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

Ambroise LAFONT[1]

[1]DAPI
IMT Atlantique

PhD, 2019

Outline

Outline

Ingredients

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

Example

λ-calculus with β-reduction:

- **Syntax:** $S, T \quad ::= x | S\ T | \lambda x.S$
- **Reductions:** $(\lambda x.t)\, u \xrightarrow{\beta} t[x \mapsto u] \quad + \quad$ congruences

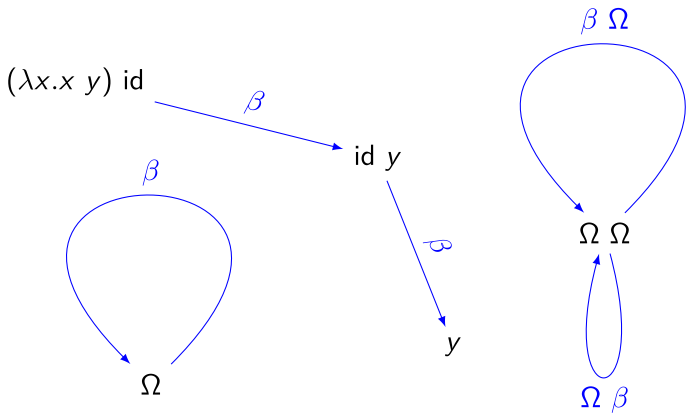
modulo α-equivalence, e.g.

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

Outline

PLs as graphs

Example: λ-calculus with β-reduction



- (**Syntax**) vertices = terms
- (**Semantics**) arrows = reductions (dedicated syntax: Cf labels)

Graphs

Definition

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

$$A \underset{\tau}{\overset{\sigma}{\rightrightarrows}} V$$

$$A = \{\text{arrows}\} \qquad V = \{\text{vertices}\}$$

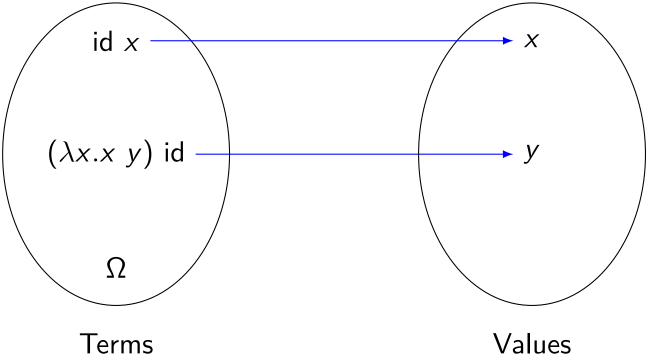
$$\sigma : \begin{array}{rl} A & \to V \\ t \xrightarrow{r} u & \mapsto t \end{array} \qquad \tau : \begin{array}{rl} A & \to V \\ t \xrightarrow{r} u & \mapsto u \end{array}$$

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

PLs as bipartite graphs

Example: λ-calculus cbv with big-step operational semantics

- term $\rightarrow$ value
- variables $=$ placeholders for values



Bipartite graphs
Definition

Bipartite graph = a quadruple (A, V_1, V_2, ∂) where

$$V_1 \xleftarrow{\sigma} A \xrightarrow{\tau} V_2$$

$$A = \{\text{arrows}\} \qquad \begin{aligned} V_1 &= \{\text{vertices in first group}\} \\ V_2 &= \{\text{vertices in second group}\} \end{aligned}$$

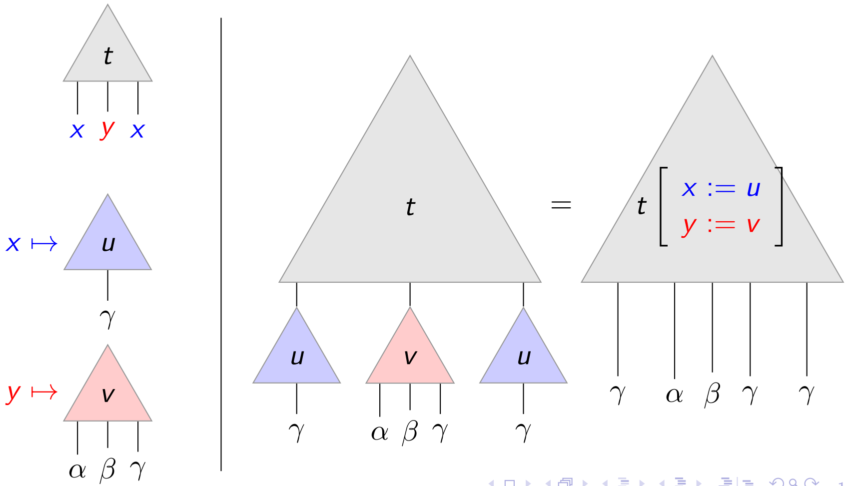
For simplicity, we focus on the particular case of **graphs**: $V_1 = V_2$.

