NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1, 2005/2006
**CS5201 - FOUNDATION IN THEORETICAL CS**

Nov 2005 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. **ALL** answers must come with the correct explanations. There is no credit for blind guesses.
5. This is an **OPEN BOOK** examination.
6. **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** Let $\Sigma = \{0, 1\}$. Let $L_R \subseteq \Sigma^\star$ be the language where a string $s \in L_R$ if and only if the number of occurrences of the symbol 0 in $s$ is not divisible by 5.

Construct a deterministic finite state automaton which accepts the language $L_R$.

**B** Let $\Sigma = \{0, 1\}$. Consider the language $L_{CF} \subseteq \Sigma^\star$ given by:
$s \in L_{CF}$ if and only if the number of times 0 appears in $s$ is less than or equal to the number of times 1 appears in $s$.

- Argue that $L_{CF}$ is *not* regular.
- Construct a pushdown automaton which accepts the language $L_{CF}$. The pushdown automaton need not be deterministic. Acceptance is by final state. In other words, if the automaton enters a final state at the end of reading the string $s$ , then $s$ is deemed to be accepted regardless of the contents of the stack.

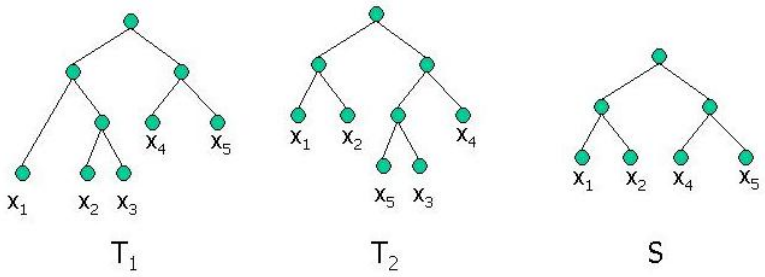


Figure 1: $S$ is the maximum agreement subtree of $T_1$ and $T_2$. It has 4 leaves.

## 2. Algorithms

**A** By calling $sort(T, 1, n)$, we can sort the numbers in $T[1..n]$ in increasing order. Can you analyze the time complexity of the function *sort*? You may assume that $sqrt(x)$ is a function which returns the integer part of $\sqrt{x}$. Show all the steps of your analysis.

```
int sort(T, ii, jj) {
  int i, j, k, s;
  if (ii >= jj) return;
  s = jj - ii + 1;
  sort(T, ii, ii+sqrt(s)-1);
  sort(T, ii+sqrt(s), jj);
  i = ii; j = ii+sqrt(s);
  for k = ii to jj {
    if (T[i] > T[j]) {
      R[k] = T[j]; j = j + 1;
    } else {
      R[k] = T[i]; i = i + 1;
    }
  }
  for k = ii to jj {
   T[k] = R[k];
  }
}
```

**B** Consider two binary rooted trees $T_1$ and $T_2$ which are distinctly leaf-labeled by $\{x_1, x_2, \ldots, x_n\}$. Figure 1 shows two example trees $T_1$ and $T_2$ which are distinctly leaf-labeled by $\{x_1, x_2, x_3, x_4, x_5\}$. Given two binary rooted trees $S$ and $T$ we say that $S$ is a topological subtree of $T$ if and only if the following hold.

- Let $Leaf_S$ be the set of leaves of $S$ and $Leaf_T$ be the set of leaves of $T$. Then $Leaf_S \subseteq Leaf_T$.
- Furthermore, let $T'$ be the subtree of $T$ induced by $Leaf_S$. Then $T'$ and $S$ are "homeomorphic", that is, we can transform $T'$ to $S$ by collapsing internal nodes of $T'$ with only one child.

A leaf-labeled rooted binary tree $S$ is called an agreement subtree of $T_1$ and $T_2$ if $S$ is a *topological subtree* of both $T_1$ and $T_2$. Furthermore a maximum agreement subtree is an agreement subtree with the largest possible number of leaves (see Figure 1 for an example). Propose an $O(n^2)$-time algorithm to compute the number of leaves of the maximum agreement subtree of $T_1$ and $T_2$.

