Replace this file with `prentcsmacro.sty` for your meeting, or with `entcsmacro.sty` for your meeting. Both can be found at the ENTCS Macro Home Page.

Modules over Monads and Linearity

André Hirschowitz [1]

Laboratoire J. A. Dieudonné
Université de Nice
France

Marco Maggesi [2]

Dipartimento U. Dini
Università di Firenze
Italy

Abstract

Inspired by the classical theory of modules over a monoid, we give a first account of the natural notion of module over a monad, which encompasses the notion of algebra. The associated notion of morphism of modules ("linear" natural transformations) captures important formal properties about substitution. In this paper, we present basic constructions of modules and we show examples concerning in particular abstract syntax and lambda-calculus.

Keywords: Modules, Monads, Abstract Syntax, Category Theory.

1 Introduction

Substitution is a major operation. Its relevance to computer sciences has been stressed constantly (see e.g. [5]). Mathematicians of the last century have coined two strongly related notions which capture the formal properties of this operation. The first one is the notion of monad, while the second one is the notion of operad. We focus on the notion of monad. A monad in the category C is a monoid in the category of endofunctors of C (see 2 below) and as such, has left and right modules. These should be endofunctors (in the same category) equipped with an action of the monad. We introduce a slightly more general notion of modules over a monad, based on the elementary observation that we can readily extend the notion of a left action of a monad in C to the case of a functor from any category B to C, and symmetrically the notion of a right action of a monad in C to the case of a functor from C to any category D. We are mostly interested in right modules. As usual, the interest of the

[1] Email: ah@math.unice.fr
[2] Email: maggesi@math.unifi.it

©2006 Published by Elsevier Science B. V.

notion of right module is that it generates a companion notion of morphism. We call *linear* those natural transformations among (right) modules which are compatible with the structure, namely which commute to substitution. The aim of this work is to provide a reference for basic properties of categories of right modules, together with basic examples of morphisms of right modules, hopefully showing the adequacy of the language of right modules for questions concerning in particular syntax and lambda-calculus.

In section 2, we briefly review the theory of monads and their algebras. In section 3, we develop the basic theory of modules. In section 4, we sketch a treatment of syntax (with variable-binding) based on modules. In the remaining sections, we show various linear transformations concerning lists and the lambda-calculus (typed or untyped).

2 Monads and Algebras

We briefly recall some standard material about monads and algebras. Experienced readers may want to skip this section or just use it as reference for our notations. Mac Lane's book [9] can be used as reference on this material.

Let C be a category. A monad over C is a monoid in the category $\mathsf{C} \to \mathsf{C}$ of endofunctors of C. In more concrete terms:

Definition 2.1 [Monad] A monad $R = \langle R, \mu, \eta \rangle$ is given by a functor $R\colon \mathsf{C} \to \mathsf{C}$, and two natural transformations $\mu\colon R^2 \to R$ and $\eta\colon I \to R$ such that the following diagrams commute: [3]

$$\begin{array}{ccc} R^3 & \xrightarrow{R\mu} & R^2 \\ {\scriptstyle \mu R}\downarrow & & \downarrow{\scriptstyle \mu} \\ R^2 & \xrightarrow{\mu} & R \end{array} \qquad \begin{array}{ccccc} I \cdot R & \xrightarrow{\eta R} & R^2 & \xleftarrow{R\eta} & R \cdot I \\ & {\scriptstyle 1_R}\searrow & \downarrow{\scriptstyle \mu} & \swarrow{\scriptstyle 1_R} & \\ & & R & & \end{array}$$

Given a monad R and an arrow $f\colon X \to RY$, we define the function $\mathsf{bind}\, f\colon RX \to RY$ given by $\mathsf{bind}\, f := \mu \cdot Rf$. The functoriality and the composition of the monad can be defined alternatively in terms of the unit and the bind operator. More precisely, we have the equations

$$\mu_X = \mathsf{bind}\, 1_X, \qquad Rf = \mathsf{bind}(\eta \cdot f).$$

Moreover, we have the following associativity and unity equations for bind

$$\mathsf{bind}\, g \cdot \mathsf{bind}\, f = \mathsf{bind}(\mathsf{bind} g \cdot f), \quad \mathsf{bind}\, \eta_X = 1_X, \quad \mathsf{bind}\, f \cdot eta = f \tag{1}$$

for any pair of arrows $f\colon X \to RY$ and $g\colon Y \to RZ$.

In fact, to give a monad is equivalent to give two operators unit and bind as above which satisfy equations 1.

[3] In the programming language Haskell, μ and η correspond to `join` and `return` respectively.

