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: LCD

The ***differentiable λ-calculus*** (LCD) was introduced by [Ehrard-Regnier 2003].

The syntax is not straightforward, as it involves some equations.

There are alternative presentations of the syntax in later articles, more or less verbose.

Motivation: LCD

The ***differentiable λ-calculus*** (LCD) was introduced by

[Ehrard-Regnier 2003].

The syntax is not straightforward, as it involves some equations.

There are alternative presentations of the syntax in later articles, more or less verbose.

The next slides give 3 variants of the syntax:

1. Mutual definition of *simple terms* and *differential λ-terms*
2. Stand-alone definition of simple terms
3. Stand-alone definition of differential λ-terms.

Syntax of LCD: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

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

Differential λ-terms:

$$\Lambda^d : \quad S, T, U, V ::= \quad 0 \mid s \mid s + T$$

Syntax of LCD: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d : \quad S, T, U, V ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

Syntax of LCD: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d : \quad S, T, U, V ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

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

Syntax of LCD: version 1/3

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

[Categorical Models for Simply Typed Resource Calculi]

Simple terms:

variable

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

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d : \quad S, T, U, V ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

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

A syntax is specified by operations and equations.

Syntax of LCD: version 1/3

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

[Categorical Models for Simply Typed Resource Calcululi]

Simple terms:

variable

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

modulo α-renaming of x

Differential λ-terms:

$$\Lambda^d: \quad S,T,U,V ::= \quad 0 \mid s \mid s + T$$

neutral element for $+$

modulo commutativity

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

A syntax is specified by operations and equations.

But which ones are allowed ? What is the limit ?

Syntax of LCD: version 2/3

Which operations/equations are allowed to specify a syntax ?

A stand-alone presentation of simple terms:

Simple terms:

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

Differential λ-terms:

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

Syntax of LCD: version 2/3

Which operations/equations are allowed to specify a syntax ?

A stand-alone presentation of simple terms:

Simple terms:

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

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

Differential λ-terms:

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

Syntax of LCD: 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

Turn [Categorical Models for Simply Typed Resource Calculi]'s abbreviations into equations:

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

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

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

Syntax of LCD: Conclusion

There is no well-established *scheme* for presenting a syntax.

We propose such a scheme (which is the counterpart of a mathematical theory of presentations of monads).

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 LCD: Conclusion

There is no well-established *scheme* for presenting a syntax.

We propose such a scheme (which is the counterpart of a mathematical theory of presentations of monads).

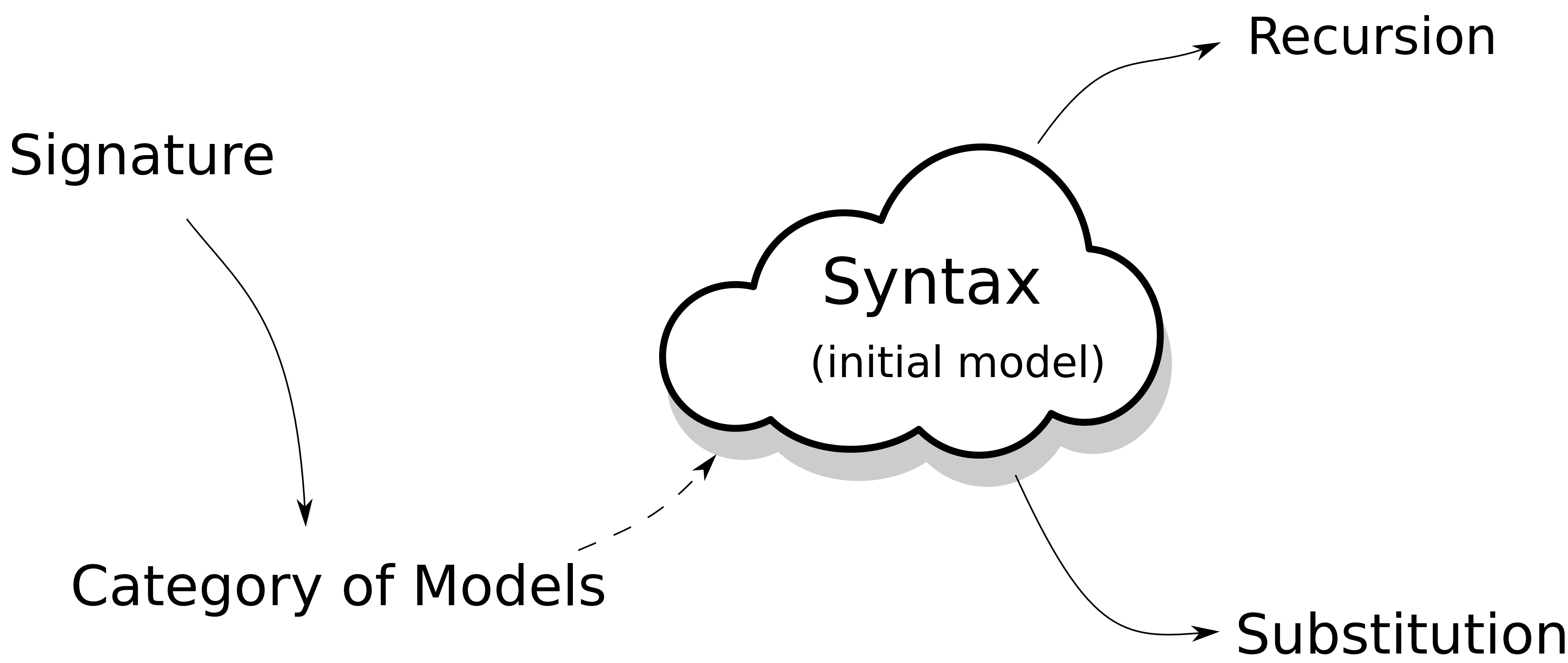
How can we compare these different versions ?

