Free Monoids as Initial Algebras

Bartosz Milewski

A. Preface

In my previous blog post I used, without proof, the fact that the initial algebra of the functor $I + h \otimes -$ is a free monoid. The proof of this statement is not at all trivial and, frankly, I would never have been able to work it out by myself. I was lucky enough to get help from a mathematician, Alex Campbell, who sent me the proof he extracted from the paper by G. M. Kelly [1].

I worked my way through this proof, filling some of the steps that might have been obvious to a mathematician, but not to an outsider. I even learned how to draw diagrams using the TikZ package for LaTeX.

What I realized was that category theorists have developed a unique language to manipulate mathematical formulas: the language of 2-dimensional diagrams. For a programmer interested in languages this is a fascinating topic. We are used to working with grammars that essentially deal with string substitutions. Although diagrams can be serialized—TikZ lets you do it—you can't really operate on diagrams without laying them out on a page. The most common operation—diagram pasting—involves combining two or more diagrams along common edges. I am amazed that, to my knowledge, there is no tool to mechanize this process.

In this post you'll also see lots of examples of using the same diagram shape (the free-construction diagram, or the algebra-morphism diagram), substituting new expressions for some of the nodes and edges. Not all substitutions are valid and I'm sure one could develop some kind of type system to verify their correctness.

Because of proliferation of diagrams, this blog post started growing out of proportion, so I decided to split it into two parts. If you are serious about studying this proof, I strongly suggest you download (or even print out) the PDF version of this blog.

I. Part I: Free Algebras

A. Introduction

Here's the broad idea: The initial algebra that defines a free monoid is a fixed point of the functor $I + h \otimes -$, which I will call the list functor. Roughly speaking, it's the result of replacing the dash in the definition of the functor with the result of the replacement. For instance, in the first iteration we would get:

$$I + h \otimes (I + h \otimes -) \cong I + h + h \otimes h \otimes -$$

I used the fact that I is the unit of the tensor product, the associativity of $\otimes$ (all up to isomorphism), and the distributivity of tensor product over coproduct.

Continuing this process, we would arrive at an infinite sum of powers of h:

$$m = I + h + h \otimes h + h \otimes h \otimes h + ...$$

Intuitively, a free monoid is a list of hs, and this representation expresses the fact that a list is either trivial (corresponding to the unit I), or a single h, or a product of two hs, and so on...

Let's have a look at the structure map of the initial algebra of the list functor:

$$I + h \otimes m \to m$$

Mapping out of a coproduct (sum) is equivalent to defining a pair of morphisms $\langle \pi, \sigma \rangle$:

$$\pi \colon I \to m$$

$$\sigma \colon h \otimes m \to m$$

the second of which may, in itself, be considered an algebra for the product functor $h \otimes -$.

Our goal is to show that the initial algebra of the list functor is a monoid so, in particular, it supports multiplication:

$$\mu \colon m \otimes m \to m$$

Following our intuition about lists, this multiplication corresponds to list concatenation. One way of concatenating two lists is to keep multiplying the second list by elements taken from the first list. This operation is described by the application of our product functor $h \otimes -$. Such repetitive application of a functor is described by a *free algebra*.

There is just one tricky part: when concatenating two lists, we have to disassemble the left list starting from the tail (alternatively, we could disassemble the right list from the head, but then we'd have to append elements to the tail of the left list, which is the same problem). And here's the ingenious idea: you disassemble the left list from the head, but instead of applying the elements directly to the right list, you turn them into functions that prepend an element. In other words you convert a list of elements into a (reversed) list of functions. Then you apply this list of functions to the right list one by one.

This conversion is only possible if you can trade product for function – the process we know as currying. Currying is possible if there is an adjunction between the product and the exponential, a.k.a, the internal hom, $[k, n]$ (which generalizes the set of functions from k to n):

$$C(m \otimes k, n) \cong C(m, [k, n])$$

We'll assume that the underlying category C is monoidal *closed*, so that we can curry morphisms that map out from the tensor product:

$$g\colon m \otimes k \to n$$

$$\bar{g}\colon m \to [k, n]$$

(In what follows I'll be using the overbar to denote the curried version of a morphism.)

The internal hom can also be defined using a universal construction, see Fig. 1. The morphism $eval$ corresponds to the counit of the adjunction (although the universal construction is more general than the adjunction).

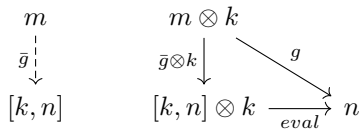

Fig. 1. Universal construction of the internal hom $[k, n]$. For any object m and a morphism $g\colon m \otimes k \to n$ there is a unique morphism $\bar{g}$ (the *curried* version of g) which makes the triangle commute.

The hard part of the proof is to show that the initial algebra produces a *free* monoid, which is a free object in the category of monoids. I'll start by defining the notion of a free object.

B. Free Objects

You might be familiar with the definition of a free construction as the left adjoint to the forgetful functor. Fig 2 illustrates the essence of such an adjunction.

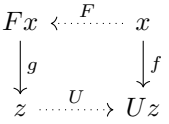

Fig. 2. Free/forgetful adjunction

The left hand side is in some category D of structured objects: algebras, monoids, monads, etc. The right hand side is in the underlying category C, often the category of sets. The adjunction establishes a one-to-one correspondence between sets of morphisms, of which g and f are examples. If U is the forgetful functor, then F is the free functor, and the object Fx is called the free object generated by x. The adjunction is an isomorphism of hom-sets, natural in both x and z:

$$D(Fx, z) \cong C(x, Uz)$$

Unfortunately, this description doesn't quite work for some important cases, like free monads. In the case of free monads, the right category is the category of endofunctors, and the left category is the category of monads. Because of size issues, not every endofunctor on the right generates a free monad on the left.

It turns out that there is a weaker definition of a free object that doesn't require a full blown adjunction; and which reduces to it, when the adjunction can be defined globally.

Let's start with the object x on the right, and try to define the corresponding free object Fx on the left (by abuse of notation I will call this object Fx, even if there is no functor F). For our definition, we need a condition that would work universally for any object z, and any morphism f from x to Uz.

We are still missing one important ingredient: something that would tell us that x acts as a set of generators for Fx. This property can be expressed by providing a morphism that inserts x into $U(Fx)$—the object underlying the free object. In the case of an adjunction, this morphism happens to be the component of the unit natural transformation:

$$\eta\colon Id \to U \circ F$$

where Id is the identity functor (see Fig. 3).

