Modules over Monads and their Algebras

Maciej Piróg[1], Nicolas Wu[2], and Jeremy Gibbons[1]

1 **Department of Computer Science, University of Oxford**
Wolfson Building, Parks Rd, Oxford OX1 3QD, UK
`{firstname.lastname}@cs.ox.ac.uk`

2 **Department of Computer Science, University of Bristol**
Merchant Venturers Building, Woodland Rd, Bristol BS8 1UB, UK
`nicolas.wu@bristol.ac.uk`

Abstract

Modules over monads (or: actions of monads on endofunctors) are structures in which a monad interacts with an endofunctor, composed either on the left or on the right. Although usually not explicitly identified as such, modules appear in many contexts in programming and semantics. In this paper, we investigate the elementary theory of modules. In particular, we identify the monad freely generated by a right module as a generalisation of Moggi's resumption monad and characterise its algebras, extending previous results by Hyland, Plotkin and Power, and by Filinski and Støvring. Moreover, we discuss a connection between modules and algebraic effects: left modules have a similar feeling to Eilenberg–Moore algebras, and can be seen as handlers that are natural in the variables, while right modules can be seen as functions that run effectful computations in an appropriate context (such as an initial state for a stateful computation).

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.3.3 Studies of Program Constructs

Keywords and phrases monad, module over monad, algebraic data types, resumptions, free object

Digital Object Identifier 10.4230/LIPIcs.CALCO.2015.287

1 Introduction

Given a monad M, a *right module* over M (or: an M-module) is an endofunctor S together with a natural transformation (called an *action*)

$$\overrightarrow{\mu} : SM \to S$$

coherent with the monadic structure of M. Dually, a *left module* over M is an endofunctor L together with a natural transformation

$$\overleftarrow{\mu} : ML \to L$$

also coherent with the monadic structure of M.

Modules over monads are special cases of modules over monoids in monoidal categories (as monads are monoids in categories of endofunctors). They are discussed, for example, by Dubuc [10] and Mac Lane [23, Sec. VI.4]), as well as, in a more general setting, by Street [32]. In this paper, by developing some elementary theory of modules, we show their connections to some constructions in semantics of programming languages and the theory of algebraic data structures.

As our primary result, we describe the monad freely generated by a right M-module S. The functor part of this monad is given by the composition MS^*, where S^* is the free

6th International Conference on Algebra and Coalgebra in Computer Science (CALCO'15).
Editors: Larry Moss and Paweł Sobociński; pp. 287–300
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

monad generated by S as an endofunctor. We also introduce the notion of *algebra for a module*, which is a coherent pair consisting of an S-algebra (for S as an endofunctor) and an Eilenberg–Moore M-algebra. We observe that algebras for a right M-module S coincide with Eilenberg–Moore algebras for the monad MS^*.

These considerations have some practical aspects as well. The monad MS^* is a generalisation of Moggi's [27] resumption monad $M(GM)^*$ for an endofunctor G, which has applications in semantics and functional programming. The universal property of MS^* subsumes Hyland, Plotkin, and Power's [20] result that $M(GM)^*$ is the coproduct of M and G^* in the category of monads, or Filinski and Støvring's [13] construction of data types that interleave data and monadic effects. Generalising the above constructions to the setting of modules gives us new, interesting instances.

In passing, we investigate more of the theory of modules. We give examples and general constructions, which suggest the ubiquity of modules. For instance, every (left or right) adjoint comonad is a module over its adjoint monad, and every endofunctor is a module over its codensity monad. We show that a large portion of the theory of monads can be transported to the theory of modules. For example, the connection between monads and adjunctions is lifted to the connection between modules and adjunctions paired with a functor, while the correspondence between distributive laws and liftings is extended to the correspondence between their obvious counterparts.

2 Modules over monads

2.1 Preliminaries

We work in a base category $\mathscr{C}$, which is locally small and complete. We reserve A, B, C, X, Y, Z to denote objects in categories, while f, g, h denote morphisms and natural transformations. Functors are usually denoted as F, U, G, H, M, T. We reserve M, T for monads, and F, U for left and right adjoints respectively. To avoid confusion, we sometimes add superscripts. Given functors G, G', H, H' and two natural transformations $g : G \to G'$ and $h : H \to H'$, we denote the composition of endofunctors by juxtaposition (for example, GH). The parallel composition of g and h is also denoted by juxtaposition, as in $gh : GH \to G'H'$.

For an endofunctor G, we write $\mathbf{Alg}(G)$ for the category of G-algebras, $\mathbf{Mnd}$ for the category of monads and monad morphisms over a base category, and $\mathbf{EM}(M)$ for the category of Eilenberg–Moore algebras for a monad M. We always denote the unit and the multiplication of a monad as η and μ respectively. If there is more than one monad in context, we add superscripts. We follow the standard abuse of notation and denote a monad by its underlying endofunctor.

Given an endofunctor $G : \mathscr{C} \to \mathscr{C}$, we denote the free monad generated by G (if it exists) as G^*, and the canonical injection by $\mathsf{emb} : G \to G^*$. For a monad T and a natural transformation $g : G \to T$, we denote by $[\![g]\!] : G^* \to T$ the monad morphism given by the freeness of G^*, that is, the unique monad morphism $[\![g]\!]$ with the property that $g = [\![g]\!] \cdot \mathsf{emb}$. Note that if the base category has binary coproducts, the functor part of G^* is given by $G^*A = \mu X.GX + A$, where $\mu X.HX$ denotes the carrier of the initial H-algebra (see Kelly [21]). In such a case, the free monad arises from the adjunction between $F^{\mathrm{Alg}} : \mathscr{C} \to \mathbf{Alg}(G)$ and $U^{\mathrm{Alg}} : \mathbf{Alg}(G) \to \mathscr{C}$, which we write as $F^{\mathrm{Alg}} \dashv U^{\mathrm{Alg}} : \mathscr{C} \rightharpoonup \mathbf{Alg}(G)$. This adjunction is strictly monadic, which means that the canonical comparison functor $K : \mathbf{Alg}(G) \to \mathbf{EM}(G^*)$ is an isomorphism.

