NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1, 2007/2008
CS 5201 - FOUNDATION IN THEORETICAL CS

November 2007 Time Allowed: 3 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) long questions and comprises **six** (**6**) pages, including this page.
2. *Answer* **three** *out of* **four** *questions.*
3. Each question should be answered in a **separate** answer book.
4. This is an *OPEN BOOK* examination.
5. Write your Matriculation number in **all** the answer books.

1. Theory of Computation (4 + 4 + 2) = 10 marks

(A) Let $\Sigma = \{0, 1\}$ and let n range over the set of non-negative integers. For each n, let $L_n \subseteq \Sigma^*$ be the language given by $L_n = \{w : |w| \leq n\}$ where $|w|$ denotes the length of string w.

- Show that L_n is a regular language for every choice of n.
- Suppose $L = \bigcup_{n \geq 0} L_n$
  In other words, L is the union of the (infinite family of) languages L_0, L_1, L_2, ..., L_n, ... for all n. Is L regular? Justify your answer.

(B) Let $\Sigma = \{0, 1\}$. Suppose w is a non-null string of even length so that w can be written as $uxyv$ with x, y in Σ and $|u| = |v|$. Then we will say that xy is the middle of w. For example, in the string 00110011 we have 10 as its middle. Let $L \subseteq \Sigma^*$ be given by:

w is in L if and only if it is of non-null string of even length and its middle is 00 or 11.

Show that L is a context free language by constructing a (non-deterministic) push-down automaton that accepts L.

(C) Consider the set of formulas of first order logic. Assume that the logic has an infinite supply of individual variables, a finite set of predicate and function symbols (each with a fixed finite arity) and a finite set of constant symbols.

Let SAT be the set of *satisfiable* formulas of the logic. Recall that a formula φ is satisfiable, if and only if there exists a universe and an interpretation of the predicate/function/constant symbols within the universe such that φ is satisfied under this interpretation. A classic computability result (that Turing himself proved) is that the satisfiability problem for first order logic is undecidable. In other words, SAT is not a recursive set.

On the other hand, let VALID be the set of *valid* formulas of first order logic. Again, recall that a formula is valid if and only if it is satisfied in all universes under all interpretations. For example $\forall x(P(x) \wedge Q(x)) \Rightarrow (\forall x(P(x)) \wedge \forall x(Q(x)))$ is a valid formula. A classic result due to Gödel is that there is a sound and complete axiomatization of the set VALID. In other words, VALID is a recursively enumerable set.

Clearly, a formula φ is valid if and only if the formula $\neg\varphi$ is not satisfiable. Is it possible for SAT to be recursively enumerable? Justify your answer.

2. Logic and Artificial Intelligence (2 + 4 + 1 + 3) = 10 marks

(A) Formalize the following statement as a first-order logic formula.

Everybody can fool all the people some of the time, and some of the people all the time, but nobody can fool all the people all the time.

You may use three predicate symbols $person$, $time$ and $fool$, with the following meanings.

$person(p)$: p is a person

$time(t)$: t is a time

$fool(p, q, t)$: person p fools person q at time t.

(B) In this question, we consider first-order logic formula interpreted over the universe of integers. We use the following:

- a constant 0 interpreted as the integer zero,
- a unary function symbol *succ* interpreted as increment by one, and
- a unary function symbol *pred* interpreted as decrement by one.

Thus, the term $succ(0)$ is interpreted as the integer one, the term $pred(succ(0))$ is interpreted as the integer zero and so on.

We also use a unary predicate *even*. The interpretation of *even* is such that it makes the following first-order logic formula true. Nothing else is known about the interpretation of *even*.

$even(0) \wedge \neg even(succ(0)) \ \wedge \ \forall x \ (\ even(x) \Leftrightarrow even(succ(succ(x)))\ )$

- Let n be a non-negative integer. Show that for the term $succ^{2n}(0)$, the predicate *even* is true. Here $succ^{2n}$ means the *succ* function applied $2n$ times.
- Suppose t is a term which is interpreted as a non-negative even integer. Will $even(t)$ be interpreted as true? Justify your answer.

(C) Recall that a clause is a finite set of literals. Considering propositional logic only, a literal is an atomic proposition or its negation. Given a clause C in propositional logic and an atomic proposition p, what are the possible clauses resulting from applying resolution to C and $\{p, \neg p\}$.

(D) A propositional logic formula in Disjunctive Normal Form (DNF) is of the form

$$(Lit_1^1 \wedge \ldots \wedge Lit_{k_1}^1) \vee \ldots \vee (Lit_1^n \wedge \ldots \wedge Lit_{k_n}^n)$$

where Lit_j^i is a literal (atomic proposition or its negative).

Show that every propositional logic formula has a DNF representation.

3. Algorithms (2 + 4 + 4) = 10 marks

(A) Consider a divide and conquer algorithm. Assume the following:

- It takes constant amount of time to solve a given problem of size up to a, where a is a constant.
- Any given problem of size $n > a$ can be divided into two sub-problems: one of size a, and one of size $n - a$.
- In order to divide a problem of size $n > a$ as above, time $O(n)$ is required.
- In order to combine the solution of a problem of size a with a solution of a problem of size $n - a$, where $n > a$, time $O(n)$ is required.