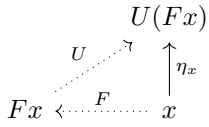

Fig. 3. Unit of adjunction

The important property of the unit of adjunction η is that it can be used to recover the mapping from the left hom-set to the right hom-set in Fig. 2. Take a morphism $g\colon Fx \to z$, lift it using U, and compose it with η_x. You get a morphism $f\colon x \to Uz$:

$$f = Ug \circ \eta_x$$

In the absence of an adjunction, we'll make the existence of the morphism η_x part of the definition of the free object.

Definition 1.1: Free object. A free object on x consists of an object Fx and a morphism $\eta_x\colon x \to U(Fx)$ such that, for every object z and a morphism $f\colon x \to Uz$, there is a unique morphism $g\colon Fx \to z$ such that:

$$Ug \circ \eta_x = f$$

The diagram in Fig. 4 illustrates this definition. It is essentially a composition of the two previous diagrams, except that we don't require the existence of a global mapping, let alone a functor, F.

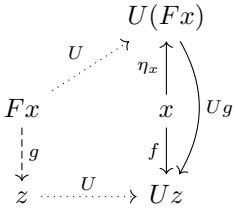

Fig. 4. Definition of a free object Fx

The morphism η_x is called the universal mapping, because it's independent of z and f. The equation:

$$Ug \circ \eta_x = f$$

is called the universal condition, because it works universally for every z and f. We say that g is *induced* by the morphism f.

There is a standard trick that uses the universal condition: Suppose you have two morphisms g and g' from the universal object to some z. To prove that they are equal, it's enough to show that they are both induced by the same f. Showing the equality:

$$Ug \circ \eta_x = Ug' \circ \eta_x$$

is often easier because it lets you work in the simpler, underlying category.

C. Free Algebras

As I mentioned in the introduction, we are interested in algebras for the product functor: $h \otimes -$, which we'll call h-algebras. Such an algebra consists of a carrier n and a structure map:

$$\nu \colon h \otimes n \to n$$

For every h, h-algebras form a category; and there is a forgetful functor U that maps an h-algebra to its carrier object n. We can therefore define a free algebra as a *free object* in the category of h-algebras, which may or may not be generalizable to a full-blown free/forgetful adjunction. Fig. 5 shows the universal condition that defines such a free algebra (m_k, σ) generated by an object k.

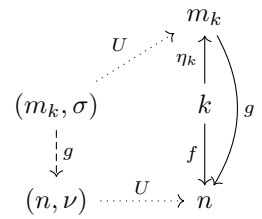

Fig. 5. Free h-algebra (m_k, σ) generated by k

In particular, we can define an important free h-algebra generated by the identity object I. This algebra (m, σ) has the structure map:

$$\sigma \colon h \otimes m \to m$$

and is described by the diagram in Fig. 15:

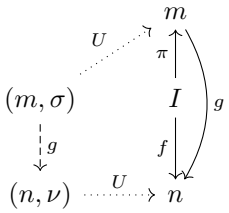

Fig. 6. Free h-algebra (m, σ) generated by I

Its universal condition reads:

$$g \circ \pi = f$$

By definition, since g is an algebra morphism, it makes the diagram in Fig. 7 commute:

$$\begin{array}{ccc} h \otimes m & \xrightarrow{\sigma} & m \\ \downarrow{\scriptstyle h \otimes g} & & \downarrow{\scriptstyle g} \\ h \otimes n & \xrightarrow{\nu} & n \end{array}$$

Fig. 7. g is an h-algebra morphism

We use a notational convenience: $h \otimes g$ is the lifting of the morphism g by the product functor $h \otimes -$. This might be confusing at first, because it looks like we are multiplying an object h by a morphism g. One way to parse it is to consider that, if we keep the first argument to the tensor product constant, then it's a functor in the second component, symbolically $h \otimes -$. Since it's a functor, we can use it to lift a morphism g, which can be notated as $h \otimes g$.

Alternatively, we can exploit the fact that tensor product is a bifunctor, and therefore it may lift a pair of morphism, as in $id_h \otimes g$; and $h \otimes g$ is just a shorthand notation for this.

Bifunctoriality also means that the tensor product preserves composition and identity in both arguments. We'll use these facts extensively later, especially as the basis for string diagrams.

The important property of m is that it also happens to be the initial algebra for the functor $I + h \otimes -$. Indeed, for any other algebra with the carrier n and the structure map a pair $\langle f, \nu \rangle$, there exists a unique g given by Fig 6, such that the diagram in Fig. 8 commutes:

$$\begin{array}{ccc} I + h \otimes m & \xrightarrow{\langle \pi, \sigma \rangle} & m \\ {\scriptstyle \langle inl,\, inr \circ (h \otimes g) \rangle} \downarrow & & \downarrow{\scriptstyle g} \\ I + h \otimes n & \xrightarrow{\langle f, \nu \rangle} & n \end{array}$$

Fig. 8. Initiality of the algebra $(m, \langle \pi, \sigma \rangle)$ for the functor $I + h \otimes -$.

Here, inl and inr are the two injections into the coproduct.

If you visualize m as the sum of all powers of h, π inserts the unit I (zeroth power) into it, and σ inserts the sum of non-zero powers.

$$\langle \pi, \sigma \rangle \colon I + h \otimes m \to m$$

The advantage of this result is that we can concentrate on studying the simpler problem of free h-algebras rather than the problem of initial algebras for the more complex list functor.

D. Example

Here's some useful intuition behind h-algebras. Think of the functor $(h \otimes -)$ as a means of forming formal expressions. These are very simple expressions: you take a variable and pair it (using the tensor product) with h. To define an algebra for this functor you pick a particular type n for your variable (i.e., the carrier object) and a recipe for evaluating any expression of type $h \otimes n$ (i.e., the structure map).

