Initial semantics for lambda calculi

Benedikt Ahrens

Initial semantics

Methodology for defining/characterizing a language:

1. Introduce a notion of signature.
2. Construct an associated notion of model. Such models should form a category.
3. Define the **syntax generated by a signature** to be its initial model, when it exists.
4. Find a satisfactory sufficient condition for a signature to generate a syntax.

From initiality one can derive a **recursion principle** for defining maps out of the syntax.

This talk

Joint work with

- André Hirschowitz
- Ambroise Lafont
- Marco Maggesi

Based on work reported in

- *High-level signatures and initial semantics*, CSL 2018
- *Modular specification of monads through higher-order presentations*, FSCD 2019
- *Reduction monads and their signatures*, under review

Contents:

- Signature for untyped languages with equations and reduction rules
- Model of a signature
- Some sufficient conditions for signatures to generate a syntax

Outline

An ubiquitous datatype

```
datatype list A =     -- lists over type A
| nil                 -- nil : list A
| cons A (list A)     -- cons : A -> list A -> list A
```

- A *model of lists over A* is a triple `(X,n,c)` with `n:X` and `c:A -> X -> X`
- Any model gives rise to a structure-preserving function

  ```
  fold : X -> (A -> X -> X) -> (list A -> X)
  ...
  ```

Initiality for lists

Theorem (Initiality for lists over A)

For any model (X, n, c) of lists over A there is exactly one structure-preserving map

$$(\mathsf{list}(A), \mathsf{nil}, \mathsf{cons}) \to (X, n, c)$$

That is, the requirement of preserving structure specifies that map uniquely.

Why is Initiality useful?

`fold` takes a model as input and returns a function from the type of lists to the model.

Use of `fold` is preferred over writing recursive functions by pattern matching oneself:

- Recursion encapsulated in the definition of `fold`
- Abstracts away from implementation details
- Fusion laws can be used for compiler optimizations

Question

Programming languages are just complicated inductive datatypes?! Can we get a `fold` operator for programming languages?

Outline

How to specify the lambda calculus

1. $$t, u \;::=\; x \;\mid\; \mathsf{app}(t,u) \;\mid\; \lambda x.t$$

   plus information about binding of variables

2. $$\frac{x \in X}{X \vdash \mathsf{var}(x)} \qquad \frac{X \vdash t \qquad X \vdash u}{X \vdash \mathsf{app}(t,u)} \qquad \frac{X + 1 \vdash t}{X \vdash \mathsf{abs}(t)}$$

What kind of mathematical object is the lambda calculus?

Lambda calculus as a functor

Definition (Model of Lambda Calculus)

- objects: quadruples

$$F : \mathsf{Set} \to \mathsf{Set}$$
$$var : 1 \Rightarrow F$$
$$app : F \times F \Rightarrow F$$
$$abs : F \circ \mathsf{option} \Rightarrow F$$

- morphisms: ...

Definition (preliminary)

The **lambda calculus** (LC, var, app, abs) is the initial object in that category.

What about substitution?

Using **Mendler Recursion**, can define

$$\mathsf{subst}_{X,Y} : \mathsf{LC}(X) \times (X \to \mathsf{LC}(Y)) \to \mathsf{LC}(Y)$$

from initiality.[1]

Proposition

$(\mathsf{LC}, \mathsf{var}, \mathsf{subst})$ *is a monad on sets.*

[1] see, e.g., [Ahrens, Matthes, Mörtberg, "From signatures to monads in UniMath"]

What about substitution?

Using **Mendler Recursion**, can define

$$\mathsf{subst}_{X,Y} : \mathsf{LC}(X) \times (X \to \mathsf{LC}(Y)) \to \mathsf{LC}(Y)$$

from initiality.[1]

Proposition

$(\mathsf{LC}, \mathsf{var}, \mathsf{subst})$ *is a monad on sets.*

Question

Are app and abs monad morphisms?

[1] see, e.g., [Ahrens, Matthes, Mörtberg, "From signatures to monads in UniMath"]

What about substitution?

Using **Mendler Recursion**, can define

$$\mathsf{subst}_{X,Y} : \mathsf{LC}(X) \times (X \to \mathsf{LC}(Y)) \to \mathsf{LC}(Y)$$

from initiality.[1]

Proposition

$(\mathsf{LC}, \mathsf{var}, \mathsf{subst})$ *is a monad on sets.*

Question

Are app and abs monad morphisms?

