Possible Solutions, Final Exam, Reactive Systems 2002

Problem 1. (8 p)

a) $\Diamond\ (y \geq 4)$ does not hold. There are executions in which the value of x oscillates between 0 and 1, and hence the thid action is not continuously enabled. Hence there is no guarantee that the third action is ever executed.

b) $\Box\ (x \geq y)$ holds. This is an invariant, which is even inductive.

c) $\Box\ ((y = 7 \implies \Diamond\ (x = 7 \vee y = 8))$ holds. The reason is that if y becomes 7, then either x will become 7 or, if x never becomes 7 afterwards, the last statement will be continuously enabled unless y is incremnted to 8.

d) $\Box\ \Diamond\ (x = y)$ does not hold. The reason is that x and y are very loosely coupled. For instance, an execution could first increment x to 100, and then make sure that the difference between x and y is always larger than (say) 95.

e) $\Box\ \Diamond\ (x > y)$ holds. One possible motivation is that we have the invariant in b), and that furthermore x will be incremented infinitely many times due to the fairness constraint on the first action.

Problem 2. (8 p)

a) Here is one of *many* possible solutions.

$$\Box \left[\begin{array}{ll} & p \implies ((q \wedge \circ\ \neg q) \vee (\neg q \wedge \circ\ q)) \\ \wedge & \neg p \implies ((q \wedge \circ\ q) \vee (\neg q \wedge \circ\ \neg q)) \\ \wedge & \Diamond\ ((q \wedge \circ\ \neg q) \vee (\neg q \wedge \circ\ q)) \end{array} \right]$$

I am sure that the formula can be optimized or made smaller in many ways. Here is a suggestion (by Henrik Björklund)

$$\Box\ [(p \Leftrightarrow (\circ\ q \Leftrightarrow \neg q)) \wedge \Diamond\ (\circ\ q \Leftrightarrow \neg q)]$$

b) A never claim should accept the computations that violate the above formula. The formula can be violated *either* because the toggling of q is not following the rules of the first two lines, *or* because q eventually becomes constant (or, equivalently because p eventually gets stuck with a false value.

```
never{
   do /* Here is an initial state */
:: p == q -> goto checknegq
:: p != q -> goto checkq
od;
checkq:     do /* Check that q holds */
```

```
:: !q  -> goto fail
:: p    -> goto checknegq   /* q should toggle */
:: !p   -> skip
:: true -> goto accept
od;
checknegq:     do /* Check that not q holds */
:: q  -> goto fail
:: p    -> goto checkq  /* q should toggle */
:: !p   -> skip
:: true -> goto accept
od;
accept:     do /* Check that p is always false */
:: !p -> skip
od;
fail: skip /* terminate in an error state */
}
```

Problem 3. (14 p)

a) The property of mutual exclusion is an invariant. In this case, it might be stated, e.g., in temporal logic as

$$\Box\ [\forall j,k : 1 \leq j < k \leq n.\ \neg(PC[j] = crit \wedge PC[k] = crit)]$$

b) The idea is to find an inductive invariant which implies the property of mutual exclusion. The inductive invariant should capture the purpose of the variables x and y. This can be done as follows:

$$\forall j, 1 \leq j \leq n.[(PC[j] = crit \implies x = j) \ \wedge\ (x = j \implies AC = granted)]$$

or, equivalently

$$\forall j, 1 \leq j \leq n.[(PC[j] = crit \implies x = j) \ \wedge\ (x \neq 0 \implies AC = granted)]$$

To check that this invariant is inductive, note that the first conjunct can only be falsified by any of the two last actions: a closer inspection shows that it is not falsified. The second conjunct can only be falsified by any of the two first actions: a closer inspection also here shows that it cannot be falsified. Together, all the first conjuncts for all j imply that we have mutual exclusion, since x can assume at most one value in any state.

c) There may be starvation, since for a specific i, the fourth action may be disabled many times, when some other process is granted entry into the critical section.

d) There may be starvation also with strong fairness, because the arbiter may never set x to the desired value i. Intuitively speaking, the arbiter chooses (in its first action) to give the resource to some process, which at that time has set y to its index, but this choice need not be fair.

Problem 4. (6 p) The problem is that the setting of variable `x` in the first two `if`-statements makes not sense: The variable `x` may assume two different values before the state `granted` is entered. In some executions, both processes could take advantage of the time slot when `xx` has the appropriate value, and so both processes could enter the critical section.

One (of several) fixes could be to replace the arbiter by something which looks more like the action system.

```
proctype Arbiter()
{
free:    (y != 0) -> x = y;
granted: (x == 0) -> y = 0; goto free
}
```

Problem 5. (4 p) (* There was a typo, $p_2 \mathcal{W} p_2$ should be $p_2 \mathcal{W} q_2$. Sorry. It seems no one was fooled by this.

a) The property $(p_1 \wedge p_2) \mathcal{W} (q_1 \wedge q_2)$ can **not** be inferred, because it requires q_1 and q_2 to become true at the same instant, which we cannot be sure of.

b) $(p_1 \vee p_2) \mathcal{W} (q_1 \vee q_2)$ **can** be inferred.