CS 477 Homework #4 Due 10:45am on 3/11

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 4 and 5.

1. Consider the following module (available in the course web page) defining the Ackermann function:

```
(fmod ACK is
  sort Nat .
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  op ack : Nat Nat -> Nat .
  vars N M : Nat .
  eq ack(0,M) = s(M) .
  eq ack(s(N),0) = ack(N,s(0)) .
  eq ack(s(N),s(M)) = ack(N,ack(s(N),M)) .
endfm)
```

   check that: (i) `ACK` is confluent using the Maude Church-Rosser Checker; (ii) `ACK` is terminating either by using some of the termination methods used in class or explained in Ölveczky's lecture notes, or by using the MTT tool (f you use MTT, indicate which transformation setting and what backend tool choice you used to prove `ACK` terminating); and (iii) `ACK` is sufficiently complete using the Maude SCC Tool.

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

```
(fmod LIST is
 sort List .
 ops a b c nil : -> List [ctor] .
 op append  : List List -> List [ctor] .
 vars L P Q : List .
 eq append(append(L, P), Q) = append(L, append(P, Q)) .
 eq append(L, nil) = L .
 eq append(nil, L) = L .
endfm)
```

   check that: (i) `LIST` is confluent using the Maude Church-Rosser Checker; and (ii) `LIST` is terminating either by using some of the termination methods used in class or explained in Ölveczky's lecture notes, or by using the MTT tool. If you use MTT, indicate which transformation setting and what backend tool choice you used to prove `LIST` terminating.

   Do exactly the same checks (i)–(ii) above for the module (also available in the course web page):

```
(fmod LIST1 is
  sort List .
  ops a b c nil : -> List [ctor] .
  op append : List List -> List [ctor] .
  vars L P Q : List .
  eq append(L, append(P, Q))  = append(append(L, P), Q) .
  eq append(L, nil) = L .
  eq append(nil, L) = L .
endfm)
```

3. Consider again the module in the previous exercise

```
(fmod LIST is
 sort List .
 ops a b c nil : -> List [ctor] .
 op append  : List List -> List [ctor] .
 vars L P Q : List .
 eq append(append(L, P), Q) = append(L, append(P, Q)) .
 eq append(L, nil) = L .
 eq append(nil, L) = L .
endfm)
```

which you were asked to check it was locally confluent using the Church-Rosser Checker. Can you check the local confluence of this module yourself *without* using any tool? That is: you are asked, without using the Church-Rosser Checker (altough you are allowed to use the `unify` command in Maude if you wish, which is explained in Chapter 12 of the Maude manual) to:

- compute all the critical pairs of this module, give a complete list of them, and explain what computations you have performed to arrive at that list; and
- show that each critical pair can be joined; you can use Maude for this purpose, for example using the `_==_` operator to compute that for each critical pair (t, t') both terms hace the same canonical form.

4. Consider the module

```
fmod LIST-of-NAT is protecting NAT .
  sort List .
  subsort Nat < List .
  op nil : -> List [ctor] .
  op _;_ : List List -> List [ctor assoc] .
  op length : List -> Nat .
  var N : Nat .
  var L : List .
  eq nil ; L = L .
  eq L ; nil = L .
  eq length(nil) = 0 .
  eq length(N) = 1 .
  eq length(N ; L) = 1 + length(L) .
endfm
```

use the ITP to prove the following theorem:

```
(goal listlength : LIST-of-NAT
  |- A{L:List ; L':List}
     ((length(L ; L')) = (length(L) + length(L'))) .)
```

Note: You can retrieve both the module and the goal from the course web page.