T-106.5600 Concurrent Programming Examination March 2011/ hsa

Be concise and clear. The shorter you can be, the better.

1. Give all possible final values of the variable x after executing the following program. Explain your answer. Assume that all the assignments "x = ... ;" to x are executed as atomic actions.

```
int x = 0; sem s1 = 1, s2 = 0;
co P(s2); x = x + 7; P(s1); x = 2 * x; V(s1);                  #p1
// P(s1); x = x + 5; V(s2); x = 5 * x; V(s1);                  #p2
// P(s1); x = 7 * x; P(s2); x = x + 3; V(s2); x = 11*x; V(s1); #p3
oc
```

2. Petterson's algorithm for two processes critical section problem.

Algorithm 3.13: Peterson's algorithm	
boolean wantp ← false, wantq ← false integer last ← 1	
p	**q**
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: last ← 1	q3: last ← 2
p4: await wantq = false or last = 2	q4: await wantp = false or last = 1
p5: critical section	q5: critical section
p6: wantp ← false	q6: wantq ← false

(a) Show that the algorithm is safe by proving and using the invariants:

(1) $(p4 \wedge q5) \Rightarrow wantq \wedge (last = 1)$

(2) $(p5 \wedge q4) \Rightarrow wantp \wedge (last = 2)$

(b) Prove the liveness of process p by proving following formulas and using them to deduce a contradiction:

(3) $p4 \wedge \Box \neg p5 \Rightarrow \Box \Diamond (wantq \wedge (last \neq 2))$ "whenever p4 is executed, the condition is false"

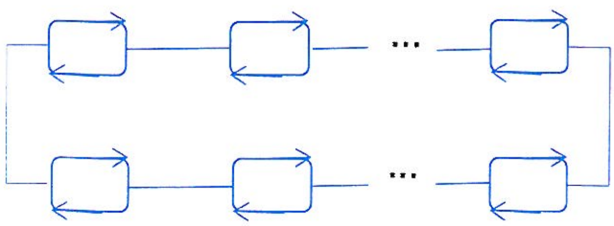
(4) $\Diamond \Box (\neg wantq) \vee \Diamond (last = 2)$ "either q stays at q1 (NCS), or it tries to enter CS"

(5) $p4 \wedge \Box \neg p5 \wedge \Diamond (last = 2) \Rightarrow \Diamond \Box (last = 2)$ " if p stays at p4, and last = 2, it stays so"

Legend: $\Box$A reads: A holds always. $\Diamond$A reads: "Eventually A", i.e. there will be a moment when A holds.

3. A long single-track railway loop connects N stations. There can be at most one train on the track between two stations. Stations have two parallel tracks capable of hosting K trains to each direction so that the trains to opposite directions may pass each other. Consecutive trains to each direction go automatically to their own tracks and do not collide to each other. Because trains can't back off, they must be synchronized with semaphores against head-on collision and deadlocks. Introduce necessary general semaphores with proper initializations and fill in their P() and V() operations to the following train code. Initially all the T trains on both directions are evenly distributed to "stay" on stations. Because of symmetry, the solution to one direction suffices. Analyze your solution for different values of K, and T : a) $T \leq K$, b) $K < T < K*N$, c) $T = K*N$.

```
while true{
    stay_on_station(i);
    i = (i+1)mod N;
    enter_track(i);
    run_on_track(i);
    leave_track(i);
    }
```



4. Symmetric solutions for the five dining philosophers problem using tuplespace. a) Solve the problem using one tuple, where the statuses of all the forks are stored, b) Solve the problem using a tuple for each fork separately, and in order to prevent deadlock, add one extra tuple to store the number of philosophers in the dining room, c) Try to solve the problem using one tuple where the statuses of all the philosophers are stored. Discuss the pros and cons of each solution. Define clearly the meaning of the different tuple types used and attach appropriate tags to them. The get_to_eat() and end_eat() operations should be simple algorithms without loops, and not using any other global variables except tuples. Linda tuple-space primitives are: `postnote ('tag', ...)`, `readnote ('tag', ...)`, `removenote ('tag',...)`. Indicate clearly in a `readnote(...)`, or `removenote(...)` operation when a matching tuple with an element value equal to a program variable value is sought for (syntax " `v=`"), from the case where an element value of a otherwise matching tuple is just assigned to a program variable (syntax " `v`").

5. One-lane bridge. Cars coming form north and south have to cross a river along a very long and narrow one-lane bridge. Cars driving to the same direction may be on the bridge at the same time, but cars heading to opposite directions can't. Consider the following generic monitor solution `One_lane_bridge` to the problem, where the cars are processes calling the public methods `cross_from_North()` and `cross_from_South()`.
a) Prove or disprove that the solution is safe and can't cause a deadlock using signal_and_wait or signal_and-continue -semantics. b) Transform it to a similar Java version, c) Provide a fair version of either a) or b), so that cars from each direction will get served in finite time.

```
monitor One_lane_bridge {
int ns =0;              //north-south cars on the bridge
int sn =0;              //south-north cars on the brigde
cond ns_c:              //condition to enter from north
cond sn_c;              //condition to enter from south
private procedure startNorth() {
      if (sn > 0) wait(ns_c);
      ns++
      }
private procedure endSouth() {
      ns--;
      if (ns == 0) signal_all(sn_c));    //signals possible waiting sn cars
      }
public cross_from_North() {   // this is needed to provide a simpler API
  startNorth();
  // north-south crossing operation is embedded here
  endSouth();
      }

// the south-north direction is symmetric
```