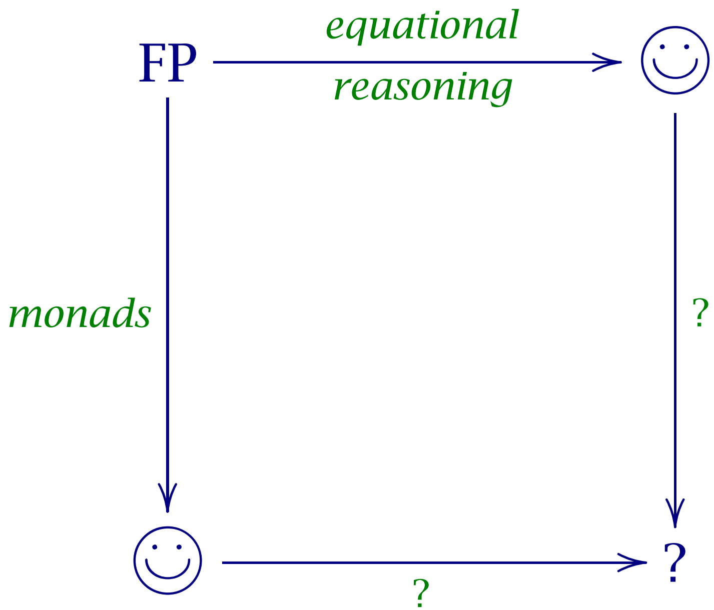
Simple
Monadic
Equational Reasoning

Jeremy Gibbons (joint work with Ralf Hinze)
University of Oxford
WG2.8, March 2011

1. Reasoning with effects?



1.1. Seeing the wood through the trees

At TFP 2008, Hutton & Fulger discuss the 'correctness' of

$$relabel :: Tree\ a \rightarrow Tree\ Int$$

as an effectful (stateful) functional program.

I think they miss two opportunities for abstraction:

- from the specific *effects* (they expand the *State* monad to state-transforming functions), and
- from the *pattern of computation* (they use explicit induction on trees).

This is an attempt to address the first question.
(The second is a story for another time.)

2. Monads

'Ordinary' monads, with the usual laws:

$$\textbf{class}\ \mathit{Monad}\ m\ \textbf{where}$$
$$\quad \mathit{return} :: a \rightarrow m\ a$$
$$\quad (\gg\!=) \ :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$$

Special cases:

$$\mathit{skip} :: \mathit{Monad}\ m \Rightarrow m\ ()$$
$$\mathit{skip} = \mathit{return}\ ()$$

$$(\gg) :: \mathit{Monad}\ m \Rightarrow m\ a \rightarrow m\ b \rightarrow m\ b$$
$$k \gg l = k \gg\!= \mathit{const}\ l$$

2.1. Fallibility

Computations may fail:

$$\textbf{class}\ \mathit{Monad}\ m \Rightarrow \mathit{MonadZero}\ m\ \textbf{where}$$
$$\quad \mathit{mzero} :: m\ a$$

such that

$$\mathit{mzero} \gg k = \mathit{mzero}$$

(I'm curious as to why it's not like this in Haskell 98. . .)

Often we just use

$$\mathit{mzero} = \bot$$

2.2. Guards

Define

$$guard :: MonadZero\ m \Rightarrow Bool \rightarrow m\ ()$$
$$guard\ b = \mathbf{if}\ b\ \mathbf{then}\ skip\ \mathbf{else}\ mzero$$

We'll write '$b!$' for '$guard\ b$'.

Familiar properties:

$$True! = skip$$
$$False! = mzero$$
$$(b_1 \wedge b_2)! = b_1! \gg b_2!$$
$$b_1! \gg k \gg b_2! = b_1! \gg k \quad \Longleftarrow \quad b_1 \Rightarrow b_2$$

2.3. Assertions

For $k :: MonadZero\ ()$, write '$k\ \{b\}$' for

$$\mathbf{do}\ \{\, k; b!\, \} = \mathbf{do}\ \{\, k\, \} \qquad (= k)$$

More generally, for $k :: MonadZero\ a$, define '$k\ \{b\}$' to be:

$$\mathbf{do}\ \{\, a \leftarrow k; b!; return\ a \} = \mathbf{do}\ \{\, a \leftarrow k; return\ a \}$$

By abuse of notation, extend to assertions about multiple statements: suppose statements $s_1; ...; s_n$ contain generators binding variables $v_1, ..., v_m$; write '$s_1; ...; s_n\ \{b\}$' for

$$\mathbf{do}\ \{\, s_1; ...; s_n; b!; return\ (v_1, ..., v_m)\, \} = \mathbf{do}\ \{\, s_1; ...; s_n; return\ (v_1, ..., v_m)\, \}$$

(A similar construction is used by Erkök and Launchbury (2000).)

