CS 476 Homework #6 Due 10:45am on 11/17

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. You should also email to Stephen Skeirik (`skeirik2@illinois.edu`) the Maude code for the exercises requiring that.

1. Solve Exercise 10.3 in Lecture 10.

2. Recall from Lecture 19 the following readers and writers example and its equational abstraction:

```
mod R&W is
  sort Nat Config .
  op <_,_> : Nat Nat -> Config [ctor] .  --- readers/writers
  op 0 : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm

mod R&W-ABS is
   including R&W-PREDS .
   eq < s(s(N:Nat)),0 > = < s(0),0 > .
endm
```

   Recall also from Lecture 17 the notion of coherence and ground coherence. Since we are mostly interested in the canonical reachability model of a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ whose states are B-equivalence classes of ground terms in canonical form, *all we need is ground coherence*. $\mathcal{R}$ may not be coherent, but may be ground coherent. In Lecture 19 a high-level explanation of why `R&W-ABS` is ground coherent was given. The goal of this problem is to help you gain a deeper understanding of coherence and ground coherence. As pointed out in pg. 14 of Lecture 17, coherence is checked by checking "critical pair like" conditions. What are they? Critical pairs are computed like for equations, but for coherence the checking of such pairs is different:

   - For each equation $u = v \in E$ and rule $p \to q \in R$ we compute critical pairs for them *as if they were both equations* $u = v$ and $t = t'$.
   - But in each such critical pair we *remember* which pair component comes from the equation and which from the rule. That is, if after B-unification we get a pair (v', q') where v' was obtained by one rewrite with $u = v \in E$ and q' by one rewrite with $p \to q$ we put q' in the *second* component of the pair.
   - We now check coherence for the pair (v', q') also in a way similar to checking local confluence but different:
     (a) as for local confluence we compute the E/B-canonical forms of v', and q', let us call them w and r, respectively;
     (b) but unlike local confluence what we check is not $w =_B r$. Instead what we do is to try to find a *one-step* rewrite $w \to^1_{R/B} w'$ with *some* rule in R such that $can_{E/B}(w') =_B r$.

If this fails to be possible for some pair $u = v \in E$, $p \to q \in R$, Maude's Coherence Checker returns the proof obligation $w \to r$. Maybe we had several one-step R/B-rewrites $w \to^1_{R/B} w'_1, \ldots, w \to^1_{R/B} w'_k$ but we never had $can_{E/B}(w'_i) =_B r$. However, if for some i, $1 \leq i \leq k$, we can prove (by hand as was done in Lecture 19, or using the ITP) that $can_{E/B}(w'_i) =_B r$ is an *inductive theorem* for the equational theory $(\Sigma, E \cup B)$ which is the equational part of $\mathcal{R} = (\Sigma, E \cup B, R)$, then $\mathcal{R}$ is ground coherent.

Now that you know what to do, you are asked to:

- compute and list all the critical pairs between the rules of `R&W-ABS` and the abstraction equation, explaining how each is formed,
- explain in detail which of these pairs are joinable in the above sense and why,
- explain how the proof obligation mentioned in Lecture 19 is generated,
- for extra credit, prove the theorem $can_{E/B}(w'_i) =_B r$ not as done by hand in Lecture 19, but using the ITP for the functional module obtained by removing the rules, keeping only the abstraction equation, and turning `mod` into `fmod` and `endm` into `endfm`.

3. Consider the following dining philosophers example, that you can retrieve from the course web page:

```
fmod NAT/4 is
   protecting NAT .
   sort Nat/4 .
   op [_] : Nat -> Nat/4 .
   op _+_ : Nat/4 Nat/4 -> Nat/4 .
   op _*_ : Nat/4 Nat/4 -> Nat/4 .
   op p : Nat/4 -> Nat/4 .
   vars N M : Nat .
   ceq [N] = [N rem 4] if N >= 4 .
   eq [N] + [M] = [N + M] .
   eq [N] * [M] = [N * M] .
   ceq p([0]) = [N] if s(N) := 4 .
   ceq p([s(N)]) = [N] if N < 4 .
endfm

mod DIN-PHIL is
   protecting NAT/4 .
   sorts Oid Cid Attribute AttributeSet Configuration Object Msg .
   sorts Phil Mode .
   subsort Nat/4 < Oid .
   subsort Attribute < AttributeSet .
   subsort Object < Configuration .
   subsort Msg < Configuration .
   subsort Phil < Cid .

   op __ : Configuration Configuration -> Configuration
                                                [ assoc comm id: none ] .
op _`,_ : AttributeSet AttributeSet -> AttributeSet
                                                [ assoc comm id: null ] .
   op null : -> AttributeSet .
   op none : -> Configuration .
   op mode`:_ : Mode -> Attribute [ gather ( & ) ] .
   op holds`:_ : Configuration -> Attribute [ gather ( & ) ] .
   op <_:_|_> : Oid Cid AttributeSet -> Object .
   op Phil : -> Phil .

   ops t h e : -> Mode .
```

```
  op chop : Nat/4 Nat/4 -> Msg [comm] .
  op init : -> Configuration .
  op make-init : Nat/4 -> Configuration .

  vars N M K : Nat .
  var C : Configuration .

  ceq init = make-init([N]) if s(N) := 4 .
  ceq make-init([s(N)])
    = < [s(N)] : Phil | mode : t , holds : none >  make-init([N])  (chop([s(N)],[N]))
    if N < 4 .
  ceq make-init([0]) =
     < [0] : Phil | mode : t , holds : none > chop([0],[N]) if  s(N) := 4 .

  rl [t2h] : < [N] : Phil | mode : t , holds : none > =>
     < [N] : Phil | mode : h , holds : none > .
  crl [pickl] :  < [N] : Phil | mode : h , holds : none > chop([N],[M])
      => < [N] : Phil | mode : h , holds :  chop([N],[M]) > if [M] = [s(N)] .
  rl [pickr] :  < [N] : Phil | mode : h , holds : chop([N],[M]) >
     chop([N],[K]) =>
     < [N] : Phil | mode : h , holds :  chop([N],[M]) chop([N],[K]) > .
  rl [h2e] :  < [N] : Phil | mode : h , holds :  chop([N],[M])
     chop([N],[K]) > => < [N] : Phil | mode : e ,
     holds :  chop([N],[M]) chop([N],[K]) > .
  rl [e2t] :  < [N] : Phil | mode : e , holds :  chop([N],[M])
     chop([N],[K]) >  =>   chop([N],[M]) chop([N],[K])
     < [N] : Phil | mode : t , holds :  none > .
endm
```

