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.

In this talk

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

Outline

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

A very simple datatype

```
datatype Nat =         -- inductive datatype
| O                    -- O : Nat
| S Nat                -- S : Nat -> Nat
```

- A *model of the natural numbers* is any triple `(X,x,s)` with `x:X` and `s:X -> X`
- A model `(X,x,s)` specifies function `rec x s : Nat -> X`

  ```
  rec : X -> (X -> X) -> (Nat -> X)
  rec x s O = x
  rec x s (S n) = s (rec x s n)
  ```

- `rec x s : Nat -> X` is structure-preserving: preserves the chosen element and endomorphism on each side

Initiality for natural numbers

Theorem (Initiality for natural numbers)

For any model (X, x, s) of natural numbers there is exactly one structure-preserving map

$$(\mathsf{Nat}, 0, S) \to (X, x, s)$$

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

Theorem (Reformulated in category theory)

$(\mathsf{Nat}, 0, S)$ *is the initial object in the category of models of the natural numbers.*

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

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

How to specify the lambda calculus

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

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

$$
\begin{aligned}
F &: \mathsf{Set} \longrightarrow \mathsf{Set} \\
var &: 1 \Rightarrow F \\
app &: F \times F \Rightarrow F \\
abs &: F \circ \mathsf{option} \Rightarrow F
\end{aligned}
$$

- morphisms: ...

Definition (preliminary)

The **lambda calculus** $(\mathsf{LC}, \mathsf{var}, \mathsf{app}, \mathsf{abs})$ 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 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

Definition

Given monad R on sets, a **module** M **over** R is a

1. function $M : \mathsf{Set} \to \mathsf{Set}$
2. family of functions $\mathsf{subst}_{X,Y} : M(X) \times (X \to RY) \to M(Y)$ satisfying two axioms

Examples of modules

Given monad R, have modules

- R
- $R \times R$
- $R \circ \mathsf{option}$

Module morphisms

Definition

Given modules M, N over monad R, a morphism from M to N is

- a nat. transformation $\tau : M \to N$
- for any $f : X \to R(Y)$,

$$\begin{array}{ccc} M(X) & \xrightarrow{\tau} & N(X) \\ {\scriptstyle \mathsf{subst}^M(f)} \downarrow & & \downarrow {\scriptstyle \mathsf{subst}^N(f)} \\ M(Y) & \xrightarrow{\tau} & N(Y) \end{array}$$

⇝ **category** $\mathsf{Mod}(R)$ **of modules over monad** R

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,

$$\begin{aligned} \mathsf{app} &: \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC} \\ \mathsf{abs} &: \mathsf{LC} \circ \mathsf{option} \to \mathsf{LC} \end{aligned}$$

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,

$$\begin{aligned} \mathsf{app} &: R \times R \to R \\ \mathsf{abs} &: R \circ \mathsf{option} \to R \end{aligned}$$

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

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

What specifies the lambda calculus

Model of LC:

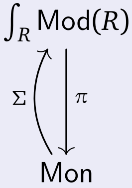
1. monad R
2. morphism of modules $(R \times R) + (R \circ \mathsf{option}) \longrightarrow R$

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

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

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

Given a pushout diagram of representable signatures

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

Modularity

Given a pushout diagram of representable signatures

$$
\begin{array}{ccc}
\Sigma_0 & \longrightarrow & \Sigma_1 \\
\downarrow & & \downarrow \\
\Sigma_2 & \longrightarrow & \Sigma
\end{array}
\qquad\qquad
\begin{array}{ccc}
\hat{\Sigma}_0 & \longrightarrow & \hat{\Sigma}_1 \\
\downarrow & & \downarrow \\
\hat{\Sigma}_2 & \longrightarrow & \hat{\Sigma}
\end{array}
$$

the corresponding diagram of representations is again a pushout, in the category $\int_{\Sigma} \mathsf{Mon}^{\Sigma}$:

Definition

object is a triple (Σ, R, r) where Σ is a signature, R is a monad, and r is an action of Σ in R.

morphism from (Σ_1, R_1, r_1) to (Σ_2, R_2, r_2) is a pair (i, m):

- signature morphism $i : \Sigma_1 \longrightarrow \Sigma_2$
- $m : (R_1, r_1) \to (R_2, i^*(r_2))$ morphism of Σ_1-models

Modularity II

Modularity follows from the projection being a Grothendieck fibration

$$\begin{array}{c} \int_{\Sigma} \mathsf{Mon}^{\Sigma} \\ \downarrow \\ \mathsf{Sig} \end{array}$$

Modularity

allows one to assemble complicated languages by gluing together simpler ones

Outline

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

Goal

Goal

Integrate some “semantic” equalities into the syntax.