Example 2.2 [Lists] To construct the monad of lists L (over Set), first take the functor $L\colon \mathsf{Set} \to \mathsf{Set}$

$$L\colon X \mapsto \Sigma_{n\in\mathbb{N}} X^n = * + X + X \times X + X \times X \times X + \dots$$

So $L(X)$ is the set of all finite lists with elements in X. Then consider as composition the natural transformation $\mu\colon L \cdot L \to L$ given by the *join* (or *flattening*) of lists of lists:

$$\mu[[a_1, \dots], [b_1, \dots], \dots, [z_1, \dots]] = [a_1, \dots, b_1, \dots, \dots, z_1, \dots].$$

The unit $\eta\colon I \to L$ is constituted by the singleton map $\eta_X\colon x \in X \mapsto [x] \in L(X)$.

Example 2.3 [Lambda Calculus] This example will be worked out with the necessary details in section 5.1, but let us give early some basic ideas (see also [1]). We denote by $FV(M)$ the set of free variables of a λ-term M. Fixed a set X, consider the collection of λ-terms (modulo α-conversion) with free variables in X:

$$\mathsf{LC}(X) := \{M |\, FV(M) \subset X\}.$$

Given a set X we take as unit morphism $\eta_X\colon X \to \mathsf{LC}(X)$ the application assigning to an element $x \in X$ the corresponding variable in $\mathsf{LC}(X)$. Every map $f\colon X \to Y$ induces a morphism $\mathsf{LC}(f)\colon \mathsf{LC}(X) \to \mathsf{LC}(Y)$ ("renaming") which makes LC a functor. The instantiation (or substitution) of free variables gives us a natural transformation

$$\mu_X\colon \mathsf{LC}(\mathsf{LC}(X)) \to \mathsf{LC}(X).$$

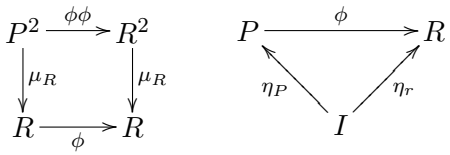
With this structure LC is a monad.

Moreover, by taking the quotient $\Lambda(X)$ of $\mathsf{LC}(X)$ modulo $\beta\eta$-conversion we still obtain a monad (i.e., the composition and the unit of the monad are compatible with $\beta\eta$-conversions).

Definition 2.4 [Maybe monad] The functor $X \mapsto X + *$ which take a set and add one point has a natural structure of monad on Set. Borrowing from the terminology of the library of the programming language Haskell, we call it the `Maybe` monad.

Definition 2.5 [Derivative] We define the *derivative* F' of a functor $F\colon \mathsf{Set} \to \mathsf{Set}$ to be the functor $F' = F \cdot \mathsf{Maybe}$. We can iterate the construction and denote by $F^{(n)}$ the n-th derivative. It is easily checked how the derivative R' of a monad R is again a monad. [4]

Definition 2.6 [Morphisms of monads] A morphism of monads is a natural transformation between two monads $\phi\colon P \to R$ which respects composition and unit:



It can be easily seen that morphisms of monads form a category.

[4] This correspond to the `MaybeT` monad transformer in Haskell.

For our purpose it is relevant to observe that there are a number of natural transformations which arise in the above examples which fail to be morphisms of monads. We take the following as paradigmatic example.

Example 2.7 [Abstraction is not a morphism of monads] Abstraction on λ-terms gives a natural transformation $\mathsf{abs}\colon \mathsf{LC}' \to \mathsf{LC}$ which takes a λ-term $M \in \mathsf{LC}(X+*)$ and binds the "variable" $*$. This fails to be a morphism of monads because it does not respect substitution in the sense of monads: a careful inspection reveals that the transformation

$$\mathsf{LC}(\mathsf{LC}(X+*)+*) \xrightarrow{\mu} \mathsf{LC}(X+*) \xrightarrow{\mathsf{abs}} \mathsf{LC}(X)$$

binds all stars under a single abstraction while

$$\mathsf{LC}(\mathsf{LC}(X+*)+*) \xrightarrow{\mathsf{abs}\,\mathsf{abs}} \mathsf{LC}(\mathsf{LC}(X)) \xrightarrow{\mu} \mathsf{LC}(X)$$

not. In fact, we will see later that LC' is a right module over LC and abs is a LC-*linear* morphism.

Now let R be a monad over C.

Definition 2.8 [Algebra] An *algebra* over R is given by an object A and a morphism $\rho\colon R(A) \to A$ in C such that the following diagrams commute:

$$\begin{array}{ccc} R^2(A) & \xrightarrow{R\rho} & R(A) \\ {\scriptstyle \mu_A}\downarrow & & \downarrow{\scriptstyle \rho} \\ R(A) & \xrightarrow{\rho} & A \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{\eta_A} & R(A) \\ & {\scriptstyle 1_A}\searrow & \downarrow{\scriptstyle \rho} \\ & & A \end{array}$$

Definition 2.9 Let A, B be two algebras over a monad R. An arrow $f\colon A \to B$ in C is said to be a morphism of algebras if it is compatible with the associated actions, i.e., the two induced morphisms from $R(A)$ to B are equal:

$$\rho_B \cdot R(f) = f \cdot \rho_A$$

As we will see later, algebras can be regarded as special kind of left modules.

