Initiality for Typed Syntax and Semantics

Benedikt Ahrens

Université Nice Sophia Antipolis

Séminaire GdT Sémantique, PPS, Paris

December 13, 2011

A Translation from *PCF* to *LC*

PCF

- Typed Language
- Abstraction
  Application
  Fixpoint Operator
  Arithmetic &
  Logic Consts.

Lambda Calculus

- Untyped
- Abstraction
  Application

Translations $i : PCF \rightarrow LC$

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

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

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

$$\ldots$$

Translation, mathematically

Mathematical structure of

$$PCF \to LC \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $PCF \to LC$ morphism in some category ?

Translation, mathematically

Mathematical structure of

$$PCF \to LC \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $PCF \to LC$ initial morphism in some category ?

Translation, mathematically

Mathematical structure of

$$PCF \to LC \qquad ?$$

Challenges:

- varying types
- capture compatibility with substitution + reduction

More precisely

- $PCF \to LC$ initial morphism in some category ?
- Extra structure on LC to specify different such translations

What is Initial Semantics?

Ingredients:

- Signature S

$\rightsquigarrow$ Category of Semantics of S

$\rightsquigarrow$ Initial Semantics $\Sigma(S)$ — the Syntax of S

Features:

- Variable Binding
- Typing
- Reductions

Why?

- Definition of inductive type
- Categorical iteration operator (Ex. `fold`)

Adding a Feature

$\hat{=}$

new notion of Signature and Semantics

Outline

Outline

Goals

- define “Signature” to specify untyped language with binding
- $\rightsquigarrow$ define Category of Semantics (Representations) of a Signature
- $\rightsquigarrow$ exhibit Initial Object: Syntax
- $+$ encode as much structure as possible in Representations

Has been done by

- Fiore, Plotkin, Turi
- Hofmann
- Gabbay, Pitts
- Hirschowitz, Maggesi

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

Encodings of variable binding

- Nominal Approach – Named Abstraction

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

- Higher-Order Abstract Syntax (HOAS)

$$lam : (T \to T) \to T$$

- Nested Datatype

$$lam : T(X + 1) \to T(X)$$

Example: Lambda Calculus

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

Cat. of Reps: ???

Initial:

```
Inductive LC (V : Set) : Set :=
  | Var : V → LC (V)
  | Abs : LC (V + 1) → LC (V)
  | App : LC (V) × LC (V) → LC (V)
```

Abstracting from *LC*, a Representation of Λ is...

$$\begin{cases} F : Set \to Set \\ Var : Id \Rightarrow F \ , \quad Abs : \ldots , \quad App : \ldots \end{cases}$$

Goal: Integrate Substitution...

Integrate Substitution

Adding Substitution & its Properties

Term Models: Functor $\rightsquigarrow$ Monad

Constructors: Nat. Transformation $\rightsquigarrow$ Morphism of Modules

Monad $=$ Functor $+$ "variables–as–terms" $+$ Substitution:

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

Term Model $\equiv$ Monad

Example ($LC : Set \to Set$, Altenkirch & Reus)

- $V \mapsto LC(V)$
- $v \mapsto Var_V(v) \in LC(V)$
- $(\gg\!=) : LC(V) \times (V \to LC(W)) \to LC(W)$

Properties encoded in monadic axioms:

- $x \gg\!= Var \;\; == \;\; x$
- $Var(v) \gg\!= f \;\; == \;\; f(v)$
- $x \gg\!= f \gg\!= g \;\; == \;\; x \gg\!= (\lambda v.f(v) \gg\!= g)$

Modules over Monads

Goal: express the property, e.g. for *Abs* and *App*

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

Module over a Monad

- Functor + Substitution (often induced by Monad)
- Domain and Codomain of Constructor = Module

Morphism of Modules

- Natural Transformation + Compatibility with Substitution
- Constructor = Morphism of Modules

Modules over Monads

Goal: express the property, e.g. for *Abs* and *App*

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

Module over a Monad

- Functor + Substitution (often induced by Monad)
- Domain and Codomain of Constructor = Module

Morphism of Modules

- Natural Transformation + Compatibility with Substitution
- Constructor = Morphism of Modules