## 3. Logic and Formal Reasoning

**A** Formalize the following argument in first-order logic (you could encode it as a single first-order logic formula). You should clearly state any predicate and function symbols that you use. Show that the argument is incorrect by showing that the corresponding first-order logic formula is not valid.

*All students are either undergraduate students or postgraduate students, but not both.*
*Only students are allowed to appear for qualifiers.*
*All postgraduate students appeared for qualifiers.*
*Raman and Melvin appeared for the qualifiers.*
*Therefore,*
*Raman and Melvin are postgraduate students.*

**B** Consider the programs given in the following. Formally prove that each of these programs computes in $z$ the product of $x$ and $y$. That is, at the end of the program $z = x0 * y0$ where $x0$ and $y0$ are the initial values of variables $x$ and $y$. You may assume that $odd(x)$ is a function which returns true if $x$ is odd and false otherwise; the code for this function is not shown. Also, $x, y, z$ are integer variables; the operation / denotes integer division, that is, $a/b$ returns the integer quotient resulting from dividing $a$ by $b$ where $a, b$ are integers. (*Hint:* You may want to prove by induction on loop iterations.)

```
z = 0;
while (x != 0){
  z = z + y;
  x = x - 1;
}
```

```
z = 0;
while (x != 0){
  if odd(x){ z = z + y;}
  y = y * 2; x = x/2;
}
```

## 4. Principles of Programming Languages

**A** Consider an imperative programming language defined by the following BNF:

$$\begin{array}{rcl} S & ::= & x \texttt{ := } E \\ & | & S \texttt{ ; } S \\ & | & \texttt{let float } x \texttt{ = } E \texttt{ in } S \texttt{ end} \\ & | & \texttt{print } E \end{array}$$

The non-terminal $x$ represents a given set $I$ of identifiers, the terminals `:=`, `;`, `let`, `=`, `in`, , `float`, `print`, and `end` are keywords, and the non-terminal $E$ is defined by the BNF

$$E \quad ::= \quad f| \; x \; |\texttt{( } E \texttt{ )}|E \texttt{ * } E$$

where the non-terminal $f$ stands for floating point numbers in the usual notation, and the terminal `*` represents multiplication of floating point numbers.

As an example, the following program

```
let float x = 7.0 in
   let float m = 2.5 in
      let float f = 0.0 in
         f := m;
         print f;
         f := f * x;
         print f
      end
   end
end
```

results in the printing of the number sequence $2.5, 17.5$.

Define a function $vars : E \to 2^I$ that returns the set of all identifiers that occur in a given expression. Example:

$$vars(\texttt{1.5 * x * y}) = \{\texttt{x}, \texttt{y}\}$$

Your description of *vars* must be constructive; it should allow to construct the set of identifiers from any given expression using simple set operators.

**B** The language $L'$ is an extension of $L$ such that boolean values with conjunction `&` can be used in addition to floats with multiplication. The operator `>` tests whether the floating point number on the left is bigger than the floating point number on the right. As an example, the following program is in $L'$.

```
let float x = 4.5 in
   let boolean b = true in
      print ( x > 0.0 ) & false
   end
end
```

Give a definition of the syntax of $L'$ in BNF.

**C** In this question, we use Java-like syntax for object-oriented languages. As in Java, we assume that all methods are virtual; in a method call `obj.f()`, the class of the receiving object `obj` defines which method `f` is called.

Some object-oriented programming languages support multiple inheritance. In this inheritance mechanism, a class can extend multiple parent classes. Example:

```
class A {int f() {return 1;}}

class B {int g() {return 2;}}

class C extends A, B {int h() {return 3;}}
```

Objects of class `C` can handle calls of the methods `f`, `g` and `h`.

Multiple inheritance leads to situations that require a careful definition of the meaning of multiple inheritance, to avoid ambiguity in language semantics.

First construct an example program to show how programs with multiple inheritance can have ambiguous meaning. Then, formalize an unambiguous definition of multiple inheritance that can handle your example without an error.