Monads and their Applications to Progamming

Report as coursework of "Categories and Quantum Informatics",
University of Edinburgh, 3 March 2017

Lukas Convent

1 Introduction

Monads play an important role in pure functional programming languages like Haskell, enabling the description of non-pure programs in a pure environment. The gist is that they identify a level within the type system, together with rules for carrying out computation on this level. While it is always possible to lift data from the pure level into the monad level, monads don't necessarily provide the possibility of lowering data back to the pure level. This can be used to encapsulate certain behavior like effectful computation. Monads are also a well-studied structure in category theory. From this more general perspective we introduce them and hope to shed some light on their connection to programming. First of all, the definition of a monad:

Definition 1.1 *A* ***monad*** $(\boldsymbol{F}, \boldsymbol{\eta}, \boldsymbol{\mu})$ *on a category* $\boldsymbol{C}$ *consists of a functor* $F : \boldsymbol{C} \to \boldsymbol{C}$ *together with two natural transformations* $\eta : Id_{\boldsymbol{C}} \Rightarrow F$ *and* $\mu : F^2 \Rightarrow F$*, making the following three diagrams commute:*

$$\begin{array}{ccccc} F(A) & \xrightarrow{F(\eta_A)} & F^2(A) & \xleftarrow[F(\mu_A)]{} & F^3(A) \\ \downarrow{\scriptstyle \eta_{F(A)}} & \searrow^{id_{F(A)}} & \downarrow{\scriptstyle \mu_A} & & \downarrow{\scriptstyle \mu_{F(A)}} \\ F^2(A) & \xrightarrow{\mu_A} & F(A) & \xleftarrow[\mu_A]{} & F^2(A) \end{array}$$

Monads can be thought of as providing a wrapping mechanism for objects. An object can always be wrapped into a monad (**unit** $\boldsymbol{\eta}$). Any further wrappings can be merged (**multiplication** $\boldsymbol{\mu}$) such that it does not matter whether a new *inner* wrapping is merged from inside or a new *outer* wrapping is merged from outside the object's original wrapping (two left diagrams). When merging two times in a row, it does not matter in which order this happens (right diagram).

The names *unit* and *multiplication* already hint at a relationship between monads and monoids. Building on the explanation above, we can describe this relationship by a well-known citation in a serious manner: A monad is just a monoid in the category of endofunctors [2]. In this category $[\boldsymbol{C}, \boldsymbol{C}]$, endofunctors are the objects, natural transformations between them form the morphisms and endofunctor composition is the tensorproduct ($[\boldsymbol{C}, \boldsymbol{C}]$ is a monoidal category with unit Id_C). A monad F is a special object in this category, because one can multiply out any two composed applications of F into a single one (binary operation μ). Furthermore, multiplying by F makes no difference (neutral element F).

1.1 Kleisli Category of a Monad

Every monad F on a category $\boldsymbol{C}$ induces an accompanying category $\boldsymbol{C^F}$, called the Kleisli category. Intuitively, if one considers the objects of $\boldsymbol{C}$ as types and morphisms as functions between types, the Kleisli category $\boldsymbol{C^F}$ is obtained by cutting out all pure functions of $\boldsymbol{C}$. Thereby, any computation left is potentially effectful in terms of the monad F. This category appears in practice for example in the *do* notation in Haskell. We follow the definition of Uustalu [3].

Definition 1.2 *Given a monad (F, η, μ) on $\boldsymbol{C}$, the **Kleisli category $\boldsymbol{C_F}$** has the same objects as $\boldsymbol{C}$, but its morphisms $A \Rightarrow B$ are morphisms $A \to F(B)$ in $\boldsymbol{C}$. For morphisms $A \overset{f}{\Rightarrow} B$ and $B \overset{g}{\Rightarrow} C$, we define composition $A \overset{g \circledcirc f}{\Longrightarrow} C := A \overset{\mu_C \circ F(g) \circ f}{\Longrightarrow} C$ and identity morphisms $A \overset{id_A^F}{\Longrightarrow} A := A \overset{\eta_A}{\Longrightarrow} A$. A corresponding **inclusion functor** $J : \boldsymbol{C} \to \boldsymbol{C^F}$ maps objects to themselves and a morphism $A \overset{f}{\to} B$ to $A \overset{J(f)}{\Longrightarrow} B$ with $J(f) = \eta_B \circ f$.*