Morphisms of Modules for LC

Modules over LC

$$LC^* : V \mapsto LC(V + 1) \qquad LC : V \mapsto LC(V)$$

$$LC \times LC : V \mapsto LC(V) \times LC(V)$$

Morphisms of Modules over LC

$$Abs : LC^* \to LC$$

$$App : LC \times LC \to LC$$

Representations of a Signature

Definition (Representation of Signature S)

- Monad $P : Set \to Set$
- Morphism of Modules over P for each Arity $s \in S$

Example (Representation of Λ)

- Monad $P : Set \to Set$
- $app : P \times P \to P \quad , \quad abs : P^* \to P$

Initial Semantics, untyped

Theorem (Hirschowitz & Maggesi)

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

Example

(LC, App, Abs) is the initial representation of Λ

Outline

Goals

- define “Signature” to specify simply–typed language with binding

$\rightsquigarrow$ define Category of Representations of a Signature

- built from Monads and Modules over Monads

$\rightsquigarrow$ exhibit Initial Object: Syntax

$+$ encode as much structure as possible in Representations

Goals

- define “Signature” to specify simply–typed language with binding

$\rightsquigarrow$ define Category of Representations of a Signature

- built from Monads and Modules over Monads

$\rightsquigarrow$ exhibit Initial Object: Syntax

$+$ encode as much structure as possible in Representations

We present 2 solutions:

1. with fixed set of sorts
2. with varying sets of sorts

Example: Simply–Typed LC

Types: $T \quad ::= \quad * \quad | \quad T \Rightarrow T$

Signature: $T\Lambda = \left\{ (abs_{s,t} : (s)t \to s \Rightarrow t)_{s,t \in T} \ , (app_{s,t} : \ldots)_{s,t} \right\}$

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) × TLC(V)(s) → ...
```

Example: Simply–Typed LC

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

Signature: $T\Lambda = \left\{ (abs_{s,t} : (s)t \to s \Rightarrow t)_{s,t \in T} \;, (app_{s,t} : \ldots)_{s,t} \right\}$

Cat. of Reps : Monads ?

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) × TLC(V)(s) → ...
```

- Monad $\quad TLC : Set^T \to Set^T$

Example: Simply–Typed LC

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

Signature: $T\Lambda = \left\{ (abs_{s,t} : (s)t \to s \Rightarrow t)_{s,t \in T} \;, (app_{s,t} : \ldots)_{s,t} \right\}$

Cat. of Reps : Monads ? Modules ?

