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

Syntax

Arity/Signature

Operation/Construction

This talk: give a mathematical account of this area

Motivation: dLC

dLC = ***differentiable λ-calculus*** [Ehrard-Regnier 2003].

- Syntax: not straightforward (equations involved).

  e.g. $\mathrm{s}+\mathrm{t} = \mathrm{t}+\mathrm{s}$

- Later articles: alternative presentations of the syntax (+/- verbose).

- No well-established *scheme* commonly used beyond BNF grammars.

Our work:

- a mathematical theory of presentations of monads,
- induces a scheme for presenting syntaxes.

Motivation: dLC

dLC = ***differentiable λ-calculus*** [Ehrard-Regnier 2003].

- Syntax: not straightforward (equations involved).

  e.g. $\mathrm{s}+\mathrm{t} = \mathrm{t}+\mathrm{s}$

- Later articles: alternative presentations of the syntax (+/- verbose).

- No well-established *scheme* commonly used beyond BNF grammars.

Next slides: 3 variants of the dLC syntax:

Our work:

- a mathematical theory of presentations of monads,
- induces a scheme for presenting syntaxes.

Syntax of dLC: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

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

Differential λ-terms:

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

Syntax of dLC: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

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: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

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 of dLC: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

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

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

Syntax: specified by operations and equations.

Syntax of dLC: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

Differential λ-terms:

modulo α-renaming of x

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

neutral element for $+$

modulo commutativity

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

Syntax: specified by operations and equations.

But which ones are allowed ? What is the limit ?

Syntax of dLC: version 2/3

Which operations/equations are allowed to specify a syntax ?

A stand-alone presentation of simple terms:

Simple terms:

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

Differential λ-terms:

$$T \in \Lambda^{\mathsf{d}} = \mathsf{FreeCommutativeMonoid}(\Lambda^{\mathsf{s}})$$

Syntax of dLC: version 2/3

Which operations/equations are allowed to specify a syntax ?

A stand-alone presentation of simple terms:

Simple terms:

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

as an operation: $\Lambda^\mathsf{s} \times \mathsf{FreeCommutativeMonoid}(\Lambda^\mathsf{s}) \to \Lambda^\mathsf{s}$

Differential λ-terms:

$$T \in \Lambda^\mathsf{d} = \mathsf{FreeCommutativeMonoid}(\Lambda^\mathsf{s})$$

Syntax of dLC: version 3/3

Which operations/equations are allowed to specify a syntax ?

A stand-alone presentation of differential λ-terms:

Allow summands 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

$$\lambda x.\Sigma_i t_i = \Sigma_i \lambda x.t_i$$
$$(\Sigma_i t_i)u = \Sigma_i t_i u$$
$$D(\Sigma_i t_i) \cdot (\Sigma_j u_j) = \Sigma_i \Sigma_j D t_i \cdot u_j$$

[Categorical Models for Simply Typed Resource Calculi]

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 ?

