The basic theory of monads and their connection to universal algebra

Anthony Voutas

May 2012

Abstract

This paper gives the definition of monads and outlines the key properties and theorems concerning how monads relate to adjunctions and universal algebra, giving the Eilenberg-Moore construction and showing that varieties can be studied with monads. It provides a summary of the key applications and potential of the monadic and comonadic view, including representing and reasoning about data structures, examining an alternative perspective on universal (co)algebra and exploring adjunctions in yet another different way. It also mentions some historical facts about the subject.

1 Introduction

From a basic point of view, category theory is a formal abstract theory of functions. However, with a little theoretical development, it provides a framework for defining and exploring mathematical concepts in general. It facilitates reasoning about objects at the appropriate level of abstraction by adding external constraints on the properties of objects instead of examining overly specific, complicated (internal) constructions of objects.

Mathematicians are more often concerned with the properties of objects, rather than their internal structure. This is not to say that internal structure is completely ignored by category theory. In fact, categorical concepts can help to reason about and elucidate the internal structure of many algebraic objects, the notion of category itself being a generalisation of monoids and partial orders.

The usefulness of category theory lies in its departure from defining new objects constructively. Instead, category theory defines objects by their unique behaviour (in the sense that all objects behaving similarly will be isomorphic – and thus interchangeable). Thus, category theory allows us to reason purely with properties, never having to examine the internals of objects. This is the source of category theory's power. Moreover, objects defined in this way don't necessarily exist in any given category. This allows category theory to operate at different levels of abstraction. We can consider categories in which a particular type of object always exists, but then we can always return to the more general case. This is not impossible to simulate in other languages of mathematics, but the language of category theory can make these distinctions natural. This is the source of category theory's versatility.

Category theory also allows the realisation of certain mathematical concept which go largely unnoticed by the rest of mathematics. Examples of adjunctions are ubiquitous in mathematics, but the general definition and exposition of adjunctions can only be seen through category theory. Even the existence of these relationships in particular circumstances usually remains unseen until category theory is applied. Monads are just as ubiquitous as adjunctions, as will be seen later, and similarly remain unseen until category theory is applied to the situations in question.

The first monad explicitly constructed was presented by Roger Godemont in 1958 [1]. Though, it

should be clarified that at the time, instead of using the name monad, he used "Standard Constructions", which indicates that they already existed without being explicitly generalised. This is true for much of basic category theory, and in fact at the time Godemont was writing about algebraic topology and sheaf theory, not explicitly category theory. This shouldn't be too surprising, as much of the foundational work in category theory was done to provide tools for either algebraic topologists or universal algebraists.

The concept of a monad has gone by a few other names over the years. Triple, triad, standard construction, fundamental construction and monad have all been used at one point or another. Both the terms standard construction and fundamental construction are vague and not particularly unique, the term triple is even less unique, triad means more or less the same as triple, and monad is a pre-existing term in the English language meaning more or less "a single unit", and is used in philosophy to refer to an indivisible entity, among other things.

A monad is essentially a monoid in an endofunctor category, and because of this the name monad is somewhat of a pun on the word monoid. However, 'monad' is highly misleading terminology as we will see that a monad can be 'split' by any of the adjunctions which can be used generate it, and moreover a monad is constructed from three other objects. An entity built from three entities and able to split into four entities (an adjunction is made up from four objects) is hardly monadic in the philosophical sense.

There was a time when all the names were used, but the convention has been settled and the name monad now predominates the literature. The popularity of the term monad is probably due to Saunders Mac Lane, via his prominent and influential book Category Theory for the Working Mathematician [2]. While discussing monads in the book, he provides justification for abandoning the term triple.

On a more technical note, monads have been used to provide and explore a different view of universal algebra. This approach is supported by the fact that varieties can be represented via monads. This is a result of the following connections which will be explored more thoroughly later in this report.

An adjunction between categories $\mathcal{A}$ and $\mathcal{B}$ gives rise to a monad $\mathbb{T}$ on $\mathcal{A}$. The monad $\mathbb{T}$ can then be used to construct an adjunction between $\mathcal{A}$ and a generalised category of algebras $\mathcal{A}^{\mathbb{T}}$ (The Eilenberg-Moore construction). There is then a comparison functor from the original category $\mathcal{B}$ to $\mathcal{A}^{\mathbb{T}}$, and this is an isomorphism if a particular property is satisfied (call it X for now).

Varieties admit free constructions (that is an adjunction between a free functor and a forgetful functor). Starting with a the free-forgetful adjunction for a variety, one can form the monad and then the Eilenberg-Moore adjunction. Then the property X is satisfied. In this way we can construct an isomorphism between the category of the variety and the category of algebras for the monad for that variety. The notion of an algebra of the variety can now be captured by the notion of an algebra for the monad.

Thus, because universal algebra is essentially the study of varieties, monads are capable of representing the objects of study of universal algebra. Moreover, it can be useful to look at monads in order to prove general theorems about universal algebra, because monads admit many nice properties. For starters, the monads for varieties exist entirely in the category of sets and functions, which is a very well understood category, whereas arbitrary variety categories can be unusual. Moreover, monads are defined in a categorical way, which allows many different things to be considered which aren't as obvious in the algebraic signature view, such as morphisms between varieties.

Monads provide a general definition for free constructions which is both general and simple.

Meanwhile, a particular free construction can be very complicated and messy. These characteristics make monads good for use in general proofs about free constructions, which takes some of the burden off those working with particularly complicated free constructions.

Monads are also very useful in the context of functional programming. They can be used to reason about and represent recursive data structures, and even provide a way of constructing a notion of structural induction for the defined data structures (via the initial algebra for the monad). Being able to define structural induction on a data structure is key to proving the correctness of recursive functions operating on that data structure.

Monads are used in functional programming languages to provide a general interface for the creation and use of data structures. Polymorphic functions can be written to be applied to any monad. Since monads can be used to represent basically any recursive data structure, polymorphic functions are extremely general when applied to the type-class of monads.

The concept of monad has been generalised to that of an *arrow* [3] in functional programming languages like Haskell, in order to provide more flexibility in the creation of data structures. Such generalisations require detailed knowledge of the concept of a monad, its strengths and its weaknesses. Monads have a slew of applications in the functional programming language Haskell [4], including simplifying program code [3] and formalising side-effect free parallelism [5].

Furthermore, in category theory, the existence of a concept automatically produces the existence of a dual concept. In the case of monads, that concept is a comonad. The properties of a comonad are directly dual to that of a mona. So, comonads are also generated by an adjunction and are useful for examining coalgebra. It so happens that coalgebra is an extremely useful way of examining state-transition systems [6], and there is a deep connection between coalgebra and modal logic [7] which is analogous to the connection between algebra and equational logic.

This connection is already well established and explored in the literature, but as coalgebraic modal logic is a relatively new subfield there are currently many opportunities for its application to be extended and for the surrounding concepts to be further explored in general.

The concept of monad extends beyond algebra though, since it is directly connected to the concept of adjunctions, which are ubiquitous in mathematics and theoretical computer science. A monad can provide a different view of any adjunction, essentially the view of the adjunction taken inside one of the categories (with a comonad being in the other). Examining these monads could lead to new insights in many fields and could be easier to manage than the adjunctions themselves.

With regards to universal algebra, Lawvere theories [8] are similar to monads, in that they are both categorical formalisms for reasoning about algebraic varieties. Monads can be criticised for obscuring the original equations of the varieties, while Lawvere theories present the equations more prominently.

