

Signatures and models for syntax and operational semantics in the presence of variable binding

Ambroise Lafont¹

¹DAPI
IMT Atlantique

PhD supervised by N. Tabareau and T. Hirschowitz, 2019

The subject in one slide

What is a programming language, mathematically?

- In the literature, no well-established consensus.

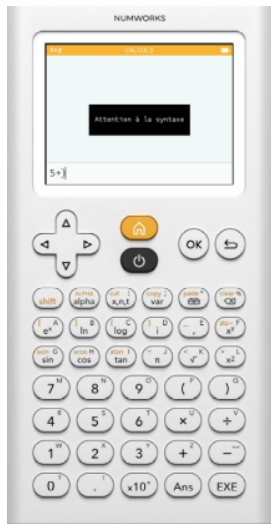
Differential λ -calculus [Ehrhard-Regnier 2003]

~10 pages (section 2 \rightarrow beginning of section 3) describing the programming language and proving some [properties](#).

- This thesis:
 - a tentative notion of programming languages, [reduction monads](#), and
 - a discipline for [automatically generating](#) well-behaved reduction monads.

What is a programming language?

Example: arithmetic expressions in a calculator



Syntax (of expressions) = formal language

- *vocabulary* : available symbols/keys
- *grammar rules* : what is a valid expression.

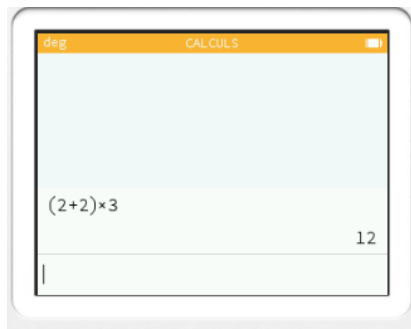
e.g. + is a *binary operation*.

What is a programming language?

Program execution

Program = valid *syntactic* text

Execution = modification of the program:

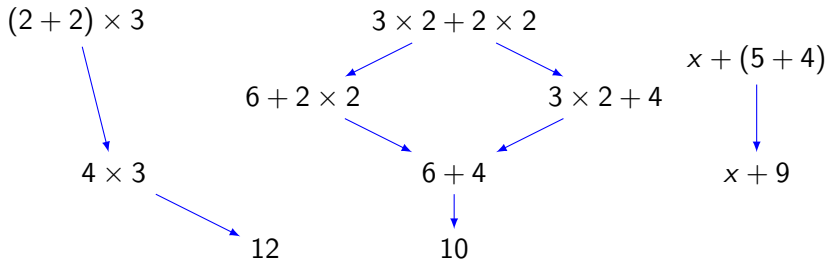


$$(2 + 2) \times 3 \xrightarrow{\text{1 execution step}} 4 \times 3 \xrightarrow{\text{1 execution step}} 12$$

Operational semantics = description of how programs execute.

What is a programming language?

A graph whose vertices are programs.



Variables = placeholders for expressions

- Substitution: $(x + (5 + 4))[x := 12] = 12 + (5 + 4)$
- Reductions are stable under substitution

$$\frac{x + (5 + 4) \rightarrow x + 9}{12 + (5 + 4) \rightarrow 12 + 9}.$$

→ Reduction monads!

Specifying programming languages: **initial semantics**

- Constructing syntax and reductions may be complex (cf. differential λ -calculus).
- Often easier to describe the **models**.

Model \approx graph with interpretation of the operations and reductions

a model of arithmetic expressions: \mathbb{Z}

- Syntactic “+” \leadsto actual “+” ,
- Syntactic “ \times ” \leadsto actual “ \times ” , ...

- Programming language = **initial** model.
- Initiality \Rightarrow **recursion principle**.

Notion of signature

- Specifies models.
- **Effective** iff the initial model exists.

