PA1 Haskell Tutorial

ANTWANE MASON

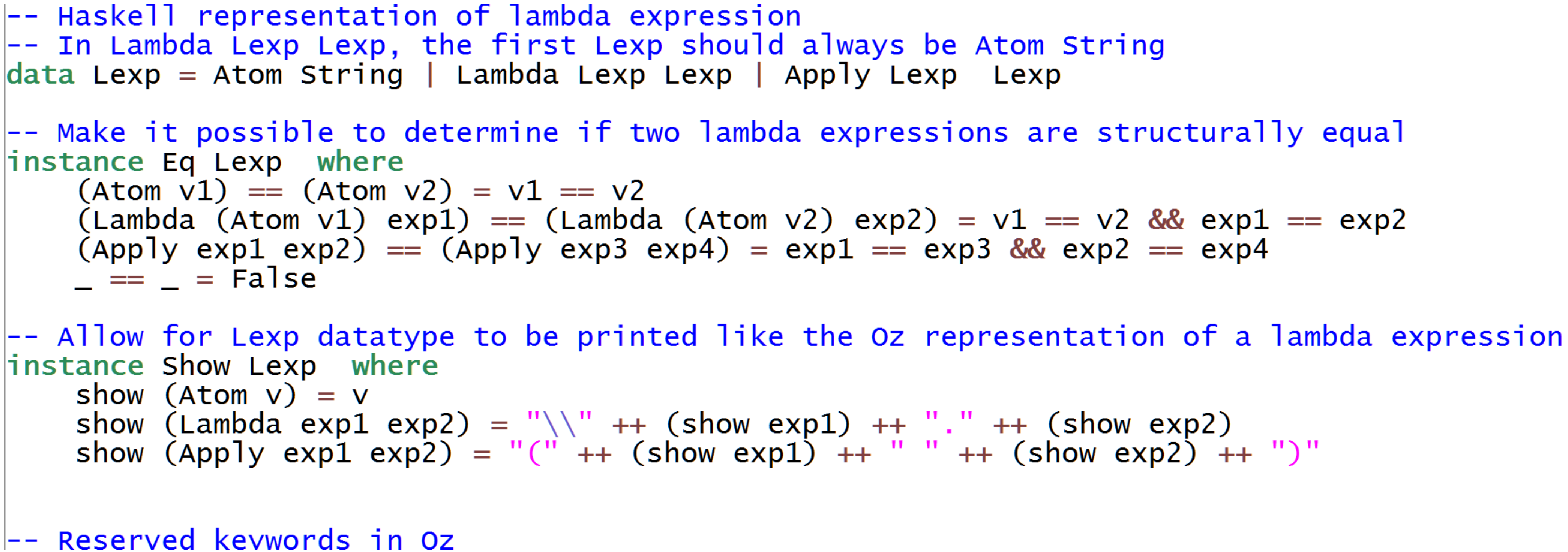
Outline

1. What you get from PA1Helper.hs module
2. Where to put your code
3. How to run code

What You get from Module (PA1Helper.hs): Part 1

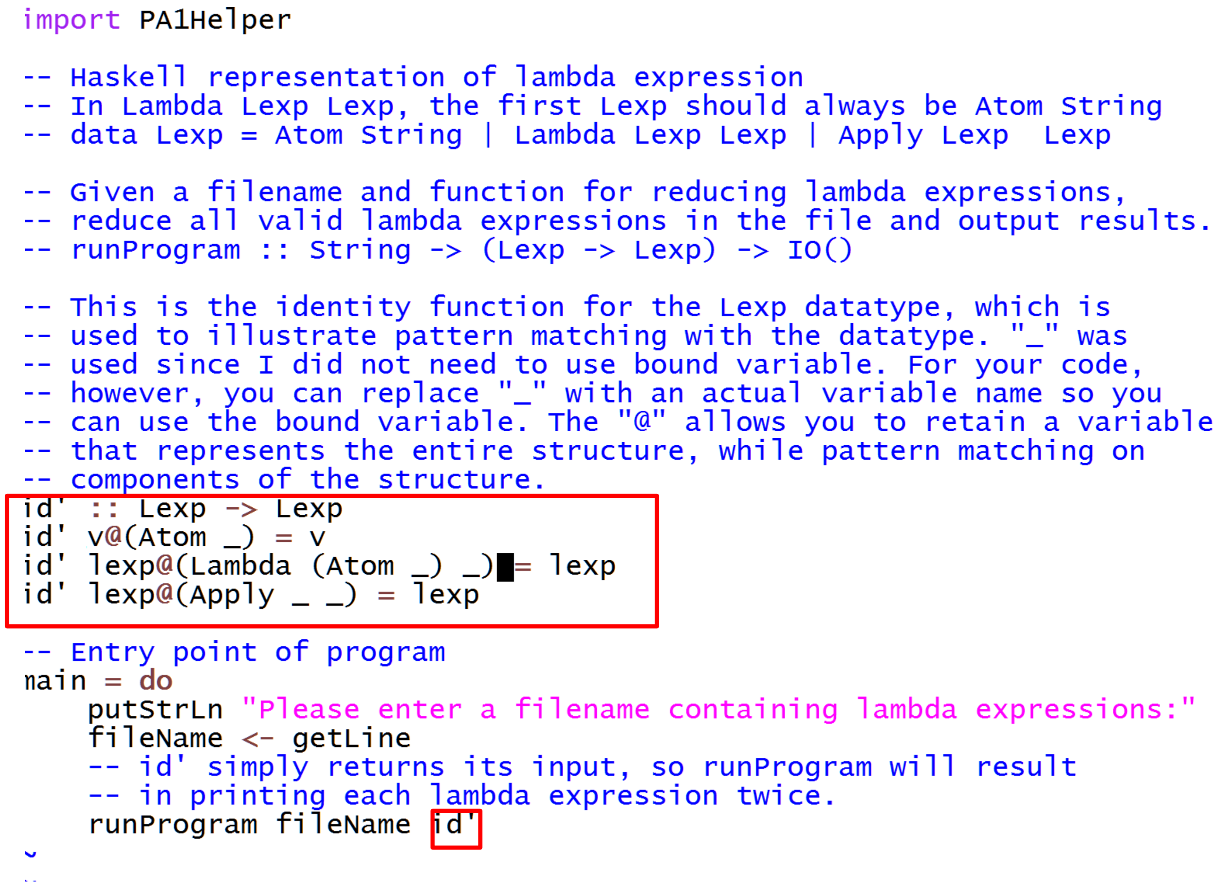
- runProgram :: String -> (Lexp->Lexp)->IO()
- data Lexp = Atom String | Lambda Lexp Lexp | Apply Lexp Lexp (line 12)
  - Instance of Eq typeclass (lines 15-19)
  - Instance of Show typeclass (lines 22-25)

What you get from Module (PA1Helper.hs): Part 2



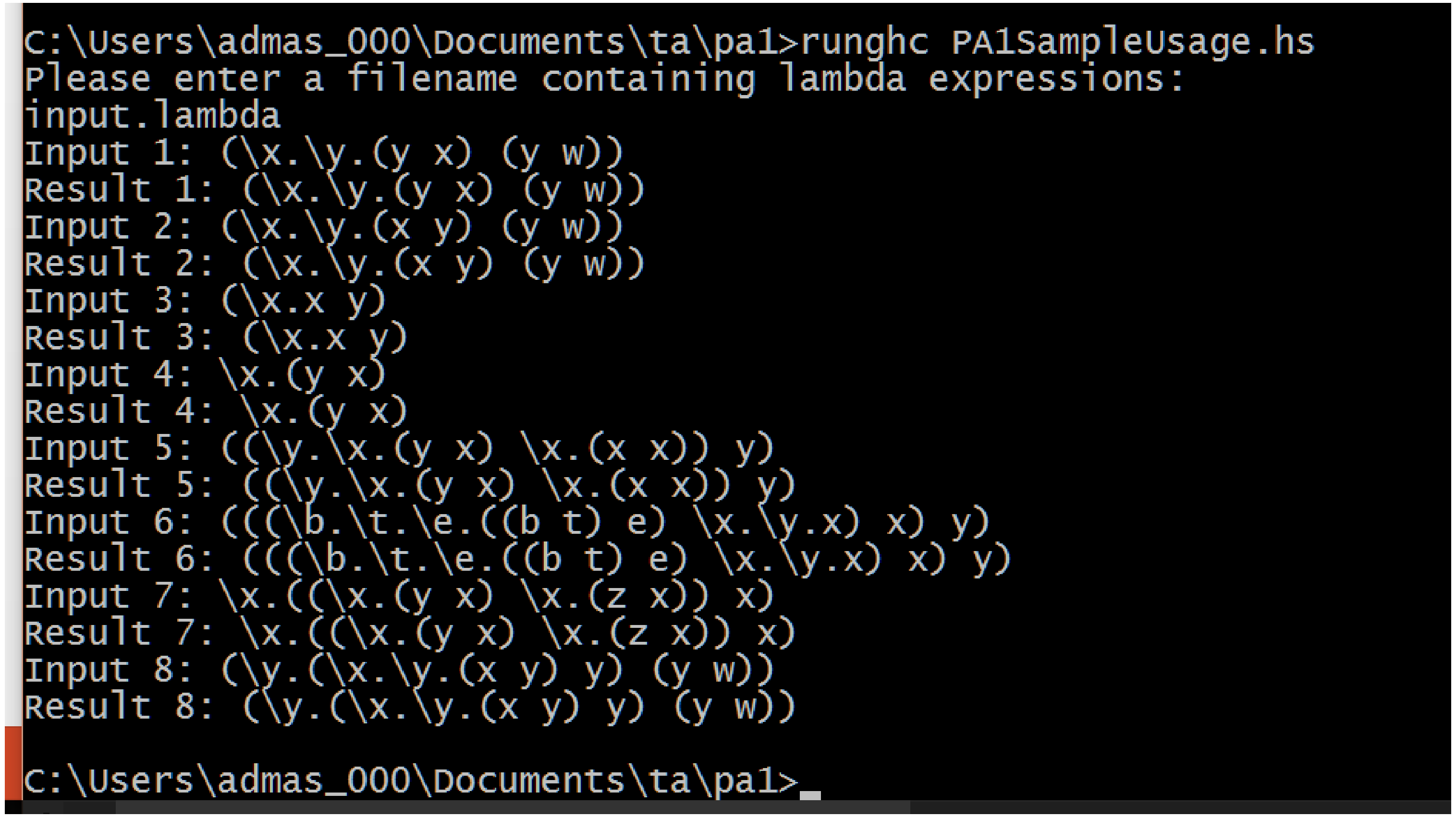
```haskell
-- Haskell representation of lambda expression
-- In Lambda Lexp Lexp, the first Lexp should always be Atom String
data Lexp = Atom String | Lambda Lexp Lexp | Apply Lexp  Lexp

-- Make it possible to determine if two lambda expressions are structurally equal
instance Eq Lexp  where
    (Atom v1) == (Atom v2) = v1 == v2
    (Lambda (Atom v1) exp1) == (Lambda (Atom v2) exp2) = v1 == v2 && exp1 == exp2
    (Apply exp1 exp2) == (Apply exp3 exp4) = exp1 == exp3 && exp2 == exp4
    _ == _ = False

-- Allow for Lexp datatype to be printed like the Oz representation of a lambda expression
instance Show Lexp  where
    show (Atom v) = v
    show (Lambda exp1 exp2) = "\\" ++ (show exp1) ++ "." ++ (show exp2)
    show (Apply exp1 exp2) = "(" ++ (show exp1) ++ " " ++ (show exp2) ++ ")"


-- Reserved kevwords in Oz
```

