02257 Applied functional programming

Michael R. Hansen
DTU COMPUTE
January 1, 2016

Project 1:
A game with a graphical user interface

In this project you shall make a program that can play the game Nim with a user. The project can be solved in groups having two, three or four persons ***only***.

To submit the solution to this project, you must:

1. give a demo on Tursday, January 7, from 13:00-16:00, and
2. submit your project on CampusNet no later than 17:00 on Tursday, January 7.

Note that: **All group members must be available at the demo**.

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 turns as long as there are matches left. 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

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

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.

Your task is to make such a Nim-program with a graphical user interface. Particular requirements for your product (program and documentation) are:

- It should be expressed using F#'s asynchronous computations.
- The documentation (see below) should contain an finite automaton specifying the allowed interactions, and the structure of your dialogue programme must correspond to this automaton in a direct manner.
- You should aim at a modular design so that your final program is composed of well-understood small pieces. In particular, you should strive at *separation of concerns* in your programs so that, for example, the dialogue program, the graphical user interface, and the game operations are suitably separated.

You are supposed to extend the basic solution described above, for example, with facilities like:

- A facility to fetch Nim games from some web pages and such fetch operations should be asynchronous. It must be possible for the user to cancel an ongoing fetch operation.
- A facility where the program teases the user, by writing "you will loose -:)", as soon as it can win the game no matter how the user plays. This provoking remark should appear in just one step of a game.
- An appealing graphical layout of the heaps.
- A facility giving the user a better chance to win over the program.

Some sources

Techniques for construction a program with a graphical user interface are presented in the lecture starting Monday, January 4, at 9:00. Slides are available on FileSharing in CampusNet. This technique is based on Chapter 13 in

HR13: *Functional programming using F#*, M.R. Hansen and H. Rischel, Cambridge University Press, 2013. Available electronically at DTU's library.

Chapter 7 in [HR13] gives an introduction to the module system of F#. Slides on this topic are also available in FileSharing on CampusNet.

Some links for further information:

- The dialogue program shown at the lecture is available in FileSharing in CampusNet in two versions. One for the Windows users one for Mono users. The differences between these two files occur in the last line only.
- The programs in [HR13] are available from: `http://www.imm.dtu.dk/~mire/FSharpBook/`.
- You may have a look at `http://msdn.microsoft.com/en-us/library/dd233207.aspx` concerning F# interfaces.
- You may have a look at `http://msdn.microsoft.com/en-us/library/dd233237.aspx` concerning object expressions in F#.
- You may have a look at `https://msdn.microsoft.com/en-us/library/dd233219.aspx` concerning name spaces in F#.

The product

You should hand in as a *group* two files on CampusNet:

- A zip-archive containing your program files.
- A pdf-file containing, with at most 3 pages including the front page:
  1. A front page with names and study numbers of the persons in the group, together with signatures by every group member. By handing in as a group, the members account for (1) the solution is made by the group members only, (2) no part of the solution is distributed to other groups, and (3) the group members have contributed equally to the solution.
  2. The following pages should clearly describe the status of the solution. In particular, it must include a finite automaton specifying the dialogue with the user. It must also include:
     - Type declarations for the main concepts in your solution.
     - Signature for the main functions in your solution.
     - A overview of the structure of your solution.
     - A brief reflection on the project.