String Diagrams for Free Monads

(Functional Pearl)

Maciej Piróg
KU Leuven, Belgium
maciej.pirog@cs.kuleuven.be

Nicolas Wu
University of Bristol, UK
nicolas.wu@bristol.ac.uk

Abstract

We show how one can reason about free monads using their universal properties rather than any concrete implementation. We introduce a graphical, two-dimensional calculus tailor-made to accommodate these properties.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming; D.3.2 [*Programming Languages*]: Language Classifications – Applicative (functional) languages

Keywords monad, free monad, universal property, string diagram, distributive law

1. Introduction

Free monads generated by endofunctors generalise monads of terms over signatures. They are indispensable for representing syntax of domain-specific languages, interpreted later by means of folding; a good example is the framework of algebraic effects (syntax) and handlers (interpreters). Since such patterns are becoming increasingly popular in functional programming, we desire reliable principles to program with and reason about free monads.

One way to proceed is to imagine that free monads are just generalised trees, while operations on free monads are just generalised operations on trees. This is an *intensional* approach, based on the way free monads are implemented. In this pearl, we advocate an *extensional* approach: we focus on the intended properties of monads that model interpretable syntax. Such properties relate a free monad to its generating endofunctor and other monads without any reference to its internal structure or implementation. And the familiar tree-like data structure just happens to have these properties.

The extensional approach is characteristic to category theory, which has been widely adopted for reasoning about functional programs. Here, we abstract to 2-categorical reasoning, which means that the properties that we tackle are all about functors and natural transformations (polymorphic functions), but do not mention objects (base types) and morphisms (functions between base types). We also do not need any additional categories, although in the literature reasoning about free monads is often based on a connection between the base category and the category of F-algebras. This framework turns out to be expressive enough to describe the intended use-cases of free monads, while providing exceptionally elegant reasoning principles.

Moreover, this level of abstraction is amenable to pictorial, two-dimensional representations. Graphical calculi focus reasoning on the essence of the proof, as some boilerplate equalities correspond to intuitive, topological invariants of geometric shapes, while others are entirely built into the notation. In this pearl, we show such a calculus, *string diagrams*, and extend it to accommodate free monads.

1.1 Free monads in Haskell

Although we are not really concerned with the implementation of free monads, to gain some intuition, we begin with the following Haskell definition:

```
data Free f a = Var a | Op (f (Free f a))
instance Functor f ⇒ Monad (Free f) where
  return  =  Var
  Var x  >>= k = k x
  Op f   >>= k = Op (fmap (>>=k) f)
```

Thus, a value of the type *Free f a* is either a variable, or a 'term' constructed with an operation from f. The monadic structure is given by embedding of variables (*return*) and substitution in variables (>>=).

For an example of how one can see free monads in a more extensional way, consider the following two functions, which will form key building blocks of our notation:

```
emb :: Functor f ⇒ f a → Free f a
emb f = Op (fmap return f)
interp :: (Functor f, Monad m) ⇒ (∀x.f x → m x)
                               → Free f a → m a
interp i (Var a) = return a
interp i (Op f) = i f >>= interp i
```

It is the case that the function *interp* is a monad morphism, and that *interp i* ∘ *emb* = *i* for all functions *i* of the appropriate type. Moreover, *interp i* is the *only* monad morphism with this property.

We can turn this around, and say that a monad F^* is a *free* monad generated by a functor F if there exist functions $emb :: F\ a \to F^*\ a$ and $interp :: Monad\ t \Rightarrow (\forall x.F\ x \to t\ x) \to F^*\ a \to t\ a$ such that for all monads T and functions $i :: F\ a \to T\ a$, the value *interp i* is a unique monad morphism g with the property $g \circ emb = i$. As shown in Section 3, many interesting functions can be expressed only in terms of *emb* and *interp*, while the uniqueness property above turns out to be powerful enough to prove non-trivial results about such functions.

The notation we introduce in this paper makes it easy to work with proofs that involve free monads. Consider, for instance, the *FreeT* monad transformer, defined in Haskell by:

```
newtype FreeT f m a = FreeT {runFreeT :: m (FreeF f m a)}
data FreeF f m a = VarF a | OpF (f (FreeT f m a))
```

It should be obvious from the structure of *FreeT f m* that it is closely connected to *Free f*. What is not so obvious is that this is also a monad, and indeed proving that this is the case using conventional tools is no trivial matter. As we will see, universal properties provide the machinery to make such a task manageable.

1.2 Overview

In this pearl, we describe a number of properties similar to the one above. Each property defines a class of free monads, and allows us to construct programs and reason about them. In particular, we discuss the following:

- A monad F^* is a *free* monad generated by an endofunctor F if natural transformations $F \to M$ (for a monad M) lift to monad morphisms $F^* \to M$.
- A free monad F^* is *distributable* if natural transformations $FG \to GH$ lift to distributive laws $F^*G \to GH^*$.
- A free monad F^* is *foldable* if natural transformations $FG \to GM$ (for a monad M) lift to distributive laws $F^*G \to GM$.
- Distributable and foldable free monads can also be *uniform*, which means that they satisfy some additional equational properties.

The data structure *Free* satisfies all of these properties, so one can readily use them to reason about functional programs. In the category of sets and other suitably well-behaved categories (cocartesian complete), all of the listed properties are equivalent. Freeness, distributability, and foldability are examples of so-called *universal properties*. Such properties lie at the heart of different frameworks used for reasoning about functional programs, including the Bird–Meertens (1987) formalism and initial algebra semantics (Meijer et al. 1991). They are often accompanied by derived equational laws, usually named *cancellation*, *reflection*, *fusion*, etc., which can be directly applied to reason about programs.

While free monads are instances of the more general mathematical definition of freeness, to the authors' best knowledge the other properties were not previously discussed as separate reasoning principles—although liftings of natural transformations to distributive laws appear in the literature as a consequence of a yet another, stronger property: *algebraic freeness*. We equip each discussed property with a graphical notation in the framework of string diagrams.

2. Endofunctors, transformations, and diagrams

In this section, we introduce the calculus that we use to define and reason about free monads. It is based on category-theoretic notions, but a reader not interested in category theory should not worry: we introduce everything from scratch as a simple axiomatic calculus, and we do not even need the definition of a category. The reader will easily recognise that the given axioms hold for the constructs used in functional programming.

2.1 The axiomatic calculus of natural transformations

To be in accord with the category-theoretic terminology, we call Haskell functors *endofunctors*. For our purposes, we treat them as atomic entities usually denoted $F, G, H, \ldots$. All we assume about them is that they form a monoid. This means that there are two operations on endofunctors—*composition*, denoted by juxtaposition, and the *identity endofunctor*, denoted Id—which satisfy the following equations for all endofunctors F, G, and H:

$$\begin{aligned} F(GH) &= (FG)H \\ F\,\mathsf{Id} &= F \\ \mathsf{Id}\,F &= F \end{aligned}$$

Polymorphic functions are modelled by *natural transformations*. A natural transformation is an atomic being with a *type*, that is, a pair of endofunctors written as $F \to G$. For example, we can denote a natural transformation as $f : FG \to GJH$ (since the composition of endofunctors is associative, we can skip parentheses in compositions of three or more endofunctors). There are three operations on natural transformations:

- *Vertical composition*: Given two natural transformations $f : F \to G$ and $g : G \to H$, we can form a new natural transformation denoted as $g \cdot f : F \to H$.
- *Horizontal composition*: Given two natural transformations $f : F \to F'$ and $g : G \to G'$, we can form a new natural transformation denoted $fg : FG \to F'G'$.
- *Identity natural transformation*: There is one identity natural transformation for each endofunctor F, which we denote with the overloaded notation $\mathsf{id} : F \to F$, or, when the context is clear, by simply F alone.

