

# High-level signatures and initial semantics

B. Ahrens, A. Hirshowitz, A. Lafont, M. Maggesi

# Language with substitutions

**Goal of our work:** construct the *syntax* associated to a large class of *signatures*.

Example of a "language" with variables and substitution (i.e. replacing variables with any expression yields a valid expression): formal arithmetic expressions with  $+$ ,  $\times$ , natural numbers.

$$\begin{array}{ccc} x + (y \times 3) & \xrightarrow[\substack{x \mapsto 2, y \mapsto z+5}]{\text{substitution}} & 2 + ((z + 5) \times 3) \end{array}$$

Other example: lambda-calculus

# Abstract

## Methodology

1. Introduce a notion of signature.
2. Construct an associated notion of model (suitable as domain of interpretation of the syntax generated by the signature). Such models should form a category.
3. Define the syntax generated by a signature as its initial model, when it exists.
4. Identify a class of signatures that generate a syntax: **presentable signatures**

## PLAN (TODO)

**1. Languages, monads and modules**

2. Induction and Initiality

3. Signatures

# Languages as monads

## A monad **A** as a language with variables:

- for each set  $X$ , a set  $A(X)$  of expressions taking free variables in  $X$ .
- any variable  $x \in X$  is a valid expression that we note  $\text{var}_X(x) = \underline{x} \in A(X)$
- given a family  $(t_x)_{x \in X}$  of expressions in  $A(Y)$ , we can perform for any expression **e** in **A(X)** the substitution  $e[x \mapsto t_x]$  lying in  $A(Y)$

Three monadic laws:

COMPOSITION OF SUBSTITUTIONS     $e[x \mapsto t_x][y \mapsto u_y] = e[x \mapsto t_x[y \mapsto u_y]]$

IDENTITY SUBSTITUTION     $e[x \mapsto x] = e$

VARIABLE SUBSTITUTION     $\forall x \in X \quad x[y \mapsto t_y] = t_x$

# Examples of monads

## Some other examples of monads:

- the (untyped) syntax of lambda-calculus  $L$  (*modulo alpha equivalence*)

$$\begin{array}{ll} \text{expr} ::= x & (\text{variable}) \\ & | t\ u & (\text{application}) \\ & | \lambda x.t & (\text{abstraction}) \end{array}$$
$$L(\emptyset) = \{\text{closed terms}\} = \{ \lambda x.x, \lambda x.\lambda y.(x\ y), (\lambda x.x\ x)(\lambda x.x\ x), \dots \}$$
$$L(\{z\}) = \{z, \lambda x.z, \lambda x.(x\ z), \dots\} \cup L(\emptyset)$$

$$(\lambda x.x\ z)[z \mapsto \lambda y.y] = \lambda x.x(\lambda y.y)$$

- the (untyped) syntax of lambda-calculus modulo beta-reduction and eta-expansion

# Examples of monads

## Some other examples of monads:

- the assignement  $X \mapsto \mathcal{P}(X) = \{ U \mid U \subset X \}$  yields a monad  $\mathcal{P}$ .

$$\forall X, \text{var}_X : X \rightarrow \mathcal{P}(X)$$
$$x \mapsto \{x\}$$

Let  $U \subset X$  (i.e.  $U \in \mathcal{P}(X)$ ) and  $(V_x)_{x \in X}$  a family of subsets of  $Y$ .

Substitution is defined as union:

$$U[x \mapsto V_x] = \bigcup_{x \in U} V_x \in \mathcal{P}(Y)$$

# Operations as module morphisms

## Arithmetic operations as module morphisms:



For each set  $X$ , the sum of two expressions  $e, e' \in A(X)$  take free variables in  $X$ :

$$\begin{aligned}\forall X, \text{ add}_X : A(X) \times A(X) &\rightarrow A(X) \\ (e, e') &\mapsto e + e'\end{aligned}$$

Note that:

$$(e + e')[x \mapsto t_x] = e[x \mapsto t_x] + e'[x \mapsto t_x]$$

We characterize this situation as follows:

$A(X) \times A(X)$ has a notion of substitution		$A \times A$ is a <b>module</b> on $A$
$\text{add}$ commutes with substitution		$\text{add}$ is a <b>module morphism</b>



# Module over a monad

## Substitution on $A \times A$ :

Let  $(t_x)_{x \in X}$  be a family of expressions in  $A(Y)$ :  $t : X \rightarrow A(Y)$

Then we can define substitution on  $A(X) \times A(X)$ :

$$\begin{aligned} \langle t \rangle : A(X) \times A(X) &\rightarrow A(Y) \times A(Y) \\ (e, e') &\mapsto (e, e')[x \mapsto t_x] := (e[x \mapsto t_x], e'[x \mapsto t_x]) \end{aligned}$$

that inherit some properties of substitution on  $A$ :

- **(identity substitution)**  $(e, e')[x \mapsto x] = (e, e')$
- **(composition of substitutions)** for any other family  $(u_y)_{y \in Y}$  of expressions in  $A(Z)$ ,  $(e, e')[x \mapsto t_x][y \mapsto u_y] = (e, e')[x \mapsto t_x[y \mapsto u_y]]_A$

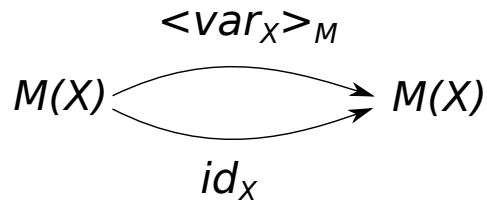
This is an example of a module over the monad  $A$

# Module over a monad

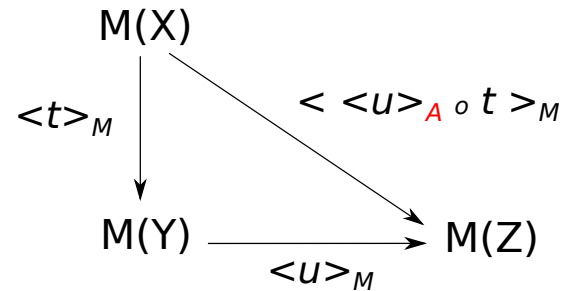
## Module over a monad:

A module over the monad  $A$ :

- associates a set  $M(X)$  to any set  $X$ :  $M(X)$  can be thought of as "generalized" expressions taking variables in  $X$ .
- is equipped, given any family  $(t_x)_{x \in X}$  of elements of  $A(Y)$ , with a substitution  $\langle t \rangle_M : M(X) \rightarrow M(Y)$  satisfying:



IDENTITY SUBSTITUTION



COMPOSITION OF SUBSTITUTIONS

# Examples of modules

## Modules over a monad:

Some examples of modules over a monad **R**:

- **R** itself (already satisfies identity substitution and composition of substitution by definition of a monad)
- **R x R** (i.e. the assignement  $X \mapsto R(X) \times R(X)$ )
- **M x N** for any module M and N

## Important example : Derivative of a module

- $X \mapsto R(X + \{n\})$  where  $n \notin X$  yields a module denoted by **R'**
- more generally, we similarly define **M'** given a module **M**

# Module morphism

## Module morphism:

Let **M** and **N** be two modules over a monad **R**. A module morphism between **M** and **N** is a family of maps  $(f_x: M(X) \rightarrow N(X))_x$  that *commutes with substitution*: for any  $e \in M(X)$  and family  $(t_x)_{x \in X}$  of elements of  $M(Y)$ ,

$$f_X(e)[x \mapsto t_x]_N = f_Y(e[x \mapsto y_x]_M)$$

$$\begin{array}{ccc} M(X) & \xrightarrow{\langle t \rangle_M} & M(Y) \\ f_X \downarrow & & \downarrow f_Y \\ N(X) & \xrightarrow{\langle t \rangle_N} & N(Y) \end{array}$$

## Example:

$$add : A \times A \rightarrow A$$

$$add(e, e')[x \mapsto t_x] = add(e[x \mapsto t_x], e'[x \mapsto t_x])$$

# Examples of module morphisms

## Some module morphisms:

- **id<sub>M</sub> : M → M** denoting the family of identity maps  $(id_{M(X)} : M(X) \rightarrow M(X))_X$  for any module **M**
- **app : L x L → L** denoting the application operation of the lambda calculus monad L:  $app(t, u) = t u$
- What about the abstraction operation  $abs : t \mapsto \lambda x. t$  of lambda calculus?