Let's try a simple example in Haskell. We'll use pairs (and tuples) as our tensor product, with the empty tuple `()` as unit

(up to isomorphism). An algebra for a functor `f` is defined as:

```
type Algebra f a = f a -> a
```

Consider h-algebras for a particular choice of h: the type of real numbers `Double`. In other words, these are algebras for the functor `((,) Double)`. Pick, as your carrier, the type of vectors:

```
data Vector = Vector { x :: Double
                     , y :: Double
                     } deriving Show
```

and the structure map that scales a vector by multiplying it by a number:

```
vecAlg :: Algebra ((,) Double) Vector
vecAlg (a, v) = Vector { x = a * x v
                       , y = a * y v }
```

Define another algebra with the carrier the set of real numbers, and the structure map multiplication by a number.

```
mulAlg :: Algebra ((,) Double) Double
mulAlg (a, x) = a * x
```

There is an algebra morphism from `vecAlg` to `mulAlg`, which takes a vector and maps it to its length.

```
algMorph :: Vector -> Double
algMorph v = sqrt((x v)^2 + (y v)^2)
```

This is an algebra morphism, because it doesn't matter if you first calculate the length of a vector and then multiply it by `a`, or first multiply the vector by `a` and then calculate its length (modulo floating-point rounding).

A *free* algebra has, as its carrier, the set of all possible expressions, and an evaluator that tells you how to convert a functor-ful of such expressions to another valid expression. A good analogy is to think of the functor as defining a grammar (as in BNF grammar) and a free algebra as a language generated by this grammar.

You can generate free h-expressions recursively. The starting point is the set of generators k as the source of variables. Applying the functor to it produces $h \otimes k$. Applying it again, produces $h \otimes h \otimes k$, and so on. The whole set of expressions is the infinite coproduct (sum):

$$k + h \otimes k + h \otimes h \otimes k + ...$$

An element of this coproduct is either an element of k, or an element of the product $h \otimes k$, and so on...

The universal mapping injects the set of generators k into the set of expressions (here, it would be the leftmost injection into the coproduct).

In the special case of an algebra generated by the unit I, the free algebra simplifies to the power series:

$$I + h + h \otimes h + ...$$

and π injects I into it.

Continuing with our example, let's consider the free algebra for the functor `((,) Double)` generated by the unit `()`. The free algebra is, symbolically, an infinite sum (coproduct) of tuples:

```
data Expr =
    ()
  | Double
  | (Double, Double)
  | (Double, Double, Double)
  | .dot.dot.dot
```

Here's an example of an expression:

```
e = (1.dot1, 2.dot2, 3.dot3)
```

As you can see, free expressions are just lists of numbers. There is a function that inserts the unit into the set of expressions:

```
pi :: () -> Expr
pi () = ()
```

The free evaluator is a function (not actual Haskell):

```
sigma (a, ()) = a
sigma (a, x)  = (a, x)
sigma (a, (x, y)) = (a, x, y)
sigma (a, (x, y, z)) = (a, x, y, z)
.dot.dot.dot
```

Let's see how the universal property works in our example. Let's try, as the target (n, ν), our earlier vector algebra:

```
vecAlg :: Algebra ((,) Double) Vector
vecAlg (a, v) = Vector { x = a * x v
                       , y = a * y v }
```

The morphism `f` picks some vector, for instance:

```
f :: () -> Vector
f () = Vector 3 4
```

There should be a unique morphism of algebras `g` that takes an arbitrary expression (a list of `Double`s) to a vector, such that `g . pi = f` picks our special vector:

```
Vector 3 4
```

In other words, `g` must take the empty tuple (the result of `pi`) to `Vector 3 4`. The question is, how is `g` defined for an arbitrary expression? Look at the diagram in Fig. 7 and the commuting condition it implies:

$$g \circ \sigma = \nu \circ (h \otimes g)$$

Let's apply it to an expression `(a, ())` (the action of the functor `(h, -)` on `()`). Applying `sigma` to it produces `a`, followed by the action of `g` resulting in `g a`. This should be the same as first applying the lifted `(id, g)` acting on `(a, ())`, which gives us

`(a, Vector 3 4)`; followed by `vecAlg`, which produces `Vector (a * 3) (a * 4)`. Together, we get:

```
g a = Vector (a * 3) (a * 4)
```

Repeating this process gives us:

```
g :: Expr -> Vector
g () = Vector 3 4
g a  = Vector (a * 3) (a * 4)
g (a, b) = Vector (a * b * 3) (a * b * 4)
.dot.dot.dot
```

This is the unique `g` induced by our `f`.

E. Properties of Free Algebras

Here are some interesting properties that will help us later: h-algebras are closed under multiplication and exponentiation. If (n, ν) is an h-algebra, then there are also h-algebras with the carriers $n \otimes k$ and $[k, n]$, for an arbitrary object k. Let's find their structure maps.

The first one should be a mapping:

$$h \otimes n \otimes k \to n \otimes k$$

which we can simply take as the lifting of ν by the tensor product: $\nu \otimes k$.

The second one is:

$$\tau_k \colon h \otimes [k, n] \to [k, n]$$

which can be uncurried to:

$$h \otimes [k, n] \otimes k \to n$$

We have at our disposal the counit of the hom-adjunction:

$$eval \colon [k, n] \otimes k \to n$$

which we can use in combination with ν:

$$\nu \circ (h \otimes eval)$$

to implement the (uncurried) structure map.

Here's the same construction for Haskell programmers. Given an algebra:

```
nu :: Algebra ((,) h) n
```

we can create two algebras:

```
alpha :: Algebra ((,) h) (n, k)
alpha (a, (n, k)) = (nu (a, n), k)
```

and:

```
tau :: Algebra ((,) h) (k -> n)
tau (a, kton) = \k -> nu (a, kton k)
```

These two algebras are related through an adjunction in the category of h-algebras:

$$Alg\big((n \otimes k, \nu \otimes k), (l, \lambda)\big) \cong Alg\big((n, \nu), ([k, l], \tau_k)\big)$$

which follows directly from hom-adjunction acting on carriers.

$$C(n \otimes k, l) \cong C(n, [k, l])$$

Finally, we can show how to generate free algebras from arbitrary generators. Intuitively, this is obvious, since it can be described as the application of distributivity of tensor product over coproduct:

$$k + h \otimes k + h \otimes h \otimes k + \ldots = (I + h + h \otimes h + \ldots) \otimes k$$

More formally, we have:

Proposition 1.1: If (m, σ) is is the free h-algebra generated by I, then $(m \otimes k, \sigma \otimes k)$ is the free h-algebra generated by k with the universal map given by $\pi \otimes k$.