It so happens that every Lawvere theory has a corresponding monad on **Set**, the converse being true only for a generalisation of Lawvere theories to arities of arbitrary size [9]. So, in addition to displaying equations more prominently, Lawvere theories can make a clearer distinction between finitary algebra and infinitary algebra, whereas monads rely on a more subtle notion of rank to represent this fact.

With all this in mind, obfuscation of the equations might be useful or required in practice. Certainly, in functional programming applications hiding the internal structure of the monad is intended and useful in practice [3]. In addition, the theory of monads has been thoroughly developed so it still has useful applications in universal algebra, even if in principle the concept of Lawvere theory might be more useful. Furthermore, the development of theory surrounding monads might

be evidence that it is an easier concept to work with in the abstract, and this is supported when examining the development of Lawvere theories for arbitrary categories [9, §5]. All in all, this does not diminish the applicability of monads to functional programming or to examining adjunctions in general.

The theory contained in this report is entirely standard, and there are no original results. This is merely a restating of work that is widely available in standard textbooks on these issues [2] [10] [1], with the exception of the discussion in Section 5, which is no more original, but slightly less prominent. This report has the following structure. Section 1 has been the introduction, informally covering some of the theoretical notions, outlining applications, and reviewing the history of the concept. In Section 2, some necessary (but slightly less than basic) category theory notions are presented to fix notation and prepare the reader for the rest of the report. Also in Section 2, two concepts of algebra are given, the standard universal algebra notion of a variety, and the notion of an algebra for a functor. Then in Section 3 the connection between monads and adjunctions are explored, and we show that a monad can be built from an adjunction, and that a monad can be split into an adjunction. In Section 4 the connection between monads and universal algebra (via free-faithful adjunctions) is explored, and we prove that all varieties have a corresponding monad, and that the Eilenberg-Moore category for that monad is isomorphic to the variety (when taken as a category). In section 5 the dual notion of a comonad is stated and some applications of coalgebra are given. Finally, and briefly, some avenues for future research are presented.

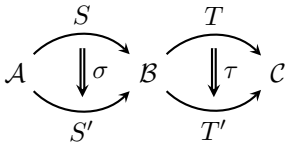
2 Preliminary Details

Before launching into the technical details of what monads are and how they relate to adjunctions and algebra, it is necessary to establish a few definitions and facts from category theory and universal algebra.

2.1 Horizontal composition of natural transformations

Recall that natural transformations can be composed in two different ways. Firstly they can be composed vertically by composing their components. A slightly less utilised composition is the horizontal composition, defined as follows [2, §II.5]:

Suppose we have the following functors and natural transformations:



Then we can form the composite functors TS, $T'S$, TS' and $T'S'$ all from $\mathcal{A}$ to $\mathcal{C}$, and we can construct a diagram (for $X \in \mathcal{A}$):

$$\begin{array}{ccc} TS(X) & \xrightarrow{\tau_{S(X)}} & T'S(X) \\ {\scriptstyle T(\sigma_X)}\downarrow & & \downarrow{\scriptstyle T'(\sigma_X)} \\ TS'(X) & \xrightarrow[\tau_{S'(X)}]{} & T'S'(X) \end{array}$$

This diagram commutes because it is the naturality diagram for τ with σ_X substituted for the general arrow. We can now define the diagonal of this square to be the horizontal composite $\tau\sigma$. Note that in this document the notation for vertical composition will be $\alpha \circ \beta$, while the notation for horizontal composition will be $\alpha\beta$ (obviously it is not possible to compose any two natural transformations, and natural transformations that can be composed horizontally cannot necessarily be composed vertically or vice versa. We will later be using natural transformations between endofunctors, which can always be composed horizontally).

In summary:

Definition 2.1.1. The natural transformation which is the horizontal composition of the two natural transformations $\tau : T \dot{\to} T'$ and $\sigma : S \dot{\to} S'$ is given by the components:

$$(\tau\sigma)_X = T'(\sigma_X) \circ \tau_{S(X)} = \tau_{S'(X)} \circ T(\sigma_X) \tag{1}$$

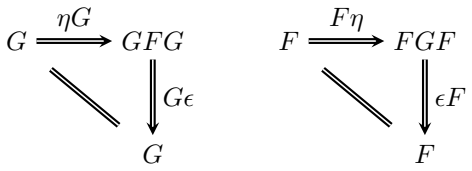
This is always well defined if the codomain of S and S' matches the domain of T and T'.

2.2 Triangle identities for an adjunction

The triangle identities for an adjunction is a convenient equivalent definition for adjunction which doesn't refer explicitly to universal arrows or bijection of homsets [2, §IV.1 Theorem 2].

Definition 2.2.1. [Triangle identities for an adjunction] Given that $\mathcal{A}$ and $\mathcal{B}$ are categories, an adjunction is a tuple $\langle F, G, \eta, \epsilon \rangle$, where F and G are functors $F : \mathcal{A} \to \mathcal{B}$, $G : \mathcal{B} \to \mathcal{A}$, and η and ϵ

are natural transformations $\eta : 1_{\mathcal{A}} \dot{\to} GF$, $\epsilon : FG \dot{\to} 1_{\mathcal{B}}$ such that the following diagrams commute:

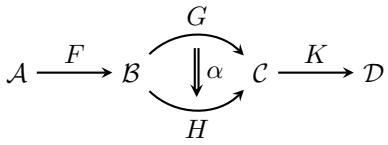


2.3 An aside about whiskering

Before going on to define a monad, it may be helpful to reflect on some facts about natural transformations. We will restrict our interests to particular types of natural transformations, for reasons which will become clear in the next section.

Using horizontal composition of natural transformations we can create a new natural transformation by *whiskering* a natural transformation by a functor [11]. Whiskering a natural transformation α by a functor F is done by horizontally composing α and the natural transformation 1_F.

In general, given the following diagram of functors F, G, H, K and natural transformation α:



We can form the following natural transformations: ($1_K\alpha$ by horizontal composition)

$$K\alpha : KG \dot{\to} KH$$
$$(K\alpha)_B : KG(B) \to KH(B)$$
$$(K\alpha)_B = K(\alpha_B)$$

and ($\alpha 1_F$ by horizontal composition)

$$\alpha F : GF \dot{\to} HF$$
$$(\alpha F)_A : GF(A) \to HF(A)$$
$$(\alpha F)_A = \alpha_{F(A)}$$

Note that for an endofunctor T, we will take $T^2 := T \circ T$ and $T^3 := T \circ T \circ T$. Now with T being an endofunctor on $\mathcal{C}$, 1_T being the identity natural transformation on T and $\eta : 1_{\mathcal{C}} \dot{\to} T$ and $\mu : T^2 \dot{\to} T$ being natural transformations, we can make the following observations (from the definition of whiskering just given).

For $\mu_X : T^2(X) \to T(X)$ we have components $(\mu T)_X = \mu_{T(X)} : T^3(X) \to T^2(X)$

For $\mu_X : T^2(X) \to T(X)$ we have components $(T\mu)_X = T(\mu_X) : T^3(X) \to T^2(X)$

For $\eta_X : X \to T(X)$ we have components $(\eta T)_X = \eta_{T(X)} : T(X) \to T^2(X)$

For $\eta_X : X \to T(X)$ we have components $(T\eta)_X = T(\eta_X) : T(X) \to T^2(X)$

2.4 Signatures and Ω-Algebras

Here we use the presentation and notations of [12, Example 1.1.7].

