ON SOME ASPECTS OF THE THEORY OF MONADS

CARSEN BERGER

Abstract. This paper provides an introduction to the theory of monads. The main result of the paper is a folkloric proof of Beck's monadicity theorem which gives an explicit construction of the equivalence involved. Several examples of monads are presented which illustrate the variety of guises in which monads can appear.

Contents

1. Introduction

Monads, like many concepts in category theory, are ubiquitous throughout mathematics. Because of the myriad guises in which they appear it is difficult to give an informal, high-level description of monad theory. It will be most instructive to examine, case by case, how monads appear in several particular areas.

Monads originally arose in algebraic topology. Perhaps the first one explicitly considered was used by Godement in 1958 to embed a flabby sheaf into a sheaf for purposes of computing homology groups. Godement, apparently not convinced that his new discovery was interesting enough to warrant an interesting name, referred to his monad as simply the "standard construction" (of a flabby sheaf embedding).

One conceptualization of a monad is as a process, or as a machine that does something. Since a monad consists of a functor together with two natural transformations, this fits with the intuitive notion of arrows as capturing change. Many of the monads we will see take some uninteresting object—a set X, for example—and turn it into something more structured, such as the set of ultrafilters on X. The notion of monad as process also explains the use of monads in computer science.

Date: September 23, 2011.

This paper is dedicated to the memory of my father, Dr. Glenn W. Berger, whose love of science was always a guiding light.

Since computation is a process, monads can be used to give models of computation. We shall see how this can be done in Section 6.

Monads can also be thought of as theories. As we will see, there is a notion of an algebra of a monad. These are the structured objects produced by the monad-as-process. In the ultrafilter example above, an algebra would be a set X together with a map from the set of ultrafilters on X to X (subject to certain conditions). In much the same way as a theory describes the structure of its models, monads describe the structure of their algebras. In fact, we will see a very real correspondence between these two notions in Section 5.

This paper introduces monad theory and offers several examples along the way. Section 2 presents the basic theory and some motivating examples. Section 3 is devoted to the concept of monadicity; monadicity refers to the conditions under which a category $\mathcal{B}$ can be "described" by a monad on another category $\mathcal{C}$. This section contains the main result of the paper, a folkloric proof of Beck's monadicity theorem. The author claims no originality for this proof, but it is different from most of the standard versions in the literature. The proof here progresses by stages, assuming conditions one by one and showing that even if monadicity is not achieved, partial results can in some cases be reached. Section 4 gives an application of Beck's theorem, using it to show that the category of groups is monadic over **Set**. Section 5 discusses the relationship between monads and algebraic theories, and Section 6 presents an example of a monad (as process) that occurs in computer science.

2. Preliminaries

In this section I present the general theory of monads and collect some basic results. I will introduce monads through a motivating example.

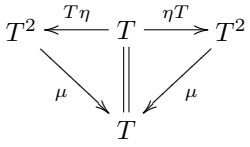
Example 2.1. Let $F : \mathbf{Set} \to \mathbf{Mon}$ be the free monoid functor which takes a set X to the monoid generated by the elements of X, and let $U : \mathbf{Mon} \to \mathbf{Set}$ be the forgetful functor which takes a monoid M to its underlying set. Then F is left adjoint to U—a point we will come back to. We can consider the composition $UF : \mathbf{Set} \to \mathbf{Set}$ which takes a set X to the underlying set of the free monoid generated by X. Elements of UFX are just lists $(x_1 \dots x_k)$ of elements of X, where $k \geq 0$.

For each set X we have a function $\eta_X : X \to UFX$ which maps an element x to the list (x). These η_X actually form a natural transformation $\eta : \mathrm{Id} \Rightarrow UF$.

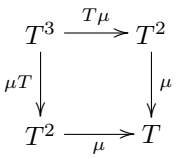
We can also think of iterating UF to get lists of lists. Elements of $UFUFX$ are of the form $((x_{1,1} \dots x_{k_1,1}) \dots (x_{1,n} \dots x_{k_n,n}))$, where the x_i are in X and both n and all the k_i are greater than or equal to 0. There is a sort of multiplication here that corresponds to the operation of erasing parentheses. That is, there is a map $\mu_X : UFUFX \to UFX$ which takes a list of lists $((x_{1,1} \dots x_{k_1,1}) \dots (x_{1,n} \dots x_{k_n,n}))$ to the list $(x_{1,1} \dots x_{k_1,1} \dots x_{1,n} \dots x_{k_n,n})$. This also gives a natural transformation $\mu : UFUF \Rightarrow UF$.

You may ask why η and μ were the "right" things to consider. Of course there are many other things we can do with lists besides erasing parentheses; how convenient that the maps we chose turned out to be natural! The justification for this will become clear when we discuss the relationship between monads and adjunctions, but for now we want to abstract a definition out of the construction of the list monad.

Definition 2.2. A *monad* on a category $\mathcal{C}$ is a triple (T, η, μ), where $T : \mathcal{C} \to \mathcal{C}$ is an endofunctor and $\eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow T$ and $\mu : T^2 \Rightarrow T$ are natural transformations such that the diagrams



and

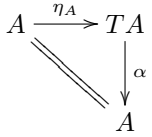


commute. We call η and μ the *unit* and *multiplication* of the monad, respectively.

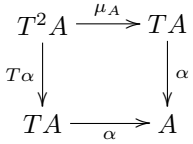
The first diagram is the two-sided identity law for monads and the second is the associativity law. In the example of the list monad, the associativity law just says that if we have a list of lists of lists, it doesn't matter in which order we erase the parentheses to get down to a single list.

Sticking with this example, one might also be interested in putting a monoid structure on the set X itself. This can be done using the notion of algebras of a monad.

