Assignments week 3

Aleksandar Ivanov, Ivaylo Ivanov

10.01.2020

Assignment 1:

This was the assignment that took us the longest. Our initial attempt at an implementation can be found in wk4_1_old.smt2. The idea was to represent every wolf and sheep with a different integer variable depending on their shore. The code quickly got very bulky and buggy so we changed our approach by representing the count of wolves and rabbits on each shore. You can see the final implementation in wk4_1.smt2. Most of the functionality is contained in the MovementRule functions, where you can find a more detailed description in the comments. We played out the scenario given by z3 and it works well. (follows all rules)

```
ivo@ivoi:~/Fontys/semester3/math/log/wk4$ z3 wk4_1.smt2
sat
(model
  (define-fun Shore2Wolfs ((x!0 Int)) Int
    (ite (= x!0 0) 0
    (ite (= x!0 11) 3
    (ite (= x!0 2) 1
    (ite (= x!0 3) 3
    (ite (= x!0 6) 1
    (ite (= x!0 7) 1
    (ite (= x!0 8) 0
      2)))))))))
  (define-fun Shore1Rabbits ((x!0 Int)) Int
    (ite (= x!0 11) 0
    (ite (= x!0 5) 1
    (ite (= x!0 6) 2
    (ite (= x!0 7) 0
    (ite (= x!0 8) 0
    (ite (= x!0 9) 0
    (ite (= x!0 10) 1
      3)))))))))
  (define-fun Shore1Wolves ((x!0 Int)) Int
    (ite (= x!0 0) 3
    (ite (= x!0 11) 0
    (ite (= x!0 2) 2
    (ite (= x!0 3) 0
    (ite (= x!0 6) 2
    (ite (= x!0 7) 2
    (ite (= x!0 8) 3
      1)))))))))
  (define-fun Shore2Rabbits ((x!0 Int)) Int
    (ite (= x!0 11) 3
    (ite (= x!0 5) 2
    (ite (= x!0 6) 1
    (ite (= x!0 7) 3
    (ite (= x!0 8) 3
    (ite (= x!0 9) 3
    (ite (= x!0 10) 2
      0)))))))))
```

Assignment 2:

This assignment was the most straightforward. We simply wrote an ite for every iteration of the normal program and let z3 do its thing. The implementation can be found in wk4_2.smt2. Below you can find a screenshot of the output: (solution is b=4 and a=271)

```
ivo@ivoi:~/Fontys/semester3/math/log/wk4$ z3 wk4_2.smt2
sat
(model
  (define-fun B ((x!0 Int)) Int
    (ite (= x!0 1) 8
    (ite (= x!0 2) 16
    (ite (= x!0 3) 32
    (ite (= x!0 4) 64
    (ite (= x!0 5) 128
    (ite (= x!0 6) 256
    (ite (= x!0 7) 251
    (ite (= x!0 8) 502
    (ite (= x!0 10) 999
    (ite (= x!0 9) 1004
      4))))))))))
  (define-fun A ((x!0 Int)) Int
    (ite (= x!0 1) 268
    (ite (= x!0 2) 265
    (ite (= x!0 3) 262
    (ite (= x!0 4) 259
    (ite (= x!0 5) 256
    (ite (= x!0 6) 253
    (ite (= x!0 7) 506
    (ite (= x!0 8) 503
    (ite (= x!0 10) 1000
    (ite (= x!0 9) 500
      271))))))))))
)
```

Assignment 3:

For this assignment we made the datatypes ROW and COLUMN and created 5 instances of each. We have helper functions that give us the next/previous row/column round-robin. The function square is the most important one. It returns a boolean, which in turns means that a rook should be placed in the square if true and not if false. More detailed explanation can be found in the code comments, in the file wk4_3.smt2. Below you can see the program output and the solution on a 5x5 chessboard:

```
ivo@ivoi:~/Fontys/semester3/math/log/wk4$ z3 wk4_3.smt2
sat
(model
  (define-fun square ((x!0 ROW) (x!1 COLUMN)) Bool
    (or (and (= x!0 Row3) (= x!1 Column1))
        (and (= x!0 Row2) (= x!1 Column5))
        (and (= x!0 Row1) (= x!1 Column3))
        (and (= x!0 Row5) (= x!1 Column2))
        (and (= x!0 Row4) (= x!1 Column4))))
  (define-fun PreviousColumn ((x!0 COLUMN)) COLUMN
    (ite (= x!0 Column5) Column4
    (ite (= x!0 Column4) Column3
    (ite (= x!0 Column3) Column2
      Column1))))
  (define-fun NextColumn ((x!0 COLUMN)) COLUMN
    (ite (= x!0 Column1) Column2
    (ite (= x!0 Column2) Column3
    (ite (= x!0 Column3) Column4
      Column5))))
  (define-fun PreviousRow ((x!0 ROW)) ROW
    (ite (= x!0 Row5) Row4
    (ite (= x!0 Row4) Row3
    (ite (= x!0 Row3) Row2
      Row1))))
  (define-fun NextRow ((x!0 ROW)) ROW
    (ite (= x!0 Row1) Row2
    (ite (= x!0 Row2) Row3
    (ite (= x!0 Row3) Row4
      Row5))))
)
```