Higher-order Arities, Signatures and Equations via Modules

Ambroise Lafont

joint work with
Benedikt Ahrens, André Hirschowitz, Marco Maggesi

Keywords associated with syntax

Induction/Recursion

Substitution

Model

Operation/Construction

Arity/Signature

This talk: give a *discipline* for specifying syntaxes

Motivating example: dLC

syntax of dLC $=$ **differential λ-calculus** [Ehrhard-Regnier 2003].

- explicitly involves **equations**   e.g. $s+t = t+s$

- specifically taylored: (not an *instance* of a general framework/scheme)

     inductive definition of a set $+$ ad-hoc structure
                                 e.g. **unary substitution**

Our proposal $=$ a discipline for presenting **monads**:

- syntax $=$ **monad** (well behaved substitution)

- presentation of monads $\Rightarrow$ presentation of syntaxes

- [Fiore-Hure 2010]: alternative approach, for simply typed syntaxes

   $\Rightarrow$ our approach explicitly relies on monads and modules (untyped case).

Motivating example: dLC

syntax of dLC $=$ ***differential λ-calculus*** [Ehrhard-Regnier 2003].

- explicitly involves **equations**   e.g. $\mathrm{s+t=t+s}$

- specifically taylored: (not an *instance* of a general framework/scheme)

    inductive definition of a set $+$ ad-hoc structure
                                         e.g. **unary substitution**

Our proposal $=$ a discipline for presenting **monads**:

- syntax $=$ **monad** (well behaved substitution)

- presentation of monads $\Rightarrow$ presentation of syntaxes

- [Fiore-Hure 2010]: alternative approach, for simply typed syntaxes

  $\Rightarrow$ our approach explicitly relies on monads and modules (untyped case).

Syntax of dLC: [Ehrhard-Regnier 2003]

Let be given a denumerable set of variables. We define by induction on k an increasing family of sets (Δ_k). We set $\Delta_0 = \emptyset$ and Δ_{k+1} is defined as follows.

Monotonicity: if t belongs to Δ_k then t belongs to Δ_{k+1}.

Variable: if $n \in \mathbb{N}$, x is a variable, $i_1, \ldots, i_n \in \mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ and $u_1, \ldots, u_n \in \Delta_k$, then

$$D_{i_1,\ldots,i_n} x \cdot (u_1, \ldots, u_n)$$

belongs to Δ_{k+1}. This term is identified with all the terms of the shape $D_{i_{\sigma(1)},\ldots,i_{\sigma(n)}} x \cdot (u_{\sigma(1)}, \ldots, u_{\sigma(n)}) \in \Delta_{k+1}$ where σ is a permutation on $\{1, \ldots, n\}$.

Abstraction: if $n \in \mathbb{N}$, x is a variable, $u_1, \ldots, u_n \in \Delta_k$ and $t \in \Delta_k$, then

$$D_1^n \lambda x \, t \cdot (u_1, \ldots, u_n)$$

belongs to Δ_{k+1}. This term is identified with all the terms of the shape $D_1^n \lambda x \, t \cdot (u_{\sigma(1)}, \ldots, u_{\sigma(n)}) \in \Delta_{k+1}$ where σ is a permutation on $\{1, \ldots, n\}$.

Application: if $s \in \Delta_k$ and $t \in R\langle \Delta_k \rangle$, then

$$(s)t$$

belongs to Δ_{k+1}.

Setting $n = 0$ in the first two clauses, and restricting application by the constraint that $t \in \Delta_k \subseteq R\langle \Delta_k \rangle$, one retrieves the usual definition of lambda-terms which shows that differential terms are a superset of ordinary lambda-terms.

The permutative identification mentioned above will be called *equality up to differential permutation*. We also work up to α-conversion.

Syntax of dLC: [Ehrhard-Regnier 2003]

Let be given a denumerable set of variables. We define by induction on k an increasing family of sets (Δ_k). We set $\Delta_0 = \emptyset$ and Δ_{k+1} is defined as follows.

Monotonicity: if t belongs to Δ_k then t belongs to Δ_{k+1}.

Variable: if $n \in \mathbb{N}$, x is a variable, $i_1, \ldots, i_n \in \mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ and $u_1, \ldots, u_n \in \Delta_k$, then

$$D_{i_1,\ldots,i_n} x \cdot (u_1, \ldots, u_n)$$

belongs to Δ_{k+1}. This term is identified with all the terms of the shape $D_{i_{\sigma(1)},\ldots,i_{\sigma(n)}} x \cdot (u_{\sigma(1)}, \ldots, u_{\sigma(n)}) \in \Delta_{k+1}$ where σ is a permutation on $\{1, \ldots, n\}$.