2.2 Modules defined

▶ **Definition 1.** Let M be a monad. An endofunctor S together with a natural transformation (an *action*) $\overrightarrow{\mu} : SM \to S$ is called a *right M-module* (or: a *right module over M*) if the following diagrams commute:

$$
\begin{tikzcd}
SMM \arrow[r, "S\mu"] \arrow[d, "\overrightarrow{\mu}M"] & SM \arrow[d, "\overrightarrow{\mu}"] \\
SM \arrow[r, "\overrightarrow{\mu}"] & S
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "S\eta"] \arrow[dr, "\mathsf{id}"] & SM \arrow[d, "\overrightarrow{\mu}"] \\
& S
\end{tikzcd}
$$

We define a morphism between an M-module S and a M'-module S' as a pair $\langle m, s \rangle$, where $m : M \to M'$ is a monad morphism and $s : S \to S'$ is a natural transformation such that the following diagram commutes:

$$
\begin{tikzcd}
SM \arrow[r, "\overrightarrow{\mu}^S"] \arrow[d, "sm"] & S \arrow[d, "s"] \\
S'M' \arrow[r, "\overrightarrow{\mu}^{S'}"] & S'
\end{tikzcd}
$$

We refer to the category of all right modules over monads as **Mod**. Its objects are pairs $\langle M, S \rangle$, where M is a monad and S is an M-module. Arrows are given by morphisms between modules.

Similarly, we define a *left M-module* as an endofunctor L together with $\overleftarrow{\mu} : ML \to L$ such that the obvious analogues of the diagrams above commute.

▶ **Example 2.** The following are examples of general constructions of right modules (most of them dualise easily to the case of left modules):

1. Let M be a monad. Then, M is itself an M-module with the action given by the multiplication $\mu^M : MM \to M$.
2. Let $m : M \to T$ be a monad morphism. Then, T is an M-module with the action given by $\mu^T \cdot Tm : TM \to T$.
3. Let S be an M-module and G be an endofunctor. Then, GS is also an M-module with the action given by $G\overrightarrow{\mu} : GSM \to GS$. An important instance of this construction is when the original module is the monad itself, that is, when the module is given by GM.
4. With the definitions as above, let $\lambda : TM \to MT$ be a distributive law between monads. Then, the composition ST is a module of the induced monad MT. The action is given by $(\overrightarrow{\mu}\mu^T) \cdot S\lambda T : STMT \to MT$.
5. If S and Q are two M-modules, their coproduct $S + Q$ is also an M-module with the action defined componentwise.
6. Let F be an endofunctor with a right adjoint U. Then, F is an UF-module with the action given by $\varepsilon F : FUF \to F$, where ε is the counit of the adjunction.
7. Let M be a monad with a left adjoint W. In such a case, W is a comonad (the situation is dual to the one observed by Eilenberg and Moore [12], in which M has a right adjoint). Also, W is an M-module with the action given by $cW \cdot W\mu W \cdot WMu : WM \to W$, where $u : \mathsf{Id} \to MW$ is the unit and $c : WM \to \mathsf{Id}$ is the counit of the adjunction.

▶ **Example 3.** The last two constructions from Example 2 can be illustrated with the 'currying' adjunction native to cartesian closed categories, $A \times \text{-} \dashv (\text{-})^A$, for a fixed object A.

As for Example 2 (6), this adjunction gives rise to the state monad $(A \times \text{-})^A$, which can be seen as a model of stateful computations. The action of the module given by the left adjoint is equal to $\overrightarrow{\mu}_X = \mathsf{eval}^A_{A\times X} : A \times (A \times X)^A \to A \times X$, where $\mathsf{eval}^A_B : A \times B^A \to B$ is the evaluation morphism for exponentials. Alternatively, using the fact that simply-typed λ-calculus is the internal language of CCCs, one could say $\overrightarrow{\mu}_X = \lambda\langle a, f\rangle{:}A \times (A \times X)^A.\, f\, a$. Intuitively, $\overrightarrow{\mu}$ takes as its arguments an initial state and a stateful computation, and returns the final state paired with the final answer. In other words, it is a morphism that 'executes' the stateful computation.

Example 2 (7) comes from the fact that $(\text{-})^A$ is a monad, known in the functional programming community as the reader monad. Its multiplication $(X^A)^A \to X^A$ is given by the diagonal $\lambda f{:}(X^A)^A.\, \lambda a{:}A.\, f\, a\, a$. Its adjoint comonad (known as the environment comonad) $A \times \text{-}$ is a $(\text{-})^A$-module. The action of this module $\overrightarrow{\mu}_X : A \times X^A \to A \times X$ is given as $\lambda\langle a, f\rangle{:}A \times X^A.\, \langle a,\ f\, a\rangle$.

▶ **Example 4.** For all $n \geq 1$, the **Set** functor of lists with at least n elements is a module of the non-empty list monad (the free semigroup monad).

▶ **Example 5.** An Eilenberg–Moore algebra $\langle A,\ f : MA \to A\rangle$ can be understood as a left M-module given by the constant endofunctor C_A and a natural transformation $f : MC_A \to C_A$. Indeed, in the literature, Eilenberg–Moore algebras are sometimes called 'modules'.

We can consider **Cat** (the category of all categories up to a certain size) as a 2-category: 0-cells are categories, 1-cells are functors, and 2-cells are natural transformations. We consider different opposites of **Cat**: the op-dual $\mathbf{Cat}^{\mathsf{op}}$ obtained by reversing 1-cells, the co-dual $\mathbf{Cat}^{\mathsf{co}}$ obtained by reversing 2-cells, and the bidual $\mathbf{Cat}^{\mathsf{coop}}$ obtained by reversing both. For example, monads and comonads are mutually co-dual concepts (that is, a monad in $\mathbf{Cat}^{\mathsf{co}}$ is a comonad in **Cat**), while both are self-op-dual (that is, a monad is an opmonad, while a comonad is a co-opmonad). Left and right modules are mutually op-dual concepts, that is, a left module is a right opmodule, while a right module is a left opmodule. In the obvious way, co-duality gives us the concepts of left and right comodules over comonads.