Syntax of dLC: Conclusion

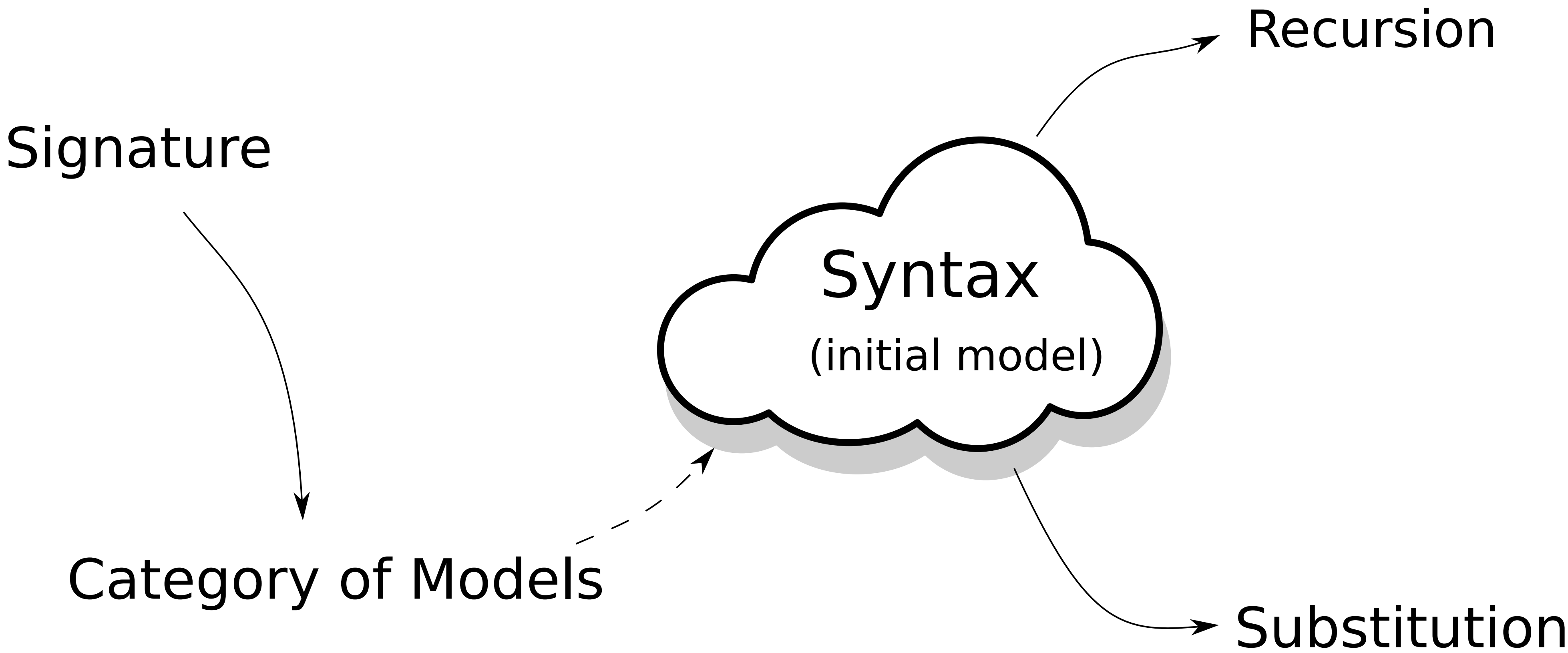
How can we compare these different versions ?

In which sense are they syntaxes ?

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

What is a syntax ?

What is a syntax?



generates a syntax = existence of the initial model

Overview

Topic: specification of untyped syntaxes (e.g. differential λ-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

Related work of Fiore-Hur 2010

[Fiore-Hur 2010]: presentations of simply typed languages with

- generating *binding* operations (e.g. λ-abstraction)
- equations among them.

Our work: a variant of their approach

- for the untyped setting,
- focus on *monads* and *modules* over them

Table of contents

Table of contents

Categorical formulation of a term language

Example: differential λ-calculus

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

Free variable indexing:

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

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

Categorical formulation of a term language

Example: differential λ-calculus

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

Free variable indexing:

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

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

Free variable renaming:

$$\begin{array}{rcl} \mathrm{dLC(f)} : \mathrm{dLC(X)} & \rightarrow & \mathrm{dLC(Y)} \\ \mathrm{t} & \mapsto & \mathrm{t[x \mapsto f(x)]} \end{array}$$

where $\mathrm{f : X \rightarrow Y}$

Categorical formulation of a term language

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:

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

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

Free variable renaming:

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

where $\mathrm{f : X \to Y}$

$\Rightarrow$ **dLC is an endofunctor on $\mathbf{Set}$**

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

$$
\begin{aligned}
+ &: \; dLC \times dLC \overset{\cdot}{\to} dLC \\
0 &: \; 1 \overset{\cdot}{\to} dLC \\
&\dots
\end{aligned}
$$

Variables as a natural transformation:

$$\mathrm{var} : \; \mathrm{Id}_{\mathrm{Set}} \overset{\cdot}{\to} dLC$$

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

$$
\begin{aligned}
+ :&\ dLC \times dLC \stackrel{\cdot}{\to} dLC \\
0 :&\ 1 \stackrel{\cdot}{\to} dLC \\
&\dots
\end{aligned}
$$