Proof: Let's show the universal property of $(m \otimes k, \sigma \otimes k)$. Take any h-algebra (n, ν) and a morphism:

$$f \colon k \to n$$

We want to show that there is a unique $g \colon m \otimes k \to n$ that closes the diagram in Fig. 9.

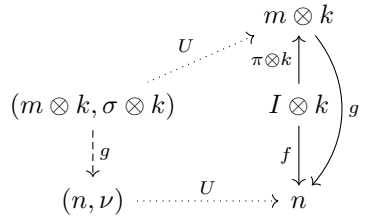

Fig. 9. Free h-algebra generated by $k \cong I \otimes k$

I have rewritten f (taking advantage of the isomorphism $I \otimes k \cong k$), as:

$$f \colon I \otimes k \to n$$

We can curry it to get:

$$\bar{f} \colon I \to [k, n]$$

The intuition is that the morphism $\bar{f}$ selects an element of the internal hom $[k, n]$ that corresponds to the original morphism $f \colon k \to n$.

We've seen earlier that there exists an h-algebra with the carrier $[k, n]$ and the structure map τ_k. But since m is the free h-algebra, there must be a unique algebra morphism $\bar{g}$ (see Fig. 10):

$$\bar{g} \colon (m, \sigma) \to ([k, n], \tau_k)$$

such that:

$$\bar{g} \circ \pi = \bar{f}$$

Uncurying this $\bar{g}$ gives us the sought after g.

The universal condition in Fig. 9:

$$g \circ (\pi \otimes k) = f$$

follows from pasting together two diagrams that define the relevant hom-adjunctions in Fig 11 (c.f., Fig. 1). ■

The immediate consequence of this proposition is that, in order to show that two h-algebra morphisms $g, g' \colon m \otimes k \to$

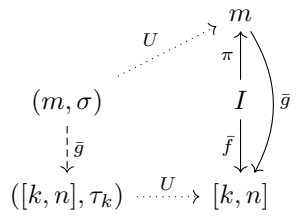

Fig. 10. The construction of the unique morphism $\bar{g}$

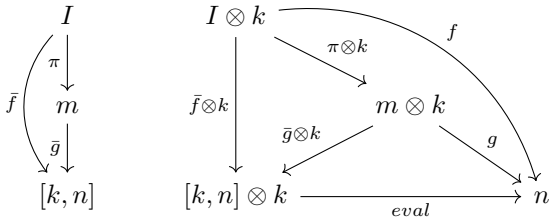

Fig. 11. The diagram defining the currying of both g and $g \circ (\pi \otimes k)$. This is the pasting together of two diagrams that define the universal property of the internal hom $[k, n]$, one for the object I and one for the object m.

n are equal, it's enough to show the equality of two regular morphisms:

$$g \circ (\pi \otimes k) = g' \circ (\pi \otimes k) \colon k \to n$$

(modulo isomorphism between k and $I \otimes k$). We'll use this property later.

It's instructive to pick apart this proof in the simpler Haskell setting. We have the target algebra:

```
nu :: Algebra ((,) h) n
```

There is a related algebra $[k, n]$ of the form:

```
tau :: Algebra ((,) h) (k -> n)
tau (a, kton) = \k -> nu (a, kton k)
```

We've analyzed, previously, a version of the universal construction of g, which we can now generalize to Fig. 10. We can build up the definition of $\bar{g}$, starting with the condition $\bar{g} \circ \pi = \bar{f}$. Here, $\bar{f}$ selects a function from the hom-set: this function is our f. That gives us the action of g on the unit:

```
g :: Expr -> (k -> n)
g () = f
```

Next, we make use of the fact that $\bar{g}$ is an algebra morphism that satisfies the commuting condition:

$$\bar{g} \circ \sigma = \tau \circ (h \otimes \bar{g})$$

As before, we apply this equation to the expression `(a, ())`. The left hand side produces `g a`, while the right hand side produces `tau (a, f)`. Next, we apply the same equation to `(a, (b, ()))`. The left hand side produces `g (a, b)`. The right hand produces `tau (a, tau (b, f))`, and so on. Applying the definition of `tau`, we get:

```
g :: Expr -> (k -> n)
g () = f
g a  = \k -> nu (a, f k)
g (a, b) = \k -> nu (a, nu (b, f k))
.dot.dot.dot
```

Notice the order reversal in function application. The list that is the argument to `g` is converted to a series of applications of `nu`, with list elements in reverse order. We first apply `nu (b, _)` and then `nu (a, _)`. This reversal is crucial to implementing list concatenation, where `nu` will prepend elements of the first list to the second list. We'll see this in the second installment of this blog post.

II. Part II: Free Monoids

A. String Diagrams

The utility of diagrams in formulating and proving theorems in category theory cannot be overemphasized. While working my way through the construction of free monoids, I noticed that there was a particular set of manipulations that had to be done algebraically, with little help from diagrams. These were operations involving a mix of tensor products and composition of morphisms. Tensor product is a bifunctor, so it preserves composition; which means you can often slide products through junctions of arrows—but the rules are not immediately obvious. Diagrams in which objects are nodes and morphisms are arrows have no immediate graphical representation for tensor products. A thought occurred to me that maybe a dual representation, where morphisms are nodes and objects are edges would be more accommodating. And indeed, a quick search for "string diagrams in monoidal categories" produced a paper by Joyal and Street, "The geometry of tensor calculus."

The idea is very simple: if you represent morphisms as points on a plane, you have two additional dimensions to play with composition and tensoring. Two morphism—represented as points—can be composed if they share an object, which can be represented as a line connecting them. By convention, we read composition from the bottom of the diagram up. We follow lines as they go through points—that's composition. Two lines ascending in parallel represent a tensor product. The geometry of the diagram just works!

Let me explain it on a simple example—the left unit law for a monoid (m, π, μ):

$$\mu \circ (\pi \otimes m) = id$$

The left hand side is a composition of two morphisms. The first morphism $\pi \otimes m$ starts from the object $I \otimes m$ (see Fig. 12).

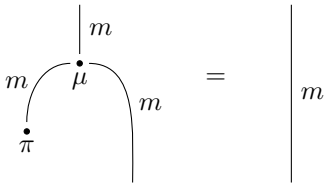

Fig. 12. Left unit law

In principle, there should be two parallel lines at the bottom, one for I and one for m; but $I \otimes m$ is isomorphic to m, so the I line is redundant and can be omitted. Scanning the diagram from the bottom up, we encounter the morphism π in parallel with the m line. That's exactly the graphical representation of $\pi \otimes m$. The output of π is also m, so we now have two upward moving m lines, corresponding to $m \otimes m$. That's the input of the next morphism μ. Its output is the single upward moving m. The unit law may be visualized as pulling the two m strings in opposite direction until the whole diagram is straightened to one vertical m string corresponding to id_m.

