Modules over monads for operational semantics

Benedikt Ahrens

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier

Journées GDR LTP
2013–11–25

Participation sponsored by project **CLIMT** (ANR-11-BS02-016)

Universal algebra

Goal of Universal algebra

specify

- terms
- equations of terms

by a signature

Signature of group theory:

- $e : 0$
- $(\_)^{-1} : 1$
- $* : 2$

$+$ equations

A group G is a set G with

operations and equations:

- $e : G$
- $(\_)^{-1} : G \to G$
- $(*) : G \times G \to G$
- $e * g = g$
- $g * g^{-1} = e$
- $\ldots$

The **free** group of a set X

inductively generated by the signature.

Goal: Universal algebra for programming languages

Goal: signatures for languages **with variable binding**

- want **in**equalities rather than equalities between terms

$\rightsquigarrow$ model reductions more faithfully

- characterize language specified by signature categorically

Signatures specifying

- types + terms
- reduction rules

Characterization of generated language

- category of **models** of signature
- language as **initial** model

Universal algebra with variable binding: related work

- Gabbay & Pitts
  - nominal syntax, no equations
- Hofmann, Miculan & Scagnetto
  - HOAS, no equations
- Fiore
  - algebraic de Bruijn, equations
- Hirschowitz & Maggesi
  - algebraic de Bruijn, equations

My work

- inspired by Hirschowitz & Maggesi
- adapted to
  - **in**equalities
  - simple typing

Content of this talk

In this talk:

- **Models** of the untyped λ-calculus with β-reduction
- the λ-calculus as **initial** such model

Not in this talk — but elsewhere:

- General notion of **signature** for languages with reduction
- Signatures and models for **simply-typed** languages with term reductions

First: take a look at H & M's work on λ-calculus without reductions

1. Review of Hirschowitz & Maggesi's work
2. Integrating reduction rules

λ-calculus as a monad on Set (Altenkirch & Reus '99)

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

- Monad: set of terms over free variables

$$\mathrm{LC}(V) = \text{ set of lambda terms over free variables in set } V$$

- Monad structure: "Variables–as–terms" and substitution

$$\mathit{Var}_V : V \to \mathrm{LC}(V)$$
$$(\gg\!=)_{V,W} : \mathrm{LC}(V) \times \big(V \to \mathrm{LC}(W)\big) \to \mathrm{LC}(W)$$

- Monad axioms: properties of variable substitution

Constructors and substitution

Goal: capture the interplay between constructors and substitution

$$\begin{aligned} App(M, N) \gg\!= f &= App(M \gg\!= f, N \gg\!= f) \\ Abs(M) \gg\!= f &= Abs\big(M \gg\!= \mathrm{shift}(f)\big) \end{aligned}$$

First try: are *Abs* and *App* **monad morphisms**?

$$Abs : \mathrm{LC}^* \to \mathrm{LC} \qquad \text{with } \mathrm{LC}^* : V \mapsto \mathrm{LC}(\mathrm{option}\ V)$$
$$App : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$$

Fails:

- LC^* a monad, but *Abs* **not** monad morphism
- $\mathrm{LC} \times \mathrm{LC}$ not a monad in a reasonable sense

Modules over monads

Modules over monads generalize monads...

and the functors

$$\mathsf{LC} : V \mapsto \mathsf{LC}(V)$$
$$\mathsf{LC}^* : V \mapsto \mathsf{LC}(\text{option } V)$$
$$\mathsf{LC} \times \mathsf{LC} : V \mapsto \mathsf{LC}(V) \times \mathsf{LC}(V)$$

are modules over the monad LC.

and the constructors are **module morphisms**:

$$\mathit{Abs} : \mathsf{LC}^* \to \mathsf{LC}$$
$$\mathit{App} : \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC}$$

Expresses precisely distributivity of substitution over *App* and *Abs*.

Initiality for pure syntax

Summary: we have

- monad LC
- module morphisms

$$App : \mathrm{LC} \times \mathrm{LC} \to \mathrm{LC}$$
$$Abs : \mathrm{LC}^* \to \mathrm{LC}$$

Def.: model of λ-calculus

- monad P
- module morphisms

$$App^P : P \times P \to P$$
$$Abs^P : P^* \to P$$

Theorem (Hirschowitz & Maggesi)

(LC, App, Abs) *is the initial object in the category of models.*

Now: integrating reduction rules.

1. Review of Hirschowitz & Maggesi's work

2. Integrating reduction rules

Integrating Semantics, untyped

Goal: define "model of λ-calculus with β-reduction"

such that λ-calculus with

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

is the initial model.

Main question:

How should "$\rightsquigarrow$" be modelled mathematically?

