Initiality for Typed Syntax and Semantics

Benedikt Ahrens

Université Nice Sophia Antipolis

Soutenance de Thèse

May 23, 2012

A Translation from PCF to LC

PCF

- Typed Language
  $T ::= Nat \mid Bool \mid T \Rightarrow T$
- Abstraction
  Application
  Fixpoint Operator
  Arithmetic Consts.
  Logic Consts.

Lambda Calculus

- Untyped
- Abstraction
  Application

Translation i : PCF $\rightarrow$ LC

$$i(\lambda x.M) = \lambda x.i(M)$$

$$i(M@N)) = i(M)@i(N)$$

$$i(Fix(f)) = \Theta@i(f) \qquad \text{or} \qquad i(Fix(f)) = \mathbf{Y}@i(f)$$

$$\ldots$$

Translation, mathematically

Mathematical structure of

$$i : \mathrm{PCF} \to \mathrm{LC} \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $i : \mathrm{PCF} \to \mathrm{LC}$ morphism in some category ?

Translation, mathematically

Mathematical structure of

$$i : \mathrm{PCF} \to \mathrm{LC} \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $i : \mathrm{PCF} \to \mathrm{LC}$ initial morphism in some category ?

Translation, mathematically

Mathematical structure of

$$i : \mathrm{PCF} \to \mathrm{LC} \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $i : \mathrm{PCF} \to \mathrm{LC}$ initial morphism in some category ?
- Extra structure on LC specifies different such translations

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

Language Features

Starting Point:

Universal Algebra

Features:

- Variable Binding
- Typing
- Reductions

Adding a Feature

- Defining a new notion of signature
- Adapting the notion of representation

Outline

Outline

Goals

- "Signature" for specifying untyped binding syntax

$\rightsquigarrow$ Category of Semantics (Representations) of a Signature

$\rightsquigarrow$ exhibit Syntax as initial object

$+$ encode as much structure as possible in representations

Has been done by

- Fiore, Plotkin, Turi '99
- Hofmann '99
- Gabbay, Pitts '99
- Hirschowitz, Maggesi '07

We review H&M and extend to types and reduction rules

Ex.: Categorical Structure of Lambda Calculus

Syntax:

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

Ex.: Categorical Structure of Lambda Calculus

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

Syntax:

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

Ex.: Categorical Structure of Lambda Calculus

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)
```

Ex.: Categorical Structure of Lambda Calculus

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)
```

Ex.: Categorical Structure of Lambda Calculus

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)
```

Ex.: Categorical Structure of Lambda Calculus

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)
```

Reps: ??

Ex.: Categorical Structure of Lambda Calculus

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

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

Ex.: Categorical Structure of Lambda Calculus

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 Representation of Σ_{LC} is...

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

Ex.: Categorical Structure of Lambda Calculus

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 Representation of Σ_{LC} is...

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

Goal: Refine "Representation" by integrating Substitution...

LC as a Monad

"Variables–as–terms" and Substitution for LC

- $V \mapsto \mathrm{LC}(V)$
- $\mathit{Var} : \mathit{Id} \Rightarrow \mathrm{LC}$
- $(\gg\!=)_{V,W} : \mathrm{LC}(V) \times (V \to \mathrm{LC}(W)) \to \mathrm{LC}(W)$

Properties:

- $M \gg\!= \mathit{Var}_V \;==\; M$
- $\mathit{Var}(v) \gg\!= f \;==\; f(v)$
- $x \gg\!= f \gg\!= g \;==\; x \gg\!= (\lambda v.f(v) \gg\!= g)$

Equips LC with structure of Monad...