Exercise

Show that $\mathsf{abs} : \mathsf{LC} \circ \mathsf{option} \to \mathsf{LC}$ is not a monad morphism.

---

[1] see, e.g., [Ahrens, Matthes, Mörtberg, "From signatures to monads in UniMath"]

Application and abstraction

$$\mathsf{app} : \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC}$$
$$\mathsf{abs} : \mathsf{LC} \circ \mathsf{option} \to \mathsf{LC}$$

- ✓ natural transformations
- ✗ monad morphisms
- ✓ morphism of modules over monad LC

app and abs are module morphisms:

$$\mathsf{subst}(\mathsf{app}(t,u))(f) = \mathsf{app}\big(\mathsf{subst}(t)(f), \mathsf{subst}(u)(f)\big)$$
$$\mathsf{subst}(\mathsf{abs}(t))(f) = \mathsf{abs}\big(\mathsf{subst}(t)(\uparrow f)\big)$$

Summary of the lambda calculus

The lambda calculus is a triple $(\mathsf{LC}, \mathsf{app}, \mathsf{abs})$,

1. monad $\mathsf{LC} : \mathsf{Set} \to \mathsf{Set}$
2. module morphisms over monad LC,

$$\mathsf{app} : \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC}$$
$$\mathsf{abs} : \mathsf{LC} \circ \mathsf{option} \to \mathsf{LC}$$

Definition

A **model of** LC is given by a triple $(R, \mathsf{app}, \mathsf{abs})$

1. monad $R : \mathsf{Set} \to \mathsf{Set}$
2. module morphisms over monad R,

$$\mathsf{app} : R \times R \to R$$
$$\mathsf{abs} : R \circ \mathsf{option} \to R$$

Category of models of LC

Definition

Given two models $(R, \mathsf{app}, \mathsf{abs})$ and $(S, \mathsf{app}, \mathsf{abs})$ of LC, a morphism of models is a monad morphism $f : R \to S$ commuting with app and abs.

Theorem (Hirschowitz & Maggesi)

$(\mathsf{LC}, \mathsf{app}, \mathsf{abs})$ *is initial in the category of models.*

Outline

What specifies the lambda calculus

Model of LC:

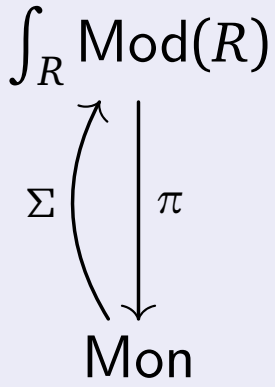
1. monad R
2. morphism of modules $(R \times R) + (R \circ \mathsf{option}) \longrightarrow R$

The data specific to the lambda calculus:

$$R \mapsto (R \times R) + (R \circ \mathsf{option})$$

Definition

Signature Σ is a section to π:



Models of a signature

Definition

A **model of** Σ is a pair of

1. a monad R
2. a morphism $\Sigma(R) \longrightarrow R$ of R-modules
   (a.k.a. an **action of** Σ **in** R)

$\rightsquigarrow$ **category** Mon^{Σ} **of models of** Σ

Definition

Call Σ **representable** if Mon^{Σ} has an initial object.

Examples of signatures

Hypotheses	On objects	Name
	$R \mapsto R$	Θ
F functor, Σ signature	$R \mapsto F \cdot \Sigma(R)$	$F \cdot \Sigma$
	$R \mapsto 1_R$	1
Σ, Ψ signatures	$R \mapsto \Sigma(R) \times \Psi(R)$	$\Sigma \times \Psi$
Σ, Ψ signatures	$R \mapsto \Sigma(R) + \Psi(R)$	$\Sigma + \Psi$
	$R \mapsto R' := R \circ \mathsf{option}$	Θ'
$n \in \mathbb{N}$	$R \mapsto R^{(n)}$	$\Theta^{(n)}$
$(a) = (a_1, \ldots, a_n) \in \mathbb{N}^n$	$R \mapsto R^{(a)} = R^{(a_1)} \times \ldots \times R^{(a_n)}$	$\Theta^{(a)}$

Definition (Signatures are called)

elementary of the form $\Theta^{(a)}$

algebraic coproduct of elementary, e.g., $\Sigma_{\mathsf{LC}} := \Theta \times \Theta + \Theta'$

Not all signatures are representable

Non-example

Let $\mathscr{P} : \mathsf{Set} \to \mathsf{Set}$ denote the powerset functor. The signature $\mathscr{P} \cdot \Theta$ associates, to any monad R, the module $\mathscr{P} \cdot R$ that sends a set X to the powerset $\mathscr{P}(RX)$ of RX.
This signature is not representable.