Lemma 1.3 *$\boldsymbol{C^F}$ is a category and J is a functor.*

Proof For $\boldsymbol{C^F}$, composition $\circledcirc$ is associative by naturality of μ and the morphisms id_A^F satisfy the identity laws by naturality of η and the left two commuting monad diagrams.
For J, we first show that given morphisms $A \overset{f}{\to} B$ and $B \overset{g}{\to} C$, we have $J(g \circ f) = J(g) \circledcirc J(f)$. This follows by the commuting monad diagrams and naturality of η. We then have to show $J(id_A) = id_A^F$ which follows trivially by the identity law in $\boldsymbol{C}$. ∎

2 Monads in Programming

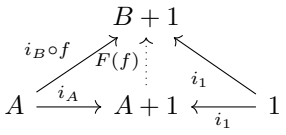
The type system of programming language can be seen as very similar to ***Set***. We therefore use ***Set*** for the following examples of monads and bear in mind that *types* are objects, *(monomorphic) functions* are morphisms, *type constructors* are functors and *polymorphic functions* are natural transformations. Furthermore we can use the monoidal structure of ***Set*** to represent *product types* as products and *sum types* as coproducts.

Monads can be interpreted to introduce side-effects in the very literal meaning: Input and and output channels can be attached to the side of data, as it is the case for the log and input monad. The exception monad allows a computation to fail and therefore to only deliver the fail message. The log monad carries along a log message and the input monad can be seen as a program description which awaits an environment (from the external world) in order to execute.

Definition 2.1 *The **exception monad** consists of three components (F, η, μ).*

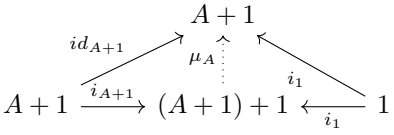
- *Functor F: Objects are mapped as $F(A) = A + 1$.*

  Morphisms $A \overset{f}{\to} B$ are mapped to $A + 1 \overset{F(f)}{\longrightarrow} B + 1$ as follows:



- *Unit η_A: $A \overset{i_A}{\longrightarrow} A + 1$*

- *Multiplication μ_A:*



Lemma 2.2 *The exception monad is indeed a monad.*

Proof We have to show that F is a functor, η and μ are natural and the monad diagrams commute. First, we show that F is a functor.

1. $F(id_A) = id_{A+1}$ for all objects A

   By def. of $F(id_A)$, we have $F(id_A) \circ i_A = i_A \circ id_A = i_A = id_{A+1} \circ i_A$. The claim follows by uniqueness of $F(id_A)$.

2. $F(g \circ f) = F(g) \circ F(f)$ for all morphisms $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$

   By def., we have $F(f) \circ i_A = i_B \circ f$ and $F(g) \circ i_B = i_C \circ g$ and $F(g \circ f) \circ i_A = i_C \circ g \circ f$. By the first two equations, we obtain $F(g \circ f) \circ i_A = F(g) \circ F(g) \circ i_A$, which by uniqueness of $F(g \circ f)$ yields the claim.

We show naturality of η and μ. Naturality of η follows by def. of $F(f)$. Naturality of μ is a bit trickier. For a morphism $A \xrightarrow{f} B$, we have to show $F(f) \circ \mu_A = \mu_B \circ F^2(f)$. We consider an element $x \in (A + 1) + 1$. By assuming that every element of $(A + 1) + 1$ is injectively hit by either i_{A+1} or i_1, we do a case analysis on x and show that $(F(f) \circ \mu_A)(x) = (\mu_B \circ F^2(f))(x)$ holds in any case.
1. $x = i_1(\bullet)$. Then $(F(f) \circ \mu_A \circ i_1)(\bullet) = (\mu_B \circ F^2(f) \circ i_1)(\bullet)$ holds by def. of μ, $F(f)$.
2. $x = (i_{A+1} \circ i_1)(\bullet)$. Very similar to 1.
3. $x = (i_{A+1} \circ i_A)(a)$ for some $a \in A$. We obtain $(F(f) \circ \mu_A \circ \eta_{A+1} \circ \eta_A)(a) = (\mu_B \circ F^2(f) \circ \eta_{A+1} \circ \eta_A)(a)$ by the usual definitions and by using the second commuting monad diagram $\mu_B \circ i_{B+1} = id_B + 1$ (which we can use because its proof as follows is obtained only by definition).

We show that the three monad diagrams commute.