Integrate Substitution

Monad: Functor + "Variables–as–terms" + Substitution

- $P : \mathcal{C} \to \mathcal{C}$
- $\eta : Id \Rightarrow P$
- $\sigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{C}(PX, PY)$ + substitution properties

Categorical Structure of LC

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

Integrate Substitution

Monad: Functor + "Variables–as–terms" + Substitution

- $P : \mathcal{C} \to \mathcal{C}$
- $\eta : Id \Rightarrow P$
- $\sigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{C}(PX, PY)$ + substitution properties

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

- Monad $P : Set \to Set$
- $Abs : P' \Rightarrow P$, $App : P \times P \Rightarrow P$

Integrate Substitution

Monad: Functor + "Variables–as–terms" + Substitution

- $P : \mathcal{C} \to \mathcal{C}$
- $\eta : Id \Rightarrow P$
- $\sigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{C}(PX, PY)$ + substitution properties

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

- Monad $P : Set \to Set$
- $Abs : P' \Rightarrow P$, $App : P \times P \Rightarrow P$

Even more refinement:

substitution distributes over constructors

Substitution and Constructors of LC

Substitution distributes over *App*

- $M, N \in \mathrm{LC}(V)$
- $f : V \to \mathrm{LC}(W)$
- $App(M)(N) \gg\!= f \quad == \quad App(M \gg\!= f)(N \gg\!= f)$

Going under a Binder:

- $M \in \mathrm{LC}(V + 1)$
- $f : V \to \mathrm{LC}(W)$
- $f' : V + 1 \to \mathrm{LC}(W + 1)$
- $Abs(M) \gg\!= f \quad == \quad Abs(M \gg\!= f')$

Constructors = Morphisms of Modules

Module over a Monad $P : \mathcal{C} \to \mathcal{C}$

- Functor $M : \mathcal{C} \to \mathcal{D}$ + Substitution (comp. w. Monad)

$$\varsigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{D}(MX, MY)$$

Morphism of Modules

- Natural Transformation + compatible with Substitution

Constructors = Morphisms of Modules

Module over a Monad $P : \mathcal{C} \to \mathcal{C}$

- Functor $M : \mathcal{C} \to \mathcal{D}$ + Substitution (comp. w. Monad)

$$\varsigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{D}(MX, MY)$$

Morphism of Modules

- Natural Transformation + compatible with Substitution

App and *Abs*

- $App(M)(N) \gg\!= f \quad == \quad App(M \gg\!= f)(N \gg\!= f)$
- $Abs(M) \gg\!= f \quad == \quad Abs(M \gg\!= f')$

Morphisms of Modules for LC

Modules over LC

$$\mathrm{LC}' : V \mapsto \mathrm{LC}(V + 1) \qquad \mathrm{LC} : V \mapsto \mathrm{LC}(V)$$

$$\mathrm{LC} \times \mathrm{LC} : V \mapsto \mathrm{LC}(V) \times \mathrm{LC}(V)$$

Morphisms of Modules over LC

$$\mathit{Abs} : \mathrm{LC}' \to \mathrm{LC}$$

$$\mathit{App} : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$$

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

- Monad $P : \mathit{Set} \to \mathit{Set}$
- $\mathit{Abs} : P' \to P$, $\mathit{App} : P \times P \to P$ modules morphisms

Representations of a Signature

Definition (Signature)

- Arity : List of natural numbers
- Signature: Family of arities

Arity s associates module $P^{(s)}$ to any monad P

Definition (Representation of Signature Σ, H&M)

- Monad $P : Set \to Set$
- Morphism of modules over P for each $s \in \Sigma$,

$$P^{(s)} \to P$$

Initial Semantics, untyped

Theorem (Hirschowitz & Maggesi)

For any signature Σ, the category of representations of Σ has an initial object.

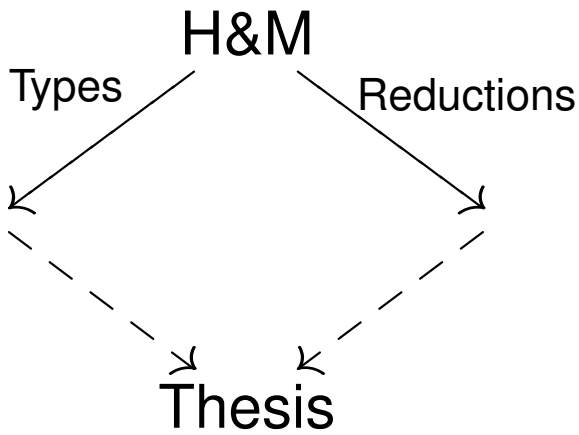
Example

(LC, *App*, *Abs*) is the initial representation of Σ_{LC}.