2.4. Queries

A special class of monadic operations, particularly amenable to manipulation.

A *query* q has no side-effects:

$$\mathbf{do}\ \{a \leftarrow q; k\} = \mathbf{do}\ \{k\} \quad \text{-- when } k \text{ doesn't depend on } a$$

and is consistent:

$$\mathbf{do}\ \{a_1 \leftarrow q; a_2 \leftarrow q; k\ a_1\ a_2\} = \mathbf{do}\ \{a \leftarrow q; k\ a\ a\}$$

(They're not just the pure operations, ie those of the form $return\ a$. Consider $get :: State\ s\ s$ of the state monad.)

3. A counter example

A counting monad:

$$\textbf{class}\ \mathit{Monad}\ m \Rightarrow \mathit{MonadCount}\ m\ \textbf{where}$$
$$\quad \mathit{tick} \ \ :: m\ ()$$
$$\quad \mathit{total} :: m\ \mathit{Int}$$

where *total* is a query, and

$$n \leftarrow \mathit{total}; \mathit{tick}; n' \leftarrow \mathit{total}\ \{n' = n + 1\}$$

(exploiting our abuse of notation).

3.1. Towers of Hanoi—specification

Given this program:

$$
\begin{aligned}
&\mathit{hanoi} :: \mathit{MonadCount}\ m \Rightarrow \mathit{Int} \rightarrow m\ ()\\
&\mathit{hanoi}\ 0 \qquad\quad\ = \mathit{skip}\\
&\mathit{hanoi}\ (n+1) = \mathbf{do}\ \{\mathit{hanoi}\ n; \mathit{tick}; \mathit{hanoi}\ n\}
\end{aligned}
$$

we claim:

$$t \leftarrow \mathit{total}; \mathit{hanoi}\ n; u \leftarrow \mathit{total}\ \{2^n - 1 = u - t\}$$

Proof by induction on n. The base case is immediate. Inductive step. . .

3.2. Reasoning

$$
\begin{array}{ll}
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ (n+1); u \leftarrow \mathit{total}; (2^{n+1} - 1 = u - t)!\} \\
= & [\![\ \text{definition of } \mathit{hanoi}\ ]\!] \\
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ n; \mathit{tick}; \mathit{hanoi}\ n; u \leftarrow \mathit{total}; (2^{n+1} - 1 = u - t)!\} \\
= & [\![\ \text{inserting some queries}\ ]\!] \\
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ n; u' \leftarrow \mathit{total}; \mathit{tick}; t' \leftarrow \mathit{total}; \\
& \quad \mathit{hanoi}\ n; u \leftarrow \mathit{total}; (2^{n+1} - 1 = u - t)!\} \\
= & [\![\ \text{inductive hypothesis; } \mathit{tick}\ ]\!] \\
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ n; u' \leftarrow \mathit{total}; (2^{n} - 1 = u' - t)!; \mathit{tick}; t' \leftarrow \mathit{total}; \\
& \quad (t' = u' + 1)!; \mathit{hanoi}\ n; u \leftarrow \mathit{total}; (2^{n} - 1 = u - t')!; (2^{n+1} - 1 = u - t)!\} \\
= & [\![\ \text{arithmetic: } 2^{n+1} - 1 = u - t \text{ follows from other guards}\ ]\!] \\
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ n; u' \leftarrow \mathit{total}; (2^{n} - 1 = u' - t)!; \mathit{tick}; t' \leftarrow \mathit{total}; \\
& \quad (t' = u' + 1)!; \mathit{hanoi}\ n; u \leftarrow \mathit{total}; (2^{n} - 1 = u - t')!\} \\
= & [\![\ \text{redundant guards, definition of } \mathit{hanoi}\ ]\!] \\
& \mathbf{do}\ \{t \leftarrow \mathit{total}; \mathit{hanoi}\ (n+1); u \leftarrow \mathit{total}\}
\end{array}
$$