At this point we ask an important question. Precisely what is an Algebra? The universal algebraist will give an answer defining the concepts of *theory* and *variety* below.

One example of an algebra is found in group theory. A group consists of a set S, operations $(\circ, (-)^{-1}, e)$ — a binary operation $\circ$, an unary operation $(-)^{-1}$ and a constant e (nullary operation) — and a set of equations which identify terms built from variables and these operations. Informally (for now) a group is an algebra for the theory for groups, which is the set of operation symbols and equations making up the group axioms.

To formalise this in general, recognise that the group theory is composed of the group operations and the group equations. First we will consider the operations without added equations.

Let Ω be a set of operation symbols (a different symbol for each operation you intend to have in your algebra - groups have three, rings have five, etc). Now let $ar : \Omega \to \mathbb{N}$, be a function which assigns to $\omega \in \Omega$ the *arity* of the operation which corresponds to ω (that is, the number of arguments the operation takes - often two or less, but this generalises to any natural number). A *signature* is comprised of Ω and ar together, and is often just referred to as Ω.

With a signature Ω, we can form Ω-*algebras*.

Definition 2.4.1. An Ω-*algebra* A is comprised of a 'carrier' for the algebra and an 'interpretation' for every operation symbol. The carrier of A is a set (often written $|A|$). The interpretation of $\omega \in \Omega$ is a function $a_\omega : |A|^{ar(w)} \to |A|$.

Note that the interpretation maps $ar(\omega)$-tuples of elements of the carrier, back into the carrier, as you would expect of an operation.

As with groups, we can define the notion of a Ω-homomorphism from Ω-algebra A to Ω-algebra B.

Definition 2.4.2. A Ω-homomorphism is a function $f : |A| \to |B|$ with the following property:

$$f(a_\omega(x_1, x_2, ..., x_{ar(\omega)})) = b_\omega(f(x_1), f(x_2), ..., f(x_{ar(\omega)}))$$

For all $ar(\omega)$-tuples of $x_i \in |A|$.

In other words the Ω-homomorphisms preserve the actions of the operations in the algebra. Note that this coincides with the definition of group homomorphism in that if $f : G \to H$ is a group homomorphism, then $f(x \circ_G y) = f(x) \circ_H f(y)$ (with the operation written in infix notation here), $f(x^{-1}) = (f(x))^{-1}$ and $f(e_G) = e_H$. In fact this general definition corresponds to many standard definitions of homomorphisms (such as for rings, monoids, lattices, etc).

This property is compositional and satisfied for identities, and so we have a category of Ω-algebras and Ω-homomorphisms, called Ω-**Alg**.

In order to consider the addition of equations, it is necessary to define Ω-*terms*. An Ω-term is either a variable, a constant, or an operation applied to the correct number of terms. An Ω-equation is a pair of Ω-terms. In order for a Ω-algebra A to satisfy an Ω-equation, the two terms of the Ω-equation must evaluate (via the application of the A-interpretations of the operation symbols) to the same carrier value under any consistent substitution of carrier values for the variables in the terms.

The signature can be augmented with a set of Ω-equations and the algebras cut down to only those that satisfy those equations. The addition of this set of equations gets us to a formal notion of variety which matches a great many intuitively algebraic constructions. Thus,

Definition 2.4.3. An algebraic theory is a pair (Ω, E) where Ω is a signature and E is a set of Ω-equations.

Thus the algebras for a theory (Ω, E) are a full subcategory of the original Ω-**Alg**. This subcategory (Ω, E)-**Alg** is called the variety, since it is a particular 'variety of algebras'. In universal algebra it is not necessary to acknowledge that a variety has arrows, it is usually taken to simply be the set/class of objects (that is, the set of algebras for the theory).

Leaving aside these added equations is convenient for now. Without equations, an Ω-Algebra is essentially a carrier $|A|$ and a single function:

$$a : (\sum_{\omega \in \Omega} |A|^{ar(\omega)}) \to |A| \tag{2}$$

In which Σ is the disjoint union operator. The domain is the set of terms which consist of a single operation applied to elements of $|A|$ and the function evaluates those terms in $|A|$.

2.5 Functors and F-Algebras

Here we use the presentation and notations of [12, §2.2].

We will now consider an even more general definition of algebra, that of the algebra for a functor. This definition will help us to work with algebras using convenient categorical insights.

Take an endofunctor $F : \mathcal{C} \to \mathcal{C}$.

Definition 2.5.1. An algebra for F (an F-algebra) is a pair (X, h) where X is an object of $\mathcal{C}$ and $h : F(X) \to X$ is an arrow of $\mathcal{C}$.

Definition 2.5.2. An F-homomorphism between an F-algebra (A, a) and an F-algebra (B, b) is a $\mathcal{C}$ arrow $f : A \to B$ such that the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array}$$

We will now show that this definition captures the original set-theoretic one. Given a signature Ω we can define a functor $F : \mathbf{Set} \to \mathbf{Set}$ (for sets X)

$$F(X) = \sum_{\omega \in \Omega} X^{ar(\omega)}$$

That is, $F(X)$ is the domain of the evaluation function given in equation (2).

Now for arrows $g : X \to X'$ and operations ω interpreted in $F(X)$ as ω_X and in $F(X')$ as $\omega_{X'}$, we have $F(g) : F(X) \to F(X')$

$$(F(g))(\omega_X(x_1, x_2, ..., x_{ar(\omega)})) = (\omega_{X'}(g(x_1), g(x_2), ..., g(x_{ar(\omega))}))$$

Then it is clear that F is a functor. Using the above definition, an F-algebra is a set X and a function

$$\sum_{\omega \in \Omega} X^{ar(\omega)} \xrightarrow{h} X$$

Which is what we concluded an Ω-algebra was.

This definition captures the original notion, but is more general. As it turns out, F-algebras are Ω-algebras for set based, polynomial functors F [10, Chpt 10], that is functors $F : \mathbf{Set} \to \mathbf{Set}$ such that:

$$F(X) = \sum_{n=0}^{N} c_n X^n$$
$$F(f)(x_1, x_2, ..., x_n) = (f(x_1), f(x_2), ..., f(x_n))$$

3 Monads and adjunctions

Some of the notation and technicalities are taken from [10, Chpt 10], and a full treatment of this subject can be found in [2, §VI].

3.1 What is a Monad?

In generalising the definition of algebra, we have lost some of the useful structure on $F(X)$ - namely the intuitive connection with terms. It seems that we may have generalised too far. Fortunately, there is a way to reintroduce this intuition categorically.

Definition 3.1.1 (Monad)**.** A Monad is a triple $\mathbb{T} = \langle T, \mu, \eta \rangle$ such that $T : \mathcal{C} \to \mathcal{C}$ is an endofunctor, μ and η are natural transformations

$$\mu : T^2 \dot{\to} T, \qquad \eta : 1_{\mathcal{C}} \dot{\to} T$$

and the following are commutative diagrams

$$\begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array} \qquad \begin{array}{ccccc} 1_{\mathcal{C}}T & \xrightarrow{\eta T} & T^2 & \xleftarrow{T\eta} & T1_{\mathcal{C}} \\ & \searrow & \downarrow{\scriptstyle \mu} & \swarrow & \\ & & T & & \end{array} \tag{3}$$

That is, $\mu \circ T\mu = \mu \circ \mu T$ and $\mu \circ T\eta = 1_T = \mu \circ \eta T$.

