Using the Documentation and Explanation of the Implementation of the game of Nim in Haskell as an excuse to learn how to use LaTeX

Craig Knott
George Hallam

May 20, 2012

Abstract

This is the result of implementing the game of Nim in Haskell using the impure IO features. This piece of work was derived from a question posed by Graham Hutton, in his G51FUN Course. The exact question can be seen here `http://www.cs.nott.ac.uk/~gmh/functional.pdf`, on slide 119.

0.1 Defining Data Types and Printing

To begin with, we needed to define a way to print the board. But to do this, we needed a data type to store the number of stars on each row. Thus, we defined;

```
type Board = [Int]
```

Which would hold the number of stars on each row. We used a function *nimDraw* which would take in a *Board* and would generate a list of strings of stars, which would be used to display the board using the data type

```
type Game = [[Char]]
```

And would be defined as follows;

```
nimDraw :: Board → Game
nimDraw []       = []
nimDraw (n : ns) = [x | x ← ['*'], y ← [1 .. n]] : nimDraw ns
```

To then print this in the format that was specified in the question, we wrote a second function *nimPrint*.

```
nimPrint :: Game → IO ()
nimPrint []       = return ()
nimPrint (x : xs) = do
  putStr (show ((length (initialBoard) + 1) − (length (x : xs))))
  putStr ". "
  putStrLn x
  nimPrint xs
```

For ease of access, we defined a constant

```
initialBoard :: Board
intialBoard = [5, 4, 3, 2, 1]
```

Which meant we could reproduce the intial output by simply calling *nimPrint* (*nimDraw initialBoard*)

0.2 Programming Game Rules and Functionality

Next we needed a way for the player to actually interact with the game. This required the prompting of the player to enter in values, checking the legitimacy of these values, and the application of these values to the current state of the game.

The function *valid* takes a *Board* and two *Ints* and returns a *Bool*. This function is used to determine whether a move that the player is attempting is a valid move using the rules of the game. It states that if the player we to remove x stars from a pile, if the contents of this pile would be 0 or less, an invalid move has been attempted.

```
valid :: Board → Int → Int → Bool
valid bs x y | ((bs !! (y − 1) − x) < 0) = False
             | otherwise = True
```

The next function, *remove*, removes x stars from the pile y. It takes a *Board* and two *Ints* and returns the modified *Board*. It is defined as

```
remove :: Board → (Int → (Int → Board))
remove ns x y = (take (y − 1) (ns))
                ++
                [ns !! (y − 1) − x]
                ++
                (drop (y) (ns))
```

0.3 The main game method

The largest section of code was the *turn* function, which dealt with each players turn, and recursively call itself until a win condition was met. It firstly checked for a win condition, failing to find one would result in the prompting of both a number of stars to remove, and a row to remove them from (both with appropriate error trapping included). Using these values, *valid* and *remove* would called, followed by the calling of *nimPrint* and *nimDraw*, using the new value of *Board*, before recursively called *turn* using this same *Board*.

This function *turn* is defined on page 5.

0.4 Playing the game

To actually begin the game loop and set up the game initially, we implemented one final function *play*. We would welcome the players to the game, print the board, and start the first run of *turn*. Defined as the following:

```
play :: IO ()
play = do
  putStrLn "\n+-----------------+"
  putStrLn "| Welcome to Nim! |"
  putStrLn "+-----------------+"
  putStrLn "  By George Hallam"
  putStrLn "  And Craig Knott\n"
  nimPrint (nimDraw initialBoard)
  turn initialBoard 2
  return ()
```

```
turn :: Board → Int → IO ()
turn b p = if sum b ≡ 0 then
    do
      if p `mod` 2 ≡ 0 then putStrLn "Player 2 Wins\n" else putStrLn "Player 1 Wins\n"
          return ()
  else
    do
      putStr "\nPlayer "
      putStrLn (show ((p `mod` 2) + 1))
      putStrLn "\nPlease type number to remove"
      x ← getChar
      if x ≡ '\n' ∨ ((digitToInt x) ⩽ 0) ∨ (isDigit x ≡ False)
        then
        do
          putStr "\nError! You've either entered a newline, a 0, or a letter\n"
          turn b p
      else
        do
          putStrLn "Please type which row to remove from"
          newline ← getChar
          y ← getChar
          newline ← getChar
          if y ≡ '\n' ∨ ((digitToInt y) ⩽ 0) ∨ (digitToInt y) > length b ∨ (isDigit x ≡ False)
            then
              do
                putStr "\nError, newline or you selected to remove from "
                putStrLn "a row out of range and can not parse a letter\n"
                turn b p
          else
            if valid b (digitToInt x) (digitToInt y) then
              do
                nimPrint (nimDraw (remove b (digitToInt x) (digitToInt y)))
                turn (remove b (digitToInt x) (digitToInt y)) (p + 1)
            else
              do
                putStrLn "\nError, you can't remove that many\n"
                nimPrint (nimDraw b)
                turn b p
                return ()
```