My Work

Extend result to integrate

- Typing
- Reductions



Outline

Goals

- “Signature” to specify simply–typed language w. binding

$\rightsquigarrow$ Category of Representations of a Signature

- Monads ? Modules over Monads ?

$\rightsquigarrow$ exhibit Initial Object: Syntax

$+$ encode as much structure as possible in representations

Goals

- "Signature" to specify simply–typed language w. binding

$\rightsquigarrow$ Category of Representations of a Signature

- Monads ? Modules over Monads ?

$\rightsquigarrow$ exhibit Initial Object: Syntax

$+$ encode as much structure as possible in representations

Generalizing Zsidó's Theorem

- Existing solution w. fixed types

$\rightsquigarrow$ Generalize to varying sets of types

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Reps: Monads?

Example: Simply–Typed LC

Types: $T ::= * \mid T \Rightarrow T$

Σ_{TLC} :

$$\left\{ \left(abs_{s,t} : \left[([s], t) \right] \to s \Rightarrow t \right)_{s,t \in T} , \right.$$
$$\left. \left(app_{s,t} : \left[([], s \Rightarrow t), ([], s) \right] \to t \right)_{s,t \in T} \right\}$$

Syntax:

```
Inductive TLC (V: T -> Set): T -> Set :=
| Var: ∀t, V(t) -> TLC(V)(t)
| Abs: ∀s t, TLC(V+s)(t) -> TLC(V)(s ⇒ t)
| App: ∀s t, TLC(V)(s⇒t) x TLC(V)(s) -> TLC(V)(t)
```

Reps: Monads? Modules?

TLC as a Monad

Monad: Functor + "Variables–as–terms" + Substitution

- $P : \mathcal{C} \to \mathcal{C}$
- $\eta : Id \Rightarrow P$
- $\sigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{C}(PX, PY)$ + substitution properties

The Monad TLC

- $\mathrm{TLC} : Set^T \to Set^T$
- $Var : Id \Rightarrow \mathrm{TLC}$
- $(\gg\!=)_{V,W} : \mathrm{TLC}(V) \times (V \to \mathrm{TLC}(W)) \to \mathrm{TLC}(W)$

Modules over TLC

Fibre Module for $t \in T$

- $\mathrm{TLC}[t] : \mathit{Set}^T \to \mathit{Set} \quad , \quad V \mapsto \mathrm{TLC}(V)(t)$

Domain Modules

- $\mathrm{TLC}^{(s)}[t] : V \mapsto \mathrm{TLC}(V + s)(t)$
- $\mathrm{TLC}[s \Rightarrow t] \times \mathrm{TLC}[s] : V \mapsto \mathrm{TLC}(V)(s \Rightarrow t) \times \mathrm{TLC}(V)(s)$

Morphisms of Modules over TLC

- $\mathit{Abs}_{s,t} : \mathrm{TLC}^{(s)}[t] \to \mathrm{TLC}[s \Rightarrow t]$
- $\mathit{App}_{s,t} : \mathrm{TLC}[s \Rightarrow t] \times \mathrm{TLC}[s] \to \mathrm{TLC}[t]$

Simply–Typed LC and its Representations

The Lambda Calculus

- Monad $\quad \mathrm{TLC} : \mathit{Set}^T \to \mathit{Set}^T$
- $\mathit{Abs}_{s,t} : \mathrm{TLC}^{(s)}[t] \to \mathrm{TLC}[s \Rightarrow t] \qquad \forall\ s\ t \in T$
- $\mathit{App}_{s,t} : \mathrm{TLC}[s \Rightarrow t] \times \mathrm{TLC}[s] \to \mathrm{TLC}[t] \qquad \forall\ s\ t \in T$

Abstracting from TLC, a Representation of Σ_{TLC} is...