If the functor T is to be interpreted as an algebra type (as we will in section 4), then μ is a transformation which encodes how terms of terms are collapsed down to terms. The square property above says that terms of terms of terms can be collapsed to terms unambiguously. Furthermore, η is a transformation which encodes how single elements of the carrier are turned into terms.

Example 3.1.2. Perhaps the simplest example of a monad is the monad for lists.

The monad $\mathbb{T} = \langle T, \mu, \eta \rangle$ is defined on **Set** as follows. The functor T takes a set X to the set of finite lists of elements of the set X, and for functions f and lists $[x_1, x_2, ..., x_n] \in TX$:

$$T(f)([x_1, x_2, ..., x_n]) = [f(x_1), f(x_2), ..., f(x_n)]$$

$\mu_X : T^2(X) \to T(X)$ is a flattening in which:

$$\begin{aligned} &\Big[[x_{1,1}, x_{1,2}, ..., x_{1,n}], [x_{2,1}, x_{2,2}, ..., x_{2,n}], ..., [x_{m,1}, x_{m,2}, ..., x_{m,n}]\Big] \\ &\mapsto \Big[x_{1,1}, x_{1,2}, ..., x_{1,n}, x_{2,1}, x_{2,2}, ..., x_{2,n}, ..., x_{m,1}, x_{m,2}, ..., x_{m,n}\Big] \end{aligned}$$

and $\eta_X : X \to T(X)$ is an injection $x_1 \mapsto [x_1]$

In this case the monad rules correspond to the claims that:

1. There is an unambiguous way of flattening a list of list of lists into a list.
2. Injecting a list into a singleton list of that list and flattening will return the first list.
3. Injecting each element of a list into a singleton list and flattening will return the first list.

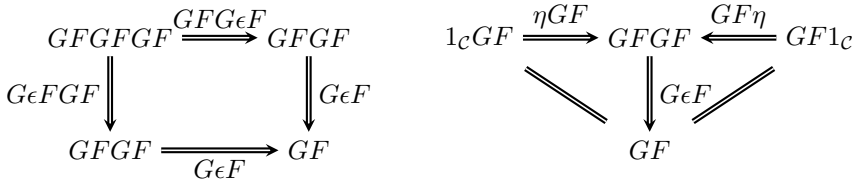
These facts should be clear on reflection of a few examples, though a general proof is tedious.

3.2 Adjunctions give rise to Monads

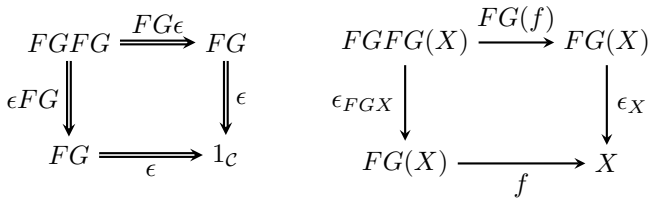
Theorem 3.2.1. *[Adjunctions to Monads] Given an Adjunction (F, G, η, ϵ) with $F \dashv G$, we can always construct a monad in the category $\mathcal{C} = dom(F)$ as follows.*

$$T : \mathcal{C} \to \mathcal{C} = GF : \mathcal{C} \to \mathcal{C}$$
$$\eta : 1_{\mathcal{C}} \dot{\to} T = \eta : 1_{\mathcal{C}} \dot{\to} GF$$
$$\mu : T^2 \dot{\to} T = G\epsilon F : GFGF \dot{\to} GF$$

Proof. Making the above substitutions, it remains to show the following diagrams commute:



The commutativity of square diagram can be shown by examining the following diagrams



(4)

If the diagram on the left commutes, then it commutes for all $\mathcal{C}$ objects, including all $F(X)$, and its commutativity is preserved by applying the functor G to all objects and arrows. Thus if it commutes, the original square diagram above commutes.

The diagram on the right of (4) commutes by the fact that ϵ is a natural transformation. It is clear that the components of the diagram on the left are actually special cases of the one on the right where $f = \epsilon_X$

It remains to show that the triangles $G\epsilon F \circ GF\eta = 1_{GF} = G\epsilon F \circ \eta GF$ commute. As it turns out, these are specific cases of the triangle identities for an adjunction (definition 2.2.1). First take the identity $\epsilon F \circ F\eta = 1_F$ under the action of G, then take the identity $G\epsilon \circ \eta G = 1_G$ for all objects $F(X)$. □

Example 3.2.2. We will now show the free-forgetful adjunction for monoids giving rise to the monad for lists seen in example 3.1.2. First we must define this adjunction.

The free functor F : **Set** $\to$ **Mon** is defined as follows. For sets X, $F(X)$ is the monoid of strings of elements of X. The monoid multiplication is concatenation and the unit is the empty string. Elements of $F(X)$ will be represented $[x_1, ..., x_n]$.

For functions f, the functor F acts as follows

$$Ff([x_1, ..., x_n]) = [f(x_1), ..., f(x_n)]$$

The forgetful functor U : **Mon** $\to$ **Set** is defined as follows. For monoids M, UM is the underlying set of M, and for monoid homomorphisms $h : M \to N$, $Uh : UM \to UN$ is just the function h.

The natural transformation $\eta : 1_{\mathbf{Set}} \dot{\to} UF$ is defined by the components $\eta_X(x) = [x]$, while

the $\epsilon : FU \dot{\to} 1_{\mathbf{Mon}}$ is defined by components $\epsilon_M([m_1, ..., m_n]) = m_1 \circ_M ... \circ_M m_n$.

We will take for granted that this is an adjunction. Then the monad generated is $\langle UF, \eta, U\epsilon F\rangle$. UF sends a set X to the set of lists of elements of X, the components η_X inject elements of X into singleton lists, and the components $U\epsilon F_X : UFUF(X) \to UF(X)$ flatten lists of lists to lists. This is the monad from example 3.1.2.

3.3 Algebras for a Monad

Theorem 3.2.1 begs the question: given a monad, can one split it into two adjoint functors? The answer is yes. In general there are many splittings for a monad. There are two important splitting constructions: The Eilenberg-Moore construction [13], and the Kleisli construction [14]. In fact, it so happens that the collection of splittings for a monad form a category in which the Eilenberg-Moore adjunction is terminal and the Kleisli adjunction is initial. The proof of this is beyond the scope of this report, but can be found in [2, §VI]. What will be proved is that the Eilenberg-Moore construction yields a splitting of any monad.

We must first define the Eilenberg-Moore category $\mathcal{C}^{\mathbb{T}}$ of a monad $\mathbb{T} = \langle T, \mu, \eta\rangle$. This category consists of $\mathbb{T}$-algebras and $\mathbb{T}$-morphisms (this is an extension of the concept of an F-algebra for an endofunctor F).