Abstraction: if $n \in \mathbb{N}$, x is a variable, $u_1, \ldots, u_n \in \Delta_k$ and $t \in \Delta_k$, then

$$D_1^n \lambda x\, t \cdot (u_1, \ldots, u_n)$$

belongs to Δ_{k+1}. This term is identified with all the terms of the shape $D_1^n \lambda x\, t \cdot (u_{\sigma(1)}, \ldots, u_{\sigma(n)}) \in \Delta_{k+1}$ where σ is a permutation on $\{1, \ldots, n\}$.

Application: if $s \in \Delta_k$ and $t \in R\langle \Delta_k \rangle$, then

$$(s)t \quad \longleftarrow \quad \text{as an operation: } \Lambda \times \text{FreeCommutativeMonoid}(\Lambda) \to \Lambda$$

belongs to Δ_{k+1}.

Setting $n = 0$ in the first two clauses, and restricting application by the constraint that $t \in \Delta_k \subseteq R\langle \Delta_k \rangle$, one retrieves the usual definition of lambda-terms which shows that differential terms are a superset of ordinary lambda-terms.

The permutative identification mentioned above will be called *equality up to differential permutation*. We also work up to α-conversion.

Syntax of dLC: [BEM 2010]

A **syntax** for the **differential λ-calculus** by **mutual induction:**

[Bucciarelli-Ehrhard-Manzonetto 2010]

Simple terms:

$$\Lambda^s : \quad s, t \qquad ::= \quad x \mid \lambda x.s \mid sT \mid \mathsf{D}\,s\cdot t$$

Differential λ-terms:

$$\Lambda^d : \quad T \qquad ::= \quad 0 \mid s \mid s + T$$
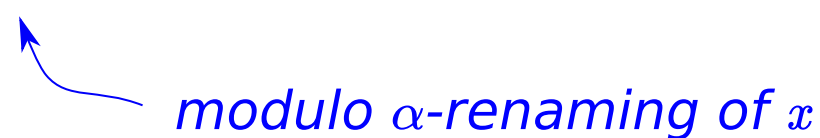
Syntax of dLC: [BEM 2010]

A **syntax** for the **differential λ-calculus** by **mutual induction:**

[Bucciarelli-Ehrhard-Manzonetto 2010]

Simple terms:

$$\Lambda^s : \quad s, t \quad ::= \quad x \mid \lambda x.s \mid sT \mid \mathrm{D}\, s \cdot t$$

variable

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d : \quad T \quad ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

Syntax of dLC: [BEM 2010]

A **syntax** for the **differential λ-calculus** by **mutual induction:**

[Bucciarelli-Ehrhard-Manzonetto 2010]

Simple terms:

$$\Lambda^s : \quad s, t \qquad ::= \quad x \mid \lambda x.s \mid sT \mid \mathrm{D}\,s \cdot t$$

variable

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d : \quad T \qquad ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

$$\mathbf{\Lambda^d = FreeCommutativeMonoid(\Lambda^s)}$$
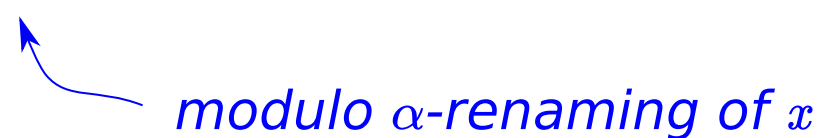
Syntax of dLC: [BEM 2010]

A **syntax** for the **_differential λ-calculus_** by **_mutual induction:_**

[Bucciarelli-Ehrhard-Manzonetto 2010]

Simple terms:

$$\Lambda^s: \quad s, t \qquad ::= \quad x \mid \lambda x.s \mid sT \mid \mathrm{D}\,s\cdot t$$

variable

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d: \qquad T \qquad ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

$$\mathbf{\Lambda^d = FreeCommutativeMonoid(\Lambda^s)}$$

Syntax: specified by operations and equations.