A difficulty

Bound variables and α -equivalence

α -equivalence:

$x \mapsto 2 \times x$ should be identified with $y \mapsto 2 \times y$

“ x is bound by \mapsto in $x \mapsto 2 \times x$ ”

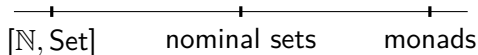
State of the art: syntax

Two main notions of syntax:

- **Substitution monoids** (\approx finitary monads) [Fiore-Plotkin-Turi, 1999].
- **Nominal sets** [Gabbay-Pitts, 1999].

wider recursion principle

more structured models



This thesis: monads

State of the art: specifying syntax

Main notions of signature for monads:

- [Pointed strong endofunctors](#) [Fiore-Plotkin-Turi, 1999].
- [Equational systems](#) [Fiore-Hur, 2010].
- [Modules](#) [Hirschowitz-Maggesi, 2007].

This thesis: modules

State of the art: semantics

Semantic notions of programming language:

- [Distributive laws](#) [Plotkin-Turi, 1997].
- [double categories](#) [Meseguer, the Montanari school].

Do not cover [higher-order](#) languages.

- [2-categories](#) [Power, Seely,...].
- [relative monads](#) [Ahrens, 2016].

Only covers [congruent](#) semantics.

Contributions

- ① Mathematical definition of programming languages as **reduction monads**.
- ② Specification of **syntactic equations**, based on modules over monads.
- ③ Specification of **semantics**.

Systematic use of monads and modules for taking care of substitution.

Articles

- CSL 2018 about 2.
- FSCD 2019 about 2. = variant of Fiore's approach.
- POPL 2020 about 1. and 3.

All in collaboration with Benedikt Ahrens, André Hirschowitz and Marco Maggesi.

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

Ingredients

- Programming languages (PLs) as graphs
 - (**Syntax**) vertices = terms
 - (**Semantics**) arrows = reductions between terms
- Simultaneous substitution: variables \mapsto terms
 - monads and modules over them

Example

λ -calculus with β -reduction:

- **Syntax:** $S, T ::= x \mid S T \mid \lambda x.S$
- Modulo α -**equivalence**, e.g.

$$\lambda x.x = \lambda y.y$$

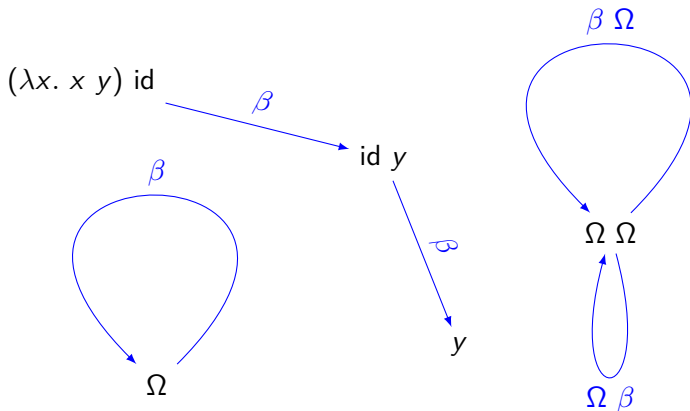
- **Reductions:** $(\lambda x.t) u \xrightarrow{\beta} t[x := u]$ + congruences

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

PLs as graphs

Example: λ -calculus with β -reduction



- **(Syntax)** vertices = terms e.g. $\Omega = (\lambda x. x x) (\lambda x. x x)$
- **(Semantics)** arrows = reductions