Definition 3.3.1. Objects of The Eilenberg-Moore category $\mathcal{C}^{\mathbb{T}}$ (the $\mathbb{T}$-algebras) will be the T-algebras (A, a) (see definition 2.5.1) such that

$$a \circ Ta = a \circ \mu_A \qquad \text{and} \qquad a \circ \eta_A = 1_A$$

That is, the following diagrams commute:

$$\begin{array}{ccc} T^2A & \xrightarrow{Ta} & TA \\ {\scriptstyle \mu_A}\downarrow & & \downarrow{\scriptstyle a} \\ TA & \xrightarrow[a]{} & A \end{array} \qquad\qquad \begin{array}{ccc} A & \xrightarrow{\eta_A} & TA \\ & \searrow & \downarrow{\scriptstyle a} \\ & & A \end{array} \tag{5}$$

Informally, one can consider TA as an object of terms over values in A, and a as an evaluation which picks out a value in A for each term in TA. Then the square above corresponds to the notion that evaluating a term of terms by applying two evaluations is equivalent to collapsing the term of terms into an term and then evaluating it. The triangle corresponds to the notion that singleton terms evaluate to the value the term is built from. That is, a $\mathbb{T}$-algebra is a T-algebra for which μ acts as a term flattener and η acts as an injection into a singleton term.

Definition 3.3.2. The morphisms of $\mathbb{T}$-algebras $f : (A, a) \to (B, b)$ are arrows $f : A \to B$ such that $f \circ a = b \circ T(f)$, that is:

$$\begin{array}{ccc} T(A) & \xrightarrow{T(f)} & T(B) \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array}$$

That is, the normal notion of T-homomorphisms. This makes $\mathcal{C}^{\mathbb{T}}$ a full subcategory of the category of T-Algebras.

Example 3.3.3. Formally, algebras for the list monad $\langle T, \eta, \mu \rangle$ (from example 3.1.2) are pairs $(A, a : TA \to A)$, such that the diagrams (5) are satisfied.

The informal discussion above can be instantiated precisely in this situation, in that the term set TA is the set of lists of elements of A, and the evaluation a is a function which picks out a value from A for any given list. The properties which this function must obey from (5) are:

1. Singleton lists must be evaluated as the value they contain.
2. The evaluation a is associative, in that we can evaluate a list by parts, forming a new list which when evaluated yields the same value in A as if evaluating the list as a whole.

These are essentially the requirements of a monoid, as such an evaluation can be entirely captured by the evaluations on lists of length two. The evaluation of longer lists can be recovered from repeated application of this binary evaluation. The evaluation is fixed on singletons. The empty list must evaluate to an identity for the binary evaluation, because every list can be taken to be the concatenation of itself with the empty list, and it is given that the evaluation of the parts matches the evaluation of the whole.

So, the collection of algebras for the list monad are the collection of all monoids.

3.4 Monads give rise to Adjunctions

Theorem 3.4.1. *Given a monad $\mathbb{T} = \langle T, \mu, \eta \rangle$ on $\mathcal{C}$ we can construct an adjunction between $\mathcal{C}$ and $\mathcal{C}^{\mathbb{T}}$.*

The adjunction will be defined by functors U (forgetful)

$$\begin{aligned} U : \mathcal{C}^{\mathbb{T}} &\to \mathcal{C} \\ U(A, a) &= A \\ U(f : (A, a) \to (B, b)) &= f : A \to B \end{aligned}$$

and F (free)

$$\begin{aligned} F : \mathcal{C} &\to \mathcal{C}^{\mathbb{T}} \\ F(A) &= (TA, \mu_A) \\ F(f : A \to B) &= T(f) : (TA, \mu_A) \to (TB, \mu_B) \end{aligned}$$

Proof. U is clearly a functor, so we will first show that F is a functor, and then show that $F \dashv U$.

Firstly, we should check that (TA, μ_A) is a $\mathbb{T}$-algebra, by examining the following diagrams:

$$\begin{array}{ccc} T^2(TA) & \xrightarrow{T(\mu_A)} & T(TA) \\ {\scriptstyle \mu_{TA}}\downarrow & & \downarrow{\scriptstyle \mu_A} \\ T(TA) & \xrightarrow[\mu_A]{} & (TA) \end{array} \qquad \begin{array}{ccc} (TA) & \xrightarrow{\eta_{TA}} & T(TA) \\ & \searrow & \downarrow{\scriptstyle \mu_A} \\ & & (TA) \end{array}$$

These are cases of:

$$\begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array} \qquad \begin{array}{ccc} T & \xrightarrow{\eta T} & T^2 \\ & \searrow & \downarrow{\scriptstyle \mu} \\ & & T \end{array}$$

Which are clearly commutative because $\mathbb{T} = \langle T, \mu, \eta \rangle$ is a monad.

Now we check that $T(f)$ is a $\mathbb{T}$-morphism by examining this diagram:

$$\begin{array}{ccc} T^2(A) & \xrightarrow{T^2(f)} & T^2(B) \\ {\scriptstyle \mu_A}\downarrow & & \downarrow{\scriptstyle \mu_B} \\ T(A) & \xrightarrow[T(f)]{} & T(B) \end{array}$$

This clearly commutes for any f because μ is a natural transformation, and this is exactly the natural transformation diagram for μ.

So, we now know that F and U are functors it now remains to show that they are adjoint (specifically, $F \dashv U$).

Notice that $T = UF$.

$$\begin{aligned} UF(A) &= U(TA, \mu_A) \\ &= TA \\ UF(f : A \to B) &= U(Tf : (TA, \mu_A) \to (TB, \mu_B)) \\ &= Tf : TA \to TB \end{aligned}$$

We will show $F \dashv U$ by the triangle identities for adjunctions. First we need to define both the unit $\eta : 1_{\mathcal{C}} \to UF$ and the counit $\epsilon : FU \to 1_{\mathcal{C}^{\mathbb{T}}}$. The notation suggests that the unit $\eta : 1_{\mathcal{C}} \to UF$ can be taken to be the $\eta : 1_{\mathcal{C}} \to T$ of the monad, and this matches the required form.

To establish the counit $\epsilon : FU \to 1_{\mathcal{C}^{\mathbb{T}}}$, we examine the action of the functor $FU : \mathcal{C}^{\mathbb{T}} \to \mathcal{C}^{\mathbb{T}}$.

$$\begin{aligned} FU(A, a) &= F(A) \\ &= (TA, \mu_A) \\ FU(f : (A, a) \to (B, b)) &= F(f : A \to B) \\ &= Tf : (TA, \mu_A) \to (TB, \mu_B) \end{aligned}$$

Let the components of the counit be $\epsilon_{(A,a)} = a$ as a $\mathbb{T}$-morphism from (TA, μ_A) to (A, a):

$$\begin{array}{ccc} T^2(A) & \xrightarrow{T(a)} & T(A) \\ {\scriptstyle \mu_A}\downarrow & & \downarrow{\scriptstyle a} \\ T(A) & \xrightarrow[a]{} & A \end{array}$$

To be a $\mathbb{T}$-morphism, the above diagram must commute, which it does because (A, a) is a $\mathbb{T}$-algebra. For ϵ to be natural, we must show that $f \circ \epsilon_{(A,a)} = \epsilon_{(B,b)} \circ T(f)$ for all $f : (A, a) \to (B, b)$. Since f is a $\mathbb{T}$-morphism, $f \circ a = b \circ T(f)$, which is precisely the property we want.

