Brief Monad Example

Andrew R. Plummer

Department of Linguistics
The Ohio State University

28 Oct, 2009

Outline

IO PROBLEM AS MOTIVATION

Suppose we have a relation R between integers and files. R simply takes an interger n and writes it to a file f.

Denote by $f[n]$ the file that results from writing the integer n to the file f. That is, $\langle n, f[n] \rangle \in R$.

Notice that $\langle n, f[n][n] \rangle$ is also in R. Thus there are two values for the integer n in R, and so R is not a function. That is, the naive approach to IO computation is not functional.

For a language to be purely functional, we need a more sophisticated approach to IO, and computation in general.

COMPUTATIONS AS FUNCTORS

The key insight into achieving pure functionality is treating values as objects of a category, and computations as functors between values.

For example, let A be an object of values (of type A). We can treat various computations on values in A as a functor M:

- **Exceptions**: $MA = (A + E)$ (E a set of exceptions).
- **Side effects**: $MA = (A \times S)^S$ (S a set of states).
- **Continuations**: $MA = R^{(R^A)}$ (R a set of results).

KLEISLI TRIPLES

We foramilze the insight as follows:

KLEISLI TRIPLE (1)

A *Kleisli triple* over a category **C** is a triple $(M, \eta, *)$, where

- $M : Ob(\mathbf{C}) \to Ob(\mathbf{C})$
- $\eta_A : A \to MA$, for all $A \in Ob(\mathbf{C})$,
- $f^* : MA \to MB$, for all $f : A \to MB$

Kleisli Triples

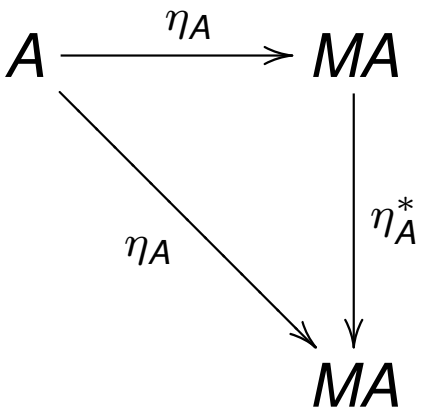
Kleisli Triple (2)

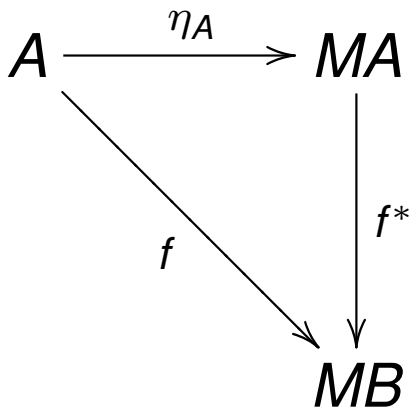
The following equations must be satisfied:

1. $\eta_A^* = \mathrm{id}_{MA}$
2. $f^* \circ \eta_A = f$, for all $f : A \to MB$
3. $g^* \circ f^* = (g^* \circ f)^*$, for all $f : A \to MB$ and $g : B \to MC$.