Graphs

Definition

Graph = a quadruple (A, V, σ, τ) where

$A = \{\text{arrows}\}$

$\sigma = \text{source of an arrow}$

$V = \{\text{vertices}\}$

$\tau = \text{target of an arrow}$

$$A \begin{array}{c} \xrightarrow{\sigma} \\ \xrightarrow{\tau} \end{array} V$$

$$\sigma : \begin{array}{c} A \\ t \xrightarrow{r} u \end{array} \rightarrow V \quad \mapsto t$$

$$\tau : \begin{array}{c} A \\ t \xrightarrow{r} u \end{array} \rightarrow V \quad \mapsto u$$

$$\sigma(r) \xrightarrow{r} \tau(r)$$

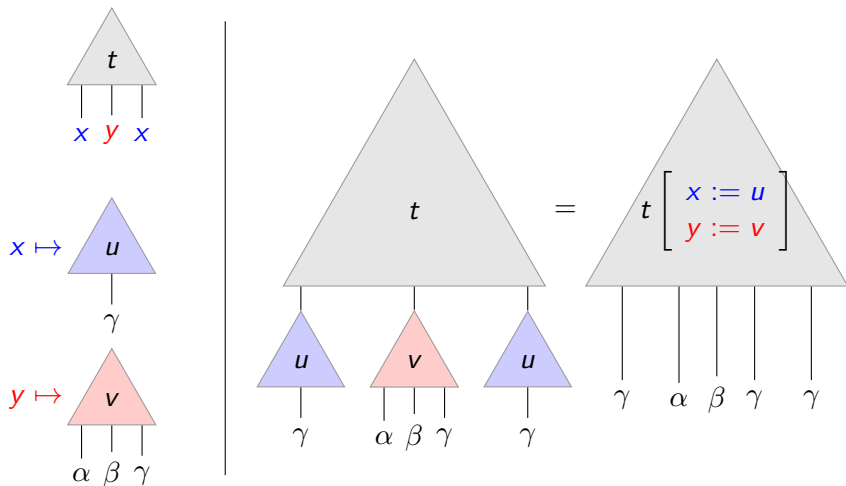
Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

Simultaneous substitution

Syntax comes with substitution

terms (e.g. λ -terms) = trees with free variables as (distinguished) leaves.



Simultaneous substitution made formal

Free variables indexing

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

Example: λ -calculus

$$L(\{x, y\}) = \left\{ \begin{array}{c} \triangle \\ \lambda z. z \end{array} , \begin{array}{c} \triangle \\ x \\ | \\ x \end{array} , \begin{array}{c} \triangle \\ y \\ | \\ y \end{array} , \begin{array}{c} \triangle \\ x \ y \\ | \quad | \\ x \quad y \end{array} , \dots \right\}$$

Simultaneous substitution

$$\forall f : X \rightarrow L(Y),$$

$$\begin{array}{lcl} L(X) & \rightarrow & L(Y) \\ t & \mapsto & t[x \mapsto f(x)] \quad (\text{or } t[f]) \end{array}$$

Monads model simultaneous substitution

λ -calculus as a monad $(L, _[_], \eta)$

① Simultaneous substitution $(L, _[_])$

② Variables are terms

$$\eta_X : X \rightarrow L(X)$$

$$x \mapsto \begin{array}{c} \triangle \\ \underline{x} \\ | \\ x \end{array}$$

③ Substitution laws:

$$\underline{x}[f] = f(x) \qquad t[x \mapsto \underline{x}] = t$$

+ associativity:

$$t[f][g] = t[x \mapsto f(x)[g]]$$

Substitution for semantics

We saw that syntax is expected to support substitution. This is also true of semantics.

Our notion of PL:

- **Syntax:** a monad $(L, _[_], \eta)$
- **Semantics:**

- graphs $R(X) \xrightleftharpoons[\tau_X]{\sigma_X} L(X)$ for each X

$R(X) =$ total set of reductions between terms taking free variables in X

- substitution of reduction: variables \mapsto **L -terms**.

$$\frac{t \xrightarrow{r} u}{t[f] \xrightarrow{r[f]} u[f]}$$

Substitution for semantics made formal

R as a **module** over L

R supports L -monadic substitution:

$$\forall f : X \rightarrow \mathbf{L}(Y), \quad \boxed{\begin{array}{l} R(X) \rightarrow R(Y) \\ r \mapsto r[x \mapsto f(x)] \quad (\text{or } r[f]) \end{array}}$$