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 and construction of untyped syntaxes with variables and a well-behaved substitution (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

Previous work of Fiore-Hur 2010

[Fiore-Hur 2010]: presentations of simply typed languages by generating *binding* operations (e.g. λ-abstraction) and equations among them.

Our work: for the untyped setting, a variant of their approach where monads and modules over them are the central notions.

Table of contents

Table of contents

Categorical formulation of a term language

Example: differential λ-calculus (last variant)

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

Free variable indexing:

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

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

Categorical formulation of a term language

Example: differential λ-calculus (last variant)

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

Free variable indexing:

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

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

Free variable renaming:

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

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

Categorical formulation of a term language

Example: differential λ-calculus (last variant)

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

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:

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

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

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

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

$$
\begin{aligned}
+ :&\ LCD \times LCD \stackrel{\cdot}{\to} LCD \\
0 :&\ \quad 1 \qquad\qquad\quad \stackrel{\cdot}{\to} LCD \\
&\ \cdots
\end{aligned}
$$

Variables as a natural transformation:

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

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

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

Variables as a natural transformation:

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

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

model $=$ endofunctor R with natural transformations:

$$\begin{aligned} + : \;& R \times R \stackrel{\cdot}{\to} R \\ 0 : \;& 1 \stackrel{\cdot}{\to} R \\ \mathrm{var} : \;& \mathsf{Id} \stackrel{\cdot}{\to} R \end{aligned}$$

or

$$(R \times R) \coprod 1 \coprod \mathsf{Id} \stackrel{\cdot}{\to} R$$

Categorical formulation of a term language

commute with variable renaming

Operations as natural transformations:

$$
\begin{aligned}
+ :&\ LCD \times LCD \xrightarrow{\cdot} LCD \\
0 :&\ 1 \xrightarrow{\cdot} LCD \\
&\ \ldots
\end{aligned}
$$

Variables as a natural transformation:

$$\mathrm{var} : \quad \mathrm{Id}_{\mathrm{Set}} \xrightarrow{\cdot} LCD$$

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

model $=$ endofunctor R with natural transformations:

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

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, +:

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

Lambda calculus:

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

Initial semantics for binding signatures

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

$$\begin{aligned} + :&\quad R \times R \xrightarrow{\cdot} R \\ 0 :&\quad 1 \xrightarrow{\cdot} R \\ \mathrm{var} :&\quad \mathsf{Id} \xrightarrow{\cdot} R \end{aligned}$$

or

$$(R \times R) \coprod 1 \coprod \mathsf{Id} \xrightarrow{\cdot} R$$

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 natural transformations:

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

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 natural transformations:

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

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]

... and initiality still holds in the subcategory of models with a well-behaved 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,+)

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

A **model of Σ** is a pair:

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

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

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,+)

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

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

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

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

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,+)

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

A **model of Σ** is a pair:

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

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

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,+)

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

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

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

monad

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

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,+)

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

A **model of Σ** is a pair:

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

monad

module morphism

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

monad := endofunctor with substitution

module over a monad := endofunctor with substitution

module morphism := natural transformation preserving substitution

Substitution and monads

Reminder:

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

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

Substitution and monads

Reminder:

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

$$LCD(\mathrm{f})(\mathrm{t}) = \mathrm{t}\underbrace{[\,\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\,]}_{} \qquad \text{where} \quad \mathrm{f} : \mathrm{X} \to \mathrm{Y} \text{ is a } \textit{renaming}$$

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

Substitution and monads

Reminder:

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

$$LCD(\mathrm{f})(\mathrm{t}) = \mathrm{t}\underbrace{[\ \mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\ ]}$$ 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**:

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

Substitution and monads

Reminder:

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

$$LCD(\mathrm{f})(\mathrm{t}) = \mathrm{t}\underbrace{[\,\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})\,]} \qquad \text{where} \quad \mathrm{f} : \mathrm{X} \to \mathrm{Y} \text{ is a } \textit{renaming}$$

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

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

$$\begin{array}{rccc} \mathrm{bind}_{\mathrm{f}} : & LCD(\mathrm{X}) & \to & LCD(\mathrm{Y}) \\ & \mathrm{t} & \mapsto & \mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})] \end{array} \qquad \text{where} \quad \mathrm{f} : \mathrm{X} \to LCD(\mathrm{Y})$$

The triple (LCD, var, bind) is called a **monad**.

monad morphism = mapping preserving var and bind.