We require the following axioms:

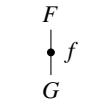
$$\begin{aligned} (f \cdot g) \cdot h &= f \cdot (g \cdot h) \\ f \cdot \mathsf{id} &= f \\ \mathsf{id} \cdot f &= f \\ f(gh) &= (fg)h \\ f'g' \cdot fg &= (f' \cdot f)(g' \cdot g) \end{aligned}$$

The horizontal composition of a natural transformation $f : F \to F'$ with the identity natural transformation for a functor G (that is, $f\mathsf{id} : FG \to F'G$) is often denoted as fG. It is also the case in the other direction: we define Gf to mean $\mathsf{id}f : GF \to GF'$. Note that instantiating the last axiom with $f = f' = \mathsf{id}$, we obtain $Fg' \cdot Fg = F(g' \cdot g)$, which is the familiar law for functors in Haskell, modulo the fact that g and g' are natural transformations.

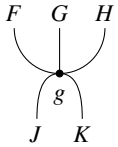
2.2 String diagrams

We reason about endofunctors and natural transformations using a two-dimensional notation called *string diagrams*. We briefly introduce the notation here, for a more detailed exposition see Curien (2008) or Hinze and Marsden (2016).

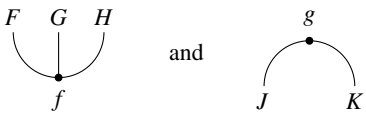
Each string diagram encodes a natural transformation. It consists of a number of lines (*strings*), each representing an endofunctor. Natural transformations are denoted as black dots. For example, a natural transformation $f : F \to G$ is denoted as follows:



Natural transformations that take compositions of endofunctors to compositions of endofunctors gather a number of strings. For example, a natural transformation $g : FGH \to JK$ is drawn as follows:

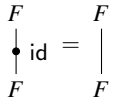


We do not draw strings for the identity endofunctor. Thus, we draw natural transformations $f : FGH \to \mathsf{Id}$ and $g : \mathsf{Id} \to JK$ respectively as:

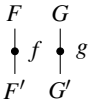


For $f : F \to G$ and $g : G \to H$, the vertical composition $g \cdot f : F \to H$ is denoted by putting g below of f:

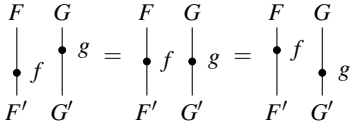
We put names of the endofunctors in between two natural transformations, like G in the example above. We do not always do that, as the types of strings can be read from the types of the involved natural transformations. Also, thanks to the associativity of vertical composition, we can simply put natural transformations one after another on a string, without worrying about parentheses. We also do not put identity natural transformations on strings, that is:



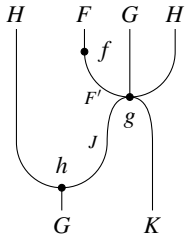
Horizontal composition is written as the name suggests. For example, consider $f : F \to F'$ and $g : G \to G'$. Then, $fg : FG \to F'G'$ is drawn as:



The axioms guarantee that we can move the dots up and down the strings, as long as we do not swap their order. For example, considering the natural transformations above, we have $fG' \cdot Fg = fg = F'g \cdot fG$. These correspond to the following equalities between string diagrams:



For a more complicated example, consider natural transformations $f : F \to F'$, $g : F'GH \to JK$, and $h : HJ \to G$. The natural transformation $hK \cdot Hg \cdot HfGH : HFGH \to GK$ is drawn as:

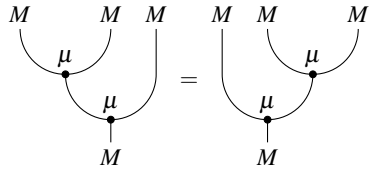


Reading diagrams back into their representation as terms is also possible: we imagine a horizontal line that sweeps from the top of the diagram, stopping at each natural transformation, where we observe the functors and natural transformations that cross this line. The observations produce subterms starting from the right, and separated by vertical composition.

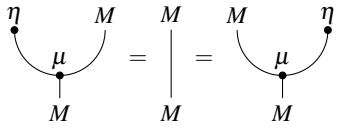
2.3 Monads

As an example of expressing natural transformations in the form of string diagrams, we present monads. In Haskell, monads are usually introduced in terms of two operations: $return :: a \to m\ a$ and $(\gg\!=) :: m\ a \to (a \to m\ b) \to m\ b$ called *bind*. An alternative description replaces bind with $join :: m\ (m\ a) \to m\ a$, where $mx \gg\!= f = join \circ map\ f \circ mx$, and $join\ mmx = mmx \gg\!= id$. To follow more traditional terminology, we use η, called the *unit*, and μ, called the *multiplication*, to refer to the categorical counterparts of *return* and *join* respectively.

Formally, a *monad* is an endofunctor M together with two natural transformations, $\eta : \mathsf{Id} \to M$ and $\mu : MM \to M$, such that $\mu \cdot \mu M = \mu \cdot M\mu$ (associativity) and $\mu \cdot \eta M = \mathsf{id} = \mu \cdot \eta M$ (left and right unit). Using string diagrams, these equations can be drawn respectively as follows:



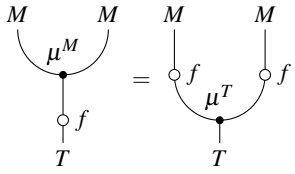
(1)



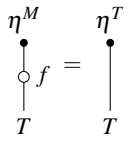
(2)

We always denote the unit and the multiplication of a monad M as η and μ respectively. When there is more than one monad in the context, we use superscripts to assign natural transformations to the corresponding monads, for example η^M and μ^M. However, since the monad can be also read from the type, we omit the superscripts later on in the text.

The next important concept are *monad morphisms*. They are natural transformations between two monads M and T that preserve the unit and the multiplication of M. Monad morphisms appear in functional programming in the context of monad transformers, where lifting of a computation in the base monad to the computation in the transformer needs to respect some equalities—these are the defining equalities of monad morphisms. Formally, a monad morphism is a natural transformation $f : M \to T$ such that $f \cdot \mu^M = \mu^T \cdot ff$ and $f \cdot \eta^M = \eta^T$. Since monad morphisms play a special role in our development, we denote them slightly differently than other natural transformations: as white circles. This allows us to immediately recognise which natural transformations are monad morphisms, without a need to look for the assumptions made in the text. Thus, the equalities for monad morphisms can be shown using string diagrams as follows:



(3)

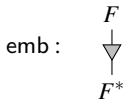


(4)

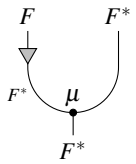
3. Free monads, categorically

A *free monad* generated by an endofunctor F consists of a monad, which we denote F^*, together with a natural transformation $\mathsf{emb} : F \to F^*$ that satisfy a certain property discussed below. Intuitively, F generalises signatures, that is, collections of operations, and emb embeds the operations in the free monad. In the concrete case of the monad of terms (when the base category is the category of sets, and F is a proper signature), emb allows us to see an expression of the

form $f(x_1,\ldots,x_n)$ as a term with one operation applied to variables. In the string diagram notation, we denote emb as a gray triangle:

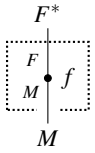
$\mathsf{emb}:$ 

In functional programming, instances of free monads are often obtained with a constructor, called *Op* in the introduction, which has the type $FF^* \to F^*$. It can be encoded using emb as $\mu \cdot \mathsf{emb}\, F^*$. With string diagrams, it can be drawn as:



We say that a monad M *supports* an endofunctor F if there exists a natural transformation $f : F \to M$. Thus, there can be many ways in which M supports F, in fact, as many as there are natural transformations of this type. What makes free monads *free* in the mathematical sense is the property that given any other monad M that also supports F, we can *interpret* a value of F^* as an M-computation. Formally, we require that free monads have the following property: for any monad M and a natural transformation $f : F \to M$, there exists a unique monad morphism $\lfloor f \rfloor : F^* \to M$ (an *interpretation*) with the property that $f = \lfloor f \rfloor \cdot \mathsf{emb}$.

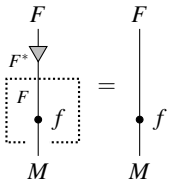
In the string diagram notation, we denote the morphism $\lfloor\text{-}\rfloor$ as a dotted box. On the way in, the endofunctor 'changes' from F^* to F. Since the codomain of f is the same as the codomain of $\lfloor f \rfloor$, we leave a hole on the way out of the box to stress that the endofunctor does not change:

$\lfloor f \rfloor:$ 

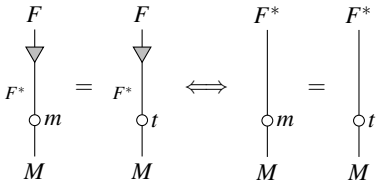
Note that except for the top bar, the shape of the box resembles the symbol $\lfloor\ \rfloor$, hence the chosen notation.