Variables as a natural transformation:

$$\mathrm{var} : \quad \mathrm{Id}_{\mathrm{Set}} \stackrel{\cdot}{\to} dLC$$

This gives a notion of model for the language (+ , 0):

model $=$ endofunctor R with natural transformations:

$$
\begin{aligned}
+ :&\ R \times R \stackrel{\cdot}{\to} R \\
0 :&\ 1 \stackrel{\cdot}{\to} R \\
\mathrm{var} :&\ \mathsf{Id} \stackrel{\cdot}{\to} R
\end{aligned}
\qquad \text{or} \qquad (R \times R) \coprod 1 \coprod \mathsf{Id} \stackrel{\cdot}{\to} R
$$

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

$$\begin{aligned} + : &\ dLC \times dLC \stackrel{\cdot}{\to} dLC \\ 0 : &\ 1 \stackrel{\cdot}{\to} dLC \\ &\ldots \end{aligned}$$

Variables as a natural transformation:

$$\mathrm{var} : \quad \mathrm{Id}_{\mathrm{Set}} \stackrel{\cdot}{\to} dLC$$

This gives a notion of model for the language (+ , 0):

model $=$ endofunctor R with natural transformations:

$$\begin{aligned} + : &\ R \times R \stackrel{\cdot}{\to} R \\ 0 : &\ 1 \stackrel{\cdot}{\to} R \\ \mathrm{var} : &\ \mathsf{Id} \stackrel{\cdot}{\to} R \end{aligned} \qquad \text{or} \qquad (R \times R) \coprod 1 \coprod \mathsf{Id} \stackrel{\cdot}{\to} R$$

Next slides: **generalize this pattern to other languages**

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, +:

$$\Big(\underset{0}{()},\ \underset{+}{(0,\, 0)}\Big)$$

$$0, \quad 0{+}x, \quad x{+}y$$
$$(0{+}x){+}y, \quad 0{+}(x{+}y),$$
$$\ldots$$

Lambda calculus:

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

Initial semantics for binding signatures

model of (0, +) $=$ endofunctor R with a natural transformation:

$$[+, 0, \mathrm{var}] : (R \times R) \coprod 1 \coprod \mathsf{Id} \xrightarrow{\cdot} R$$

morphism $=$ natural transformation commuting with 0, + and var.

Similarly, any binding signature gives rise to a category of models.

Well-established theorem

The initial model of a binding signature Σ always exists.

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

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

Initial semantics for binding signatures

model of (0, +) $=$ endofunctor R with a natural transformation:

$$[+, 0, \mathrm{var}] : \ (R \times R) \coprod 1 \coprod \mathsf{Id} \stackrel{\cdot}{\to} R$$

morphism $=$ natural transformation commuting with 0, + and var.

Similarly, any binding signature gives rise to a category of models.

Well-established theorem

The initial model of a binding signature Σ always exists.

Cf next section

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

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

Initial semantics for binding signatures

model of (0, +) $=$ endofunctor R with a natural transformation:

$$[+, 0, \mathrm{var}] : \ (R \times R) \coprod 1 \coprod \mathsf{Id} \stackrel{\cdot}{\to} R$$

morphism $=$ natural transformation commuting with 0, + and var.

Similarly, any binding signature gives rise to a category of models.

Well-established theorem

The initial model of a binding signature Σ always exists.

Cf next section

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

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

Initiality still holds in the subcategory of models with a substitution.

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:

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

Example: (0,+)

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

A **model of Σ** is a pair:

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

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

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

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

Example: (0,+)

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

A **model of Σ** is a pair:

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

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

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

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

Example: (0,+)

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

A **model of Σ** is a pair:

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

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

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

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

Example: (0,+)

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

A **model of Σ** is a pair:

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

monad

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

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

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

Example: (0,+)

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

A **model of Σ** is a pair:

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

monad

module morphism

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

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

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

Substitution and monads

Reminder:

- $dLC(X) = \{$ differential λ-terms taking free variables in X $\}$
- Variables as a natural transformation $\mathrm{var} : \mathrm{Id}_{\mathrm{Set}} \to dLC$
- **Variable renaming** by functoriality:

$$dLC(\mathrm{f})(\mathrm{t}) = \mathrm{t}[\,\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\,]$$ where $\mathrm{f} : \mathrm{X} \to \mathrm{Y}$ is a *renaming*

Substitution and monads

Reminder:

- $dLC(X) = \{$ differential λ-terms taking free variables in $\mathrm{X}\ \}$
- Variables as a natural transformation $\mathrm{var} : \mathrm{Id}_{\mathrm{Set}} \to dLC$
- **Variable renaming** by functoriality:

$$dLC(\mathrm{f})(\mathrm{t}) = \mathrm{t}[\,\underbrace{\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})}\,]$$ where $\mathrm{f} : \mathrm{X} \to \mathrm{Y}$ is a *renaming*

still makes sense if f(x) is an arbitrary term (rather than just a variable)

Substitution and monads

Reminder:

- $dLC(X) = \{$ differential λ-terms taking free variables in X $\}$
- Variables as a natural transformation $\mathrm{var} : \mathrm{Id}_{\mathrm{Set}} \to dLC$
- **Variable renaming** by functoriality:

$$dLC(\mathrm{f})(\mathrm{t}) = \mathrm{t}\underbrace{[\ \mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\ ]}$$ where $\mathrm{f} : \mathrm{X} \to \mathrm{Y}$ is a *renaming*

still makes sense if f(x) is an arbitrary term (rather than just a variable)

Variable renaming $=$ special case of **substitution**:

$$\begin{array}{rcccc} \mathrm{bind}_{\mathrm{f}} & : & dLC(\mathrm{X}) & \to & dLC(\mathrm{Y}) \\ & & \mathrm{t} & \mapsto & \mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})] \end{array}$$ where $\mathrm{f} : \mathrm{X} \to dLC(\mathrm{Y})$

Substitution and monads

Reminder:

- $dLC(X) = \{$ differential λ-terms taking free variables in X $\}$
- Variables as a natural transformation $\mathrm{var} : \mathrm{Id}_{\mathrm{Set}} \to dLC$
- **Variable renaming** by functoriality:

$$dLC(\mathrm{f})(\mathrm{t}) = \mathrm{t}\underbrace{[\ \mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\ ]}$$ where $\mathrm{f} : \mathrm{X} \to \mathrm{Y}$ is a *renaming*

still makes sense if f(x) is an arbitrary term (rather than just a variable)

Variable renaming $=$ special case of **substitution**:

$$\begin{array}{rcccc} \mathrm{bind}_{\mathrm{f}} & : & dLC(\mathrm{X}) & \to & dLC(\mathrm{Y}) \\ & & \mathrm{t} & \mapsto & \mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})] \end{array}$$ where $\mathrm{f} : \mathrm{X} \to dLC(\mathrm{Y})$

(dLC, var, bind) = a **monad**.

monad morphism = mapping preserving variables and substitutions.

Preview: Operations are module morphisms

+ 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 $\rightsquigarrow$ $dLC \times dLC$ is a **module over** $\boldsymbol{dLC}$

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

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 $M'(X) = M(X \coprod \{\diamond\})$.

  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.

$$id_M : M \to M$$

$$0 : 1 \to dLC$$

$$+ : dLC \times dLC \to dLC$$

$$app : dLC \times dLC \to dLC$$

$$abs : dLC' \to dLC$$

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

  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

Theorem [Hirschowitz & Maggesi 2007]

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 binding signatures through the embedding:

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

Question: Can we enforce some equations in the syntax ?
For example: commutativity of $+$ for the differential λ-calculus.

Table of contents

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

Example: a commutative binary operation

Specification of a binary operation

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

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

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:

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

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 (2^{nd} 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 (2^{nd} variant)

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

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 (2^{nd} variant)

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

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 $\rightrightarrows$ B is a functorial assignment:

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

model of Σ

parallel pair of module morphisms over R

e.g. commutativity:

$$R \mapsto \left(R \times R \overset{+}{\underset{+\circ swap}{\rightrightarrows}} R \right)$$

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 $A(R) \rightrightarrows B(R)$ are equal

Initial semantics for algebraic 2-signatures

Algebraic 2-signature:

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

algebraic 1-signature

set of **elementary** Σ-equations

> **Theorem**
> Syntax exists for any algebraic 2-signature

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", such as $\mathrm{R} \mapsto \mathrm{R}{\times}\mathrm{R}$)

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

Example: λ-calculus modulo βη

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

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

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

η-equation: $t = \lambda x.(t\ x)$

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

Example: λ-calculus modulo βη

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

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

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

η-equation: $t = \lambda x.(t\ x)$

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

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

Example: fixpoint operator

Definition [AHLM CSL 2018]

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

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

i.e. $\color{blue}{(1)}$

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

commutes.

