CS 476 Homework #5 Due 10:45am on 10/3

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 1 and 2. Code solutions should be emailed by the same deadline to `hildenb2@illinois.edu`.

1. 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] .
 eq (L:List ; P:List) ; Q:List = L:List ; (P:List ; Q:List) .
 eq L:List ; nil = L:List .
 eq nil ; L:List = L:List .
endfm)
```

   This module is enclosed in parentheses only because you will need to enter it this way to the Church-Rosser Checker and the SCC Tool. There is no need for such parentheses otherwise.

   You are asked to check that:

   - `LIST-EXAMPLE` is terminating by using the `AvCRPO` tool (note that since `LIST-EXAMPLE` makes no use of any axioms this is included as a special case of the `>AC` term order.) For your convenience, the `AvCRPO` tool directory contains the two examples used in class (files for both their code and their corresponding sig- and check- files), as well as templates for the sig- and check- files. You can copy such templates to write your own sig- and check- files for `LIST-EXAMPLE` and then invoke the tool with your chosen order on the symbols of `LIST-EXAMPLE`.
   - `LIST-EXAMPLE` is confluent using the Maude Church-Rosser Checker, which is part of the Maude Formal Environment (MFE) also available in the course web page.
   - `LIST-EXAMPLE` is sufficiently complete using the Maude SCC Tool (also part of MFE).

   Note: As already mentioned, the outer parentheses around `LIST-EXAMPLE` are only needed when using the MFE. Email you code for all the checks to `hildenb2@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. A second moral is that the essential distinction between "cons" and "append" as two *different* functions evaporates in an order-sorted setting. A third moral is that, by making "cons" a constructor and "append" a defined function, we can make the list constructor *free.* Note that only in an order-setting is it possible for the *same* overloaded operator to be a constructor for some typing and a defined symbol for another typing.

2. The above exercise was an easy exercise to help you become familiar with some of the Maude tools and learn how you can check key executability conditions for a module. In this second exercise you are asked to actually

prove yourself that the module LIST-EXAMPLE in Exercise 1 is in fact *locally confluent* (and therefore confluent since you have checked its termination) by:

- Considering all possible overlaps between the three rules in the module (including overlap of a rule with a renamed version of itself).
- Computing for such overlaps the corresponding critical pairs.
- Checking that each such critical pair can be joined by rewriting with the rules of LIST-EXAMPLE.

Of course, to compute each critical pair you must perform a unification. You can do that by hand yourself; but it may be convenient for you to know that Maude can perform any desired unification for you using the `unify` command (see Section 12.4 of the Maude manual).

Another thing that Maude can do for you is to check the joinability of a critical pair $u = v$. All you need to do is to: (i) enter the module LIST-EXAMPLE into Maude, and (ii) for each critical pair $u = v$ whose variables should be *declared on the fly*, e.g., `L:List`, `E:Element`, etc., execute the command:

```
red u == v .
```

so that the critical pair $u = v$ will be *joinable* if and only if you get the result `true`.

If you use Maude to help you in these two ways, you should include a screenshot of the answers you get from Maude for each problem you ask Maude about.