Modular specification of monads through higher-order presentations

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 *discipline* for specifying syntaxes

Main result of the paper

Our proposal $=$ a discipline for presenting monads/syntaxes

- signature = operations + equations
- [Fiore-Hur 2010]: alternative approach, for simply typed syntaxes

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

Main result: every *algebraic 2-signature* generates a syntax

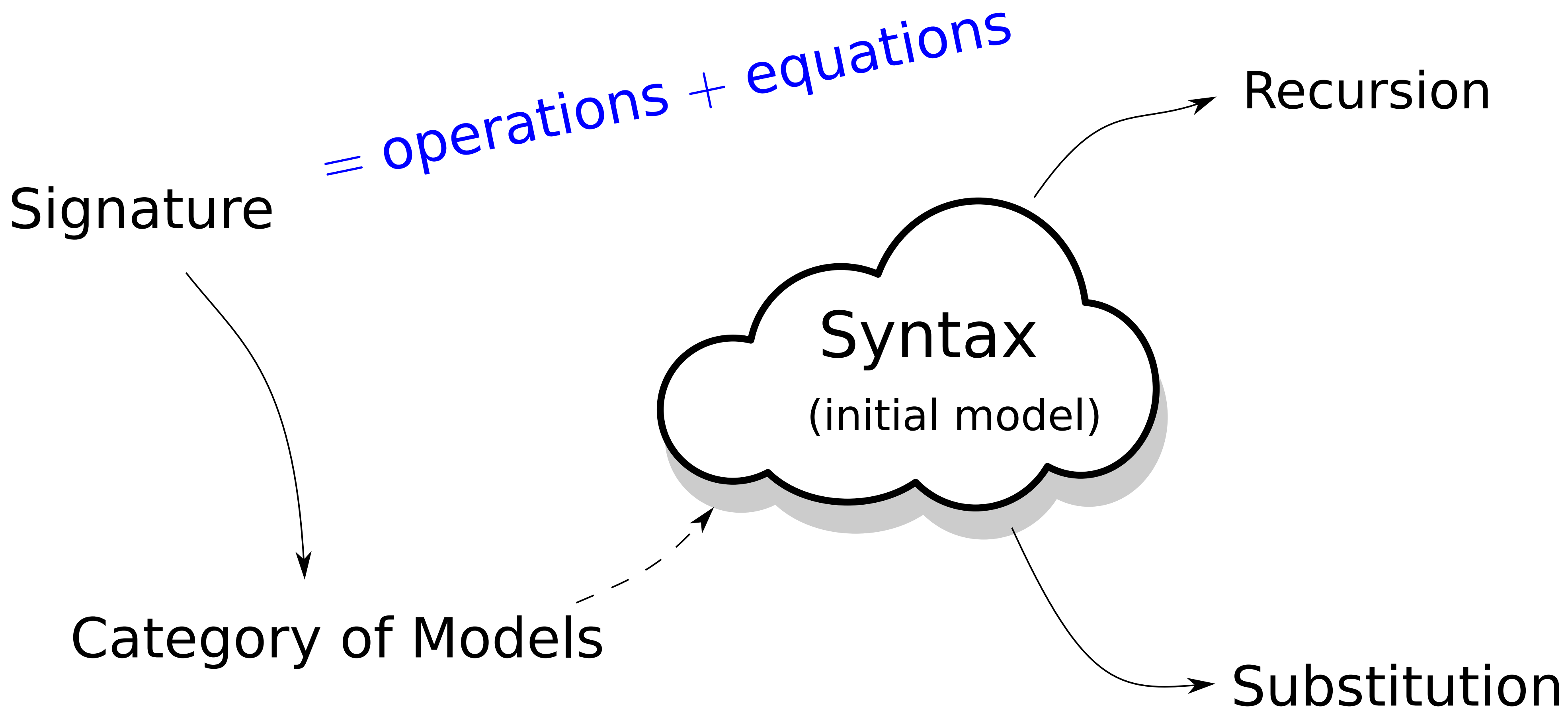
Examples

- λ-calculus modulo β- and η-equation
- ... with a fixpoint operator
- free monoid monad as a syntax:

  a binary associative operation +

  with a neutral element 0

What is a syntax?



signature generates a syntax = existence of the initial model

Table of contents

Table of contents

Substitution and monads

Example: λ-calculus

$$S,T \quad ::= \quad x \mid \lambda x.\, S \mid S\, T$$

Free variable indexing:

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

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

Substitution and monads

Example: λ-calculus

$$S, T \quad ::= \quad x \mid \lambda x.\, S \mid S\, T$$

Free variable indexing:

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

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

Parallel substitution:

$$\mathrm{t} \quad \mapsto \quad \mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})]$$

Substitution and monads

Example: λ-calculus

$$S,T \quad ::= \quad x \mid \lambda x.\, S \mid S\, T$$

Free variable indexing:

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

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

Parallel substitution:

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

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

$\Rightarrow$ (LC, var_X : X ⊂ LC(X) , bind) = **monad on $\mathbf{Set}$** [Altenkirch-Reus 99]

Substitution and monads

Example: λ-calculus

$$S,T \quad ::= \quad x \mid \lambda x.\, S \mid S\, T$$

Free variable indexing:

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

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

Parallel substitution:

$$\begin{array}{rcccl} \mathrm{bind_f} & : & LC(\mathrm{X}) & \to & LC(\mathrm{Y}) \\ & & \mathrm{t} & \mapsto & \color{blue}{\mathrm{t}[\mathrm{x} \mapsto \mathrm{f(x)}]} \end{array}$$

where $\mathrm{f} : \mathrm{X} \to \color{red}{LC(\mathrm{Y})}$

$\Rightarrow$ (LC, var_X : X ⊂ LC(X) , bind) $=$ **monad on $\mathbf{Set}$** [Altenkirch-Reus 99]

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

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

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

[Hirschowitz-Maggesi 2007 : Modules over Monads and Linearity]

Building blocks for specifying operations

module over a monad $\boldsymbol{R}$: supports the R-monadic substitution

- R itself

- $M \times N$ for any modules M and N

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

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

Building blocks for specifying operations

module over a monad R: supports the R-monadic substitution

- R itself

- $M \times N$ for any modules M and N

  e.g. $\mathrm{R} \times \mathrm{R}$: $f : X \to R(Y)$

  $(\mathrm{t,u})[\mathrm{x} \mapsto \mathrm{f(x)}] := (\mathrm{t}[\mathrm{x} \mapsto \mathrm{f(x)}], \mathrm{u}[\mathrm{x} \mapsto \mathrm{f(x)}])$

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

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

Building blocks for specifying operations

module over a monad R: supports the R-monadic substitution

- R itself
- $M \times N$ for any modules M and N

  e.g. $\mathrm{R} \times \mathrm{R}$: $f : X \to R(Y)$

  $(\mathrm{t},\mathrm{u})[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})] := (\mathrm{t}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})], \mathrm{u}[\mathrm{x} \mapsto \mathrm{f}(\mathrm{x})])$

- M' = **derivative of a module M**: $M'(X) = M(\underbrace{X \coprod}_{\text{disjoint union}} \{ \underbrace{\diamond}_{\text{fresh variable}} \})$.

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

Syntactic operations are module morphisms

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

$$\mathrm{app} : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$$

$$\mathrm{abs} : \mathrm{LC}' \to \mathrm{LC}$$

$$\begin{aligned} \mathrm{abs}_X : \mathrm{LC}(\mathrm{X} \coprod \{\diamond\}) &\to \mathrm{LC}(X) \\ t &\mapsto \lambda\diamond.t \end{aligned}$$

Syntactic operations are module morphisms

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

$$\mathrm{app} : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$$

$$\mathrm{abs} : \mathrm{LC}' \to \mathrm{LC}$$

$$\begin{aligned} \mathrm{abs}_X : \mathrm{LC}(X \coprod \{\diamond\}) &\to \mathrm{LC}(X) \\ t &\mapsto \lambda\diamond.t \end{aligned}$$

Combining operations into a single one using disjoint union

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

1-signatures and their models

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

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

Example: (app,abs)

$$\Sigma_{\mathrm{app,abs}}(R) = (R \times R) \coprod R'$$

A **model of Σ** is a pair:

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

LC $=$ model of $\Sigma_{\text{app,abs}}$

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

+ suitable notion of model morphism [Hirschowitz-Maggesi 2012]

1-signatures and their models

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

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

monad

Example: (app,abs)

$$\Sigma_{\mathrm{app,abs}}(R) = (R \times R) \coprod R'$$

A **model of Σ** is a pair:

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

LC $=$ model of $\Sigma_{\text{app,abs}}$

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

+ suitable notion of model morphism [Hirschowitz-Maggesi 2012]

1-signatures and their models

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

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

monad

module over R

Example: (app,abs)

$$\Sigma_{\text{app,abs}}(R) = (R \times R) \coprod R'$$

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

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

LC $=$ model of $\Sigma_{\text{app,abs}}$

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

+ suitable notion of model morphism [Hirschowitz-Maggesi 2012]

1-signatures and their models

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

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

monad

module over R

A **model of Σ** is a pair:

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

monad

Example: (app,abs)

$$\Sigma_{\mathrm{app,abs}}(R) = (R \times R) \coprod R'$$

LC $=$ model of $\Sigma_{\text{app,abs}}$

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

+ suitable notion of model morphism [Hirschowitz-Maggesi 2012]

1-signatures and their models

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

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

A **model of Σ** is a pair:

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

Example: (app,abs)

$$\Sigma_{\mathrm{app,abs}}(R) = (R \times R) \coprod R'$$

LC $=$ model of $\Sigma_{\text{app,abs}}$

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

+ suitable notion of model morphism [Hirschowitz-Maggesi 2012]

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.