Syntax of dLC: [BEM 2010]

A **syntax** for the **_differential λ-calculus_** by **_mutual induction:_**

[Bucciarelli-Ehrhard-Manzonetto 2010]

Simple terms:

$$\Lambda^s: \quad s, t \quad ::= \quad x \mid \lambda x.s \mid sT \mid \mathrm{D}\, s \cdot t$$

variable

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d: \quad T \quad ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

$$\mathbf{\Lambda^d = FreeCommutativeMonoid(\Lambda^s)}$$

Syntax: specified by operations and equations.

But which ones are allowed ? What is the limit ?

Syntax of dLC: Our version

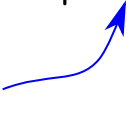**Which operations/equations are allowed to specify a syntax ?**

A stand-alone presentation of differential λ-terms:

Allow sums everywhere (not only in the right arg of application)

Differential λ-terms:

$$\Lambda^{\mathrm{d}} : \quad S, T \quad ::= \quad x \mid \lambda x. S \mid S\, T \mid \mathsf{D}S \cdot T$$

$$\mid \; 0 \; \mid \; S + T$$

neutral element for $+$

modulo commutativity and associativity

Macros in [BEM 2010]:

$$\lambda x. \Sigma_i t_i := \Sigma_i \lambda x. t_i$$

$$(\Sigma_i t_i) u := \Sigma_i t_i u$$

$$D(\Sigma_i t_i) \cdot (\Sigma_j u_j) := \Sigma_i \Sigma_j D t_i \cdot u_j$$

How can we compare these different versions ?

In which sense are they syntaxes ?

Which operations/equations are we allowed to specify in a syntax ?
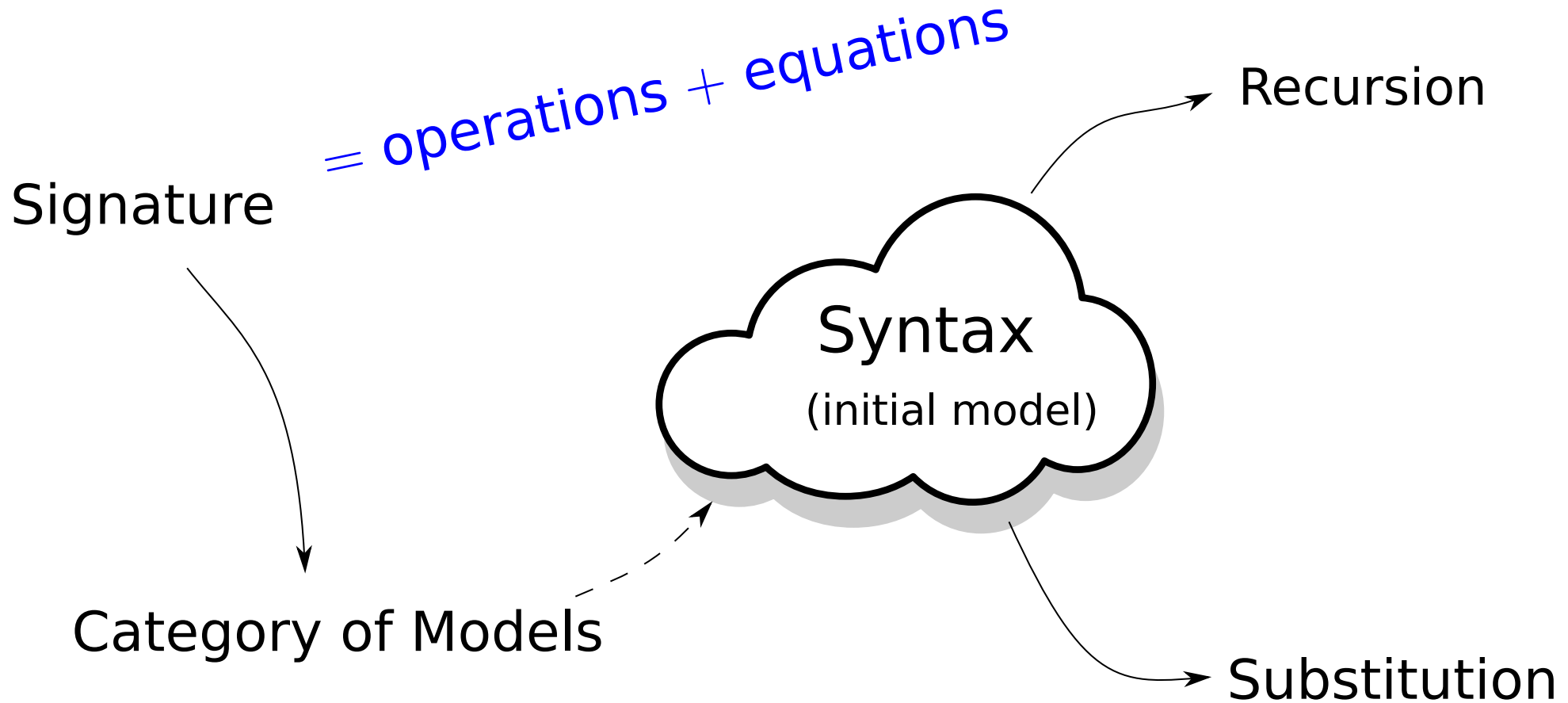
Syntax of dLC: Conclusion