Here's the right unit law:

$$\mu \circ (m \otimes \pi) = id$$

It works like a mirror image of the left unit law:

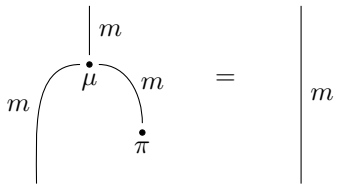

Fig. 13. Right unit law

The associativity law can be illustrated by the following diagram:

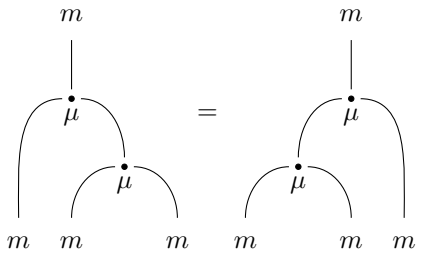

Fig. 14. Associativity law

The important property of a string diagram is that, because of functoriality, its value—the compound morphism it represents—doesn't change under continuous transformations.

B. Monoid

First we'd like to show that the carrier of the free h-algebra (m, σ) which, as we've seen before, is also the initial algebra for the list functor $I + h \otimes -$, is automatically a monoid. To show that, we need to define its unit and multiplication—two morphisms that satisfy monoid laws. The obvious candidate for unit is the universal mapping $\pi \colon I \to m$. It's the morphism in the definition of the free algebra from the previous post (see Fig 15).

Multiplication is a morphism:

$$\mu \colon m \otimes m \to m$$

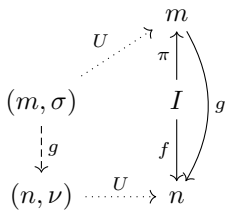

Fig. 15. Free h-algebra (m, σ) generated by I

which, if you think of a free monoid as a list, is the generalization of list concatenation.

The trick is to show that $m \otimes m$ is also a free h-algebra whose generator is m itself. We could then use the universality of $m \otimes m$ to generate the unique algebra morphism from it to m (which is also an h-algebra). That will be our μ.

Proposition 2.1: Monoid.

The free h-algebra (m, σ) generated by the unit object I is a monoid whose unit is:

$$\pi \colon I \to m$$

and whose multiplication:

$$\mu \colon m \otimes m \to m$$

is the unique h-algebra morphism

$$(m \otimes m, \sigma \otimes m) \to (m, \sigma)$$

induced by the identity morphism id_m.

Proof: In the previous post we've shown that, if (m, σ) is a free algebra generated by the unit object I with the universal map π, then $(m \otimes k, \sigma \otimes k)$ is a free algebra generated by k with the universal map $\pi \otimes k$ (see Fig. 16).

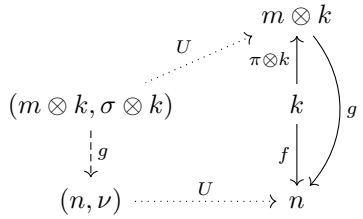

Fig. 16. Free h-algebra generated by $k \cong I \otimes k$

We get μ by redrawing this diagram: using m as both the generator and the target algebra, and replacing f with id_m (see Fig. 17):

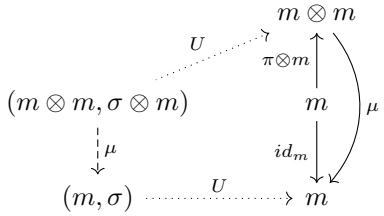

Fig. 17. Monoid multiplication as an h-algebra morphism

Since so defined μ is an h-algebra morphism, it makes the following diagram, Fig. 18, commute.

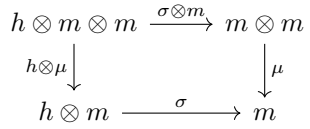

Fig. 18. μ is an h-algebra morphism

This commuting condition can be redrawn as the identity of two string diagrams (Fig. 19) corresponding to the two paths through the original diagram.

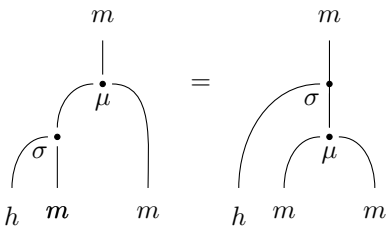

Fig. 19. String diagram showing that μ is an algebra morphism

The universal condition in Fig. 17:

$$\mu \circ (\pi \otimes m) = id_m$$

gives us immediately the left unit law for the monoid.

The right unit law:

$$\mu \circ (m \otimes \pi) = id_m$$

requires a little more work.

There is a standard trick that we can use to show that two morphisms, whose source (in this case m) is a free algebra, are equal. It's enough to prove that they are algebra morphisms, and that they are both induced by the same morphism (in this case π). Their equality then follows from the uniqueness of the universal construction.

We know that μ is an algebra morphism so, if we can show that $m \otimes \pi$ is also an algebra morphism, their composition will be an algebra morphism too. Trivially, id_m is an algebra morphism so, if we can show that the two are induced by the same regular morphism π, then they must be equal.

To show that $m \otimes \pi$ is an h-algebra morphism, we have to show that the diagram in Fig. 20 commutes.

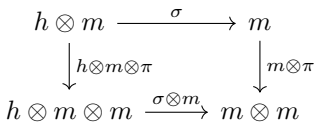

Fig. 20. $m \otimes \pi$ as an h-algebra morphism

We can redraw the two paths through Fig. 20 as two string diagrams in Fig. 21. They are equal because they can be deformed into each other.

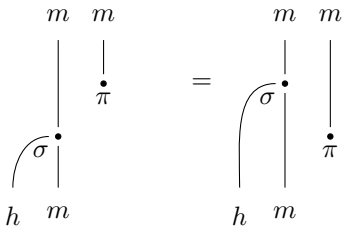

Fig. 21. String diagram showing that $m \otimes \pi$ is an algebra morphism

Therefore the composition $\mu \circ (m \otimes \pi)$ is also an h-algebra morphism. The string diagram that illustrates this fact is shown in Fig. 22.