- Monad $\quad P : \mathit{Set}^T \to \mathit{Set}^T$
- $\mathit{Abs}_{s,t} : P^{(s)}[t] \to P[s \Rightarrow t] \qquad \forall\ s\ t \in T$
- $\mathit{App}_{s,t} : P[s \Rightarrow t] \times P[s] \to P[t] \qquad \forall\ s\ t \in T$

Typed Signatures

Definition (Arity, Signature over set of types T)

- Arity: $\left[([t_{1,1}, \ldots, t_{1,m_1}], t_1), \ldots, ([t_{n,1}, \ldots, t_{n,m_n}], t_n)\right] \to t_0$

  $t_{i,j}, t_i \in T$
- Signature : family of arities

Arity $\ell \to t_0$ associates modules $P^{(\ell)}$ and $P[t_0]$ to any P

Definition (Representation of signature Σ over types T)

- Monad P on Set^T
- Morphism of modules over P for each arity $\ell \to t_0 \in \Sigma$,

$$P^{(\ell)} \to P[t_0]$$

Initiality, typed

Definition (Representation of signature Σ over types T)

- Monad P on Set^T
- Morphism of modules over P for each arity $s \in \Sigma$

Theorem (Zsidó)

The category of representations of Σ has an initial object.

Initiality, typed

Definition (Representation of signature Σ over types T)

- Monad P on Set^T
- Morphism of modules over P for each arity $s \in \Sigma$

Problem: The set T of types is hard–coded.

$\rightsquigarrow$ does not account for $i : \mathrm{PCF} \to \mathrm{LC}$

Initiality, typed

Definition (Representation of signature Σ over types T)

- Monad P on Set^T
- Morphism of modules over P for each arity $s \in \Sigma$

Problem: The set T of types is hard–coded.

$\rightsquigarrow$ does not account for $i : \mathrm{PCF} \to \mathrm{LC}$

Definition II (Representation)

+ Set U
+ $g : T \to U$
- Monad P on Set^U
- Representation of "Σ transported along g" in P

Ex.: A Representation of PCF is given by...

... a representation of the types of PCF ...

- Set U
- $Nat : U \qquad Bool : U \qquad (\Rightarrow) : U \times U \to U$

... and one of the terms of PCF:

- Monad $\quad P : Set^U \to Set^U$
- $Abs_{u,v} : P^{(u)}[v] \to P[u \Rightarrow v] \qquad \forall\ u\ v \in U$
- $App_{u,v} : P[u \Rightarrow v] \times P[u] \to P[v] \qquad \forall\ u\ v \in U$
- $Fix_u : P[u \Rightarrow u] \to P[u] \qquad \forall\ u \in U$
- ...

Ex.: A Representation of PCF is given by...

... a representation of the types of PCF ...

- Set U
- $Nat : U \qquad Bool : U \qquad (\Rightarrow) : U \times U \to U$

... and one of the terms of PCF:

- Monad $\quad P : Set^U \to Set^U$
- $Abs_{u,v} : P^{(u)}[v] \to P[u \Rightarrow v] \qquad \forall\ u\ v \in U$
- $App_{u,v} : P[u \Rightarrow v] \times P[u] \to P[v] \qquad \forall\ u\ v \in U$
- $Fix_u : P[u \Rightarrow u] \to P[u] \qquad \forall\ u \in U$
- ...

Ex.: A Representation of PCF in LC...

... a representation of the types of PCF

- $U := \{*\}$
- $Nat := *$ $\quad$ $Bool := *$ $\quad$ $x \Rightarrow y := *$

... and one of the terms of PCF:

- Monad $\quad$ LC : $Set^{\{*\}} \to Set^{\{*\}}$
- $Abs_{u,v} : \mathrm{LC}^u[v] \to \mathrm{LC}[u \Rightarrow v]$ $\quad$ $\forall\, u\ v \in \{*\}$
- $App_{u,v} : \mathrm{LC}[u \Rightarrow v] \times \mathrm{LC}[u] \to \mathrm{LC}[v]$ $\quad$ $\forall\, u\ v \in \{*\}$
- $Fix_u : \mathrm{LC}[u \Rightarrow u] \to \mathrm{LC}[u]$ $\quad$ $\forall\, u \in \{*\}$
- ...