Definition 2.3. If T is a monad on a category $\mathcal{C}$, a *T-algebra* is a pair (A, α) where A is an object of $\mathcal{C}$ and $\alpha : TA \to A$ is an arrow, called the *structure map*, such that the diagrams

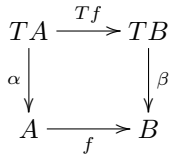


and



commute.

Definition 2.4. If (A, α) and (B, β) are T-algebras, then a *morphism of T-algebras* is an arrow $f : A \to B$ in $\mathcal{C}$ such that the diagram



commutes.

If we think of the monad as creating a structured object TA, then the structure map carries this structure back down to A. Algebras will play a central role in Section 3. The category of algebras of the list monad, which arises from an adjunction between **Set** and **Mon**, is precisely **Mon**, the category of monoids.

To move deeper into monad theory we need to understand the fundamental relationship between monads and adjunctions. I mentioned above that the free and forgetful functors in the definition of the list monad form an adjunction. In fact, this adjunction completely determined everything else that followed in the construction; the list monad was induced in a canonical way. P. Huber proved the following result in 1961.

Theorem 2.5. *Suppose $U : \mathcal{B} \to \mathcal{C}$ has a left adjoint $F : \mathcal{C} \to \mathcal{B}$ with the adjunction given by the natural transformations $\eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow UF$ and $\varepsilon : FU \Rightarrow \mathrm{Id}_{\mathcal{B}}$. Then $(UF, \eta, U\varepsilon F)$ is a monad on $\mathcal{C}$.*

The reader may wish to supply the proof; it is just a matter of verifing the correct identities. The unitary identity is verified using the triangle identities that characterize an adjunction, and the associative identity is verified using the definition of natural transformation.

The converse, that every monad arises from an adjunction (usually from more than one, in fact), was proved independently, using two distinct constructions, by Eilenberg and Moore and by H. Kleisli, both in 1965.

Theorem 2.6. *Let (T, η, μ) be a monad on $\mathcal{C}$. Then there is a category $\mathcal{B}$ and an adjoint pair $F \dashv U : \mathcal{B} \to \mathcal{C}$ such that $T = UF$, $\eta : \mathrm{Id}_{\mathcal{C}} \Rightarrow UF = T$ is the unit, and $\mu = U\varepsilon F$, where ε is the counit of the adjunction.*

To illustrate the beauty of this observation, let's look at an example that plays off the connection between order theory and category theory.

Construction 2.7. Given a poset $(P, \leq)$ we can construct the *poset category* $\mathcal{P}$ whose objects are elements of P; there is a unique arrow $f : x \to y$ if and only if $x \leq y$.

Definition 2.8. Let P and Q be posets. A *Galois connection* is a pair of maps $F : P \to Q$ and $G : Q \to P$ such that $F(p) \leq q$ if and only if $p \leq G(q)$ for all $p \in P$, $q \in Q$.

Galois connections, in addition to being interesting in their own right, are important both to Galois theory and to theoretical computer science. They have strong ties to the calculus of fixed points.

Definition 2.9. Let P be a poset. A *closure operator* on P is a map $c : P \to P$ such that for all $x, y \in P$,

(1) $x \leq c(x)$,
(2) $x \leq y$ implies $c(x) \leq c(y)$, and
(3) $c(c(x)) = c(x)$.

An element $x \in P$ is called *closed* if $c(x) = x$.

The terminology comes from the following standard example.

Example 2.10. Let T be a topological space and let $\mathcal{P}(T)$ be its power set. Then $\mathcal{P}(T)$ ordered by inclusion is a poset, and the map c which takes a set $X \subseteq T$ to its closure X^c is a closure operator. The closed elements of $\mathcal{P}(T)$ as defined above are precisely the closed sets of T under the given topology.

If F and G form a Galois connection, then the composite FG is a closure operator. This follows from the cancellation, monotonicity, and semi-inverse rules for Galois connections, which I have not stated here but which are all easy consequences of the definition. Conversely, if $c : P \to P$ is any closure operator, we can recognize that it arises from a Galois connection by setting $Q := \{p \in P : c(p) = p\}$, $F : P \to Q$ to be such that $F(p) = c(p)$, and $G : Q \to P$ to be the inclusion map. Then $c = FG$.

We could also have arrived at this result through abstract nonsense by noting that Galois connections are precisely adjunctions between poset categories and that closure operators are precisely monads on poset categories. Thus in a very real way, the entire theory of Galois connections is just monad theory on poset categories. Quite a large swathe of order theory, and in particular lattice theory, can be interpreted in this context.

3. Monadicity

Recall that a T-algebra (A, α) can be thought of as endowing the object A with some structure determined by T. A natural question to ask, then, is when can a category $\mathcal{B}$ be thought of as a "category of algebras" of a monad on some category $\mathcal{C}$? Intuitively, this question is asking for a monad that "describes" $\mathcal{B}$. This notion of description will be made more precise in Section 5 when we deal with monads as algebraic theories. For now, we can formalize the question by introducing the concepts of a category of T-algebras and of monadicity.

Definition 3.1. The *Eilenberg-Moore category* of a monad T on a category $\mathcal{C}$, denoted $\mathcal{C}^T$, is the category whose objects are T-algebras and whose arrows are morphisms of T-algebras, as defined in Definitions 2.3 and 2.4.

Definition 3.2. Let $F \dashv U : \mathcal{B} \to \mathcal{C}$ be an adjunction with unit η and counit ε and let $T = (UF, \eta, U\varepsilon F)$ be the induced monad. The *Eilenberg-Moore comparison functor* is the functor $\Phi : \mathcal{B} \to \mathcal{C}^T$ which takes an object B to the algebra $(UB, U\varepsilon_B)$ and a map f to Uf.

