

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

Ambroise LAFONT<sup>1</sup>

<sup>1</sup>DAPI  
IMT Atlantique

PhD, 2019

# Motivation

How do we formally specify a **programming language**?

In the literature: no common well-established discipline.

Differential  $\lambda$ -calculus [Ehrhard-Regnier 2003]

~10 pages (section 2  $\rightarrow$  beginning of section 3) describing the programming language<sup>a</sup> and proving some **properties**.

---

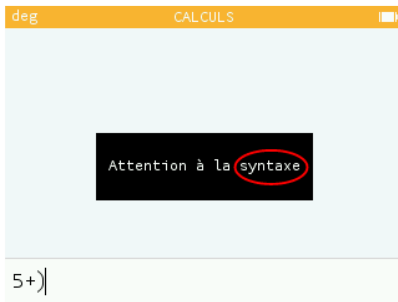
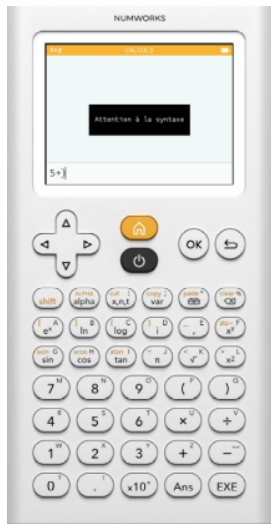
<sup>a</sup>not yet satisfyingly addressed by this PhD.

This PhD: a discipline for presenting programming languages

- from small elementary data
- automatically ensuring some **properties**

# 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*.

# Syntax and variables

Focus of this PhD

## Variables in expressions

$$(x + 5) \times y$$

$x, y = \mathbf{variables}$  = placeholders for other expressions

**Substitution:** variables  $\mapsto$  expressions:

$$\begin{cases} \text{replace } x \text{ with } 3 \\ \text{replace } y \text{ with } z \times z \end{cases} \leadsto (3 + 5) \times (z \times z)$$

## Bound variables

Example: syntax of arithmetic propositions with quantifiers.

$\exists x. x > 100$  should be identified with  $\exists y. y > 100$

“ $x$  is bound by  $\exists$  in  $\exists x. x > 100$ ”

# Syntax and recursion

**Recursion** (for syntax) = principle for investigating a piece of valid syntactic data.

## Examples of use of recursion

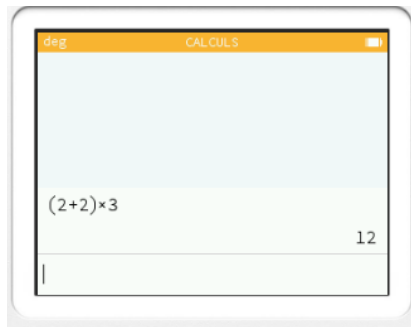
- count the number of operations in an arithmetic expression
- compute an arithmetic expression

# What is a programming language?

## Program execution

*Program* = valid *syntactic* text

*Execution* = modification of the program:



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

**Operational semantics** = description of how programs execute.

# What is a programming language?

Finally

**Programming language (PL)** = syntax + operational semantics.

**Specification** of a PL = features uniquely characterizing a PL.

In 2 steps:

- 1 syntax
- 2 semantics

Example: specification of the syntax of arithmetic expressions

- numbers = constants
- $+$  and  $\times$ : operations expecting two expressions.

**Caveat**

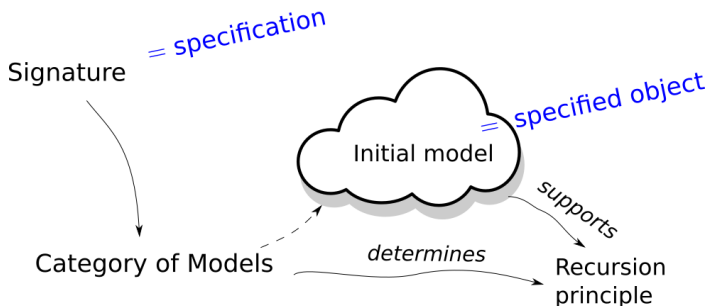
There are **ineffective specifications**: no PL satisfies them.

**Stupid example**

Syntax with two constants 0 and 1 s.t.  $0 = 1$  and  $0 \neq 1$ .

