NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1, 2004/2005

CS6214 - AUTOMATED SOFTWARE VALIDATION

Nov/Dec 2004 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **five**(**5**) long questions and comprises **eleven** (**11**) pages, including this page.
2. Answer **ALL** questions in the space provided in this booklet.
3. **ALL** answers must come with the correct explanations. There is no credit for blind guesses.
4. This is an **OPEN BOOK** examination.
5. **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	8		
Question C	9		
Question D	11		
Question E	5		
Total	45		

A. **Modeling and Temporal Logics**
(5 + 4 + 3) = 12 marks

1. Are the two following Linear time Temporal Logic formula equivalent ? If yes, give a proof. If not, construct example traces to show that they are not equivalent.

$$\mathbf{F}(p\mathbf{U}q) \Leftrightarrow \mathbf{F}p \ \mathbf{U} \ \mathbf{F}q$$

You can assume that p and q are atomic propositions.

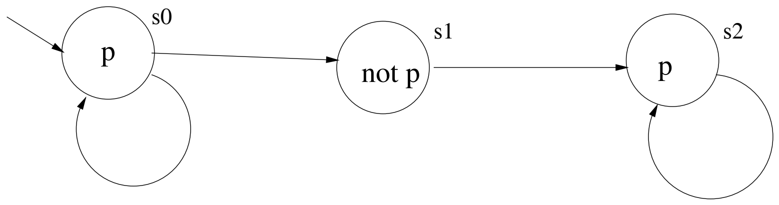
Answer: The two formulae are equivalent. Consider a trace π satisfying $F(pUq)$. Then by the definition of the F, U operators, there must exist a state in π which satisfies q. Let the first position in π where q is true be k. Then clearly $\pi^k \models pUq$, and hence $\pi \models F(pUq)$. Since $\pi^k \models q$ We see that $\pi \models Fq$; By definition of the until operator $\pi \models \varphi \Rightarrow \pi \models \psi U \varphi$ for any LTL properties φ, ψ. Thus, $\pi \models FpUFq$.

Now, consider any trace π such that $\pi \models FpUFq$. Again it means that there exists $k \geq 0$ such that $\pi^k \models Fq$ which means that there exists $m \geq k \geq 0$ such that $\pi^m \models q$. Then $\pi^m \models pUq$ and hence $\pi \models F(pUq)$.

This concludes the proof of equivalence of the two formulae. In fact we see that any trace with at least one state in which q is true, satisfies both the formulae and vice-versa.

2. Construct an example Kripke Structure to show that the CTL property **AFAG**p is not equivalent to the LTL property **FG**p, where p is an atomic proposition.

   Answer:



   The execution traces of this system model are of the form $s_0^* s_1 s_2^\omega$ or s_0^ω. These traces reach either s_0 or s_2 where they infinitely loop. Thus, they satisfy FGp. On the other hand, if we unfold the Kripke structure to construct the computation tree, in the leftmost path, we can never find a state which satisfies AG p.

3. Consider a resource allocation protocol where n processes $P_1, \ldots, P_n$ are contending for exclusive access of a shared resource. Access to the shared resource is controlled by an arbiter process. The atomic proposition req_i is true only when P_i explicitly sends an access request to the arbiter. The atomic proposition gnt_i is true only when the arbiter grants access to P_i. Now suppose that the following LTL formula holds for our resource allocation protocol.

$$\forall 1 \leq i \leq n \; G(req_i \Rightarrow (req_i \; U \; gnt_i))$$

Explain in English what the property means. Discuss whether this property is desirable in implementations of such protocols.

Answer: This means that for any process P_i which submits a request to the arbiter

- the request is kept asserted until it is granted.
- the request is eventually granted.

Overall, this is a desirable property of the protocol. This means that none of the processes suffer from the problem of starvation (*i.e.* a request not being serviced at all). However, the fact the request is kept asserted makes distributed implementation difficult (there are additional communication overheads).

B. **Model Checking**
(4 + 4 = 8 marks)

1. During our discussion on CTL model checking we argued that all operators can be expressed using EG, EU, EX. Devise a model checking algorithm which directly checks $M \models \mathbf{AF}p$ where p is an atomic proposition and M is a Kripke Structure whose labeling function specifies the truth/falsehood of p. In other words, you should not convert $\mathbf{AF}p$ to $\neg(\mathbf{EG}\neg p)$ as a pre-processing step; instead devise an algorithm for checking $\mathbf{AF}p$ directly.