Definition 3.3. A functor U is *monadic* if U has a left adjoint for which the corresponding Eilenberg-Moore comparison functor Φ is an equivalence of categories. If Φ is full and faithful then U is *premonadic*. We say that U is of *descent type* if it is premonadic and of *effective descent type* if it is monadic.

We will prove a theorem due to Beck which gives necessary and sufficient conditions on U and $\mathcal{B}$ for U to be monadic. First it will be necessary to state some definitions used in the theorem.

Definition 3.4. We collect some standard terminology concerning epimorphisms.

(1) An arrow $f : A \to B$ is an *epimorphism* (and is said to be *epic* or an *epi* for short) if it is right-cancellable: if $g \circ f = h \circ f$, then $g = h$.
(2) An arrow is a *regular epimorphism* if it is the coequalizer of some pair of arrows.

(3) An epimorphism is *absolute* if it is preserved by any functor; that is, Ff is also epic for any functor F. (This is not in general true of arbitrary epimorphisms.)
(4) An arrow is a *split epimorphism* if it has a right inverse; that is, there exists $g : B \to A$ such that $f \circ g = \mathrm{id}_B$.

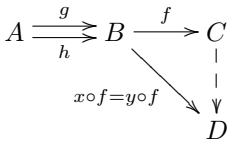
Lemma 3.5. *A split epimorphism is an absolute epimorphism; a regular epimorphism is an epimorphism.*

Proof. Suppose $f : A \to B$ is split epic with right inverse $g : B \to A$. Let $x, y : B \rightrightarrows C$ be such that $x \circ f = y \circ f$. Then

$$\begin{aligned} & x \circ f \circ g = y \circ f \circ g \\ \implies & x \circ \mathrm{id}_B = y \circ \mathrm{id}_B \\ \implies & x = y, \end{aligned}$$

and so f is epic. Now let F be any functor. Then $Ff \circ Fg = F(f \circ g) = F(\mathrm{id}_B) = \mathrm{id}_{FB}$, so Ff is split epic and thus epic. Therefore f is an absolute epi.

Now suppose $f : B \to C$ is a regular epi; suppose it coequalizes $g, h : A \rightrightarrows B$. Let $x, y : C \rightrightarrows D$ be such that $x \circ f = y \circ f$. Then this composition also equalizes g and h, so by the universal property of f there is a unique arrow making



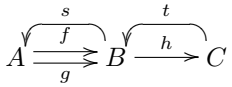
commute. But both x and y make the above triangle commute, so we must have $x = y$. Therefore f is epic. □

Definition 3.6. A *parallel pair* in a category is a pair of arrows with the same domain and codomain:

$$A \underset{g}{\overset{f}{\rightrightarrows}} B$$

A parallel pair is *split* if there is an arrow $s : B \to A$ such that $f \circ s = \mathrm{id}_B$ and $g \circ s \circ f = g \circ s \circ g$.

Definition 3.7. A *split coequalizer* is a collection of objects and arrows



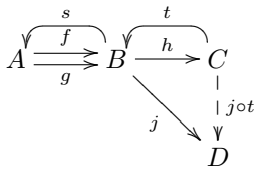
such that

(1) $f \circ s = \mathrm{id}_B$,
(2) $g \circ s = t \circ h$,
(3) $h \circ t = \mathrm{id}_C$, and
(4) $h \circ f = h \circ g$.

Definition 3.8. An *absolute coequalizer* is a coequalizer which remains a coequalizer upon application of any functor.

Lemma 3.9. *A split coequalizer is an absolute coequalizer.*

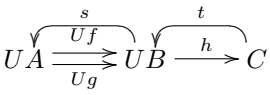
Proof. Suppose we have a split coequalizer as in Definition 3.7 above and a map $j : B \to D$ such that $j \circ f = j \circ g$. Then we have the following diagram.



The arrow $j \circ t$ makes the triangle commute because $t \circ h = \mathrm{id}_C$, and it is unique because h is a split epi. This gives the required universal property, and so h is a coequalizer.

Since a split coequalizer is defined in terms of equations involving composition and identities, which are preserved by any functor, split coequlizers are preserved by any functor. Thus h is absolute. □

Definition 3.10. If $U : \mathcal{B} \to \mathcal{C}$ is a functor, a *U-split pair* is a parallel pair f, g in $\mathcal{B}$ for which there is a split coequalizer



in $\mathcal{C}$.

Definition 3.11. A functor F is said to *reflect isomorphisms* if whenever Ff is an isomorphism, so is f.

Note that this is *not* the same as saying that if FX is isomorphic to FY, then X is isomorphic to Y. For example, the underlying set functor $U : \mathbf{Grp} \to \mathbf{Set}$ reflects isomorphisms—this is the same as saying that a group homomorphism is an isomorphism if it is a bijection. However, we might have two groups with isomorphic underlying sets which are not isomorphic as groups.

Although we will only prove one direction of Beck's theorem, the converse holds: the three conditions are necessary as well as sufficient for monadicty.

Theorem 3.12. *Let $U : \mathcal{B} \to \mathcal{C}$ be a functor with a left adjoint F, let $T = (UF, \eta, U\varepsilon F)$ be the associated monad, let $\mathcal{C}^T$ be its Eilenberg-Moore category, and let $\Phi : \mathcal{B} \to \mathcal{C}^T$ be the Eilenberg-Moore comparison functor. Then*

1. *if $\mathcal{B}$ has coequalizers of U-split pairs, Φ has a left adjoint Ψ;*
2. *if, furthermore, U preserves coequalizers of U-split pairs, the unit $\eta' : \mathrm{Id}_{\mathcal{C}^T} \Rightarrow \Phi\Psi$ is an isomorphism;*
3. *if, furthermore, U reflects isomorphisms, the counit $\varepsilon' : \Psi\Phi \Rightarrow \mathrm{Id}_{\mathcal{B}}$ is an isomorphism.*

Thus, if all three conditions are satisfied, U is monadic.

