CIS 425 12-5-2019 notes

Jonathan Fujii ; Toby Wong

December 2019

1 Recap

This is a recap from the previous class.

We started with something like:

$$head(show(x/y))$$

and we made safe versions like:

$$apply\ Maybe\ safeHead(map\ Maybe\ Show(safeDiv\ x\ y))$$

EQ 1

In practice, we don't see "apply Maybe" but rather "bind" and the operator ">>=" with the order of variables reversed

Example :

$$applyLog(mulLog\ 2)(addLog\ 3\ 4)$$

$$==\ addLog\ 3\ 4\ \ >>=\ mulLog\ 2$$

$$==\ addLog\ 3\ 4\ \ >>=\ \backslash x\ =>\ mulLog\ 2\ x$$

We can break this down as to what this function does.

First adding, then multiplying, and $\backslash x$ is always a function.

$$indentations\ =\ addLog\ 3\ 4\ \ >>=\ \backslash x\ \rightarrow$$

$$mulLog\ 2\ x$$

From above, we are assigning x to the addition result of 3 and 4.

Furthering into the neater syntax form,

$$do\ x\ \leftarrow\ addLog\ 3\ 4;$$

$$mulLog\ 2\ x$$

Now we try the converting EQ 1.

$$\begin{aligned}do\ z\ \leftarrow\ safeDiv\ x\ y;\\ s\ =\ show\ z;\\ safeHead\ s\end{aligned}$$

From the first line, the $<-$ operator gets the result from $safeDiv\ x\ y$ and binds the return result to z. Which it $MAYBE$ null or something else might happen.

On the second line, s just gets assigned the result of $show\ z$. We don't expect a side effect.

$$\begin{aligned}Monad\ =\ & type\ enhancement\ (operator)\\ & +bind\ (apply)\\ & +return\ (just)\end{aligned}$$

2 Overloading

You don't necessarily need to know this section.

Example :

$1+2$	(We should use integer addition at runtime)
$0.5+3.2$	(We should use float addition at runtime)
$addLog\ 3\ 4$	(We should use the addLog definition at runtime)

Let's look at different types of A : *do* - statements.

M	(This would be the last line of a do expression)
$x\ \leftarrow\ M\ ;\ s$	(Run M, some side effects might occur, but if it returns, bind the value to x)
$M\ ;\ s$	(Run M soley to look at the side effects)
$x\ =\ M\ ;\ s$	(If M has no side effects, bind the return value to x)

Showed the syntax and it gives the intuition as to what each statement does.

Example :

$do\, M = \ M$ (Has no difference than just running M)

$do\, x \ \leftarrow \ M \ ; \ s \ = \ M \ >>= \ /x \ \rightarrow \ do\, s$

$do\, M \ ; \ s \ = \ M \ >> \ S$ (>> operator means "then")

$do\, x \ = \ M \ ; \ s = \ let\ x \ = \ M\ in\ do\ s$

Example for "then" operator:

$m1 \ >> \ m2 \ = \ m1 \ >>= \ /x \ \rightarrow \ m2$

$$tell :: string \rightarrow ((), string)$$

tell takes a string that we intend to write to the log

$$tell\ L = ((), L)$$

Another way to define that function

$$addLog\ x\ y = do\ tell\ \text{``}added.\text{''};$$

$$return(x + y)$$

$$(de - sugar) = tell\text{``}added.\text{''} >> return(x + y)$$

$$(inline) = ((\ ), \text{``}added.\text{''}) >> (x + y, \text{`` ''})$$

$$(>>) = (x + y, \text{``}added.\text{''} + + \text{`` ''})$$

e.g.

$(x, L1) >> (y, L2) = (y, L1\ + + \ L2)$ (++ means decantenated)
$return\ x = (x,\ \text{`` ''})$

We can define the function *guard* which can act as an if else function without repeating code.

$guard :: Bool- > Maybe\ L$

$guard\ b = if\ b$

$then\ Just()$ (basically just return())

$else\ Nothing$

The function below basically says: $if\ y\ isn't\ 0,\ return\ x/y$:

$$safeDiv\ x\ y\ =\ do\ guard\ (y\ ! = 0)$$

$$return\ (x/y)$$

The function below basically says: $if\ there\ is\ nothing,\ then\ stop$:

$$Nothing\ >>\ m = Nothing$$

$$Just\ x\ >>\ m = m$$

$$return\ x\ =\ Just\ x$$

Question: What happens if we have safeDiv and pass in some number 0? (Which most likely an error.)

$$safeDiv\ x\ 0\ =\ do\ guard(0\ ! = 0);$$

$$return\ (x/0)$$

De-sugar Walkthrough:

$$=\ guard\ (0\ \neq\ 0)\ >>\ return\ (x\ /\ 0)$$

$$=\ Nothing\ >>\ Just\ (x\ /\ 0)$$

$=\ Nothing$ (Since "Nothing" followed by another statement is nothing, we ignore the rest.)

Now let's try an actual valid value

$$safeDiv\ x\ 2\ =\ do\ guard\ (2\ ! =\ 0);$$

$$return\ (x\ /\ 2)$$

De-sugar Walkthrough:

$$= guard\ (2\ \neq\ 0)\ >>\ return\ (x\ /\ 2)$$

$$= Just()\ >>\ Just\ (x\ /\ 2)$$

$$= Just(x\ /\ 2)$$

3 The Monad Laws

Law 1 (Unit right):
$do\ \ x\ \leftarrow\ M$
$\quad return\ x$
$=\ do\ M$

Law 2 (Unit left):
$do\ \ x\ \leftarrow return\ M,$
$\quad S$
$=\ let\ x = M\ in\ do\ S$

Law 3 (Association):
$do\ \ x\ \leftarrow (do\ \ y \leftarrow\ M,\ S1);$
$\quad S2$
$=\ do\ \ y \leftarrow\ M,$
$\quad x \leftarrow do\ \ S1;$
$\quad S2$

Other Examples of Monad Laws:

Law 1: $m\ \ >>=\ return\ \ =\ \ m$

Law 2: $return\ x\ \ >>=\ f\ \ =\ \ fx$

Law 3: $(m\ \ >>=\ f)\ \ >>=\ g$
$\quad m\ \ >>=\ (\backslash x \rightarrow\ fx\ \ >>=\ g)$

Monad for logging from previous class:

$return\ x\ =\ (x,\ \text{“ ”})$

$(x,\ L)\ \ >>=\ f\ \ =\ Let(\ y, L') = fx$

$in\ (y, L\ \ ++\ L')$

(x, $L)\ \ >>=\ return$

$= let\ (y,\ L')\ \ =\ return\ x$

$in\ (y,\ L\ \ ++\ L')$

$= let\ (y,\ L')\ \ =\ (x,\ \text{“ ”})\ in\ (y,\ L\ \ ++\ L')$

$= (x,\ L\ \ ++\ \text{“ ”})\ \ =\ (x,\ L)$