   Answer:

   Input:
   Kripke Structure $M = (S, S_0, R, L)$

   Output:
   St_{AFp}, the set of states in M satisfying AFp

   Algorithm:

```
$St_{AFp} = \{s \mid s \in S \wedge p \in L(s)\}$;
do {
    $Temp = \{s \mid s \notin St_{AFp} \wedge \forall s' \; sRs' \Rightarrow s' \in St_{AFp}\}$
    $St_{AFp} = St_{AFp} \cup Temp$
} until no change in $St_{AFp}$
return $St_{AFp}$
```

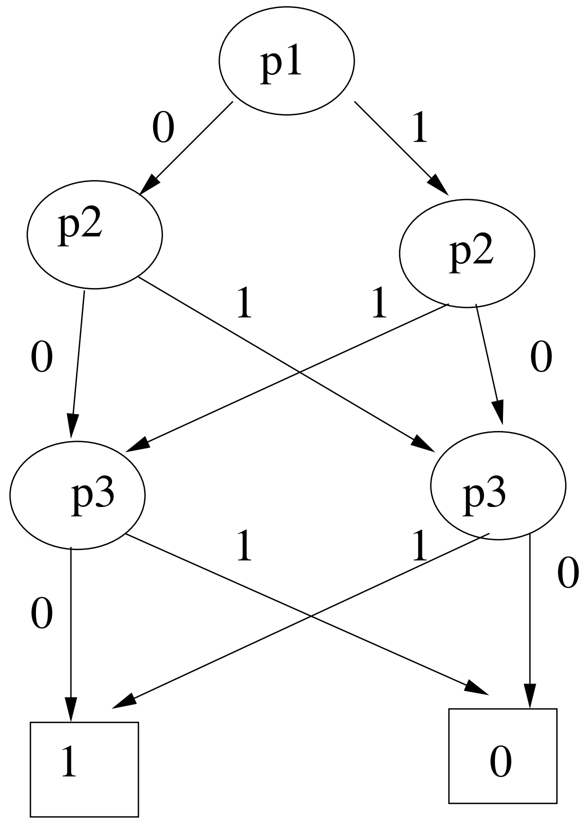
2. As discussed in class, Reduced Ordered Binary Decision Diagrams (ROBDD) can be used for symbolically describing the possible data states at a control location of a program. Let the possible data states of a particular control location be captured by the boolean function $f(p_1, \ldots, p_n)$ whose output is 1 if and only if the number of inputs p_i with value 1 is even; $p_1, \ldots, p_n$ are predicates which abstract the program's data store.

   What is the number of nodes in the ROBDD for this boolean function given the ordering $p_1 < p_2 < \ldots < p_n$? Explain your answer.

   How does the size of the ROBDD for this boolean function change with the ordering of $p_1, \ldots, p_n$. Explain your answer.

Answer:

The size of the ROBDD is independent of the variable ordering since changing the positions of any two inputs does not change the output. The ROBDD has a polynomial representation which essentially keeps track of the number of 1 inputs seen so far. The number of nodes is $2n + 1$ – the breakup being 1 node for p_1 (or the first variable in the variable order), 2 nodes each for all other predicates and 2 leaf nodes. For example, the ROBDD for n=3 is shown in the following.



C. **Model Extraction and Refinement**

$(4 + 5\ ) = 9$ marks

1. Consider the program

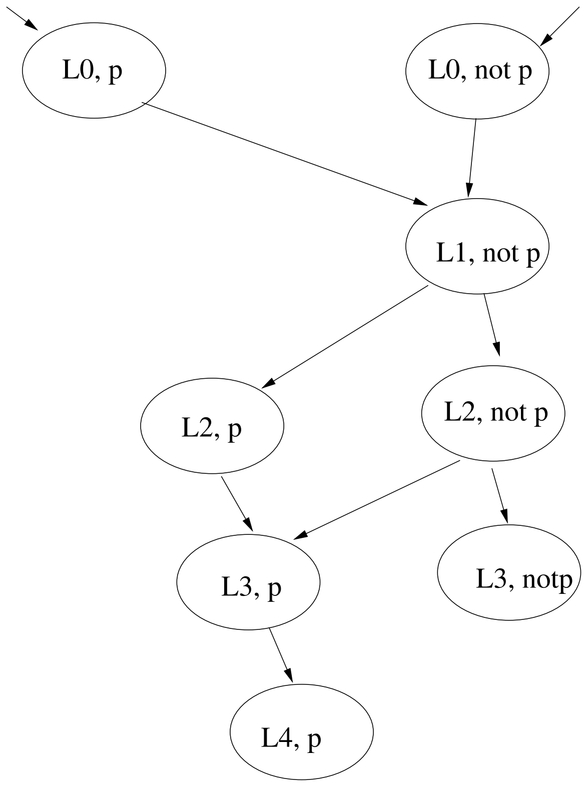
```
x = 0; x = x + 1; x = x + 1;
if (x > 2){ error }
```

Suppose we want to prove that the “error” location is never reached, that is, any trace reaching “error” is a counter-example. Show that the predicate abstraction $\{x > 2\}$ is insufficient to prove this property. You need to construct the abstract transition system for this purpose.

Answer: We first number the control locations

```
L0:  x = 0;
L1:  x = x + 1;
L2:  x = x + 1;
L3:  if x > 2
L4          error
```

The finite state transition system generated for the abstraction $\{x > 2\}$ is given below. We note $p \equiv x > 2$. This finite state transition system shows the reachability of location L4.