▶ **Example 6.** Given an endofunctor $G : \mathscr{C} \to \mathscr{C}$, its codensity monad is given by the right Kan extension of G along itself:

$$
\begin{tikzcd}
\mathscr{C} \arrow[r, "G"] \arrow[d, "G"'] \arrow[dr, phantom, "\Uparrow\kappa"] & \mathscr{C} \\
\mathscr{C} \arrow[ur, "\mathsf{Ran}_G G"'] &
\end{tikzcd}
$$

In this case, G is a left module of $\mathsf{Ran}_G G$ with the natural transformation $\kappa : (\mathsf{Ran}_G G)G \to G$ being the module action. Intuitively, we can see the codensity monad as a generalised type of computations in continuation-passing style. The transformation κ executes the computation by supplying it with the identity continuation. Moreover, if G happens to have a left adjoint F, the codensity monad is equal to GF, and the situation simplifies to the op-dual of Example 2 (6).

Examples 3 and 6 suggest that some actions of modules can be intuitively seen as functions that *run* computations, while the functor parts provide *context* for the execution. We say more about this view on modules in Section 5.

2.3 Adjunctions paired with a functor

Monads and pairs of adjoint functors are closely related: every adjunction induces a monad, and every monad is induced by a number of adjunctions. This can be extended to modules over monads: M-modules are induced by adjunctions that induce M together with an additional functor. The only condition imposed on the functor is that it has the same type as the right adjoint (for right modules) or the left adjoint (for left modules). In detail:

▶ **Theorem 7.** *Let $F : \mathscr{C} \to \mathscr{D}$ be a functor with a right adjoint $U : \mathscr{D} \to \mathscr{C}$. We denote the unit and the counit as η and ε respectively. Then:*

- *Let $L : \mathscr{C} \to \mathscr{D}$ be a functor. Then, UL is a left UF-module with the action given as $U\varepsilon L : UFUL \to UL$.*
- *Let $R : \mathscr{D} \to \mathscr{C}$ be a functor. Then, RF is a right UF-module with the action given as $R\varepsilon F : RFUF \to RF$.*

Conversely, every M-module arises from an adjunction that induces M together with a functor of the appropriate type. In the case of left modules, this fact was noticed by Dubuc [10]; we complement it with a suitable counterpart of the Kleisli construction for right modules.

▶ **Theorem 8.** *Let $F^{\mathrm{EM}} \dashv U^{\mathrm{EM}}$ be the Eilenberg–Moore adjunction for a monad M on a category $\mathscr{C}$. For a left M-module S, we define a functor $L : \mathscr{C} \to \mathbf{EM}(M)$ as follows:*

$$LA = \langle SA,\ MSA \xrightarrow{\overleftarrow{\mu}_A} SA\rangle$$

$$L(f : A \to B) = SA \xrightarrow{Sf} SB$$

The induced module $U^{\mathrm{EM}}L$ is equal to S.

▶ **Theorem 9.** *Let $F^{\mathrm{Kl}} \dashv U^{\mathrm{Kl}}$ be the Kleisli adjunction for a monad M on a category $\mathscr{C}$. For a right M-module S, we define a functor $R : \mathbf{Kleisli}(M) \to \mathscr{C}$ as follows:*

$$RA = SA$$

$$R(f : A \to MB) = SA \xrightarrow{Sf} SMB \xrightarrow{\overrightarrow{\mu}_B} SB$$

The induced module RF^{Kl} is equal to S.

2.4 Distributive laws and liftings

Since most results about monads boil down to results about adjunctions, the construction above suggests that we can generalise much of the theory of monads to the theory of modules. As an example, we now consider distributive laws and liftings, introduced and proved equivalent by Beck [8] (see also Barr and Wells [7, Sec. 9.2] or Tanaka's PhD dissertation [33]):

▶ **Definition 10.** A *distributive law* of an endofunctor G over a monad M is a natural transformation $\lambda : GM \to MG$ such that $\lambda \cdot G\mu = \mu G \cdot M\lambda \cdot \lambda M$ and $\lambda \cdot G\eta = \eta G$. Similarly, a distributive law of a monad T over an endofunctor G is a natural transformation $\lambda : TG \to GT$ such that $\lambda \cdot \mu G = G\mu \cdot \lambda G \cdot M\lambda$ and $\lambda \cdot \eta G = G\eta$. A distributive law between monads T and M is a natural transformation $\lambda : TM \to MT$ that is both a distributive law of T as a endofunctor over M as a monad and T as a monad over M as an endofunctor.

▶ **Definition 11.** Given a monad T and an endofunctor G, we call an endofunctor $\overline{G} : \mathbf{EM}(T) \to \mathbf{EM}(T)$ a *lifting* of G to $\mathbf{EM}(T)$ if $U^{\mathrm{EM}}\overline{G} = GU^{\mathrm{EM}}$, where $U : \mathbf{EM}(T) \to \mathscr{C}$ is the forgetful functor. Let M be a monad. We call a monad $\overline{M} : \mathbf{EM}(T) \to \mathbf{EM}(T)$ a lifting of M (as a monad) if it is a lifting as an endofunctor and additionally the identities $\mu^M U^{\mathrm{EM}} = U^{\mathrm{EM}}\mu^{\overline{M}}$ and $\eta^M U^{\mathrm{EM}} = U^{\mathrm{EM}}\eta^{\overline{M}}$ hold.

▶ **Theorem 12.** *Let M and T be monads, and G be an endofunctor. Liftings of G to $\mathbf{EM}(T)$ are in 1-1 correspondence with distributive laws of T over G. Moreover, liftings of M (as a monad) are in 1-1 correspondence with distributive laws of T over M (as monads).*

We extend these notions and the correspondence to include modules:

▶ **Definition 13.** A *distributive law* of a monad T over a right M-module S consists of a distributive law between monads $\lambda : TM \to MT$ together with a distributive law of a monad over an endofunctor $\overrightarrow{\lambda} : TS \to ST$ such that $\overrightarrow{\lambda} \cdot T\overrightarrow{\mu} = \overrightarrow{\mu}T \cdot S\lambda \cdot \overrightarrow{\lambda}M$.

