Functional programming
Tallinn

Michael R. Hansen
DTU Compute
November 14, 2014

The game Nim

The game of Nim is played as follows. Any number of matches are arranged in heaps, the number of heaps, and the number of matches in each heap, being arbitrary. There are two players A and B. The first player A takes any number of matches from a heap; he may take one only, or any number up to the whole of the heap, but he must touch one heap only. B then makes a move conditioned similarly, and the players continue to take alternately. The player who takes the last match wins the game.

The game has a precise mathematical theory: We define an operator xorb for non-negative integers by forming the *exclusive or* of each binary digit in the binary representation of the numbers, for example

$$\begin{array}{rcl} 109 & = & 1101101_2 \\ 70 & = & 1000110_2 \\ 109 \text{ xorb } 70 & = & 0101011_2 \; = \; 43 \end{array}$$

The xorb operator in F# is written ^^^, for example:

```
109 ^^^ 70;;
val it : int = 43
```

The operator xorb is associative and commutative, and 0 is the unit element for the operator.

Let the non-negative integers $a_1, \ldots, a_n$ be the number of matches in the n heaps, and let m denote the integer:

$$m = a_1 \text{ xorb } a_2 \text{ xorb } \cdots \text{ xorb } a_n$$

The following can then be proved:

1. If $m \neq 0$ then there exists an index k such that $a_k \text{ xorb } m < a_k$. Replacing the number a_k by $a_k \text{ xorb } m$ then gives a new set of a_i's with $m = 0$.
2. If $m = 0$ and if one of the numbers a_k is replaced by a smaller number, then the m-value for the new set of a_i's will be $\neq 0$.

This theory gives a strategy for playing Nim:

1. If $m \neq 0$ before a move, then make a move to obtain $m = 0$ after the move (cf. the above remark 1).
2. If $m = 0$ before a move, then remove one match from the biggest heap (hoping that the other player will make a mistake, cf. the above remark 2).

This strategy should be used to make a program playing Nim with the user.

You should complete the program skeleton handed out in connection with this exercise. A game state g is represented by an array of integers (the number of matches in the heaps) and a move is represented by an integer pair $mv = (i, n)$, expressing that n matches should be removed from heap $g.[i]$:

```
type Heap = int
type Game = Heap[]    // invariant: there is at least one heap

type Move = int*int   // (i,m): remove m matches from heap g.[i]
```

The program skeleton contains a primitive graphical user-interface and conversion functions between textual and internal representations of games and moves. For example:

```
gameOfString: string -> Game option
moveOfString: string -> Game -> Move option
```

For example:

```
let (Some g) = gameOfString " 2 3 4 ";;
val g : int [] = [|2; 3; 4|]

moveOfString g "0   1";;
val it : (int * int) option = Some (0, 1)

moveOfString g "3   1";;
val it : (int * int) option = None
```

In both cases input is assumed to be a single line and both functions return `None` when this input is not in the correct form.

Part 1

Study the program skeleton and derive the automaton specifying the desirable interactions involving **NewGame** and `UserMove` on the basis of the three functions `init`, `myTurn`, and `userTurn`. (Notice that the automaton may have epsilon transitions.)

Extend the program skeleton with implementations of the following functions:

- `allHeapsEmpty: Game -> bool`. This function is used to test whether a player has won, that is, all heaps are empty
- `myMove: Game -> Move`. This function returns the computer's move for a given state of the game, by use of the strategy described above.
- `performMove: Game -> Move -> Game`. This function performs a move (i, n) in a game state g.

You can now play Nim: enter a game in a text box and engage in alternating move events with the computer until a winner is found.

Part 2

So far it is just two buttons: `NewGame` and `Move`, that have an effect. Extend your automaton and program so that the user gets an option to quit an unfinished game by pressing the `Quit` button.

When the user enters an ill-formed move, the program write "illegal input" in the text box `statusBox`. This text box is not reset properly in the program and the text will remain even after the user have entered a well-formed move. Solve this problem.

Part 3

Extend your automaton and program so that the user get an option to fetch a game from a web page by pressing the `Fetch` button. The user should also have an option to cancel an ongoing download. The webpage `http://www2.compute.dtu.dk/~mire/nim.game` contains a game you can fetch.

You may also extend the program so that the computer teases the user, by writing "you will loose -:)" in a text box, as soon as the it can win the game no matter how the user plays. This provoking remark should appear in just one step of a game.