Proof. Assume first that $\mathcal{B}$ has coequalizers of U-split pairs; we will construct the left adjoint Ψ.

Let $(A, \alpha) \in \mathcal{C}^T$ be a T-algebra with $\alpha : UFA \to A$ in $\mathcal{C}$. We then have the following parallel pair in $\mathcal{B}$:

$$FUFA \underset{F\alpha}{\overset{\varepsilon_{FA}}{\rightrightarrows}} FA$$

This is a U-split pair as follows.

$$UFUFA \underset{UF\alpha}{\overset{U\varepsilon_{FA}}{\rightrightarrows}} UFA \xrightarrow{\alpha} A \qquad \eta_{UFA} \quad \eta_A$$

We have $U\varepsilon_{FA} \circ \eta_{UFA} = 1_{UFA}$ by the unitary identity for monads, $UF\alpha \circ \eta_{UFA} = \eta_A \circ \alpha$ by naturality of η, and $\alpha \circ \eta_A = 1_A$ and $\alpha \circ U\varepsilon_{FA} = \alpha \circ UF\alpha$ by definition of a T-algebra. Thus, by assumption, there is some coequalizer of ε_{FA} and $F\alpha$ in $\mathcal{B}$ which is unique up to isomorphism; using choice, fix one and call it $\Psi(A, \alpha)$. We shall denote the associated coequalizer arrow by $\kappa_{(A,\alpha)}$, or just κ_A when there is no risk of confusion.

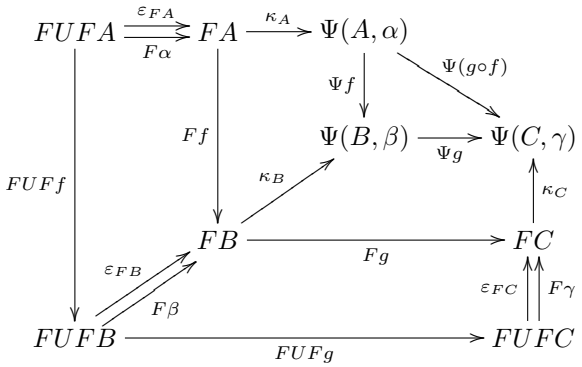
If we have $f : (A, \alpha) \to (B, \beta)$ in $\mathcal{C}^T$, the above construction gives us the following diagram in $\mathcal{B}$:

$$\begin{array}{ccccc} FUFA & \underset{F\alpha}{\overset{\varepsilon_{FA}}{\rightrightarrows}} & FA & \xrightarrow{\kappa_A} & \Psi(A,\alpha) \\ {\scriptstyle FUFf}\downarrow & & {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle \Psi f} \\ FUFB & \underset{F\beta}{\overset{\varepsilon_{FB}}{\rightrightarrows}} & FB & \xrightarrow{\kappa_B} & \Psi(B,\beta) \end{array}$$

The left square commutes serially, the upper square by naturality of ε and the lower square by definition of a morphism of algebras. This, together with the fact that κ_B coequalizes ε_{FB} and $F\beta$, says exactly that $\kappa_B \circ Ff \circ \varepsilon_{FA} = \kappa_B \circ Ff \circ F\alpha$. The universal property of $\Psi(A, \alpha)$ then gives us the unique map which we call Ψf.

This universal property also forces the equality $\Psi(1_{(A,\alpha)}) = 1_{\Psi(A,\alpha)}$.

To check that Ψ preserves composition, consider the diagram



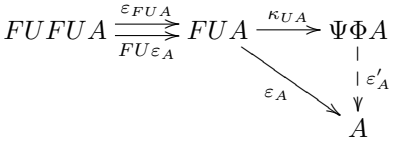
and note also that $Fg \circ Ff = F(g \circ f)$ and $FUFg \circ FUFf = FUF(g \circ f)$ by functoriality of F and U. By the above we know that everything in the above diagram commutes except for the top-right-most triangle, and we would like to show that that, too, commutes.

A bit of diagram chasing tells us

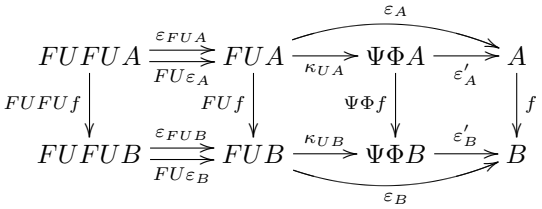
$$\begin{aligned} \Psi(g \circ f) \circ \kappa_A &= \kappa_C \circ F(g \circ f) \\ &= \kappa_C \circ Fg \circ Ff \\ &= \Psi g \circ \kappa_B \circ Ff \\ &= \Psi g \circ \Psi f \circ \kappa_A \end{aligned}$$

and so, since κ_A is a regular epimorphism, this implies $\Psi(g \circ f) = \Psi g \circ \Psi f$. Thus Ψ is a functor.

We now prove that Ψ is left adjoint to Φ; we begin by constructing the counit $\varepsilon' : \Psi\Phi \Rightarrow \mathrm{Id}_{\mathcal{B}}$. Starting with an object $A \in \mathcal{B}$, taking ΦA gives us the algebra $(UA, U\varepsilon_A)$, so $\Psi\Phi A$ is the coequalizer of ε_{FUA} and $FU\varepsilon_A$. By naturality of ε we have $\varepsilon_A \circ FU\varepsilon_A = \varepsilon_A \circ \varepsilon_{FUA}$, so the dashed arrow ε'_A in the diagram



exists. We show now that ε' is natural.