3.1 Equational laws

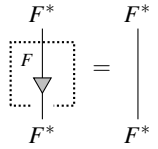
The property $f = \lfloor f \rfloor \cdot \mathsf{emb}$ is called *cancellation* (intuitively, emb cancels $\lfloor\text{-}\rfloor$). Using string diagrams, it can be depicted as follows:

cancellation:  (5)

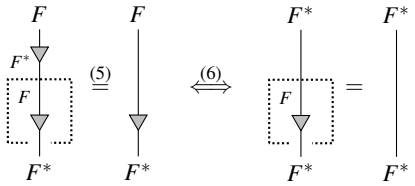
The uniqueness of $\lfloor f \rfloor$ gives us that two monad morphisms $m,t : F^* \to M$ are equal if and only if $m \cdot \mathsf{emb} = t \cdot \mathsf{emb}$, which we can depict as follows:

 (6)

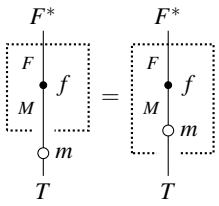
This allows us to prove some useful equational laws. *Reflection* states that $\lfloor \mathsf{emb} \rfloor = \mathsf{id}$:

reflection:  (7)

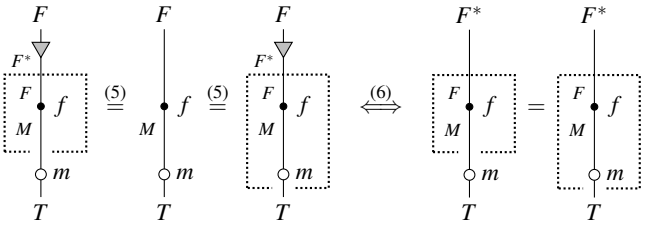
It can be shown as follows. Since both sides are monad morphisms, it is enough to show that they are equal when composed with emb:



Another law is called *fusion*. It states that for a natural transformation $f : F \to M$ and a monad morphism $m : M \to T$, it is the case that $m \cdot \lfloor f \rfloor = \lfloor m \cdot f \rfloor$. Pictorially, it means that a monad morphism can slide in and out of the box:

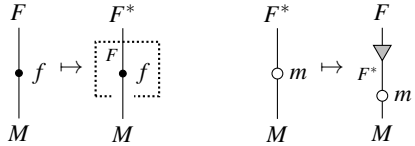
fusion:  (8)

We prove fusion as follows. Both sides are monad morphisms (the fact that the composition of two monad morphisms is a monad morphism can be easily verified using string diagrams), so, again, it is enough to compare their respective compositions with emb:

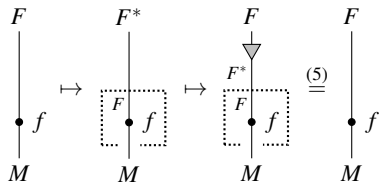


3.2 Example: natural transformations vs monad morphisms

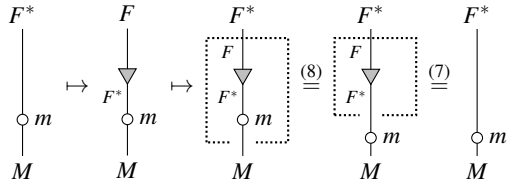
As an application of the equational laws described above, we prove the following property: given an endofunctor F and a monad M, natural transformations of the type $f : F \to M$ are in 1-1 correspondence with monad morphisms of the type $m : F^* \to M$. This correspondence is given by $f \mapsto \lfloor f \rfloor$ in one direction, and $m \mapsto m \cdot \mathsf{emb}$ in the other. Pictorially:



To see that the two mappings are mutual inverses, consider the following:

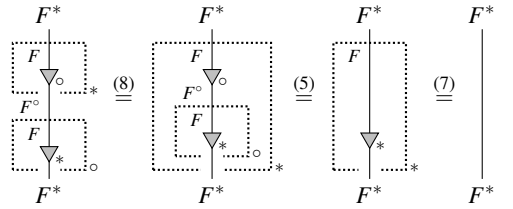


In the other direction:



3.3 Example: 'a' free monad vs 'the' free monad

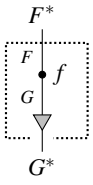
As another application of the equational laws, we show that an endofunctor generates at most one free monad up to isomorphism. Let F^* with emb_* and F° with $\mathsf{emb}_\circ$ be two free monads generated by an endofunctor F. We denote the interpretations as $\lfloor\text{-}\rfloor_*$ and $\lfloor\text{-}\rfloor_\circ$ respectively. We define two monad morphisms: $\lfloor\mathsf{emb}_\circ\rfloor_* : F^* \to F^\circ$ and $\lfloor\mathsf{emb}_*\rfloor_\circ : F^\circ \to F^*$. To see that they are mutual inverses, consider the following:



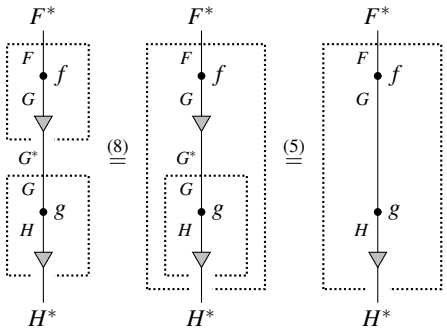
The above proves that $\lfloor\mathsf{emb}_*\rfloor_\circ \cdot \lfloor\mathsf{emb}_\circ\rfloor_* = \mathsf{id}$. The other direction, $\lfloor\mathsf{emb}_\circ\rfloor_* \cdot \lfloor\mathsf{emb}_*\rfloor_\circ = \mathsf{id}$, is similar.

3.4 Example: renaming is functorial

An important operation on free monads is *renaming*: a natural transformation $f : F \to G$ can be spread across the entire structure to transform F^* to G^*. We define this operation as the monad morphism $\lfloor\mathsf{emb} \cdot f\rfloor$, or, using a string diagram:



We denote this operation as $f^* : F^* \to G^*$, as if $(\text{-})^*$ was a functor from the category of endofunctors that generate free monads and natural transformations, to the category of monads and monad morphisms. Indeed, as another application of the introduced equational laws, we show that $(\text{-})^*$ satisfies the equalities defining a functor: it preserves composition (that is, $g^* \cdot f^* = (g \cdot f)^*$) and identities ($\mathsf{id}^* = \mathsf{id}$). The former can be shown as follows:



The latter is simply reflection (7).

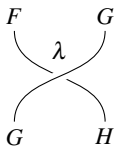
4. Distributable free monads

While freeness is a powerful property, it does not allow us to capture all operations that we want to perform on free monads in functional programming. Thus, in this and the next section, we introduce families of free monads with some additional properties. In practice, this is admissible, since in suitably well-behaved categories (cocartesian complete, such as the category of sets), as well as in Haskell, all of the additional properties follow from freeness.

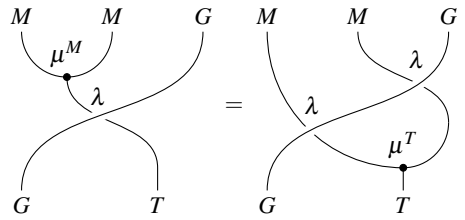
In this section, we describe the property that a natural transformation lifts to a distributive law of a free monad over an endofunctor. As an application, we use it to construct a generalisation of the resumption monad, also known as the free monad transformer, originally proposed by Moggi (1989). We use string diagrams to prove that it is indeed a monad.