Goal

Identify sufficient conditions for signatures to be representable.

Algebraic signatures

Theorem (Hirschowitz & Maggesi)

Algebraic signatures are representable.

Earlier variants of this theorem with essentially the same notion of signature by Fiore, Plotkin & Turi, by Gabbay & Pitts, by Hofmann, each using a different notion of model.

Outline

Category of signatures

Definition

Given signatures Σ and Ψ, a **morphism $\Sigma \to \Psi$ of signatures** is a natural transformation that is the identity when postcomposed with $\int \mathsf{Mod} \to \mathsf{Mon}$.
$\rightsquigarrow$ **category** Sig **of signatures**

Proposition

Sig *is cocomplete.*

Modularity

Theorem (CSL 2018)

Given a pushout diagram of representable signatures

$$\begin{array}{ccc} \Sigma_0 & \longrightarrow & \Sigma_1 \\ \downarrow & & \downarrow \\ \Sigma_2 & \longrightarrow & \Sigma \end{array}$$

the corresponding diagram of representations

$$\begin{array}{ccc} \hat{\Sigma}_0 & \longrightarrow & \hat{\Sigma}_1 \\ \downarrow & & \downarrow \\ \hat{\Sigma}_2 & \longrightarrow & \hat{\Sigma} \end{array}$$

is again a pushout, in the category $\int_{\Sigma} \mathsf{Mon}^{\Sigma}$.

Modularity II

Modularity follows from the projection being a Grothendieck fibration

$$\int_{\Sigma} \mathsf{Mon}^{\Sigma} \downarrow \mathsf{Sig}$$

Modularity

allows one to assemble complicated languages by gluing together simpler ones

Outline

Goal

1. Develop an explicit notion of **equation over a signature**
2. Define a **2-signature** to be a pair of a (1-)signature and a family of equations over it
3. Define **models** and **representability** for a 2-signature
4. Identify sufficient criteria for a 2-signature to be representable

η and β for lambda calculus

$$\mathsf{abs}(\mathsf{app}(\iota f, *)) = f$$
$$\mathsf{app}(\mathsf{abs}(f), a) = f[* := a]$$

What are f and a?
Consider

$$\begin{aligned} \mathsf{LC} &\to \mathsf{LC} \\ f &\mapsto \mathsf{abs}(\mathsf{app}(\iota f, *)) && \text{(lhs)} \\ f &\mapsto f && \text{(rhs)} \end{aligned}$$

and

$$\begin{aligned} \mathsf{LC}' \times \mathsf{LC} &\to \mathsf{LC} \\ (f, a) &\mapsto \mathsf{app}(\mathsf{abs}(f), a) && \text{(lhs)} \\ (f, a) &\mapsto f[* := a] && \text{(rhs)} \end{aligned}$$

Equations over Σ_{LC}

We can abstract from LC:

$$\eta \;:\; (R, \mathsf{app}, \mathsf{abs}) \;\mapsto\; \begin{array}{l} R \to R \\ f \mapsto \mathsf{abs}(\mathsf{app}(\iota f, *)) \\ f \mapsto f \end{array}$$

and

$$\beta \;:\; (R, \mathsf{app}, \mathsf{abs}) \;\mapsto\; \begin{array}{l} R' \times R \to R \\ (f, a) \mapsto \mathsf{app}(\mathsf{abs}(f), a) \\ (f, a) \mapsto f[* := a] \end{array}$$

Definition (Equation over Σ)

An **equation** $e_1 = e_2 : \Psi \to \Phi$ **over** Σ associates, to any model M of Σ, a pair of parallel module morphisms

$$e_1(M), e_2(M) : \Psi(M) \to \Phi(M)$$

2-signatures

Definition (2-signature)

A **2-signature** is a pair (Σ, E) where Σ is a (1-)signature and E is a family of equations over Σ.

Example

$(\Sigma_{\mathsf{LC}}, (\beta, \eta))$ is the 2-signature of lambda calculus with β- and η-equality.

