Mandatory Homework Assignment 2 – IN5100

Peter C. Ölveczky

September 20, 2023

1 Formalities

- Work alone or in groups of two.
- Deadline: Friday October 13, 2023 at 23.59 GMT +2.
- Submit using the appropriate channel (Devilry?).

2 The Assignment

You are supposed to use Maude meta-level functionality to implement a new Maude "command" for performing *randomized simulations* in Maude.

It might be useful to have a data type for either sets or lists of (meta-)terms. I thought it was semantically cleaner to use lists of terms. If you agree, you can either define a new data type `ListOfTerms` for this purpose, or use sorts like `META-LEVEL`'s `TermList`. For some reason, I chose the former approach (also to get better formatting of the output). If you choose a different approach, just replace the sort `ListOfTerms` below with your desired sort.

2.1 Exercise 1: One Randomized Simulation

We first implement a function that performs *one* random simulation of the system from a given state. One parameter is the number of steps to simulate, which can also be `unbounded`. Another parameter is the "seed" for the function `random` (or whichever random function you will use). This is not really the seed (although that can also be changed; see the Maude manual), but just denote *which* random number you start with.

That is, you should define a function of the form

```
  op simulate_stepsFrom_in_seed_ : Bound Term Module Nat -> Term .
```

where the parameters are, respectively, the number of steps to simulate, the state from which to start the simulation, the module in which the simulation takes place, and the index/parameter of the function `random`.

As always, test your function on a bunch of examples; some given later.

2.2 Exercise 2: Multiple Randomized Simulations

Now, we want to run not just one, but multiple, randomized simulations from some initial state. The number of such single randomized simulations is the first parameter in the following function/command/analysis method that you should implement:

```
op repeat_simulate_stepsFrom_in_seed_ : Nat Bound Term Module Nat -> ListOfTerms .
```

The other parameters are as above. Remember to start with a different "index" to `random` in each simulation.

2.3 Test Examples

You should test your simulation functions/commands on a bunch of examples. Some simple ones are given in the following file, which will be available from the course web page (`http://olveczky.se/IN5100-23/football.maude`). So will also a core Maude version of the dining philosophers, where we hope that each philosopher can eat often.

```
mod GAME is protecting NAT . protecting STRING .
  sort Game .

  op _-_ _:_ : String String Nat Nat -> Game [ctor] .

  vars HOME AWAY : String .  vars M N K : Nat .

  rl [home-goal] :
     HOME - AWAY M : N  =>  HOME - AWAY (M + 1) : N .

  rl [away-goal] :
     HOME - AWAY M : N  =>  HOME - AWAY M : (N + 1) .
endm

*** Perform one, and then many, simulations, where each simulation
*** simulates, say 10, steps from some state "Vasco" - "Botafogo" 0 : 0 .



mod END-GAME is including GAME .
    op _-_finalScore_:_ : String String Nat Nat -> Game [ctor] .

    vars HOME AWAY : String .
    vars M N : Nat .
    rl [endGame] :
       HOME - AWAY M : N   =>  HOME - AWAY finalScore M : N .
endm

*** Perform many simulations--without bound on the length
*** of each simulation--from some initial state like
*** "Fluminense" - "Palmeiras" 0 : 0 .




*** Famous "whiteboard" game from the course:
***  Any two numbers on a whiteboard can be rewritten to their
*** artithmetic mean; what is the remaining number?

mod WHITEBOARD is protecting NAT .
  sort WhiteBoard .
  subsort NzNat < WhiteBoard .
  op __ : WhiteBoard WhiteBoard -> WhiteBoard [assoc comm ctor] .

  vars M N : NzNat .

  rl [replace] : M N => (M + N) quo 2 .

  op init : -> WhiteBoard .
  eq init = 1 34 7 99 65 2 13 .
endm

*** Do many, unbounded obviously, simulations from some
*** initial state like 1 34 7 99 65 2 13 .


*** How we need to check, using some kind of binary trees, that each rule
*** gets to execute, and also in deep subterms.

mod TEST-SIM is protecting NAT .

   sort Name .  ops a b c d f g h : -> Name [ctor] .

   sort Node .
   op node : Name Nat Nat Nat -> Node [ctor] .

   sort Tree .
   op _^_^_ : Tree Node Tree -> Tree [ctor] .
   op e : -> Tree [ctor] .   --- empty tree

   var NAME : Name .
   vars N1 N2 N3 : Nat .

   rl [r1] : node(NAME, N1, N2, N3) => node(NAME, N1 + 1, N2, N3) .
   rl [r2] : node(NAME, N1, N2, N3) => node(NAME, N1, N2 + 1, N3) .
   rl [r3] : node(NAME, N1, N2, N3) => node(NAME, N1, N2, N3 + 1) .

   op init : -> Tree .
   eq init =
          ((e ^ node(a,0,0,0) ^ e)
           ^ node(b,0,0,0) ^ (( e ^ node(c,0,0,0) ^ e) ^ node(d,0,0,0) ^ e))
                 ^
           node(f,0,0,0)
         ^
           ( (e ^ node(g,0,0,0) ^ e) ^ node(h,0,0,0) ^ e) .
endm

*** simulate randomly (quite many steps) from init,
*** and make sure that each node, a to g, gets values in all three
*** arguments
```