4.1 Distributive laws

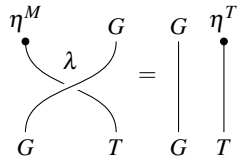
We focus on natural transformations of the type $FG \to GH$, for some endofunctors F, G, and H. Traditionally, we denote them with the Greek letter λ (not to be confused with function abstraction from λ-calculus) and draw them as crossings of strings, where the continuous one represents G:



When some of the involved functors are monads, we also consider such natural transformations with additional properties. For example, we call a natural transformation $\lambda : MG \to GT$ a *distributive law of a monad over an endofunctor* if M and T are monads, and it is the case that $\lambda \cdot \mu^M G = G\mu^T \cdot \lambda T \cdot M\lambda$ and $\lambda \cdot \eta^M G = G\eta^T$. We can depict these equations as follows:



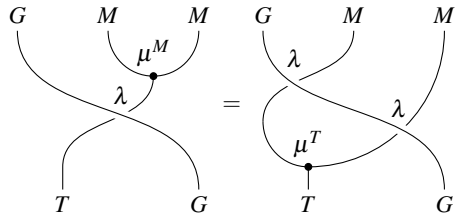
(9)



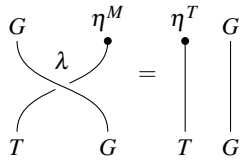
(10)

In the literature, distributive laws of a monad over an endofunctor are usually defined for $M = T$. We make heavy use of the additional generality in Section 5.

We also define the mirror image of the above: a *distributive law of an endofunctor over a monad*. It is a natural transformation $\lambda : GM \to TG$ such that $\lambda \cdot G\mu^M = \mu^T G \cdot M\lambda \cdot \lambda M$ and $\lambda \cdot G\eta^M = \eta^T G$:

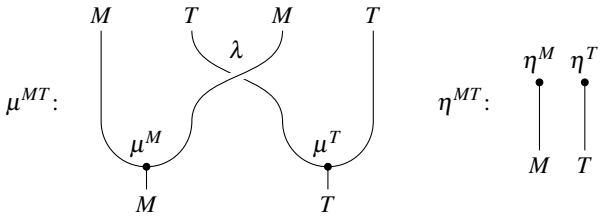


(11)



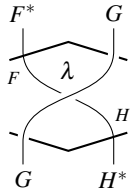
(12)

One point of studying distributive laws is that they give us a way to compose monads. Consider two monads M and T, and a natural transformation $\lambda : TM \to MT$ that is a distributive law of a monad over an endofunctor, as well as a distributive law of an endofunctor over a monad (in such a case, we say that λ is a distributive law *between monads*). Then, the composition MT is a monad with the monadic structure given as $M\mu^T \cdot \mu^M TT \cdot T\lambda M$ and $M\eta^T \cdot \eta^M$:



4.2 Liftings of natural transformations

As discussed in Section 3, a monad morphism $\lfloor f \rfloor : F^* \to M$ is uniquely determined by a natural transformation $f : F \to M$. We extend this idea to distributive laws. We say that a free monad F^* is *distributable* if for every natural transformation $\lambda : FG \to GH$ there exists a unique distributive law $\langle \lambda \rangle : F^*G \to GH^*$ (the *lifting* of λ) of a monad over an endofunctor such that $G\mathsf{emb} \cdot \lambda = \langle \lambda \rangle \cdot \mathsf{emb}\, G$, which we call *cancellation*. In string diagrams, we denote $\langle \lambda \rangle : F^*G \to GH^*$ as follows:

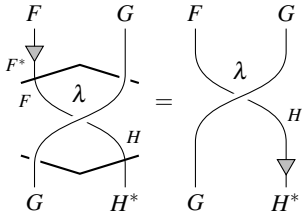


Obviously, the horizontal angle brackets are supposed to mimic the notation $\langle\ \rangle$. Since the type of the endofunctor over which we distribute (G) does not change, we leave little holes in the top-right and bottom-left arms to ensure that the string continuously enters and leaves the bracket. Moreover, if more strings enter one of the arms, we sometimes slightly shift the tips of the brackets, as in the right-hand side of the equation (16).

4.3 Equational laws of liftings

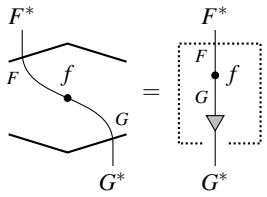
Similarly to free monads, distributable free monads enjoy their own versions of the cancellation, reflection, and fusion laws. The condition $G\mathsf{emb} \cdot \lambda = \langle \lambda \rangle \cdot \mathsf{emb}\, G$ (*cancellation*) can be depicted as follows:

cancellation:



(13)

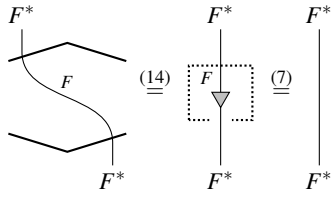
It is easy to verify that a distributive law of a monad over the identity endofunctor is simply a monad morphism. In such a case, for a natural transformation $f : F \to G$ (alternatively written as $f : F\,\mathsf{Id} \to \mathsf{Id}\,G$), the cancellation property above becomes $\mathsf{emb} \cdot f = \langle f \rangle \cdot \mathsf{emb}$. Thus, from the uniqueness of interpretation of free monads, we obtain:



(14)

This gives us the *reflection* law for distributable free monads, $\langle \mathsf{id} \rangle = \mathsf{id}$:

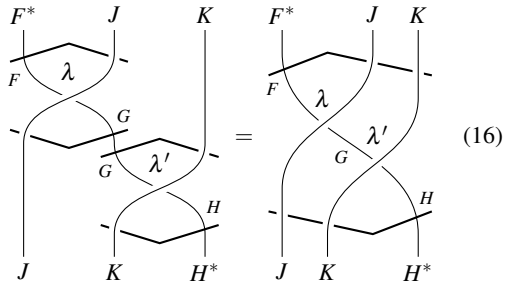
reflection:



(15)

The *fusion* law states that a composition of two liftings is a lifting of a composition. In detail, let $\lambda : FJ \to JG$ and $\lambda' : GK \to KH$ be two natural transformations. Then, $J\langle \lambda' \rangle \cdot \langle \lambda \rangle K = \langle J\lambda' \cdot \lambda K \rangle$:

fusion:



(16)

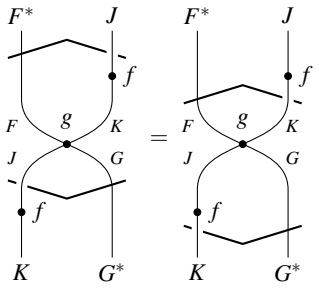
We can prove this law in two steps. First, we show that the left-hand side of the equation (16) is a distributive law of the monad F^* over the endofunctor JK. This follows from a more general lemma: every composition of distributive laws of monads over endofunctors (not necessarily of the shape $\langle \text{-} \rangle$) as in the left-hand side of (16) is also a distributive law of a monad over the composite endofunctor.

Then, we can use the uniqueness of $\langle\text{-}\rangle$: it is enough to compare the respective compositions of both sides of (16) with emb. We encourage the reader to draw the appropriate diagrams.

4.4 Uniformity

There is one more property that is satisfied by the Haskell definition of the free monad, and which turns out to be indispensable for reasoning about distributive laws. Unfortunately, it does not follow from the definition of distributable monads. Hence, we need to introduce another family of free monads: *uniform* distributable monads.

Uniformity means that the 'way' natural transformations are lifted to distributive laws does not rely on the endofunctor over which we distribute. In other words, there are two ways in which we can easily construct a distributive law out of two natural transformations $f : J \to K$ and $g : FK \to JG$. Uniformity states that they are equal. Formally, $fG^* \cdot \langle g \cdot Ff \rangle = \langle fG \cdot g \rangle \cdot F^* f$. Using string diagrams, this equality renders as follows:

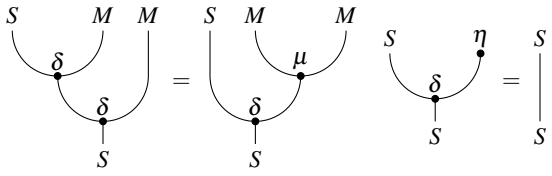
uniformity:  (17)

Uniformity is another reason for the notation $\langle\ \rangle$, since it resembles an arrow pointing in two directions, and $\langle\text{-}\rangle$ can 'slide' up and down the diagram, as shown above.

