NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2005/2006
CS5201 - FOUNDATION IN THEORETICAL CS

April 2006 Time Allowed: 3 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) long questions and comprises **five** (**5**) 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. **Please write your Matriculation Number below.**

MATRICULATION NO.:

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

Question	Marks	P/F	Remark
1. Theory of Comp.			
2. Algorithms			
3. Logic			
4. Prog. Languages			
Total			

1. Theory of Computation

A Consider a set of strings L_c over the alphabet $\Sigma = \{0, 1\}$ where L_c contains exactly the set of strings over Σ with length c (for some positive integer constant c). Your professor has tried to construct the following argument in an attempt to show that the language L_c is not regular.

Let us assume that L_c is regular. Consider the string $0^c \in L_c$. From the pumping lemma we know that there should be strings u, v, w s.t. $0^c = uvw$ and $uv^k w \in L_c$ where $k \geq 0$ and $|v| > 0$. Clearly, for all $k \geq 2$ the length of $uv^k w$ is more than c and hence $uv^k w \notin L_c$. This results in a contradiction showing that our original assumption was wrong, that is, L_c is not regular.

Comment on the correctness of the professor's argument. Also, irrespective of whether you think the professor's argument is correct or not, comment on whether the language L_c is regular for all c. Give detailed justification for all your answers.

B Recall that a context-free grammar is a 4-tuple $(Vars, \Sigma, Start, Prod)$ where $Vars$ is the set of non-terminal symbols, Σ is the set of terminal symbols, $Vars \cap \Sigma = \emptyset$, $Start \in Vars$ is the start symbol and $Prod$ is a finite set of production rules. Each production rule is of the form

$$A \rightarrow \alpha$$

where $A \in Vars$ and $\alpha \in (Vars \cup \Sigma)^*$.

- Construct a context free grammar for the following language L. The set of terminal symbols is $\{0, 1, 2\}$.
$$L = \{0^l 1^m 2^n \mid l > m + n,\ m > 0,\ n > 0\}$$
- Prove that any string in L is accepted by your grammar and vice-versa.

C We consider a variation of the finite-state automata accepting regular languages. Define

$$A = (S, \Sigma, S_0, \rightarrow, Acc)$$

where, as in regular languages, S is a finite set of states, Σ is a finite alphabet, $S_0 \subseteq S$ is the set of initial states, $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation and $Acc \subseteq S$ is the set of accepting states.

However, the notion of acceptance of a string is *different*. Our automaton accepts collections of *infinite* strings. The language $L(A)$ of the automaton A defined above is the following set of *infinite* strings over the alphabet Σ.

$$L(A) = \{\sigma \mid \sigma \in \Sigma^\omega \text{ and } \sigma \text{ has a } \textbf{run } r \text{ in } A \text{ such that } inf(r) \cap Acc \neq \emptyset\}$$

A **run** r of a string σ in A is an infinite sequence of states of A obtained by running σ from an initial state. That is, $r[0] \in S_0$ and for all $i \geq 0$, $r[i] \xrightarrow{\sigma[i]} r[i+1]$. Also, for a run r, $inf(r)$ is the set of states appearing infinitely often in r.

Using the above notion of acceptance, construct an automaton which accepts all infinite strings σ where (a) each symbol of σ is 0 or 1, and (b) σ contains only finitely many occurrences of 1. Your automaton need *not* be deterministic.

2. Algorithms

A Array `a[1..n]` can be sorted by insertion as follows:

```
sort( a, n ) {
  for( i = 2;  i <= n;  i++ ) {
    v = a[i];
    for( j = i-1;  j >= 1;  j-- ){
      if( a[j] > v ) a[j+1] = a[j];  else break;
    }
    a[j+1] = v;
  }
}
```

Give three inputs such that for each input the `if` condition is evaluated $(n-1)n/2$ times.

B Assume the availability of a program `longestCS( a, b )` that finds a longest common subsequence of the arrays `a[1..m]` and `b[1..n]`. Give an algorithm to find a longest non-decreasing subsequence of a sequence of numbers.

C A Conjunctive Normal Form (CNF) is a boolean expression that is an AND of boolean expressions, each of which is the OR of one or more literals. A literal is a boolean variable or its negation.

Assume the availability of a program `IsSatisfiable( c )` that determines if a CNF `c` is satisfiable. Find a satisfying assignment for a satisfiable CNF in time $O(nf(n))$ where $O(f(n))$ is the running time of `IsSatisfiable( c )` for a CNF `c` of length n.