# Initial Semantics

Specification through initial semantics for justifying **recursion**.



This PhD:

- 1 Proposes a notion of **signature** with associated **category of models**, for specifying the syntax and semantics of a PL;
- 2 Rules out **ineffective** signatures: identifies a criterion ensuring existence of the initial model.



# 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
- (untyped PLs)

## Example

$\lambda$ -calculus with  $\beta$ -reduction:

• **Syntax:**  $S, T ::= x \mid S \ T \mid \lambda x. S$

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

modulo  $\alpha$ -equivalence, e.g.

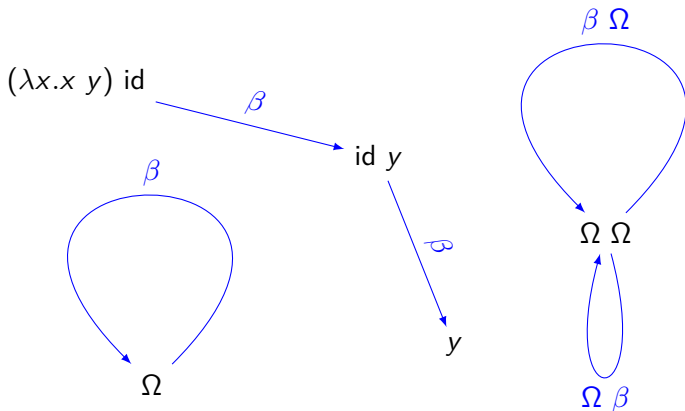
$$\lambda x. x = \lambda y. y$$

# Outline

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

# PLs as graphs

Example:  $\lambda$ -calculus with  $\beta$ -reduction



- **(Syntax)** vertices = terms e.g.  $\Omega = (\lambda x.x x) (\lambda x.x x)$
- **(Semantics)** arrows = reductions (dedicated syntax: Cf labels)

# Graphs

## Definition

Graph = a quadruple  $(A, V, \sigma, \tau)$  where

$$A \begin{smallmatrix} \xrightarrow{\sigma} \\ \xrightarrow{\tau} \end{smallmatrix} V$$

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

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

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

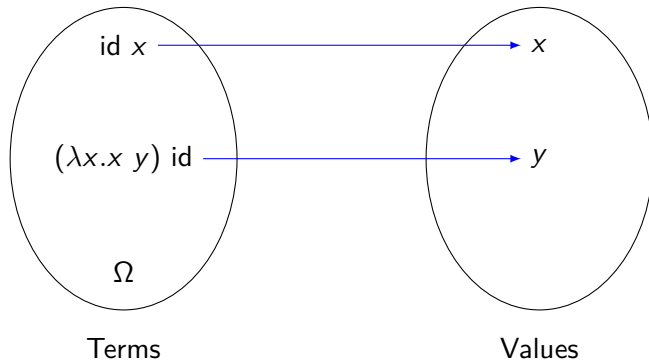
$$\tau : \begin{array}{ccc} A & \rightarrow & V \\ t \xrightarrow{r} u & \mapsto & u \end{array}$$