Models of 2-signatures

Definition

A **model of** (Σ, E) is a model M of Σ such that, for any equation (e_1, e_2) of E, we have $e_1(M) = e_2(M)$.

$$\mathsf{Mon}^{(\Sigma,E)} \subset \mathsf{Mon}^{\Sigma}$$

Example

Let $\Sigma := \Theta$. The equation

$$\begin{array}{rcl} & & R \to R + R \\ (R, r) & \mapsto & x \mapsto \mathsf{inl}(x) \\ & & x \mapsto \mathsf{inr}(x) \end{array}$$

is never satsified.

Elementary equations

Definition

An equation is **elementary** if

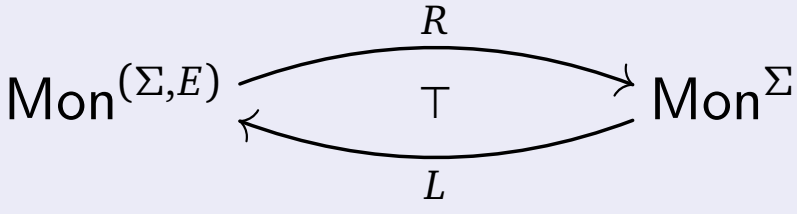
1. the source is of the form $(R, r) \mapsto R^{(a_1)} \times \ldots \times R^{(a_n)}$
2. the target is of the form $(R, r) \mapsto R^{(a)}$

Theorem (FSCD 2019)

If Σ is representable, and E is a family of elementary Σ-equations, then (Σ, E) is representable.

Theorem (FSCD 2019, with axiom of choice)

Reflection



Outline

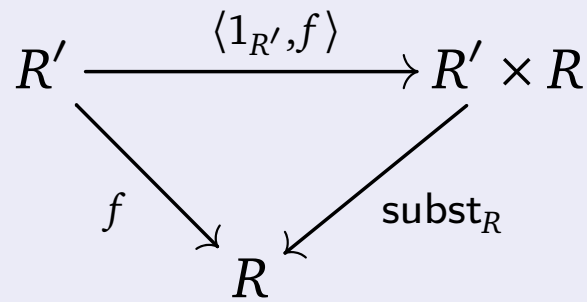
Fixpoint operator

Definition

A fixed point combinator is a lambda term Y s.t. for any term t, $\mathsf{app}(t, \mathsf{app}(Y, t)) = \mathsf{app}(Y, t)$.

Definition

A **unary fixpoint operator for a monad** R is a module morphism $f : R' \to R$ such that



Lemma

There is a one-to-one correspondence between fixpoint operators in $\mathsf{LC}_{\beta\eta}$ *and fixpoint combinators* Y.

2-Signature of an explicit fixpoint operator

Signature Θ'

Model of signature $(R, \mathsf{fix} : R' \to R)$

Equation

$$(R, \mathsf{fix} : R' \to R) \quad \mapsto \quad \begin{array}{ccccc} R' & \xrightarrow{\langle 1,\mathsf{fix}\rangle} & R' \times R & \xrightarrow{\mathsf{subst}_R} & R \\ R' & & \xrightarrow[\mathsf{fix}]{} & & R \end{array}$$

Recursion principle

Proposition (Recursion principle)

Let S be the monad underlying the initial model of the 2-signature Υ. To any action a of Υ in T is associated a unique monad morphism $\hat{a} : S \to T$.

Translation from LC with explicit fixpoint operator to LC

Definition

$\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}} := \Upsilon_{\mathsf{LC}_{\beta\eta}} + \Upsilon_{\mathsf{fix}}$ with

- $\Upsilon_{\mathsf{LC}_{\beta\eta}} = (\Sigma_{\mathsf{LC}}, (\beta, \eta))$
- $\Upsilon_{\mathsf{fix}} = (\Theta', \mathsf{fix})$

To specify a translation $\mathsf{LC}_{\beta\eta,\mathsf{fix}}$ to $\mathsf{LC}_{\beta\eta}$, it suffices to

- specify an action a of Θ' in $\mathsf{LC}_{\beta\eta}$
- such that the equation $\mathsf{fix}(\mathsf{LC}_{\beta\eta}, a)$ is satisfied

Can pick $a : \mathsf{LC}'_{\beta\eta} \to \mathsf{LC}_{\beta\eta}$ to by induced, e.g., by Curry fixpoint combinator.

