Monads in category theory and computer science

W. Troiani[1] D. Murfet[2]

[1]Department of Mathematics and Statistics (Masters Student in Pure Mathematics)
University of Melbourne

[2]Department of Mathematics and Statistics (Lecturer)
University of Melbourne

DST Conference, 2017

Outline

1. Initial definitions
   - Categories, Functors, and Natural Transformations
   - Monads
2. Monads from the Programming Perspective
   - Kleisli Triples (Moggi)
   - Kleisli Triples in Haskell
3. Relationships Between the Two
   - Why These are the Same

Outline

1. Initial definitions
   - Categories, Functors, and Natural Transformations
   - Monads
2. Monads from the Programming Perspective
   - Kleisli Triples (Moggi)
   - Kleisli Triples in Haskell
3. Relationships Between the Two
   - Why These are the Same

The Formal Definition

as well as terminology

Definition (Category)

A **Category** $\mathscr{C}$ *consists of*

- *A class of objects* $obj(\mathscr{C}) = X, Y, Z, \ldots$
- *For each pair of objects* (X, Y)*, a set of morphisms* $Hom_{\mathscr{C}}(X, Y) = f : X \to Y, g : X \to Y, \ldots$
- *For all* $X \in obj(\mathscr{C})$*, there exists* $id_X : X \to X \in Hom_{\mathscr{C}}(X, X)$
- *For each triple of objects* (X, Y, Z)*, a function*

$$\circ : Hom_{\mathscr{C}}(X, Y) \times Hom_{\mathscr{C}}(Y, Z) \to Hom_{\mathscr{C}}(X, Z)$$
$$f \times g \mapsto g \circ f$$

The Formal Definition

Definition (Category)

Which satisfy the following conditions:

- *Associativity:* For all $X, Y, Z, W \in obj(\mathscr{C})$, and $f : X \to Y$, $g : Y \to Z$, $h : Z \to W$,

$$(h \circ g) \circ f = h \circ (g \circ f)$$

- *Identity:* For all $g : Y \to X$, and all $h : X \to Z$, we have that

$$h \circ id_X = h \text{ and } id_X \circ g = g$$

Terminology and an Example

Common Terminology

- Will often write $\mathscr{C}(X, Y)$ for $\mathrm{Hom}_{\mathscr{C}}(X, Y)$.
- Will often write $\mathscr{C}$ for $\mathrm{obj}(\mathscr{C})$.
- Morphisms are often called Arrows.
- Brackets are often dropped.

Example (The Category of Sets)

- $\mathrm{obj}(\underline{Set})$ is the class of all sets.
- $\mathrm{Hom}(X, Y) =$ Set of functions from X to Y
- Composition is function composition.
- For each X, id_X is just the identity function on X.

Functor

Definition (Functor)

A Functor is a map between Categories $\mathscr{C} \to \mathscr{D}$ *such that*

- *For all* $X \in \mathscr{C}$, $F(X) \in \mathscr{D}$, *and for all* $f : X \to Y$, $F(f) : F(X) \to F(Y)$
- *For all* $X \in obj(\mathscr{C})$,
$$F(1_X) = 1_{FX}$$
- *For all morphisms* $f : Y \to Z, g : X \to Y$, *in* $\mathscr{C}$,
$$F(f \circ g) = F(f) \circ F(g)$$

Natural Transformation

Definition (Natural Transformation)

Given two Categories $\mathscr{C}$, $\mathscr{D}$, and two functors $F, G : \mathscr{C} \to \mathscr{D}$, a Natural Transformation $\mu : F \Rightarrow G$ assigns to each objects $X \in \mathscr{C}$, a morphism $\mu_X : F(X) \to G(X)$ so that for any morphism $f : X \to Y$ in $\mathscr{C}$, the following diagram,

$$\begin{tikzcd} FX \arrow[r, "Ff"] \arrow[d, "\mu_X"'] & FY \arrow[d, "\mu_Y"] \\ GX \arrow[r, "Gf"'] & GY \end{tikzcd}$$

commutes

Outline

1. Initial definitions
   - Categories, Functors, and Natural Transformations
   - Monads
2. Monads from the Programming Perspective
   - Kleisli Triples (Moggi)
   - Kleisli Triples in Haskell
3. Relationships Between the Two
   - Why These are the Same

Formal Definition of a Monad

Definition (Monad)

A Monad on a category $\mathscr{C}$ is a triple (T, μ, η), consisting of

- *A functor $T : \mathscr{C} \to \mathscr{C}$.*
- *Two natural transformations, $\mu : T^2 \Rightarrow T$ and $\eta : 1_{\mathscr{C}} \Rightarrow T$ such that for all $X \in \mathscr{C}$, the following diagrams,*

$$\begin{tikzcd}
T(T(T(X))) \arrow[r, "\mu_{TX}"] \arrow[d, "T\mu_X"'] & T(T(X)) \arrow[d, "\mu_X"] \\
T(T(X)) \arrow[r, "\mu_X"] & T(X)
\end{tikzcd}
\qquad
\begin{tikzcd}
T(X) \arrow[r, "\eta_{TX}"] \arrow[dr, "id_{TX}"'] & T(T(X)) \arrow[d, "\mu_X"] & T(X) \arrow[l, "T(\eta_X)"'] \arrow[dl, "id_{TX}"] \\
& T(X) &
\end{tikzcd}$$

commute

Outline

1. Initial definitions
   - Categories, Functors, and Natural Transformations
   - Monads
2. Monads from the Programming Perspective
   - Kleisli Triples (Moggi)
   - Kleisli Triples in Haskell
3. Relationships Between the Two
   - Why These are the Same