Claim: The running time of the algorithm is $O(n \lg n)$, for any constant a.

Indicate whether the claim is true or false. Justify your answer.

(B) The suitability of algorithms often depends on the context in which it is used. What is preferable in one context may be undesirable in another. For each of the following, give *one* context (e.g. limited space, limited time, specific usage requirements, etc.) where one algorithm may be preferable to another algorithm together with a reason to support the preference for each of the algorithm compared.

Example: *When might insertion sort be preferable to merge sort and vice versa?*
Insertion sort might be preferable to merge sort when space is a limiting resource because unlike merge sort, insertion sort operates in-place. Merge sort might be preferable to insertion sort when running time is important as merge sort runs in $O(n \lg n)$ time while insertion sort runs in $\Omega(n^2)$ time.

1. Suppose we need a data structure for storing a set of integers. When might hashing be preferable to using a balanced binary search tree and vice versa?
2. Suppose we are interested in collision resolution in hashing. When might chaining be preferable to open addressing and vice versa?

(C) You are developing a new path finding algorithm for your mobile ad-hoc network. Your network is represented as a graph $G = (V, E)$ with n vertices and m edges. Each edge $(u, v) \in E$ has a probability $p(u, v) \in [0, 1)$ that a message passed along the edge will be corrupted by error. Assume that the probability of error on different links are independent.

1. Argue that the probability of receiving an uncorrupted message is $\prod_{i=1}^{k-1}[1 - p(v_i, v_{i+1})]$ if the message is passed along the path $v_1 \to v_2 \to \cdots \to v_k$.
2. Give an efficient algorithm to find the "best path" from a vertex $v_s \in V$ to another vertex $v_t \in V$. Your "best path" should maximize the probability that the message is received uncorrupted from v_s to v_t.

   Furthermore, the best path that you find must satisfy the following — the distances from v_s along the best path $v_s \to v_2 \to \cdots \to v_t$ must be strictly increasing.

   Assume that you are given a sorted list of distances of each node from the vertex v_s.

   For simplicity, your algorithm may return just the probability under the best path, rather than the path itself. Give pseudo-code.

4. Principles of Programming Languages (2+ 3 + 2 + 3) = 10 marks

(A) Most modern programming languages realize lexical scoping as the main mechanism for associating identifier with their declaration. In lexical scoping, the declaration of an identifier is found by working from the identifier occurrence outwards until a declaration is found that matches the variable. Consider the following program.

```
function f(x) {
              function g(z) {
                              return x;
                            };
              function h(x) {
                              x = x + 3;
                              return g(x);
                            };
              return h(2)+x;
           }
f(0)
```

What is the result of executing this program using lexical scoping? Provide an explanation for your answer.

(B) Some programming languages provide the possibility of returning functions as results of calling other functions. The language JavaScript, for example, permits the following program:

```
function k(x) {
     x = x + 3;
     return function(y) {
                 return x + y;
            }
}
var z = k(1);
var w = z(2);
```

Note that the inner function is not named; it is "anonymous".

i) What is the result of executing the program above?

ii) Choose any object-oriented programming language that you are familiar with. Discuss whether you can achieve a similar effect in your chosen language.

(C) JavaScript is a dynamically typed language, which means that the types of arguments are checked at runtime, possibly resulting in runtime errors. Other languages such as Java require that the programmer declares the types of all identifiers.

Assuming that all functions have single arguments, and as primitive values, we only consider integers, all possible types can be described by the following context-free grammar:

$$\begin{array}{lcl} \textit{type} & ::= & \texttt{int} \\ & | & \texttt{(}\ \textit{type}\ \texttt{->}\ \textit{type}\ \texttt{)} \end{array}$$

where *type* is a non-terminal and "`int`", "`->`", "`(`", and "`)`" are terminal symbols.

i) Give the type of function `h` in part **(A)**, using the syntax described by the grammar.

ii) Give the type of function `k` in part **(B)**, using the syntax described by the grammar.

(D) In compiler construction, the issue of memory allocation arises whenever the source program requires it. The compiled code needs to allocate appropriate memory at runtime. We distinguish three types of memory:

Static allocation

Stack allocation

Dynamic (heap) allocation

Assume a compilation of the following object-oriented program (which uses C++/Java syntax) to efficient *native code* (machine code that can directly run on processor).

```
class A
   static public int x;
   public int y;
   public static void main(String args[]) {
      A a = new A(3);
      int w = a.f(5));
   }
   public A(int z) {
      y = z;
   }
   public f(int v) {
      int localvar = x + 1;
      return g(v)+localvar;
   }
   public g(int u) {
      return u + y;
   }
}
```

Assume that the program is executed by calling the function `main` with arbitrary argument strings. Describe the flow of the program execution and whenever memory is allocated, describe what type of allocation is used, and give the rationale for using the respective type of allocation.