## Binding variables:

In  $\lambda x. t$ , the term  $t$  depends on an additional free variable  $x$ :  
If  $\lambda x. t \in L(Y)$ , then  $t \in L(Y + \{x\}) = \mathbf{L}'(\mathbf{Y})$

**abs: L' → L** is a module morphism

## PLAN

1. Languages, monads and modules

**2. Induction and Initiality**

3. Signatures

# Induction

## Example: computing the free variables of a lambda-term

We compute it by induction on the syntax:

$$fv(x) = \{x\} \quad \text{(variable)}$$

$$fv(tu) = fv(t) \cup fv(u) \quad \text{(application)}$$

$$fv(\lambda x.t) = fv(t) \setminus \{x\} \quad \text{(abstraction)}$$

This is formalized in our setting as a family of maps  $(fv_x: L(X) \rightarrow \mathcal{P}(X))_x$  which *commutes with variable and substitution*:

$$\begin{aligned} fv(var_L(x)) &= \{x\} \\ &= var_{\mathcal{P}}(x) \end{aligned} \qquad \begin{aligned} fv(u[x \mapsto t_x]_L) &= \bigcup_{y \in fv(u)} t_y \\ &= fv(u)[x \mapsto fv(t_x)]_{\mathcal{P}} \end{aligned}$$

(This is a definition of a monad morphism)

# Induction

## Example: computing the free variables of a lambda-term

*fv* also commutes with 'application' and 'abstraction'

$$\begin{aligned} app_{\mathcal{P}} : \mathcal{P} \times \mathcal{P} &\rightarrow \mathcal{P} \\ (V, V') &\mapsto V \cup V' \end{aligned}$$

$$\begin{aligned} abs_{\mathcal{P}, X} : \overbrace{\mathcal{P}'(X)}^{\mathcal{P}(X + \{n\})} &\rightarrow \mathcal{P} \\ V &\mapsto V \setminus \{n\} \end{aligned}$$

Actually, these commutations **define** *fv* uniquely by induction:

$$fv(x) = \{x\} \quad \text{(commutation with variable)}$$

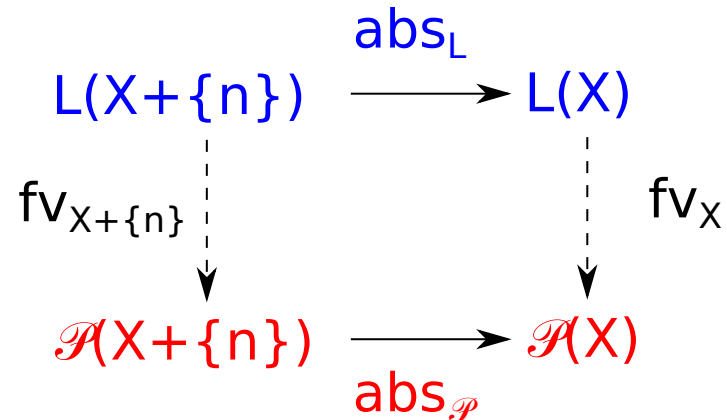
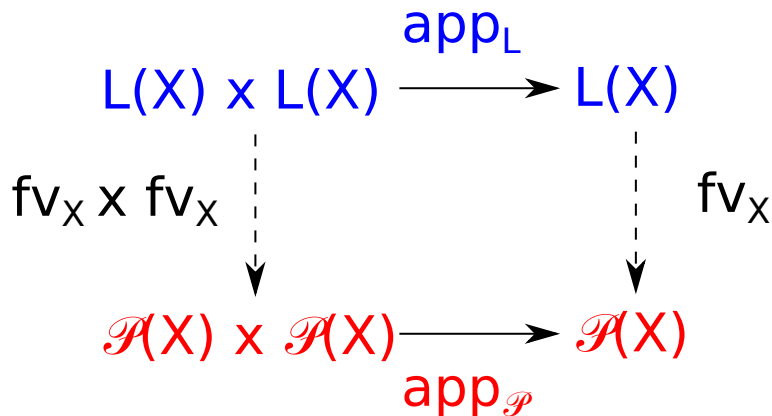
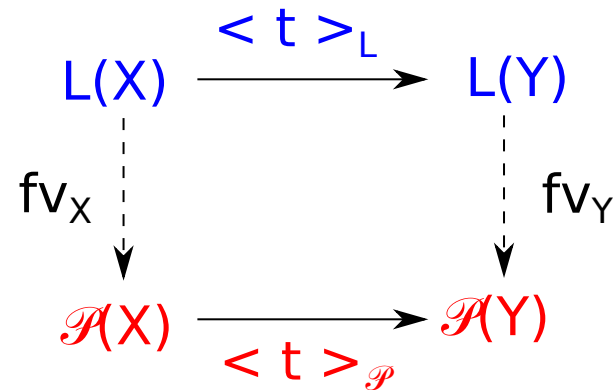
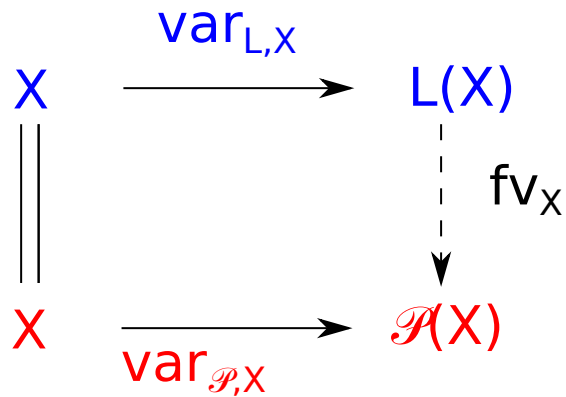
$$fv(tu) = fv(t) \cup fv(u) \quad \text{(commutation with application)}$$

$$fv(\lambda x.t) = fv(t) \setminus \{x\} \quad \text{(commutation with abstraction)}$$



# Induction and initiality

$fv$  is the unique family of maps that makes the following diagrams commute:



# Induction and initiality

More generally, let  $R$  be a monad with application and abstraction.

$$X \xrightarrow{\text{var}_{R,X}} R(X)$$

$$R(X) \xrightarrow{\langle t \rangle_R} R(Y)$$

$$R(X) \times R(X) \xrightarrow{\text{app}_R} R(X)$$

$$R(X + \{n\}) \xrightarrow{\text{abs}_R} R(X)$$

# Induction and initiality

More generally, let  $R$  be a monad with application and abstraction. Then there is a unique family  $(f_x)_x$  of maps (defined by induction) that makes the following diagrams commute:

$$\begin{array}{ccc}
 X & \xrightarrow{\text{var}_{L,X}} & L(X) \\
 \parallel & & \downarrow f_X \\
 X & \xrightarrow{\text{var}_{R,X}} & R(X)
 \end{array}$$