Ex.: A Representation of PCF in LC...

... a representation of the types of PCF

- $U := \{*\}$
- $Nat := *$ $\quad$ $Bool := *$ $\quad$ $x \Rightarrow y := *$

... and one of the terms of PCF:

- Monad $\quad \mathrm{LC} : Set \to Set$
- $Abs : \mathrm{LC}' \to \mathrm{LC}$
- $App : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$
- $Fix : \mathrm{LC} \to \mathrm{LC}$
- ...

Initiality w. Type Change

Definition (Signature)

- a signature S for types
- a signature Σ for terms over those types

Definition (Representation of (S, Σ))

- a representation of S in a set U
- a representation of Σ in a monad P on Set^U

Theorem

The category of representations of (S, Σ) *has an initial object.*

Initiality w. Type Change

Definition (Signature)

- a signature S for types
- a signature Σ for terms over those types

Definition (Representation of (S, Σ))

- a representation of S in a set U
- a representation of Σ in a monad P on Set^U

Theorem

The category of representations of (S, Σ) *has an initial object.*

Translation from PCF to LC by Initiality

A representation of PCF in LC

specifies an initial morphism of representations

Representing *Fix* in LC

- $M \mapsto App(\Theta, M) : \mathrm{LC} \to \mathrm{LC}$ or
- $M \mapsto App(\mathbf{Y}, M) : \mathrm{LC} \to \mathrm{LC}$

yields the two aforementioned translations

Translation from Classical to Intuitionistic Logic

Curry–Howard Isomorphism

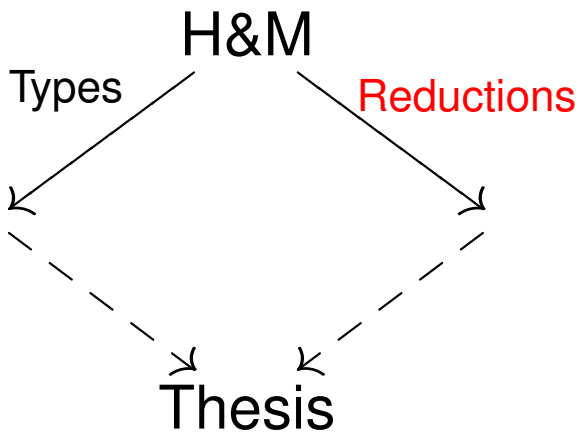
- Propositions as Types
- Proofs as Terms
- Propositional Logic as Simple Type System

Double Negation Translation $g : CL \to IL$ via Initiality

- Translation of Propositions is endo, but not identity
- Type Safety of Translation:

$$\Gamma \vdash_C A \quad \text{implies} \quad \Gamma^g \vdash_I A^g$$

And now



Outline

Introduction

Untyped Syntax with Binding

Representations, Take I

Integrating Substitution

Adding Types

Fixed Types

Varying Types

Adding Reduction Rules

LCβ as Relative Monad

Reps. in Relative Monads

Combining Types and Reduction

Goals

- "2–Signature" to specify language with reduction rules

$\rightsquigarrow$ Category of Representations of a 2–Signature

- Monads? Modules over Monads?

$\rightsquigarrow$ exhibit Initial Object: Syntax with Reduction Rules

$+$ encode as much structure as possible in representations

Goals

- “2–Signature” to specify language with reduction rules

$\rightsquigarrow$ Category of Representations of a 2–Signature

- Monads? Modules over Monads?

$\rightsquigarrow$ exhibit Initial Object: Syntax with Reduction Rules

$+$ encode as much structure as possible in representations

Restrict ourselves to untyped syntax

for now

2–Signatures

Definition (2–Signature)

- a (1–)signature Σ
- a set of Σ–inequations

Example (Σ_{LC}, β)

- 1–signature Σ_{LC}
- $\lambda x.M(N) \quad \leq \quad M[x := N]$

Propagation into Subterms

follows from formalism ?

