrm{S}$

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

1. Give a module morphism $\mathrm{s} : \Sigma(\mathrm{S}) \to \mathrm{S}$

   $\Rightarrow$ induces a Σ-model $(\mathrm{S}\ , \mathrm{s})$

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

1. Give a module morphism $\mathrm{s} : \Sigma(\mathrm{S}) \to \mathrm{S}$

   $\Rightarrow$ induces a Σ-model $(\mathrm{S}\ , \mathrm{s})$

2. Show that all the equations in E are satisfied for this model

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

1. Give a module morphism $\mathrm{s} : \Sigma(\mathrm{S}) \to \mathrm{S}$

   $\Rightarrow$ induces a Σ-model $(\mathrm{S}\,,\mathrm{s})$

2. Show that all the equations in E are satisfied for this model

   $\Rightarrow$ induces a model of $(\Sigma\,,\mathrm{E})$

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$\mathrm{f} : \mathrm{R} \to \mathrm{S}$$

initial model of a 2-signature (Σ, E)

1. Give a module morphism $\mathrm{s} : \Sigma(\mathrm{S}) \to \mathrm{S}$

   $\Rightarrow$ induces a Σ-model $(\mathrm{S}\,,\mathrm{s})$

2. Show that all the equations in E are satisfied for this model

   $\Rightarrow$ induces a model of $(\Sigma\,,\mathrm{E})$

Initiality of R $\Rightarrow$ model morphism $\mathrm{R} \to \mathrm{S}$ $\Rightarrow$ monad morphism $\mathrm{R} \to \mathrm{S}$

Example: Computing the set of free variables

LC = initial model of $(\Sigma_{\mathrm{LC}}\, , \varnothing)$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R}'$

$\mathcal{P}$ = power set monad

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

$$\mathrm{fv}(\mathrm{app}(\mathrm{t},\mathrm{u})) = \mathrm{fv}(\mathrm{t}) \cup \mathrm{fv}(\mathrm{u}) \qquad \mathrm{fv}(\mathrm{abs}(\mathrm{t})) = \mathrm{fv}(\mathrm{t}) \setminus \{\diamond\}$$

Example: Computing the set of free variables

LC = initial model of $(\Sigma_{\mathrm{LC}}, \varnothing)$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R}'$

$\mathcal{P}$ = power set monad

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

$$\mathrm{fv}(\mathrm{app}(\mathrm{t},\mathrm{u})) = \mathrm{fv}(\mathrm{t}) \cup \mathrm{fv}(\mathrm{u}) \qquad \mathrm{fv}(\mathrm{abs}(\mathrm{t})) = \mathrm{fv}(\mathrm{t}) \setminus \{\diamond\}$$

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

$$\cup : \mathcal{P} \times \mathcal{P} \to \mathcal{P} \qquad \_\setminus\{\diamond\} : \mathcal{P}' \to \mathcal{P}$$

Example: Computing the set of free variables

LC = initial model of $(\Sigma_{\mathrm{LC}}\,, \varnothing)$ $\qquad$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R}'$

$\mathcal{P}$ = power set monad

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

$$\mathrm{fv}(\mathrm{app}(\mathrm{t},\mathrm{u})) = \mathrm{fv}(\mathrm{t}) \cup \mathrm{fv}(\mathrm{u}) \qquad \mathrm{fv}(\mathrm{abs}(\mathrm{t})) = \mathrm{fv}(\mathrm{t}) \setminus \{\diamond\}$$

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

$$\cup : \mathcal{P} \times \mathcal{P} \to \mathcal{P} \qquad \_\setminus\{\diamond\} : \mathcal{P}' \to \mathcal{P}$$

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

Example: Translating λ-calculus with fixpoint

$\mathrm{LC}_{\beta\eta\mathrm{fix}}$ = initial model of $(\Sigma_{\mathrm{LC}\beta\eta}, \mathrm{E}_{\mathrm{LC}\beta\eta}) + (\Sigma_{\mathrm{fix}}, \mathrm{E}_{\mathrm{fix}})$