Example 2.10 [Monoids] Monoids in the category of sets are algebras over the monad L of lists. Given a monoid A, the action $L(A) \to A$ is given by the product of list of elements in A.

3 Modules over monads

Being a monoid in a suitable monoidal category, a monad has associated right and left modules which, a-priori, are objects in the same category, acted upon by the monoid.

From this classical point of view, algebras over a monad, which have been extensively considered, are not (left-)modules. We give a slightly more general definition of modules which is still completely natural. According to this extended definition, algebras turn out to be left-modules.

Despite the natural ideas involved, modules over monads seem to have only sparsely been considered until now (see also the related definition in [6]).

We think that the notions of module and linear transformations deserve more attention. Therefore, we give an account on the theory of modules over a monad with an emphasis on right modules which appear to be more relevant for the application to abstract syntax and semantics that we aim to treat.

3.1 Right modules

We start first by concentrating ourselves on right modules over a given monad R over a category C.

Definition 3.1 [Right modules] A right R-module is given by a functor $M\colon \mathsf{C} \to \mathsf{D}$ equipped with a natural transformation $\rho\colon M \cdot R \to M$, called *action*, which is compatible with the monad composition, more precisely, we require that the following diagrams commute

$$\begin{array}{ccc} M\cdot R^2 & \xrightarrow{M\mu} & M\cdot R \\ {\scriptstyle \rho R}\downarrow & & \downarrow{\scriptstyle \rho} \\ M\cdot R & \xrightarrow{\rho} & M \end{array} \qquad \begin{array}{ccc} M\cdot R & \xleftarrow{M\eta} & M\cdot I \\ {\scriptstyle \rho}\downarrow & \swarrow{\scriptstyle 1_M} & \\ M & & \end{array}$$

We will refer to the category D as the *range* of M.

Given a right R-module M, we can introduce the mbind operator which, to each arrow $f\colon X \to RY$, associates an arrow $\mathsf{mbind}\, f := MX \to MY$ defined by $\mathsf{mbind}\, f := \rho \cdot Rf$. The axioms of right module are equivalent to the following equations over mbind:

$$\mathsf{mbind}\, g \cdot \mathsf{mbind}\, f = \mathsf{mbind}(\mathsf{bind}\, g \cdot f), \qquad \mathsf{mbind}\, \eta_X = 1_X$$

Example 3.2 We can see our monad R as a right module over itself (with range C), which we call the *tautological* module.

Example 3.3 Any constant functor $\underline{W}\colon \mathsf{C} \to \mathsf{D}$, $W \in \mathsf{D}$ is a trivial right R-module.

Example 3.4 For any functor $F\colon \mathsf{D} \to \mathsf{E}$ and any right R-module $M\colon \mathsf{C} \to \mathsf{D}$, the composition $F \cdot M$ is a right R-module (in the evident way).

Definition 3.5 [Derived module] As for functors and monads, derivation is well-behaved also on right modules: for any right R-module M, the derivative $M' = M \cdot \mathsf{Maybe}$ has natural structure of right R-module where the action $M' \cdot P \to M'$ is the composition

$$M \cdot \mathsf{Maybe} \cdot R \xrightarrow{M\gamma} M \cdot R \cdot \mathsf{Maybe} \xrightarrow{\rho\mathsf{Maybe}} M \cdot \mathsf{Maybe}$$

and γ is the natural arrow $\mathsf{Maybe} \cdot R \to R \cdot \mathsf{Maybe}$.

Definition 3.6 [Morphisms of right modules] We say that a natural transformation

of right R-modules $\tau\colon M \to N$ is *linear* if it is compatible with substitution:

$$\begin{array}{ccc} M \cdot R & \xrightarrow{\tau R} & N \cdot R \\ {\scriptstyle \rho_M}\downarrow & & \downarrow{\scriptstyle \rho_N} \\ M & \xrightarrow{\tau} & N \end{array}$$

We take linear natural transformations as right module morphisms.

Example 3.7 We easily check that the natural transformation of a right module into its derivative is linear. Note that there are two natural inclusions of the derivative M' into the second derivative M''. Both are linear.

Example 3.8 Consider again the monad of lists L. The concatenation of two lists is a L-linear morphism $L \times L \to L$.