+ substitution laws

Other examples of L -modules: L , $L \times L$, 1 , \dots

σ and τ as L -module morphisms

$$t \xrightarrow{r} u \rightsquigarrow t' \xrightarrow{r[f]} u' \quad \text{with} \quad \begin{cases} t' = t[f] \\ u' = u[f] \end{cases} \quad \text{i.e.,} \quad \begin{cases} \sigma(r[f]) = \sigma(r)[f] \\ \tau(r[f]) = \tau(r)[f] \end{cases}$$

Commutation with substitution \Leftrightarrow Module morphisms $\sigma, \tau : R \rightarrow L$.

Reduction monads

Summary: graphs + substitution.

Definition

A **reduction monad** $R \xrightarrow[\tau]{\sigma} T$ consists of

- T = monad (= module over itself)
- R = module over T
- $\sigma, \tau : R \rightarrow T$ are T -module morphisms.

Example

λ -calculus with β -reduction.

How can we specify a reduction monad?

- 1 signature for the (syntactic) operations for the monad;
- 2 reduction rules.

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

Overview

- Syntax = monad L
- Operations = module morphisms $\Sigma(L) \rightarrow L$
- 1-signatures specify operations
- 2-signatures specify operations + equations.


Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

Operations as module morphisms

For any model of λ -calculus (in particular for L),

Application commutes with substitution

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


Categorical formulation

$L \times L$ supports
 L -substitution



$L \times L$ is a **module over L**

application commutes
with substitution



$\text{app} : L \times L \rightarrow L$ is a
module morphism

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

Examples of modules

We argued that syntactic operations are **module morphisms**. Basic examples of modules?

Module over a monad T : supports the T -monadic substitution

Examples

- T itself
- $M \times N$ for any modules M and N :

$$\forall (t, u) \in M(X) \times N(X), \quad X \xrightarrow{f} T(Y),$$

$$\boxed{(t, u)[f] = (t[f], u[f])} \in M(Y) \times N(Y)$$

- $M' = \mathbf{derivative}$ of a module M :

$$X \text{ extended with a fresh variable } \diamond$$

$$M'(X) = M(\overbrace{X \amalg \{\diamond\}})$$

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

Operations as module morphisms

Operations can be combined into a single one.

Operations = module morphisms = maps commuting with substitution:

Example: λ -calculus

$$\begin{array}{ll} \text{app} : L \times L & \rightarrow L \\ \text{abs} : L' & \rightarrow L \end{array} \quad \left\{ \begin{array}{l} \text{abs}_X : L(X \amalg \{\diamond\}) \rightarrow L(X) \\ t \mapsto \lambda \diamond . t \end{array} \right.$$

Combine operations into a single one:

$$[\text{app}, \text{abs}] : (L \times L) \amalg L' \rightarrow L$$

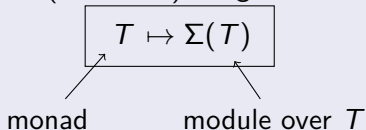
where (*coproducts* of modules M and N)

$$(M \amalg N)(X) = M(X) \amalg N(X)$$

1-signatures specify operations

Definition

A **1-signature** Σ is a (functorial) assignment



Definition (model of a 1-signature Σ)

A **model** of Σ is a pair (T, m) denoted by $\Sigma(T) \xrightarrow{m} T$ s.t.

- T is a monad
- $\Sigma(T) \xrightarrow{m} T$ is a T -module morphism

Example: λ -calculus

$$[\text{app}, \text{abs}] : \Sigma_{LC}(L) \rightarrow L \quad \text{where } \Sigma_{LC}(L) = (L \times L) \amalg L'$$

Syntax

We defined 1-signatures and their models. When is a signature effective?

(suitable notion of model morphism [Hirschowitz-Maggesi 2012])


Definition

The **syntax** specified by a 1-signature Σ is the initial object in its category of models.