▸ Click here to hurry up

Which Monads to Use?

$$\lambda x.M(N) \rightsquigarrow M[x := N]$$

- X Terms modulo Relations, Quotienting ?
- X Monads $Pre \to Pre$?
- ✓ *Relative* Monads $Set \to Pre$

Relative Monad on Functor J, Altenkirch et al '10

- \+ $J : \mathcal{C} \to \mathcal{D}$
- $P : \mathcal{C} \to \mathcal{D}$
- $\eta : J \Rightarrow P$
- $\sigma_{X,Y} : \mathcal{D}(JX, PY) \to \mathcal{D}(PX, PY)$ + substitution properties

Example : LCβ as Relative Monad

- $\Delta : \mathit{Set} \to \mathit{Pre}, \quad X \mapsto (X, \mathit{diagonal})$
- $V \mapsto \mathsf{LC}\beta(V) := (\mathsf{LC}(V), \beta^*_V)$
- $\mathit{Var} : \Delta \Rightarrow \mathsf{LC}\beta$
- $(\gg\!=)_{V,W} : \mathit{Pre}(\Delta V, \mathsf{LC}\beta(W)) \to \mathit{Pre}\big(\mathsf{LC}\beta(V), \mathsf{LC}\beta(W)\big)$

Example : LCβ as Relative Monad

- $\Delta : \mathit{Set} \to \mathit{Pre}, \quad X \mapsto (X, \mathit{diagonal})$
- $V \mapsto \mathsf{LC}\beta(V) := (\mathsf{LC}(V), \beta^*_V)$
- $\mathit{Var} : \Delta \Rightarrow \mathsf{LC}\beta$
- $(\gg\!=)_{V,W} : \mathit{Pre}(\Delta V, \mathsf{LC}\beta(W)) \to \mathit{Pre}\big(\mathsf{LC}\beta(V), \mathsf{LC}\beta(W)\big)$

Adjunction $\Delta \dashv U$ with forgetful $U : \mathit{Pre} \to \mathit{Set}$

Example : LCβ as Relative Monad

- $\Delta : \mathit{Set} \to \mathit{Pre}, \quad X \mapsto (X, \mathit{diagonal})$
- $V \mapsto \mathsf{LC}\beta(V) := (\mathsf{LC}(V), \beta^*_V)$
- $\mathit{Var} : \Delta \Rightarrow \mathsf{LC}\beta$
- $(\gg\!=)_{V,W} : \mathit{Pre}(\Delta V, \mathsf{LC}\beta(W)) \to \mathit{Pre}\big(\mathsf{LC}\beta(V), \mathsf{LC}\beta(W)\big)$

Adjunction $\Delta \dashv U$ with forgetful $U : \mathit{Pre} \to \mathit{Set}$

- $\mathit{Var} : \mathit{Id} \Rightarrow \mathsf{LC}$
- $(\gg\!=)_{V,W} : \mathit{Set}(V, \mathsf{LC}(W)) \to \mathit{Pre}\big(\mathsf{LC}\beta(V), \mathsf{LC}\beta(W)\big)$

Representations of 1–Signatures

Modules over Monads...

carry over to modules over relative monads

Definition (Representation of 1–Signature Σ)

- Relative Monad $P : Set \to Pre$ on Δ
- Morphism of modules over P for each arity $s \in \Sigma$,

$$P^{(s)} \to P$$

Jump to Def. of Inequation

Representations of 2–Signatures

Definition (Representation of a 2–signature (Σ, A))

- Representation R of Σ
- s.t. R verifies each inequation $\alpha \leq \gamma$ of A

Example (Representation of $(\Sigma_{\mathrm{LC}}, \beta)$)

- Representation (P, Abs, App) of Σ_{LC}
- $\forall\ V \in Set$,
  $\forall\ M \in P(V+1),\ \forall\ N \in PV$,

$$App(Abs(M), N) \quad \leq \quad M[* := N]$$

Initiality for 2–Signatures

Representations of (Σ, A)