Definition 3.9 [Category of right modules] We check easily that linear morphisms among right R-modules with the same range yield a subcategory of the functor category. We denote by $\mathrm{Mod}^{\mathsf{D}}(R)$ the category of right R-modules with range D.

Definition 3.10 [Product of right modules] We check easily that the cartesian product of two right R-modules as functors (having as range a cartesian category D) is naturally a right R-module again and is the cartesian product also in the category of $\mathrm{Mod}^{\mathsf{D}}(R)$. We also have finite products as usual. The final right module $*$ is the product of the empty family.

Example 3.11 Given a right R-module M, we have a natural "evaluation" morphism $\mathsf{eval}\colon M' \times R \to M$.

Proposition 3.12 *Derivation yields a cartesian functor from right R-modules to right R-modules.*

3.2 Left modules

Let R be a monad over a category C. The definition of left module is similar to that of right module.

Definition 3.13 [Left modules] A left R-module is given by a functor $M\colon \mathsf{D} \to \mathsf{C}$ equipped with a natural transformation $\rho\colon R \cdot M \to M$ which makes the following diagrams commutative

$$\begin{array}{ccc} R^2 \cdot M & \xrightarrow{\mu M} & R \cdot M \\ {\scriptstyle R\rho}\downarrow & & \downarrow{\scriptstyle \rho} \\ R \cdot M & \xrightarrow{\rho} & M \end{array} \qquad \begin{array}{ccc} I \cdot M & \xrightarrow{\eta M} & R \cdot M \\ & {\scriptstyle 1_M}\searrow & \downarrow{\scriptstyle \rho} \\ & & M \end{array}$$

As for right modules, we will call *range* of M the category D.

Example 3.14 Our monad R is a left module over itself.

Example 3.15 If A is an R algebra, then for any category D the constant functor $\underline{A}\colon X \mapsto A$ has structure of left R-module. In particular, we can identify our algebra

with the corresponding functor $\underline{A}\colon * \to \mathsf{C}$, where $*$ is the category with one object and one arrow.

Example 3.16 Let $\phi\colon R \to P$ be a morphism of monads. Then P is a left and right R-module with actions given respectively by $P \cdot R \xrightarrow{\phi R} R \cdot R \xrightarrow{\mu_R} R$ and $R \cdot P \xrightarrow{P\phi} R \cdot R \xrightarrow{\mu_R} R$.

Definition 3.17 [Morphisms of left modules] A morphism of left R-modules is natural transformation $\tau\colon M \to N$ which is compatible with substitution:

$$\begin{array}{ccc} R \cdot M & \xrightarrow{R\tau} & R \cdot N \\ {\scriptstyle \rho_M}\downarrow & & \downarrow{\scriptstyle \rho_N} \\ M & \xrightarrow{\tau} & N \end{array}$$

Definition 3.18 [Category of left R-modules] We check easily that module morphisms among left R-modules with the same range yield a subcategory of the functor category. We denote by $\mathrm{Mod}_{\mathsf{D}}(R)$ the category of left R-modules with range D.

3.3 Limits and colimits of right modules

Limits and colimits in the category of right modules can be constructed point-wise. For instance:

Lemma 3.19 (Limits and colimits of right modules) *If* D *is complete (resp. cocomplete), then* $\mathrm{Mod}^{\mathsf{D}}(R)$ *is complete (resp. cocomplete).*

Proof. Suppose first that D be a complete category and $G\colon \mathsf{I} \to \mathrm{Mod}^{\mathsf{D}}(R)$ be a diagram of right modules over the index category I. For any object $X \in \mathsf{C}$ we have a diagram $G(X)\colon \mathsf{I} \to \mathsf{D}$ and for any arrow $f\colon X \to Y$ in C we have a morphism of diagrams $G(X) \to G(Y)$. So define

$$U(X) := \varprojlim G(X)$$

Next, given an arrow $f\colon X \to R(Y)$, we have an induced morphism of diagrams $G(X) \to G(Y)$ by the module structure on each object of the diagram. This induces a morphism $\mathsf{mbind}\, f\colon U(X) \to U(Y)$. It is not hard to prove that mbind satisfies the module axioms and that U it is the limit of G. The colimit construction is carried analogously. □