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

Answer: No.

Counter-example: $\Sigma(R) = \mathcal{P} \circ R$

 Powerset endofunctor on *Set*.

(for cardinality reasons)

Initial semantics for algebraic 1-signatures

We gave examples of effective 1-signatures. They were all **algebraic**.

Definition

Algebraic 1-signatures = 1-signatures built out of derivatives, finite products, disjoint unions, and the 1-signature $\Theta : T \mapsto T$.

Algebraic 1-signatures \simeq binding signatures [Fiore-Plotkin-Turi 1999]
 \Rightarrow specification of n -ary operations, possibly binding variables.

Theorem (Fiore-Plotkin-Turi 1999)

Syntax exists for any algebraic 1-signature.

Example

λ -calculus

Question: Specify syntactic operations subject to some equations?

(*commutative associative* binary operation + of diff. λ -calculus)

Quotient of algebraic signatures

We saw that algebraic signatures are effective. Can we specify effectively operations subject to equations?

Theorem (CSL 2018)

Syntax exists for any “quotient” of algebraic 1-signatures.

Example

a *commutative* binary operation $+$:

$$\forall a, b, \quad a + b = b + a$$

What about an
associative
operation?



Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 Semantics
 - Reduction rules
 - Reduction signatures

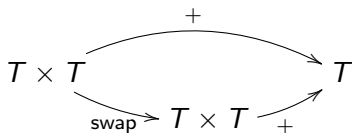
Example: a commutative binary operation

Specification of a binary operation

1-signature	$T \mapsto T \times T$
model	$\begin{array}{c} T \times T \\ \downarrow^+ \\ T \end{array}$

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

- a monad T
 - with a binary operation
 - s.t.
- } a model $T \times T \xrightarrow{+} T$ of $\Theta \times \Theta$



where $\text{swap}(t, u) = (u, t)$

Equations

$\Sigma = 1$ -signature (e.g. binary operation $\Sigma(T) = T \times T$)

Definition

A Σ -**equation** $A \xRightarrow[u]{v} B$ is a (functorial) assignment

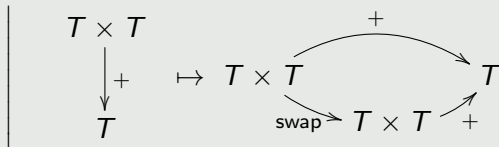
$$\boxed{M = (\Sigma(T) \rightarrow T) \mapsto \left(A(M) \xRightarrow[u_M]{v_M} B(M) \right)}$$

model of Σ

parallel pair of T -module morphisms

Example (Binary commutative operation)

$$\Sigma(T) = T \times T$$



2-signatures and their models

We defined equations. A set of equations yields a 2-signature.

Definition

A **2-signature** is a pair (Σ, E) where

- Σ is a 1-signature for monads
- E is a set of Σ -equations

Definition

A **model** of a 2-signature (Σ, E) consists of:

- a model $M = \begin{pmatrix} \Sigma(T) \\ \downarrow \\ T \end{pmatrix}$ of Σ s.t.

$$\forall A \xRightarrow[u]{v} B \in E, \quad \boxed{u_M = v_M} : A(M) \rightarrow B(M)$$

morphism of models = morphisms as models of Σ .

Initial semantics for algebraic 2-signatures

We defined 2-signatures and their models. When is a 2-signature effective?

Theorem (FSCD 2019)

Any **algebraic** 2-signature has an initial model.

Definition

A 2-signature (Σ, E) is **algebraic** if:

- Σ is algebraic
- E consists of **elementary** Σ -equations

Main instances of elementary Σ -equations

$$A \rightrightarrows B \text{ s.t. } A \left(\begin{array}{c} \Sigma(T) \\ \Downarrow \\ T \end{array} \right) = \Phi(T) \quad B \left(\begin{array}{c} \Sigma(T) \\ \Downarrow \\ T \end{array} \right) = T$$

for some *algebraic* 1-signature Φ .

