NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2004/2005

CS4271 - CRITICAL SYSTEMS AND THEIR VERIFICATION

April 2005 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) questions and comprises **twelve** (**12**) printed pages including this page.
2. Answer **ALL** questions in the space provided in this booklet.
3. This is an **OPEN BOOK** examination.
4. **Please write your Matriculation Number below.**

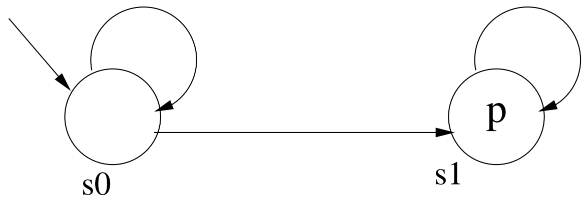
MATRICULATION NO.:

(This portion is reserved for the examiner's use only)

Question	Marks	Remark
Question A 12		
Question B 15		
Question C 11		
Question D 7		
Total 45		

A. **(4 + 4 + 4) = 12 marks**

1. Assume p is an atomic proposition. Does the following Kripke Structure satisfy the LTL formula GFp ? Does it satisfy the CTL formula $AGEFp$? Explain your answer.



Answer:

It does not satisfy GFp, because the trace s_0^ω does not satisfy GFp. It satisfies AGEFp since from all reachable states, a state satisfying p is reachable.

2. Assume p is an atomic proposition. Describe the following property in LTL: "along any path, a state satisfying p occurs at most once". Explain your answer.

   Answer:

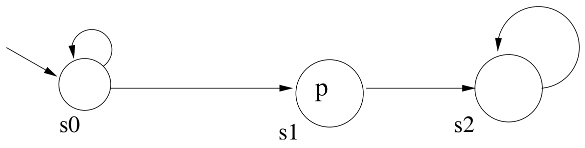
   $G\neg p \vee (\neg p U (p \wedge XG\neg p))$

   $G\neg p$ is true when p never occurs.

   If p occurs exactly once then the path starting from the state in which p occurs must satisfy $p \wedge XG\neg p$ (i.e. p occurs at the start and never occurs again). This explains the answer.

3. Construct an example Kripke Structure to show that the CTL^* formula $EGFp$ is not equivalent to the CTL formula $EGEFp$. Again, you may assume that p is an atomic proposition.

   Answer:



   The traces starting from the initial state are $s0^\omega$ and $s0^*s1s2^\omega$. None of these traces satisfy GFp simply because no state satisfying p lies inside a loop. So, the model does not satisfy $EGFp$. However, the trace $s0^\omega$ satisfies $GEFp$ since $s1$ (a state satisfying p) is reachable from s0. So, the model satisfies $EGEFp$.

B. (2 + 6 + 7) = 15 marks

1. Consider a traffic light controller which initially shows green light. It senses traffic data every second. Thus, starting from time=0, the traffic is sensed at time = 1 sec, 2 sec, and so on. If there is no traffic movement, the light turns from green to yellow. If there is traffic movement, the light stays green, but it can stay green for a maximum of 3 seconds at a stretch. The light always stays yellow for exactly one second after which it turns red. Once the light is red, again traffic is sensed every second. If there is traffic, then the light becomes green after staying red for a minimum of 2 seconds. If there is no traffic, the light eventually becomes green, after staying red for 3 seconds.

   Suppose you were trying to model the above controller as a Kripke structure. What aspects of the problem can you model and what aspects you cannot ? In particular, can you model the precise timing information *e.g.* traffic data being sensed every second.

   Answer:

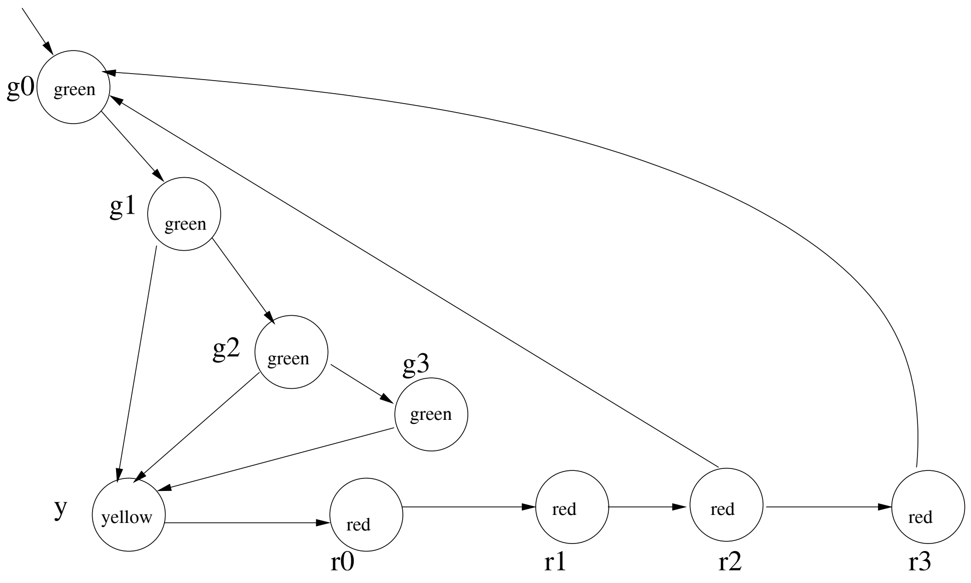
   We can model all the information given above except the precise timing information. For example, we can model that while the light stays green, the controller receives traffic data once or twice or thrice. But we cannot model that the light stays green for either 1 second or 2 seconds or 3 seconds. This is because the Kripke Structure only allows representation of time steps and not timing.

2. Model the traffic light controller as a Kripke Structure. Try to keep your model as close as possible to the informal description. Clearly state the set of atomic propositions used.

   Answer:

   The atomic propositions we model are *green*, *red* and *yellow*, that is, *green* is true if and only if the light is green and so on.

   Note that we also need to model the environment providing the traffic data. Since the environment's behavior is non-deterministic, we consider all possibilities. That is each time the behavior of the controller depends on traffic flow, we consider both possibilities – presence/absence of traffic.



3. We want to verify that the controller will never reach a state from where it is possible to stay red/yellow forever. Let us call it the **There-is-Hope** property. Specify this property in CTL. Use the explicit-state model checking algorithm discussed in class to show that your modeled Kripke Structure satisfies **There-is-Hope**. You need to show the main steps of the model checking computation.

   Answer:

   The property translated literally to CTL is

   $$\neg EFEG\neg green$$

   This is equivalent to the property $AGEFgreen$.

   To employ the model checking algorithm discussed in class we first rewrite the property as

   $$\neg E(trueUEG\neg green)$$

   The steps are as follows. St_varphi denotes the set of states in the model which satisfy φ.

   S = Set of all states = $\{g0, g1, g2, g3, y, r0, r1, r2, r3\}$

   $St_{green} = \{g0, g1, g2, g3\}$

   $St_{\neg green} = S - St_{green} = \{y, r0, r1, r2, r3\}$

   $St_{EG\neg green} = \{\}$

   $St_{E(trueUEG\neg green)} = \{\}$

   $St_{\neg E(trueUEG\neg green)} = S \supseteq \{g0\}$ where $g0$ is the only initial state. This completes the proof.

C. (3 + 4 + 4) = 11 marks

1. Here is a generic bit of a shift register implemented in SMV, that we discussed in one of our tutorials.

```
MODULE cell(left, lval)
{

   content: boolean;

   init(content) := 0;

   next(content) := case{
       left  : lval;
       1     : content;
   };

}
```

Define the value of the bit in the "next" clock cycle shown above as `next(content)`, as a boolean function $F_{next}(left, lval, content)$. You should write the function as a propositional logic formula.

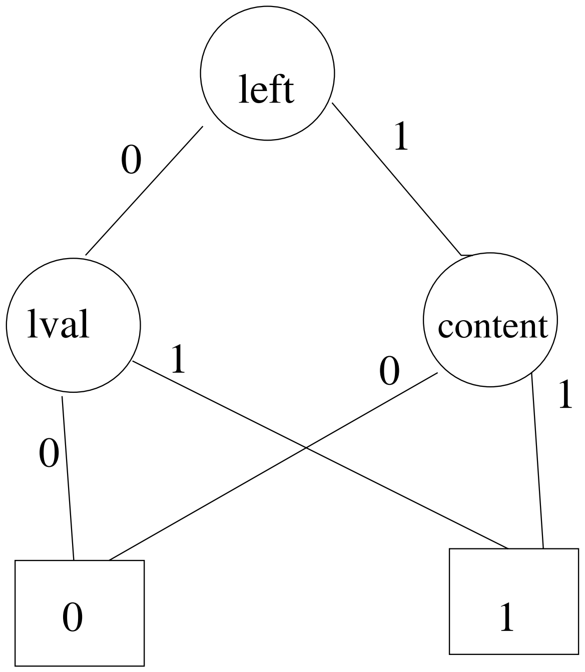
Answer:

$F_{next}(left, lval, content) = (left \Rightarrow lval) \wedge (\neg left \Rightarrow content)$

2. Give a variable ordering that produces the minimum sized Reduced Ordered Binary Decision Diagram for F_{next}. How can you exploit your understanding of the SMV program to construct this ROBDD directly without trying out all possible variable orders and the reduction steps for each of these variable orders ?

   Answer:

   Clearly, *left* must appear first in the variable order since it decides what variable's value goes to the output. The two orders that produce the minimum sized ROBDD are $left < lval < content$ and $left < content < lval$. The ROBDD for both of these orders will be the following.



3. Suppose we construct the Kripke structure model M_{bit} corresponding to our shift register bit. The state variables of this model should be *left*, *lval*, *content* – that is, we capture the bit's behavior for all possible values of *left* and *lval*. Define the boolean function that captures the set of states of M_{bit} which satisfy the CTL formula $AG(left \wedge lval \Rightarrow AX content)$. You can use the truth table notation or you can write it as a propositional logic formula. Explain your answer.

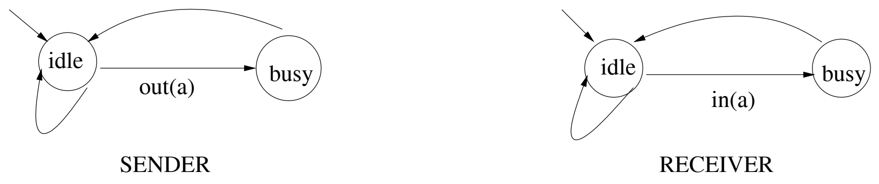
   Answer:

   We do not even need to construct M_{bit} to answer this question. The model M_{bit} has 8 states corresponding to the valuations of the three state variables. By definition of the transition relation of our SMV program (see definition of `next(content)`), any of these states will satisfy $AG(left \wedge lval \Rightarrow AX content)$. Hence the boolean function is

$$F_{AG(left \wedge lval \Rightarrow AX content)}(left, lval, content) = true$$

D. (5+2) = 7 marks

1. Consider two *parallel* processes which communicate via matching actions shown below. In the figure, some of the transitions are labeled. Any transition $s \rightarrow s'$ which is not labeled denotes an internal action; it is always enabled once control reaches s. But any transition $s \xrightarrow{in(a)} s'$ ($s \xrightarrow{out(a)} s'$) can only take place together with another process making a transition labeled with $out(a)$ ($in(a)$). In other words, such transitions denote a handshake between two processes. At any time step, any process can make (a) an internal action or (b) an input/output action provided the other process makes a matching output/input action. If the other process is not ready to make such a matching move, and there are no internal moves to make, then a process remains blocked. *At any time step, all the processes which are not blocked execute an action, that is, the processes are running in parallel.*



Model the example in SMV's input language. Your modeling should preferably be modular, that is, you should not put all the code into a main module.

Answer:

```
module main()
{
   S: myproc(R.label);
   R: myproc(S.label);
}

module myproc(msg)
{
   label: {a, none};
   state: {initial, busy};

   init(state) := initial;
   next(state) := case{
       state = initial & label = a & msg = a : busy;
       state = busy : initial;
       1    :  state;
   };
   label := case{
       state = initial : {a, none};
       state = busy    : none;
   };
}
```

2. Again, we consider the example in last page, but with a different mechanism for input/output. Assume that the sender outputs message a to a buffer. Similarly, the receiver inputs message a from the buffer. In other words, input/output no longer occurs via handshake; instead it is accomplished by message passing. Construct a CTL property which is true for our example under the handshake mechanism, but is false under the message-passing mechanism.

   Answer

   AG(S.state = R.state), or $AG\varphi$ where

   $$\varphi = (S.state = idle \Leftrightarrow R.state = idle) \wedge \\ (S.state = busy \Leftrightarrow R.state = busy)$$

   This is because in the message passing mechanism the sender can execute $out(a)$ and move to $busy$ before the receiver executes $in(a)$.

END-OF-PAPER