- For instance, a binary operator is semantically symmetric (e.g., addition, parallel-or,...).
- Instead of defining that a posteriori, integrate this symmetry in the syntax.

Can be expressed as quotients of signatures

Presentations of a signature

Definition

A **presentation of** Σ is an algebraic signature Ψ and an epimorphism

$$\begin{array}{c}\Psi \\ \big\downarrow\!\!\!\downarrow \\ \Sigma\end{array}$$

Theorem (Ahrens, Hirschowitz, Lafont, Maggesi)

Presentable signatures are representable.

Signature for a commutative binary operator

$$\begin{array}{c} \Theta \times \Theta \\ \big\downarrow\!\!\!\downarrow \\ \mathscr{S}_2 \cdot \Theta \end{array}$$

A model of $\mathscr{S}_2 \cdot \Theta$ is a pair $(R, m : R \times R \to R)$ such that $m_X(a, b) = m_X(b, a)$.

Example: explicit substitution

Consider p-ary substitution

$$\mathsf{subst}_p : R^{(p)} \times R^p \longrightarrow R$$

Calculi with **explicit substitution** allow delaying substitutions. If $u : [p] \longrightarrow [q]$ a function, we expect

$$\begin{array}{ccc}
R^{(p)} \times R^q & \xrightarrow{1 \times R^u} & R^{(p)} \times R^p \\
{\scriptstyle R^{(u)} \times 1} \big\downarrow & & \big\downarrow {\scriptstyle \mathsf{subst}_p} \\
R^{(q)} \times R^q & \xrightarrow[\mathsf{subst}_q]{} & R
\end{array}$$

Signature for a coherent family of explicit substitutions

$$\int^{p:\mathbb{N}} \Theta^{(\underline{p})} \times \Theta^{\overline{p}}$$

Outline

1 Recursion from initiality by example
2 Initial semantics for the untyped lambda calculus
3 Signatures and their models
4 Building complicated languages from simpler ones
5 Integrating equations between terms
6 2-signatures: equations between terms
7 Example: Translation of lambda calculus with fixpoint operator
8 Unified view of arities, signatures, and equations
9 A glimpse at reduction rules

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:

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

and

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

Equations over Σ_{LC}

We can abstract from LC:

$$\eta \quad : \quad (R, \mathsf{app}, \mathsf{abs}) \quad \mapsto \quad \begin{aligned} &R \to R \\ f &\mapsto \mathsf{abs}(\mathsf{app}(\iota f, *)) \\ f &\mapsto f \end{aligned}$$

and

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

Source and target of an equation over Σ are given by
Σ-modules. . .

Σ-modules

Definition

A Σ-**module** is a functor T from the category Mon^{Σ} of Σ-monads to the category $\int_R \mathsf{Mod}(R)$ commuting with the forgetful functors to the category Mon of monads,

$$\begin{array}{ccc} \mathsf{Mon}^{\Sigma} & \xrightarrow{\;T\;} & \int_R \mathsf{Mod}(R) \\ & \searrow \quad \swarrow & \\ & \mathsf{Mon} & \end{array}$$

Example

- $(R, app, abs) \mapsto R$
- $(R, app, abs) \mapsto R' \times R$

2-signatures

Definition (Equation over Σ)

A Σ-equation is a pair

$$e_1, e_2 : \Psi \longrightarrow \Phi$$

of parallel morphisms of Σ-modules.