KLEISLI TRIPLES

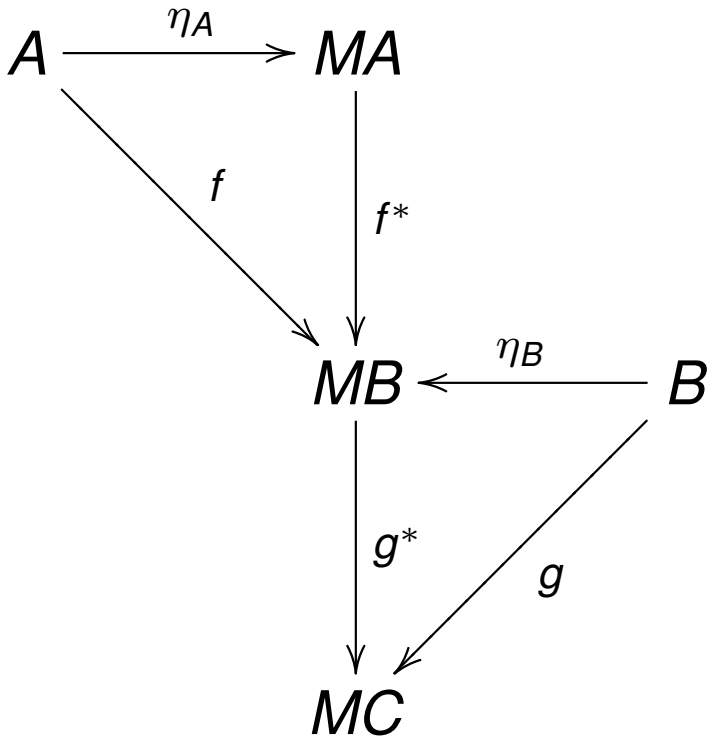
Equations 1. and 2. are visualized as





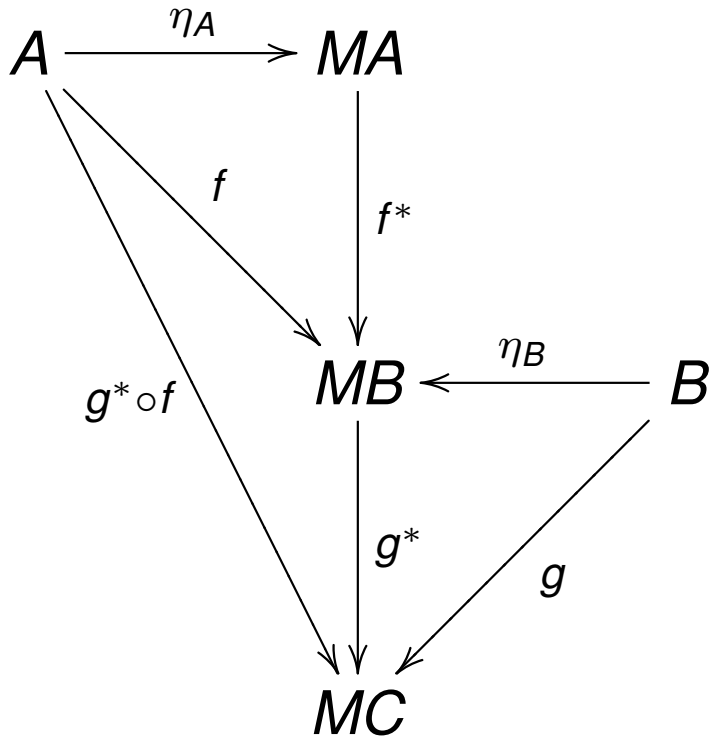
KLEISLI TRIPLES

Equation 3. is visualized as



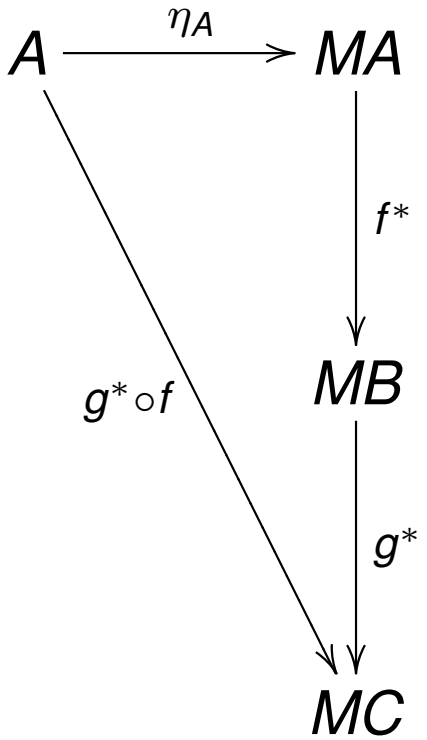
KLEISLI TRIPLES

Equation 3. is visualized as



KLEISLI TRIPLES

Equation 3. is visualized as



HASKELL EXAMPLE

Haskell monads are Kleisli triples. They are called monads since every Kleisli triple can be extended to a category theoretic monad (a kind of endofunctor we won't define).