2. Refine your abstraction $\{x > 2\}$ by traversing the counter-example obtained. Show and explain all steps. Your refined abstraction should be sufficient to prove the unreachability of the "error" location.

Answer: A counter-example obtained is $(L0, p), (L1, \neg p), (L2, p), (L3, p), (L4, p)$. A forward traversal yields

```
Statement      Valuations     Constraint

  x = 0          (x, 0)         x == 0
  x =x+1         (x, 1)         x == 1
  x =x+1         (x, 2)         x == 2
  if (x >2)      (x, 2)         x== 2 /\ x > 2 ==> false
```

This shows the infeasibility of our counter-example. We could have proved this via a backwards traversal as well. To collect new predicates, we can employ many methods, but the only method which guarantees elimination of the encountered counter-example would require us to construct predicates out of all valuations and primitive constraints encountered in the counter-example traversal. In our case, this produces the abstraction $\{x = 0, x = 1, x = 2, x > 2\}$. This refined abstraction is clearly sufficient to prove the property (that is it gets rid of all counter-examples not just the one encountered) since it tracks all the values assumed by x and all conditionals on x in the program under verification.

D. Theorem Proving
(6 + 5) = 11 marks

1. Prove that the following Hoare triple specification is totally correct. Show all steps.

   $\{x \geq 0\}$

```
a = x; y = 1;
while (a > 0) {
     y = y * a;  a = a - 1;
}
```

   $\{y = x!\}$

   Answer: The first task is to construct a loop invariant. In this case we can choose $y = (x!/a!) \wedge a \geq 0$. The choice of $a \geq 0$ as a conjunct in the loop invariant is crucial for reasoning about both pre-loop and post-loop states.

   For one iteration of the loop body, by applying the rule of assignment

$$\{y * a = x!/(a-1)! \wedge a - 1 \geq 0\}\ \texttt{y = y * a; a = a - 1;}\ \{y = (x!/a!) \wedge a \geq 0\}$$

   This yields

$$\{y = (x!/a!) \wedge a > 0\}\ \texttt{y = y * a; a = a - 1;}\ \{y = (x!/a!) \wedge a \geq 0\}$$

   The above Hoare triple forms the premise of the while-rule so we can prove

$$\{y = (x!/a!) \wedge a \geq 0\}\ \texttt{while-loop}\ \{y = (x!/a!) \wedge a \geq 0 \wedge \neg(a < 0)\}$$

   which simplifies to

$$\{y = (x!/a!) \wedge a \geq 0\}\ \texttt{while-loop}\ \{y = x!\}$$

   Reasoning about the pre-loop states now involves showing

$$\{x \geq 0\}\ \texttt{a = x; y =1;}\ \{y = (x!/a!) \wedge a \geq 0\}$$

   which follows by laws of assignment.

   The proof for total correctness can be completed by showing termination with a as the measure.

2. The following program computes the quotient and remainder resulting from the division of x with y as divisor. Use the laws of Hoare logic to prove that the following program, if it terminates, correctly computes the quotient and remainder of (x/y). Does your reasoning need the assumption that division by zero is disallowed ?

```
r = x; d = 0;
while (r >= y) {
     r = r - y;  d = d + 1;
}
```

Answer: In the above program r is the remainder and d is the quotient computed from diving x by y. So, a loop invariant is $x = (d * y + r)$.

By applying the laws of assignment we have

$$\{x = (d + 1) * y + r - y\}\ \texttt{r = r - y; d = d + 1;}\ \{x = d * y + r\}$$

This simplifies to

$$\{x = d * y + r\}\ \texttt{r = r - y; d = d + 1;}\ \{x = d * y + r\}$$

We can also verify that this invariant holds for the pre-loop states.

$$\{true\}\ \texttt{r = x; d = 0}\ \ \{x = d * y + r\}$$

Also, using the while rule of Hoare logic we can show that $x = d * y + r \wedge r < y$ holds for the post-loop states which defines d as quotient and r as remainder.

The assumption of non-zero divisor is required for proving termination.

E. Miscellaneous
5 marks

Consider a multi-threaded Java program where n threads running on a single processor are trying to access a shared object using a round-robin scheme. We want to prove mutual exclusion of access of the shared object for any n. Can we employ the predicate abstraction and abstraction refinement based verification discussed in class ? Justify your answer. If your answer is yes, explain how. If your answer is no, can you suggest any alternative verification methods ?

Answer: In this case, the number of global control locations itself is unbounded. So, we cannot keep the control flow exact and abstract only the data store. Hence we cannot employ the conventional predicate abstraction and abstraction refinement discussed in class.

One possibility is to abstract the control flow, by maintaining the count of threads in critical section. Then if the program statements executed by the unboundedly many threads have a bounded representation (for example each thread executes the same code but with different thread-id), we can check that for each executed statement, the invariant *number of threads in critical section* < 2 is preserved. This constitutes a proof since we show that each transition preserves the invariant.

-END OF PAPER-