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

# PLs as bipartite graphs

Example:  $\lambda$ -calculus cbv with big-step operational semantics

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



# Bipartite graphs

## Definition

Bipartite graph = a quadruple  $(A, V_1, V_2, \partial)$  where

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

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

$$V_1 = \{\text{vertices in first group}\}$$

$$V_2 = \{\text{vertices in second group}\}$$

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



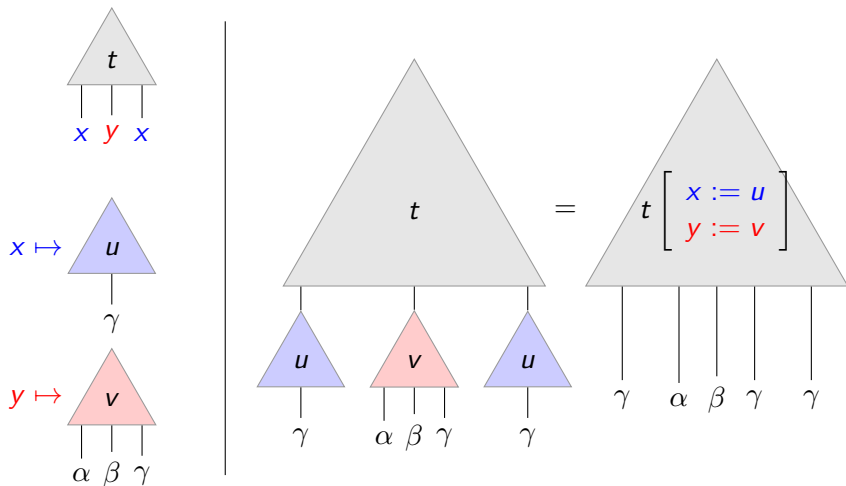
# Outline

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

# Simultaneous substitution

Syntax comes with substitution

terms (e.g.  $\lambda$ -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: $\lambda$ -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

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

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

# Monads capture simultaneous substitution

$\lambda$ -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 Monadic 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  $R(X) \xrightleftharpoons[\tau]{\sigma} L(X)$  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:

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

$$\begin{array}{l} R(X) \rightarrow R(Y) \\ r \mapsto r[x \mapsto f(x)] \quad (\text{or } t[f]) \end{array}$$

$$r[x \mapsto \underline{x}] = r$$

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

**Remark:** any monad  $T$  is a module over itself.

## $\sigma$ and $\tau$ as $L$ -**module morphisms**

By definition of  $\sigma$  and  $\tau$ ,

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

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

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

# 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

$\lambda$ -calculus with  $\beta$ -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

- 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

## 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**. Now: 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  $N$ :

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

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

- $M' =$  **derivative** of a module  $M$ :

$X$  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:  $\lambda$ -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:


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

# 1-signatures specify operations

## Definition

A **1-signature**  $\Sigma$  is a (functorial) assignment

$$T \mapsto \Sigma(T)$$



e.g.  $\Sigma_{LC}(T) = (T \times T) \amalg T'$

## Definition (model of a 1-signature $\Sigma$ )

A **model** of  $\Sigma$  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: $\lambda$ -calculus

$$[\text{app}, \text{abs}] : \Sigma_{LC}(L) \rightarrow L$$

# 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  $\Sigma$  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*.

# Examples of 1-signatures generating syntax

We saw that 1-signatures may not be effective. Examples of effective ones?

## $\lambda$ -calculus

<b>Signature</b>	$T \mapsto (T \times T) \amalg T'$
<b>Model</b>	$(T \times T) \amalg T' \rightarrow T$ , or $\left( \begin{array}{c} T \times T \rightarrow T \\ T' \rightarrow T \end{array} \right)$
<b>Syntax</b>	initial model: $(L \times L) \amalg L' \xrightarrow{[\text{app}, \text{abs}]} L$

## Language with a constant and a binary operation

<b>Signature</b>	$T \mapsto 1 \amalg (T \times T)$
<b>Model</b>	$1 \amalg (T \times T) \rightarrow T$ , or $\left( \begin{array}{c} 1 \rightarrow T \\ T \times T \rightarrow T \end{array} \right)$
<b>Syntax</b>	initial model

Can we generalize this pattern?



# 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.*

**Question:** Can we enforce some equations in the syntax?

e.g. *commutativity* or *associativity* of a binary operation.

# Quotient of algebraic signatures

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

Theorem (Ahrens-*Lafont*-Hirschowitz-Maggesi 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

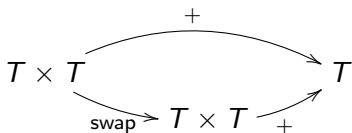
# Example: a commutative binary operation

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

# Equations

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

## Definition

A  $\Sigma$ -**equation**  $A \underset{v}{\overset{u}{\rightrightarrows}} B$  is a (functorial) assignment

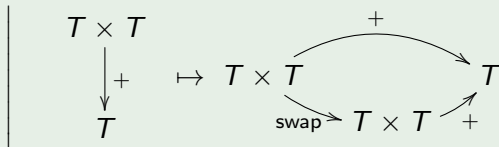
$$\boxed{M = (\Sigma(T) \rightarrow T) \mapsto \left( A(M) \underset{v_M}{\overset{u_M}{\rightrightarrows}} B(M) \right)}$$

model of  $\Sigma$

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  $(\Sigma, E)$  where

- $\Sigma$  is a 1-signature for monads
- $E$  is a set of  $\Sigma$ -equations

### Definition

A **model** of a 2-signature  $(\Sigma, E)$  consists of:

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

# Initial semantics for algebraic 2-signatures

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

Theorem (Ahrens-*Lafont*-Hirschowitz-Maggesi FSCD 2019)

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

## Definition

A 2-signature  $(\Sigma, E)$  is **algebraic** if:

- $\Sigma$  is algebraic
- $E$  consists of **elementary**  $\Sigma$ -equations

## Main instances of elementary $\Sigma$ -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  $\Phi$ .

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



# Examples of elementary equations

We saw that elementary  $\Sigma$ -equations yield effective 2-signatures. Examples of them?

- *associativity* of a binary operation
- $\beta$ -reduction as an equation:

$$(\lambda x. t) u = t[x := u]$$

- fixpoint equation

$$\lambda_{\text{rec}} x. t = t[x := \lambda_{\text{rec}} x. t]$$

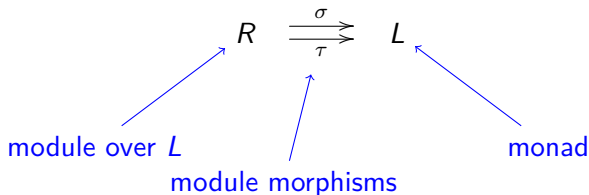
What if we want  $\beta$ -reduction as a *reduction* rather than an *equation*?

# Outline

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

# Specifying reduction monads

$\lambda$ -calculus with  $\beta$ -reduction as a reduction monad:



- vertices =  $L$  = initial model of the signature of  $\lambda$ -calculus.
- arrows =  $R, \sigma, \tau = ?$ 
  - **Idea:** specified through reduction rules.

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

# 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  $L^4$

$$\overbrace{t, t', u, u'} \mapsto$$

 $\mapsto$ 

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

hypotheses

conclusion

Hypothesis/conclusion = pair of  $\lambda$ -terms using metavariables

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

$$\text{e.g. } t u \rightarrow t' u' : \quad \begin{aligned} (t, t', u, u') &\mapsto t u \\ (t, t', u, u') &\mapsto t' u' \end{aligned}$$

- Generalization:**  $L \rightsquigarrow$  any model  $T$  of  $\Sigma_{LC}$ , with application denoted by  $\text{app} : T \times T \rightarrow T$ ,

$$\text{e.g. } t u \rightarrow t' u' : \quad \begin{aligned} (t, t', u, u') &\mapsto \text{app}(t, u) \\ (t, t', u, u') &\mapsto \text{app}(t', u') \end{aligned}$$

# Reduction rules

## Definition

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

### Definition of $\Sigma$ -reduction rules

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

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

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

- $V(T)$  =  $T$ -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

## Definition

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

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

## Example: $\lambda$ -calculus with $\beta$ -reduction

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

$$\frac{u \rightarrow u'}{\lambda x. u \rightarrow \lambda x. u'} \rightsquigarrow \frac{\pi_1 \rightarrow \pi_2}{\text{abs} \circ \pi_1 \rightarrow \text{abs} \circ \pi_2} \quad T' \times T' \xRightarrow[\pi_2, T]{\pi_1, T} T'$$

- $\beta$ -reduction



# Models

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

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

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

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

- 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?

(appropriate notion of model morphisms)

Theorem (Ahrens-Lafont-Hirschowitz-Maggesi POPL 2020)

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

## Examples

- $\lambda$ -calculus with  $\beta$ -reduction (as in the previous slide)
- $\lambda$ -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  $\lambda$ -calculus.
- $\pi$ -calculus

# Conclusion

## Summary

- PLs as reduction monads
- Signatures for reduction monads with initiality theorem
- Main theorems regarding syntax: formalized within the Coq UniMath library

## Articles

- AHLM CSL 2018 about *quotient of algebraic 1-signatures*
- AHLM FSCD 2019 about *algebraic 2-signatures*
- AHLM POPL 2020 about *reduction monads*
- HHL FoSSaCS 2020 (submitted): extension to bipartite graphs and simply typed PLs

AHLM = Ahrens, A. Hirschowitz, *Lafont*, Maggesi

HHL = A. Hirschowitz, T. Hirschowitz, *Lafont*

# 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.