Now, the triangle identities for the adjunction can be seen as follows:

$$\begin{array}{ccc} G(A,a) & \xrightarrow{1} & G(A,a) \\ & {\scriptstyle \eta_{G(A,a)}} \searrow \quad \nearrow {\scriptstyle G\epsilon_{(A,a)}} & \\ & GFG(A,a) & \end{array} \qquad \text{becomes} \qquad \begin{array}{ccc} A & \xrightarrow{1} & A \\ & {\scriptstyle \eta_A} \searrow \quad \nearrow {\scriptstyle a} & \\ & TA & \end{array} \tag{6}$$

(7)

(6) commutes because (A, a) is a $\mathbb{T}$-algebra (see triangle diagram in (5)), and (7) commutes because $\mathbb{T}$ is a monad (see η diagram in (3)).

□

Example 3.4.2. We will now show the adjunction which arises from the list monad.

If the monad is $\mathbb{T} = \langle T, \eta, \mu \rangle$ on **Set**, then the Eilenberg-Moore adjunction is given by the functors U (forgetful)

$$\begin{aligned} U : \mathbf{Set}^{\mathbb{T}} &\to \mathbf{Set} \\ U(A, a) &= A \\ U(f : (A, a) \to (B, b)) &= f : A \to B \end{aligned}$$

and F (free)

$$\begin{aligned} F : \mathbf{Set} &\to \mathbf{Set}^{\mathbb{T}} \\ F(A) &= (TA, \mu_A) \\ F(f : A \to B) &= T(f) : (TA, \mu_A) \to (TB, \mu_B) \end{aligned}$$

It can be seen from the previous examples that $\mathbf{Set}^{\mathbb{T}}$ is isomorphic to **Mon**, and the forgetful and free functors are isomorphic to those given by the original free-forgetful adjunction for monoids as follows, U (forgetful):

$$\begin{aligned} U : \mathbf{Mon} &\to \mathbf{Set} \\ U(M) &= |M| \\ U(h : M \to N) &= h : |M| \to |N| \end{aligned}$$

and F (free)

$$\begin{aligned} F : \mathbf{Set} &\to \mathbf{Mon} \\ F(A) &= (TA, \mu_A) \\ F(f)([x_1, ..., x_n]) &= T(f)([x_1, ..., x_n]) \\ &= [f(x_1), ..., f(x_n)] \end{aligned}$$

In the next section we will see that this fact generalises to all algebraic varieties.

4 Monads and universal algebra

In this section we will show for any theory (signature with equations), the traditional category of algebras (the variety) is isomorphic to the Eilenberg-Moore category for the monad arising from the free construction corresponding to that variety.

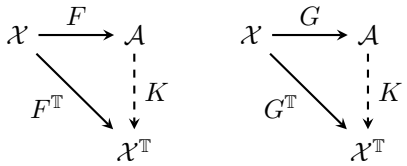
4.1 Comparison functor

Suppose we have an adjunction between categories $\mathcal{X}$ and $\mathcal{A}$, then we construct the monad $\mathbb{T}$ in $\mathcal{X}$ from that adjunction, then construct the category $\mathcal{X}^{\mathbb{T}}$. We have already said that there is more than one construction of an adjunction from a monad. So it is not always the case that $\mathcal{A} \cong \mathcal{X}^{\mathbb{T}}$. However, there is a relationship in general, which is given by the following theorem:

Theorem 4.1.1 (Comparison functor to the Eilenberg-Moore category)**.** *Let*

$$\langle F : \mathcal{X} \to \mathcal{A},\ G : \mathcal{A} \to \mathcal{X},\ \eta : 1_{\mathcal{X}} \dot{\to} GF,\ \epsilon : FG \dot{\to} 1_{\mathcal{A}} \rangle$$

be an adjunction and let $\mathbb{T} = \langle GF, \eta, G\epsilon F \rangle$ *be the monad it defines in* $\mathcal{X}$*. Then there is a unique functor* $K : \mathcal{A} \to \mathcal{X}^{\mathbb{T}}$ *such that* $G^{\mathbb{T}}K = G$ *and* $KF = F^{\mathbb{T}}$*. That is, the following diagrams commute for unique* K*:*



Note that $\langle F^{\mathbb{T}}, G^{\mathbb{T}}, \eta^{\mathbb{T}}, \epsilon^{\mathbb{T}} \rangle$ *is the Eilenberg-Moore adjunction between* $\mathcal{X}$ *and* $\mathcal{X}^{\mathbb{T}}$*.*

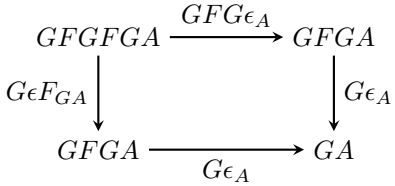
Proof. Define the comparison functor $K : \mathcal{A} \to \mathcal{X}^{\mathbb{T}}$ in the following way, for objects a and arrows f of $\mathcal{A}$.

$$KA = \langle GA, G\epsilon_A \rangle \tag{8}$$

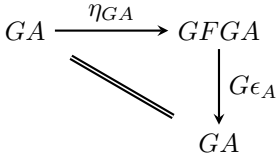
$$Kf = Gf : \langle GA, G\epsilon_A \rangle \to \langle GA', G\epsilon_{A'} \rangle \tag{9}$$

We will now prove that K is a functor. First we will show that $\langle GA, G\epsilon_A \rangle$ is a $\mathbb{T}$-algebra as (8) claims.

Consider the proposed structure map $G\epsilon_A : GFGA \to GA$, and the proposed carrier GA. The first of the $\mathbb{T}$-algebra properties is then (from (5))



Which is the definition of $G\epsilon\epsilon$ by definition 2.1.1 (recall that the square commutes in general). The second $\mathbb{T}$-algebra property is



Which happens to be one of the triangle identities from definition 2.2.1. So, K maps objects of $\mathcal{A}$ to objects of $\mathcal{X}^{\mathbb{T}}$.

Now we must show that $Kf = Gf$ is a $\mathbb{T}$-morphism as per (9). It's certainly a $\mathcal{X}$ arrow, so we need the following diagram to commute:

$$\begin{array}{ccc} GFGA & \xrightarrow{GFGf} & GFGA' \\ {\scriptstyle G\epsilon_A}\downarrow & & \downarrow{\scriptstyle G\epsilon_{A'}} \\ GA & \xrightarrow[Gf]{} & GA' \end{array}$$

This is G applied to the natural transformation square for ϵ, and thus, it commutes. So, from the above, and the fact that G is a functor we get that K is a functor in the following way:

$$K(f \circ g) = G(f \circ g) = G(f) \circ G(g) = K(f) \circ K(g)$$
$$K(1_A) = G(1_A) = 1_{GA} = 1_{KA}$$

The last step above being because the identity on $KA = \langle GA, G\epsilon_A \rangle$ is the identity on GA.

Now, given this definition of K, it should be apparent that

$$KF = F^{\mathbb{T}}, \qquad G^{\mathbb{T}}K = G$$

The left because $GF = T$ and $F^{\mathbb{T}}f = Tf$, and the right because $G^{\mathbb{T}}$ just forgets that Kf is a $\mathbb{T}$-morphism. It has been shown that a comparison functor K exists and satisfies the required equations, and so it remains to show that this functor is unique to prove the above theorem.

The property $G^{\mathbb{T}}K = G$ requires that $Gf = G^{\mathbb{T}}Kf = Kf$. A functor can be fully determined by its action on arrows, and so K is fully determined by this property, and is thus unique.

□

4.2 Beck's Theorem

Now that we have the comparison functor, we can examine when it is an isomorphism. This will later allow us to prove that the comparison functor is an isomorphism for the free-forgetful adjunction of a variety. Beck's theorem provides a suitable method of discerning whether the comparison functor is an isomorphism.

