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:

• \( e : G \)
• \( (_-)^{-1} : G \rightarrow G \)
• \( (*) : G \times G \rightarrow G \)
• \( e * g = g \)
• \( g * g^{-1} = e \)
• …

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 inequalities rather than equalities between terms
  \( \Rightarrow \) 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
  - inequalities
  - simple typing
Content of this talk

In this talk:

- **Models** of the untyped \( \lambda \)-calculus with \( \beta \)-reduction
- the \( \lambda \)-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 \( \lambda \)-calculus without reductions
1. Review of Hirschowitz & Maggesi’s work

2. Integrating reduction rules
\textbf{Inductive} \textsc{lc} (\textit{v} : \text{set}) : \text{set} :=
\begin{itemize}
  \item \textit{var} : \textit{v} \to \text{lc} (\textit{v})
  \item \textit{abs} : \text{lc} (\text{option} \textit{v}) \to \text{lc} (\textit{v})
  \item \textit{app} : \text{lc} (\textit{v}) \times \text{lc} (\textit{v}) \to \text{lc} (\textit{v})
\end{itemize}

- \textbf{Monad}: set of terms over free variables

\textit{lc} (\textit{v}) = \text{set of lambda terms over free variables in set} \textit{v}

- \textbf{Monad structure}: “Variables–as–terms” and substitution

\textit{var}_\textit{v} : \textit{v} \to \textit{lc} (\textit{v})

\[(\gg)_{\textit{v}, \textit{w}} : \text{lc} (\textit{v}) \times (\textit{v} \to \text{lc} (\textit{w})) \to \text{lc} (\textit{w})\]

- \textbf{Monad axioms}: properties of variable substitution
Constructors and substitution

Goal: capture the interplay between constructors and substitution

\[
\begin{align*}
\text{App}(M, N) \gg f & = \text{App}(M \gg f, N \gg f) \\
\text{Abs}(M) \gg f & = \text{Abs}(M \gg \text{shift}(f))
\end{align*}
\]

First try: are \textit{Abs} and \textit{App} monad morphisms?

\[
\begin{align*}
\text{Abs} : \text{LC}^* & \to \text{LC} \\
\text{App} : \text{LC} \times \text{LC} & \to \text{LC}
\end{align*}
\]

with \( \text{LC}^* : V \mapsto \text{LC}(\text{option } V) \)

Fails:

- \text{LC}^* a monad, but \textit{Abs} not monad morphism
- \text{LC} \times \text{LC} not a monad in a reasonable sense
Modules over monads generalize monads...

and the functors

\[
\begin{align*}
LC : V &\mapsto LC(V) \\
LC^* : V &\mapsto LC(\text{option } V) \\
LC \times LC : V &\mapsto LC(V) \times LC(V)
\end{align*}
\]

are modules over the monad LC.

and the constructors are module morphisms:

\[
\begin{align*}
Abs : LC^* &\to LC \\
App : LC \times LC &\to LC
\end{align*}
\]

Expresses precisely distributivity of substitution over \( \text{App} \) and \( \text{Abs} \).
Initiability for pure syntax

Summary: we have

- monad LC
- module morphisms
  \[ \text{App} : \text{LC} \times \text{LC} \to \text{LC} \]
  \[ \text{Abs} : \text{LC}^* \to \text{LC} \]

Def.: model of λ-calculus

- monad \( P \)
- module morphisms
  \[ \text{App}^P : P \times P \to P \]
  \[ \text{Abs}^P : P^* \to P \]

Theorem (Hirschowitz & Maggesi)

\((\text{LC}, \text{App}, \text{Abs})\) is the initial object in the category of models.

Now: integrating reduction rules.
① Review of Hirschowitz & Maggesi’s work

② Integrating reduction rules
Goal: define “model of $\lambda$-calculus with $\beta$-reduction”
such that $\lambda$-calculus with

$$\lambda x. M(\mathcal{N}) \rightsquigarrow M[x := \mathcal{N}]$$
is the initial model.

Main question:

How should “$\rightsquigarrow$” be modelled mathematically?

- Terms modulo relations, quotienting
- Monads $\text{Pre} \rightarrow \text{Pre}$
- Relative Monads $\text{Set} \rightarrow \text{Pre}$ with $\text{Pre} :=$ category of preordered sets
Relative monads

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

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

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

+ $F : \mathcal{C} \rightarrow \mathcal{D}$
- $P : \mathcal{C} \rightarrow \mathcal{D}$
- $\eta_X : \mathcal{D}(FX, PX)$
- $\sigma_{X,Y} : \mathcal{D}(FX, PY) \rightarrow \mathcal{D}(PX, PY)$
- relative monad laws
The functor $\Delta : \text{Set} \rightarrow \text{Pre}$

**Definition $(\Delta : \text{Set} \rightarrow \text{Pre})$**

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

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

**Definition**

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

- preorder given by reflexive-transitive closure of $\beta$-reduction
Relative monad structure on $\text{LC}_\beta$

- Relative monad structure: Variables and substitution

$$\text{Var}_V : \text{Set}(V, \text{LC}(V))$$

$$\sigma_{V,W} : \text{Set}(V, \text{LC}(W)) \rightarrow \text{Pre}(\text{LC}_\beta(V), \text{LC}_\beta(W))$$

- $\sigma \equiv (\ggg) \text{ modulo currying}$
- needs proof that $(\ggg)$ is compatible with $\beta$ 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 $\text{App}$ and $\text{Abs}$. 
Summary: we have
- relative monad $LC_\beta$ over $\Delta$
- rel. module morphisms
  $$App_\beta : LC_\beta \times LC_\beta \to LC_\beta$$
  $$Abs_\beta : LC^*_\beta \to LC_\beta$$

Def.: model of $\lambda$-calculus w. $\beta$
- relative monad $P$ over $\Delta$
- 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 $\beta$-reduction

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

$$\sigma_{V,W} : \text{Pre}(\Delta(V), \text{LC}_\beta(W)) \rightarrow \text{Pre}(\text{LC}_\beta(V), \text{LC}_\beta(W))$$

$$f \leadsto g \implies \sigma(f) \leadsto \sigma(g)$$

pointwise  pointwise

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

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

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

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