Programming Paradigms Fall 2023 — Problem Sets

by Nikolai Kudasov and Khaled Ismaeel

October 19, 2023

1 Problem set №8

1. What is the type of `guess` in the following program? Justify your answer.

```
guess p g = do
  s <- getLine
  x <- g s
  case p x of
    True -> return x
    _    -> guess p g
```

2. Implement a program `shoutBack :: IO ()` that goes through an infinite loop of reading user input and printing it back in CAPS. Use `toUpper :: Char -> Char` to convert a single character to upper case. Remember that `type String = [Char]`.

```
import Data.Char (toUpper)

shoutBack :: IO ()
```

3. Implement the following functions over `IO`:

   (a) `foreverIO :: IO a -> IO b` — run a given program forever in an infinite loop:

   ```
   >>> foreverIO (putStrLn "Hello!")
   Hello!
   Hello!
   Hello!
   ...
   ```

   (b) `whenIO :: Bool -> IO () -> IO ()` — run a given program if a condition is satisfied;

   (c) `maybeIO :: Maybe (IO a) -> IO (Maybe a)` — run a given program if there is one;

   (d) `sequenceMaybeIO :: [IO (Maybe a)] -> IO [a]` — run a sequence of programs and collect all results of type `a`;

   (e) `whileJustIO :: (a -> IO (Maybe a)) -> a -> IO ()` — starting with an initial value of type `a`, apply a given function to run a program and either get `Nothing` and stop or get the next value and repeat;

(f) `forStateIO_ :: s -> [a] -> (s -> a -> IO s) -> IO s` — starting with an initial state of type `s`, go over values in the list of type `[a]` from left to right, applying a step function `s -> a -> IO s` to update intermediate state on every element; return the final state of type `s`:

```
verboseSnoc :: [Int] -> Int -> IO [Int]
verboseSnoc xs x = do
  putStrLn ("appending " ++ show x ++ " to the end of " ++ show xs)
  return (x:xs)
```

```
>>> forStateIO_ [] [1, 2, 3] verboseSnoc
```

```
appending 1 to the end of []
appending 2 to the end of [1]
appending 3 to the end of [1,2]
[1,2,3]
```

4. Implement a **polymorphic** higher-order function `iforIO_` that runs a program for each element and its index in a given list (using given function). Provide an explicit type signature for `iforIO_`.

```
example = do
  iforIO_ [1, 2] (\i n ->
    iforIO_ "ab" (\j c ->
      print ((i, j), replicate n c)))
```

```
>>> example
```

```
((0,0),"a")
((0,1),"b")
((1,0),"aa")
((1,1),"bb")
```