4. Tree relabelling

A monad for generating fresh symbols:

type *Symbol* = ...
instance *Eq Symbol* **where** ...

class *Monad m* ⇒ *MonadGensym m* **where**
  fresh :: *m Symbol*
  used :: *m* (*Set Symbol*)

such that *used* (only used in reasoning) is a query, and

$$x \leftarrow used; n \leftarrow fresh; y \leftarrow used\ \{x \subseteq y \land n \in y - x\}$$

4.1. Specification

Tree relabelling:

$$\mathbf{data}\ \mathit{Tree}\ a = \mathit{Tip}\ a \mid \mathit{Bin}\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

$$\begin{aligned}
&\mathit{relabel} :: \mathit{MonadGensym}\ m \Rightarrow \mathit{Tree}\ a \to m\ (\mathit{Tree}\ \mathit{Symbol})\\
&\mathit{relabel}\ (\mathit{Leaf}\ a) = \mathbf{do}\ \{ n \leftarrow \mathit{fresh}; \mathit{return}\ (\mathit{Leaf}\ n) \}\\
&\mathit{relabel}\ (\mathit{Bin}\ t\ u) = \mathbf{do}\ \{ t' \leftarrow \mathit{relabel}\ t; u' \leftarrow \mathit{relabel}\ u; \mathit{return}\ (\mathit{Bin}\ t'\ u') \}
\end{aligned}$$

(in fact, an idiomatic *traverse*), satisfies

$$x \leftarrow \mathit{used}; t' \leftarrow \mathit{relabel}\ t; y \leftarrow \mathit{used}\ \{\mathit{distinct}\ t' \wedge \mathit{labels}\ t' \subseteq y - x\}$$

where

$$\begin{aligned}
&\mathit{distinct} :: \mathit{Tree}\ \mathit{Symbol} \to \mathit{Bool}\\
&\mathit{labels}\ \ :: \mathit{Tree}\ \mathit{Symbol} \to \mathit{Set}\ \mathit{Symbol}
\end{aligned}$$

(written d and l below, for short).

4.2. Reasoning: base case

$$
\begin{array}{ll}
 & \textbf{do}\ \{x \leftarrow \mathit{used}; v \leftarrow \mathit{relabel}\ (\mathit{Leaf}\ a); y \leftarrow \mathit{used}; (d\ v \wedge l\ v \subseteq y - x)!\} \\
= & [\![\ \text{definition of } \mathit{relabel}\ ]\!] \\
 & \textbf{do}\ \{x \leftarrow \mathit{used}; n \leftarrow \mathit{fresh}; \textbf{let}\ v = \mathit{Leaf}\ n; y \leftarrow \mathit{used}; (d\ v \wedge l\ v \subseteq y - x)!\} \\
= & [\![\ \text{definition of } d, l\ ]\!] \\
 & \textbf{do}\ \{x \leftarrow \mathit{used}; n \leftarrow \mathit{fresh}; \textbf{let}\ v = \mathit{Leaf}\ n; y \leftarrow \mathit{used}; (\mathit{True} \wedge \{n\} \subseteq y - x)!\} \\
= & [\![\ \text{axiom for } \mathit{fresh}\ ]\!] \\
 & \textbf{do}\ \{x \leftarrow \mathit{used}; n \leftarrow \mathit{fresh}; \textbf{let}\ u = \mathit{Leaf}\ n; y \leftarrow \mathit{used}\} \\
= & [\![\ \text{folding definitions}\ ]\!] \\
 & \textbf{do}\ \{x \leftarrow \mathit{used}; v \leftarrow \mathit{relabel}\ (\mathit{Leaf}\ a); y \leftarrow \mathit{used}\}
\end{array}
$$