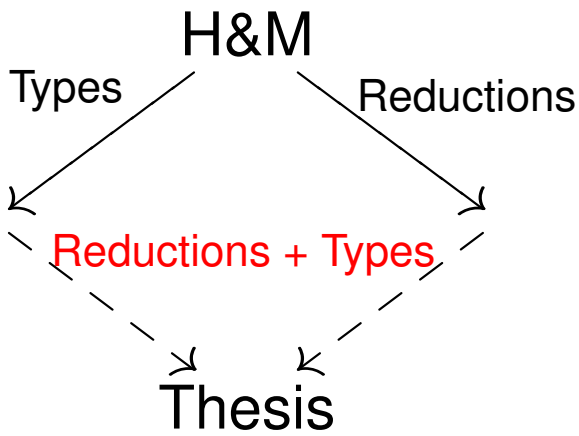
$$Rep(\Sigma, A) \quad \subseteq \quad Rep(\Sigma) \qquad \text{full subcategory}$$

Theorem (in Coq)

The category of representations of (Σ, A) *has an initial object.*

▸ Jump to Proof

And now



Outline

2–Signatures for Simple Type Systems

2–Signature

- 1–Signature (S, Σ) for types and terms over those types
- Reduction Rules on Terms

Theorem

The category Rep$((S, \Sigma), A)$ *has an initial object.*

In Coq: PCF $\to$ LC

- via initiality
- compatible with reduction
- painful due to use of dependent types

Future work

- More sophisticated type systems, e.g., dependent types, polymorphism
- more term formers, e.g. flattening $\mu_X : T(TX) \to TX$
- syntactic specification of inequations
- richer modelling of reduction (graphs, categories,...)
- enriching Coq implementation with useful features, make specification of signature as easy as possible

Related Work


Thorsten Altenkirch, James Chapman and Tarmo Uustalu.
Monads Need Not Be Endofunctors. 2010.


Thorsten Altenkirch and Bernhard Reus.
Monadic Presentations of Lambda Terms Using Generalized Inductive Types. 1999.


Marcelo P. Fiore and Chung-Kil Hur.
Second-order equational logic (extended abstract). 2010.


André Hirschowitz and Marco Maggesi.
Modules over monads and linearity. 2007.


Julianna Zsidó.
PhD thesis, University of Nice, France, 2010.

Thanks for your attention

Module over a Monad $P : \mathcal{C} \to \mathcal{C}$

Module M over P with codomain $\mathcal{D}$

- $M : \mathcal{C} \to \mathcal{D}$
- $\varsigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{D}(MX, MY) +$ substitution properties

Tautological Module

$M := P$ and $\varsigma := \sigma$

Product Module $M \times N$

Pointwise, if $\mathcal{D}$ has a product

Derived Module of M

$X \mapsto M(X + 1)$ substitution adjusted to extended context

(for suitable $\mathcal{C}$)

Specifying a language — User perspective

Specification of Syntax

```
Inductive Lambda_index := ABS | APP.
Definition Lambda : Signature := {|
  sig_index := Lambda_index ;
  sig := fun x : Lambda_index => match x with
                                 | ABS => [[ 1 ]]
                                 | APP => [[ 0 ; 0]]
                                 end |}.
```

specifies syntax, certified substitutition, iteration principle from initiality

Specifying a language — User perspective II

Semantics — Inequations

```
Definition beta_rule : ineq_classic Lambda := {|
  half_eq_l := beta_half_eq ;
  half_eq_r := subst_half_eq Lambda |}.

Definition Lambda_beta_Cat := INEQ_REP
   (S:=Lambda)(A:=unit)(fun x : unit => beta_rule
      ).
```

but definition of **beta_half_eq** and **subst_half_eq** involves some proof (ca. 20 lines each).

Monadic Substitution Monotone?

For LC we have

1. $M \leq N$ implies $M[* := A] \leq N[* := A]$
2. $A \leq B$ implies $M[* := A] \leq M[* := B]$.

Only property 1 coded in definition of Relative Monad!

