NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2007/2008

CS5219 - AUTOMATED SOFTWARE VALIDATION

April 2008 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) long questions and comprises **nine** (**9**) pages.
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 10		
Question C 12		
Question D 16		
Total 50		

A. Model Checking (6 + 6) = 12 marks

1. The **AU** operator of Computation Tree Logic (CTL) can be recursively defined as follows ($g1$ and $g2$ are arbitrary CTL properties).

$$\mathbf{A}(g1\mathbf{U}g2) = g2 \vee (g1 \wedge \mathbf{AX}\ \mathbf{A}(g1\mathbf{U}g2))$$

   Use the duality of the **AU** and **ER** operators in Computation Tree Logic (CTL) to get the recursive characterization of **E** ($g1$ **R** $g2$). Again, you may assume that $g1$ and $g2$ are arbitrary CTL properties.

   Can you also get a recursive characterization of **A** ($g1$ **R** $g2$) in a similar fashion? Explain your answer.

2. Exploit your recursive characterization of **E** ($g1$ **R** $g2$) to develop an algorithm for checking whether a given *finite-state* Kripke Structure M satisfies **E** ($g1$ **R** $g2$). You may assume that $g1$ and $g2$ are arbitrary CTL properties. Also assume that the following are available to you — (a) Set of states in M satisfying $g1$, (b) Set of states in M satisfying $g2$.

B. Theorem Proving
10 marks

Consider the following program fragment.

```
temp = src; target = 0;
while (temp != 0){
    target = target +1; temp = temp - 1;
}
```

- Under what conditions does the above program terminate?
- Show that whenever the program terminates, we will have `target == src` at the end of the program.

C. Model Extraction and Refinement
(5+7=12 marks)

1. Consider the following program fragment. Can you use the core model checking technique covered in this course to prove that `x == 0` holds at the end of the program? Give *detailed* explanation of your answer.

```
void main(){ int x = 0; x = x + 1;  x = x - 1; }
```

2. Consider the following program fragment.

   ```
   x = 0; while (x < 100){ x = x + 1; }
   ```

   Suppose we want to prove that `(x == 100)` at the end of the program. What is the initial abstract transition system we start with if we follow the abstract-modelcheck-refine methodology?

   What are the abstractions of the memory store (predicate abstractions) that we will encounter if we prove the property by abstraction refinement? Justify your answer in *details*.

D. **Software Testing and Debugging**
(5 + 5 + 6 = 16) marks

1. One method of software testing for inputs with large domains is called "equivalence partitioning". In this method, the domain of an input variable is *partitioned* into equivalence classes, so that from each equivalence class only one test input will be tried out. Now, there is a wide choice of when we define two test inputs to be "equivalent" and put them into an equivalence class. Suppose we define two test inputs to be equivalent when they produce the same path in the program.

    - Give an example program, where such an equivalence partitioning will lead to efficient testing, that is, only few test cases to try.
    - Give an example program, where such an equivalence partitioning will lead to inefficient testing, that is, too many test cases to try.

2. Consider the following program.

```
if (x > 2) y = x + z else y = x - z;

if (y > 0) return 0 else return 1;
```

- Give one sample test input for which the above code will return the value `0`.
- Characterize the set of all test inputs which cause the above code to return the value `0`. Explain your answer.

3. A Fibonacci sequence is given as follows.
   $fib(0) = 1$
   $fib(1) = 1$
   $fib(n) = fib(n-1) + fib(n-2)$ for $n \geq 2$.

   Following is a (buggy) program to compute the first n fibonacci numbers. What is the bug? Suggest a fix to the bug.

   Will the effect of the bug be observable in the printed values (note that you do not need to start computing all the printed values). Explain your answer.

```
int fib(int n){
int f, f0 =1, f1 = 1;
while (n>1){ n=n-1; f = f0+f1; f0 = f1; f1 = f;}  return f;
}
void main(){
int n = 99;
while (n > 0){ printf("fib(%d)=%d\n",n,fib(n)); n=n-1; }
}
```

-END OF PAPER-