Outline

Parallel substitution

Syntax comes with substitution

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



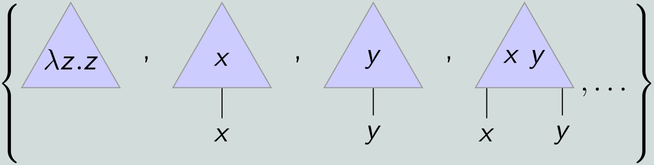
Parallel substitution made formal

Free variables indexing

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

Example: λ-calculus

$L(\{x, y\}) =$



Parallel substitution

For any $f : X \to L(Y)$,

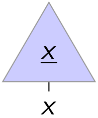
$$\begin{array}{rrcl} \mathrm{bind}_f : & L(X) & \to & L(Y) \\ & t & \mapsto & t[x \mapsto f(x)] \qquad (\text{or } t[f]) \end{array}$$

Monads

λ-calculus as a monad (L, bind, η)

1. Parallel substitution (L, bind)
2. Variables are terms

$$\begin{array}{cccc} \eta_X : & X & \to & L(X) \\ & x & \mapsto & \end{array}$$



3. Monadics laws:

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

$+$ associativity:

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

Substitution for semantics

Our notion of PL:

- **Syntax**: a monad (L, bind, η)
- **Semantics**:
  - graphs $R(X) \underset{\tau}{\overset{\sigma}{\rightrightarrows}} L(X)$ for each X

$$R(X) = \begin{array}{c}\text{total set of reductions between}\\ \text{terms taking free variables in } X\end{array}$$

  - 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

For any $f : X \to \boldsymbol{L}(Y)$,

$$\begin{aligned} \mathrm{bind}_f : \quad R(X) &\to R(Y) \\ r &\mapsto r[x \mapsto f(x)] \quad (\text{or } r[f]) \end{aligned}$$

s.t.

$$r[x \mapsto \underline{x}] = r \qquad\qquad r[f][g] = r\,[x \mapsto f(x)[g]]$$

σ and τ as L-**module morphisms**

$$\sigma(r[f]) \xrightarrow{r[f]} \tau(r[f])$$

Then, $\dfrac{\sigma(r) \xrightarrow{r} \tau(r)}{\sigma(r)[f] \xrightarrow{r[f]} \tau(r)[f]}$ enforces $\begin{array}{l} \sigma(r[f]) = \sigma(r)[f] \\ \tau(r[f]) = \sigma(r)[f] \end{array}$

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

Reduction monads

Definition

Reduction monad: a quadruple (L, R, σ, τ) s.t.

- L = monad
- R = module over L
- $\sigma, \tau : R \to L$ are L-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, involving some specified syntactic operations.

Use of a general notion of **signature** managing this dependency.

Outline

Specify reduction monads

Overview

- A signature is a sequence of arities $A_1, \ldots, A_n$

Outline

Overview

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

Outline

Operations as module morphisms

Application 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** LC

application commutes with substitution $\rightsquigarrow$ $\mathrm{app} : LC \times LC \to LC$ is a **module morphism**

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

Examples of modules

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: $M'(X) = M(X \underbrace{\coprod}_{\text{disjoint union}} \{ \underbrace{\diamond}_{\text{fresh variable}} \})$.

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

Operations as module morphisms

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

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

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

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

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:

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

monad

module over R

A **model of** Σ is a pair:

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

monad

module morphism

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.

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

  Syntax: initial model

- **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: 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.

Quotient of algebraic 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?

Outline

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

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

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

Equations

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

a Σ-**equation** A $\rightrightarrows$ B is a functorial assignment:

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

model of Σ

parallel pair of module morphisms over R

e.g. commutativity:

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

A **2-signature** is a pair

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

1-signature

set of Σ-equations

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

- a model R of Σ
- s.t. $\forall$ (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: (Σ, E)

algebraic 1-signature

set of **elementary** Σ-equations

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

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

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

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:

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

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

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

Outline

Specifying reduction monads

λ-calculus with β-reduction as a reduction monad:

$$R \;\underset{\tau}{\overset{\sigma}{\rightrightarrows}}\; L$$

module over L

module morphisms

monad

- vertices $=$ L $=$ initial model of the signature of λ-calculus.
- arrows $=$ R, σ, τ $=$?
  - **Idea**: defined inductively through reduction rules.

$$(\lambda x.t)\, u \to t[x := u] \qquad \frac{t \to t'}{t\, u \to t'\, u} \qquad \ldots$$

Model of a reduction rule

Example: binary congruence for application.