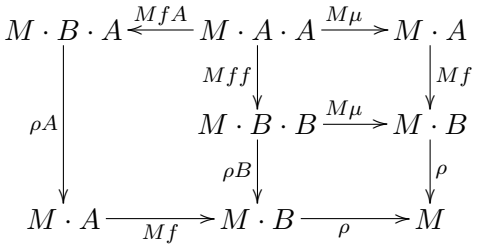
3.4 Pull-backs

Definition 3.20 [Pull-back] Given a morphism $f\colon A \to B$ of monads and a right B-module M, we define its *pull-back* f^*M as follows: we set $f^*M(X) := M(X)$ and we define the action $\rho_{f^*M}\colon f^*M \cdot A \to f^*M$ as the composition of $Mf\colon M \cdot A \to M \cdot B$ with the action of B on M:

$$\rho_{f^*M} := \rho_M \cdot Mf.$$

Lemma 3.21 *The pull-back of a right module is a right module.*

Proof. Our thesis is the commutativity of the diagram



which follows from the commutativity of the three pieces since M is a right B-module, the map from $M(B(-)) \to M(-)$ is functorial, and f is a morphism. □

Definition 3.22 [Pull-back (functoriality)] We upgrade the pull-back into a functor $f^*\colon \mathrm{Mod}^{\mathsf{D}}(B) \to \mathrm{Mod}^{\mathsf{D}}(A)$ by checking that if $g\colon M \to N$ is a morphism of right B-modules, then so is the natural transformation $f^*g\colon f^*M \to f^*N$.

Proposition 3.23 *Pull-back commutes with products and with derivation.*

Proposition 3.24 *Any morphism of monads $f\colon A \to B$ yields a morphism of right A-modules, still denoted f, from A to f^*B.*

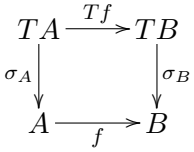
4 Initial Algebra Semantics

To ease the reading of the forthcoming sections, we collect in this section some classical ideas about Initial Algebra Semantics.

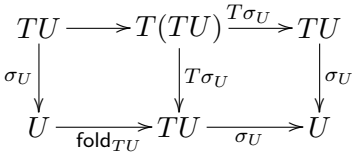
Given a category C and an endofunctor $T\colon \mathsf{C} \to \mathsf{C}$, a T-algebra[5] is given by an object $A \in \mathsf{C}$ and an arrow

$$\sigma_A\colon TA \to A.$$

A morphism of T-algebras is an arrow $f : A \to B$ which commutes with the *structural morphism* σ



This defines the category of T-algebras. Notice that, for any T-algebra A, there is an induced T-algebra structure on TA given by $T\sigma_A\colon T(TA) \to TA$. An initial T-algebra is called a *(least) fix-point* of T. Given one such fix-point U and any other T-algebra A we denote by $\mathsf{fold}_A\colon U \to A$ the induced initial morphism. From the universal property of U and the following diagram



[5] There is a lexical conflict here with algebra of monads introduced in section 2, which is deeply rooted in the literature anyway. We hope that this will not lead to any confusion.

it follows that the σ_U is an isomorphism whose inverse is fold_{TU}.

Let us show how this general framework can put to work in the case of (polymorphic) lists.

Example 4.1 Take $\mathsf{C} = \mathsf{Set} \to \mathsf{Set}$ the category of endofunctors of small sets and consider $T\colon (\mathsf{Set} \to \mathsf{Set}) \to (\mathsf{Set} \to \mathsf{Set})$ the functor defined by

$$T\colon F := X \mapsto * + X \times FX.$$

The least fix point of T is the monad of lists L introduced in section 2. The T-algebra structure $* + X \times LX = TL \simeq L$ gives the constructors (nil, cons) and the corresponding destructor. We would like to recognise this structural isomorphism as a L-linear morphism. Unfortunately, this cannot happens at this stage since TL is not a right L-module (notice that the identity functor $I = X \mapsto X$ is not a morphism of right L-modules). We will explain in section 6 how this phenomenon can be considered a consequence of the lack of typing.

5 Monads over sets

In this section we consider more examples of linear morphisms over monads on the category of sets.

5.1 Untyped Syntactic Lambda Calculus

Consider the functor $T\colon (\mathsf{Set} \to \mathsf{Set}) \to (\mathsf{Set} \to \mathsf{Set})$ given by

$$TF\colon X \mapsto X + FX \times FX + F'X$$

where F' denotes the derived functor $X \mapsto F(X + *)$. It can be shown that T possesses a least fix point that we denote by LC (for λ-calculus, cfr. the example in section 2). We regard $\mathsf{LC}(X)$ the set of λ-terms with free variables taken from X (see also [2]). In fact, the structural morphism $T\mathsf{LC} \to \mathsf{LC}$ gives the familiar constructors for λ-calculus in the *locally nameless* encoding, namely, the natural transformations

$$\begin{aligned}
&\mathsf{var}\colon I \to \mathsf{LC},\\
&\mathsf{app}\colon \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC},\\
&\mathsf{abs}\colon \mathsf{LC}' \to \mathsf{LC}.
\end{aligned}$$

As already observed, the substitution (instantiation) of free variables gives a monad structure on LC where var is the unit.