4.3. Reasoning: inductive step

$$
\begin{array}{ll}
& \mathbf{do}\ \{x \leftarrow used; v \leftarrow relabel\ (Bin\ t\ u); z \leftarrow used; (d\ v \wedge l\ v \subseteq z - x)!\} \\
= & [\![\ \text{definition of } relabel\ ]\!] \\
& \mathbf{do}\ \{x \leftarrow used; t' \leftarrow relabel\ t; u' \leftarrow relabel\ u; \mathbf{let}\ v = Bin\ t'\ u'; z \leftarrow used; \\
& \qquad (d\ v \wedge l\ v \subseteq z - x)!\} \\
= & [\![\ \text{definition of } d, l\ ]\!] \\
& \mathbf{do}\ \{x \leftarrow used; t' \leftarrow relabel\ t; u' \leftarrow relabel\ u; \mathbf{let}\ v = Bin\ t'\ u'; z \leftarrow used; \\
& \qquad (d\ t' \wedge d\ u' \wedge l\ t' \cap l\ u' = \emptyset \wedge l\ t' \cup l\ u' \subseteq z - x)!\} \\
= & [\![\ \text{induction}\ ]\!] \\
& \mathbf{do}\ \{x \leftarrow used; t' \leftarrow relabel\ t; y \leftarrow used; (d\ t' \wedge l\ t' \subseteq y - x)!; \\
& \qquad u' \leftarrow relabel\ u; z \leftarrow used; (d\ u' \wedge l\ u' \subseteq z - y)!; \mathbf{let}\ v = Bin\ t'\ u'; \\
& \qquad (d\ t' \wedge d\ u' \wedge l\ t' \cap l\ u' = \emptyset \wedge l\ t' \cup l\ u' \subseteq z - x)!\} \\
= & [\![\ \text{queries, redundant guards, folding definitions}\ ]\!] \\
& \mathbf{do}\ \{x \leftarrow used; v \leftarrow relabel\ (Bin\ t\ u); z \leftarrow used\}
\end{array}
$$

5. Towers of Hanoi, more directly

Hoare-style reasoning is a bit painfully long-winded:
repeat the program on every line, gradually discharging guards.

Sometimes a more direct approach works. In fact,

$$hanoi\ n = rep\ (2^n - 1)\ tick$$

where

$$\begin{aligned} & rep :: Monad\ m \Rightarrow Int \to m\ () \to m\ () \\ & rep\ 0\ \quad\quad\ ma = skip \\ & rep\ (n+1)\ ma = ma \gg rep\ n\ ma \end{aligned}$$

In particular, note that

$$rep\ (m + n)\ ma = rep\ m\ ma \gg rep\ n\ ma$$

5.1. More direct proof

...by induction on n. Base case is trivial. For inductive step,

$$
\begin{aligned}
& \mathit{hanoi}\ (n+1) \\
= \quad & [\![\ \text{definition of } \mathit{hanoi}\ ]\!] \\
& \mathit{hanoi}\ n \gg \mathit{tick} \gg \mathit{hanoi}\ n \\
= \quad & [\![\ \text{inductive hypothesis}\ ]\!] \\
& \mathit{rep}\ (2^n - 1)\ \mathit{tick} \gg \mathit{tick} \gg \mathit{rep}\ (2^n - 1)\ \mathit{tick} \\
= \quad & [\![\ \text{composition}\ ]\!] \\
& \mathit{rep}\ ((2^n - 1) + 1 + (2^n - 1))\ \mathit{tick} \\
= \quad & [\![\ \text{arithmetic}\ ]\!] \\
& \mathit{rep}\ (2^{n+1} - 1)\ \mathit{tick}
\end{aligned}
$$

But I don’t see how to do tree relabelling in this more direct style...

6. Probabilistic computations

Probability distributions form a monad (Giry, Jones, Ramsey, Erwig. . .).

For simplicity, only finitely-supported distributions here:

$$\mathbf{class}\ \mathit{Monad}\ m \Rightarrow \mathit{MonadProb}\ m\ \mathbf{where}$$
$$\quad \mathit{choice} :: \mathit{Rational} \to m\ a \to m\ a \to m\ a$$

where the rationals are constrained to the unit interval.

Following Hoare, let's write '$mx \triangleleft p \triangleright my$' for '$\mathit{choice}\ p\ mx\ my$'.