1. $$\begin{aligned} \mu_A \circ F(\eta_A) &= id_{A+1} \\ &= \mu_A \circ i_{A+1} && \text{(by def. of } \mu_A\text{)} \\ &= \mu_A \circ F(\eta_A) && \text{(by univ. prop., } F(\eta_A) \text{ is uniquely } i_{A+1}\text{)} \end{aligned}$$

2. $$\mu_A \circ \eta_{A+1} = id_{A+1} \qquad \text{(by def. of } \mu_A, \eta_{A+1}\text{)}$$

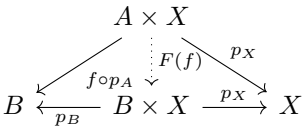
3. $$\mu_A \circ \mu_{A+1} = \mu_A \circ F(\mu_A) \qquad \text{(the def. of } \mu_{A+1} \text{ and } F(\mu_A) \text{ are exactly the same, considering that } id_{(A+1)+1} = \mu_A \circ i_{A+1} \text{ by 1.)}$$

■

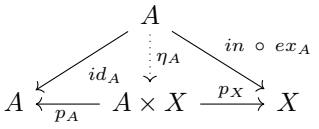
Definition 2.3 *For a fixed set X with distinguished initial state $1 \xrightarrow{in} X$, the **log monad** consists of three components (F, η, μ).*

- *Functor F: Objects are mapped as $F(A) = A \times X$.*

  Morphisms $A \xrightarrow{f} B$ are mapped to $A \times X \xrightarrow{F(f)} B \times X$ as follows:



- *Unit η_A:*



with unique effect $A \xrightarrow{ex_A} 1$

- *Multiplication* μ_A*:* $(A \times X) \times X \xrightarrow{p_{A\times X}} A \times X$

Lemma 2.4 *The log monad is indeed a monad.*

Proof Again, we show functoriality, naturality and commutativity. We first have to show that F is a functor. This follows by the same argument as for the exception monad (Lemma 2.2). Naturality of μ follows by def. of $F(f)$. For naturality of η, assume a morphism $A \xrightarrow{f} B$ and $a \in A$. We have to show $(F(f) \circ \eta_A)(a) = (\eta_B \circ f)(a)$. By the univ. prop. of $F(f)$ and η_A, both sides of the equation are equal to $(f(a), (in \circ ex_A)(a))$.

We show that the three monad diagrams commute.

1. $\mu_A \circ F(\eta_A) = id_{A\times X}$
   $= p_{A\times X} \circ \eta_{A\times X}$ (by def. of $\eta_{A\times X}$)
   $= \eta_A \circ p_A$ (by naturality of η)
   $= \mu_A \circ F(\eta_A)$ (by def. of $F(\eta_A), \mu_A$)
2. $\mu_A \circ \eta_{A\times X} = id_{A\times X}$ (by def. of μ_A, $\eta_{A\times X}$)
3. $\mu_A \circ \mu_{A\times X} = \mu_A \circ F(\mu_A)$ (by def. of μ_A, $F(\mu_A)$)

■

Definition 2.5 *For a fixed set* X*, the* ***input monad*** *consists of three components* (F, η, μ)*.*

- *Functor* F*: Objects are mapped as* $F(A) = A^X$*.*
  Morphisms $A \xrightarrow{f} B$ *are mapped to* $A^X \xrightarrow{F(f)} B^X$ *with* $F(f)(a) = f \circ a$
- *Unit* $A \xrightarrow{\eta_A} A^X$ *is defined as* $\eta_A(a)(\_) = a$
- *Multiplication* $A^{X^X} \xrightarrow{\mu_A} A^X$ *is defined as* $\mu_A(a)(x) = (a\ x)\ x$

Lemma 2.6 *The input monad is indeed a monad.*

Proof Again, we show functoriality, naturality and commutativity. We first have to show that F is a functor.

1. $F(id_A)\ a = id_A \circ a = a = id_{A^X}(a)$ for all $a \in A^X$
2. $F(g \circ f)\ a = g \circ f \circ a = ((\lambda b.g \circ b) \circ (\lambda a'.f \circ a'))\ a = (F(g) \circ F(f))\ a$ for all $A \xrightarrow{f} B$, $B \xrightarrow{g} C$ and $a \in A^X$

Naturality of η and μ is easily shown by rewriting the λ-terms on both sides of the equations such that they are syntactically equal and therefore represent the same morphism (similar to the functoriality proof above). Commutativity of the monad diagrams is easily shown in the same way without anything interesting happening. ■

3 Strong Monads

For monoidal categories, a monad can be attached with a tensorial strength. Intuitively, such a strength provides a way to pull a left-tensored neighbour onto the same monad level and can be understood as a kind of cooperation between the monad and the monoidal structure. We follow in our definition Uustalu [5].