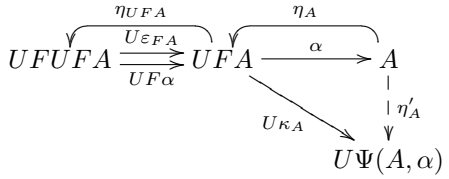


Using naturality of ε, we have

$$\begin{aligned} f \circ \varepsilon'_A \circ \kappa_{UA} &= f \circ \varepsilon_A \\ &= \varepsilon_B \circ FUf \\ &= \varepsilon'_B \circ \kappa_{UB} \circ FUf \\ &= \varepsilon'_B \circ \Psi\Phi f \circ \kappa_{UA} \end{aligned}$$

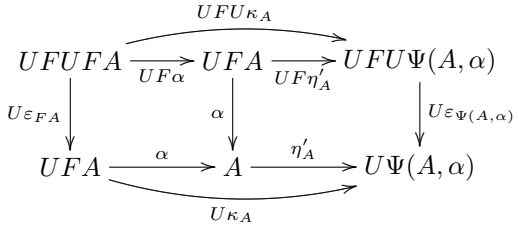
and so, since κ_{UA} is a regular epimorphism, $f \circ \varepsilon'_A = \varepsilon'_B \circ \Psi\Phi f$. Thus ε' is a natural transformation.

Let us now construct the unit $\eta' : \mathrm{Id}_{\mathcal{C}^T} \Rightarrow \Phi\Psi$. Fix some algebra (A, α) in $\mathcal{C}^T$ and consider the following diagram.



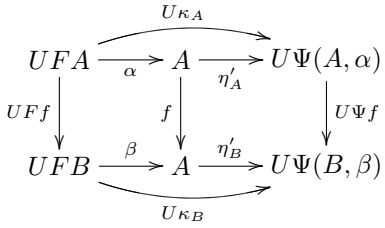
The top row, as was seen earlier, is a split coequalizer. Since $U\kappa_A$ equalizes $U\varepsilon_{FA}$ and $UF\alpha$ (because κ_A equalizes ε_{FA} and $F\alpha$ in $\mathcal{B}$), we get the unique arrow η'_A by the universal property of A. It remains to show that η'_A is in fact an arrow in $\mathcal{C}^T$; we must check that it commutes with the algebra structure maps. The left

square in the diagram



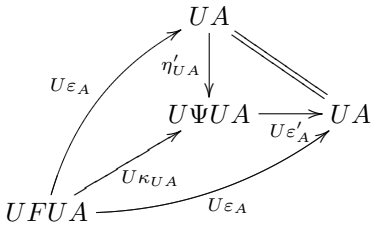
commutes because α is coequalizer of $U\varepsilon_{FA}$ and $UF\alpha$. The upper and lower triangles commute by definition of η'_A, and the outer rectangle commutes by naturality of ε. The right square will then be seen to commute because $UF\alpha$ is an epimorphism, which follows from the fact that α is a split epi (with right inverse η_A). Thus η'_A is a morphism of algebras.

We now show that the η' so defined is a natural transformation. The left square in the diagram



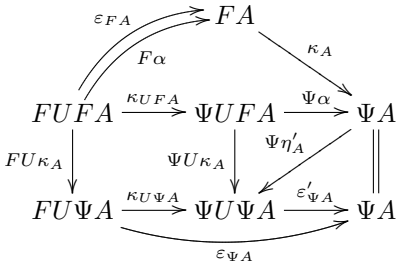
commutes because f is a morphism of algebras. The outer rectangle commutes by definition of Ψf, and the two triangles commute by definition of η'. Since α is split epic (with right inverse η_A) the right square also commutes, as wanted. Thus η' is a natural transformation.

It just remains to show that η' and ε' satisfy the triangle identities for an adjunction. To prove $\Phi\varepsilon' \circ \eta'\Phi = \mathrm{Id}$, recall that $\Phi A = (UA, U\varepsilon_A)$. The two outer triangles in the diagram



commute by definition of η' and ε', respectively. Thus the top right triangle commutes because $U\varepsilon_A$, as a structure map, is epic.

Proving $\varepsilon'\Psi \circ \Psi\eta' = \mathrm{Id}$ is slightly more complicated; it boils down to showing commutativity of the following diagram.

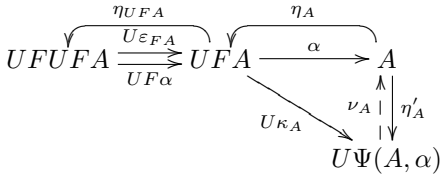


This can be seen to commute by applying the definitions of η', Ψ, and ε', naturality of ε, the definition of κ_A, and the fact that $\Psi\alpha \circ \kappa_{UFA}$ is epic (because α is an absolute epi and κ_{UFA} is a regular epi).

Therefore Ψ is left adjoint to Φ.

To prove the second claim of the theorem, assume now additionally that U preserves coequalizers of U-split pairs.

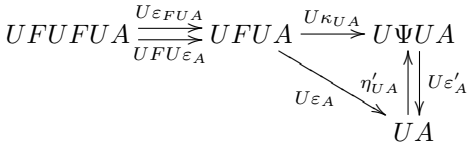
Let (A, α) in $\mathcal{C}^T$ be an algebra and consider the diagram:



Since κ_A is a coequalizer of the U-split pair ε_{FA} and $F\alpha$, $U\kappa_A$ is a coequalizer of $U\varepsilon_{FA}$ and $UF\alpha$; we thus get the unique arrow ν_A such that $\nu_A \circ U\kappa_A = \alpha$. But this means that $\nu_A \circ \eta'_A \circ \alpha = \alpha$, so $\nu_A \circ \eta'_A = \mathrm{id}_A$ since α is epic. Similarly, $\eta'_A \circ \nu_A \circ U\kappa_A = \eta'_A \circ \alpha = U\kappa_A$, so $\eta'_A \circ \nu_A = \mathrm{id}_{U\Psi(A,\alpha)}$ since $U\kappa_A$ is epic. Thus η'_A is an iso for every (A, α), and so η' is a natural isomorphism.