6.1. Laws of choice

Unit, idempotence, commutativity:

$$
\begin{aligned}
mx \lhd 0 \rhd my &= my \\
mx \lhd 1 \rhd my &= mx \\
mx \lhd p \rhd mx &= mx \\
mx \lhd p \rhd my &= my \lhd 1 - p \rhd mx
\end{aligned}
$$

A kind of associativity:

$$
\begin{aligned}
& mx \lhd p \rhd (my \lhd q \rhd mz) = (mx \lhd r \rhd my) \lhd s \rhd mz \\
& \quad \Longleftarrow p = r\, s \wedge (1 - s) = (1 - p)\,(1 - q)
\end{aligned}
$$

Bind distributes over choice, in both directions:

$$
\begin{aligned}
mx \gg\!= \lambda a \to (k_1\ a) \lhd p \rhd (k_2\ a) &= (mx \gg\!= k_1) \lhd p \rhd (mx \gg\!= k_2) \\
mx \lhd p \rhd my \gg\!= k &= (mx \gg\!= k) \lhd p \rhd (my \gg\!= k)
\end{aligned}
$$

6.2. Normal form

Finite mappings from outcomes to probabilities (ignore order, disregard weightless entries, weights sum to one, amalgamate duplicates):

$$\mathbf{newtype}\ \mathit{Distribution}\ a = D\{\ \mathit{unD} :: [\,(a, \mathit{Rational})\,]\,\}$$

All you need to interpret a distribution is *choice*:

$$\begin{aligned}&\mathit{fromDist} :: \mathit{MonadProb}\ m \Rightarrow \mathit{Distribution}\ a \to m\ a\\&\mathit{fromDist}\ d = \mathit{fst}\ (\mathit{foldr}1\ \mathit{combine}\ [\,(\mathit{return}\ a, p) \mid (a, p) \leftarrow \mathit{unD}\ d, p > 0\,])\\&\quad \mathbf{where}\ \mathit{combine}\ (\mathit{mx}, p)\ (\mathit{my}, q) = (\mathit{mx} \lhd {}^{p}/_{p+q} \rhd \mathit{my}, p + q)\end{aligned}$$

For example,

$$\begin{aligned}&\mathit{uniform} :: \mathit{MonadProb}\ m \Rightarrow [\,a\,] \to m\ a\\&\mathit{uniform}\ x = \mathit{fromDist}\ (D\,[\,(a, p) \mid a \leftarrow x\,])\ \mathbf{where}\ p = 1\ /\ \mathit{length}\ x\end{aligned}$$

6.3. Implementation

Moreover, *Distribution* itself is a fine instance of *MonadProb*:

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Monad}\ \mathit{Distribution}\ \textbf{where}\\
&\quad \mathit{return}\ a = D\,[\,(a,1)\,]\\
&\quad \mathit{px} \gg\!= f\ \ = D\,[\,(b, p \times q) \mid (a,p) \leftarrow \mathit{unD}\ \mathit{px}, (b,q) \leftarrow \mathit{unD}\ (f\ a)\,]\\
&\textbf{instance}\ \mathit{MonadProb}\ \mathit{Distribution}\ \textbf{where}\\
&\quad \mathit{ma} \lhd p \rhd \mathit{mb} = D\ (\mathit{scale}\ p\ (\mathit{unD}\ \mathit{ma}) +\!\!+ \mathit{scale}\ (1-p)\ (\mathit{unD}\ \mathit{mb}))\\
&\qquad \textbf{where}\ \mathit{scale}\ r\ \mathit{pas} = [\,(a, r \times p) \mid (a,p) \leftarrow \mathit{pas}\,]
\end{aligned}
$$

(Kidd points out that *Distribution* = *WriterT Rational* (*ListT Identity*), using the writer monad from the monoid of rationals with multiplication.)

6.4. Monty Hall

$$\mathbf{data}\ \mathit{Door} = A \mid B \mid C\ \mathbf{deriving}\ (\mathit{Eq}, \mathit{Show})$$
$$\mathit{doors} = [A, B, C]$$

$$\mathit{hide} :: \mathit{MonadProb}\ m \Rightarrow m\ \mathit{Door}$$
$$\mathit{hide} = \mathit{uniform}\ \mathit{doors}$$

$$\mathit{pick} :: \mathit{MonadProb}\ m \Rightarrow m\ \mathit{Door}$$
$$\mathit{pick} = \mathit{uniform}\ \mathit{doors}$$

$$\mathit{tease} :: \mathit{MonadProb}\ m \Rightarrow \mathit{Door} \to \mathit{Door} \to m\ \mathit{Door}$$
$$\mathit{tease}\ h\ p = \mathit{uniform}\ (\mathit{doors} \setminus\!\setminus [h, p])$$

$$\mathit{switch} :: \mathit{MonadProb}\ m \Rightarrow \mathit{Door} \to \mathit{Door} \to m\ \mathit{Door}$$
$$\mathit{switch}\ p\ t = \mathit{return}\ (\mathit{head}\ (\mathit{doors} \setminus\!\setminus [p, t]))$$

$$\mathit{stick} :: \mathit{MonadProb}\ m \Rightarrow \mathit{Door} \to \mathit{Door} \to m\ \mathit{Door}$$
$$\mathit{stick}\ p\ t = \mathit{return}\ p$$