Definition 3.1 *For a monad F, a **tensorial strength** is a natural transformation $st_{A\otimes B} : A \otimes F(B) \to F(A \otimes B)$ such that the following four diagrams commute.*

$$\begin{array}{ccc} 1 \otimes F(A) & \xrightarrow{st_{1,A}} & F(1 \otimes A) \\ & \searrow^{\lambda_{F(A)}} & \downarrow{\scriptstyle F(\lambda_A)} \\ & & F(A) \end{array} \qquad \begin{array}{ccc} (A \otimes B) \otimes F(C) & \xrightarrow{st_{A\otimes B,C}} & F((A \otimes B) \otimes C) \\ \downarrow{\scriptstyle \alpha_{A,B,F(C)}} & & \downarrow{\scriptstyle F(\alpha_{A,B,C})} \\ A \otimes (B \otimes F(C)) & \xrightarrow{id_A \otimes st_{B,C}} A \otimes F(B \otimes C) & \xrightarrow{st_{A,B\otimes C}} F(A \otimes (B \otimes C)) \end{array}$$

$$\begin{array}{ccc} A \otimes B & \xrightarrow{id_A \otimes \eta_B} & A \otimes F(B) \\ & \searrow^{\eta_{A\otimes B}} & \downarrow{\scriptstyle st_{A,B}} \\ & & T(A \otimes B) \end{array} \qquad \begin{array}{ccc} A \otimes F^2(B) & \xrightarrow{st_{A,F(B)}} & F(A \otimes F(B)) \xrightarrow{F(st_{A,B})} F^2(A \otimes B) \\ \downarrow{\scriptstyle id_A \otimes \mu_B} & & \downarrow{\scriptstyle \mu_{A\otimes B}} \\ A \otimes F(B) & \xrightarrow{st_{A,B}} & F(A \otimes B) \end{array}$$

The four commuting diagrams can be seen as properties that make sure the monoidal structure is preserved. For example, strengthening should not break the associativity of tensor products (upper right diagram). A monad with a strength is called a **strong monad**. It turns out that in ***Set***, every monad is strong through a canonical strength; we follow the definition of Uustalu [5] and provide a proof.

Lemma 3.2 *For every monad F in **Set**, the natural transformation $st_{A,B}(a, c) = F(\lambda b.(a, b))\ c$ is a tensorial strength.*

Proof We first show that $st_{A,B}$ is a natural transformation. Assume two morphisms $A \xrightarrow{f} A'$, $B \xrightarrow{g} B'$. We have to show $(F(f \otimes g) \circ st_{A,B})(a, b) = st_{A',B'} \circ (f \otimes F(g))(a, b)$ for all $a \in A$, $b \in F(B)$. It is easy to show that by unfolding definitions and rewriting the λ-terms this equation holds. We then have to show that the four strength diagrams commute. Using the same strategy of rewriting suffices to show this. ∎

4 Commutative Monads

As one would expect, there is a counterpart to the notion of strength, which allows to pull the right tensored neighbour onto the monad level. A **tensorial costrength** is a natural transformation $ts_{A,B} : F(A) \otimes B \to F(A \otimes B)$ with commuting diagrams analogue to the ones of tensorial strength. We can now construct a natural transformation $F(A) \otimes F(B) \to F(A \otimes B)$ in two ways [5]:

$$\begin{array}{ccccc} F(A) \otimes F(B) & \xrightarrow{st_{F(A),B}} & F(F(A) \otimes B) & \xrightarrow{F(ts_{A,B})} & F^2(A \otimes B) \\ \downarrow{\scriptstyle ts_{A,F(B)}} & & & & \downarrow{\scriptstyle \mu_{A\otimes B}} \\ F(A \otimes F(B)) & \xrightarrow{F(st_{A,B})} & F^2(A \otimes B) & \xrightarrow{\mu_{A\otimes B}} & F(A \otimes B) \end{array}$$

We call a monad **commutative**, if it is strong and costrong and the two constructions above coincide. Intuitively, this means that the monad object one obtains by these transformations is independent of the order of strengthening. We now come back to the three monad examples given before and determine whether they are commutative.

