IO and Interaction

1 Batch programs

- So far, we've only written batch programs
- That is, programs that take all their inputs at the start and provide output at the end
- To change what we compute, need to change source code a rerun

2 Interactive programs

- What if we want to use haskell to write interactive programs?
- These read from the keyboard and write to the screen as they are running

3 A problem

- Haskell problems are pure mathematical functions

⇒ Haskell programs therefore have no side effects

Definition: Side effect

Modifying some (internal/hidden) state as well as returning a value

- Reading from the keyboard and writing to the screen are side effects

⇒ Interactive programs have side effects

4 Conceptual idea

- We can think of an interactive program as a pure function of type `World -> World`
- That is, it takes the current state of the world as input and produces a modified world as output
- New World object reflects any side effects that were performed

IO actions

```
type IO a  = World -> (a,World)
```

Input/output eats the world and produces a result of type a, along with a new world

5 Actions

- Copying the world is too expensive in practice
- Introduce new types to distinguish pure expressions from impure actions
- Use the concept, but Haskell uses a primitive type: implementation details are hidden
- These actions may have side effects
- Now we can write interactive programs in Haskell and ”hide” the side effects behind type

5.1 Basic actions

5.1.1 Reading

```
getChar :: IO Char
getChar = ...
```

Read a character from the keyboard, echo it to the screen and return it

5.1.2 Writing

```
putChar :: Char -> IO ()
putChar c = ...
```

Write a character to the screen and return noting (indicated by the empty tuple)

5.2 Bridging from expressions into actions

- For type safety, we need a way of ”wrapping” values into actions
- Allows us to bring side-effect-free expressions into the ”action” world

```
return :: a -> IO a
return v = ...
```

”Lift” a pure expression into an impure action

Note: no way of turning an action back again

Important: Return

The name return is rather misleading when coming from imperative languages. Calling return does not affect control flow

5.3 Sequencing actions

We can combine a sequence of IO actions using do notation

```
do v1 <- a1
   v2 <- a2
   ...
   vn <- an
   return (f v1 v2 ... vn)
```

Binds results of actions to values then applies f to the values and lifts into ”action-land” with return

Similarity with list comprehensions

- Each expression is called a generator
- If we want to execute an action, but don’t care about the result, we can use `_ <- ai` or just ai

5.4 Example: reading characters

```
act :: IO(Char, Char)
act = do x<- getChar
         getChar
         y <- getChar
         return (x,y)
```

- Read three characters, discard the second, and return the first and third
- Note the use of return, without it we would get a type error

5.5 When is an action performed

- Actions never require arguments `act :: IO` is not a function
- Just specify that something will be done
- Must run to execute
- GHCi knows to run actions at the prompt
- Conversely when writing a program to be compiled, GHC only ends up running the main action

6 Pure vs impure

Pure

- Always produces the same result when applied to the same arguments
- Never has side effects
- Never alters state

Impure

- May produce different results when applied to the same arguments
- May have side effects
- May alter state

Definition: Referential transparency

Replacing an expression by its value does not change the behaviour of the program

7 Actions as promises

- To fix the issue of referential transparency, `IO` is introduced
- We can think then of a type `IO Char` as a placeholder for a char that will only materialise once the program executes
- Moreover, it encapsulates a promise that this `Char` will actually appear
- Manipulating an `IO Char` is equivalent to setting up ”plans” to be executed when the `Char` materialises
- This way, we maintain type safety ”inside” the action