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

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

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

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

$$=$$

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

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:

$$\mathbf{\Sigma}_{\mathrm{mon}}(R) := 1 \coprod (R \times R)$$

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 $(\Sigma_{\text{mon}}, E_{\text{mon}})$ for the free commutative monoid monad:

$$\mathbf{\Sigma}_{\text{mon}}(\mathrm{R}) := 1 \coprod (\mathrm{R} \times \mathrm{R})$$

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

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

4 elementary Σ-equations:

$$\begin{tikzcd} \underset{t}{R} \arrow[r, bend left, "0+t"] \arrow[r, bend right, "t"'] & R \end{tikzcd} \qquad \begin{tikzcd} \underset{s,t,u}{R \times R \times R} \arrow[r, bend left, "(s+t)+u"] \arrow[r, bend right, "s+(t+u)"'] & R \end{tikzcd}$$

$$\begin{tikzcd} \underset{t}{R} \arrow[r, bend left, "t+0"] \arrow[r, bend right, "t"'] & R \end{tikzcd} \qquad \begin{tikzcd} \underset{s,t}{R \times R} \arrow[r, bend left, "s+t"] \arrow[r, bend right, "t+s"'] & R \end{tikzcd}$$

Our target: dLC

Syntax of the *differentiable λ-calculus*:

Differential λ-terms

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

λ-calculus

free commutative monoid

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

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

Algebraic 1-signature for dLC

Syntax of the *differentiable λ-calculus*:

Differential λ-terms Corresponding 1-signature

$$
\begin{array}{rcll}
s,t & ::= & \mathrm{x} & \\
& | & \lambda \mathrm{x.t} & \multirow{2}{*}{$\left.\right\}\ \Sigma_{\mathrm{LC}}(\mathrm{R}) = \mathrm{R}' \coprod (\mathrm{R} \times \mathrm{R})$} \\
& | & \mathrm{s\ t} & \\
& | & \mathrm{Ds} \cdot \mathrm{t} & \mathrm{R} \mapsto \mathrm{R} \times \mathrm{R} \\
& | & \mathrm{s} + \mathrm{t} & \multirow{2}{*}{$\left.\right\}\ \Sigma_{\mathrm{mon}}(\mathrm{R}) = 1 \coprod (\mathrm{R} \times \mathrm{R})$} \\
& | & 0 &
\end{array}
$$

Algebraic 1-signature for dLC

Syntax of the *differentiable λ-calculus*:

Differential λ-terms Corresponding 1-signature

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

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

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

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

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

Elementary equations for dLC

Commutative monoidal structure:

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

Linearity:

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

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

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

...

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

$\mathcal{P}$ = power set monad

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

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

Example: Computing the set of free variables

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

$\mathcal{P}$ = power set monad

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

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

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

$\mathcal{P}$ = power set monad

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

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

$\Rightarrow$ 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

$\text{LC}_{\beta\eta\text{fix}}$ = 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 $\mathrm{fix} : \mathrm{LC}_{\beta\eta\mathrm{fix}}' \to \mathrm{LC}_{\beta\eta\mathrm{fix}}$

$\text{LC}_{\beta\eta}$ = 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

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.**

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

a chosen fixpoint combinator

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

$\mathrm{app, abs}$

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

Example: Translating λ-calculus with fixpoint

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

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

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

λ-calculus modulo βη

monad morphism

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

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

a chosen fixpoint combinator

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

app, abs

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

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

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

Example: Computing the size of a term

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

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

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

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

Example: Computing the size of a term

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

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

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

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

Example: Computing the size of a term

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

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

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

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

Solution [CSL AHLM 2018]:

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

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

Example: Computing the size of a term

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

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

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

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

Solution [CSL AHLM 2018]:

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

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

affects an arbitrary size to each variable

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

Example: Computing the size of a term

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

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

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

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

Solution [CSL AHLM 2018]:

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

1. define $f : \mathrm{LC} \to \color{red}{C}$ 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 $g : \mathrm{LC}(X) \times \color{blue}{\mathbb{N}^X} \to \mathbb{N}$

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

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!