There are four philosophers, that you can imagine eating in a circular table. Initially they are all in thinking mode (`t`), but they can go into hungry mode (`h`), and after picking the left and right chopsticks (they eat Chinese food) into eating mode (`e`), and then can return to thinking.

The identities of the philosophers are naturals modulo 4, with contiguous philosophers arranged in increasing order from left to right (but wrapping around to 0 at 4). The chopsticks are numbered, with each chopstick indicating the two philosophers next to it.

Prove, by giving appropriate search commands from the initial state `init`, the following properties:

- (contiguous mutual exclusion): it is never the case that two *contiguous* philosophers are eating simultaneously.
- (mutual non-exclusion): it is however possible for two philosophers to eat simultaneously.
- (three exclusion): it is impossible for three philosophers to eat simultaneously.
- (deadlock) the system can deadlock.

4. Given a concurrent system specified by a rewrite theory $\mathcal{R}$, recall that an *invariant* is specified as a Boolean-valued predicate $I : \mathit{State} \rightarrow \mathit{Bool}$ with equations protecting `BOOL`, where *State* is the chosen sort of states in $\mathcal{R}$. This can be generalized to the notion of a *parametric invariant*, as a Boolean-valued function $I : \mathit{State}\ A_1 \dots A_n \rightarrow \mathit{Bool}$, where $A_1 \dots A_n$ are sorts in the signature of $\mathcal{R}$, and the equations defining I again protect `BOOL`. Then, in $I(S, x_1, \dots, x_n)$ the $x_1 : A_1, \dots, x_n : A_n$ are called the *data parameters* of the invariant.

   Let now $\mathit{init}(x_1 : A_1, \dots, x_n : A_n)$ be a term of sort *State* whose only variables are the $x_1 : A_1, \dots, x_n : A_n$. We then say that the parametric invariant $I(S, x_1, \dots, x_n)$ *holds* in the initial reachability model $\mathcal{T}_\mathcal{R}$ for the parametric family of initial states $\mathit{init}(x_1 : A_1, \dots, x_n : A_n)$ with data parameters $x_1 : A_1, \dots, x_n : A_n$ if an only if for each ground substitution $\rho = \{x_1 \mapsto u_1, \dots, x_n \mapsto u_n\}$ of the variables $x_1 : A_1, \dots, x_n : A_n$ the (unparametric) invariant $I(S, u_1, \dots, u_n)$ holds from the (ground) initial state $\mathit{init}(u_1, \dots, u_n)$.

   Consider now the following unordered communication channel between a sender and a receiver:

```
mod COMM-CHANN is protecting NAT .  protecting QID .
  sorts Msg MsgMSet QidList State .
  subsort Msg <  MsgMSet .
  subsort Qid < QidList .

  op nil : -> QidList [ctor] .
  op _;_ : QidList QidList -> QidList [ctor assoc id: nil] .
  op [_,_] : Qid Nat -> Msg [ctor] .
  op null : -> MsgMSet [ctor] .
  op _ _ :  MsgMSet MsgMSet -> MsgMSet [ctor assoc comm id: null] .
  op [_:_|_|_:_] : QidList Nat MsgMSet Nat QidList -> State [ctor] .

  vars N M I J : Nat . var MS : MsgMSet . vars L L' : QidList .
  vars A B : Qid .

  rl [send] : [A ; L : I | MS | J : L'] => [L : s(I) | [A,I] MS | J : L'] .
  rl [receive] : [L : I | MS [A,J] | J : L'] => [L : I | MS | s(J) : L' ; A] .
endm
```

where both the sender (on left) and the receiver (on right) hold buffers (lists) with lists of Qids to be sent (resp. already received), and the channel (in the middle) is a multiset of messages. We assume both counters will initially be at 0, and that the receiver's buffer will initially hold the `nil` list of Qids.

Intuitively, the counters are used to ensure *in order reception*. That is, even though the channel is a multiset so that the messages *can get out of order*, we expect and desire that, *at any given time* during the sending and receiving process, the list of Qids in the receiver's buffer will be arranged *in the exact same order* in which they were initially in the sender's buffer, although, of course, only at the end of the sending and receiving process will the receiver hold the *entire list* sent by the sender.

You are asked to do the following:

- Define the above property of in-order-reception as a parametric invariant of the form:

  ```
  op in-order-reception : State QidList -> Bool .
  ```

  Hint: the use of Maude's `owise` feature can make the definition easier.
- Write out the parametric family $init(L : QidList)$ of initial states for which the parametric invariant is an invariant.
- Show a screenshot of your results for verifying the invariant when L is instantiated to the Qid lists:

  ```
  'a ; 'b ; 'c

  'a ; 'b ; 'c ; 'd

  'a ; 'b ; 'c ; 'd ; 'e
  ```