We would like to express that constructors are well behaved with respect to substitution. Again, as in the case of lists, $T\mathsf{LC}$ has no natural structure of right LC-module. However, we can consider the functor T as built of two parts $TF = I + T_0F$ where $T_0F := F \times F + F'$ (in other words we are tackling apart var, the unit of the monad, from the other two constructors app and abs). Now $T_0\mathsf{LC}$ is a right LC-module and we can observe that the algorithm of substitution is carried precisely in such a way that the induced morphism

$$\mathsf{app}, \mathsf{abs}\colon T_0\mathsf{LC} \to \mathsf{LC}$$

is LC-linear or, which is the same, the natural transformations $\mathsf{app}\colon \mathsf{LC}\times\mathsf{LC}\to\mathsf{LC}$ and $\mathsf{abs}\colon \mathsf{LC}'\to\mathsf{LC}$ are LC-linear. To make the idea more clear, we report this short code in the *Haskell* programming language which implements the algorithm of substitution.

```
module LC where
import Monad (liftM)

data LC a = Var a | App (LC a) (LC a) | Abs (LC (Maybe a))

instance Monad LC where
    return = Var
    Var x >>= f = f x
    App x y >>= f = App (x >>= f) (y >>= f)
    Abs x >>= f = Abs (x `mbind` f)

mbind :: LC (Maybe a) -> (a -> LC b) -> LC (Maybe b)
mbind x f = x >>= maybe (Var Nothing) (liftM Just . f)
```

In the above code, `mbind` constitute the right LC-module structure on LC'. It is now evident that the recursive call in the definition of `(>>=)` are exactly those given by the linearity of app and abs.

We can go further and try to make the linearity more explicit in the syntactic definition of λ-calculus. This can be done as follows.

Theorem 5.1 *Consider the category* Mon^{T_0} *of monads* R *over sets endowed with a* R*-linear morphism* $T_0R\to R$ *with arrows given by commutative diagrams*

$$\begin{array}{ccc} T_0R & \xrightarrow{T_0f} & f^*T_0P \\ \downarrow & & \downarrow \\ R & \xrightarrow{f} & f^*P \end{array}$$

where all morphism are R*-linear (we are using implicitly the fact that pull-back commutes with derivation and products). The monad* LC *is initial in* Mon^{T_0}*.*

In fact, the previous theorem can be generalized as follows (interested readers may also want to look at other works on higher order abstract syntax, e.g., [4], [8], [7]). Let R be a monad. We define an *arity* to be a list of non negative integers. We denote by $\mathbb{N}^*$ the set of arities. To an arity $(a_1,\dots,a_r)$ we associate the endofunctor T^a over left R-modules given by

$$T^aM = M^{(a_1)}\times\cdots\times M^{(a_r)}.$$

and we say that a linear morphism $T^aM\to M$ is a *R-representation* in M of a. For instance, the app and abs constructors are LC-representations of the arities $(0,0)$ and (1) respectively.

Next, we consider *signatures* which are family of arities. To each signature

$\Sigma\colon I \to \mathbb{N}^*$ we associate the endofunctor T^Σ over left R-modules given by

$$T^\Sigma = \sum_{i\in I} T^{\Sigma_i}$$

and we say that a linear morphism $T^\Sigma M \to M$ is a *R-representation* in M of Σ. The app and abs altogether gives a LC-representation of the signature $((0,0),(1))$.

As in the special case of λ-calculus, representations of a given signature Σ form a category.

Theorem 5.2 *For any signature Σ, the category of Σ-representations has an initial object.*

5.2 Untyped Semantic Lambda Calculus

For any set X, consider the equivalence relation $\equiv_{\beta\eta}$ on $\mathsf{LC}(X)$ given by the reflexive symmetric transitive closure of β and η conversions and define $\Lambda(X) := \mathsf{LC}(X)/\equiv_{\beta\eta}$. It can be shown that $\equiv_{\beta\eta}$ is compatible with the structure of LC so Λ has a structure of monad, the projection $\mathsf{LC} \to \Lambda$ is a morphism of monad and we have an induced morphism $T_0\Lambda \to \Lambda$ which is Λ-linear.

Now the key fact is that the abstraction $\mathsf{abs}\colon \Lambda' \to \Lambda$ is in fact a linear isomorphism! In fact, it is easy to construct its inverse $\mathsf{app}_1\colon \Lambda \to \Lambda'$:

$$\mathsf{app}_1\, x = \mathsf{app}(\hat{x}, *)$$

where $x \mapsto \hat{x}$ denotes the natural inclusion $\Lambda \to \Lambda'$. The two equations

$$\mathsf{app}_1 \cdot \mathsf{abs} = 1_{\Lambda'}, \qquad \mathsf{abs} \cdot \mathsf{app}_1 = 1_\Lambda$$

can be considered the ultimate formulations of the β and η rules.