▶ **Definition 14.** Let T be a monad and S be a right M-module. An *Eilenberg–Moore lifting* of S as a module consists of $\overline{M}$ and $\overline{S}$ together with a natural transformation $\overrightarrow{\mu}^{\overline{S}} : \overline{S}\,\overline{M} \to \overline{S}$ such that:

- $\overline{M} : \mathbf{EM}(T) \to \mathbf{EM}(T)$ is a lifting of M as a monad,
- $\overline{S} : \mathbf{EM}(T) \to \mathbf{EM}(T)$ is a lifting of S as a functor,
- $\overline{S}$ together with $\overrightarrow{\mu}^{\overline{S}}$ form a right $\overline{M}$-module,
- it is the case that $U^{\mathrm{EM}}\overrightarrow{\mu}^{\overline{S}} = \overrightarrow{\mu}^{S}U^{\mathrm{EM}}$.

▶ **Theorem 15.** *Distributive laws of a monad T over an M-module S and Eilenberg–Moore liftings of S are in 1-1 correspondence.*

3 Resumptions: monads freely generated by modules

In this section, we introduce the monad MS^* freely induced by a right M-module S. Its monadic structure is an obvious generalisation of Hyland, Plotkin, and Power's construction [20] for Moggi's [27] monad $M(GM)^*$ for an endofunctor G.

Since in this and the next sections, we are interested only in right modules, when no direction (left or right) is given, we mean *right* modules.

▶ **Theorem 16.** *Given a monad M, let $\langle S, \overrightarrow{\mu}\rangle$ be an M-module. The functor MS^* can be given monadic structure via a distributive law $\lambda : S^*M \to MS^*$.*

Proof. First, consider the natural transformation $\delta = \eta\overrightarrow{\mu} : SM \to MS$. It is easy to verify that it is a distributive law of the functor S over the monad M. Such a distributive law gives us the following lifting $\overline{M} : \mathbf{Alg}(S) \to \mathbf{Alg}(S)$ of M to $\mathbf{Alg}(S)$:

$$\overline{M}\langle A,\, SA \xrightarrow{a} A\rangle = \langle MA,\, SMA \xrightarrow{\delta_A} MSA \xrightarrow{Ma} A\rangle$$
$$\overline{M}f = Mf$$

Since the category $\mathbf{Alg}(S)$ is monadic over $\mathscr{C}$ (hence, $\mathbf{Alg}(S) \cong \mathbf{EM}(S^*)$), this lifting can be seen as a lifting to $\mathbf{EM}(S^*)$:

$$\overline{M} : \mathbf{EM}(S^*) \to \mathbf{EM}(S^*)$$

Applying Theorem 12, we obtain a distributive law $\lambda : S^*M \to MS^*$, which gives a monadic structure to MS^*. ◀

Using the definitions of the appropriate isomorphisms and with some calculation, we can read off a direct definition of the distributive law in terms of a fold, that is, the unique algebra homomorphism from the initial $(S(\text{-}) + MA)$-algebra to the following algebra, where $\mathsf{cons}_A : SS^*A \to S^*A$ is the action of the free S-algebra generated by the object A:

$$\lambda_A = (\!| f |\!) : S^*MA \to MS^*A,$$

where $(\!| f |\!)$ is the unique algebra homomorphism from the initial algebra to $\langle MS^*A,\, f\rangle$ for

$$f = [\mu^M_{S^*A} \cdot \mathsf{cons}_A \cdot \overrightarrow{\mu}^S_{S^*A},\, M\eta^{S^*}_A] : SMS^*A + MA \to MS^*A.$$

▶ **Example 17.** The monad MS^* is a generalisation of Moggi's resumption monad $M(GM)^*$ for an endofunctor G. Moggi's monad arises as the special case for $S = GM$. It follows from Example 2 (3) that GM is an M-module. Using the 'rolling rule' [6], Moggi's monad can be rewritten as $A \mapsto \mu X.M(GX + A)$. A distinctive feature of our construction is that in general it is not given by an initial algebra.

Moggi's monad is an important data structure in functional programming, as it is often used to implement a form of algebraic effects. The endofunctor G represents a signature, while M is a background monad. Handling of the signature takes G to M, which in Haskell is often the *IO* monad. Important examples of this pattern are given by streaming I/O libraries, which help to manage resources efficiently without losing purity; see, for example, Kiselyov [22].

▶ **Example 18.** We instantiate our resumption monad with the reader monad $(\text{-})^A$ together with its module $A \times (\text{-})$ (see Example 3) to obtain $((A \times (\text{-}))^*)^A$. It is a version of the state monad that accumulates the intermediate states in a sequence. (We have previously [28] given a 'coinductive' version of this example.)

The monad MS^* is an important construction in the theory of modules, since it is freely generated by S understood as a module. First, we notice that the monad M can be seen as its own module with $\overrightarrow{\mu} = \mu$. Moreover, this construction is functorial:

▶ **Definition 19.** We define a functor $\Delta : \mathbf{Mnd} \to \mathbf{Mod}$ as follows:

$$\Delta M = \langle M, M\rangle$$
$$\Delta f = \langle f, f\rangle$$

The above functor can be seen as a form of a (dependent) diagonal, hence the notation Δ. Mac Lane [23] calls the module ΔM the *right regular representation* of M, referring to a similar concept from representation theory in abstract algebra. Hirschowitz and Maggesi [18] call ΔM the *tautological* module of M.