Kleisli Triple

Definition (Kleisli Triple over a Category $\mathscr{C}$)

A triple $(T, \eta, \_^*)$ *consisting of a function*

$$T : obj(\mathscr{C}) \to obj(\mathscr{C})$$

For each object $A \in \mathscr{C}$*, a morphism* $\eta_A : A \to TA$*, and for each* $f : A \to TB$*, a morphism* $f^* : TA \to TB$*, satisfying,*

- $\eta_A^* = id_T A$
- *For any* $f : A \to TB$*, that* $f^* \eta_A = f$
- *For any* $f : A \to TB$ *and* $g : B \to TC$*, that* $g^* f^* = (g^* f)^*$

Kleisli Category

Definition (Kleisli Category)

Given a Kleisli triple $(T, \eta, \_^*)$ *over some Category* $\mathscr{C}$, *the Category* $\mathscr{C}_T$ *where*

- $obj(\mathscr{C}_T) = obj(\mathscr{C})$
- $\mathscr{C}_T(X, Y) = \mathscr{C}(X, TY)$
- id_X *(in* $\mathscr{C}_T$*) is* $\eta_X : X \to TX$
- *Given* $f \in \mathscr{C}_T(A, B)$, $g \in \mathscr{C}_T(B, C)$, *the composition is* $g^* f : A \to TC$

Note: The Kleisli Triple axioms are defined to make the Kleisli Category a Category.

It is within the Kleisli Category that computation is modeled.

Example 1, Partiality

Consider the category Set, and two functions $f : A \to B, g : B \to C$. Can these be extended to "functions" which might fail?

Example (Partiality)

- $TA = A \sqcup \{\perp\}$
- $\eta_A : A \to TA$ is inclusion.
- Given $f : A \to TB$, take $f^* : TA \to TB$ to be $f^*(a) = f(a), \forall a \in A$, and $f^*(\perp) = \perp$

This defines a Kleisli Triple.

Example 2, Side-effects

Again, take the category Set, and two functions $f : A \to B, g : B \to C$. Can these functions be extended to take into account the state of a machine? Fix a set of possible states S, then

Example (Side-effects)

- $TA = (A \times S)^S$
- $\eta_A : A \to TA = (A \times S)^S$ is the map $\eta_A(a)(s) = (a, s)$.
- Given $f : A \to TB = (B \times S)^S$, take $f^* : TA \to TB = (A \times S)^S \to (B \times S)^S$ to be, for $g \in (A \times S)^S$, $f^*(g)(s) = f(\pi_1(g(s)))(\pi_2(g(s)))$

Outline

1 Initial definitions
- Categories, Functors, and Natural Transformations
- Monads

2 Monads from the Programming Perspective
- Kleisli Triples (Moggi)
- Kleisli Triples in Haskell

3 Relationships Between the Two
- Why These are the Same

Haskell Monad Typeclass

A Haskell type is in the Monad typeclass once two functions

- $(>>=) :: ma \to (a \to mb) \to mb$
- Return $:: a \to ma$

The bind function, $(>>=)$, acts as $\_^*$, and Return is η. Here, m can be read as a mapping from a type a to a new type. This is a mapping from a type to a type corresponding to the notion of computation associated to this Monad.

The connection comes from the fact that

$$\mathrm{Hom}(a \to mb, ma \to mb) \cong \mathrm{Hom}(ma, (a \to mb) \to mb)$$

Outline

1. Initial definitions
   - Categories, Functors, and Natural Transformations
   - Monads
2. Monads from the Programming Perspective
   - Kleisli Triples (Moggi)
   - Kleisli Triples in Haskell
3. Relationships Between the Two
   - Why These are the Same

The upshot is,

Theorem

There is a one-one correspondence between Kleisli triples and monads.

Proof sketch

Given a Kleisli Triple $(T, \eta, \_^*)$, the corresponding functor $\hat{T}$ is T on objects, and given $f : A \to B$, $\hat{T}(f) = (\eta_B f)^*$. The multiplication map $\mu_A = \mathrm{id}^*_{TA}$.
Conversely, restricting a Monad functor T to objects, and taking $f^* = \mu_B(Tf)$ for $f : A \to TB$ gives a Kleisli Triple.

Direct Comparison

Example (Partiality)

- $TA = A \sqcup \{\perp\}$
- $\eta_A : A \to TA$ is inclusion.
- Given $f : A \to TB$, take $f^* : TA \to TB$ to be $f^*(a) = f(a)$, $\forall a \in A$, and $f^*(\perp) = \perp$

Corresponds to

Example (Partiality)

- $(A \xrightarrow{f} B) \overset{T}{\mapsto} \begin{pmatrix} \perp \to \perp \\ A \xrightarrow{f} B \end{pmatrix}$
- μ_A ...

Project Summary

The research expected outcomes is to scope the application of type theory and develop a "higher order monadic computation model" as a means of producing the foundational logic to address the defects in current proof assistants. Given the limited time of this project, the focus could be in any of the following:

- Review current proof-assistants (Coq, PVS,...) which are less well known to trustworthy systems, with view to confirming their limitations.
- Explore the interplay of typing features, including in particular record subtyping, arbitrary recursion, dependent types (essential for the FMME goals).
- Explore the development of higher-order monads within this type theory.
- Explore the application of these in a term logic with a fundamentally monadic notion of computation.

The project deliverable in this year is a report on any or all of the desired research outcomes, with recommended directions for the development of the FMME. Further identifying the most applicable university partners and suggested approach to future research, development and investment.