```
void readFile() {
  FILE *b;
  char c;

  b = fopen("Yorick", "r");
  c = fgetc(b);
  while (!feof(b)) { putchar(c); c = fgetc(b); }
  fclose(b); }
```

McBride of Strathclyde

presents

arrows of

outrageous fortune

McBride of Strathclyde

presents

Kleisli arrows of outrageous fortune

McBride of Strathclyde

presents

Kleisli arrows of outrageous fortune

(Hamlet, Hoare, Haskell)

NO
RIFF
RAFF

Now that's what I call a spade!

programming to an interface

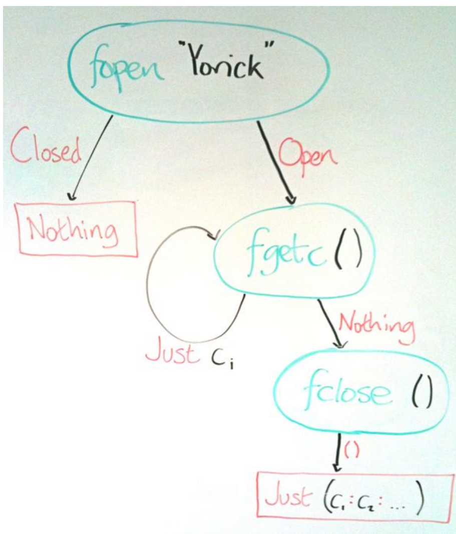
```
data State = Open | Closed

fopen  :: FilePath → State
fgetc  :: ()       → Maybe Char
fclose :: ()       → ()
```

I've hidden the FILE* variable naming the resource, and given a command-response interface.

a program is a strategy tree

nodes are commands
edges cover responses
values delivered at leaves



Strategy trees as data

```
data Strategy x
  = Return x   -- value returned at leaf
  | FOpen FilePath (State      → Strategy x)
  | FGetC ()       (Maybe Char → Strategy x)
  | FClose ()      (()         → Strategy x)
```

One constructor per command, carrying arguments and a callback.

A Kleisli Category

$$\text{Return} :: x \rightarrow \text{Strategy}\ x$$

$$(\ggg) :: (x \rightarrow \text{Strategy}\ y) \rightarrow (y \rightarrow \text{Strategy}\ z) \rightarrow (x \rightarrow \text{Strategy}\ \{\text{-grafting... -}\}\ z)$$

Composition grafts the second strategy to the leaves of the first. The interface determines the strategy type, which has the structure of a *monad*.

commands as monadic operations

$$\begin{aligned}
&\mathsf{fopen} :: \mathsf{FilePath} \rightarrow \mathsf{Strategy\ State}\\
&\mathsf{fopen}\ f = \mathsf{FOpen}\ f\ \mathsf{Return}\\
&\mathsf{fgetc} :: () \rightarrow \mathsf{Strategy\ (Maybe\ Char)}\\
&\mathsf{fgetc}\ v = \mathsf{FGetC}\ v\ \mathsf{Return}\\
&\mathsf{fclose} :: () \rightarrow \mathsf{Strategy}\ ()\\
&\mathsf{fclose}\ v = \mathsf{FClose}\ v\ \mathsf{Return}
\end{aligned}$$

We can implement a *monad homomorphism* or *device driver*

$$\mathsf{runStrategy} :: \mathsf{Strategy}\ x \rightarrow \mathsf{IO}\ x$$

which actually talks to the world.

the general picture (Plotkin-Power)

```
data (:>>:) c r x = c :& (r -> x)
                        -- how to make an x by command-response
data (:+:) f g x = L (f x)
                 | R (g x)
                        -- offer a choice of commands
data f :* x      = Return x
                 | Do (f (f :* x))
                        -- build f-noded trees
```