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) × TLC(V)(s) → ...
```

- Monad $TLC : Set^T \to Set^T$
- Fibre Module $TLC[t]$ over monad TLC

$$V \mapsto TLC(V)(t) : Set^T \to Set$$

Initiality, typed

Definition (Representation of a Signature S over types T)

- Monad L on Set^T
- Morphism of Modules over L for each Arity $s \in S$

Simply–typed LC

- $P : Set^T \to Set^T$
- $abs_{s,t} : P^s[t] \to P[s \Rightarrow t]$ $\quad$ $app_{s,t} : P[s \Rightarrow t] \times ....$

Initiality, typed

Definition (Representation of a Signature S over types T)

- Monad L on Set^T
- Morphism of Modules over L for each Arity $s \in S$

Theorem (Zsidó)

The Category of Representations of S has an initial object.

Implemented in Coq

B. A. and J. Zsidó, JFR, 2011

Initiality, typed

Definition (Representation of a Signature S over types T)

- Monad L on Set^T
- Morphism of Modules over L for each Arity $s \in S$

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

Initiality, typed

Definition (Representation of a Signature S over types T)

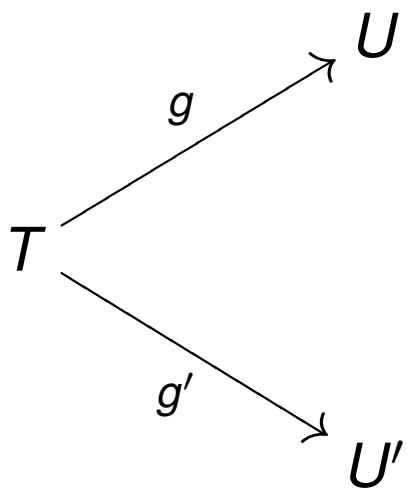
- Monad L on Set^T
- Morphism of Modules over L for each Arity $s \in S$

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

Definition II (Representation)

- Set U
- $g : T \to U$ morphism of types
- Monad L on Set^U
- representation of "S transported along g" in L

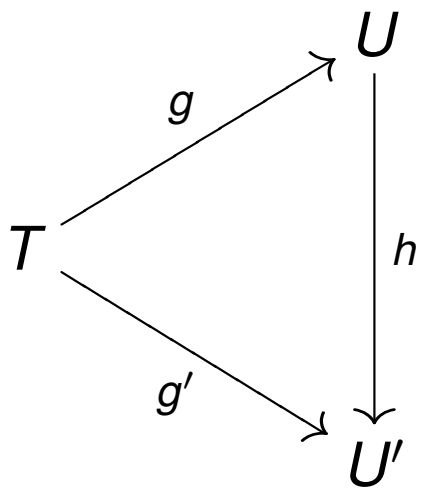
Morphisms of Representations — Changing Types



$$Set^{U} \xrightarrow{L} Set^{U}$$

$$Set^{U'} \xrightarrow[L']{} Set^{U'}$$

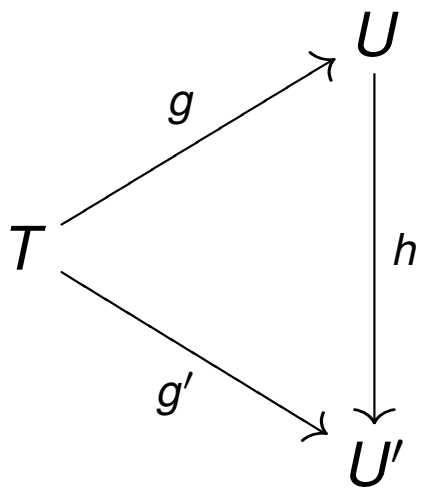
Morphisms of Representations — Changing Types

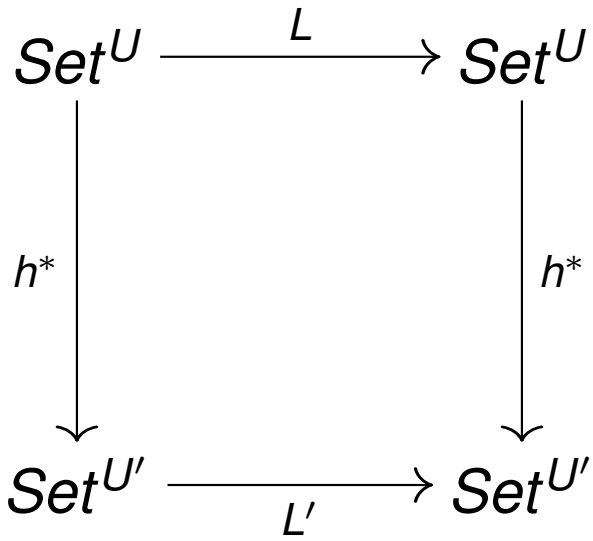


$$\begin{array}{ccc} Set^{U} & \xrightarrow{L} & Set^{U} \\ & & \\ Set^{U'} & \xrightarrow[L']{} & Set^{U'} \end{array}$$

- $h : U \to U'$

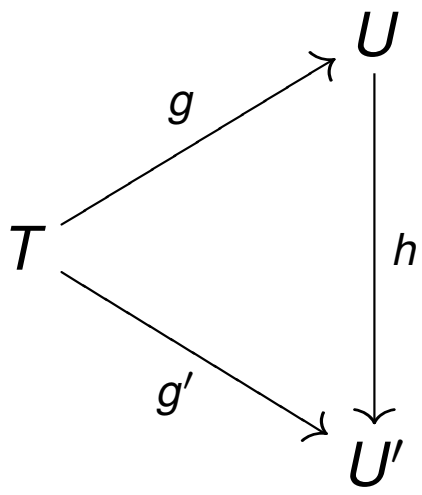
Morphisms of Representations — Changing Types

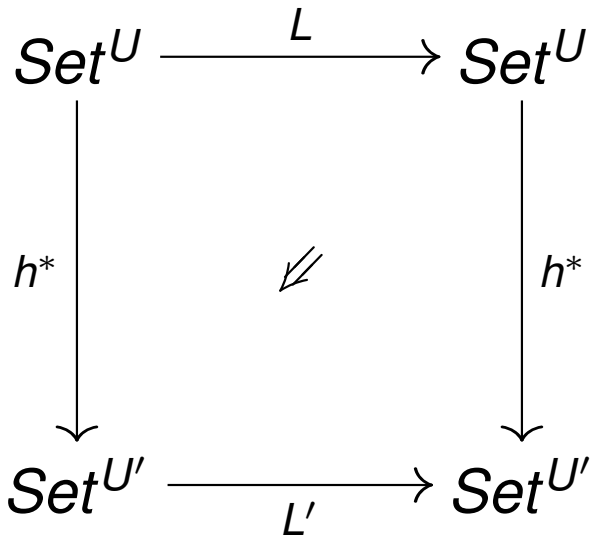




- $h : U \to U'$
- induces $h^* : Set^U \to Set^{U'}$

Morphisms of Representations — Changing Types





- $h : U \to U'$
- induces $h^* : Set^U \to Set^{U'}$
- Morphism of Representations:

$$h^* \circ L \Rightarrow L' \circ h^*$$

+ commutative diagrams

Initiality w. Type Change

Theorem (Generalized Initiality)

The category of representations of S has an initial object.

Translations from *PCF* to *LC*

Specified by Representation of *PCF* in *LC*

- map of types $T_{PCF} \to \{*\}$
-

Arity	**Rep in LC**
$app_{s,t} : \ldots$	$App : LC \times LC \to LC$