Again, we can present this situation under a more syntactical point of view. For this, consider the category of *exponential monads* given by monad R endowed with a R-linear isomorphism with its derivative. Arrow in the category of exponential monads are given by commutative diagrams

$$\begin{array}{ccc} R' & \xrightarrow{T_0 f} & f^*P' \\ \downarrow & & \downarrow \\ R & \xrightarrow{f} & f^*P \end{array}$$

where all arrows are R-linear (we are implicitly using the commutativity of pull-back with derivation).

Theorem 5.3 *The monad Λ is initial in the category of exponential monads.*

We have developed a formal proof of the above theorem in the Coq proof assistant [3]. Our code can be found at `http://www.math.unifi.it/~maggesi`

6 Monads over types

So far we mostly considered examples of monads and modules on the category $\mathsf{C} = \mathsf{Set}$ of small sets. Other interesting phenomena can be captured by taking in to account monads and modules on different categories. In this section we consider the case $\mathsf{C} = \mathsf{Set}_\tau$ the category of sets fibered over a fixed set τ. This is the category given by maps $\phi_X \colon X \to \tau$, called τ-sets, where arrows $\langle X, \phi_X\rangle \to \langle Y, \phi_Y\rangle$ are given by maps $f \colon X \to Y$ which commutes with the structural morphisms, i.e., $\phi_Y \cdot f = \phi_X$. For each $t \in \tau$ and each τ-set X, we denote by $X_\tau := \phi_X^{-1}(t)$ the preimage of t in t. We regard τ as a "set of types" and the fibers X_t as a set of "terms of type t".

6.1 Typed lists

We revisit one more time the example of the functor of lists under a different perspective. In particular, we would like to find a suitable structure which enables us to reveal the linearity (i.e., compatibility with respect to substitution) of the constructors nil and cons.

We first introduce the general construction. Take $\tau = \mathbb{N}$ and consider an endofunctor F on the category of sets. We define an associated endofunctor F_∞ on the category of $\mathbb{N}$-sets as follows. First consider the functor F_n from $\mathbb{N}$-sets to sets given by

$$F_n(X) := \sum_{i+j=n} F^i(X_j)$$

and define $R_\infty := \sum_n R_n$. Obviously, every $\mathbb{N}$-set X can be regarded as $\mathbb{N}$-subset of $F_\infty(X)$ since $F_0(X) = X$, so we have a natural transformation $\eta \colon I \to F_\infty$. Given a $\mathbb{N}$-map $f \colon X \to F_\infty(Y)$ we have an induced $\mathbb{N}$-map $\mathsf{bind}\, f \colon F_\infty(X) \to F_\infty(Y)$. It is not hard to verify that this gives a structure of monad on F_∞. Notice that in this case η is a F_∞-linear morphism.

Given a $\mathbb{N}$-set X we have the associated *shifts* $X[n]$ which are obtained by adding n to the structural map $X \to \mathbb{N}$. The shift $(\cdot)[n] \colon X \mapsto X[n]$ gives an endofunctor over $\mathbb{N}$-sets. Given a functor F from any category to the category of $\mathbb{N}$-sets, we consider the *shifted functors* $F[n]$ obtained as composition of F followed by $(\cdot)[n]$. From the remarks of section 3, it follows at once that the shifts $F_\infty[n]$ are right F_∞-modules. We can recover F from F_∞ by restricting to the subcategory $\mathsf{Set}_0 \simeq \mathsf{Set}$ given by $\mathbb{N}$-sets concentrated on 0.

Now we apply the above construction on the monad L of lists (notice however that we are interested here only in the functorial properties of L). We can regard $\tau = \mathbb{N}$ as the inductive set of types generated by the grammar $\tau = * \mid \mathsf{list}\, \tau$. The nil and cons constructors constitute a family of morphism parametrized by $t \in \tau$

$$\mathsf{nil}_t \colon * \longrightarrow L_{\mathsf{list}\, t} \qquad \mathsf{cons}_t \colon L_t \times L_{\mathsf{list}\, t} \longrightarrow L(\mathsf{list}\, t).$$

which provide us two L_∞-linear morphisms

$$\mathsf{nil}_\infty \colon *[1] \longrightarrow L_\infty[1], \qquad \mathsf{cons}_\infty \colon I \times L_\infty[1] \longrightarrow L_\infty[1]$$

where $*$ and I denote respectively the final and identity functor in the category of

endofunctors of $\mathbb{N}$-sets. Altogether, nil_∞ and cons_∞ give us a F_∞-linear morphism $TL \to L_\infty[1]$ where $TF = *[1] + I \times F_\infty[1]$.