How can we compare these different versions ?

In which sense are they syntaxes ?

Which operations/equations are we allowed to specify in a syntax ?

What is a syntax ?

$=$ operations $+$ equations

Recursion

Signature

Syntax
(initial model)

Category of Models

Substitution

generates a syntax = existence of the initial model

Table of contents

Table of contents

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

- Categorical formulation of term languages

- Initial semantics for binding signatures

2. Equations

3. Recursion

Substitution and monads

Example: differential λ-calculus

$$\Lambda^{\mathrm{d}} : \quad S, T \quad ::= \quad x \mid \lambda x. S \mid S\, T \mid \mathsf{D}S \cdot T$$
$$\mid\ 0 \mid\ S + T$$

Free variable indexing:

$$dLC : X \mapsto \{\text{terms taking free variables in } X\}$$

$$dLC(\emptyset) = \{0, \lambda z.z, \dots\}$$
$$dLC(\{x, y\}) = \{0, \lambda z.z, \dots, x, y, x + y, \dots\}$$

Substitution and monads

Example: differential λ-calculus

$$\Lambda^{\mathrm{d}} : \quad S,T \quad ::= \quad x \mid \lambda x.S \mid S\,T \mid \mathsf{D}S\cdot T$$
$$\mid \ 0 \ \mid \ S + T$$

Free variable indexing:

$$dLC : X \mapsto \{\text{terms taking free variables in } X\}$$

$$dLC(\emptyset) = \{0, \lambda z.z, \dots\}$$
$$dLC(\{x, y\}) = \{0, \lambda z.z, \dots, x, y, x + y, \dots\}$$

Parallel substitution:

Substitution and monads

Example: differential λ-calculus

$$\Lambda^{\mathrm{d}} : \quad S, T \quad ::= \quad x \mid \lambda x.\, S \mid S\, T \mid \mathsf{D}S \cdot T$$
$$\mid\; 0 \;\mid\; S + T$$

Free variable indexing:

$$dLC : X \mapsto \{\text{terms taking free variables in } X\}$$

$$dLC(\emptyset) = \{0, \lambda z.z, \dots\}$$
$$dLC(\{x, y\}) = \{0, \lambda z.z, \dots, x, y, x + y, \dots\}$$

Parallel substitution:

$$\begin{aligned}
\mathrm{bind_f} \;:\; dLC(\mathrm{X}) \;&\to\; dLC(\mathrm{Y}) \\
\mathrm{t} \;&\mapsto\; \mathrm{t}[\mathrm{x} \mapsto \mathrm{f(x)}]
\end{aligned}$$

where $\quad \mathrm{f} : \mathrm{X} \to dLC(\mathrm{Y})$

$\Rightarrow$ (dLC, var$_\mathrm{X}$: X $\subset$ dLC(X) , bind) = **monad on** $\mathbf{S}\mathrm{et}$

Substitution and monads

Example: differential λ-calculus

$$\Lambda^{\mathrm{d}} \;:\;\; S,T \quad\quad ::= \quad x \mid \lambda x.S \mid S\,T \mid \mathsf{D}S \cdot T$$
$$\mid\; 0 \;\mid\; S + T$$

Free variable indexing:

$$dLC : X \mapsto \{\text{terms taking free variables in } X\}$$

$$dLC(\emptyset) = \{0, \lambda z.z, \dots\}$$
$$dLC(\{x, y\}) = \{0, \lambda z.z, \dots, x, y, x + y, \dots\}$$

Parallel substitution:

$$\begin{aligned} \mathrm{bind_f} \;:\; dLC(\mathrm{X}) &\rightarrow\; dLC(\mathrm{Y}) \\ \mathrm{t} &\mapsto\; \mathrm{t[x \mapsto f(x)]} \end{aligned} \qquad \text{where}\quad \mathrm{f} : \mathrm{X} \rightarrow dLC(\mathrm{Y})$$

$\Rightarrow$ (dLC, $\mathrm{var_X}$: X $\subset$ dLC(X) , bind) = **monad on $\mathbf{Set}$**

monad morphism = mapping preserving variables and substitutions.