$abs_{s,t} : \ldots$	$Abs : LC^* \to LC$
$true : Bool$	$True : LC$
$Fix_t : (t \Rightarrow t) \to t$	$\lambda M.\Theta(M) : LC \to LC$ or $\lambda M.\mathbf{Y}(M) : LC \to LC$
...	...

Yields different translations $PCF \to LC$

Outline

Goals

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

⇝ define Category of Representations of a 2–Signature
  - built from Monads and Modules over Monads

⇝ exhibit Initial Object: Syntax with Reduction Rules

\+ encode as much structure as possible in Representations

Goals

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

⇝ define Category of Representations of a 2–Signature
  - built from Monads and Modules over Monads

⇝ 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 S
- a set of S–inequations

Example ($\Lambda\beta$)

- signature Λ
- $app \circ (abs, id) \quad \leq \quad \_[* := \_]$

Propagation into Subterms

automatically

Which Monads to Use?

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

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

Relative Monad $=$ Functor $+$ Monad–like Data

- $+$ $F : \mathcal{C} \to \mathcal{D}$
- $P : \mathcal{C} \to \mathcal{D}$
- $\eta_X : \mathcal{D}(FX, PX)$
- $\sigma : \mathcal{D}(FX, PY) \to \mathcal{D}(PX, PY)$ $+$ substitution properties

Example : LCβ as Relative Monad on Δ

$$\Delta : \mathit{Set} \to \mathit{Ord}, \quad X \mapsto (X, \mathit{diagonal})$$

LCβ as Relative Monad on Δ:

- $V \mapsto \mathit{LC}\beta(V) := (\mathit{LC}(V), \beta^*)$
- $\mathit{Var}_V : \mathit{Ord}(\Delta V, \mathit{LC}\beta(V))$
- $(\gg\!=) : \mathit{Ord}(\Delta V, \mathit{LC}\beta(W)) \to \mathit{Ord}\big(\mathit{LC}\beta(V), \mathit{LC}\beta(W)\big)$

Example : LCβ as Relative Monad on Δ

$$\Delta : \mathit{Set} \to \mathit{Ord}, \quad X \mapsto (X, \mathit{diagonal})$$

LCβ as Relative Monad on Δ:

- $V \mapsto \mathit{LC}\beta(V) := (\mathit{LC}(V), \beta^*)$
- $\mathit{Var}_V : \mathit{Ord}(\Delta V, \mathit{LC}\beta(V))$
- $(\gg\!=) : \mathit{Ord}(\Delta V, \mathit{LC}\beta(W)) \to \mathit{Ord}\big(\mathit{LC}\beta(V), \mathit{LC}\beta(W)\big)$

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

Example : LCβ as Relative Monad on Δ

$$\Delta : \mathit{Set} \to \mathit{Ord}, \quad X \mapsto (X, \mathit{diagonal})$$

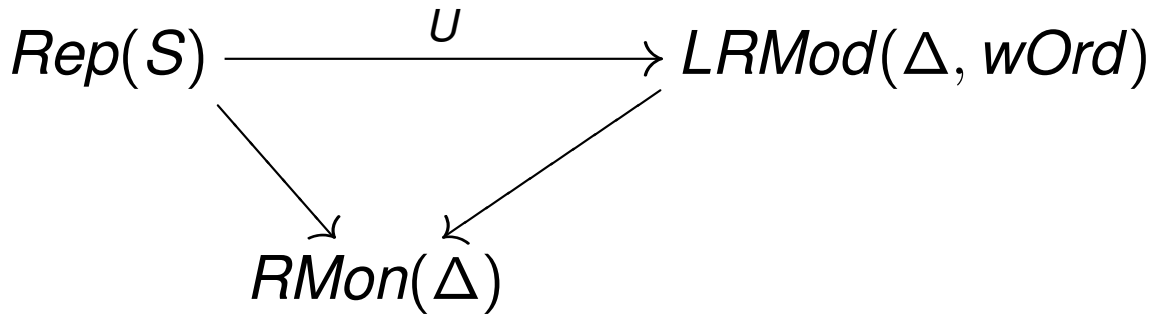
LCβ as Relative Monad on Δ:

- $V \mapsto LC\beta(V) := (LC(V), \beta^*)$
- $Var_V : Set(V, LC(V))$
- $(\gg\!=) : \quad Set(V, LC(W)) \quad \to Ord\big(LC\beta(V), LC\beta(W)\big)$

Adjunction $\Delta \dashv U$ with forgetful $U : Ord \to Set$