▶ **Theorem 20.** *The monad MS^* is the free object in the category* $\mathbf{Mnd}$ *generated by S with respect to the functor Δ. More precisely, this means that for monads M and T, an M-module S, and a module morphism $\langle m, f\rangle : \langle M, S\rangle \to \Delta T$, there exists a unique monad morphism $k : MS^* \to T$ such that the following diagram commutes:*

$$\begin{tikzcd}
M \arrow[rr, "M\eta^{S^*}"] \arrow[drr, "m"'] & & MS^* \arrow[d, "k"] & S^* \arrow[l, "\eta^M S^*"'] & S \arrow[l, "\mathsf{emb}^S"'] \arrow[dll, "f"] \\
& & T & &
\end{tikzcd} \tag{1}$$

Proof. We define $k = \mu^T \cdot m[\![f]\!]$. Using the direct definition of λ, it can be shown using the properties of initial algebras that k is indeed a monad morphism.

It is easy to see that the diagram (1) commutes from the properties of monads and the freeness of S^*. To see that k is unique such a morphism, consider a monad morphism $r : MS^* \to T$ such that the diagram (1) commutes if we substitute r for k. Since $\eta^M S^* : S^* \to MS^*$ is a monad morphism, the composition $r \cdot \eta^M S^* : S^* \to T$ is a monad morphism, hence, from the freeness of S^*, we obtain that

$$r \cdot \eta^M S^* = [\![f]\!] \tag{2}$$

We calculate:

$$\begin{aligned}
r &= r \cdot \mu^{MS^*} \cdot \eta^{MS^*} MS^* && \text{(monads)}\\
&= r \cdot \mu^M \mu^{S^*} \cdot M\lambda S^* \cdot \eta^M \eta^{S^*} MS^* && \text{(def.)}\\
&= r \cdot \mu^M \mu^{S^*} \cdot \eta^M M\eta^{S^*} S^* && \text{(distr. law)}\\
&= r \cdot \mu^M \mu^{S^*} \cdot M\eta^M \eta^{S^*} S^* && \text{(monads)}\\
&= r \cdot \mu^M \mu^{S^*} \cdot M\lambda S^* \cdot M\eta^{S^*} \eta^M S^* && \text{(distr. law)}\\
&= r \cdot \mu^{MS^*} \cdot M\eta^{S^*} \eta^M S^* && \text{(def.)}\\
&= \mu^T \cdot rr \cdot M\eta^{S^*} \eta^M S^* && \text{(monad morphism)}\\
&= \mu^T \cdot m[\![f]\!] && \text{(LHS of (1) and (2))}\\
&= k && \text{(def.)}
\end{aligned}$$

◀

4 Algebras for modules

In this section, we introduce the notion of an algebra for a module. We show that the category of all such algebras for an M-module S coincides with the category of Eilenberg–Moore algebras for the monad MS^*.

▶ **Definition 21.** An algebra for an M-module S is a triple $\langle A,\ f : MA \to A,\ g : SA \to A\rangle$ such that:

1. The morphism f is an Eilenberg–Moore M-algebra.
2. The morphism g is an S-algebra.
3. The following coherence diagram commutes:

$$\begin{tikzcd}
SMA \arrow[r, "Sf"] \arrow[d, "\vec{\mu}_A"'] & SA \arrow[d, "g"] \\
SA \arrow[r, "g"] & A
\end{tikzcd}$$

A morphism between two algebras $\langle A, f, g\rangle$ and $\langle B, f', g'\rangle$ is a morphism $h : A \to B$ that is both an S-algebra homomorphism $f \to f'$ and an M-algebra homomorphism $g \to g'$. We denote the category of algebras for an M-module S as $\mathbf{ModAlg}(M, S)$.

▶ **Theorem 22.** *Let S be an M-module. If S^* exists, the obvious forgetful functor $U^{\mathrm{ModAlg}} : \mathbf{ModAlg}(M, S) \to \mathscr{C}$ has a left adjoint F^{ModAlg} given by:*

$$F^{\mathrm{ModAlg}} A = \langle MS^*A,\ f,\ g\rangle, \text{ where}$$

$$f = MMS^*A \xrightarrow{\mu^M_{S^*A}} MS^*A$$

$$g = SMS^*A \xrightarrow{\vec{\mu}_{S^*A}} SS^*A \xrightarrow{\mathsf{cons}_A} S^*A \xrightarrow{\eta^M_{S^*A}} MS^*A$$

$$F^{\mathrm{ModAlg}} h = MS^*h$$

The monad induced by this adjunction is equal to MS^*.

Proof. Consider the adjunction $F^{\mathrm{Alg}} \dashv U^{\mathrm{Alg}} : \mathscr{C} \rightharpoonup \mathbf{Alg}(S)$. The lifting $\overline{M}$ defined in the proof of Theorem 16 can be seen as a monad on $\mathbf{Alg}(S)$. It gives rise to an Eilenberg–Moore adjunction $F^{\mathrm{EM}} \dashv U^{\mathrm{EM}} : \mathbf{Alg}(S) \rightharpoonup \mathbf{EM}(\overline{M})$. The objects of $\mathbf{EM}(\overline{M})$ are algebras of the following shape:

$$\langle\langle A, g : SA \to A\rangle,\ f : \overline{M}\langle A, g\rangle \to \langle A, g\rangle\rangle$$

They satisfy the following conditions:

- The morphism g is an S-algebra (obviously).
- The morphism f has the Eilenberg–Moore property. Since $\overline{M}$ inherits its monadic structure from M, the morphism $f : MA \to A$ understood as a $\mathscr{C}$-morphism has the Eilenberg–Moore property for M.
- The morphism f is an algebra homomorphism between $\overline{M}\langle A, g\rangle = \langle MA,\ SMA \xrightarrow{\vec{\mu}_A} SA \xrightarrow{g} A \xrightarrow{\eta^M_A} MA\rangle$ and $\langle A, g\rangle$. The homomorphism diagram is then as follows:

$$\begin{tikzcd}
SMA \arrow[r, "\vec{\mu}_A"] \arrow[d, "Sf"'] & SA \arrow[r, "g"] & A \arrow[r, "\eta^M_A"] \arrow[dr, dashed, "\mathsf{id}_A"] & MA \arrow[d, "f"] \\
SA \arrow[rrr, "g"] & & & A
\end{tikzcd}$$

  Since f has the Eilenberg–Moore property, it is the case that $f \cdot \eta^M_A = \mathsf{id}_A$ (as indicated by the dashed arrow).