EXCEPTION HANDLING

An Exception monad *(Exc, eta, ast)* can be defined as follows:

- Exc a = ThrowException | Iden a
- eta :: a -> Exc a
  eta a = Iden a
- ast f = f, for all *f* :: a -> Exc b

HASKELL EXAMPLE

EXCEPTION HANDLING

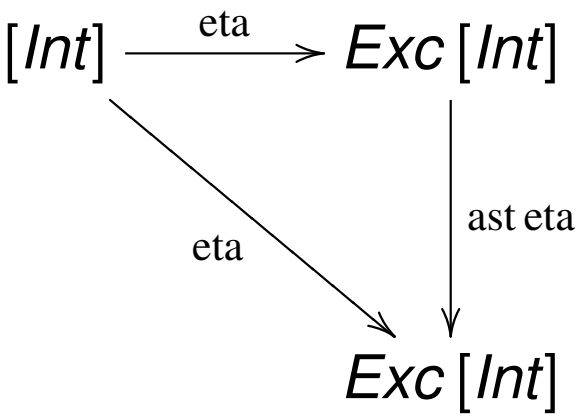
Let fstItem be a function that returns the first item in a nonempty list. This function is not defined on the empty list. Thus fstItem ([ ]) should throw an exception.

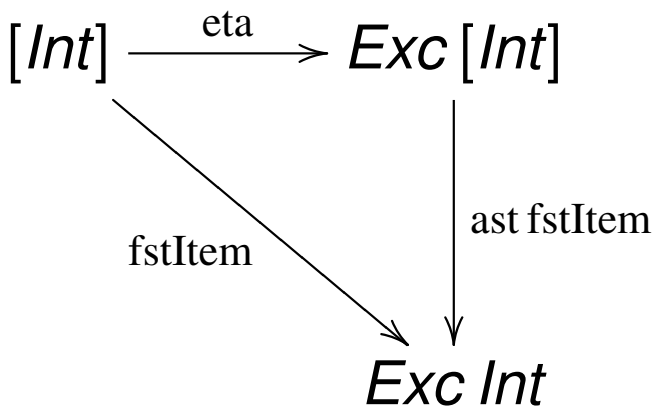
We can use the Exception monad to achieve the desired behavior:

- fstItem :: [a] -> Exc a
- fstItem ([ ]) = ThrowException
- fstItem (a:as) = Iden a