+ commutes with substitution

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

Categorical formulation

$dLC \times dLC$ supports dLC-substitution $\quad\rightsquigarrow\quad$ $dLC \times dLC$ is a **module over** $\boldsymbol{dLC}$

+ commutes with substitution $\quad\rightsquigarrow\quad$ $+ : dLC \times dLC \rightarrow dLC$ is a **module morphism**

Building blocks for binding signatures

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

- R itself

- $M \times N$ for any modules M and N (e.g., $R \times R$)

- M' = **derivative of a module M**: $M'(X) = M(X \coprod \{\diamond\})$.

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

Syntactic operations are module morphisms

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

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

$$0 : \qquad 1 \qquad \to \mathrm{dLC}$$
$$+ : \mathrm{dLC} \times \mathrm{dLC} \to \mathrm{dLC}$$

$$\mathrm{abs}_X : \mathrm{dLC}(X + \{\diamond\}) \to \mathrm{dLC}(X)$$

$$t \qquad \mapsto \quad \lambda\diamond.t$$

Combining operations into a single one using disjoint union

$$[\mathrm{app}, \mathrm{abs}] : (\mathrm{dLC} \times \mathrm{dLC}) \coprod \mathrm{dLC'} \to \mathrm{dLC}$$

$$[0, +] : 1 \coprod (\mathrm{dLC} \times \mathrm{dLC}) \qquad \to \mathrm{dLC}$$

The Big Picture of 1-signatures and models

A **1-signature** Σ = functorial assignment:

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

Example: $(0,+)$

$$\Sigma_{0,+}(R) = 1 \coprod (R \times R)$$

A **model of** Σ is a pair:

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

dLC $=$ model of $\Sigma_{0,+}$

$$[0, +] : 1 \coprod (dLC \times dLC) \to dLC$$

A **model morphism** $m : (R,\rho) \to (S,\sigma) =$ monad morphism commuting

with the module morphism:

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

The Big Picture of 1-signatures and models

A **1-signature** Σ = functorial assignment:

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

monad

Example: (0,+)

$$\Sigma_{0,+}(R) = 1 \coprod (R \times R)$$

A **model of** Σ is a pair:

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

dLC $=$ model of $\Sigma_{0,+}$

$$[0, +] : 1 \coprod (dLC \times dLC) \to dLC$$

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

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

The Big Picture of 1-signatures and models

A **1-signature** Σ = functorial assignment:

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

monad   module over R

Example: (0,+)

$$\Sigma_{0,+}(R) = 1 \coprod (R \times R)$$

A **model of** Σ is a pair:

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

dLC $=$ model of $\Sigma_{0,+}$

$$[0, +] : 1 \coprod (dLC \times dLC) \to dLC$$

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

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

A **1-signature** Σ = functorial assignment:

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

monad   module over R

Example: $(0,+)$

$$\Sigma_{0,+}(R) = 1 \coprod (R \times R)$$

A **model of** Σ is a pair:

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

monad

dLC $=$ model of $\Sigma_{0,+}$

$$[0,+] : 1 \coprod (dLC \times dLC) \to dLC$$

A **model morphism** $m : (R, \rho) \to (S, \sigma) =$ monad morphism commuting

with the module morphism:

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

A **1-signature** Σ = 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** $m : (R,\rho) \to (S,\sigma) =$ monad morphism commuting

with the module morphism:

Example: $(0,+)$

$$\Sigma_{0,+}(R) = 1 \coprod (R \times R)$$

dLC $=$ model of $\Sigma_{0,+}$

$$[0,+] : 1 \coprod (dLC \times dLC) \to dLC$$

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

Syntax

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

- **(0,+) language**:

  Signature: $R \mapsto 1 \coprod (R \times R)$

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

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

- **lambda calculus**:

  Signature: $R \mapsto R' \coprod (R \times R)$

  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

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

Algebraic 1-signatures correspond to the binding signatures described in [Fiore-Plotkin-Turi 1999]

(binding signatures: lists of natural numbers specify n-ary

operations, possibly binding variables)

Question: Can we enforce some equations in the syntax ?