These are exactly the conditions for $\langle A, f, g\rangle$ to be an algebra for the module S, which means that $\mathbf{ModAlg}(M, S) \cong \mathbf{EM}(\overline{M})$. The adjunction in question is then given by the following composite adjunction:

$$F^{\mathrm{EM}} F^{\mathrm{Alg}} \dashv U^{\mathrm{Alg}} U^{\mathrm{EM}} : \mathscr{C} \rightharpoonup \mathbf{ModAlg}(M, S) \cong \mathbf{EM}(\overline{M})$$

It is easy to see that U^{ModAlg} agrees with $U^{\mathrm{Alg}} U^{\mathrm{EM}}$, so its left adjoint is given by $F^{\mathrm{EM}} F^{\mathrm{Alg}}$ modulo the isomorphism. Simple unfolding of the definitions of F^{Alg} and F^{EM} gives us that the direct definition of their composition is as specified in the theorem. ◀

▶ **Example 23.** We can instantiate the theorem above to the 'bialgebraic' proof by Hyland, Plotkin, and Power's [20] that $M(GM)^*$ is a coproduct of M and G^* in $\mathbf{Mnd}$. First, for two monads M and T, we define an $\langle M, T\rangle$-bialgebra as a triple $\langle A,\ f : MA \to A,\ g : TA \to A\rangle$, where f and g are Eilenberg–Moore algebra actions. All $\langle M, T\rangle$-bialgebras form a category, $\mathbf{BiAlg}(M, T)$, with morphisms given by $\mathscr{C}$-morphisms that are both M- and T-algebra homomorphisms. As shown by Kelly [21], in a category with coproducts, if the obvious forgetful functor from $\mathbf{BiAlg}(M, T)$ to the base category has a left adjoint, the induced monad is a coproduct of M and T in $\mathbf{Mnd}$. Indeed, for an M-module GM (see Example 2 (3) and (1)), one can prove that the category $\mathbf{ModAlg}(M, GM)$ is isomorphic to $\mathbf{BiAlg}(M, G^*)$ as follows.

Since $\mathbf{EM}(G^*) \cong \mathbf{Alg}(G)$, we can work with G-algebras (instead of Eilenberg–Moore G^*-algebras) in the third component of bialgebras. Given an algebra for a module $\langle A,\ f : MA \to A,\ g : GMA \to A\rangle$, we define the corresponding bialgebra as $\langle A,\ f,\ g \cdot G\eta_A : GA \to A\rangle$. Given a bialgebra $\langle A,\ f : MA \to A,\ g : GA \to A\rangle$, we define the corresponding algebra for a module as $\langle A,\ f,\ g \cdot Gf : GMA \to A\rangle$. The coherence condition follows easily from the fact that f is an Eilenberg–Moore algebra action. Simple calculation reveals that that the two transformations are mutual inverses. It is also easy to verify that a morphism between two algebras for a module is also a morphism between the corresponding bialgebras and *vice versa.*

Theorem 22 characterises the left adjoint to U^{ModAlg} (and so, to the forgetful functor $\mathbf{BiAlg}(M, G^*)$). The induced monad is indeed the free monad generated by the module GM, that is, $M(GM)^*$.

▶ **Example 24.** As defined by Atkey *et al.* [5], following Filinski and Støvring [13], a G-and-M-algebra is a triple $\langle A,\ m : MA \to A,\ f : GA \to A\rangle$, where M is a monad, G is an endofunctor, f is a morphism, and m is an Eilenberg–Moore algebra action. Morphisms between two G-and-M-algebras are $\mathscr{C}$-morphisms that are both G- and M-algebra homomorphisms. The initial G-and-M-algebra (whose carrier is given by $\mu MG \cong M(\mu GM)$) is used to model effectful datatypes, which interleave structure and monadic effects. Employing the isomorphism $\mathbf{EM}(G^*) \cong \mathbf{Alg}(G)$, one can easily see that the category of G-and-M-algebras is isomorphic to $\mathbf{BiAlg}(M, G^*)$, and so, as described in the previous example, isomorphic to $\mathbf{ModAlg}(M, GM)$. Since $F^{\mathrm{ModAlg}} : \mathscr{C} \to \mathbf{ModAlg}(M, GM)$ is cocontinuous (since it is a left adjoint), the initial G-and-M-algebra can be obviously reconstructed as $F^{\mathrm{ModAlg}}0$, where 0 is the initial object of $\mathscr{C}$.

▶ **Theorem 25.** *If S^* exists, the functor U^{ModAlg} is strictly monadic. This entails that the category $\mathbf{ModAlg}(M, S)$ is isomorphic to $\mathbf{EM}(MS^*)$.*

Proof. We use the strict version of Beck's monadicity theorem (see Mac Lane [23, Sec. VI.7]). We have already shown that U^{ModAlg} is a right adjoint, so it remains to show that it creates coequalisers for those parallel h_0, h_1 in $\mathbf{ModAlg}(M, S)$ for which $U^{\mathrm{ModAlg}}h_0$ and $U^{\mathrm{ModAlg}}h_1$ have a split coequaliser in $\mathscr{C}$.