In order to state Beck's theorem, we must define the notions of split coequaliser and absolute coequaliser [2].

Definition 4.2.1. An arrow e is called an absolute coequaliser of δ_0 and δ_1 in $\mathcal{C}$ if for any functor $T : \mathcal{C} \to \mathcal{X}$ the resulting fork

$$Ta \underset{T\delta_1}{\overset{T\delta_0}{\rightrightarrows}} Tb \xrightarrow{Te} Tc$$

makes Te the coequaliser of $T\delta_1$ and $T\delta_0$.

Note that taking T above to be the identity functor, one can observe that all absolute coequalisers are coequalisers.

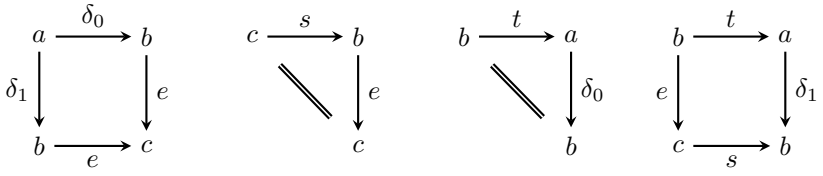
Definition 4.2.2. An arrow e is called a split coequaliser of δ_0 and δ_1 in $\mathcal{C}$ if the following is a split fork

$$a \underset{\delta_1}{\overset{\delta_0}{\rightrightarrows}} b \xrightarrow{e} c$$

That is, $s : c \to b$ an $t : b \to a$ exist, such that

$$e \circ \delta_0 = e \circ \delta_1, \quad e \circ s = 1, \quad \delta_0 \circ t = 1, \quad \delta_1 \circ t = s \circ e$$

That is, the following diagrams commute:



(10)

Since the definition of split coequaliser is given equationally, (and not in terms of unique arrows) application of a functor will again yield a split coequaliser. Thus, all split coequalisers are absolute coequalisers, and are thus coequalisers.

Before we get to Beck's Theorem, we must also define what it means for a functor to create coequalisers. The general definition for creating (co)limits can be found at the end of [2, §V.1], but here it is simpler to give the specific definition for coequalisers.

Definition 4.2.3. A functor $G : \mathcal{A} \to \mathcal{X}$ *creates* coequalisers if to every coequaliser fork in $\mathcal{X}$ there is exactly one fork in $\mathcal{A}$ which maps to that coequaliser fork under G, and that fork is a coequaliser fork in $\mathcal{A}$.

Theorem 4.2.4 (Beck's Theorem)**.** *Let*

$$\langle F : \mathcal{X} \to \mathcal{A},\ G : \mathcal{A} \to \mathcal{X},\ \eta : 1_{\mathcal{X}} \dot{\to} GF,\ \epsilon : FG \dot{\to} 1_{\mathcal{A}} \rangle$$

be an adjunction, $\langle T, \mu, \eta \rangle$ be the monad it defines in $\mathcal{X}$, $\mathcal{X}^{\mathbb{T}}$ be the Eilenberg-Moore category of $\mathbb{T}$-algebras, and

$$\langle F^{\mathbb{T}} : \mathcal{X} \to \mathcal{X}^{\mathbb{T}},\ G^{\mathbb{T}} : \mathcal{X}^{\mathbb{T}} \to \mathcal{X},\ \eta : 1_{\mathcal{X}} \dot{\to} G^{\mathbb{T}}F^{\mathbb{T}},\ \epsilon : F^{\mathbb{T}}G^{\mathbb{T}} \dot{\to} 1^{\mathbb{T}}_{\mathcal{X}} \rangle$$

be the corresponding adjunction. Then the following are equivalent:

1. *The comparison functor $K : A \to \mathcal{X}^{\mathbb{T}}$ is an isomorphism.*
2. *The functor $G : \mathcal{A} \to \mathcal{X}$ creates coequalizers for those parallel pairs f, g in $\mathcal{A}$ for which Gf, Gg has an absolute coequalizer in $\mathcal{X}$.*
3. *The functor $G : \mathcal{A} \to \mathcal{X}$ creates coequalizers for those parallel pairs f, g in $\mathcal{A}$ for which Gf, Gg has a split coequalizer in $\mathcal{X}$.*

Proof. We will show that 1 implies 2, that 2 implies 3 and that 3 implies 1.

That 2 implies 3 is clear because all split coequalisers are absolute coequalisers.

That 1 implies 2 will now be shown. So, given a unique comparison functor, we need to consider the parallel pairs of arrows in $\mathcal{A}$ which correspond via G to parallel pairs of arrows with an absolute coequaliser in $\mathcal{X}$. Then for those particular parallel pairs of arrows in $\mathcal{A}$ we must show that the functor G creates coequalisers for them. For such absolute coequalisers the image under $F^{\mathbb{T}}$ is a coequaliser in $\mathcal{X}^{\mathbb{T}}$.

$$\langle x, h\rangle \underset{d_1}{\overset{d_0}{\rightrightarrows}} \langle y, k\rangle$$

of $\mathbb{T}$-algebras for which the corresponding arrows in $\mathcal{X}$ have an absolute coequaliser.

$$x \underset{d_1}{\overset{d_0}{\rightrightarrows}} y \xrightarrow{e} z$$

To create a coequaliser for this parallel pair we must first find a unique T-algebra structure $m : Tz \to z$ on z such that e becomes a map of T-algebras □

4.3 Free constructions for varieties

The fact that varieties admit free constructions is a standard result in universal algebra and category theory. Despite this, I have been unable to locate a proof which is even vaguely satisfactory. The two best I found were [15, Theorem 8.3.3], which didn't include equations and didn't prove that there was an adjunction, and a single sentence in [2, §V.6] claiming that the result is an easy generalisation of the case for groups which is given in that section.

Theorem 4.3.1. *Given a variety* (Ω, E)-**Alg**, *we can construct an adjunction (the free-forgetful adjunction to be precise) between the categories* **Set** *and* (Ω, E)-**Alg**.

Instead of proving this, I will give some intuitive sense of what the free and forgetful functors should be. Firstly, the forgetful functor is fairly clear. On algebras it returns the carrier set and on morphisms it returns the underlying function, and this is clearly a functor because composition and identities are the same.

The free functor acts on sets X and takes them to algebras in which the carrier set is the set of ground terms of the theory taking constants from X (and obviously the nullary operations in the theory). The operations are given the obvious interpretation, which is that applying them takes a list of terms to a new term which is just the old terms encased with the operation symbol being interpreted. Since there are equations, the actual terms of the algebra need to be the equivalence classes under the quotient by the theory's equations, but this doesn't really affect how the operations work.

The free functor also acts on functions $f : X \to X'$ in that it distributes their effect over all the operation symbols in the theory so that they are directly applied to the constants $x \in X$ and only those constants. These form acceptable homomorphisms because the equations of the theory must remain satisfied when functions are applied in this way.

4.4 Universal algebra

We will now show that the forgetful functor U above creates coequalisers for those parallel pairs f, g in (Ω, E)-**Alg** for which Gf,Gg has an absolute/split coequaliser in **Set**. Thus, by Beck's Theorem, we will have shown that the comparison functor $K : (\Omega, E)\text{-}\mathbf{Alg} \to \mathbf{Set}^{\mathbb{T}}$ is an isomorphism.