4.5 Example: the generalised resumption monad

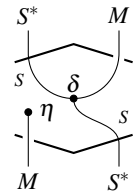
As an application of the properties described above, we reconstruct the resumption monad—more specifically, a generalisation proposed by Piróg et al. (2015). In fact, this is the solution to the puzzle we set out in the introduction: we can show that *FreeT f m* is indeed a monad.

For this, we first need to define a *right module* over a monad M. It consists of an endofunctor S and a natural transformation $\delta : SM \to S$ such that $\delta \cdot \delta M = \delta \cdot S\mu$ and $\delta \cdot S\eta = \mathsf{id}$. In string diagrams, they look like the diagrams for monads with the left-most string representing S instead of M:

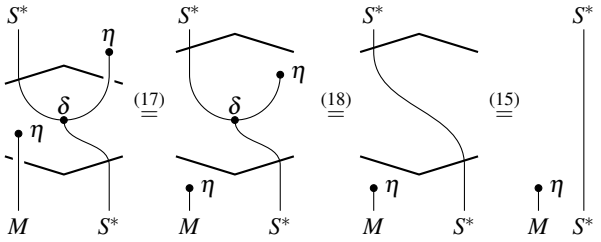
 (18)

If the uniform distributable monad S^* exists, the resumption monad is given by the composition MS^*. An important instance of this construction is the ordinary resumption monad (Moggi 1989), also known as the *free monad transformer*: given any endofunctor F, the composition FM is a right module over M with $\delta = F\mu$ (it is easy to verify that it is indeed a right module using string diagrams), and the generalised resumption monad instantiates to $M(FM)^*$. This construction appears in functional programming (Piróg and Gibbons 2012; Schrijvers et al. 2014) as *FreeT f m*, but it is not unquestionably trivial to prove that it is indeed a monad. However, using distributability and uniformity, we can reduce reasoning about the whole structure to reasoning about the local behaviour of the lifted natural transformation.

So, for a general S, we give a monadic structure to MS^* via a distributive law between monads $\beta : S^*M \to MS^*$. It is given as $\langle \eta S \cdot \delta \rangle$. Using string diagrams, it can be expressed as:

β:  (19)

We need to check that β is indeed a distributive law between monads. From the definition of $\langle\text{-}\rangle$ it follows that it is a distributive law of the monad S^* over M understood as an endofunctor. It is left to verify that it is also a distributive law of S^* as an endofunctor over the monad M. The coherence with μ is given in Figure 1. The coherence with η can be shown as follows:



Hinze and Marsden (2016) also use the monadic structure of the resumption monad as an example of reasoning with string diagrams. However, their approach is different, since they work with *algebraically free* monads, that is, monads that arise from adjunctions between a base category and categories of F-algebras. Our approach is more axiomatic and it avoids introducing other categories, which brings it closer to reasoning about functional programs.

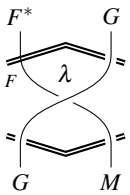
As another example that falls out of this construction, consider *MaybeT* monad transformer, which is defined in Haskell as:

$$\textbf{data}\ \mathit{MaybeT}\ m\ a = \mathit{MaybeT}\ \{\mathit{runMaybeT} :: m\ (\mathit{Maybe}\ a)\}$$

This is an instance of our theory where the functor F is the constant functor $K1$ that returns an element in 1 regardless of its input.

5. Foldable free monads

Foldable free monads are similar to distributable free monads, but they allow an arbitrary monad in the codomain of the lifted natural transformation, not necessarily a free one. Intuitively, they not only 'distribute' the natural transformation across the structure, they also multiply out the result. Formally, for an endofunctor F, we call the free monad F^* *foldable* if it satisfies the following property: given an endofunctor G, a monad M, and a natural transformation $\lambda : FG \to GM$, there exists a unique distributive law of a monad over an endofunctor $\langle\langle\lambda\rangle\rangle : F^*G \to GM$ (a *generalised fold* induced by λ) such that $\lambda = \langle\langle\lambda\rangle\rangle \cdot \mathsf{emb}\,G$. In the string diagram notation, we depict $\langle\langle\lambda\rangle\rangle$ using double brackets:



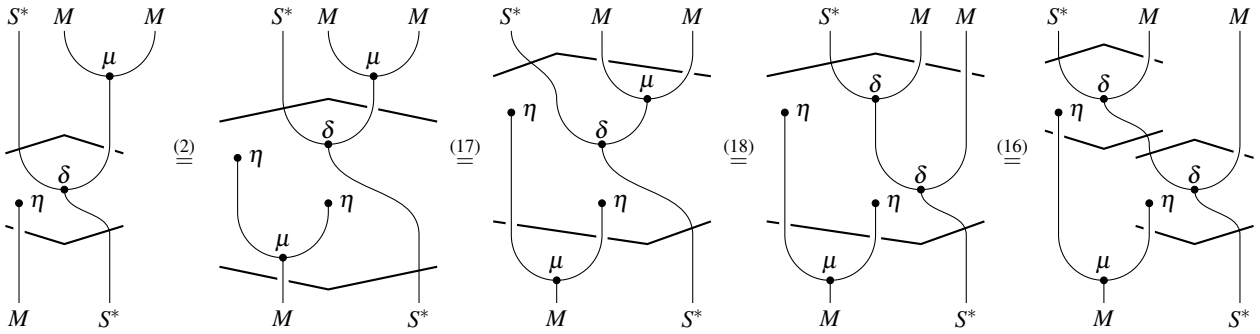


Figure 1. Coherence of β and the multiplication of M

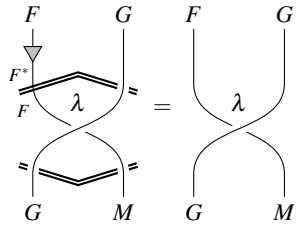
Note that there are now holes in three arms of the brackets, as the endofunctor changes only when entering via the left-hand arm of the top bracket.

We call such monads foldable, since we can obtain a generalisation of the usual fold operation over the structure by instantiating M with the identity monad Id. In such a case, the natural transformation $\lambda : FG \to G$ is the algebra, while $\langle\!\langle \lambda \rangle\!\rangle : F^*G \to G$ is the resulting fold.

5.1 Equational laws

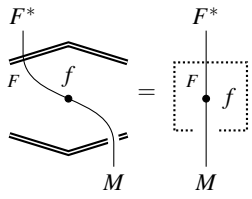
The property $\langle\!\langle \lambda \rangle\!\rangle \cdot \mathsf{emb}\, G = \lambda$ (the *cancellation* law of foldable free monads) can be depicted as follows:

cancellation:



(20)

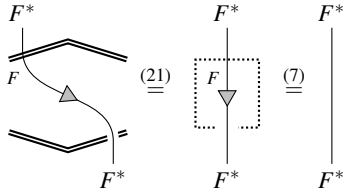
Again, we define the *reflection* law. First, we note that for a natural transformation $f : F \to M$ (that could be seen as $f : F\mathsf{Id} \to \mathsf{Id}M$), its generalised fold $\langle\!\langle f \rangle\!\rangle : F^* \to M$ is a monad morphism. Hence, from the uniqueness of $\lfloor\text{-}\rfloor$, we obtain that $\langle\!\langle f \rangle\!\rangle = \lfloor f \rfloor$:



(21)

In particular, by instantiating f with emb, we obtain $\langle\!\langle \mathsf{emb} \rangle\!\rangle = \mathsf{id}$:

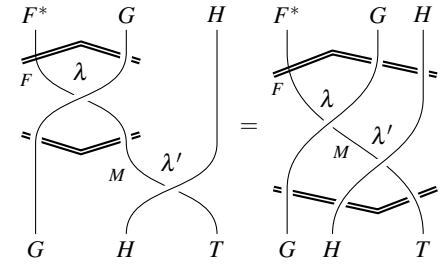
reflection:



(22)

We state the *fusion* law as follows. For a monad M, a natural transformation $\lambda : FG \to GM$, and a distributive law of a monad over an endofunctor $\lambda' : MH \to HT$, it is the case that $G\lambda' \cdot \langle\!\langle \lambda \rangle\!\rangle H = \langle\!\langle G\lambda' \cdot \lambda H \rangle\!\rangle$:

fusion:



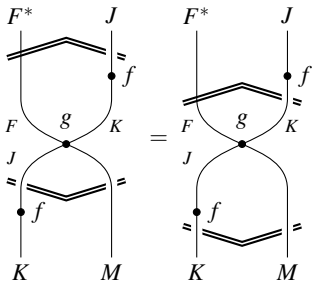
(23)

Fusion can be proven similarly to the fusion law for distributable monads. Both sides of the equation above are distributive laws of a monad over an endofunctor. Thus, it is enough to compare their compositions with emb, which agree due to the cancellation property of foldable free monads.