(e.g. $\Phi(T) = T \times T$ for commutativity)

Example: algebraic 2-signature for differential λ -calculus

Lionel Vaux's version

$$L^d : \quad s, t ::= x \mid s \, t \mid \lambda x. s \quad \mid \quad Ds \cdot t \quad \mid \quad 0 \quad \mid \quad s + t$$

$$\Sigma_{LC^d}(T) \quad = \quad \Sigma_{LC}(T) \quad \amalg \quad T \times T \quad \amalg \quad 1 \quad \amalg \quad T \times T$$

Equations

- *associativity* and *commutativity* of $+$, neutrality of 0 for $+$
- bilinearity of $D_ \cdot _$ with respect to $+$, left linearity of application, linearity of abstraction

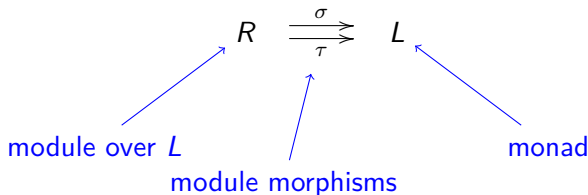
$$\lambda x. (s + t) = \lambda x. s + \lambda x. t \qquad \lambda x. 0 = 0$$

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 **Semantics**
 - Reduction rules
 - Reduction signatures

Specifying reduction monads

λ -calculus with (small-step) β -reduction as a reduction monad:



- vertices = L = initial model of the signature of λ -calculus.
- arrows = $R, \sigma, \tau = ?$
 - specified through *reduction rules* (to be made formal):

$$(\lambda x.t) u \rightarrow t[x := u] \qquad \frac{t \rightarrow t'}{t u \rightarrow t' u} \qquad \dots$$

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 **Semantics**
 - **Reduction rules**
 - Reduction signatures

Analysis of a reduction rule

Example: binary congruence for application.

metavariables: as a L -module L^4

t, t', u, u'

\mapsto

$$\frac{t \rightarrow t' \quad u \rightarrow u'}{t u \rightarrow t' u'}$$

hypotheses

conclusion

Hypothesis/conclusion = pair of λ -terms using metavariables

- as parallel module morphisms $L^4 \rightrightarrows L$
- **Generalization:** $L \leadsto$ any model $\Sigma_{LC}(T) \rightarrow T$ of Σ_{LC} :
(application denoted by $\text{app} : T \times T \rightarrow T$)

e.g., $t u \rightarrow t' u'$:

$T^4 \rightarrow T$

$(t, t', u, u') \mapsto \text{app}(t, u)$

$(t, t', u, u') \mapsto \text{app}(t', u')$

Reduction rules

Definition

Let Σ = signature for monads (e.g. Σ_{LC} for congruence for application).

Definition of Σ -reduction rules

A Σ -**reduction rule** $(\vec{\sigma}, \vec{\tau})$

$$\boxed{\frac{\sigma_1 \rightarrow \tau_1 \quad \dots \quad \sigma_n \rightarrow \tau_n}{\sigma_0 \rightarrow \tau_0}}$$

assigns (functorially) to each model $\Sigma(T) \rightarrow T$:

- $V(T) = T$ -module of metavariables (e.g. $V(T) = T^4$)
- parallel T -module morphisms $V(T) \begin{matrix} \xrightarrow{\sigma_{i,T}} \\ \xRightarrow{\tau_{i,T}} \end{matrix} T'^{\dots'}$

We write

$$\sigma_i, \tau_i : V \rightarrow \Theta^{(n_i)} \quad n_i = \text{number of derivatives}$$

Outline

- 1 Reduction monads
 - Graphs
 - Substitution
- 2 Syntax
 - Operations
 - Equations
- 3 **Semantics**
 - Reduction rules
 - **Reduction signatures**

Reduction signatures

Reduction signatures specify reduction monads.

