An introduction to that which shall not be named

Harley Eades III
Computer Science
Augusta University
heades@augusta.edu

September 2016

1 A quick example from programming

Consider a C program with signature `int f(int)`. Now describe all possible computations this program can do. This is a difficult task, because this function could do a lot, like, prompt the user for input, send packets across the network, modify global state, and much more, but then eventually return an integer.

Now consider a purely functional programming language like Haskell [1] and list all the possible computations the function `f :: Int -> Int` can do. The list is a lot smaller. We know without a doubt that this function must take an integer as an input, and then do integer computations, and finally return an integer. No funny business went on inside this function. Thus, reasoning about pure programs is a lot easier.

However, a practical programmer might now be asking, "How do we get any real work done in a pure setting?" From stage left enters the monad. These allow for us to annotate the return types of functions to indicate which side effects the function will use. For example, say we wanted `f` to use a global state of a integers, then its type would be `f :: Int -> State [Int] Int` to indicate that `f` will take in an integer input, then during computation will use a global state consisting of a list of integers, but then eventually return an integer. Thus, the return type of `f` literally lists the side effects the function will use. Now while reasoning about programs we know exactly which side effects to consider.

2 What is a monad really?

3 A short history

[2]

4 Jumping inside a monad

4.1 The Kleisli category

4.2 The Eilenberg-Moore category

4.3 Why not an example from linear logic?

5 Monads are modular, right?

References

[1] The haskell programming language. Online: `http://www.haskell.org`.

[2] Eugenio Moggi. Computational lambda-calculus and monads. pages 14–23. IEEE Computer Society Press, 1988.