Syntax and Semantics

Benedikt Ahrens

Institute for Advanced Study, Princeton

Sept. 25, 2012

Initial Semantics

Ingredients:

- Signature Σ

$\rightsquigarrow$ specifies a Language $\hat{\Sigma}$

$\rightsquigarrow$ Category of Semantics of Σ with Initial Semantics $\hat{\Sigma}$

Why Initial Semantics?

- Characterization of language $\hat{\Sigma}$ via universal property
- Categorical iteration operator

A First Example: Naturals

Signature

- Zero : 0
- Succ : 1

Language of Naturals

```
Inductive ℕ : Set :=
  | Zero : ℕ
  | Succ : ℕ → ℕ
```

Semantics

- $(1 + X)$ –algebras, i.e. triples of the form

$$(X, Z \in X, S : X \to X)$$

Initial Semantics for $\mathbb{N}$

Initial Semantics

$(\mathbb{N}, \texttt{Zero}, \texttt{Succ})$ is the initial $(1 + X)$–algebra

Categorical Iteration Operator

specify a map $\mathbb{N} \to X$ by turning X into $(1 + X)$–algebra

Language Features

Starting Point:

Universal Algebra, Birkhoff ’35

Features:

- Variable Binding
- Typing
- Reductions / Equations

Adding a Feature

- Defining a new notion of Signature
- Adapting the notion of Semantics

Encodings of variable binding

- Nominal Approach – Named Abstraction

$$\lambda : [A]T \to T$$

- Higher-Order Abstract Syntax (HOAS)

$$\lambda : (T \to T) \to T$$

- Nested Datatype

$$\lambda : T(V + 1) \to T(V)$$

Example: Lambda Calculus and its Semantics

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Semantics of Σ_{LC} ?

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
      | Var : V -> LC(V)
      | Abs : LC(V+1) -> LC(V)
      | App : LC(V) x LC(V) -> LC(V)
```

Categorical Structure of LC

- $\mathrm{LC} : \mathit{Set} \to \mathit{Set}$
- $\mathit{Var} : \mathit{Id} \Rightarrow \mathrm{LC}$, $\mathit{Abs} : \mathrm{LC}' \Rightarrow \mathrm{LC}$, $\mathit{App} : \mathrm{LC} \times \mathrm{LC} \Rightarrow \mathrm{LC}$

Example: Lambda Calculus and its Semantics

Signature: $\Sigma_{LC} = \{abs : [1] \ , \quad app : [0, 0]\}$

Syntax:

```
Inductive LC (V : Set) : Set :=
        | Var : V -> LC(V)
        | Abs : LC(V+1) -> LC(V)
        | App : LC(V) x LC(V) -> LC(V)
```

Abstracting from LC, a Semantics of Σ_{LC} is...

- $F : Set \to Set$
- $Var : Id \Rightarrow F \ , \ Abs : F' \Rightarrow F \ , \ App : F \times F \Rightarrow F$

Initial Semantics for LC

Initial Semantics

(LC, Var, Abs, App) is the Initial Semantics of Σ_{LC}

Categorical Iteration Operator

specify a map LC $\to F$ by equipping $F : Set \to Set$ with

- $Var : Id \to F$
- $Abs : F' \to F$
- $App : F \times F \to F$

Can we get a Better Iteration Operator?

A good iteration operator

- can be used to specify “good” morphisms
- requires a richer category of Semantics

Summary

Recursor	Type Theory	Category Theory
Purpose	Generality	Certification

Which Properties to Certify?

- many possible answers
- domain–specific

Certified Translations via Initiality I

Goal:

obtain a categorical iteration operator for good translations between functional programming languages

Good translations are compatible with

- substitution
- typing
- reductions, e.g., $t \rightsquigarrow t' \quad \Longrightarrow \quad f(t) \rightsquigarrow f(t')$

Certified Translations via Initiality II

Step 1

- arrange languages and translations between languages in a category $\mathcal{L}$
- s.t. only good translations are morphisms

Step 2: For a signature Σ...

- define a category $\mathcal{L}_\Sigma$ of Σ–Semantics such that
- the language $\hat{\Sigma}$ is initial in $\mathcal{L}_\Sigma$

Yields categorical iterator

- specifies just the good translations
- in a uniform way

Summary & Results

Summary

Recursor	Type Theory	Category Theory
Purpose	Generality	Certification

“Done” for

- Simple Type Systems
- “done” does not make sense (choice of properties)

Some things to do

- dependent types
- polymorphism

Connection to HoTT and UF

Higher Inductive Types

Recursor	Type Theory	Category Theory
Purpose	Some Certification (Reductions)	More Certification

Lambda Calculus with Eta

Eta for LC

$$\lambda x.M(x) \equiv M$$

For some translation f : LC $\rightarrow L$

we would want

$$f(\lambda x.M(x)) \equiv f(M)$$

Modelling reductions as paths...

we can integrate this property into type–theoretic recursor

LC with Eta as HIT

```
Inductive LC (V : Type) : Type :=
 | Var : V -> LC V
 | Abs : LC (V+1) -> LC V
 | App : LC V -> LC V -> LC V
 | eta : forall M : LC (V+1),
              Abs (App M (Var inr)) ~~> M
```

Recursor for LC with Eta

```
LC_rect :
∀(P : ∀V : Type, LC V -> Type)
 (d_Var : ∀(V : Type) (v : V), P V (Var V v))
 (d_Abs : ∀(V : Type) (t : LC (V+1)),
     P (V+1) t -> P V (Abs V t))
 (d_App : ∀(V : Type) (t : LC V),
     P V t -> ∀t0 : LC V, P V t0 -> P V (App V t
         t0))

 (d_eta : ∀(V : Type)(M : LC (V+1))(t : P (V+1) M),
   transport (eta M)
  d_Abs V M (d_App (V+1) M t inr (d_Var (V+1) inr))
   ~~>
   t
 ),

   ∀(V : Type) (l : LC V), P V l
```

Property ensured by recursor

A map f specified using `LC_rect`

- is certified to satisfy

$$f(\lambda x.M(x)) \equiv f(M)$$

- cf. family of paths `d_eta`

Would be nice to have...

Higher Induction–Recursion

```
Inductive LC (V : Type) : Type :=
 | Var : V -> LC V
 | Abs : LC (V+1) -> LC V
 | App : LC V -> LC V -> LC V
 | beta : ∀(M : LC (V+1))(N : LC V),
             App (Abs M) N ~~> subst M N
with
Fixpoint subst (M : LC (V+1)) (N : LC V) : LC V :=
...
```

Thanks for your attention