6.2 Typed Lambda Calculus

Our second example of typed monad is simply-typed λ-calculus. We denote by τ the set of simple types $\tau := * \mid \tau \to \tau$. Following [10], we consider the syntactic typed λ-calculus as an assignment $V \mapsto \mathsf{LC}_\tau(V)$, where $V = \sum_{t\in\tau}(V_t)$ is a (variable) set (of typed variables) while

$$\mathsf{LC}_\tau(V) = \sum_{t\in\tau}(\mathsf{LC}_\tau(V))_t$$

is the set of typed λ-terms (modulo α-conversion) built on free variables taken in V. As in the untyped case, we can consider the functor Λ_τ obtained by quotient modulo $\beta\eta$ conversion. We observe that both assignments are monads over the category of τ-sets and that the natural transformation $\mathsf{LC}_\tau \to \Lambda_\tau$ is a morphism of monads.

Given a type t we set $\mathsf{LC}_t(X) := (\mathsf{LC}_\tau(X))_t$ which gives a functor over τ-sets, which is equipped with substitution, turning it into a right module over LC_τ. And given two types s, t, we have

$$\mathsf{app}_{s,t}\colon \Lambda_{s\to t} \times \mathsf{LC}_s \longrightarrow \mathsf{LC}_t$$

which is linear.

For the abs construction, we need a notion of partial derivative for a module. For a right module M over τ-sets, and a type $t \in \tau$, we set

$$\delta_t M(V) := M(V + *_t)$$

where $V + *_t$ is obtained from V by adding one element with type t. It is easily checked how $\delta_t M$ is again a right module. Now, given two types s and t, it turns out that

$$\mathsf{abs}_{s,t}\colon \delta_s \mathsf{LC}_t \longrightarrow \mathsf{LC}_{s\to t}$$

is linear.

Taking app and abs together we obtain a LC_τ-linear morphism $T\mathsf{LC}_\tau \to \mathsf{LC}_\tau$ where $TF := F \times F + \delta F$ and $\delta F := \sum_{t\in\tau} \delta_t F$. We can define in the natural way the category of these T algebras and observe that LC_τ is initial there. The same constructions applies to the monad Λ_τ and the morphism $\mathsf{LC}_\tau \to \Lambda_\tau$ is the initial morphism.

7 Monads over preordered sets

Our last example is about monads and modules over the theory of partially ordered sets (sets with a reflexive and transitive binary relation). Preordering is used here to model the relation $\xrightarrow{\beta\eta}_*$ generated by the reflexive and transitive closure of the β and η conversions. In fact, the construction given in this section can be considered a refinement of those of section 5.1 where we used the reflexive, symmetric and transitive closure $\equiv_{\beta\eta}$.

Let us consider again the monad LC of λ-terms. Given a preordered set X, we consider the preordering on $\mathsf{LC}(X)$ given by the rules

$$\begin{aligned} x \leq y &\implies \mathsf{var}\, x \leq \mathsf{var}\, y, \\ S \leq S' \wedge T \leq T' &\implies \mathsf{app}(S,T) \leq \mathsf{app}(S',T'), \\ T \leq T' &\implies \mathsf{abs}(T) \leq \mathsf{abs}(T), \\ T \longrightarrow_{\beta\eta} T' &\implies T \leq T'. \end{aligned}$$

It is not hard to verify that with this new structure LC is now a monad over preordered sets. It turns out that the app and abs constructions are still LC-linear with respect to this richer structure.

References

[1] Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In *CSL*, pages 453–468, 1999.

[2] Richard S. Bird and Ross Paterson. De Bruijn notation as a nested datatype. *Journal of Functional Programming*, 9(1):77–91, 1999.

[3] The Coq Proof Assistant. http://coq.inria.fr.

[4] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *14th Symposium on Logic in Computer Science (Trento, 1999)*, pages 193–202. IEEE Computer Soc., Los Alamitos, CA, 1999.

[5] Marcelo P. Fiore. On the structure of substitution. Invited address for the 22nd Mathematical Foundations of Programming Semantics Conf. (MFPS XXII), 2006. DISI, University of Genova (Italy).

[6] Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *Logic in Computer Science*, pages 93–104, 2001.

[7] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *14th Symposium on Logic in Computer Science (Trento, 1999)*, pages 214–224. IEEE Computer Soc., Los Alamitos, CA, 1999.

[8] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Symposium on Logic in Computer Science (Trento, 1999)*, pages 204–213. IEEE Computer Soc., Los Alamitos, CA, 1999.

[9] Saunders Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1998.

[10] Julianna Zsidó. Le lambda calcul vu comme monade initiale. Master's thesis, Université de Nice – Laboratoire J. A. Dieudonné, 2005/06. Mémoire de Recherche – master 2.