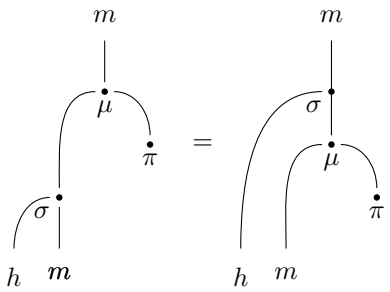

Fig. 22. String diagram showing that $\mu \circ (m \otimes \pi)$ is an algebra morphism

Since the identity h-algebra morphism is induced by π, we'd like to show that $\mu \circ (m \otimes \pi)$ is also induced by π (Fig. 23).

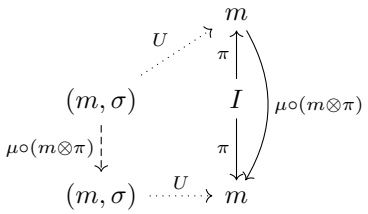

Fig. 23. Universal property of the free h-algebra generated by I, with the algebra morphism induced by π

To do that, we have to prove the universal condition in Fig. 23:

$$\mu \circ (m \otimes \pi) \circ \pi = \pi$$

This is represented as a string diagram identity in Fig. 24. We can deform this diagram by sliding the left π node up, past the right π node, and then using the left identity.

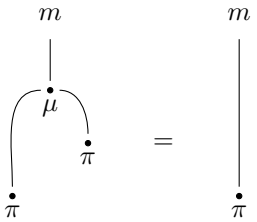

Fig. 24. Universal condition in Fig. 23.

This concludes the proof of the right identity.

The proof of associativity is very similar, so I'll just sketch it. We have to show that the two diagrams in Fig. 14 are equal. We'll use the same trick as before. We'll show that they are both algebra morphisms. Their source is a free algebra generated by $m \otimes m$ (see Fig. 25—the other diagram has $\mu \circ (\mu \otimes m)$ replaced by $\mu \circ (m \otimes \mu)$). The universal condition follows from the unit law for m. Associativity condition:

$$\mu \circ (\mu \otimes m) = \mu \circ (m \otimes \mu)$$

will then follow from the uniqueness of the universal construction.

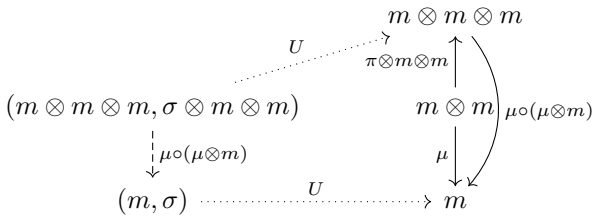

Fig. 25. One part of associativity as an h-algebra morphism

You can easily convince yourself that showing that something is an h-algebra morphism can be done by first attaching the h leg to the left of the string diagram and then sliding it to the top of the diagram, as illustrated in Fig. 26. This can be accomplished by repeatedly using the fact that μ is an h-algebra morphism.

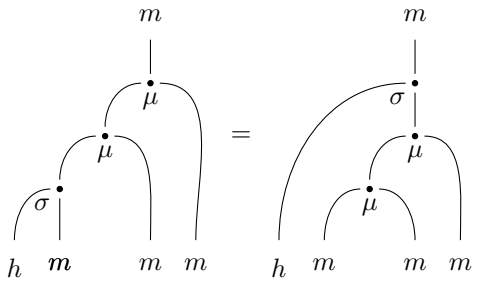

Fig. 26. String diagram showing that one of the associativity diagrams is an h-algebra morphism

The same process can be applied to the second associativity diagram, thus completing the proof.

■

For Haskell programmers, recall from the previous post our construction of the free h-algebra generated by k and the derivation of the algebra morphism g from it to the internal-hom algebra:

```
g :: Expr -> (k -> n)
g () = f
g a  = \k -> nu (a, f k)
g (a, b) = \k -> nu (a, nu (b, f k))
.dot.dot.dot
```

In the current proof we have replaced k with m, which generalizes the list of hs, f became id, and ν is a function that prepends an element to a list. In other words, g concatenates its list-argument in front of the second list, and it does it in the correct order.

C. Free Monoid

The monoid whe have just constructed from the free algebra is a free monoid. As we did with free algebras, instead of using the free/forgetful adjunction to prove it, we'll use the free-object universal construction.

Theorem 1: Free Monoid.

The monoid (m, π, μ) is a *free* monoid generated by h, with a universal mapping given by $u = \sigma \circ (h \otimes \pi)$:

$$u \colon h \xrightarrow{h \otimes \pi} h \otimes m \xrightarrow{\sigma} m$$

That is, for any monoid (n, η, ν) and any morphism $s \colon h \to n$, there is a unique *monoid morphism* t from (m, π, μ) to (n, η, ν) such that the universal condition holds:

$$t \circ u = s$$

(see Fig. 27).

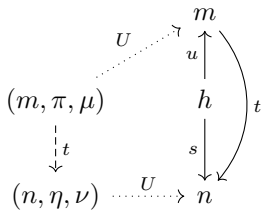

Fig. 27. Free monoid diagram

Proof: Recall that (m, σ) is a free h-algebra generated by I. It turns out that any monoid (n, η, ν), for which there is a morphism $s \colon h \to n$, is automatically a carrier of an h-algebra. We construct its structure map λ by combining s with monoid multiplication ν:

$$\lambda \colon h \otimes n \xrightarrow{s \otimes n} n \otimes n \xrightarrow{\nu} n$$

We can use n's monoidal unit η to insert I into n. Because (m, σ) is a free h-algebra, there is a unique algebra morphism, let's call it t, from it to (n, λ), which is induced by η, such that $t \circ \pi = \eta$ (see Fig. 28). We want to show that this algebra morphism is also a *monoid morphism*. Furthermore, if we can show that this is the *unique* monoid morphism induced by s, we will have a proof that m is a free monoid.

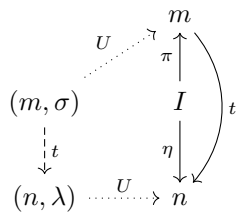

Fig. 28. Algebra morphism between monoids

Since t is an algebra morphism, the rectangle in Fig. 29 commutes.