Outline

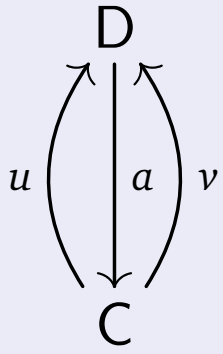
Goal

To give a notion of arity/signature that encompasses

- signatures for terms
- equations between terms
- reductions between terms

Definition

An **arity over** C is a quadruple (D, a, u, v)



with $u, v : \mathsf{C} \to \mathsf{D}$ sections of $a : \mathsf{D} \to \mathsf{C}$.

Actions of an arity

Definition

An **action of an arity** $A = (\mathsf{D}, a, u, v)$ **on an object** $c \in \mathsf{C}$ is a morphism $h : u(c) \to v(c)$ such that $a(h) = 1_c$.

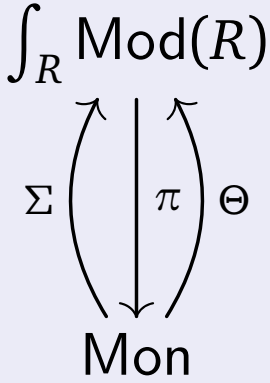
Definition (Model of an arity over C)

A **model of an arity over** C is a pair of an object $c : \mathsf{C}_0$ and an action $r : u(c) \to v(c)$.

Arities encompass signatures

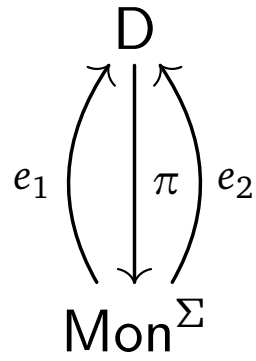
Definition

Any signature $\Sigma : \mathsf{Mon} \to \int_R \mathsf{Mod}(R)$ gives rise to an arity over Mon as



Arities encompass equations

Any equation $e_1 = e_2 : \Psi \to \Phi$ over a signature
$\Sigma : \mathsf{Mon} \to \int_R \mathsf{Mod}(R)$ gives rise to an arity over Mon^Σ as



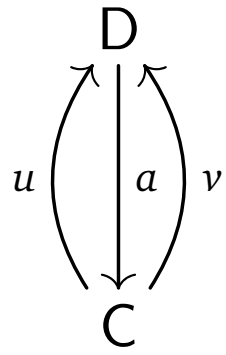
with

- object of D is pair (R, h) with $h : \Psi(R) \to \Phi(R)$
- morphism $(R, h) \to (S, i)$ is model morphism $f : R \to S$ such that

$$\begin{array}{ccc} \Psi(R) & \xrightarrow{h} & \Phi(R) \\ {\scriptstyle \Psi(f)}\downarrow & & \downarrow{\scriptstyle \Phi(f)} \\ \Psi(S) & \xrightarrow[i]{} & \Phi(S) \end{array}$$

Outline

An alternative definition of arities



Lemma

When a is a Grothendieck fibration, this is the same as

- *a pseudofunctor* $a : \mathsf{C}^{op} \to \mathsf{Cat}$
- *natural transformations* $u, v : 1 \to a$.

Reductions as edges in a graph

- To integrate reduction rules, we consider monads relative to an inclusion Set $\to$ Graph; we call these "reduction monads"
- A reduction rule is an arity over the category RedMon^{Σ} of models of Σ in such relative monads
- $$a : (\mathsf{RedMon}^{\Sigma})^{\mathsf{op}} \to \mathsf{Cat}$$
$$R \mapsto \mathsf{Mod}(R)/\mathsf{MVar}_{\mathscr{A}}(R)$$
- $$u : 1 \Rightarrow a$$
$$u(R)(*) := \mathsf{hyp}_{\mathscr{A}}(R) : \mathsf{Hyp}_{\mathscr{A}}(R) \to \mathsf{MVar}_{\mathscr{A}}(R)$$
- $$v : 1 \Rightarrow a$$
$$v(R)(*) := \mathsf{con}_{\mathscr{A}}(R) : \mathsf{Con}_{\mathscr{A}}(R) \to \mathsf{MVar}_{\mathscr{A}}(R)$$

Initial semantics for reduction signatures

A reduction signature consists of

- a signature Σ over Mon
- a family $\mathfrak{R}$ of reduction rules over RedMon^{Σ}

Theorem (submitted)

Let $(\Sigma, \mathfrak{R})$ be a reduction signature. If Σ is representable, then so is $(\Sigma, \mathfrak{R})$.

Thanks for your attention!