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

$$(R, r) \quad \mapsto \quad \begin{array}{l} R \longrightarrow R + R \\ x \mapsto \mathsf{inl}(x) \\ x \mapsto \mathsf{inr}(x) \end{array}$$

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 (A-H-L-M)

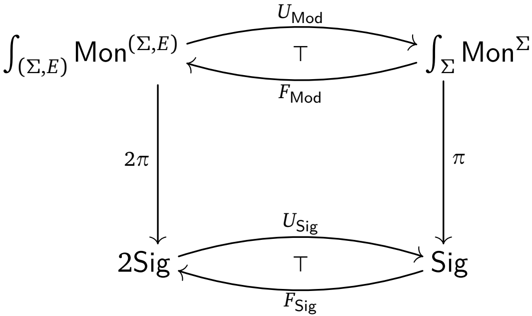
If Σ is representable, and E is a family of elementary Σ-equations, then (Σ, E) is representable.

Theorem (with axiom of choice)

Reflection

$$\mathsf{Mon}^{(\Sigma,E)} \underset{L}{\overset{R}{\underset{\top}{\rightleftarrows}}} \mathsf{Mon}^{\Sigma}$$

Modularity



A pushout diagram of representable 2-signatures yields a pushout diagram of representations “above”.

Outline

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

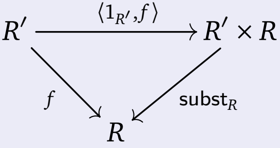
Fixpoint operator

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

$$(R, \mathsf{fix} : R' \to R) \quad \mapsto \quad \begin{array}{l} R' \xrightarrow{\langle 1, \mathsf{fix} \rangle} R' \times R \xrightarrow{\mathsf{subst}_R} R \\ R' \xrightarrow[\mathsf{fix}]{\hspace{4cm}} R \end{array}$$

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

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

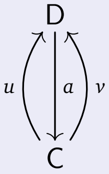
Goal

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

$A = (\mathsf{D}, a, u, v)$ arity over C, and $c_1, c_2 : \mathsf{C}$ with actions $h_1 : u(c_1) \to v(c_1)$ and $h_2 : u(c_2) \to v(c_2)$. A morphism $f : c_1 \to c_2$ is **compatible with the actions** h_1 **and** h_2 if

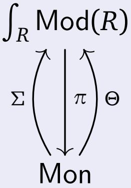
$$\begin{array}{ccc} u(c_1) & \xrightarrow{h_1} & v(c_1) \\ {\scriptstyle u(f)}\downarrow & & \downarrow{\scriptstyle v(f)} \\ u(c_2) & \xrightarrow[h_2]{} & v(c_2) \end{array}$$

commutes.

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

$$\begin{array}{c} \mathsf{D} \\ e_1 \uparrow \;\; \downarrow \pi \;\; \uparrow e_2 \\ \mathsf{Mon}^\Sigma \end{array}$$

with

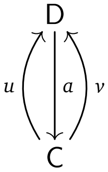
- object of D is pair (R, h) with $h : \Psi(R) \to \Phi(R)$
- morphism $(R, h) \to (S, i)$ is model morphism $f : R \to S$ such that

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

Outline

1. Recursion from initiality by example
2. Initial semantics for the untyped lambda calculus
3. Signatures and their models
4. Building complicated languages from simpler ones
5. Integrating equations between terms
6. 2-signatures: equations between terms
7. Example: Translation of lambda calculus with fixpoint operator
8. Unified view of arities, signatures, and equations
9. A glimpse at reduction rules

An alternative definition of arities



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 into the picture, we consider monads relative to an inclusion $\mathsf{Set} \to \mathsf{Graph}$; we call these "reduction monads"
- A reduction rule is an arity over the category RedMon^{Σ} of models of Σ in such relative monads
- $a : \mathsf{D} \to \mathsf{RedMon}^{\Sigma}$ is the Grothendieck fibration corresponding (through the Grothendieck construction) to the functor mapping a reduction Σ-monad R to the category $\mathsf{Mod}(R)/\mathsf{MVar}_{\mathscr{A}}(R)$;
- u maps a reduction Σ-monad R to $\mathsf{hyp}_{\mathscr{A}}(R) : \mathsf{Hyp}_{\mathscr{A}}(R) \to \mathsf{MVar}_{\mathscr{A}}(R)$
- v maps a reduction Σ-monad R to $\mathsf{con}_{\mathscr{A}}(R) : \mathsf{Con}_{\mathscr{A}}(R) \to \mathsf{MVar}_{\mathscr{A}}(R)$

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

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

Thanks for your attention!