HASKELL EXAMPLE

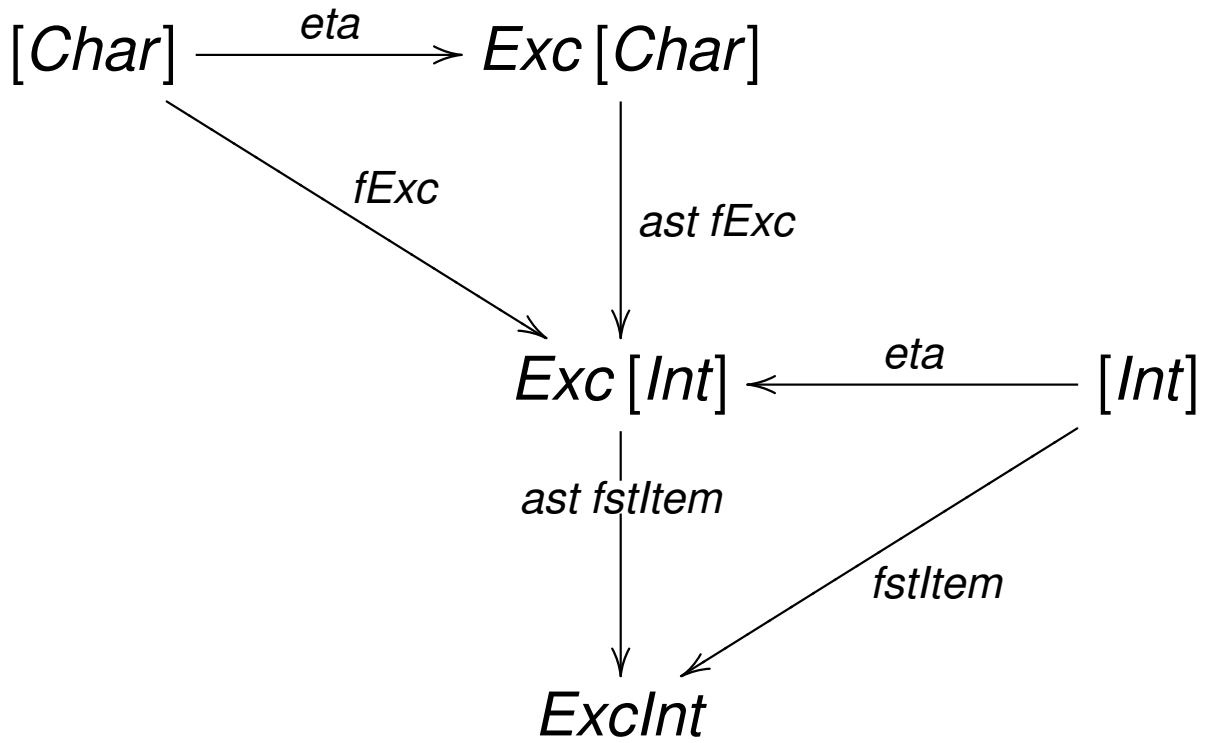
We need to verify that eta and ast satisfy Kleisli Equations 1. 2. and 3. We do so for fstItem on [Int]. Equations 1. and 2. are as follows:





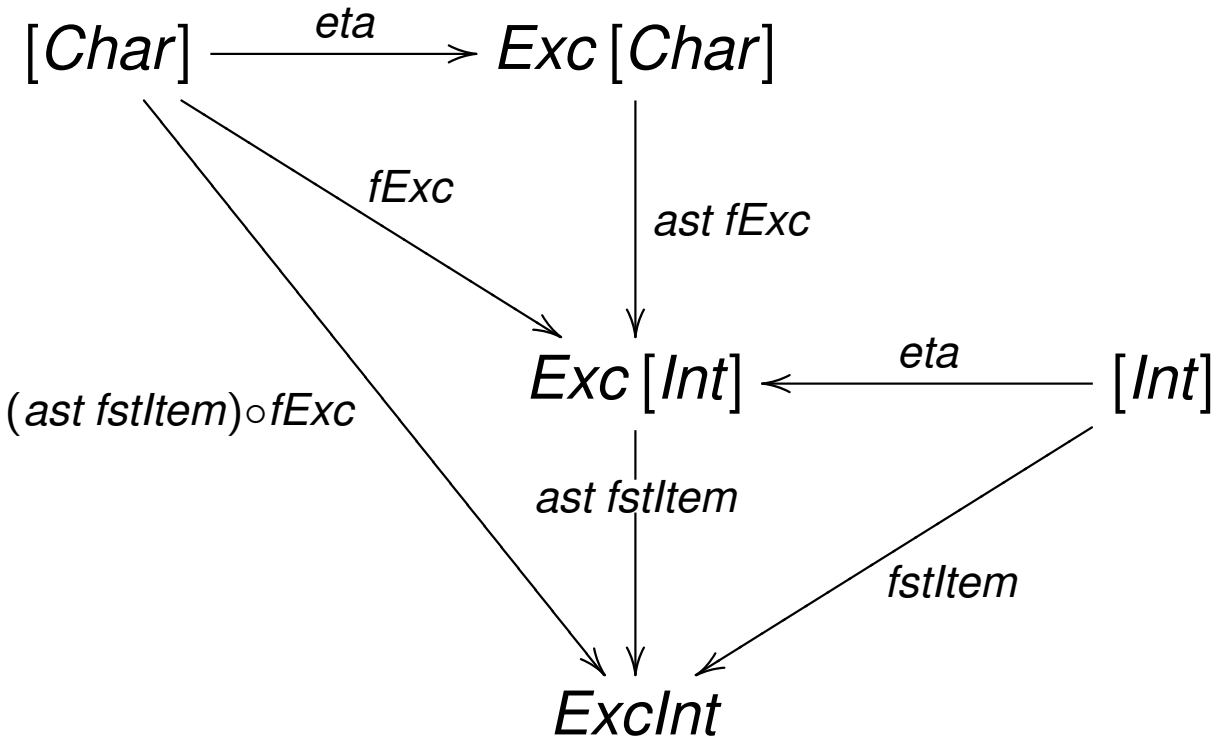
HASKELL EXAMPLE

To verify Equation 3, consider fExc :: [Char] -> Exc [Int], which maps lists of chars to lists of ints, throwing an exception on the empty list. We then have:



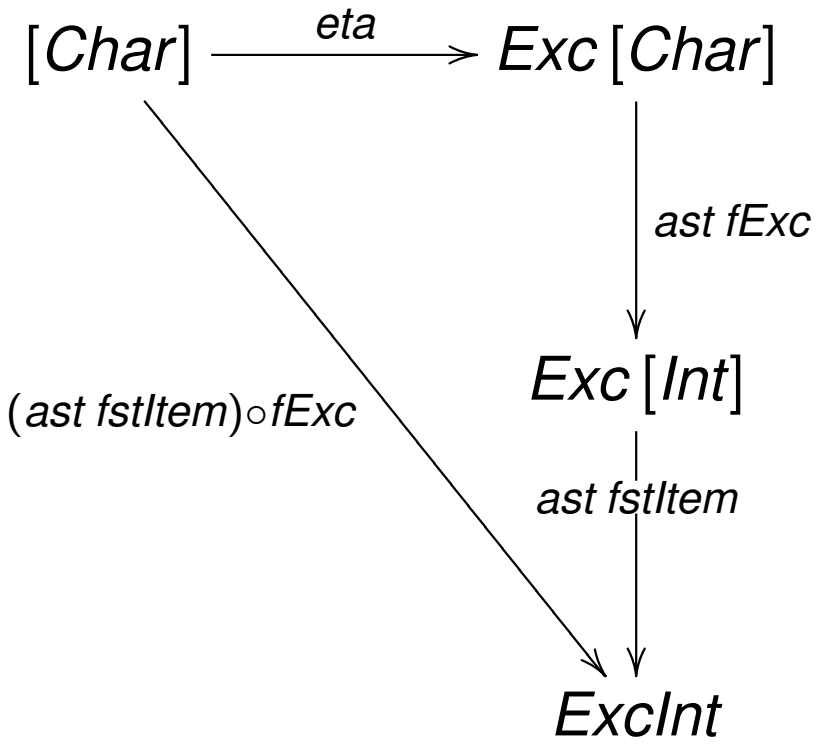
HASKELL EXAMPLE

To verify Equation 3, consider fExc :: [Char] -> Exc [Int], which maps lists of chars to lists of ints, throwing an exception on the empty list. We then have:



HASKELL EXAMPLE

To verify Equation 3, consider fExc :: [Char] -> Exc [Int], which maps lists of chars to lists of ints, throwing an exception on the empty list. We then have:



HASKELL SYNTAX

BIND AND RETURN

In haskell jargon:

- $*$ (ast) is called 'bind', and is represented by »=
- η (eta) is called 'return', and is represented by return

These are the building blocks of monadic computation. To become a super haskell guru, you must master them.

For Great Good

The Haskell

More information on Haskell Syntax and Theory:

- The Haskell: http://www.haskell.org
- Haskell Beginners: http://www.haskell.org/mailman/listinfo/beginners
- Haskell Cafe: http://www.haskell.org/mailman/listinfo/haskell-cafe