To prove the third part of the theorem, assume now additionally that U reflects isomorphisms.

Fix A in $\mathcal{B}$ and consider the diagram:



The structure map of the algebra $(UA, U\varepsilon_A)$ is $U\varepsilon_A$, so we get the induced η'_{UA} as above. We thus have $U\varepsilon'_A \circ \eta'_{UA} \circ U\varepsilon_A = U\varepsilon'_A \circ U\kappa_{UA} = U\varepsilon_A$, so $U\varepsilon'_A \circ \eta'_{UA} = \mathrm{id}_{UA}$ since $U\varepsilon_A$, as a structure map, is (split) epic. Similarly, $\eta'_{UA} \circ U\varepsilon'_A \circ U\kappa_{UA} = \eta'_{UA} \circ U\varepsilon_A = U\kappa_{UA}$, so $\eta'_{UA} \circ U\varepsilon'_A = \mathrm{id}_{U\Psi UA}$ since $U\kappa_{UA}$, as a coequalizer (due to our earlier assumption that U preserves coequlizers of U-split pairs), is epic. This says that $U\varepsilon'_A$ is an iso, and so ε'_A is also an iso by the hypothesis that U reflects isomorphisms. Thus ε' is a natural isomorphism.

Therefore Φ is an equivalence of categories and U is monadic. □

4. Groups are monadic

We will now use Beck's theorem to show that **Grp**, the category of groups and group homomorphisms, is monadic over **Set**. In other words, **Grp** can be thought of as the category of algebras of some monad on **Set**. This is a particularly simple application of Beck's theorem; the proof is essentially the familiar statement that groups can be defined as quotients of free groups. Nevertheless, it illustrates how Beck's theorem can give a general formal framework for such situations.

Note first that the underlying set functor $U : \mathbf{Grp} \to \mathbf{Set}$ has as left adjoint the free group functor F.

Theorem 4.1. *The underlying set functor $U : \mathbf{Grp} \to \mathbf{Set}$ is monadic.*

Proof. We check first that **Grp** has coequalizers of U-split pairs. In fact, **Grp** has arbitrary coequalizers. Let $f, g : A \rightrightarrows B$ be a pair of homomorphisms; their coequalizer is the canonical projection map π onto the quotient B/S, where S is the normal closure of the set $\{f(a)g(a)^{-1} : a \in A\}$. The projection can be seen to equalize f and g because for any a in A, $g(a) = f(a)f(a^{-1})g(a^{-1})^{-1} \in f(a)S$, so $g(a)S = f(a)S$. This satisfies the universal property of a coequalizer because if $\gamma : B \to C$ equalizes f and g, we must have $ker(\gamma)$ contained in S. The lattice isomorphism theorem can then be used to arrive at the unique map.

Suppose now that f and g form a U-split pair. Their coequalizer in **Set** is the quotient of UB by the equivalence relation generated by setting $b_1 R b_2$ if there is some $a \in A$ such that $f(a) = b_1$ and $g(a) = b_2$; that is, the quotient by the smallest equivalence relation which identifies $f(a)$ and $g(a)$. Call this coequalizer $p : B \to B/R$; then U will preserve coequalizers of U-split pairs if B/R is isomorphic to $U(B/S)$.

Since $U(B/S)$ equalizes Uf and Ug, we have a unique map $\pi_* : B/R \to U(B/S)$ such that $\pi_* \circ p = \pi$. We will show that π_* is an isomorphism. First, note that π_* must be the map which sends $[b]$ to bS in order for the coequalizer diagram to commute. Well-definition of this map can be checked by induction on the definition of R. In the base case, if $b_1 R b_2$ then there is some $a \in A$ such that $f(a) = b_1$ and $g(a) = b_2$. Thus $\pi_*([b_1]) = f(a)S$ and $\pi_*([b_2]) = g(a)S$, but these cosets are the same by definition of S. It is straightforward to check that well-definition also holds for pairs (b_1, b_2) added to the relation under the reflexive, symmetric, and transitive closures, because being in the same coset is itself an equivalence relation.

The map π_* is clearly surjective, because for all $bS \in U(B/S)$, $[b]$ is in B/R.

To check injectivity, suppose that $xS = yS$. Then there is some $a \in A$ and $b \in B$ such that $y = xb^{-1}f(a)g(a)^{-1}b$, so we must check that $xRxb^{-1}f(a)g(a)^{-1}b$. If there is some $a_0 \in A$ such that $f(a_0) = x$, this follows by letting $a = a_0^{-1}$ and $b = 1$. If neither x nor y is in the image of f, we can use transitivity to get xRy by showing $xRf(a)$ and $yRf(a)$ for some $f(a)$ such that $\pi(f(a)) = xS$. (We could use g instead of f, but without loss of generality we need check only one.) Once we have such an $f(a)$, $xRf(a)$ and $yRf(a)$ will be easy to show using the argument above. To prove such an $f(a)$ exists, we must show that $\pi \circ f$ is onto. The projection π is onto by definition, and f is onto because B/R is a split coequalizer, which means that there is a retract $t : B \to A$ with $f \circ t = \mathrm{id}_B$. Thus their composite is onto. This tells us that π_* is injective, and thus an isomorphism. Therefore U preserves coequalizers of U-split pairs.

The statement that U reflects isomorphisms is just the statement that a group homomorphism is an isomorphism if it is bijective.