5.2 Uniformity

In a similar way to uniform distributable monads, we define *uniform* foldable monads. They are foldable free monads with the property that for all natural transformations $f : J \to K$ and $g : FK \to JG$, it is the case that $fM \cdot \langle\!\langle g \cdot Ff \rangle\!\rangle = \langle\!\langle fM \cdot g \rangle\!\rangle \cdot F^*f$:

uniformity:

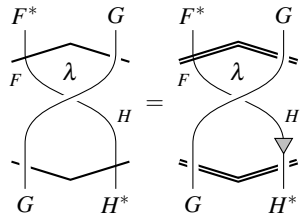


(24)

5.3 Foldability vs distributability

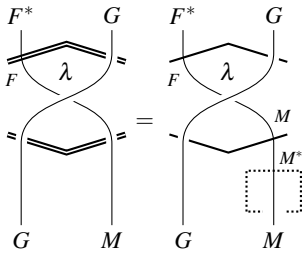
The property defining foldable free monads is not much different from the property defining distributable free monads. Indeed, the operation $\langle\!\langle\text{-}\rangle\!\rangle$ subsumes both $\lfloor\text{-}\rfloor$ and $\langle\text{-}\rangle$. For endofunctors F, G, and H, a monad M, and a natural transformation $f : F \to M$, we can express $\lfloor f \rfloor$ using $\langle\!\langle\text{-}\rangle\!\rangle$ as in the equation (21). For a natural transformation $\lambda : FG \to GH$, its lifting $\langle \lambda \rangle$ can be expressed

as $\langle\!\langle G\mathsf{emb}\cdot\lambda\rangle\!\rangle$:



(25)

On the other hand, foldable monads are not much more powerful than distributable monads. As long as the monad M (understood as an endofunctor) generates a free monad, the generalised fold $\langle\!\langle\lambda\rangle\!\rangle$ of a natural transformation $\lambda : FG \to GM$, can be defined in terms of $\lfloor\text{-}\rfloor$ and $\langle\text{-}\rangle$ as $\langle\!\langle\lambda\rangle\!\rangle = G\lfloor\mathsf{id}\rfloor\cdot\langle\lambda\rangle$:



(26)

Even though $\langle\text{-}\rangle$ can be easily defined using $\langle\!\langle\text{-}\rangle\!\rangle$ and emb, we discuss both distributable and foldable free monads in this article. They are useful in different situations, so it is good to have both properties at our disposal.

5.4 Example: Hyland, Plotkin, and Power's theorem

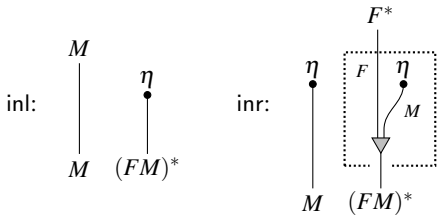
Now, we argue that the framework of universal properties of foldable free monads is powerful enough to prove a real-life theorem: Hyland, Plotkin, and Power's (2006) construction of the coproduct of a monad and a free monad. As discussed by Lüth and Ghani (2002), the coproduct of two monads is the least-interacting composition of their effects. This entails a lot of properties, which are valuable for functional programmers. In the specific example of Hyland, Plotkin, and Power's construction, it gives us—via Kelly's (1980) results—a strong connection between the resumption monad and reasoning about interleaving data and effects: algebras of the $M(FM)^*$ monad are exactly F-and-M-algebras (Atkey and Johann 2015).

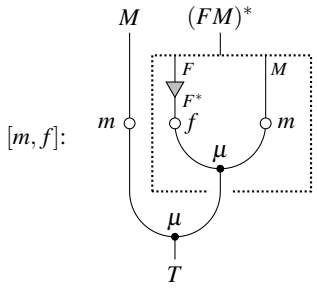
Theorem 1 (Hyland et al. 2006)**.** *Consider an endofunctor F and a monad M. If the uniform foldable free monads F^* and $(FM)^*$ exist, the resumption monad $M(FM)^*$ from Section 4.5 is the coproduct of M and F^* in the category of monads and monad morphisms.*

In detail, this means that there exist two natural transformations $\mathsf{inl} : M \to M(FM)^*$ and $\mathsf{inr} : F^* \to M(FM)^*$ such that:

1. Both inl and inr are monad morphisms,
2. For a monad T and monad morphisms $f : F^* \to T$ and $m : M \to T$, we define a natural transformation $[m, f] : M(FM)^* \to T$,
3. The natural transformation $[m, f]$ is a monad morphism,
4. It is the case that $[m, f]\cdot\mathsf{inl} = m$ and $[m, f]\cdot\mathsf{inr} = f$,
5. $[m, f]$ is a unique monad morphism with the property above.

The involved morphisms can be defined as shown in the following diagrams:



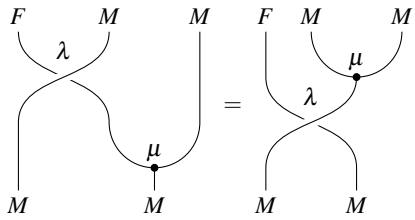


For brevity, we do not prove all the properties mentioned above. We focus on the one that is the trickiest, number 3. All the others follow rather easily from chains of equalities. To prove that $[m, f]$ is a monad morphism, we first need to introduce a couple of auxiliary definitions and lemmata.

Lemma 2. *For a monad M, its multiplication $\mu : MM \to M$ id is a distributive law of a monad over an endofunctor, where id is the identity monad.*

Proof. Apply the associativity of the multiplication (1). □

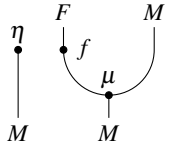
For a monad M and a functor F, we say that a natural transformation $\lambda : FM \to MM$ is *right-biased* if $M\mu\cdot\lambda M = F\mu$, or, depicted using string diagrams:



(27)

Intuitively, a natural transformation is right-biased if it puts all the non-trivial computations in the right-hand component of the composition MM, while the left-hand side consists of pure computations.

Lemma 3. *For an endofunctor F, a monad M, and a natural transformation $f : F \to M$, the natural transformation*



is right-biased.

Proof. Apply the associativity of the multiplication (1). □

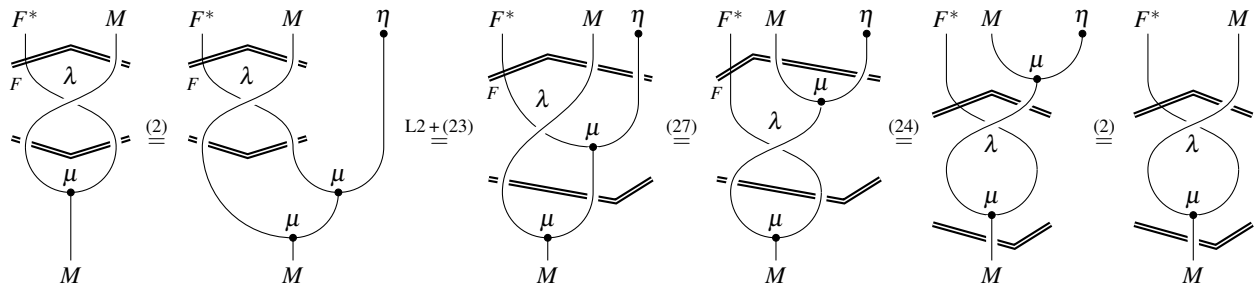
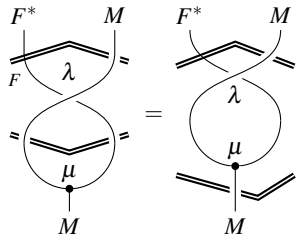


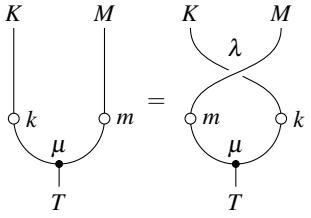
Figure 2. Proof of Lemma 4

Lemma 4. *For a right-biased $\lambda : FM \to MM$, the following holds:*



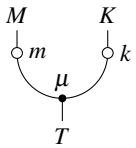
Proof. See Figure 2. □

Lemma 5. *Let M, T, K be monads, $k : K \to T$ and $m : M \to T$ be monad morphisms, and $\lambda : KM \to MK$ be a distributive law between monads such that:*



(28)

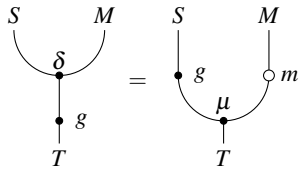
Then,



is a monad morphism.