Our example becomes

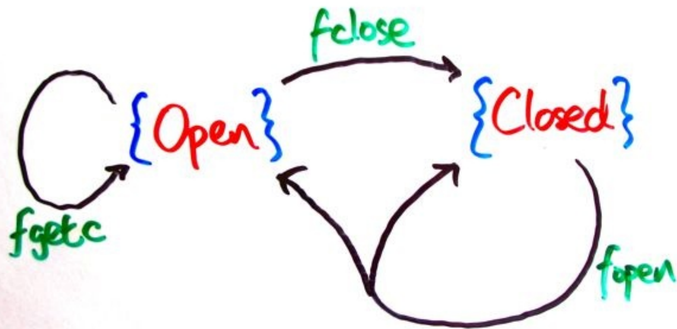
```
type Interface = ((FilePath :>>: State)       :+:
                  (()       :>>: Maybe Char) :+:
                  (()       :>>: ())
                 )
```

and Strategy x = Interface :* x

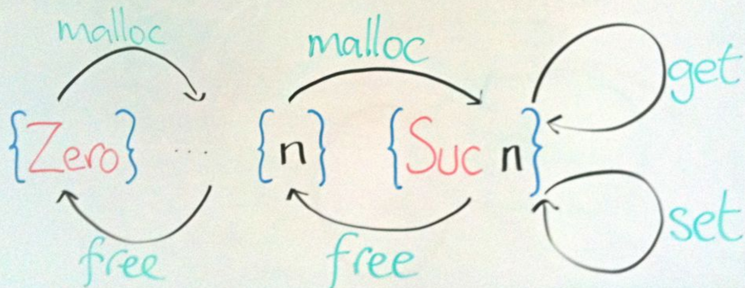
what's missing?

what's missing?
No model of reality.
No checking that action makes sense with respect to state.

spot the problem



less diabolical



Atkey's 'parametrized' monads

Idea: index by *initial* and final *final* states of type i, modelling the world. Equip a type

$$\mathsf{M} :: \{\, i \,\} \to \{\, i \,\} \to * \to *$$

with

$$\begin{aligned}
&\mathsf{return} :: x \to \mathsf{M}\ \{\, i \,\}\ \{\, i \,\}\ x \\
&(\ggg) :: (x \to \mathsf{M}\ \{\, i \,\}\ \{j\}\ y) \to (y \to \mathsf{M}\ \{j\}\ \{\, k \,\}\ z) \to \\
&\qquad\quad (x \to \mathsf{M}\ \{\, i \,\} \qquad\qquad\qquad\qquad\qquad\qquad \{\, k \,\}\ z)
\end{aligned}$$

Grafting with dominoes!

Atkey's 'parametrized' monads

Idea: index by *initial* and final *final* states of type i, modelling the world. Equip a type

$$\mathsf{M} :: \{i\} \to \{i\} \to * \to *$$

with

$$\begin{aligned}
&\mathsf{return} :: x \to \mathsf{M}\ \{i\}\ \{i\}\ x \\
&(\ggg) :: (x \to \mathsf{M}\ \{i\}\ \{j\}\ y) \to (y \to \mathsf{M}\ \{j\}\ \{k\}\ z) \to \\
&\qquad\quad (x \to \mathsf{M}\ \{i\}\ \{k\}\ z)
\end{aligned}$$

Grafting with dominoes!

We might have

$$\begin{aligned}
&\mathsf{malloc} :: () \to \mathsf{M}\ \{n\}\ \{\mathsf{Suc}\ n\}\ () \\
&\mathsf{free} :: () \to \mathsf{M}\ \{\mathsf{Suc}\ n\}\ \{n\}\ () \\
&\mathsf{get} :: \mathsf{Var}\ \{n\} \to \mathsf{M}\ \{n\}\ \{n\}\ \mathsf{Val} \\
&\mathsf{set} :: (\mathsf{Var}\ \{n\}, \mathsf{Val}) \to \mathsf{M}\ \{n\}\ \{n\}\ ()
\end{aligned}$$

social mobility in modern day Haskell – new kinds for types

$$\kappa ::= * \mid \kappa \to \kappa$$

social mobility in modern day Haskell – new kinds for types

$$
\begin{aligned}
\kappa &::= * \\
&\mid \kappa \to \kappa \\
&\mid \{\tau\} \\
&\mid \forall \alpha :: \kappa .\ \kappa
\end{aligned}
$$

$$\tau ::= \ldots \mid \{ce\}$$

what if space doesn't go on forever?

How can we model a malloc which might fail?
We can't predict the outcome state.
Best available bet, *a control operator*:

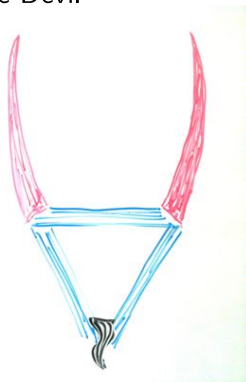
```
ifmalloc :: M {Suc i} {j} x  {-plan for success -}  →
            M {i}     {j} x  {-backup plan -}       →
            M {i}     {j} x
```

We've stepped outside the generic command-response setup.

what's missing?

what's missing?