For example: commutativity of $+$ for the differential λ-calculus.

Table of contents

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
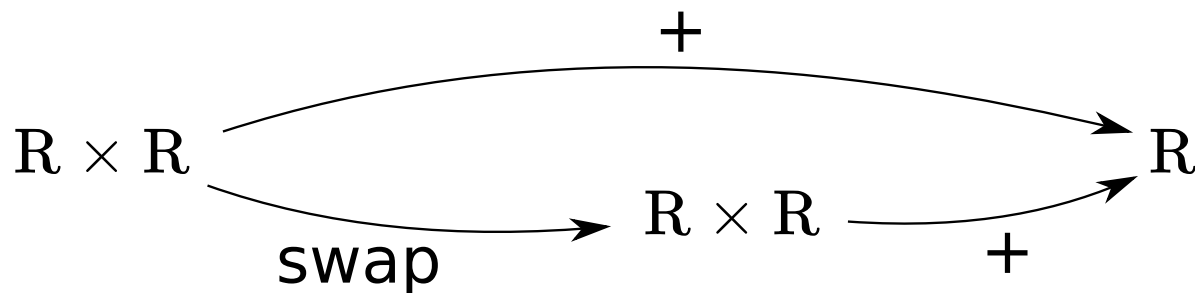
Specification of a commutative binary operation

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:

Review: Signatures with equations

- [Fiore-Hur 2010]: inductively defined set of possible equations.

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures

  Examples:
  - a binary commutative operation
  - application of the simple terms of differential λ-calculus (2nd variant)

  $$\text{app} : \text{dLC} \times \text{FreeCommutativeMonoid(dLC)} \rightarrow \text{dLC}$$

- [Fiore-Hur 2010]: inductively defined set of possible equations.

This work: alternative approach where monads and modules are the central notions.

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures

Examples:
- a binary commutative operation
- application of the simple terms of differential λ-calculus (2nd variant)

$$\text{app} : \text{dLC} \times \text{FreeCommutativeMonoid(dLC)} \to \text{dLC}$$

Review: Signatures with equations

- [Fiore-Hur 2010]: inductively defined set of possible equations.

This work: alternative approach where monads and modules are the central notions.

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures

    Examples:
    - a binary commutative operation
    - application of the simple terms of differential λ-calculus (2nd variant)

$$\text{app} : \text{dLC} \times \text{FreeCommutativeMonoid(dLC)} \rightarrow \text{dLC}$$

This work: more general equations (e.g. associativity of a binary op).

Equations

Given a 1-signature Σ,   (e.g. binary operation: $\Sigma(\mathrm{R}) = \mathrm{R} \times \mathrm{R}$)

a Σ-**equation** A $\Rrightarrow$ B is a functorial assignment: e.g. commutativity:

$$R \mapsto \left(A(R) \Longrightarrow B(R) \right) \qquad\qquad R \mapsto \left(R \times R \overset{+}{\underset{+ \circ swap}{\Longrightarrow}} R \right)$$

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** (Σ, E):

- a model R of Σ

- s.t. $\forall$ (A $\Rrightarrow$ B) $\in$ E, the two morphisms $A(R) \Rrightarrow B(R)$ are equal

Algebraic 2-signature:

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

algebraic 1-signature

set of **elementary** Σ-equations

Syntax exists for any algebraic 2-signature

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

1. A "preserves pointwise epimorphisms"

    (e.g., any "algebraic 1-signature", such as $\mathrm{R} \mapsto \mathrm{R} \times \mathrm{R}$)

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

Example: λ-calculus modulo βη

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

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

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

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

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

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

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

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

model of Σ_{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**: $t = \lambda x.(t\, x)$

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

Example: fixpoint operator

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

for any term $t \in R(X \amalg \{ \diamond \})$, $f(t) = t[\diamond \mapsto f(t)]$,

i.e. (1)

$$
\begin{array}{ccc}
 & f & \\
R' & \longrightarrow & R \\
 & \searrow \quad \nearrow & \\
(\mathrm{id}_{R'}, f) & R' \times R & \sigma_R
\end{array}
$$

commutes.