Therefore, by Beck's theorem, **Grp** is monadic over **Set**. □

5. Monads as algebraic theories

An *algebraic theory* is the theory of a class of algebraic strucures characterized by one or more finitary operations defined everywhere and satisfying certain axioms expressed by equalities. Examples of structures whose theories are algebraic include groups, rings, modules, boolean algebras, and lattices. A *model* of a theory is then a particular structure satisfying the axioms of the theory. For example, the Klein four-group V is a model of the theory of groups. The general study of such theories is the subject of universal algebra. The theory of fields is an example of a theory which is not algebraic; it has an operation—multiplicative inverse—which is not defined everywhere.

In this section we will be concerned with categorical, as opposed to classical, universal algebra, and to that end we will use the following definition.

Definition 5.1. An *algebraic theory* $\mathcal{T}$ is a category $\mathcal{T}$ with a countable set $\{T^0, T^1, \ldots, T^n, \ldots\}$ of distinct objects such that each object T^n is the n-fold categorical product of T^1. A *model* of $\mathcal{T}$ is a functor $F : \mathcal{T} \to \mathbf{Set}$ which preserves finite products. A *homomorphism* of $\mathcal{T}$-models is a natural transformation.

The maps $T^n \to T^1$ are the n-ary operations. Presentations of algebraic theories in the classical sense determine algebraic theories in the categorical sense, and conversely any categorical algebraic theory is determined by such a presentation. As an example, if $\mathcal{G}$ is the theory of groups, there will be maps $e : T^0 \to T^1$, $i : T^1 \to T^1$, and $m : T^2 \to T^1$ in $\mathcal{G}$ giving the unit (considered as a nullary operation), inverse, and multiplication, respectively; these maps will satisfy equations for the group axioms. The model $V : \mathcal{G} \to \mathbf{Set}$ corresponding to the Klein four-group will take an object T^n to the n-fold cartesian product of the set $\{e, a, b, c\}$ of elements of the four-group, and the maps Ve, Vi, and Vm will be the familiar unit, inverse, and multiplication on the four-group.

Theorem 5.2. *Let* $\mathbf{Mod}_{\mathcal{T}}$ *be the category of models of an algebraic theory* $\mathcal{T}$*. Then* $\mathbf{Mod}_{\mathcal{T}}$ *is equivalent to the category of algebras of some finitary monad on* **Set**.

"Finitary" is a size condition whose precise definition need not concern us here. For a proof of this, see Borceux [2, Section 4.6.2]. The proof relies on machinery whose development would take us too far afield, but the fundamental idea is to use Beck's theorem to show that the forgetful functor $U : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}$ of evaluation at T^1 is monadic.

It is in this sense that algebraic theories are just monads. Such a result provides an appealing unified treatment of much of universal algebra, but what about theories which are not algebraic? Attempts have been made to generalize this notion to other classes of theories, such as "essentially algebraic" theories. The generalizations rely on higher category theory to resolve the coherence problems that arise.

Generalization in a different direction leads us to consider monads not on **Set**, but on **Cat**. This leads to the notion of equationally defined theories of *categories* with structure, as opposed to sets with structure. Examples include cartesian closed

categories, elementary toposes, and monoidal categories. As expected, we have in this case a result analogous to Theorem 5.2. The theory has been well established for some time now.

For more on monads as theories, see Lawvere [4] and Power [7].

6. State monads

This section will be of a somewhat different flavor than the rest of the paper; it presents an example of a class of monads that is particularly important in computer science. We will see thereby how even the most abstract concepts can find real applications.

We begin with a discussion of the problem. Traditional programming languages operate on data by reading from and writing to the computer memory. This memory can be thought of as a collection of blocks, each with an index, and each containing a unit of data. The contents of the memory at a given time are referred to as the *state* of the system, or the state of the program. Programming languages which allow modification of the memory are said to have *mutable state*. We can formalize the notion as follows, though the precise definition we use is unimportant.

Definition 6.1. A *state* is an element of 2^n for some fixed n; that is, it is a finite sequence of 1s and 0s. The space 2^n is called the *state space*.

The number n is the size of the memory in bits, and will usually be rather large. On modern personal computers at the time of this writing, n would be on the order of several billion.

Functional programming languages are languages whose programs are pure mathematical functions. That is, they take an input and produce an output. Programs are written by composing simple functions into more complicated ones. In particular, functional programs—insofar as they are purely functional—cannot produce side-effects, such as modifying the contents of memory or writing output to the screen. They can only produce a value within the language. Cognoscenti will recognize that such languages can be thought of as theories of the λ-calculus, or equivalently, of the combinator calculus. Functional languages have many advantages—being mathematical in nature, it is relatively easy to prove correctness of their programs—so we would like a way to mimic, in a functional manner, non-functional side-effects such as input/output and mutable state. I will focus on the issue of mutable state, but the ideas can be adapted to other types of side-effects as well.

As is well known, the category of functional languages—the category of λ-theories, to be precise—is equivalent to the category of small cartesian closed categories. This development can be found in Johnstone [3, Section D4.2] and elsewhere; I will only wave my hands at it here so that we can move into the categorical realm as quickly as possible. Given a small cartesian closed category $\mathcal{C}$, we can regard $\mathcal{C}$ as a language by letting morphisms $f : A \to B$ be programs which take an input of type A and produce an output of type B. The objects of $\mathcal{C}$ are the *types* (or datatypes) of the language. Logicians should take note that these are *not* the same types encountered in model theory. The types we are interested in here are also sometimes called *sorts*, but only by logicians and rarely by computer scientists (for whom "sort" is a verb). We require $\mathcal{C}$ to be cartesian closed so that our language can operate on tuples of data (product types) and on functions themselves (exponential types).