λ-calculus modulo βη with a fixpoint operator $\mathrm{fix} : \mathrm{LC}_{\beta\eta\mathrm{fix}}{}' \to \mathrm{LC}_{\beta\eta\mathrm{fix}}$

$\mathrm{LC}_{\beta\eta}$ = initial model of $(\Sigma_{\mathrm{LC}\beta\eta}, \mathrm{E}_{\mathrm{LC}\beta\eta})$

λ-calculus modulo βη

monad morphism

Definition of a translation $\mathrm{f} : \mathrm{LC}_{\beta\eta\mathrm{fix}} \to \mathrm{LC}_{\beta\eta}$ **s.t.**

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

a chosen fixpoint combinator

Example: Translating λ-calculus with fixpoint

LC$_{\beta\eta\mathrm{fix}}$ = initial model of $(\Sigma_{\mathrm{LC\beta\eta}}\,,\mathrm{E}_{\mathrm{LC\beta\eta}}) + (\Sigma_{\mathrm{fix}}\,,\ \mathrm{E}_{\mathrm{fix}})$

λ-calculus modulo βη with a fixpoint operator $\mathrm{fix} : \mathrm{LC}_{\beta\eta\mathrm{fix}}{}' \to \mathrm{LC}_{\beta\eta\mathrm{fix}}$

LC$_{\beta\eta}$ = initial model of $(\Sigma_{\mathrm{LC\beta\eta}}\,,\mathrm{E}_{\mathrm{LC\beta\eta}})$

λ-calculus modulo βη

monad morphism

Definition of a translation $\mathrm{f} : \mathrm{LC}_{\beta\eta\mathrm{fix}} \to \mathrm{LC}_{\beta\eta}$ **s.t.**

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

a chosen fixpoint combinator

$\Rightarrow$ make LC$_{\beta\eta}$ a model of $(\Sigma_{\mathrm{LC\beta\eta}}\,,\mathrm{E}_{\mathrm{LC\beta\eta}}) + (\Sigma_{\mathrm{fix}}\,,\ \mathrm{E}_{\mathrm{fix}})$:

$\mathrm{app}, \mathrm{abs}$

$$\hat{\mathrm{Y}} : \mathrm{LC}_{\beta\eta}{}' \to \mathrm{LC}_{\beta\eta}$$
$$\mathrm{t} \mapsto \mathrm{app}(\mathrm{Y},\mathrm{abs}(\mathrm{t}))$$

Example: Translating λ-calculus with fixpoint

LC$_{\beta\eta\mathrm{fix}}$ = initial model of $(\Sigma_{\mathrm{LC}\beta\eta}\,, \mathrm{E}_{\mathrm{LC}\beta\eta}) + (\Sigma_{\mathrm{fix}}\,,\ \mathrm{E}_{\mathrm{fix}})$

λ-calculus modulo βη with a fixpoint operator $\mathrm{fix} : \mathrm{LC}_{\beta\eta\mathrm{fix}}{}' \to \mathrm{LC}_{\beta\eta\mathrm{fix}}$

LC$_{\beta\eta}$ = initial model of $(\Sigma_{\mathrm{LC}\beta\eta}\,, \mathrm{E}_{\mathrm{LC}\beta\eta})$

λ-calculus modulo βη

monad morphism

Definition of a translation $\mathrm{f} : \mathrm{LC}_{\beta\eta\mathrm{fix}} \to \mathrm{LC}_{\beta\eta}$ **s.t.**

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

a chosen fixpoint combinator

$\Rightarrow$ make LC$_{\beta\eta}$ a model of $(\Sigma_{\mathrm{LC}\beta\eta}\,, \mathrm{E}_{\mathrm{LC}\beta\eta}) + (\Sigma_{\mathrm{fix}}\,,\ \mathrm{E}_{\mathrm{fix}})$:

app, abs

$\hat{\mathrm{Y}} : \mathrm{LC}_{\beta\eta}{}' \to \mathrm{LC}_{\beta\eta}$

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

Initiality of $\mathrm{LC}_{\beta\eta\mathrm{fix}} \ \Rightarrow\ \mathrm{f} : \mathrm{LC}_{\beta\eta\mathrm{fix}} \to \mathrm{LC}_{\beta\eta}$