Encoding Property 2

- Pre enriched over itself

$$f \leq g \overset{def}{\iff} \mathit{forall}\ x,\ f(x) \leq g(x)$$

- consider relative Monads $P : Set \overset{\Delta}{\to} Pre$ s.t.

$$\sigma^{P}_{X,Y} : Pre(\Delta X, PY) \to Pre(PX, PY) \quad \text{functorial}$$

Proof of Initiality for (Σ, A)

Proof.

- start with initial object $\hat{\Sigma}$ of $Rep(\Sigma)$ (diag. preorder)
- define preorder $\leq_A$ on elements of $\hat{\Sigma}$ by

$$x \leq_A y \overset{def}{\iff} \forall R \in Rep(\Sigma, A),\ i_R(x) \leq i_R(y)$$

- yields rel. monad $\hat{\Sigma}_A$ on Δ
- representation structure of $\hat{\Sigma}$ can be used for $\hat{\Sigma}_A$
- $\hat{\Sigma}_A$ verifies A

□

Back to Theorem

β — an Inequation over Σ_{LC}

Meta–Variables

$$\lambda M(N) \rightsquigarrow M[* := N]$$

β — an Inequation over Σ_{LC}

Meta–Variables through Abstraction

$$\texttt{app\_of\_abs} : (M, N) \mapsto \lambda M(N)$$

β — an Inequation over Σ_{LC}

Meta–Variables through Abstraction

$$\texttt{app\_of\_abs} : (M, N) \mapsto \lambda M(N)$$

$$\texttt{subst} : (M, N) \mapsto M[* := N]$$

β — an Inequation over Σ_{LC}

Meta–Variables through Abstraction

$$\begin{aligned} \texttt{app\_of\_abs} &: (M, N) \mapsto \lambda M(N) \\ \texttt{subst} &: (M, N) \mapsto M[* := N] \\ \mathrm{LC}(V+1) \times \mathrm{LC}(V) &\to \mathrm{LC}(V) \end{aligned}$$

β — an Inequation over Σ_{LC}

Meta–Variables through Abstraction

$$\begin{aligned} \texttt{app\_of\_abs} &: (M, N) \mapsto \lambda M(N) \\ \texttt{subst} &: (M, N) \mapsto M[* := N] \\ \mathrm{LC}(V+1) \times \mathrm{LC}(V) &\to \mathrm{LC}(V) \\ P(V+1) \times P(V) &\to P(V) \end{aligned}$$

for any representation $R = (P, Abs, App)$ of Σ_{LC}

β — an Inequation over Σ_{LC}

Meta–Variables through Abstraction

$$\begin{aligned}
\texttt{app_of_abs} &: (M, N) \mapsto \lambda M(N) \\
\texttt{subst} &: (M, N) \mapsto M[* := N] \\
\mathrm{LC}(V+1) \times \mathrm{LC}(V) &\to \mathrm{LC}(V) \\
P(V+1) \times P(V) &\to P(V) \\
P' \times P &\to P
\end{aligned}$$

for any representation $R = (P, Abs, App)$ of Σ_{LC}

Example: Beta rule $\beta_l \leq \beta_r$ over Σ_{LC}

Source arity: $\mathrm{dom}(P) := P' \times P$

Target arity: $\mathrm{cod}(P) := P$

Substitution β_r

- $\mathtt{subst}(P, Abs, App) : P' \times P \to P$
- definable for any signature

Beta–sensitive terms β_l

- $\mathtt{app\_of\_abs}(P, Abs, App) : P' \times P \to P$
- $\mathtt{app\_of\_abs}(P, Abs, App) := App \circ (Abs \times Id)$

Inequations over Σ

Source and Target by Functors

$$\mathrm{dom}, \mathrm{cod} : \mathit{Mon}(\Delta) \to \mathit{LMod}(\Delta, \mathit{wPre})$$

Half–Equation α

associates to any representation $R = (P, \mathit{rep})$ a morphism of modules

$$\alpha^R : \mathrm{dom}(P) \to \mathrm{cod}(P)$$

Inequation $\alpha \leq \gamma$

a pair of parallel half–equations (α, γ)

▸ Back to Rep. of 1–Signature