Theorem 4.4.1. *The comparison functor $K : (\Omega, E)\text{-}\mathbf{Alg} \to \mathbf{Set}^{\mathbb{T}}$ is an isomorphism.*

Proof. Consider an arbitrary parallel pair of arrows $f, g : A \rightrightarrows B$ in $\mathcal{X}^{\mathbb{T}}$ for which the underlying functions have an absolute coequaliser e:

$$GA \underset{Gg}{\overset{Gf}{\rightrightarrows}} GB \xrightarrow{e} X$$

We will prove that $G : (\Omega, E)\text{-}\mathbf{Alg} \to \mathbf{Set}$ creates coequalisers for these pairs. To do so, we must show that e lifts to a unique morphism of algebras and then that this map is a coequaliser of f and g.

Thus, consider an arbitrary n-ary operator $\omega \in \Omega$. We can construct the following diagram:

$$\begin{array}{ccccc} A^n & \underset{g^n}{\overset{f^n}{\rightrightarrows}} & B^n & \xrightarrow{e^n} & X^n \\ {\scriptstyle \omega_A}\downarrow & & \downarrow{\scriptstyle \omega_B} & & \vdots{\scriptstyle \omega_X} \\ A & \underset{g}{\overset{f}{\rightrightarrows}} & B & \xrightarrow[e]{} & X \end{array} \tag{11}$$

f and g are (Ω, E)-morphisms, so the left squares commute. The function e is an absolute coequaliser in **Set** and therefore its n-th power is also a coequaliser (of f^n and g^n). So since

$$e \circ \omega_B \circ f^n = e \circ f \circ \omega_A = e \circ g \circ \omega_A = e \circ \omega_B \circ g^n$$

the universal property of e^n will force $e \circ \omega_B$ to factor uniquely through the coequaliser via $e \circ \omega_B = \omega_X \circ e^n$. This unique factorisation uniquely defines all the operation ω_X on X so that the square on the right of (11) commutes for all $\omega \in \Omega$.

It must now be shown that e is a coequaliser of f and g as (Ω, E)-morphisms. So, consider another morphism h such that $h \circ f = h \circ g$. Then $h \circ f = h \circ g$ in **Set** (by applying the forgetful functor G), so in **Set** $h = h'e$ for unique function h'. So we must show that

$$\begin{array}{ccc} X^n & \overset{h'^n}{\dashrightarrow} & C^n \\ {\scriptstyle \omega_X}\downarrow & & \downarrow{\scriptstyle \omega_C} \\ X & \underset{h'}{\dashrightarrow} & C \end{array} \tag{12}$$

commutes for every operation ω. Now, h is a morphism of algebras, so

$$h' \circ \omega_X \circ e^n = h' \circ e \circ \omega_B = h \circ \omega_B = \omega_C \circ h^n = \omega_C \circ h'^n \circ e^n$$

and since e^n is a coequaliser, e^n is epi, so $h' \circ \omega_X = \omega_C \circ h'^n$, as required by (12). □

5 Comonads

The comonad is the dual concept of a monad. A formal definition is easily obtained by considering monads in $\mathcal{C}^{op}$. All the results pertaining to monads, including those above can be dualised to the comonad case. Thus, this section will not be concerned as much with the theory of comonads, but rather with examples of their applications.

Importantly the concept of algebra dualises to that of a coalgebra, for which there are both functor and comonad versions. Coalgebra is useful for reasoning about state transition systems [6] and modal logic [7]. For algebra, equations were key, but for coalgebra, they must be replaced with behaviours [16] or bisimulations [17]. While monads are useful for constructively defined (algebraic) data types, comonads can provide a formal model for unbounded data types [17].

5.1 State transition systems

Where algebraic induction has a notion of recursive construction, coalgebraic coinduction has a notion of recursive observation. There is a conceptual connection between this kind of observation and any kind of state transistion system. As such, we expect, and do find that coalgebra provides a general and neat language to describe all manner of state transition systems. [6]

5.2 Modal Logic

Modal logic is a large field unto itself, and coalgebraic methods have only been explored relatively recently. Coalgebraic methods allow generic tools to be created, both for showing soundness and completeness results [7] and for generating automated theorem provers [18].

5.3 Behaviours and Bisimulations

Bisimulation is a concept which stems from concurrency. One wants to be able to formally specify certain aspects of the behaviour of concurrent systems. Coalgebra shows promise of being able to provide formalisms for providing enough freedom to concurrent systems to maintain the use of multicore and multiprocessor systems, while providing enough boundaries to ensure program correctness. [17]

5.4 Unbounded data types

Unbounded data types might initially seem like an odd thing to desire. The truth is that in any real-time system, modelling data as a stream of input is much more natural than modelling it with finite data structures. Coalgebra allows the formal specification of infinite data types with coinduction [6].

References

[1] Barr M. & Wells C. *Toposes, Triples and Theories.* A Series of Comprehensive Studies in Mathematics. Springer, New York, first edition, 1985.

[2] Mac Lane S. *Categories for the working mathematician.* Graduate Texts in Mathematics. Springer, New York, second edition, 1978.

[3] Hughes J. Generalising monads to arrows. *Science of Computer Programming*, 37:66–111, 2000.

[4] HaskellWiki. Applications of monads in Haskell. `http://www.haskell.org/haskellwiki/Research_papers/Monads_and_arrows`. Website accessed: 1 June 2012.

[5] Marlow S. & Newton R. & Peyton-Jones S. A monad for deterministic parallelism. In *Proceedings of the 4th ACM symposium on Haskell*, Haskell '11, pages 71–82, New York, NY, USA, 2011. ACM.

[6] Jacobs B. & Rutten J. A tutorial on (co)algebras and (co)induction. *European Association for Theoretical Computer Science*, 62:222–259, 1997.

[7] Pattinson D. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1-3):177 – 193, 2003.

[8] Lawvere F. W. *Functorial Semantics of Algebraic Theories.* PhD thesis, Columbia University, 1963. (Available with commentary as TAC Reprint 5).

[9] Hyland M. & Power J. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, 2007.

[10] Awodey S. *Category Theory.* Oxford Logic Guides. Oxford Press, Oxford, second edition, 2010.

[11] Willerton S. String diagrams 2. `http://www.youtube.com/watch?v=JeGhNhgOTuk#t=133`, 2007. YouTube video.

[12] Pierce B.C. *Basic Category Theory for Computer Scientists.* Foundations of Computing. MIT Press, Cambridge, Massachusetts, first edition, 1991.

[13] Eilenberg S. & Moore J.C. Adjoint functors and triples. *Illinois J. Math.*, 9(3):381–398, 1965.

[14] Kliesli H. Every standard construction is induced by a pair of adjoint functors. In *Proceedings of the American Mathematical Society*, volume 16, pages 544–546, June 1965.

[15] Bergman G.M. *An Invitation to General Algebra and Universal Constructions.* Henry Helson, 15 the Crescent, Berkeley CA, second edition, 1998. An earlier edition appeared as Berkeley Mathematics Lecture Notes 7.

[16] Gumm H.P. Elements of the general theory of coalgebras. Preprint, 1999.

[17] Rutten J. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[18] Gore R & Kupke C & Pattinson D. Optimal tableau algorithms for coalgebraic logics. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 114–128. Springer Berlin / Heidelberg, 2010.