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**: a constant 0 and a binary operation +

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

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

  Syntax: initial model

- **lambda calculus**:

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

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

  Syntax: initial model

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 the binding signatures described in [Fiore-Plotkin-Turi 1999]

(binding signature $=$ lists of natural numbers specify n-ary operations, possibly binding variables)

Question: Can we enforce some equations in the syntax ?

e.g. commutativity and associativity of a binary operation.

Quotients of algebraic 1-signatures

Theorem [AHLM CSL 2018]

Syntax exists for any "*quotient*" of algebraic 1-signature.

Example: a commutative binary operation

Quotients of algebraic 1-signatures

Theorem [AHLM CSL 2018]

Syntax exists for any "*quotient*" of algebraic 1-signature.

Example: a commutative binary operation

... what about an associative binary operation?

Table of contents

Example: a commutative binary operation

Specification of a binary operation

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

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

What is an appropriate notion of model for a commutative binary operation ?

Example: a commutative binary operation

Specification of a commutative binary operation

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

Model: $(R\,,\ + : R \times R \to R)$ s.t. $t + u = u + t$ (1)

What is an appropriate notion of model for a commutative binary operation ?

Answer: a monad equipped with a commutative binary operation

Example: a commutative binary operation

Specification of a commutative binary operation

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

Model: $(R\ ,\ +: R \times R \to R)$ s.t. $t + u = u + t$ (1)

What is an appropriate notion of model for a commutative binary operation ?

Answer: a monad equipped with a commutative binary operation

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

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

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:

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

e.g. commutativity:

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

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

e.g. commutativity:

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

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:

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

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

model of Σ

parallel pair of module morphisms over R

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:

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

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

model of Σ

parallel pair of module morphisms over R

A **2-signature** is a pair

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

1-signature

set of Σ-equations

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

- 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

Our main theorem

Syntax exists for any algebraic 2-signature.

Algebraic 2-signature:

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

algebraic 1-signature

set of **elementary** Σ-equations

Initial semantics for algebraic 2-signatures

Our main theorem

Syntax exists for any algebraic 2-signature.

Algebraic 2-signature: (Σ, E)

algebraic 1-signature

set of **elementary** Σ-equations

*a Σ-equation A $\rightrightarrows$ B is **elementary** if A maps pointwise epis to pointwise epis, and B(R) = R'..'*

Main instances of **elementary** Σ-equations A $\rightrightarrows$ B:

- $A =$ algebraic 1-signature e.g. A$(\mathrm{R}) = \mathrm{R}\times\mathrm{R}$
- $B(R) = R$

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

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

η-equation: $t = \lambda x.(t\ x)$

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

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

Example: fixpoint operator

Definition [AHLM CSL 2018]

A **fixpoint operator** in a monad R is a module morphism $\mathrm{fix}\colon \mathrm{R}' \to \mathrm{R}$

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

Intuition:

$\mathtt{fix}(t) \;:=\;$ `let rec ◇ = t in t`

Example: fixpoint operator

Definition [AHLM CSL 2018]

A **fixpoint operator** in a monad R is a module morphism $\mathrm{fix}\colon R' \to R$

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

Intuition:

$\mathtt{fix}(t) \quad := \quad$ `let rec ◇ = t in t`

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

$$\Sigma_{\mathrm{fix}}(R) := R' \qquad\qquad E_{\mathtt{fix}} = \left\{ \begin{tikzcd} \begin{matrix} R' \\ t \end{matrix} \arrow[r, bend left, "\mathtt{fix}(t)"] \arrow[r, bend right, "{t[\diamond \mapsto \mathtt{fix}(t)]}"'] & R \end{tikzcd} \right\}$$

Combining algebraic 2-signatures

Theorem

Coproducts of algebraic 2-signatures are algebraic.

fixpoint operator

λ-calculus modulo βη

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

$$=$$

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

λ-calculus modulo βη with an explicit fixpoint operator

Example: free commutative monoid

Algebraic 2-signature $(\Sigma_{\mathrm{mon}}, \mathrm{E}_{\mathrm{mon}})$ for the free commutative monoid monad:

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

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

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

Example: free commutative monoid

Algebraic 2-signature $(\Sigma_{\mathrm{mon}}, \mathrm{E}_{\mathrm{mon}})$ for the free commutative monoid monad:

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

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

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

3 elementary Σ-equations:

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

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

Conclusion

Summary of the talk:

- notion of 1-signature and models based on monads and modules
- 2-signature = 1-signature + set of equations
- *algebraic* 2-signatures generate a syntax.

Main theorems 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:

- notion of 1-signature and models based on monads and modules
- 2-signature = 1-signature + set of equations
- *algebraic* 2-signatures generate a syntax.

Main theorems formalized in Coq using the UniMath library.

Future work:

- add the notion of reductions;
- extend our work to simply typed syntaxes.

Thank you!