CS 476 Homework #3 Due 10:45am on 10/6

Note: Answers to the exercises listed below should be handed to the instructor *in hardcopy* and in *typewritten form* (latex formatting preferred) by the deadline mentioned above. Your hardcopy should include a screenshot of your interactions with the different tools for problems 3 and 4. Code solutions should be emailed by the same deadline to `skeirik2@illinois.edu`.

1. Solve **Ex.** 108 in *STACS*.

2. Solve Exercise 89 (pages 117–118) in Peter Ölveczky's Lecture Notes. Include your solution and evaluation of some test cases in the hard copy, but email also the code and test cases to `skeirik2@illinois.edu`.

3. Consider the following module (available in the course web page) defining the Ackermann function, which, is total recursive and therefore terminating, but not primitive recursive (because it grows faster than any primitive recursive function). It can however be shown to be primitive recursive *if higher-order functions are allowed* (see suggested reading below). Although all these facts are not needed to solve the problem below, you may be curious about this famous function in the *Theory of Computation* and you can in that case consult Section 6.2 of *STACS*, where all these ideas about primitive recursion and the Ackerman function are spelled out.

```
(fmod ACKERMAN is
  sort Natu .
  op o : -> Natu [ctor] .
  op s : Natu -> Natu [ctor] .
  op ack : Natu Natu -> Natu .
  vars N M : Natu .
  eq ack(o,M) = s(M) .
  eq ack(s(N),o) = ack(N,s(o)) .
  eq ack(s(N),s(M)) = ack(N,ack(s(N),M)) .
endfm)
```

   check that: (i) `ACKERMAN` is confluent using the Maude Church-Rosser Checker; (ii) `ACKERMAN` is terminating; prove this (a) by hand using some of the termination methods used in class or explained in Ölveczky's lecture notes, or (b) by using a semi-automated or automated tool such as (b.1) the semi-automated `AACRPO` tool, presented in class and available from the course web page, where you would define an order on the above symbols. For your convenience, in the AACRPO tool the `tests` directory contains the two examples used in class (files for both their code and their corresponding termination proofs); you may wish to add to that directory your own example(s) and enter them in Maude to check termination if you choose to use AACRPO as your termination tool; or (b.2) the MTT tool which is part of the Maude Formal Environment (MFE) also available in the course web page (if you use MTT, indicate which transformation setting and what backend tool choice you used to prove `ACKERMAN` terminating); and (iii) `ACKERMAN` is sufficiently complete using the Maude SCC Tool (also part of MFE). **Note:** The outer parentheses around `ACKERMAN` are only needed when using the MFE. Email you code for the checks needing it to `skeirik2@illinois.edu`.

4. Consider the following module (available in the course web page) of lists with a list append functions that is associative and has an identity, but where associativity and identity are explicitly defined by equations:

```
(fmod LIST-EXAMPLE is
 sorts Element List .
 subsorts Element < List .
 op a : -> Element [ctor] .
 op b : -> Element [ctor] .
 op c : -> Element [ctor] .
 op nil : -> List [ctor] .
 op _;_  : List List -> List .
 op _;_  : Element List -> List [ctor] .
 vars L P Q : List .
 eq (L ; P) ; Q = L ; (P ; Q) .
 eq L ; nil = L .
 eq nil ; L = L .
endfm)
```

check that: (i) LIST-EXAMPLE is confluent using the Maude Church-Rosser Checker; (ii) LIST-EXAMPLE is terminating either (a) by hand using some of the termination methods used in class or explained in Ölveczky's lecture notes, or (b) by using a semi-automated or automated tool such as (b.1) the semi-automated AACRPO tool, presented in class and available from the course web page, where you would define an order on the above symbols (**Note**: all instances of the same subsort overloaded symbol must have the value when defining the total order). For your convenience, in the AACRPO tool the tests directory contains the two examples used in class (files for both their code and their corresponding termination proofs); you may wish to add to that directory your own example(s) and enter them in Maude to check termination if you choose to use AACRPO as your termination tool; or (b.2) by using the MTT tool (if you use MTT, indicate which transformation setting and what backend tool choice you used to prove LIST-EXAMPLE terminating); and (iii) LIST-EXAMPLE is sufficiently complete using the Maude SCC Tool. Are the constructors free? (justify your answer). **Note:** The outer parentheses around ACKERMAN are only needed when using the MFE. Email you code for the checks needing it to skeirik2@illinois.edu.

A moral of this example is that subsorts and subsort overloading are very powerful, since they allow us in this example to tighten the append constructor exactly where we want it as a "cons-like" operator. Another moral is that the essential distinction between "cons" and "append" as two *different* functions evaporates in an order-sorted setting.