$$\boxed{\dfrac{t \to t' \qquad u \to u'}{t\ u \to t'\ u'}}$$

Model for this reduction rule

- reduction monad: $R \overset{\sigma}{\underset{\tau}{\rightrightarrows}} T$
- module morphism: $\mathrm{app} : T \times T \to T$

$\left.\right\}$ $(T, \mathrm{app}) = \text{model of } \Theta \times \Theta$

- $\forall\ t \xrightarrow{r_1} t'$ and $u \xrightarrow{r_2} u'$, a reduction

$$\mathrm{app}(t, u) \xrightarrow{\text{app-cong}(r_1, r_2)} \mathrm{app}(t', u')$$

- compatibility with substitution:

$$\text{app-cong}(r_1, r_2)[f] = \text{app-cong}(r_1[f], r_2[f])$$

Analysis of a reduction rule

Example: binary congruence for application.

metavariables: $\overbrace{t, t', u, u'}^{\text{as a } L\text{-module } L^4} \quad \mapsto \quad \boxed{\dfrac{t \to t' \qquad u \to u'}{t\,u \to t'\,u'}}$

hypotheses

conclusion

Hypothesis/conclusion = pair of λ-terms using metavariables

- as parallel module morphisms $L^4 \rightrightarrows L$

$$\text{e.g.} \quad t\,u \to t'\,u' : \qquad \begin{array}{l} (t, t', u, u') \mapsto t\,u \\ (t, t', u, u') \mapsto t'\,u' \end{array}$$

- **Generalization**: $L \mapsto$ any model T of Σ_{LC}, with application denoted $\mathrm{app} : T \times T \to T$,

$$\text{e.g.} \quad t\,u \to t'\,u' : \qquad \begin{array}{l} (t, t', u, u') \mapsto \mathrm{app}(t, u) \\ (t, t', u, u') \mapsto \mathrm{app}(t', u') \end{array}$$

Reduction rules

Definition

Let Σ = signature for monads (e.g. $\Theta \times \Theta$ for congruence for application).

Definition of Σ-reduction rules

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

$$\boxed{\dfrac{\sigma_1 \to \tau_1 \qquad \ldots \qquad \sigma_n \to \tau_n}{\sigma_0 \to \tau_0}}$$

assigns (functorially) to each Σ-model T:

- $V(T) = T$-module of metavariables (e.g. $V(T) = T^4$)
- parallel T-module morphisms $V(T) \underset{\tau_{i,T}}{\overset{\sigma_{i,T}}{\rightrightarrows}} T'^{\cdots\prime}$

We write

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

Reduction signatures

Definition

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

- Σ is a signature for monads
- $\mathfrak{R}$ is a family of Σ-reduction rules

Example: λ-calculus with β-reduction

- $\Sigma = \Theta \times \Theta + \Theta'$ for app and abs.
- Σ-reduction rules:
  - congruence for application
  - congruence for abstraction:

$$\frac{u \to u'}{\lambda x.u \to \lambda x.u'} \quad \rightsquigarrow \quad \frac{\pi_1 \to \pi_2}{\mathrm{abs} \circ \pi_1 \to \mathrm{abs} \circ \pi_2} \qquad T' \times T' \overset{\pi_{1,T}}{\underset{\pi_{2,T}}{\rightrightarrows}} T'$$

  - β-reduction

Models

Definition

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

- a reduction monad $R \overset{\sigma}{\underset{\tau}{\rightrightarrows}} T$ with a Σ-model structure on T
- for each reduction rule

$$\boxed{\dfrac{\sigma_1 \to \tau_1 \qquad \dots \qquad \sigma_n \to \tau_n}{\sigma_0 \to \tau_0}} \qquad V \overset{\sigma_i}{\underset{\tau_i}{\rightrightarrows}} \Theta^{(n_i)} \quad \text{in } \mathfrak{R},$$

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

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

  - compatible with substitution:

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

Initiality

(appropriate notion of model morphisms)

Theorem

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

Examples

- The reduction signature of the previous slide for λ-calculus with β-reduction has an initial model.
- λ-calculus with explicit substitution [Kesner 2009].

A Theory of Explicit Substitutions with Safe and Full Composition

Generalizing from graphs to bipartite graphs yields more examples:

Examples

- (big step) cbv λ-calculus.
- π-calculus

Summary

- The first main message of your talk in one or two lines.
- The second main message of your talk in one or two lines.
- Perhaps a third message, but not more than that.

- Outlook
  - What we have not done yet.
  - Even more stuff.

For Further Reading I

A. Author.
Handbook of Everything.
Some Press, 1990.

S. Someone.
On this and that.
Journal on This and That. 2(1):50–100, 2000.