Let $h_0, h_1 : \langle A, f^A, g^A\rangle \to \langle B, f^B, g^B\rangle$ be such a pair. Let c be a split coequaliser of $U^{\mathrm{ModAlg}}h_0$ and $U^{\mathrm{ModAlg}}h_1$. In other words, there exist morphisms s and t such that the following diagram commutes in $\mathscr{C}$ and in which the two horizontal compositions are the identities:

$$\begin{tikzcd} B \arrow[r, "t"] \arrow[d, "c"] & A \arrow[r, "h_0"] \arrow[d, "h_1"] & B \arrow[d, "c"] \\ C \arrow[r, "s"] & B \arrow[r, "c"] & C \end{tikzcd} \tag{3}$$

We need to show that there exist unique $f^C : MC \to C$ and $g^C : SA \to A$ such that $\langle C, f^C, g^C\rangle$ is an algebra for a module, and $c : \langle B, f^B, g^B\rangle \to \langle C, f^C, g^C\rangle$ is a homomorphism and a coequaliser of h_0 and h_1. From the monadicity of the forgetful functors $U^{\mathrm{EM}} : \mathbf{EM}(M) \to \mathscr{C}$ and $U^{\mathrm{Alg}} : \mathbf{Alg}(S) \to \mathscr{B}$, we obtain that there exist a unique Eilenberg–Moore

M-algebra $\langle C, f^C \rangle$ and a unique S-algebra $\langle C, g^C \rangle$, where

$$f^C = MC \xrightarrow{Ms} MB \xrightarrow{f^B} B \xrightarrow{c} C,$$
$$g^C = SC \xrightarrow{Ss} SB \xrightarrow{g^B} B \xrightarrow{c} C,$$

such that c is the coequaliser of $h_0, h_1 : \langle A, f^A \rangle \to \langle B, f^B \rangle$ understood as Eilenberg–Moore M-algebra homomorphisms and simultaneously the coequaliser of $h_0, h_1 : \langle A, g^A \rangle \to \langle B, g^B \rangle$ understood as S-algebra homomorphisms. Thus, it is left to check that $\langle C, f^C, g^C \rangle$ is an algebra for a module, that is, that the tuple f^C and g^C satisfy the condition (3) from Definition 21:

$$\begin{aligned}
g^C \cdot Sf^C &= c \cdot g^B \cdot Ss \cdot Sc \cdot Sf^B \cdot SMs && \text{(def.)}\\
&= c \cdot g^B \cdot Sh_1 \cdot St \cdot Sf^B \cdot SMs && \text{(diag. (3))}\\
&= c \cdot h_1 \cdot g^A \cdot St \cdot Sf^B \cdot SMs && (h_1 \text{ homomorph.})\\
&= c \cdot h_0 \cdot g^A \cdot St \cdot Sf^B \cdot SMs && (c \text{ coequaliser})\\
&= c \cdot g^B \cdot Sh_0 \cdot St \cdot Sf^B \cdot SMs && (h_0 \text{ homomorph.})\\
&= c \cdot g^B \cdot Sf^B \cdot SMs && \text{(diag. (3))}\\
&= c \cdot g^B \cdot \overrightarrow{\mu}_B \cdot SMs && \text{(coherence)}\\
&= c \cdot g^B \cdot Ss \cdot \overrightarrow{\mu}_C && (\overrightarrow{\mu} \text{ nat.})\\
&= g^C \cdot \overrightarrow{\mu}_C && \text{(def.)}
\end{aligned}$$

◀

5 Summary and future work

In this paper, we have taken a closer look at the notion of module over a monad, focusing mainly on right modules. We illustrate our results with a number of examples, some of them new, some just being a reformulation in the language of modules of previously known results.

One important application we hope for is in functional programming, where structures similar to resumptions appear in the form of streaming I/O libraries [22] and adaptation of algebraic effects [31]. Corollary 20 and Theorem 25 give universal properties of the monad MS^*, which can be used for equational reasoning about programs that utilise them [24]. Providing a simple description of an adjunction that gives rise to MS^* gives us a more efficient implementation via the codensity monad trick [17].

Another question is how modules relate to monadic effects, especially their algebraic presentations, extensively studied by Plotkin and Power [29], and handlers, in the sense of Plotkin and Pretnar [30]. An algebraic theory induces a monad M as the family of its free models, while handlers are given by other models, that is, Eilenberg–Moore algebras of the monad M. As mentioned in Example 5, every Eilenberg–Moore algebra is a module for a constant endofunctor, but there are families of models that are parametric in variables (for example, the free model). These can be modelled by general left modules.

As suggested by Example 3, the actions of right modules may represent functions that run the computations in some context. In these case, the context is a global state A; recall the types: $A \times (A \times X)^A \to A \times X$ and $A \times X^A \to A \times X$. Is this a more general situation? Below, we give another example:

▶ **Example 26.** Consider the monad T of binary trees on **Set** with variables in leaves and the monad multiplication given by substitution. Intuitively, we interpret them as choice trees of randomised computations, in which every choice is equally probable. The context of execution is given by $CX = X \times \mathbb{B}^\omega$, where $\mathbb{B}^\omega$ is the set of infinite binary streams, representing possible future sequences of coin tosses. Now, to run the computation in the context, we go down the tree, choosing a branch based on the front element of the stream (left upon 0, right upon 1), at each step discarding the front element. Then, the result of $\overrightarrow{\mu}_X : TX \times \{0,1\}^\omega \to X \times \mathbb{B}^\omega$ is a pair consisting of the variable in the leaf that is reached by going down the tree as specified in the prefix of an appropriate length paired with the unused 'tail' of the stream.

6 Related work

The research presented in this paper is inspired by our previous work [28], in which the *coinductive* resumption monad MS^∞ was studied, where S^∞ is the free completely iterative monad [3] defined as $S^\infty A = \nu X.SX + A$. There, we use an arbitrary right module S instead of GM mainly to simplify the presentation and the proofs, although the main result considers the monad $M(GM)^\infty$, which is similar to Moggi's monad.

Modules are used by Adámek, Milius, and Velebil [2, 25] to capture the notion of guardedness in their study of iterative monads. They define an *idealised* monad as a right M-module S together with a suitably coherent natural transformation $\sigma : S \to M$. The general intuition for idealised monads is that S is a 'subset' (especially if σ is monic) of computations that have some good properties, which are retained after composing with any other computation. For instance, consider Example 4, in which the 'ideal' of the non-empty list monad is given by lists of length at least n. An important example of idealised monads are *ideal* monads, which are defined by the property $M = S + \mathsf{Id}$; see also Ghani and Uustalu [15] for an extended discussion.