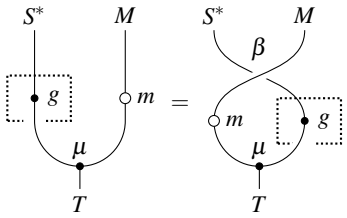
Proof. Coherence with the unit is easy. Coherence with the multiplication is shown in Figure 3; for readability, we wipe out the labels—the crossing of strings corresponds to λ, the circles represent m or k (depending on the type of the string), while the black dots are the multiplications of the appropriate monads. □

Recall from Section 4.5, equation (18), the definition of a right module S over a monad M with an action $\delta : SM \to S$. For a monad T, we define a *module-to-monad morphism* as a natural transformation $g : S \to T$ paired with a monad morphism $m : M \to T$ such that $\mu \cdot Tm \cdot gM = g \cdot \delta$, or, using string diagrams:



(29)

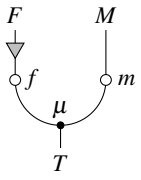
Lemma 6. *Let S be a right module over a monad M with an action $\delta : SM \to S$. Let $g : S \to T$ and $m : M \to T$ form a module-to-monad morphism. Then, the following holds, where β is as defined in* (19)*:*



Proof. See Figure 4. □

Corollary 7. *For monad morphisms $f : F^* \to T$ and $m : M \to T$, the natural transformation $[m, f]$ is a monad morphism.*

Proof. Note that FM is a right module over M. Moreover, the natural transformation



together with m form a module-to-monad morphism, which is easy to verify. Now, it is enough to apply Lemmata 5 and 6. □

6. Implementation: algebraically free monads

How do we know that the particular implementation of the free monad data structure as known from functional programming really satisfies all the properties described in this article? As shown in Section 5.3, it is enough to define the operation $\langle\langle$-$\rangle\rangle$, and prove that it satisfies the universal property and uniformity. In Haskell, $\langle\langle$-$\rangle\rangle$ can be given as follows:

```
genFold :: (Functor f, Functor g, Monad m)
        ⇒ (∀x.f (g x) → g (m x))
        → Free f (g a) → g (m a)
```

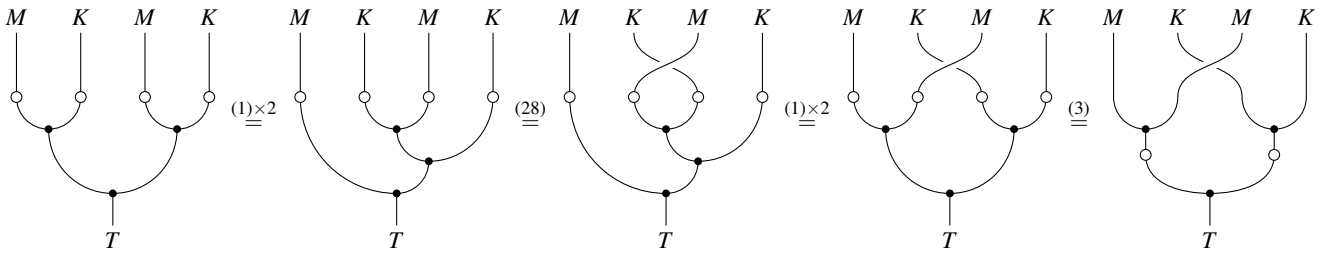


Figure 3. Proof of Lemma 5

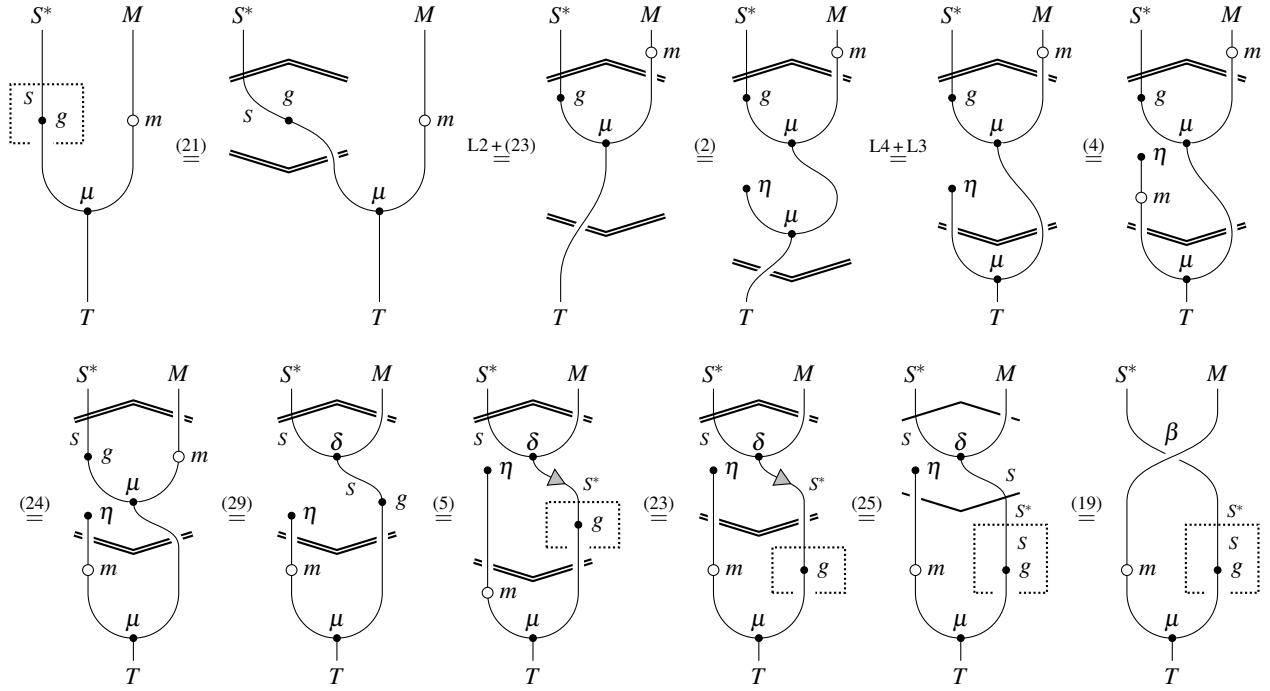


Figure 4. Proof of Lemma 6

genFold d (*Var g*) = *fmap return g*
genFold d (*Op f*) = *fmap join* (*d* (*fmap* (*genFold d*) *f*))

Now, it is enough to use the initial algebra semantics (the function *genFold* is clearly a fold of the initial algebra) and prove the necessary properties using basic algebra of programming (Bird and de Moor 1997). However, this would be a very tedious task. To make it a bit easier, we can employ some known results from category theory. The details are beyond the scope of this article, so we give only a short overview to connect the universal properties with the more traditional way of reasoning about free monads.