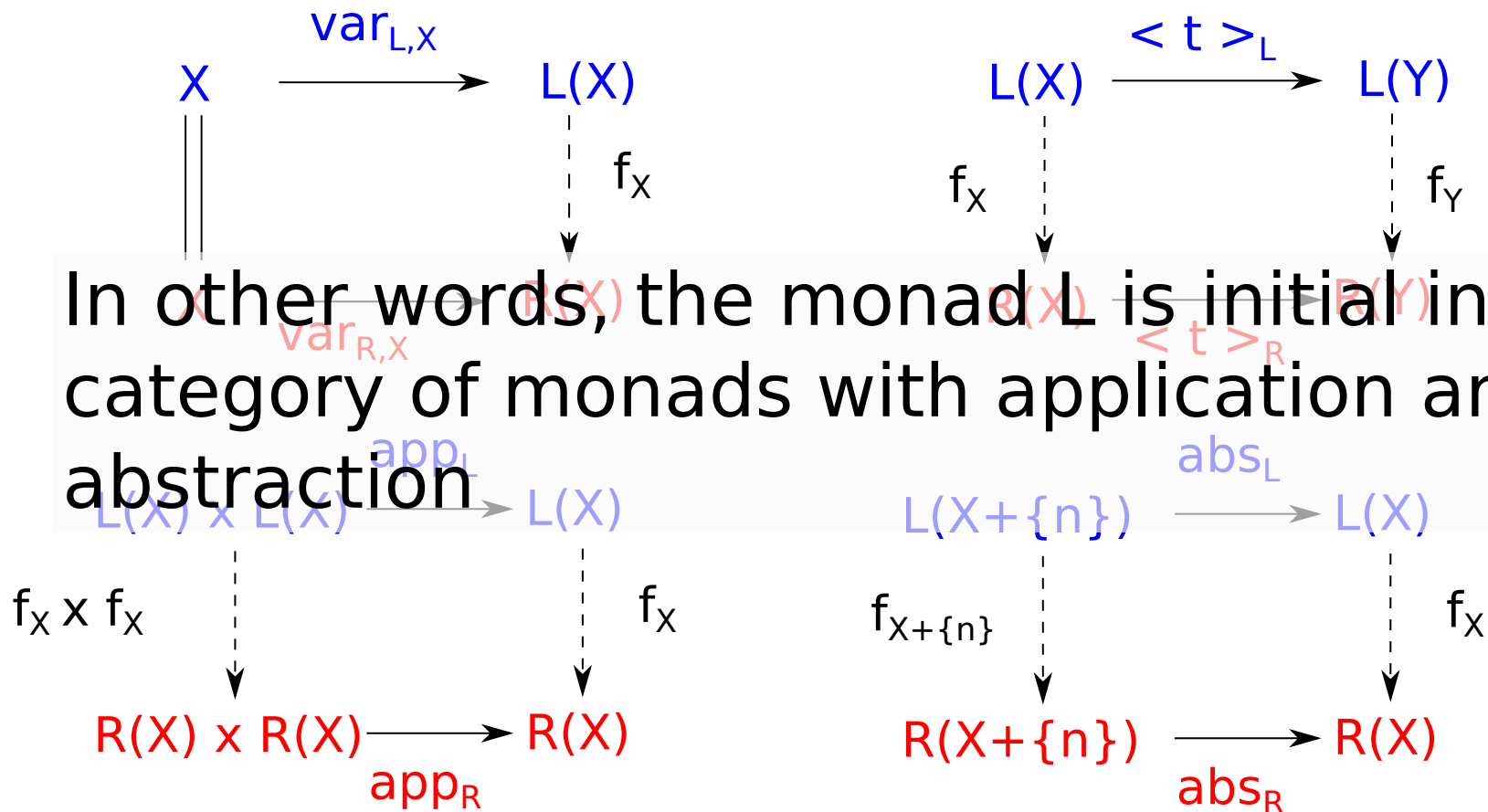
$$\begin{array}{ccc}
 L(X) & \xrightarrow{\langle t \rangle_L} & L(Y) \\
 \downarrow f_X & & \downarrow f_Y \\
 R(X) & \xrightarrow{\langle t \rangle_R} & R(Y)
 \end{array}$$

$$\begin{array}{ccc}
 L(X) \times L(X) & \xrightarrow{\text{app}_L} & L(X) \\
 \downarrow f_X \times f_X & & \downarrow f_X \\
 R(X) \times R(X) & \xrightarrow{\text{app}_R} & R(X)
 \end{array}$$

$$\begin{array}{ccc}
 L(X + \{n\}) & \xrightarrow{\text{abs}_L} & L(X) \\
 \downarrow f_{X+\{n\}} & & \downarrow f_X \\
 R(X + \{n\}) & \xrightarrow{\text{abs}_R} & R(X)
 \end{array}$$