We can now use a monad on the category $\mathcal{C}$ to give a functional version of state. Monads were first introduced into functional programming by Eugenio Moggi in the early 1990s and the ideas were further developed by Philip Wadler. The idea is that while a functional program can't directly modify state, it can produce a value which encapsulates the state modifications to be made; such a value can then be applied at a convenient time. Monads (or more precisely, their algebras) provide the appropriate notion of encapsulation.

Definition 6.2. Let $\mathcal{C}$ be a small cartesian closed category and let S be an object of $\mathcal{C}$. The monad induced by the adjoint pair $(- \times S) \dashv (-)^S : \mathcal{C} \to \mathcal{C}$ is called a *state monad*. The object S is the *state type*.

To get the intuition, we will look at the details of this monad in the case where $\mathcal{C}$ is the essentially small category **FinSet** of finite sets. (Choose your favorite skeleton to get smallness.) We can let S be any finite set, but may as well think of it as 2^n. The functor part of the monad, $(- \times S)^S : \mathbf{FinSet} \to \mathbf{FinSet}$, takes a type A to the function type $(A \times S)^S$. The elements of this function type can be thought of as "state transition functions;" we shall call them "computations." A computation c takes a state s_1 and returns a pair (a, s_2) where a is a value of type A and s_2 is the new state.

The unit η at an object A will be the curried function which takes a value a and produces a computation which is a in the first coordinate and the identity on states in the second coordinate. We can write this as

$$\eta_A : a \mapsto (s \mapsto (a, s)).$$

We can do this in an element-free way for an arbitrary ccc by currying the identity map $\mathrm{id}_A \times \mathrm{id}_S$; that is, we take its image under the canonical bijection between $\mathrm{Hom}(A \times S, A \times S)$ and $\mathrm{Hom}(A, (A \times S)^S)$.

For the multiplication we want a map $\mu_A : ((A \times S)^S \times S)^S \to (A \times S)^S$. The counit of the adjunction is the evaluation arrow ε; thus by Theorem 2.5, we know that the multiplication is given by $\mu_A = \varepsilon^S_{A \times S}$. We think of multiplication as the process of chaining together two successive computations to get their composite. We can write this as

$$\mu_A : \Big(s_1 \mapsto \big((s_2 \mapsto (a, s_3)), s_2 \big) \Big) \mapsto \Big(s_1 \mapsto (a, s_3) \Big).$$

This allows the monad to build up a computation which represents all the aggregate state changes of the program. The resulting computation gives the same result a and the same final state as carrying out each subcomputation in succession would have. We need now only extract the value a to use it, and we can do so using the algebras of the state monad. An algebra $\alpha : (A \times S)^S \to A$ takes a computation $c : s_1 \mapsto (a, s_n)$ and returns a.

We will end with a quick word on the monadicity of $(-)^S$. First, some terminology.

Definition 6.3. If a category $\mathcal{C}$ has a terminal object $\top$ and A is any object of $\mathcal{C}$, a *global element* of A is a morphism $\top \to A$.

This generalizes the notion of set-theoretic element, since the elements of a set A correspond precisely to the functions $1 = \{0\} \to A$ which pick out individual elements.

Definition 6.4. A category $\mathcal{C}$ is *Cauchy complete* if every idempotent splits. That is, for every object A of $\mathcal{C}$ and morphism $e : A \to A$ such that $e \circ e = e$, there is an object B and morphisms $f : A \to B$ and $g : B \to A$ such that $g \circ f = e$ and $f \circ g = \mathrm{id}_B$.

Theorem 6.5. *Let $\mathcal{C}$ be a cartesian closed category. Then the exponential functor $(-)^S : \mathcal{C} \to \mathcal{C}$ is monadic if*

(1) S has a global element, and
(2) C is Cauchy complete.

In particular, **FinSet** is Cauchy complete (it is not a very strong condition), so as long as the state type S is nonempty, $(-)^S$ will be monadic. The proof of this theorem can be found in Mesablishvili [5]. This proof does not directly involve Beck's theorem; rather, it relies on the theory of separable functors and a slew of results about Cauchy complete categories. A corollary to this theorem is a result of Métayer [6] which says that $(-)^S$ is monadic provided S has a global element and $\mathcal{C}$ has a proper factorization system. It is not hard to prove that every Cauchy complete category has a proper factorization system.

Acknowledgments. It is a pleasure to thank my mentor, Daniel Schaeppi, for pointing me in the right direction numerous times and for his attention to detail in helping to edit this paper. I would also like to thank J. Peter May for making me aware, perhaps inadvertently, of several interesting aspects of monad theory.

References

[1] M. Barr and C. Wells, *Toposes, triples and theories*, Springer-Verlag, 1985.
[2] F. Borceux, *Handbook of categorical algebra 2: categories and structures*, Cambridge University Press, 1994.
[3] P. T. Johnstone, *Sketches of an elephant: a topos theory compendium, vol. II*, Clarendon Press, 2002.
[4] F. W. Lawvere, *Functorial semantics of algebraic theories*, Proc. Nat. Acad. Sci., 50, 1963, pp. 869-872.
[5] B. Mesablishvili, *Monads of effective descent type and comonadicity*, Th. and App. of Cats., 16, 1, 2006, pp. 1-45.
[6] F. Métayer, *State monads and their algebras*, arXiv:math/0407251v1 [math.CT], 2004.
[7] A. J. Power, *Why tricategories?*, Inf. and Comp., 120, 2, 1995, pp. 251-262.
[8] H. A. Priestley, *Ordered sets and complete lattices*, in Algebraic and Coalgebraic Methods in the Mathematics of Program Construction (R. Backhouse, R. Crole, and J. Gibbons, eds.), Springer, 2002, pp. 21-78.