Example: Computing the size of a term

LC = initial model of $(\Sigma_{\mathrm{LC}}, \emptyset)$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R}'$

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

$$\mathrm{s}(\mathrm{app}(\mathrm{t},\mathrm{u})) = 1 + \mathrm{s}(\mathrm{t}) + \mathrm{s}(\mathrm{u}) \qquad \mathrm{s}(\mathrm{abs}(\mathrm{t})) = 1 + \mathrm{s}(\mathrm{t})$$

Example: Computing the size of a term

LC = initial model of $(\Sigma_{\mathrm{LC}}\,,\,\varnothing)$ $\qquad$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R}\times\mathrm{R} + \mathrm{R}'$

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

$$\mathrm{s}(\mathrm{app}(\mathrm{t},\mathrm{u})) = 1 + \mathrm{s}(\mathrm{t}) + \mathrm{s}(\mathrm{u}) \qquad \mathrm{s}(\mathrm{abs}(\mathrm{t})) = 1 + \mathrm{s}(\mathrm{t})$$

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

Example: Computing the size of a term

LC = initial model of $(\Sigma_{\mathrm{LC}}, \emptyset)$ $\quad\quad$ $\Sigma_{\mathrm{LC}}(R) = R \times R + R'$

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

$$s(\mathrm{app}(t,u)) = 1 + s(t) + s(u) \qquad\qquad s(\mathrm{abs}(t)) = 1 + s(t)$$

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

Solution [CSL AHLM 2010]:

replace $\quad s : \mathrm{LC} \to \mathbb{N}$ $\quad$ continuation monad $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

with $\quad f : \mathrm{LC} \to C$

Example: Computing the size of a term

LC = initial model of $(\Sigma_{\mathrm{LC}}, \varnothing)$ $\qquad$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R'}$

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

$$\mathrm{s}(\mathrm{app}(\mathrm{t},\mathrm{u})) = 1 + \mathrm{s}(\mathrm{t}) + \mathrm{s}(\mathrm{u}) \qquad \mathrm{s}(\mathrm{abs}(\mathrm{t})) = 1 + \mathrm{s}(\mathrm{t})$$

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

Solution [CSL AHLM 2010]:

replace $\mathrm{s} : \mathrm{LC} \to \mathbb{N}$ $\qquad$ continuation monad $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

with $\mathrm{f} : \mathrm{LC} \to \mathrm{C}$

affects an arbitrary size to each variable

Intuition: uncurrying $\mathrm{f}_X : \mathrm{LC}(X) \to \mathbb{N}^{(\mathbb{N}^X)}$ yields $\mathrm{g} : \mathrm{LC}(X) \times \mathbb{N}^X \to \mathbb{N}$

Example: Computing the size of a term

LC = initial model of $(\Sigma_{\mathrm{LC}}, \emptyset)$ $\qquad$ $\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R} \times \mathrm{R} + \mathrm{R}'$

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

$$\mathrm{s}(\mathrm{app}(\mathrm{t},\mathrm{u})) = 1 + \mathrm{s}(\mathrm{t}) + \mathrm{s}(\mathrm{u}) \qquad \mathrm{s}(\mathrm{abs}(\mathrm{t})) = 1 + \mathrm{s}(\mathrm{t})$$

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

Solution [CSL AHLM 2010]:

replace $\mathrm{s} : \mathrm{LC} \to \mathbb{N}$ $\qquad$ continuation monad C(X) = $\mathbb{N}^{(\mathbb{N}^{X})}$

with $\mathrm{f} : \mathrm{LC} \to \mathrm{C}$

affects an arbitrary size to each variable

Intuition: uncurrying $\mathrm{f}_{\mathrm{X}} : \mathrm{LC}(\mathrm{X}) \to \mathbb{N}^{(\mathbb{N}^{\mathrm{X}})}$ yields $\mathrm{g} : \mathrm{LC}(\mathrm{X}) \times \mathbb{N}^{\mathrm{X}} \to \mathbb{N}$

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

variables are of size 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!