Definition

A **reduction signature** is a pair (Σ, \mathfrak{R}) where

- Σ is a signature for monads (1- or 2-signature)
- \mathfrak{R} is a family of Σ -reduction rules

Example: λ -calculus with β -reduction

- $\Sigma = \Sigma_{LC}$
- Σ -reduction rules:
 - β -reduction
 - congruence for application and abstraction

Models

We defined **reduction signatures**. What are their models?

A **model** of a signature (Σ, \mathfrak{R}) consists of:

- a reduction monad $R \xRightarrow[\tau]{\sigma} T$ with a Σ -model structure on T
- for each reduction rule

$$\boxed{\frac{\sigma_1 \rightarrow \tau_1 \quad \dots \quad \sigma_n \rightarrow \tau_n}{\sigma_0 \rightarrow \tau_0} op} \quad V \xRightarrow[\tau_i]{\sigma_i} \Theta^{(n_i)} \quad \text{in } \mathfrak{R},$$

- a mapping, for each $v \in V(T)(X)$,

$$\begin{pmatrix} \sigma_1(v) \xrightarrow{r_1} \tau_1(v) \\ \dots \\ \sigma_n(v) \xrightarrow{r_n} \tau_n(v) \end{pmatrix} \mapsto \sigma_0(v) \xrightarrow{op(r_1, \dots, r_n)} \tau_0(v)$$

- compatible with substitution:

$$op(r_1, \dots, r_n)[f] = op(r_1[f], \dots, r_n[f])$$

Initiality

We defined **models** of a **reduction signature**. When is a signature effective?

(suitable notion of model morphism)

Theorem (POPL 2020)

Σ has an initial model (e.g. Σ is algebraic) $\Rightarrow (\Sigma, \mathfrak{R})$ has an initial model.

Examples

- λ -calculus with small-step β -reduction
- λ -ex = λ -calculus with explicit substitutions [Kesner 2009].
A Theory of Explicit Substitutions with Safe and Full Composition

Reduction signature for λ -ex

Syntax

λ -ex = λ -calcul + explicit substitution $t[x/u]$ s.t. x is bound in t :

as a module morphism $L^{ex'} \times L^{ex} \rightarrow L^{ex}$

subject to the equation

$$t[x/u][y/v] = t[y/v][x/u] \quad \text{if } y \notin fv(u) \text{ and } x \notin fv(v)$$

as a $\Sigma_{L^{ex}}$ -equation $L^{ex''} \times L^{ex} \times L^{ex} \rightrightarrows L^{ex}$.

Semantics

congruences, β -reduction $(\lambda x.t) u \rightarrow t[x/u], \dots$

$$t[x/u][y/v] \rightarrow t[y/v][x/u[y/v]] \quad \text{if } x \notin fv(u) \text{ and } y \in fv(u)$$

metavariable module: $L^{ex''} \times L^{ex} \times L_{\diamond}^{ex} \quad (L_{\diamond}^{ex} \subset L^{ex'})$

Extension of reduction monads

with associated effectivity theorem

- ① Vertices: syntax/monad \leadsto module of “configurations” over the syntax

Examples

- λ -calculus with small-step β -reduction cbv:
 - variables \mapsto **values** (rather than terms)
 - Thus, monad of **values** (rather than terms)
 - Still, reductions between **terms** (rather than values) = “configurations” over the monad of values
- π -calculus
- differential λ -calculus (without its signature though)

- ② Graph \leadsto Bipartite graph

Example

λ -calculus with big-step β -reduction cbv: term \rightarrow value.

Conclusion

Summary

- PLs as reduction monads
- Signatures for reduction monads with effectivity theorem

Perspectives

- Generalize reduction monads and their signatures
 - specify the differential λ -calculus
- Generalize on the category of sets:
 - specify simply-typed PLs: category of families of sets (indexed by simple types)
 - specify Finster-Mimram's monad of weak ω -groupoids: category of globular sets

Thank you!