Modular specification of monads through higher-order presentations

Ambroise Lafont

joint work with Benedikt Ahrens, André Hirschowitz, Marco Maggesi

Submitted to FSCD 2019

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

Operations covered by our result

Some examples:

- Symmetric operations

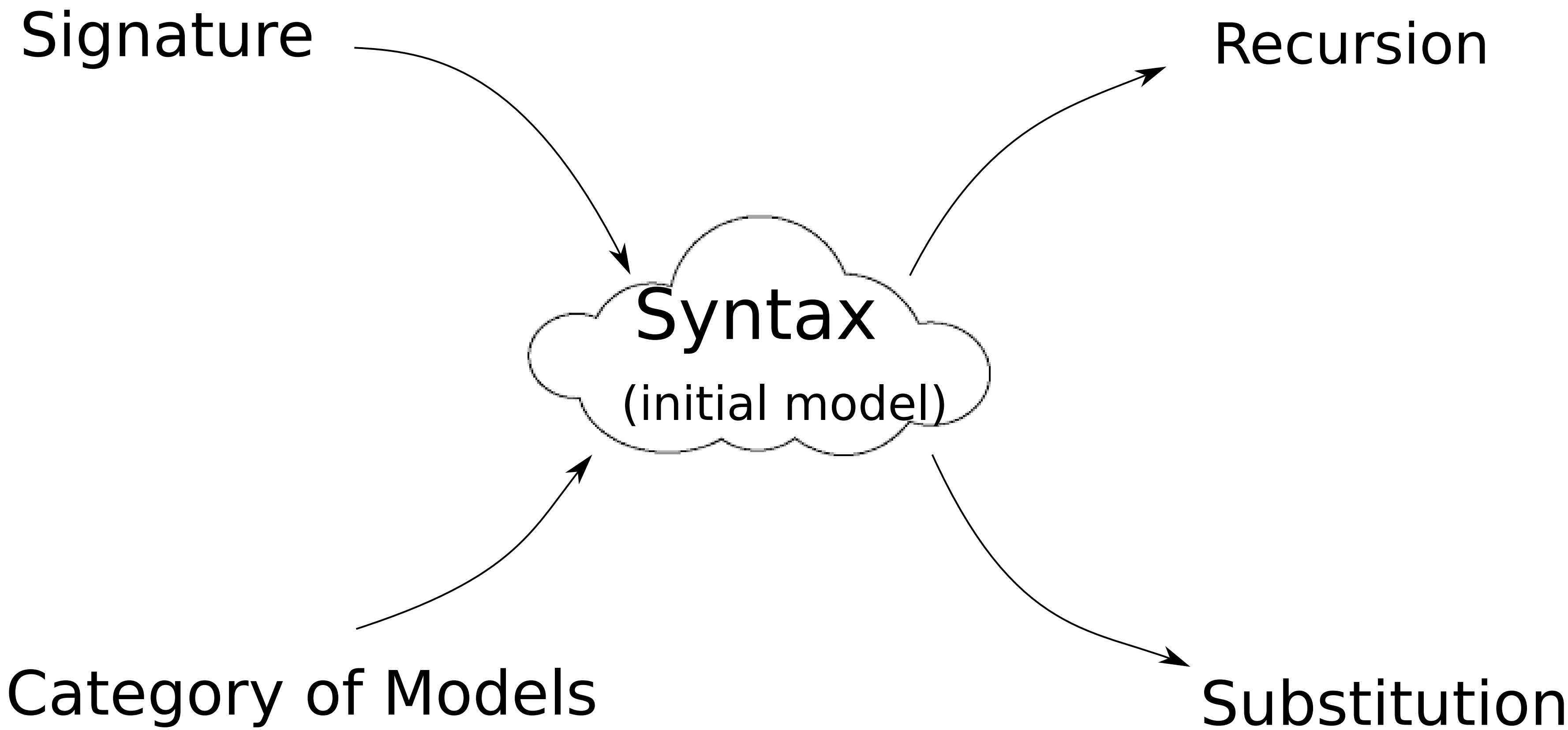
$$m : T \times T \to T \qquad \text{s.t.} \qquad m(t, u) = m(u, t)$$

