CS 476 Homework #14, due at 10:45am on 5/4

Note: Answers to the exercises listed below should be emailed to `reedoei2@illinois.edu` in *typewritten form* (latex formatting preferred) by the deadline mentioned above. You should also email to `reedoei2@illinois.edu` the Maude code and screenshots for Problem 2.

1. **Part I**. Let $\mathcal{A} = (\mathbb{A}, \rightarrow_{\mathcal{A}})$ be a Σ-transition system. Let $a \in A_{[s]}$ be a state, and let $a' \in \mathit{Reach}_{\mathcal{A}}(a)$. Logically speaking, as for any other two sets, for any such $\mathcal{A}$, $a \in A_{[s]}$ and $a' \in \mathit{Reach}_{\mathcal{A}}(a)$, there are three possibilities:

   (a) $\mathit{Reach}_{\mathcal{A}}(a') \supseteq \mathit{Reach}_{\mathcal{A}}(a)$,

   (b) $\mathit{Reach}_{\mathcal{A}}(a') \subseteq \mathit{Reach}_{\mathcal{A}}(a)$, or

   (c) $\mathit{Reach}_{\mathcal{A}}(a') \cup \mathit{Reach}_{\mathcal{A}}(a) \neq \mathit{Reach}_{\mathcal{A}}(a')$ and $\mathit{Reach}_{\mathcal{A}}(a') \cup \mathit{Reach}_{\mathcal{A}}(a) \neq \mathit{Reach}_{\mathcal{A}}(a)$.

   Does any of those three possibilities *always* hold for any $\mathcal{A}$, $a \in A_{[s]}$ and $a' \in \mathit{Reach}_{\mathcal{A}}(a)$? If so, give a proof; and if not, give a counterexample.

   Part II. Suppose, as above, a Σ-transition system $\mathcal{A} = (\mathbb{A}, \rightarrow_{\mathcal{A}})$ where $\mathbb{A}$ protects the Boolean data type, i.e., the signature Σ_{Bool} of Boolean operations is a subsignature $\Sigma_{Bool} \subseteq \Sigma$, and $\mathbb{A}|_{\Sigma_{Bool}} \cong \mathbb{B}$, with $\mathbb{B}$ the standard, two-element Boolean Algebra. Let $a \in A_{[s]}$ and $a' \in \mathit{Reach}_{\mathcal{A}}(a)$, and let I be a unary Boolean-valued predicate whose input kind is $[s]$. Prove that there is implication relation [in one of the directions] between the following two statements:

$$\mathcal{A}, a \models \Box I \quad ?? \quad \mathcal{A}, a' \models \Box I$$

   and give a counterexample showing that the inverse implication does not hold in general. **Hint**. Note that if Σ is unsorted and its set F of function symbols is empty, a Σ-transition system is just a transition system, i.e., a *directed graph* (see *STACS*, §7.2). Therefore, your counterexample can just be a simple directed graph example.

   For Extra Credit. (Backwards Reachability Analysis). You can earn 10 more points on Problem 1 if you solve correctly the following problem. The problem's solution is not just of theoretical interest: it is eminently practical, since it is the basis of the so-called *backwards reachability analysis* of a system's properties. Let $\mathcal{A} = (\mathbb{A}, \rightarrow_{\mathcal{A}})$, $a \in A_{[s]}$, and I, a state predicate, be exactly as in **Part II** above. Suppose that I is an invariant from initial state a, i.e., $\mathcal{A}, a \models \Box I$. Let $[\![\neg I]\!]_{\mathbb{A}} = \{a \in A_{[s]} \mid I(a) = \mathit{false}_{\mathbb{A}}\}$. By the protecting Booleans assumption, we of course have, $[\![\neg I]\!]_{\mathcal{A}} = A_{[s]} \setminus [\![I]\!]_{\mathcal{A}}$, i.e., $[\![\neg I]\!]_{\mathcal{A}}$ is the set of states in $A_{[s]}$ where I does *not* hold. Call $[\![\neg I]\!]_{\mathcal{A}}$, a *co-invariant* from initial state a (in the exact sense that its complement *is* an invariant from a).

   Let $\mathcal{A}^{-1} = (\mathbb{A}, \rightarrow_{\mathcal{A}}^{-1})$ be the *inverse*, or *reverse*, Σ-transition system of $\mathcal{A}$, where for each kind $[s]$, $\rightarrow_{\mathcal{A},[s]}^{-1}$ is the *inverse relation* of $\rightarrow_{\mathcal{A},[s]}$, i.e., for each $a, a' \in A_{[s]}$ we have the equivalence,

$$a \rightarrow_{\mathcal{A},[s]} a' \quad \Leftrightarrow \quad a' \rightarrow_{\mathcal{A},[s]}^{-1} a.$$

   Prove the following equivalence:

$$\mathcal{A}, a \models \Box I \quad \Leftrightarrow \quad (\forall\, a' \in [\![\neg I]\!]_{\mathcal{A}})\ \ a \notin \mathit{Reach}_{\mathcal{A}^{-1}}(a').$$

2. 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.