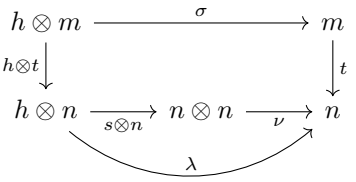

Fig. 29. t is an h-algebra morphism

Let's redraw it as an identity of string diagrams, Fig. 30. We'll make use of it in a moment.

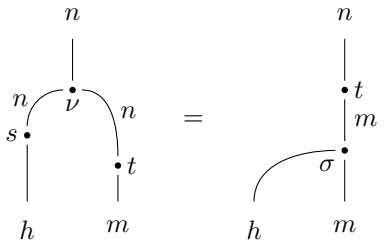

Fig. 30. t is an h-algebra morphism

Going back to Fig. 27, we want to show that the universal condition holds, which means that we want the diagram in Fig. 31 to commute (I have expanded the definition of u).

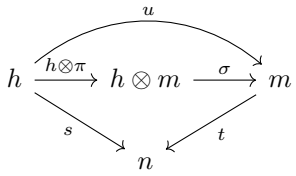

Fig. 31. Free monoid universal condition

In other words we want show that the following two string diagrams are equal:

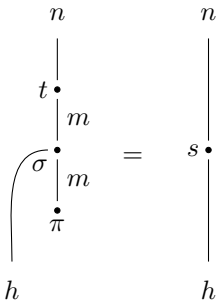

Fig. 32. Free monoid universal condition

Using the identity in Fig. 30, the left hand side can be rewritten as:

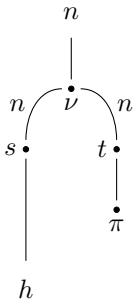

Fig. 33. Step 1 in transforming Fig 32

The right leg can be shrunk down to η using the universal condition in Fig. 28:

$$t \circ \pi = \eta$$

which, incidentally, also expresses the fact that t preserves the monoidal unit.

Finally, we can use the right unit law for the monoid n, Fig. 34,

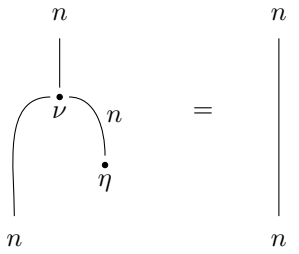

Fig. 34. Right unit law for monoid n

to arrive at the right hand side of the identity in Fig. 32. This completes the proof of the universal condition in Fig. 27.

Now we have to show that t is a full-blown monoid morphism, that is, it preserves multiplication (Fig. 35).

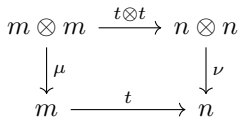

Fig. 35. Preservation of multiplication

The corresponding string diagrams are shown in Fig. 36.

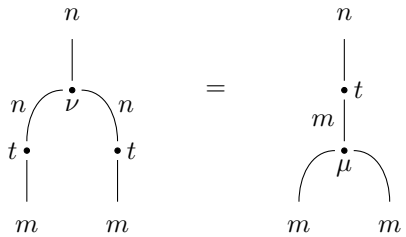

Fig. 36. Preservation of multiplication

Let's start with the fact that $m \otimes m$ is the free h-algebra generated by m. We will show that the two paths through the diagram in Fig. 35 are both h-algebra morphisms, and that they are induced by the same regular morphism $t\colon m \to n$. Therefore they must be equal.

The bottom path in Fig. 35, $t \circ \mu$, is an h-algebra morphism by virtue of being a composition of two h-algebra morphisms. This composite is induced by morphism t in the diagram Fig. 37.

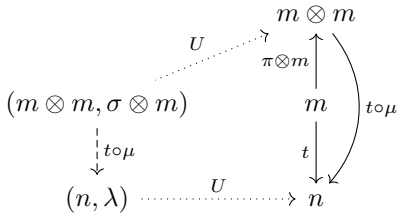

Fig. 37. h-algebra morphism $t \circ \mu$

The universal condition in Fig. 37 follows from the diagram in Fig. 38, which follows from the left unit law for the monoid (m, π, μ).

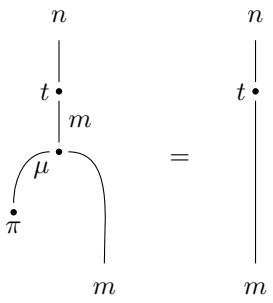

Fig. 38. Universal condition in Fig. 37

We want to show that the top path in Fig. 35 is also an h-algebra morphism, that is, the diagram in Fig. 39 commutes.

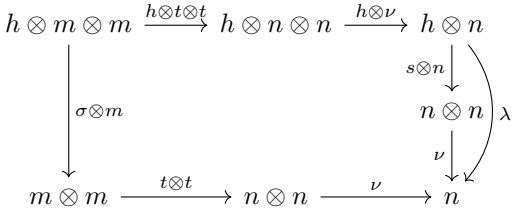

Fig. 39. h-algebra morphism diagram for $\nu \circ (t \otimes t)$

We can redraw this diagram as a string diagram identity in Fig. 40.

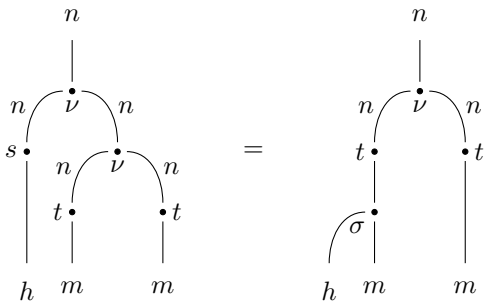

Fig. 40. h-algebra morphism diagram for $\nu \circ (t \otimes t)$

First, let's use the associativity law for the monoid n to transform the left hand side. We get the diagram in Fig. 41.

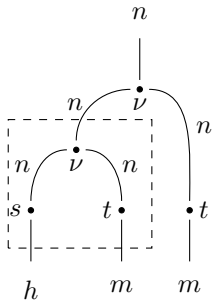

Fig. 41. After applying associativity, we can apply Fig. 30

We can now apply the identity in Fig. 30 to reproduce the right hand side of Fig. 40.

We have thus shown that both paths in Fig. 35 are algebra morphisms. We know that the bottom path is induced by morphism t. What remains is to show that the top path, which is given by $\nu \circ (t \otimes t)$ is induced by the same t. This will be true, if we can show the universal condition in Fig. 42.