First, assume that for an endofunctor F on a category $\mathscr{C}$ with coproducts, the free algebra $F^*A = \mu X.FX + A$ exists for all objects A. Then, the tuple $\langle F^*A, \mu_A \cdot \mathsf{emb}_{F^*A} : FF^*A \to F^*A \rangle$ is the free F-algebra generated by the object A (Barr 1970). The monad F^* is obtained as the monad induced by the free–forgetful adjunction, and we say that it is an *algebraically free* monad. The adjointness means that F-algebra homomorphisms $\langle F^*A, \mu_A \cdot \mathsf{emb}_{F^*A} \rangle \to \langle B, b \rangle$ are in a one-to-one correspondence with morphisms $A \to B$. In symbols, we have the following natural isomorphism between hom-sets:

$$\Phi_{A,\langle B,b\rangle} : F\text{-Alg}[\langle F^*A, \mu_A \cdot \mathsf{emb}_{F^*A} \rangle, \langle B, b \rangle] \to \mathscr{C}[A, B]$$

Given a natural transformation $\lambda : FG \to GM$, we define the components of the distributive law $\langle\langle \lambda \rangle\rangle_A : F^*GA \to GMA$ as:

$$\Phi^{-1}_{GA,\langle GMA, G\mu^M_A \cdot \lambda_{MA}\rangle}(G\eta^M_A)$$

To show that $\langle\langle \lambda \rangle\rangle$ is a distributive law, one can employ the fact that the category F-Alg is isomorphic to the Eilenberg–Moore category of F^*, which is a known result that can be shown using Beck's theorem (1967). Then, the result follows from the correspondence between distributive laws and liftings, also introduced by Beck (1969). For a detailed exposition, see, for example, Bartels' PhD dissertation (2004).

Uniformity is much simpler, since it follows directly from the naturality of Φ. For natural transformations as in the equality (24), we can prove it as follows (for brevity, we omit the subscript of the

natural transformation Φ^{-1}):

$$\begin{aligned}\Phi^{-1}(K\eta_A)\cdot F^* f_A &= \Phi^{-1}(K\eta_A \cdot f_A) && \text{(naturality of } \Phi^{-1}\text{)}\\ &= \Phi^{-1}(f_{MA}\cdot J\eta_A) && \text{(naturality of } f\text{)}\\ &= f_{MA}\cdot \Phi^{-1}(J\eta_A) && \text{(naturality of } \Phi^{-1}\text{)}\end{aligned}$$

7. Conclusion

Reasoning with universal properties is an established principle in functional programming, for example, in the form of initial algebra semantics that allows one to program with and reason about algebraic data structures in terms of folds and unfolds (Fokkinga 1992). The point of this pearl is that one can treat free monads in a way that is more abstract than merely as algebraic data structures. Free monads come from category theory bringing along their own universal properties, which are better suited for high-level reasoning than simple folds. In particular, they respect the structure of the involved monads on the nose, which greatly simplifies proofs that some things are monads, monad morphisms, or distributive laws between monads.

Reasoning with graphical calculi, such as string diagrams, is of course a matter of taste. One can work out every proof in this article using expressions written down in the more traditional fashion, or by diagram chasing. On the other hand, a well-suited graphical calculus can aid reasoning by presenting complicated expressions in a more intuitive form. The amount of administrative steps is reduced, which means that the possible next steps are easier to identify, and one can focus on the important aspects of the proof.

String diagrams were introduced by Penrose (1971) to reason about the calculus of tensors. Since then, graphical calculi have been used to manage complexity in different areas of category theory and its applications. Examples include diagrams tailored for monoidal categories (Selinger 2011), traced monoidal categories (Joyal et al. 1996), or logic (Brady and Trimble 1998). There exist 3-dimentional graphical calculi (Barrett et al. 2016), or even n-dimentional ones (Kock et al. 2010). The current development is heavily inspired by string diagrams introduced for Kan extensions by Hinze (2012).

Acknowledgments

Diagrams similar to the ones shown in Section 3 first appeared in an unpublished appendix to a paper by the present authors and Jeremy Gibbons (2015). Back then, we didn't know how to use the 2-categorical framework to show a theorem similar to Corollary 7, and we had to resort to the 'intensional' means. The authors would like to thank Tom Schrijvers and Alexander Vandenbroucke for their notes on an early draft of this paper, and the anonymous reviewers for their constructive remarks.

References

R. Atkey and P. Johann. Interleaving data and effects. *Journal of Functional Programming*, 25:e20 (44 pages), 2015.

M. Barr. Coequalizers and free triples. *Mathematische Zeitschrift*, 116(4): 307–322, 1970.

J. W. Barrett, C. Meusburger, and G. Schaumann. Gray categories with duals and their diagrams, 2016. To appear in *Journal of Differential Geometry*.

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats: Distributive Laws in Coalgebraic Modelling*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

J. Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer Berlin Heidelberg, 1969.

J. M. Beck. *Triples, Algebras and Cohomology*. PhD thesis, Columbia University, 1967.

R. Bird and O. de Moor. *Algebra of Programming*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

R. S. Bird and L. Meertens. Two exercises found in a book on algorithmics. In L. Meertens, editor, *Program Specification and Transformation*, pages 451–457. North-Holland, 1987.

G. Brady and T. Trimble. A string diagram calculus for predicate logic and C. S. Peirces system beta, 1998. URL `people.cs.uchicago.edu/ brady/beta98.ps`. preprint.

P.-L. Curien. *Computer Science Logic: 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, chapter The Joy of String Diagrams, pages 15–22. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

M. M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Enschede, February 1992. URL `http://doc.utwente.nl/66251/`.

R. Hinze. Kan extensions for program optimisation or: Art and Dan explain an old trick. In *Mathematics of Program Construction - 11th International Conference, MPC 2012, Madrid, Spain, June 25-27, 2012. Proceedings*, pages 324–362, 2012.

R. Hinze and D. Marsden. Equational reasoning with lollipops, forks, cups, caps, snakes, and speedometers. *Journal of Logical and Algebraic Methods in Programming*, pages –, 2016. In press.

M. Hyland, G. D. Plotkin, and J. Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1-3):70–99, 2006.

A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, 1996.

G. M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22:1–83, 8 1980. ISSN 1755-1633.

J. Kock, A. Joyal, M. Batanin, and J.-F. Mascari. Polynomial functors and opetopes. *Advances in Mathematics*, 224:2690–2737, 2010.

C. Lüth and N. Ghani. Composing monads using coproducts. In M. Wand and S. L. P. Jones, editors, *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP '02), Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, pages 133–144. ACM, 2002. ISBN 1-58113-487-8. doi: 10.1145/581478.581492. URL `http://doi.acm.org/10.1145/581478.581492`.

E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture*, pages 124–144, New York, NY, USA, 1991. Springer-Verlag New York, Inc. URL `http://dl.acm.org/citation.cfm?id=127960.128035`.

E. Moggi. An Abstract View of Programming Languages. Technical report, Edinburgh University, 1989. URL `http://www.lfcs.inf.ed.ac.uk/reports/90/ECS-LFCS-90-113/`.

R. Penrose. Applications of negative dimensional tensors. In D. Welsh, editor, *Combinatorial Mathematics and its Applications*, pages 221–244. Academic Press, New York, 1971.

M. Piróg and J. Gibbons. Tracing monadic computations and representing effects. In J. Chapman and P. B. Levy, editors, Proceedings Fourth Workshop on *Mathematically Structured Functional Programming,* Tallinn, Estonia, 25 March 2012, volume 76 of *Electronic Proceedings in Theoretical Computer Science*, pages 90–111. Open Publishing Association, 2012. doi: 10.4204/EPTCS.76.8.

M. Piróg, N. Wu, and J. Gibbons. Modules over monads and their algebras. In L. S. Moss and P. Sobocinski, editors, *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands*, volume 35 of *LIPIcs*, pages 290–303. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

T. Schrijvers, N. Wu, B. Desouter, and B. Demoen. Heuristics entwined with handlers combined. In *PPDP 2014 : proceedings of the 16th international symposium on principles and practice of declarative programming*, page 12. Association for Computing Machinery (ACM), 2014.

P. Selinger. A survey of graphical languages for monoidal categories. In B. Coecke, editor, *New Structures for Physics*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.