# Induction and initiality

More generally, let  $R$  be a monad with application and abstraction. Then there is a unique family  $(\mathbf{f}_X)_X$  of maps (defined by induction) that makes the following diagrams commute:



# Syntax and initiality

## A definition of a syntax:

A **syntax** is a monad that comes with an *induction principle*, i.e. which is initial in a suitable category of *monads + operations that it implements*.

## Example:

The monad  $L$  of lambda calculus is initial in the category of *monads + application and abstraction*.

We say that  $L$  is the **syntax generated by the signature of application and abstraction**.

We will now present a general definition of **signatures**.

# Signatures

## What a signature should be:

$L$  is initial among the monads  $R$  that model the signature  $\Sigma_L$  of application and abstraction, i.e. monads  $R$  that come with module morphisms:

$$app_R : R \times R \rightarrow R$$

$$abs_R : R' \rightarrow R$$

or  $[app_R, abs_R] : R \times R + \underbrace{R'}_{\Sigma_L(R)} \rightarrow R$



A syntax  $S$  is initial among the monads  $R$  that model its associated signature  $\Sigma$ , i.e. monads  $R$  that come with a module morphism:

$$\sigma_R : \Sigma_R \rightarrow R$$

Thus, a signature  $\Sigma$  should assign to any monad  $R$  a module  $\Sigma_R$  over it.

# Signatures

Let  $\mathbf{R}$  be a monad that models the signature of application and abstraction. Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{L} \rightarrow \mathbf{R}$  which commutes with abstraction and application:

$$\begin{array}{ccc}
 L(X) \times L(X) & \xrightarrow{\text{app}_L} & L(X) \\
 \downarrow \mathbf{f}_X \times \mathbf{f}_X & & \downarrow \mathbf{f}_X \\
 R(X) \times R(X) & \xrightarrow{\text{app}_R} & R(X)
 \end{array}$$

$$f_X(\text{app}_L(t, u)) = \text{app}_R(f_X(t), f_X(u))$$



(and similarly for abs)

$$f_X(\text{abs}_L(t)) = \text{abs}_R(f_{X+\{n\}}(t))$$

Let  $\mathbf{R}$  be a monad that models a signature  $\Sigma$  (there is a module morphism  $\sigma_R : \Sigma_R \rightarrow \mathbf{R}$ ). Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{S} \rightarrow \mathbf{R}$  which commutes with  $\sigma$ :

$$\begin{array}{ccc}
 \Sigma_L(X) & \xrightarrow{\sigma_L} & L(X) \\
 \downarrow \text{??} & & \downarrow \mathbf{f}_X \\
 \Sigma_R(X) & \xrightarrow{\sigma_R} & R(X)
 \end{array}$$

# Signatures

Let  $\mathbf{R}$  be a monad that models the signature of application and abstraction. Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{L} \rightarrow \mathbf{R}$  which commutes with abstraction and application:

$$\begin{array}{ccc}
 L(X) \times L(X) & \xrightarrow{\text{app}_L} & L(X) \\
 \downarrow \mathbf{f}_X \times \mathbf{f}_X & & \downarrow \mathbf{f}_X \\
 R(X) \times R(X) & \xrightarrow{\text{app}_R} & R(X)
 \end{array}$$

$$f_X(\text{app}_L(t, u)) = \text{app}_R(f_X(t), f_X(u))$$



(and similarly for abs)

$$f_X(\text{abs}_L(t)) = \text{abs}_R(f_{X+\{n\}}(t))$$

Let  $\mathbf{R}$  be a monad that models a signature  $\Sigma$  (there is a module morphism  $\sigma_R : \Sigma_R \rightarrow \mathbf{R}$ ). Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{S} \rightarrow \mathbf{R}$  which commutes with  $\sigma$ :

$$\begin{array}{ccc}
 \Sigma_L(X) & \xrightarrow{\sigma_L} & L(X) \\
 \downarrow \Sigma(\mathbf{f})_X & & \downarrow \mathbf{f}_X \\
 \Sigma_R(X) & \xrightarrow{\sigma_R} & R(X)
 \end{array}$$