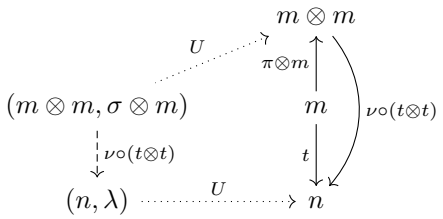

Fig. 42. h-algebra morphism $\nu \circ (t \otimes t)$

This universal condition can be expanded to the diagram in Fig. 43.

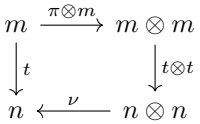

Fig. 43. Universal condition in Fig. 42

Here's the string diagram that traces the path around the square (Fig. 44).

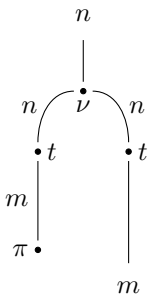

Fig. 44. Path around Fig. 43 as a string diagram

First, let's use the preservation of unit by t, Fig. 45, to shrink the left leg,

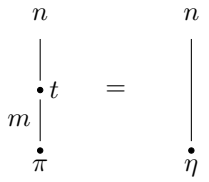

Fig. 45. Preservation of unit by t

and follow it with the left unit law for the monoid (n, η, ν) (Fig. 46). The result is that Fig. 44 shrinks to the single morphism t, thus making Fig. 43 commute.

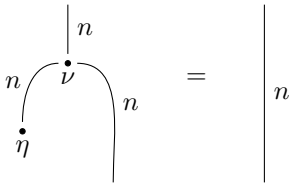

Fig. 46. Left unit law for the monoid n

This completes the proof that t is a monoid morphism.

The final step is to make sure that t is the *unique* monoid morphism from m to n. Suppose that there is another monoid morphism t' (replacing t in Fig. 28). If we can show that t' is also an h-algebra morphism induced by the same η, it will have to, by universality, be equal to t. In other words, we have to show that the diagram in Fig. 28 also works for t'. Our assumptions are that both t and t' are monoid morphisms, that is, they preserve unit and multiplication; and they both satisfy the universal condition in Fig 27. In particular, t' satisfies the condition in Fig. 47.

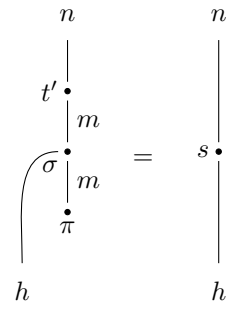

Fig. 47. Free monoid universal condition for t' as a string diagram

Notice that, in the first part of the proof, we started with an h-algebra morphism t and had to show that it's a monoid morphism. Now we are going in the opposite direction: we know that t' is a monoid morphism, and have to show that it's an h-algebra morphism, and that the universal condition in Fig. 28

$$t' \circ \pi = \eta$$

holds. The latter simply restates our assumption that t' preserves the unit.

To show that t' is an algebra morphism, we have to show that the diagram in Fig 48 commutes.

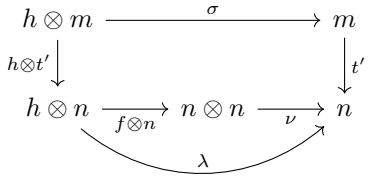

Fig. 48. t' as an h-algebra morphism

This diagram may be redrawn as a pair of string diagrams, Fig 49.

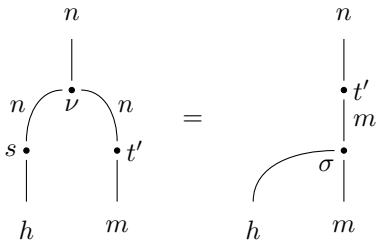

Fig. 49. t' as an algebra morphism

The proof of this identity relies on redrawing string diagrams using the identities in Figs. 12, 19, 36, and 47. Before we continue, you might want to try it yourself. It's an exercise well worth the effort.

We start by expanding the s node using the diagram in Fig. 47 to get Fig. 50.

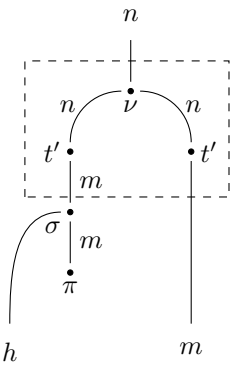

Fig. 50. After expanding the left leg of the diagram in Fig. 49, we can apply preservation of multiplication by t'.

We can now use the preservation of multiplication by t' to obtain Fig 51.

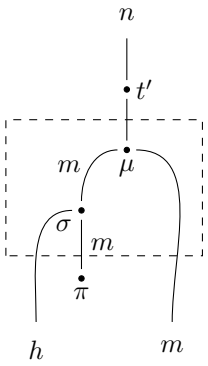

Fig. 51. Applying the fact that μ is an h-algebra morphism

Next, we can use the fact that μ is an h-algebra morphism, Fig. 19, to slide the σ node up, and obtain Fig. 52.

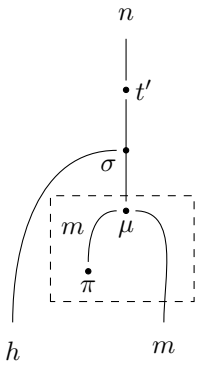

Fig. 52. Applying left unit law

We can now use the left unit law for the monoid m:

$$\mu \circ (\pi \otimes m) = id$$

as illustrated in Fig. 12, to arrive at the right hand side of Fig. 49.

This concludes the proof that t' must be equal to t.

■

D. Conclusion

To summarize, we have shown that the free monoid can be constructed from a free algebra of the functor $h \otimes -$. This is a very general result that is valid in any monoidal closed category. Earlier we've seen that this free algebra is also the initial algebra of the list functor $I + h \otimes -$. The immediate consequence of this theorem is that it lets us construct free monoids in functor categories with interesting monoidal structures. In particular, we get a free monad as a free monoid in the category of endofunctors with functor composition as tensor product. We can also get a free applicative, or free lax monoidal functor, if we define the tensor product as Day convolution—the latter can be also constructed in the profunctor category.

References

[1] Kelly, G.M. "A Unified Treatment of Transfinite Constructions for Free Algebras, Free Monoids, Colimits, Associated Sheaves, and so On." Bulletin of the Australian Mathematical Society, vol. 22, no. 01, 1980, p. 1., doi:10.1017/s0004972700006353.