Where to Put Code



```
import PA1Helper

-- Haskell representation of lambda expression
-- In Lambda Lexp Lexp, the first Lexp should always be Atom String
-- data Lexp = Atom String | Lambda Lexp Lexp | Apply Lexp  Lexp

-- Given a filename and function for reducing lambda expressions,
-- reduce all valid lambda expressions in the file and output results.
-- runProgram :: String -> (Lexp -> Lexp) -> IO()

-- This is the identity function for the Lexp datatype, which is
-- used to illustrate pattern matching with the datatype. "_" was
-- used since I did not need to use bound variable. For your code,
-- however, you can replace "_" with an actual variable name so you
-- can use the bound variable. The "@" allows you to retain a variable
-- that represents the entire structure, while pattern matching on
-- components of the structure.
id' :: Lexp -> Lexp
id' v@(Atom _) = v
id' lexp@(Lambda (Atom _) _) = lexp
id' lexp@(Apply _ _) = lexp

-- Entry point of program
main = do
    putStrLn "Please enter a filename containing lambda expressions:"
    fileName <- getLine
    -- id' simply returns its input, so runProgram will result
    -- in printing each lambda expression twice.
    runProgram fileName id'
~
```

How to Run Code



```
C:\Users\admas_000\Documents\ta\pa1>runghc PA1SampleUsage.hs
Please enter a filename containing lambda expressions:
input.lambda
Input 1: (\x.\y.(y x) (y w))
Result 1: (\x.\y.(y x) (y w))
Input 2: (\x.\y.(x y) (y w))
Result 2: (\x.\y.(x y) (y w))
Input 3: (\x.x y)
Result 3: (\x.x y)
Input 4: \x.(y x)
Result 4: \x.(y x)
Input 5: ((\y.\x.(y x) \x.(x x)) y)
Result 5: ((\y.\x.(y x) \x.(x x)) y)
Input 6: (((\b.\t.\e.((b t) e) \x.\y.x) x) y)
Result 6: (((\b.\t.\e.((b t) e) \x.\y.x) x) y)
Input 7: \x.((\x.(y x) \x.(z x)) x)
Result 7: \x.((\x.(y x) \x.(z x)) x)
Input 8: (\y.(\x.\y.(x y) y) (y w))
Result 8: (\y.(\x.\y.(x y) y) (y w))

C:\Users\admas_000\Documents\ta\pa1>_
```

Note: pa1.hs and PA1Helper.hs should be in same folder

Questions?