# Signatures

Let  $\mathbf{R}$  be a monad that models the signature of application and abstraction. Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{L} \rightarrow \mathbf{R}$  which commutes with abstraction and application. Thus, a signature  $\Sigma$  assigns to any monad morphism  $\mathbf{f} : \mathbf{R} \rightarrow \mathbf{R}'$  a family of maps  $(\Sigma(\mathbf{f})_X : \Sigma_R(X) \rightarrow \Sigma_{R'}(X))_X$ .

As for module morphisms, we require that this family commutes with substitution:

$$\Sigma(\mathbf{f})_Y(e[x \mapsto t_x]_{\Sigma_R}) = \Sigma(\mathbf{f})_X(e)[x \mapsto f_X(t_x)]_{\Sigma'_R}$$

Let  $\mathbf{R}$  be a monad that models a signature  $\Sigma$  (there is a module morphism  $\sigma_R : \Sigma_R \rightarrow \mathbf{R}$ ). Then there exists a unique monad morphism  $\mathbf{f} : \mathbf{S} \rightarrow \mathbf{R}$  which commutes with  $\sigma$ :

$$\begin{array}{ccc} \Sigma_L(X) & \xrightarrow{\sigma_L} & L(X) \\ \Sigma(\mathbf{f})_X \downarrow & & \downarrow f_X \\ \Sigma_R(X) & \xrightarrow{\sigma_R} & R(X) \end{array}$$

## PLAN

1. Languages, monads and modules
2. Induction and Initiality
- 3. Signatures**

# Definition of signatures

A **signature**  $\Sigma$  is given by:

- for each monad  $R$ , a module  $\Sigma_R$  over it
- for each monad morphism  $f : R \rightarrow S$ , a family  $\Sigma(f) : \Sigma_R \rightarrow \Sigma_S$  of morphisms which commutes with substitution:

$$\Sigma(f)_Y(e[x \mapsto t_x]_{\Sigma_R}) = \Sigma(f)_X(e)[x \mapsto f_X(t_x)]_{\Sigma'_R}$$

- such that (functoriality)

$$\Sigma(f \circ g) = \Sigma(f) \circ \Sigma(g) \quad \text{and} \quad \Sigma(id_R) = id_{\Sigma_R}$$

A **model** of a signature  $\Sigma$  is a monad  $R$  together with a morphism of modules  $\sigma_R : \Sigma_R \rightarrow R$

A **model morphism** of a signature  $\Sigma$  between two models  $R$  and  $R'$  is a monad morphism  $f : R \rightarrow S$  which commutes with  $\sigma$ :  $\sigma_R \circ f = \Sigma_f \circ \sigma_{R'}$

The **syntax generated by** a signature  $\Sigma$  is its initial model.

# Syntax generated by a signature

This notion of signature is very general so that we do not expect that all of them generate a syntax.

## Examples of syntax generating signatures:

- $R \mapsto R \times R$ :

models are monads  $R$  that comes with a module morphism  $R \times R \rightarrow R$ .

The syntax corresponds to a language with variables and a binary

operator  $b$ :      $\text{expr} ::= x$             *(variable)*  
                        |  $b(t, u)$     *where  $t$  and  $u$  are any expressions*

$$- R \mapsto R \times R + R':$$

By universal property of the disjoint sum  $+$ , models are monads  $R$  equipped with two modules morphisms  $R \times R \rightarrow R$  and  $R' \rightarrow R$ .

## The syntax corresponds to lambda calculus

# Syntax generated by a signature

This notion of signature is very general so that we do not expect that all of them generate a syntax.