Monads

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

2. A collection of functions $(\mathrm{var_X} : X \to LCD(X))_X$

   Variables are expressions

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

   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

+ commutes with substitution

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

Categorical formulation

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

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

Modules VS Monads

Monad

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

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

   Variables are expressions

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

   Parallel substitution

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

4. Substitution laws:

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

Modules VS Monads

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

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

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

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

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

   Parallel substitution

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

4. Substitution laws:

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

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

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

Building blocks for binding signatures

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

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

  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.

$$id_M : M \to M$$

$$0 : 1 \to LCD$$

$$+ : LCD \times LCD \to LCD$$

$$app : LCD \times LCD \to LCD$$

$$abs : LCD' \to LCD$$

The Big Picture again

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

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

monad

module over R

A **model of Σ** is a pair:

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

monad

module morphism

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

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

Syntax

Definition

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

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

Answer: No.

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

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

Examples of 1-signatures generating syntax

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

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

  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

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

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

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

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures generate a syntax.

  Examples:

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

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

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

- [AHLM CSL 2018]: "quotients" of algebraic 1-signatures generate a syntax.
  Examples:
  - a binary commutative operation
  - application of the simple terms of differential λ-calculus (2nd variant)

$$\mathsf{app} : \mathsf{LCD} \times \mathsf{FreeCommutativeMonoid}(\mathsf{LCD}) \to \mathsf{LCD}$$

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: e.g. commutativity:

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

model of Σ

parallel pair of module morphisms over R

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

A **2-signature** is a pair

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

1-signature

set of Σ-equations

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

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

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

model of $\mathbf{\Sigma_{LC}}$ = 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:

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

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. (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:

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

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

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:

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

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

Our target: LCD

Syntax of the *differentiable λ-calculus*:

Differential λ-terms $\mathrm{s},\mathrm{t} \in \Lambda$

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

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

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

Algebraic 1-signature for LCD

Syntax of the *differentiable λ-calculus*:

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

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

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

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

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

Algebraic 1-signature for LCD

Syntax of the *differentiable λ-calculus*:

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

Corresponding 1-signature

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

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

Elementary equations for LCD

Commutative monoidal structure:

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

Linearity:

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

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

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

$$\ldots$$

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

$\mathcal{P}$ = power set monad

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

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

Example: Computing the set of free variables

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

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

$\mathcal{P}$ = 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}:

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

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.**

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

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

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

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

Example: Translating λ-calculus with fixpoint

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

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

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

λ-calculus modulo βη

monad morphism

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

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

a chosen fixpoint combinator

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 $\mathrm{fix} : LC_{\beta\eta fix}' \to LC_{\beta\eta fix}$

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

λ-calculus modulo βη

monad morphism

Definition of a translation $f : LC_{\beta\eta fix} \to LC_{\beta\eta}$ **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 $(\Sigma_{LC\beta\eta}, E_{LC\beta\eta}) + (\Sigma_{fix}, E_{fix})$:

app, abs

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

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

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

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

a chosen fixpoint combinator

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

app, abs

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

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

Example: Computing the size of a term

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

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

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

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.**

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

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

Example: Computing the size of a term

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

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 2010]: $\qquad$ 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 $(\Sigma_{LC}, \varnothing)$ $\quad\quad$ $\Sigma_{LC}(R) = (R \times R) \coprod R'$

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

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

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

Solution [CSL AHLM 2010]:

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

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

affects an arbitrary size to each variable

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

Example: Computing the size of a term

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

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 2010]:

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

1. define $f : \mathrm{LC} \to C$ 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!

Some examples covered by our result

Operations:

- Commutative binary operation

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

- Fixed point operation

Some examples covered by our result

Operations:

- Commutative binary operation

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

- Fixed point operation

More extensive examples (set of operations with equations):

- λ-calculus modulo βη

- differential λ-calculus

Module morphism VS monad morphism

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

Copie de Our target: LCD

Syntax of the *differentiable λ-calculus*:

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

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

Copie de Our target: LCD

Syntax of the *differentiable λ-calculus*:

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

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

subject to the following equation:

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

Copie de Our target: LCD

Syntax of the *differentiable λ-calculus*:

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

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

subject to the following equation:

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

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

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

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
	$B : \mathrm{Set} \to \mathrm{Set}$	$M : \mathrm{Set} \to \mathrm{Set}$
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		

Modules VS Monads

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

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