Lemma 4.1 *The exception and input monads are commutative, the log monad is not.*

Proof First of all, we note that all three monads possess a canonical strength $st_{A,B} : F(A) \otimes B \to F(A \otimes B)$ and costrength $ts_{A,B} : F(A) \otimes B \to F(A \otimes B)$. The costrength is defined analogously $ts_{A,B}(c, b) := F(\lambda a.(a, b))\ c$ and is indeed a costrength by an argument analogous to Lemma 3.2.

We now show that the exception monad is commutative. The essential thing to notice is that *no matter when* the unique exception is thrown, it will appear on top level. Assuming $(x, y) \in (A+1) \otimes (B+1)$, we have to show $(\mu_{A\otimes B} \circ F(ts_{A,B}) \circ st_{A,B})(x, y) = (\mu_{A\otimes B} \circ F(st_{A,B}) \circ ts_{A,B})(x, y)$. By case analysis on (x, y), for the first cases $(i_1(\bullet), \_)$, $(\_, i_1(\bullet))$, both sides of the equation reduce to $i_1(\bullet)$. For the remaining case $(i_A(a'), i_B(b'))$ for some $a' \in A$, $b' \in B$, both sides of the equation reduce to $i_{A\otimes B}(a', b')$.

We now show that the log monad is not commutative. The essential thing to notice is that because one log message is discarded when merging two monads, the *order matters*. Assume that X has two distinct elements z_1, z_2 and let $x = (a, z_1) \in A \otimes X$ and $y = (b, z_2) \in B \times X$ for some elements a, b. Commutativity requires the equation $(\mu_{A\otimes B} \circ F(ts_{A,B}) \circ st_{A,B})(x, y) = (\mu_{A\otimes B} \circ F(st_{A,B}) \circ ts_{A,B})(x, y)$ to hold, but the left side reduces to $((a', b'), z_1)$ while the right side reduces to $((a', b'), z_2)$.

We now show that the input monad is commutative. The essential thing to notice is that the same input to a monad will be handed down to every nested monad, thus *nesting order does not matter*. Indeed, by syntactic rewriting, both sides of the commutativity equation reduce to the same term. ∎

5 Adjunctions

As we know, adjunctions are everywhere. So it is not surprising that there is also a fundamental connection to the notion of monads. For better intuition and to have examples of their application, let us again consider the ***Set*** category. We recall that *types* are sets and *program functions* are functions. Furthermore, monads are type constructors which possess a special structure.

We discuss the connections as follows and thereby follow notes of Schalk [4] and Hagino [1]: First, we consider how a type can be defined in terms of an adjunction on the example of the input monad, then we present how one can derive the monadic structure from the adjunction.

Consider the input monad as a type constructor which constructs a type A^X for every type A. We can describe these types by two endofunctors $F = (\bullet \otimes X) : \boldsymbol{Set} \to \boldsymbol{Set}$ and $G = \bullet^X : \boldsymbol{Set} \to \boldsymbol{Set}$ which form an adjunction $(\bullet \otimes X) = F \dashv G = \bullet^X$. This means that there have to exist natural transformations $\eta_A : A \to (A \otimes X)^X$ and $\epsilon_A : A^X \otimes X \to A$, with $\epsilon_{F(A)} \circ F_{\eta A} = id_{F(A)}$ and $G_{\epsilon_A} \circ \eta_{G(A)} = id_{G(A)}$. We can look at η as a pair constructor and ϵ as an evaluator. We can define a monad (M, η, μ) now out of any such given adjunction by setting the monad functor $M := GF$. We already have η and define $\mu_A := G\epsilon_{F(A)}$. Commutation of the monad diagrams can then be obtained by the adjunction properties.

References

[1] Tatsuya Hagino. "A categorical programming language". PhD thesis. University of Edinburgh, 1987.

[2] Saunders Mac Lane. *Categories for the Working Mathematician*. 1969.

[3] *Monads, compatible compositions of monads (Tarmo Uustalu)*. URL: `http://cs.ioc.ee/~tarmo/ssgep15/ssgep-1.pdf` (visited on 02/02/2017).

[4] *Some notes on monads (Andrea Schalk)*. URL: `http://www.cs.man.ac.uk/~schalk/notes/monads.pdf` (visited on 02/02/2017).

[5] *Strong functors, strong monads (Tarmo Uustalu)*. URL: `http://cs.ioc.ee/~tarmo/ssgep15/ssgep-1a.pdf` (visited on 02/02/2017).