Inequations over S

S–Module U



Half–Equation $\alpha : U \to V$

a natural transformation $U \to V$

Inequation over S

- two parallel half–equations
- $\alpha \leq \gamma : U \to V$

Example: Beta rule $\beta_l \leq \beta_r$ over Λ

- Source Λ–Module $U(R) := R^* \times R$
- Target Λ–Module $V(R) := R$

Substitution β_r

- $\_[* := \_](R) : R^* \times R \to R$
- definable for any signature

Beta–sensitive terms β_l

- $Rep(S) \ni R \mapsto App^R \circ (Abs^R, id^R) : R^* \times R \to R$

Representations of 2–Signatures

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

- Representation R of S in Relative Monad on Δ
- s.t. R verifies each inequation of A

Example (Representation of $\Lambda\beta$)

- Rep. of Λ $\left\{ \begin{array}{l} P : Set \overset{\Delta}{\to} Ord \\ App : P \times P \to P \quad , \quad Abs : P^* \to P \end{array} \right.$
- $App \circ (Abs, Id) \quad \leq \quad \_[* := \_]$ pointwise

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

Initiality for 2–Signatures

Theorem (in Coq)

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

Proof.

- start with initial object Σ of $Rep(S)$ (diag. preorder)
- define preorder $\leq_A$ on elements of Σ by

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

- yields rel. monad Σ_A on Δ
- representation structure of Σ can be used for Σ_A
- Σ_A verifies A

2–Signatures for Simple Type Systems

2–Signature

- Signature T for Types
- Signature S for Terms over those Types
- Reduction Rules on Terms

Theorem (on paper)

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

In Coq: $PCF \to LC$

- via initiality
- compatible with reduction
- painful due to intrinsic typing

Future work

- More sophisticated type systems (dependent types, polymorphism,...)
- more term formers, e.g. flattening $\mu_X : T(TX) \to TX$
- richer modelling of reduction (graphs, categories,...)
- enriching Coq implementation with useful features, e.g. *computable* reduction
- ...

Future work

- More sophisticated type systems (dependent types, polymorphism,...)
- more term formers, e.g. flattening $\mu_X : T(TX) \to TX$
- richer modelling of reduction (graphs, categories,...)
- enriching Coq implementation with useful features, e.g. *computable* reduction
- ...

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

- *Ord* enriched over itself

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

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

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