Algebras of Relative Monads

James Chapman
Institute of Cybernetics, Tallinn

joint work with
Thorsten Altenkirch (University of Nottingham)
and
Tarmo Uustalu (Institute of Cybernetics, Tallinn)

Executive Summary

- Monads are *the* most successful programming pattern arising in functional programming.
- There is a lot of research on different variations of monads (arrows, comonads, idioms).
- Haskell doesn't even give us the full power of monads!
- The extra expressivity of dependently typed functional programming gives us an opportunity to consider other variations that are invisible in functional programming.
- Relative monads are a generalisation of monads. Arrows and monads are instances.

(Our) Motivation

1. If you like functional programming and category theory then you will find monads everywhere.
2. But there are some things which are almost like monads but not quite. The satisfy the rules even but the types are wrong!
3. Abstract the common pattern in these examples.
4. Invent relative monads!
5. Generalise monad theory and study examples in this new light.

What's a monad?

- A monad is just an algebraic structure like a monoid, a group, a ring, etc.

some data:

a map $T : |\mathbf{C}| \to |\mathbf{C}|$ (in Haskell $\mathbf{C} = \mathbf{Set}$, $|\mathbf{C}| = \mathrm{Set}$)

for any X, a map return : $X \to T\,X$

for any X and Y, a map bind : $(X \to T\,Y) \to T\,X \to T\,Y$

subject to the following conditions:

bind return = id -- left unit

bind f . return = f -- right unit

bind f . bind g = bind (bind f . g) -- associativity

Doublenegation monad

False is the empty set

$\neg X = X \to \text{False}$

$T = \neg\neg$

return : $X \to \neg\neg X$ *or* $X \to (X \to \text{False}) \to \text{False}$

bind : $(X \to \neg\neg Y) \to (\neg\neg X \to \neg\neg Y)$

which is the same as

$(X \to \neg Y \to \text{False}) \to$ *hyp. 1*

$(\neg X \to \text{False}) \to$ *hyp. 2*

$\neg Y \to$ *hyp. 3*

$\text{False})$ *by hyp. 2 $ (hyp 1. $ hyp 3)*

and the laws hold trivially (up to definitional equality)

What's an algebra (for a monad)

- A pair of (A,a) where
  - A is a object of **C**
  - For any X a map a : $(X \to A) \to (TX \to A)$
- such that
  - a f . return = f
  - a (a f . g) = a f . bind g

Relationship to to the category **C**

- The algebras are objects of a category called the Eilenberg-Moore category for the monad
- The morphisms are algebra morphisms
- There is an adjunction between this category EM(T) and the category **C**
- Monads can be split into adjunctions, this is one canonical way.
- The other is due to Kleisli.

What is an algebra for double negation?

- It should be a pair of a proposition A and for any X an operation a : $(X \to A) \to \neg\neg X \to A$
- What does this mean?
  - a would be a operation that broadens the implication to take classical evidence instead of constructive evidence (note. for a fixed A).
  - A should be a proposition which is true classically and constructively. Right?

What's a relative monad?

- A relative monad is like a monad but it includes a

data:

a functor $J : \mathbf{J} \to \mathbf{C}$

a map $T : |\mathbf{J}| \to |\mathbf{C}|$

a map for any X, return : $J\ X \to T\ X$

a map for any X and Y, bind : $(J\ X \to T\ Y) \to T\ X \to T\ Y$

subject to the (same!) following conditions:

bind return = id -- left unit

bind f . return = f -- right unit

bind f . bind g = bind (bind f . g) -- associativity

Eg 1- Untyped lambda terms

```
data Lam : Nat → Set where
  var : Fin n → Lam n
  lam : Lam (suc n) → Lam n
  app : Lam n → Lam n → Lam n

T = Lam,
J = Fin,
return : Fin n → Lam n
return = var

bind : (Fin m → Lam n) → Lam m → Lam n
```

The monadic structure shows you how to implement substitution (notoriously fiddly) and what properties to verify

Eg 2 - Vector spaces

```
F : Nat → Set
F n = Fin n → R -- any semiring would do

where T = F and J = Fin and:

return : ∀{n} → Fin n → F n
return a = λ b → if a == b then 1 else 0

bind : ∀{m n} → (Fin m → F n) → F m → F n
bind f v = λ b → Σ m (λ a → v a * f a b)
```

Next... relative algebras

- Why should we care?
  - Algebras for a monad are a standard construction in category theory
  - We've generalised monads. So, our generalisation should work for their algebras too.
  - It would be very nice if our construction gives some interesting algebras for our motivating examples (it does!).

What's an algebra of a relative monad?

- An algebra is a pair (A,a) of
  - an object A of **C**
  - for any X a map a : (JX → A) → T X → A
- subject to the same laws as before:
  - a . return = id
  - a (a f . g) = a f . bind g

Relationship to to the functor J

- The rel. algebras are objects of a category called the rel. Eilenberg-Moore category for the rel. monad
- The morphisms are rel. algebra morphisms
- There is a rel. adjunction between this category REM(T) and the functor J.
- Rel. monads can be split into rel. adjunctions, this is one canonical way.
- The other is Rel. Kleisli.

Eg. 1 - ext. lambda models

- A triple (S,eval,ap) of
  - S : Set
  - for any n, a map eval : (Fin n → S) → Tm n → S
  - a map ap : S → S → S
- subject to some laws:
  - eval γ (var i) = γ i
  - eval γ (app t u) = ap (eval γ t) (eval γ u)
  - ap (eval γ (lam t)) s = eval (γ << s) t
  - ((a : S) → ap f a = ap g a) → f = g

(S,eval) is an algebra!

Eg. 2 - a right module over a semiring R

- a monoid $(A, \varepsilon, \cdot)$ and
- an operation $\& : A \to R \to A$
- subject to some laws
  - $\varepsilon \mathbin{\&} r \cong \varepsilon$
  - $(a \cdot a') \mathbin{\&} r \cong (a \mathbin{\&} r) \cdot (a' \mathbin{\&} r)$
  - $a \mathbin{\&} \mathrm{zero} \cong \varepsilon$
  - $a \mathbin{\&} (r + r') \cong (a \mathbin{\&} r) \cdot (a \mathbin{\&} r')$
  - $a \mathbin{\&} \mathrm{one} \cong a$
  - $a \mathbin{\&} (r * r') \cong (a \mathbin{\&} r) \mathbin{\&} r'$

Output

- A very dense conference paper "Monads need not be endofunctors" at FoSSaCS 2010, Paphos
- A journal paper "Relative monads formalised" in the journal of formalized reasoning (final version pending).
- A journal paper "Relative Monads" for a special issue on FoSSaCS 2010 which is excruciatingly late.

Future work

- Complete formalisation
- Investigate relationship between our work and related ideas
- Find more examples