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

$$\Sigma_{\mathrm{fix}}(R) := R' \qquad\qquad E_{\mathrm{fix}} = \{ (1) \}$$

Fixpoint operators in $LC_{\beta\eta}$ are in one to one correspondance with

fixpoint combinators (i.e. λ-terms Y s.t. $t(Yt) = Yt$ for any t).

Combining algebraic 2-signatures

Algebraic 2-signatures can be combined:

fixpoint operator

λ-calculus modulo βη

$$(\Sigma_{\mathrm{fix}}, \mathrm{E}_{\mathrm{fix}}) \qquad + \qquad (\Sigma_{\mathrm{LCβη}}, \mathrm{E}_{\mathrm{LCβη}})$$

$$=$$

$$(\Sigma_{\mathrm{fix}} \coprod \Sigma_{\mathrm{LCβη}} \ , \ \mathrm{E}_{\mathrm{fix}} \cup \mathrm{E}_{\mathrm{LCβη}})$$

λ-calculus modulo βη with an explicit fixpoint operator

Example: free commutative monoid

An algebraic 2-signature $(\Sigma_{\mathrm{mon}}, E_{\mathrm{mon}})$ for the free commutative monoid

monad:
$$\Sigma_{\mathrm{mon}}(R) := 1 \coprod (R \times R)$$

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

$$0 : 1 \to R \qquad + : R \times R \to R$$

Example: free commutative monoid

An algebraic 2-signature $(\Sigma_{\mathrm{mon}}, E_{\mathrm{mon}})$ for the free commutative monoid

monoid:
$$\Sigma_{\mathrm{mon}}(R) := 1 \coprod (R \times R)$$

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

$$0 : 1 \to R \qquad + : R \times R \to R$$

4 elementary Σ-equations:

Syntax of the *differential λ-calculus*:

Differential λ-terms

$$
\begin{aligned}
s,t \ ::= \ & x \\
| \ & \lambda x.t \\
| \ & s\ t \\
| \ & Ds \cdot t \\
| \ & s + t \\
| \ & 0
\end{aligned}
$$

$\left.\begin{array}{l} \lambda x.t \\ s\ t \end{array}\right\}$ λ-calculus

$\left.\begin{array}{l} s + t \\ 0 \end{array}\right\}$ free commutative monoid

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

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

Algebraic 1-signature for dLC

Syntax of the *differential λ-calculus*:

Differential λ-terms   Corresponding 1-signature

$$s,t \ ::= \ \mathrm{x}$$
$$| \quad \lambda\mathrm{x}.\mathrm{t}$$
$$| \quad \mathrm{s}\,\mathrm{t}$$
$$| \quad \mathrm{Ds} \cdot \mathrm{t}$$
$$| \quad \mathrm{s} + \mathrm{t}$$
$$| \quad 0$$

$$\color{blue}{\Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R}' \coprod (\mathrm{R} \times \mathrm{R})}$$

$$\color{blue}{\mathrm{R} \mapsto \mathrm{R} \times \mathrm{R}}$$

$$\color{blue}{\Sigma_{\mathrm{mon}}(\mathrm{R}) = 1 \coprod (\mathrm{R} \times \mathrm{R})}$$

Algebraic 1-signature for dLC

Syntax of the *differential λ-calculus*:

Differential λ-terms   Corresponding 1-signature

$$s,t \; ::= \; x$$
$$| \quad \lambda x.t$$
$$| \quad s\, t$$
$$| \quad Ds \cdot t$$
$$| \quad s + t$$
$$| \quad 0$$

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

$$R \mapsto R \times R$$

$$\Sigma_{\mathrm{mon}}(R) = 1 \coprod (R \times R)$$

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

Elementary equations for dLC

Commutative monoidal structure:

$$\mathrm{E}_{\mathrm{mon}} \begin{cases} s + t = t + s & \mathrm{R} \times \mathrm{R} \rightrightarrows \mathrm{R} \\ s + (t + u) = (s + t) + u & \mathrm{R} \times \mathrm{R} \times \mathrm{R} \rightrightarrows \mathrm{R} \\ 0 + t = t & \mathrm{R} \rightrightarrows \mathrm{R} \\ t + 0 = t & \mathrm{R} \rightrightarrows \mathrm{R} \end{cases}$$

Linearity:

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

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

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

$$\cdots$$

Table of contents

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$f : R \to S$$

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

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$f : R \to S$$

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

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

Principle of recursion

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

Recipe for constructing "by recursion" a monad morphism:

$$f : R \to S$$

initial model of a 2-signature (Σ, E)