The Devil



consider *indexed* sets

$$p :: \{\, i \,\} \rightarrow *$$

where the index type i represents the state of the world
(heap size, Open or Closed, etc)
  p is like a *predicate*, but...
...some value

$$v :: p\ \{\, i \,\}$$

represents *concrete evidence* that p holds for i.

By inspecting v at run-time, we might get the goods on i.

I've got negatives.

Two useful kinds of evidence (1)

$$\mathbf{data}\ (\coloneqq) :: * \rightarrow \{\,i\,\} \rightarrow \{\,i\,\} \rightarrow * \ \mathbf{where}$$
$$\quad \mathsf{V} :: a \rightarrow (a \coloneqq \{\,k\,\})\ \{\,k\,\}$$

$(a \coloneqq \{\,k\,\})$ is pronounced "a at key k"

Two useful kinds of evidence (1)

$$\textbf{data}\ (\coloneqq) :: * \rightarrow \{i\} \rightarrow \{i\} \rightarrow *\ \textbf{where}$$
$$\mathsf{V} :: a \rightarrow (a \coloneqq \{k\})\ \{k\}$$

$(a \coloneqq \{k\})$ is pronounced "a at key k"

It means "I have an a and the state is k.".
If you have some $v :: (a \coloneqq \{k\})\ \{i\}$, then you know i is k.

Two useful kinds of evidence (2)

Singletons reify the typing judgment, and act as a run-time witness of the state.

$$(::\text{State}) :: \{\text{State}\} \longrightarrow *$$

$$\begin{aligned} \{\text{Open}\} &:: (::\text{State})\ \{\text{Open}\} \\ \{\text{Closed}\} &:: (::\text{State})\ \{\text{Closed}\} \end{aligned}$$

Two useful kinds of evidence (2)

Singletons reify the typing judgment, and act as a run-time witness of the state.

$$(::\mathsf{State}) :: \{\mathsf{State}\} \rightarrow *$$

$$\{\mathsf{Open}\} \;\; :: (::\mathsf{State})\ \{\mathsf{Open}\}$$
$$\{\mathsf{Closed}\} :: (::\mathsf{State})\ \{\mathsf{Closed}\}$$

If you have some $v :: (::\mathsf{State})\ \{i\}$, then *case analysis* on v will tell you whether i is Open or Closed.

what are the morphisms?

$$\textbf{type}\ p \mathrel{\dot{\to}} q = \forall\ i \cdot p\ \{i\} \to q\ \{i\}$$

Index-respecting functions!

what are the morphisms?

$$\textbf{type}\ p \mathrel{\dot{\to}} q = \forall\ i \cdot p\ \{i\} \to q\ \{i\}$$

Index-respecting functions! Predicate implication!

what are the morphisms?

$$\textbf{type}\ p \mathrel{\dot{\rightarrow}} q = \forall\ i \cdot p\ \{\, i \,\} \rightarrow q\ \{\, i \,\}$$

Index-respecting functions! Predicate implication!

The usual polymorphic identity and composition are the identity and composition. We have a category of *i*-indexed Haskell types.

what are the monads?

Consider

$$\mathsf{M} :: (\{\, i \,\} \to *) \to (\{\, i \,\} \to *)$$

what are the monads?

Consider

$$\mathsf{M} :: (\{\, i \,\} \to *) \to (\{\, i \,\} \to *)$$

A ‘predicate transformer’.

what are the monads?

Consider

$$\mathsf{M} :: (\{\, i \,\} \rightarrow *) \rightarrow (\{\, i \,\} \rightarrow *)$$

A ‘predicate transformer’.
$\mathsf{M}\ p\ \{\, i \,\}$ is a strategy for reaching some state satisfying p, starting in state i.

what are the monads?