6.5. The whole story

Monty's script:

$$
\begin{array}{l}
play :: MonadProb\ m \Rightarrow (Door \to Door \to m\ Door) \to m\ Bool \\
play\ strategy = \\
\quad \mathbf{do} \\
\quad\quad h \leftarrow hide \qquad\qquad \text{-- host hides the car behind door } h \\
\quad\quad p \leftarrow pick \qquad\qquad \text{-- you pick door } p \\
\quad\quad t \leftarrow tease\ h\ p \qquad \text{-- host teases you with door } t\ (\not\equiv h, p) \\
\quad\quad s \leftarrow strategy\ p\ t \quad \text{-- you choose, based on } p \text{ and } t \text{ (but not } h\text{!)} \\
\quad\quad return\ (s \equiv h) \qquad \text{-- you win iff your choice } s \text{ equals } h
\end{array}
$$

6.6. In support of Marilyn Vos Savant

It's a straightforward proof by equational reasoning that

$$\begin{aligned} \mathit{play\ switch} &= \mathit{uniform}\ [\mathit{True}, \mathit{True}, \mathit{False}] \\ \mathit{play\ stick} &= \mathit{uniform}\ [\mathit{False}, \mathit{False}, \mathit{True}] \end{aligned}$$

The key is that separate uniform distributions are independent:

$$\mathbf{do}\ \{a \leftarrow \mathit{uniform}\ x; b \leftarrow \mathit{uniform}\ y; \mathit{return}\ (a, b)\} = \mathit{uniform}\ (\mathit{cp}\ x\ y)$$

where

$$\begin{aligned} &\mathit{cp} :: [a] \rightarrow [b] \rightarrow [(a, b)] \\ &\mathit{cp}\ x\ y = [(a, b) \mid a \leftarrow x, b \leftarrow y] \end{aligned}$$

(Ask me over a beer. . .)

7. Combining probability and nondeterminism

Nobody said that Monty has to play fair.
He has a free choice in hiding the car, and in teasing you.

To model this, we need to combine probabilism with nondeterminism:

$$\textbf{class}\ \mathit{MonadZero}\ m \Rightarrow \mathit{MonadPlus}\ m\ \textbf{where}$$
$$\quad \mathit{mplus} :: m\ a \rightarrow m\ a \rightarrow m\ a$$

such that *mzero* and *mplus* form a monoid, and

$$(m\ \text{`}\mathit{mplus}\text{`}\ n) \gg\!= k = (m \gg\!= k)\ \text{`}\mathit{mplus}\text{`}\ (n \gg\!= k)$$

Happily, although monads do not compose in general, $[\,\mathit{Distribution}\ a\,]$ is a monad. Moreover, it is a *MonadProb* and a *MonadPlus* too.

(So is $\mathit{Distribution}\ [\,a\,]$, but I think that doesn't help.)

(There's a nice tale in terms of monad transformers.)

7.1. A simple example: mixing choices

A fair coin:

$$\mathit{coin} :: \mathit{MonadProb}\ m \Rightarrow m\ \mathit{Bool}$$
$$\mathit{coin} = (\mathit{return}\ \mathit{True}) \lhd {}^{1}/_{2} \rhd (\mathit{return}\ \mathit{False})$$

An arbitrary choice:

$$\mathit{arb} :: \mathit{MonadPlus}\ m \Rightarrow m\ \mathit{Bool}$$
$$\mathit{arb} = \mathit{return}\ \mathit{True}\ \text{`}\mathit{mplus}\text{`}\ \mathit{return}\ \mathit{False}$$

Two combinations:

$$\mathit{arbcoin}, \mathit{coinarb} :: (\mathit{MonadPlus}\ m, \mathit{MonadProb}\ m) \Rightarrow m\ \mathit{Bool}$$
$$\mathit{arbcoin} = \mathbf{do}\ \{ a \leftarrow \mathit{arb}; c \leftarrow \mathit{coin}; \mathit{return}\ (a == c) \}$$
$$\mathit{coinarb} = \mathbf{do}\ \{ c \leftarrow \mathit{coin}; a \leftarrow \mathit{arb}; \mathit{return}\ (a == c) \}$$

What do you think they do?