D Recall that a topological sort of a directed acyclic graph (DAG) is a total ordering of its vertices such that if $i \rightarrow j$ is an edge in the DAG, then vertex i appears before vertex j in the ordering.

Consider a DAG with n vertices and m edges where the vertices are identified with the integers $1, \ldots, n$. Give a $\Theta(m+n)$ algorithm to determine if a permutation of the integers $1, \ldots, n$ is a topological sort of the given DAG.

E Given a set of positive integers S and a positive integer t, the subset-sum problem asks if there is a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$. It is well-known that this problem is NP-complete.

We can define the subset relationship among multi-sets as follows. If S and S' are multi-sets, $S' \subseteq_{multiset} S$ if and only if the set of distinct elements in S' is contained in the set of distinct elements in S. Thus,

$$\{3,3,3\} \subseteq_{multiset} \{2,2,3,4\}$$

since

$$\{3\} \subseteq \{2,3,4\}$$

This produces a subset-sum problem for multi-sets stated as follows.

Given a multi-set of positive integers S and a positive integer t, is there a multi-set $S' \subseteq_{multiset} S$ such that $\sum_{s \in S'} s = t$.

Give a polynomial-time solution to this problem.

3. Logic and Formal Reasoning

Analyze the following programs and write formulas which describe the dependence of the output from the input. For example, the program

```
function d(n)
  { var a = 0;                                  // initializing variable a
    while (a+a+a+2<n) { a=a+1; }                // { means "begin", } means "end"
    return(a); }                                // d(n) takes current value of a
```

is uniquely specified by the formula

$$\forall x \forall y \, [(3 * d(x) \leq x) \wedge (3 * y \leq x \Rightarrow y \leq d(x))].$$

You can use the standard arithmetical operations $+, -, *$ in formulas, existential and universal quantifiers, variables taking the values of natural numbers and the usual logical connectives. The inputs and outputs in the following functions are always natural numbers. Comments explaining syntax are behind a double slash.

```
[A]  function e(m,n)
       { if ((m<1)||(n<1))                      // || means "or"
           { return(0); }                       // e(m,n) is 0 if m is 0 or n is 0
       var a=m; var b=n;                        // initializing variables
       while ((a<b)||(b<a))
           { if (a<b) { a=a+m; }
           else { b=b+n; } }
       return(a); }

[B]  function f(n)
       { var a = 0; var b = 0; var c = 0;
       while (a+a+b < n)
           { c = c+b+b+b+a+a+a+1;
           b = b+a+a+1; a = a+1; }
       return(c); }

[C]  function g(n)
       { var a=1; var b=0; var c=2;
       while ((c<n)&&(n>1))                     // && means "and"
           { while (b<n) { b=b+c; }
           if (b>n) { b=0; }
           else { a=c; b=0; }
           c=c+1; }
       return(a); }

[D]  function h(n)
       { var a=1; var b=0;
       while (b<n) { a = a+b+b+1; b=b+1; }
       while (b+1<n+n) { a = a+1; b=b+2; }
       return(a); }
```

4. Principles of Programming Languages

A Consider the following expression syntax of a programming lanaguage:

$$e \quad ::= \quad v \mid c \mid e_1 - e_2 \mid e_1 * e_2$$

Note that only two binary operators $-$ and $*$ are available. c ranges over all integers, and v ranges over all variables.

1. Use the above syntax to provide a syntax tree for the following expression:

$$x - 2 * y - 4$$

2. Change the syntax above so that it will generate only syntax trees that obey a certain precedence among the operators. Clearly state what precedence of operators you are assuming for constructing the changed syntax definition.
3. Change the syntax obtained in (2) so that $-$ is left associative and $*$ is right associative.

B The `while`-loop syntax such as the following

`while` b `do` S

is usually considered indispensable in any imperative language. Nevertheless, it is also widely agreed that `while`-loop programs can always be converted to programs without involvement of loops, through the deployment of recursive functions.

Provide a systematic and semantics-preserving translation of `while`-loop codes to ones without `while`-loop (by employing function calls). Discuss how the correctness of your translation may be affected by the calling semantics of the subject programming language under translation. Specifically, would your translation be valid if the subject code adheres to call-by-value semantics? What about if it adheres to call-by-reference semantics?