- Fixed point operation

- Syntactic closure operator with coherences

- λ-calculus modulo βη

What is a syntax?



generates a syntax = existence of the initial model

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

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:

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

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, ★:

$$\big((), (0, 0)\big)$$

0 $\quad\quad\quad$ $\star$

Lambda calculus:

$$\big((0\ ,\ 0)\ ,\ (1)\big)$$

app $\quad\quad\quad$ abs

Initial semantics for binding signatures

Reminder

The syntax $(0,\star)$ 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

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** 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 Σ** 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:

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

monad

module over R

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

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

monad

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, ★ 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 B**

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

Modules VS Monads

	Monad B	Module M over a monad B
	$\mathrm{B} : \mathrm{Set} \to \mathrm{Set}$	$\mathrm{M} : \mathrm{Set} \to \mathrm{Set}$
Variables		
Substitution		
Substitution laws		

Modules VS Monads

	Monad B	Module M over a monad B
	$\mathrm{B} : \mathrm{Set} \to \mathrm{Set}$	$\mathrm{M} : \mathrm{Set} \to \mathrm{Set}$
Variables	$(\mathrm{var}_{\mathrm{X}} : X \to B(X))_X$	
Substitution		
Substitution laws		

Modules VS Monads

	Monad B	Module M over a monad B
	$\mathrm{B} : \mathrm{Set} \to \mathrm{Set}$	$\mathrm{M} : \mathrm{Set} \to \mathrm{Set}$
Variables	$(\mathrm{var}_{\mathrm{X}} : X \to B(X))_X$	
Substitution	$\forall\, \mathrm{u} : \mathrm{X} \to \mathrm{B}(\mathrm{Y}),$ $\mathrm{bind}_{\mathrm{u}} : \mathrm{B}(\mathrm{X}) \to \mathrm{B}(\mathrm{Y})$ $\mathrm{t} \mapsto \mathrm{t}[\mathrm{x} \mapsto \mathrm{u}(\mathrm{x})]^{\mathrm{B}}$	$\forall\, \mathrm{u} : \mathrm{X} \to \mathrm{B}(\mathrm{Y}),$ $\mathrm{bind}_{\mathrm{u}} : \mathrm{M}(\mathrm{X}) \to \mathrm{M}(\mathrm{Y})$ $\mathrm{t} \mapsto \mathrm{t}[\mathrm{x} \mapsto \mathrm{u}(\mathrm{x})]^{\mathrm{M}}$
Substitution laws		

Modules VS Monads

	Monad B	Module M over a monad B
	$B : \mathrm{Set} \to \mathrm{Set}$	$M : \mathrm{Set} \to \mathrm{Set}$
Variables	$(\mathrm{var}_X : X \to B(X))_X$	
Substitution	$\forall\, u : X \to B(Y),$ $\mathrm{bind}_u : B(X) \to B(Y)$ $t \mapsto t[x \mapsto u(x)]^B$	$\forall\, u : X \to B(Y),$ $\mathrm{bind}_u : M(X) \to M(Y)$ $t \mapsto t[x \mapsto u(x)]^M$
Substitution laws	$\mathrm{var}(y)[x \mapsto u(x)]^B = u(y)$	
	$t[x \mapsto \mathrm{var}(x)]^B = t$	
	$t[x \mapsto f(x)]^B[y \mapsto g(y)]^B =$ $t[x \mapsto f(x)[y \mapsto g(y)]^B\,]^B$	

Modules VS Monads

	Monad B	Module M over a monad B
	$B : \mathrm{Set} \to \mathrm{Set}$	$M : \mathrm{Set} \to \mathrm{Set}$
Variables	$(\mathrm{var}_X : X \to B(X))_X$	
Substitution	$\forall\, u : X \to B(Y)$, $\mathrm{bind}_u : B(X) \to B(Y)$ $t \mapsto t[x \mapsto u(x)]^B$	$\forall\, u : X \to B(Y)$, $\mathrm{bind}_u : M(X) \to M(Y)$ $t \mapsto t[x \mapsto u(x)]^M$
Substitution laws	$\mathrm{var}(y)[x \mapsto u(x)]^B = u(y)$	
	$t[x \mapsto \mathrm{var}(x)]^B = t$	$t[x \mapsto \mathrm{var}(x)]^M = t$
	$t[x \mapsto f(x)]^B[y \mapsto g(y)]^B = t[x \mapsto f(x)[y \mapsto g(y)]^B\,]^B$	$t[x \mapsto f(x)]^M[y \mapsto g(y)]^M = t[x \mapsto f(x)[y \mapsto g(y)]^B\,]^M$