7.2. ...as sets of distributions

Define

$$\textbf{type}\ \mathit{NondetProb}\ a = [\,\mathit{Distribution}\ a\,]$$

Then (with suitable *shows*):

$$*\mathit{Main}\rangle\ \mathit{arbcoin} :: \mathit{NondetProb}\ \mathit{Bool}$$
$$[\,[\,(\mathit{True}, {}^{1}/_{2}), (\mathit{False}, {}^{1}/_{2})\,],$$
$$[\,(\mathit{False}, {}^{1}/_{2}), (\mathit{True}, {}^{1}/_{2})\,]\,]$$

$$*\mathit{Main}\rangle\ \mathit{coinarb} :: \mathit{NondetProb}\ \mathit{Bool}$$
$$[\,[\,(\mathit{True}, {}^{1}/_{2}), (\mathit{False}, {}^{1}/_{2})\,],$$
$$[\,(\mathit{True}, {}^{1}/_{2}), (\mathit{True}, {}^{1}/_{2})\,],$$
$$[\,(\mathit{False}, {}^{1}/_{2}), (\mathit{False}, {}^{1}/_{2})\,],$$
$$[\,(\mathit{False}, {}^{1}/_{2}), (\mathit{True}, {}^{1}/_{2})\,]\,]$$

7.3. . . . as expectations

$$\textbf{class}\ \mathit{MonadProb}\ m \Rightarrow \mathit{MonadExpect}\ m\ \textbf{where}$$
$$\quad \mathit{expect} :: (\mathit{Ord}\ n, \mathit{Fractional}\ n) \Rightarrow m\ a \to (a \to n) \to n$$

$$\textbf{instance}\ \mathit{MonadExpect}\ \mathit{NondetProb}\ \textbf{where} \qquad \text{-- morally}$$
$$\quad \mathit{expect}\ \mathit{px}\ h = \mathit{minimum}\ (\mathit{map}\ (\mathit{mean}\ h \circ \mathit{unD})\ \mathit{px})\ \textbf{where}$$
$$\qquad \mathit{mean}\ h\ \mathit{aps} = \mathit{sum}\,[\,p \times f\ a \mid (a, p) \leftarrow \mathit{aps}\,]\ /\ \mathit{sum}\ (\mathit{map}\ \mathit{snd}\ \mathit{aps})$$

Your reward is 1 if the booleans agree, and 0 otherwise:

$$\mathit{reward}\ b = \textbf{if}\ b\ \textbf{then}\ 1\ \textbf{else}\ 0$$

Then:

$$*\mathit{Main}\rangle\ \mathit{expect}\ (\mathit{arbcoin} :: \mathit{NondetProb}\ \mathit{Bool})\ \mathit{reward}$$
$$^{1}/_{2}$$
$$*\mathit{Main}\rangle\ \mathit{expect}\ (\mathit{coinarb} :: \mathit{NondetProb}\ \mathit{Bool})\ \mathit{reward}$$
$$0$$

7.4. Back to nondeterministic Monty...

We could define instead:

$$hide :: MonadPlus\ m \Rightarrow m\ Door$$
$$hide = arbitrary\ doors$$

$$tease :: MonadPlus\ m \Rightarrow Door \rightarrow Door \rightarrow m\ Door$$
$$tease\ h\ p = arbitrary\ (doors \setminus\setminus [h, p])$$

where

$$arbitrary :: MonadPlus\ m \Rightarrow [a] \rightarrow m\ a$$
$$arbitrary = foldr\ mplus\ mzero \circ map\ return$$

I believe that the calculation carries through just as before: still

$$play\ switch = uniform\ [True, True, False]$$
$$play\ stick = uniform\ [False, False, True]$$

8. Summary

- axiomatic approach to reasoning with effects
- simple and generic
- smacks of ‘algebraic theories of effects’ (Plotkin & Power, Lawvere) (in particular, partiality and continuations do not arise from algebraic theories)
- IO is uninteresting?
- more examples wanted!