Consider

$$\mathsf{M} :: (\{\,i\,\} \to *) \to (\{\,i\,\} \to *)$$

A ‘predicate transformer’.
$\mathsf{M}\ p\ \{\,i\,\}$ is a strategy for reaching some state satisfying p, starting in state i.

$$\begin{aligned}
&\mathsf{return} :: p \mathrel{\dot\to} \mathsf{M}\ p \\
&(\ggg) :: (p \mathrel{\dot\to} \mathsf{M}\ q) \to (q \mathrel{\dot\to} \mathsf{M}\ r) \to \\
&\qquad\qquad (p \mathrel{\dot\to} \mathsf{M}\ \ \text{\{-grafting -\}}\ \ r)
\end{aligned}$$

what are the monads?

Consider

$$\mathsf{M} :: (\{i\} \rightarrow *) \rightarrow (\{i\} \rightarrow *)$$

A 'predicate transformer'.
$\mathsf{M}\ p\ \{i\}$ is a strategy for reaching some state satisfying p, starting in state i.

$$\begin{aligned} &\texttt{skip} :: p \mathrel{\dot\rightarrow} \mathsf{M}\ p \\ &; \quad :: (p \mathrel{\dot\rightarrow} \mathsf{M}\ q) \rightarrow (q \mathrel{\dot\rightarrow} \mathsf{M}\ r) \rightarrow \\ &\qquad (p \mathrel{\dot\rightarrow} \mathsf{M}\ \ \{\text{-grafting -}\}\ \ r) \end{aligned}$$

A Kleisli arrow

$$f :: p \mathrel{\dot\rightarrow} \mathsf{M}\ q$$

is a *Hoare triple*!

Variations on the theme of ‘bind’ (a.k.a. ‘let’)

Demonic bind

$$(?\!\!\succcurlyeq) :: \mathsf{M}\ p\ \{i\} \rightarrow (p \mathrel{\dot{\rightarrow}} \mathsf{M}\ q) \rightarrow \mathsf{M}\ q\ \{i\}$$
$$p \mathbin{?\!\!\succcurlyeq} f = (id \ggg f)\ p$$

Variations on the theme of 'bind' (a.k.a. 'let')

Demonic bind

$$(?\!\!\succcurlyeq) :: \forall\, i \cdot \mathsf{M}\ p\ \{i\} \rightarrow (\forall\, j \cdot p\ \{j\} \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$
$$p\ ?\!\!\succcurlyeq f = (id \ggg f)\ p$$

Variations on the theme of 'bind' (a.k.a. 'let')

Demonic bind

$$(?\!\succcurlyeq) :: \forall\, i \cdot \mathsf{M}\ p\ \{i\} \rightarrow (\forall\, j \cdot p\ \{j\} \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$
$$p\ ?\!\succcurlyeq\ f = (id \ggg f)\ p$$

You choose i, but the *devil* chooses j.

Variations on the theme of 'bind' (a.k.a. 'let')

Demonic bind

$$(?\!\succcurlyeq) :: \forall\, i \cdot \mathsf{M}\ p\ \{i\} \rightarrow (\forall\, j \cdot p\ \{j\} \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$
$$p \mathrel{?\!\succcurlyeq} f = (id \ggg f)\ p$$

You choose i, but the *devil* chooses j.

$$(\Rightarrow\!\succcurlyeq) :: \mathsf{M}\ (a \coloneqq \{j\})\ \{i\} \rightarrow (a \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$

If p is some $a \coloneqq \{j\}$, we don't need to quantify over an unknown state.

Variations on the theme of 'bind' (a.k.a. 'let')

Demonic bind

$$(?\!\!\succeq) :: \forall\, i \cdot \mathsf{M}\ p\ \{i\} \rightarrow (\forall\, j \cdot p\ \{j\} \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$
$$p\ ?\!\!\succeq f = (id \ggg f)\ p$$

You choose i, but the *devil* chooses j.

$$(\Rightarrow\!\!\succeq) :: \mathsf{M}\ (a \coloneqq \{j\})\ \{i\} \rightarrow (a \rightarrow \mathsf{M}\ q\ \{j\}) \rightarrow \mathsf{M}\ q\ \{i\}$$

If p is some $a \coloneqq \{j\}$, we don't need to quantify over an unknown state.
If *all* predicates are $(\coloneqq)$, we get the behaviour of Atkey's parametrized monads.

Demonstration.

conclusions

Monads on indexed sets allow us to model outrageous fortune.

Instead of using Hoare Logic as ‘logical superstructure’ for reasoning, yank it across the Curry-Howard correspondence and use it as ‘logical infrastructure’ for programming.