There are some obvious generalisations possible. For example, we can allow S to be a functor to a different category. This definition was used by Street [32] to define the Eilenberg–Moore object in a 2-category: it is a universal left module (in the generalised sense). Hirschowitz and Maggesi [18, 19] and Ahrens [4] use generalised left modules to capture the construction of higher-order syntax and semantics. They, too, discuss the elementary theory of modules, but from a slightly different angle: instead of **Mod**, they study the category of modules over a single monad M, that is, a fibre of **Mod** with respect to the functor $\mathbf{Mod} \to \mathscr{C}^\mathscr{C}$ that extracts the functor part of a module. This functor has some nice properties: it has a left adjoint given by $G \mapsto GM$ and reflects (co)limits, see Example 2 (5).

Resumptions were introduced by Milner [26] to capture the semantics of concurrency (see also Abramsky [1]). In programming, resumptions (known also as 'trampolined style' [14] or 'engines' [11, 16]) were first used to control program flow. The first use of resumptions (although, of course, not explicitly named so) was probably the famous result on structured programming by Böhm and Jacopini [9].

Acknowledgements

We would like to thank the Algebra of Programming group at Oxford University for discussions; the anonymous referees for their constructive comments, which helped us to improve this paper; and the EPSRC, whose grant on *Unifying Theories of Generic Programming* (EP/J010995/1) partially supported this work.

References

1 Samson Abramsky. Retracing some paths in process algebra. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science (LNCS)*, pages 1–17. Springer, 1996.

2 Jiří Adámek, Stefan Milius, and Jiří Velebil. On rational monads and free iterative theories. *Electronic Notes in Theoretical Computer Science*, 69:23–46, 2002.

3 Jiří Adámek, Stefan Milius, and Jiří Velebil. Free iterative theories: A coalgebraic view. *Mathematical Structures in Computer Science*, 13(2):259–320, 2003.

4 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, FirstView:1–35, 12 2014.

5 Robert Atkey, Neil Ghani, Bart Jacobs, and Patricia Johann. Fibrational induction meets effects. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures*, volume 7213 of *Lecture Notes in Computer Science (LNCS)*, pages 42–57. Springer, 2012.

6 Roland Carl Backhouse, Marcel Bijsterveld, Rik van Geldrop, and Jaap van der Woude. Categorical fixed point calculus. In David H. Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, volume 953 of *Lecture Notes in Computer Science (LNCS)*, pages 159–179. Springer, 1995.

7 Michael Barr and Charles F. Wells. *Toposes, Triples, and Theories*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985.

8 Jonathan M. Beck. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer Berlin / Heidelberg, 1969. 10.1007/BFb0083084.

9 Corrado Böhm and Giuseppe Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, May 1966.

10 Eduardo J. Dubuc. *Kan extensions in enriched category theory*. Lecture Notes in Mathematics. Springer, Berlin, 1970.

11 R. Kent Dybvig and Robert Hieb. Engines from continuations. *Computer Languages*, 14(2):109–123, 1989.

12 Samuel Eilenberg and John C. Moore. Adjoint functors and triples. *Illinois Journal of Mathematics*, 9(3):381–398, 09 1965.

13 Andrzej Filinski and Kristian Støvring. Inductive reasoning about effectful data types. In *International Conference on Functional Programming*, pages 97–110. ACM, 2007.

14 Steven E. Ganz, Daniel P. Friedman, and Mitchell Wand. Trampolined style. In Didier Rémy and Peter Lee, editors, *International Conference on Functional Programming*, pages 18–27. ACM, 1999.

15 Neil Ghani and Tarmo Uustalu. Coproducts of ideal monads. *Theoretical Informatics and Applications*, 38(4):321–342, 2004.

16 Christopher T. Haynes and Daniel P. Friedman. Engines build process abstractions. In *LISP and Functional Programming*, pages 18–24, 1984.

17 Ralf Hinze. Kan extensions for program optimisation, or: Art and Dan explain an old trick. In Jeremy Gibbons and Pablo Nogueira, editors, *Mathematics of Program Construction - 11th International Conference, MPC 2012, Madrid, Spain, June 25-27, 2012. Proceedings*, volume 7342 of *Lecture Notes in Computer Science*, pages 324–362. Springer, 2012.

18 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In Daniel Leivant and Ruy J. G. B. de Queiroz, editors, *Workshop on Logic, Language, Information and Computation*, volume 4576 of *Lecture Notes in Computer Science (LNCS)*, pages 218–237. Springer, 2007.

19 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, 2010.

20 Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1-3):70–99, 2006.

21 Gregory M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22:1–83, 8 1980.

22 Oleg Kiselyov. Iteratees. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming*, volume 7294 of *Lecture Notes in Computer Science (LNCS)*, pages 166–181. Springer, 2012.

23 Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 1998.

24 Erik Meijer, Maarten M. Fokkinga, and Ross Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In John Hughes, editor, *Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 124–144. Springer, 1991.

25 Stefan Milius. Completely iterative algebras and completely iterative monads. *Information and Computation*, 196:1–41, 2005.

26 Robin Milner. Processes: A mathematical model of computing agents. In *Logic Colloquium '73*, Studies in logic and the foundations of mathematics, pages 157–173. North-Holland Pub. Co., 1975.

27 Eugenio Moggi. An abstract view of programming languages. Technical report, Edinburgh University, 1989.

28 Maciej Piróg and Jeremy Gibbons. The coinductive resumption monad. *Electronic Notes in Theoretical Computer Science*, 308:273–288, 2014. Mathematical Foundations of Programming Semantics (MFPS XXX).

29 Gordon D. Plotkin and A. John Power. Computational effects and operations: An overview. *Electronic Notes in Theoretical Computer Science*, 73:149–163, 2004.

30 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013.

31 Tom Schrijvers, Nicolas Wu, Benoit Desouter, and Bart Demoen. Heuristics entwined with handlers combined. In *PPDP 2014: Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming.* Association for Computing Machinery (ACM), 2014.

32 Ross Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2(2):149–168, 1972.

33 Miki Tanaka. *Pseudo-Distributive Laws and a Unified Framework for Variable Binding.* PhD thesis, University of Edinburgh, 2005.