## Examples of syntax generating signatures:

- $R \mapsto R \times R$ :

models are monads  $R$  that comes with a module morphism  $R \times R \rightarrow R$ .

The syntax corresponds to a language with variables and a binary

operator  $b$ :      $\text{expr} ::= x$             *(variable)*  
                        |  $b(t, u)$     *where  $t$  and  $u$  are any expressions*

$$- R \mapsto R \times R + R':$$

By universal property of the disjoint sum  $+$ , models are monads  $R$  equipped with two modules morphisms  $R \times R \rightarrow R$  and  $R' \rightarrow R$ .

## The syntax corresponds to lambda calculus

# Algebraic signatures

More generally, any signature of the form  $R \mapsto R' \times R'' \times R''' + R \times R'' \times R''' \times R + \dots$  (i.e. any disjoint sum of products of finite derivatives of the monad) generates a syntax. We call them **algebraic signatures**: they correspond to languages with n-ary operations that can bind a finite number of variables in their arguments.

**Our main result:** quotients of algebraic signatures also generate a syntax

## Example:

-  $R \mapsto (R \times R) / S_2$  associates to any monad  $R$  the module of its unordered pairs. Models (in particular the syntax) are monads equipped with a binary *commutative* operation.

# Algebraic signatures

More generally, any signature of the form  $R \mapsto R' \times R'' \times R''' + R \times R'' \times R''' \times R + \dots$  (i.e. any disjoint sum of products of finite derivatives of the monad) generates a syntax. We call them **algebraic signatures**: they correspond to languages with n-ary operations that can bind a finite number of variables in their arguments.

**Our main result:** quotients of algebraic signatures also generate a syntax

## Example:

-  $R \mapsto (R \times R) / S_2$  associates to any monad  $R$  the module of its unordered pairs. Models (in particular the syntax) are monads equipped with a binary *commutative* operation.

# Quotient of a signature

## Quotient of a set:

A quotient of a set  $X$  is a set  $Y$  together with a surjection  $f : X \rightarrow Y$ .  
( $x \sim x'$  iff  $f(x) = f(y)$ ).

## Quotient of a signature:

A quotient of a signature  $\Sigma$  is a signature  $\Psi$  together with a family of module morphisms  $(f_R : \Sigma_R \rightarrow \Psi_R)_R$  that is pointwise surjective and commutes with any monad morphism  $m : R \rightarrow R'$  in the sense that:

$$f_{R'} \circ \Sigma(m) = \Psi(m) \circ f_R \quad (\text{naturality condition})$$



# Quotient of a signature

## Quotient of a set:

A quotient of a set  $X$  is a set  $Y$  together with a surjection  $f : X \rightarrow Y$ .  
( $x \sim x'$  iff  $f(x) = f(y)$ ).

## Quotient of a signature:

A quotient of a signature  $\Sigma$  is a signature  $\Psi$  together with a family of module morphisms  $(f_R : \Sigma_R \rightarrow \Psi_R)_R$  that is pointwise surjective and commutes with any monad morphism  $m : R \rightarrow R'$  in the sense that:

$$f_{R'} \circ \Sigma(m) = \Psi(m) \circ f_R \quad (\text{naturality condition})$$

# Quotients of algebraic signatures

**Theorem:** Let  $\mathbf{S}$  be the syntax generated by an algebraic signature  $\Sigma$ . Then any quotient  $\Psi$  of  $\Sigma$  generates a syntax (obtained by quotienting adequately the syntax  $\mathbf{S}$ )

**Examples of quotient algebraic signatures:**

TODO

# Quotients of algebraic signatures

**Theorem:** Let  $\mathbf{S}$  be the syntax generated by an algebraic signature  $\Sigma$ . Then any quotient  $\Psi$  of  $\Sigma$  generates a syntax (obtained by quotienting adequately the syntax  $\mathbf{S}$ )

**Examples of quotient algebraic signatures:**

TODO