- X Terms modulo relations, quotienting
- X Monads $\quad \mathsf{Pre} \to \mathsf{Pre}$
- ✓ *Relative* Monads $\quad \mathsf{Set} \to \mathsf{Pre}$

  with Pre := category of preordered sets

Relative monads

Definition (Monad on $\mathcal{C}$)

- $P : \mathcal{C} \to \mathcal{C}$
- $\eta_X : \mathcal{C}(X, PX)$
- $\sigma_{X,Y} : \mathcal{C}(X, PY) \to \mathcal{C}(PX, PY)$
- monad laws

Definition (Relative monad over a functor F — Altenkirch et al.)

+ $F : \mathcal{C} \to \mathcal{D}$
- $P : \mathcal{C} \to \mathcal{D}$
- $\eta_X : \mathcal{D}(FX, PX)$
- $\sigma_{X,Y} : \mathcal{D}(FX, PY) \to \mathcal{D}(PX, PY)$
- relative monad laws

The functor $\Delta : \mathsf{Set} \to \mathsf{Pre}$

Definition ($\Delta : \mathsf{Set} \to \mathsf{Pre}$)

$\Delta : \mathsf{Set} \to \mathsf{Pre} :=$ left adjoint to forgetful $U : \mathsf{Pre} \to \mathsf{Set}$, i.e.

$$\Delta : X \mapsto (X, \mathit{diagonal})$$

Definition

$$\mathsf{LC}_\beta(V) := \textbf{preordered} \text{ set of } \lambda\text{-terms over variables in } V$$

- preorder given by reflexive-transitive closure of β-reduction

λ-calculus as relative monad

Relative monad structure on LC_β

- Relative monad structure: Variables and substitution

$$Var_V : \mathsf{Set}\big(V, \mathsf{LC}(V)\big)$$
$$\sigma_{V,W} : \mathsf{Set}\big(V, \mathsf{LC}(W)\big) \to \mathsf{Pre}\big(\mathsf{LC}_\beta(V), \mathsf{LC}_\beta(W)\big)$$

  - $\sigma \equiv (\gg\!=)$ modulo currying
  - needs proof that $(\gg\!=)$ is compatible with β in 1. argument
- relative monad laws: substitution properties as before

There are also modules over relative monads

and morphisms of such modules describe distributivity of substitution over *App* and *Abs*.

Initiality for syntax with reductions

Summary: we have

- relative monad LC_β over Δ
- rel. module morphisms

$$App_\beta : \mathsf{LC}_\beta \times \mathsf{LC}_\beta \to \mathsf{LC}_\beta$$
$$Abs_\beta : \mathsf{LC}_\beta^* \to \mathsf{LC}_\beta$$

Def.: model of λ-calculus w. β

- relative monad P over Δ
- rel. module morphisms

$$App^P : P \times P \to P$$
$$Abs^P : P^* \to P$$

Theorem

$(LC_\beta, App_\beta, Abs_\beta)$ is the initial object in the category of models.

Refinement: integrate higher-order compatibility

Substitution is compatible with β-reduction

also in the **higher-order** argument:

$$\sigma_{V,W} : \mathsf{Pre}\big(\Delta(V), \mathsf{LC}_\beta(W)\big) \to \mathsf{Pre}\big(\mathsf{LC}_\beta(V), \mathsf{LC}_\beta(W)\big)$$

$$f \rightsquigarrow g \quad \Longrightarrow \quad \sigma(f) \rightsquigarrow \sigma(g)$$

pointwise pointwise

Captured by relative monad towards category **enriched over itself**:

Definition (Relative monad over a functor $F : \mathcal{C} \to \mathcal{D}$)

with $\mathcal{D}$ enriched over itself (e.g., Pre):

$$\sigma_{X,Y} : \mathcal{D}\big(\mathcal{D}(FX, PY), \mathcal{D}(PX, PY)\big)$$

What I have not told

- Signatures
- works with types, too
- allows to define translations with **good properties by construction**
- implemented in the proof assistant Coq

Future work

- Automation in Coq for specifying inequalities
  - $\rightsquigarrow$ automate proofs of compatibility properties
- more fine-grained modeling of reduction
  - $\rightsquigarrow$ by using a better category than Pre
- Non-wellfounded syntax
  - $\rightsquigarrow$ as relative monads from sets to setoids
- Coinductive data types as relative comonads

Future work

- Automation in Coq for specifying inequalities
  - $\rightsquigarrow$ automate proofs of compatibility properties
- more fine-grained modeling of reduction
  - $\rightsquigarrow$ by using a better category than Pre
- Non-wellfounded syntax
  - $\rightsquigarrow$ as relative monads from sets to setoids
- Coinductive data types as relative comonads

Thanks for your attention!