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

   $\Rightarrow$ induces a Σ-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:

$$f : R \to S$$

initial model of a 2-signature (Σ, E)

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

    $\Rightarrow$ induces a Σ-model (S, 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:

$$f : R \to S$$

initial model of a 2-signature (Σ, E)

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

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

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

   $\Rightarrow$ induces a model of (Σ, E)

Principle of recursion

Recursion on the syntax ≃ Initiality in the category of models

Recipe for constructing "by recursion" a monad morphism:

$$f : R \to S$$

initial model of a 2-signature (Σ, E)

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

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

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

   $\Rightarrow$ induces a model of (Σ, E)

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

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

$\mathcal{P}$ = power set monad

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

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

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

$\mathcal{P}$ = power set monad

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

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

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

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

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

$\mathcal{P}$ = power set monad

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

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

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

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

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

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

  λ-calculus modulo βη with a fixpoint operator $fix : LC_{\beta\eta fix}' \rightarrow LC_{\beta\eta fix}$

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

  λ-calculus modulo βη

monad morphism

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

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

a chosen fixpoint combinator

Example: Translating λ-calculus with fixpoint

$LC_{βηfix}$ = initial model of $(\Sigma_{LCβη}, E_{LCβη}) + (\Sigma_{fix}, E_{fix})$

   λ-calculus modulo βη with a fixpoint operator $\mathrm{fix} : LC_{βηfix}' \to LC_{βηfix}$

$LC_{βη}$ = initial model of $(\Sigma_{LCβη}, E_{LCβη})$

   λ-calculus modulo βη

monad morphism

Definition of a translation $\mathrm{f} : LC_{βηfix} \to LC_{βη}$ **s.t.**

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

a chosen fixpoint combinator

$\Rightarrow$ make $LC_{βη}$ a model of $(\Sigma_{LCβη}, E_{LCβη}) + (\Sigma_{fix}, E_{fix})$:

app, abs $\hat{Y} : LC_{βη}' \to LC_{βη}$

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

Example: Translating λ-calculus with fixpoint

$LC_{βηfix}$ = initial model of $(\Sigma_{\mathrm{LC}βη}, E_{\mathrm{LC}βη}) + (\Sigma_{\mathrm{fix}}, E_{\mathrm{fix}})$
  λ-calculus modulo βη with a fixpoint operator $\mathrm{fix} : LC_{βηfix}' \to LC_{βηfix}$

$LC_{βη}$ = initial model of $(\Sigma_{\mathrm{LC}βη}, E_{\mathrm{LC}βη})$
  λ-calculus modulo βη

monad morphism

Definition of a translation $\mathrm{f} : LC_{βηfix} \to LC_{βη}$ **s.t.**

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

a chosen fixpoint combinator

$\Rightarrow$ make $LC_{βη}$ a model of $(\Sigma_{\mathrm{LC}βη}, E_{\mathrm{LC}βη}) + (\Sigma_{\mathrm{fix}}, E_{\mathrm{fix}})$:

app, abs
$\hat{\mathrm{Y}} : LC_{βη}' \to LC_{βη}$
$\mathrm{t} \mapsto \mathrm{app(Y,abs(t))}$

Initiality of $LC_{βηfix}$ $\Rightarrow$ $\mathrm{f} : LC_{βηfix} \to LC_{βη}$

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

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

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

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

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 !**

Example: Computing the size of a term

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

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 [CSL AHLM 2018]:

continuation monad $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

1. define $\quad \mathrm{f} : \mathrm{LC} \to C$ by recursion
2. deduce $\quad \mathrm{s} : \mathrm{LC} \to \mathbb{N}$

Example: Computing the size of a term

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

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 2018]:

continuation monad C(X) = $\mathbb{N}^{(\mathbb{N}^X)}$

1. define $\quad f : \mathrm{LC} \to \mathrm{C}$ by recursion

2. deduce $\quad s : \mathrm{LC} \to \mathbb{N}$

affects an arbitrary size to each variable

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

Example: Computing the size of a term

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

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 2018]:

continuation monad $C(X) = \mathbb{N}^{(\mathbb{N}^X)}$

1. define $\quad f : \mathrm{LC} \to \textcolor{red}{C}$ by recursion

2. deduce $\ s : \mathrm{LC} \to \mathbb{N}$

affects an arbitrary size to each variable

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

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

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 work 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 work to simply typed syntaxes.

Thank you!