Module morphism VS monad morphism

	Monad morphism B $\to$ C	B-Module morphism M → N
	$(\mathrm{m}_{\mathrm{X}} : B(X) \to C(X))_X$	$(\mathrm{m}_{\mathrm{X}} : M(X) \to N(X))_X$
Variables	$\mathrm{m}(\mathrm{var}^{\mathrm{B}}(\mathrm{x})) = \mathrm{var}^{\mathrm{C}}(\mathrm{x})$	
Substitution	$\forall\, \mathrm{f} : \mathrm{X} \to \mathrm{B}(\mathrm{Y}),$ $\mathrm{m}(\mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})]^{\mathrm{B}}) =$ $\mathrm{m}(\mathrm{t})[\ \mathrm{x} \mapsto \mathrm{m}(\mathrm{f}(\mathrm{x}))\ ]^{\mathrm{C}}$	$\forall\, \mathrm{f} : \mathrm{X} \to \mathrm{B}(\mathrm{Y}),$ $\mathrm{m}(\mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})]^{\mathrm{M}}) =$ $\mathrm{m}(\mathrm{t})[\ \mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\ ]^{\mathrm{N}}$

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 Set

Examples of 1-signatures generating syntax

- **(0,★) language**:

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

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

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

- **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 with a binary commutative 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 with a binary commutative operation

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

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

Example: a commutative binary operation

Specification of a commutative binary operation

1-Signature: $R \mapsto R \times R$ and parallel morphisms

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

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 with a binary commutative operation

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

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

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

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

model of $\boldsymbol{\Sigma}_{\mathbf{LC}}$ = monad R with module morphisms:

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

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

$$\begin{tikzcd}
R' \times R \arrow[r, "\mathrm{abs} \times R"'] \arrow[rr, bend left, "\sigma_R"] & R \times R \arrow[r, "\mathrm{app}"'] & R
\end{tikzcd}$$

η-equation: $t = \lambda x.(t\ x)$

$$\begin{tikzcd}
R \arrow[r, "\iota_1"'] \arrow[rr, bend left, "\mathrm{id}_R"] & R' \arrow[r, "\mathrm{abs}"'] & R
\end{tikzcd}$$

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

Example: fixpoint operator

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.

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

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

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

$$=$$

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

λ-calculus modulo βη with an explicit fixpoint operator

Example: free monoid

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

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

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:

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

associativity — left unit — right unit

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

$\mathcal{P}$ = 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}:

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

Example: Computing the set of free variables

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

$\mathcal{P}$ = 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

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

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

Example: Translating λ-calculus with fixpoint

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

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 (Σ, E):

$$\mathrm{app} : \mathrm{LC}_{\beta\eta} \times \mathrm{LC}_{\beta\eta} \to \mathrm{LC}_{\beta\eta}$$

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

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

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

Example: Translating λ-calculus with fixpoint

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

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.**

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

a chosen fixpoint combinator

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

$$\mathrm{app} : \mathrm{LC}_{\beta\eta} \times \mathrm{LC}_{\beta\eta} \to \mathrm{LC}_{\beta\eta}$$

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

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

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

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

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

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

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

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 !**

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

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

Initiality of $\mathrm{LC} \;\Rightarrow\; \mathrm{f} : \mathrm{LC} \to \mathrm{C}$

Example: Computing the size of a term

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

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

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

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

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

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

Initiality of $\mathrm{LC} \;\Rightarrow\; \mathrm{f} : \mathrm{LC} \to \mathrm{C}$

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 $\mathrm{g} : \mathrm{LC}(\mathrm{X}) \times \mathbb{N}^{\mathrm{X}} \to \mathbb{N}$

Example: Computing the size of a term

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

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

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

$\mathbb{N}$ **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 $\Sigma_{LC}(C) \to C$

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

affects an arbitrary size to each variable

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

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

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!