NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2006/2007
CS 5201 - FOUNDATION IN THEORETICAL CS

April 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

(A) You are familiar with the Pumping Lemma for regular languages (abbreviated as **PLem** in this question), given in the following.

(**PLem**) For any regular language L, there exists an integer N such for any string $\sigma \in L$ with length $\geq N$, we can find strings x, y, z such that

- $\sigma = xyz$
- length of xy is $\leq N$
- length of y is > 0
- for all $k \geq 0$, $xy^k z \in L$.

Now, here is another Pumping Lemma, which we call **PLem′**.

(**PLem′**) For any regular language L representing an infinite collection of strings, there exist integers N_1 and N_2 such that

- $N_2 > 0$
- for all $k \geq 0$, L contains a string of length $N_1 + k \times N_2$.

Using the fact that **PLem** is true, show that **PLem′** holds. In other words, prove that

$$\textbf{PLem} \Rightarrow \textbf{PLem}'.$$

(B) Consider any language of the following form over the alphabet $\{0, 1\}$.

$$\{\sigma \mid num(0, \sigma) = func(num(1, \sigma))\}$$

where $num(0, \sigma)$ and $num(1, \sigma)$ denote the number of occurrences of 0s and 1s respectively in the string σ. Furthermore, $func$ is some function from natural numbers to natural numbers, that is, $func : nat \rightarrow nat$ where nat denotes the set of natural numbers.

Show that any language of the above form cannot be regular unless there exists an integer N_{bound} such that for all natural numbers n, we have $func(n) \leq N_{bound}$. You may want to use the Pumping Lemma.

(C) Consider the following language over the alphabet $\{0, 1\}$.

$$L = \{\sigma \mid num(0, \sigma) = num(1, \sigma)\}$$

where $num(0, \sigma)$ and $num(1, \sigma)$ denote the number of occurrences of 0s and 1s respectively in the string σ. From Part (B) above, we know that this language is not regular. It turns out that it is context-free and we will construct a context-free grammar for L.

(C.1) For any non-null string $\sigma \in L$, if the first and last symbol of σ are the same (*i.e.* both are 0 or both are 1), show that $\sigma = xy$ where $x \in L$ and $y \in L$.

(C.2) Use the claim in Part (C.1) above to construct a context-free grammar for L.

2. Logic and Artificial Intelligence

(A.1) Express the following statements using first order logic. Note that you should use a consistent notation for your predicates in your solutions to this part and part **A.2**.

1. All blogs are webpages.
2. All webpages that have dated entries and are linked to by a blog are blogs.

(A.2) Using your answers for part **A.1**, prove or disprove the assertion *Steve's page is a blog* using the following knowledge base.

1. Both Steve's and Mary's webpages have dated entries.
2. Jim's blog links to Mary's webpage.
3. Mary's webpage links to Steve's webpage.

(B) Discuss the potential difficulties in using propositional logic to encode the rules of chess. Would a first-order version of this rule-set be more suitable? Defend your answer.

(C) One method of proving a formula in first order logic (FOL) is to transform the formula into propositional logic and use propositional logic proving techniques to do the work. Explain under what type of conditions this would be a plausible technique for inference in an FOL knowledge base.

3. Algorithms

(A) What does the following algorithm accomplish? Give and explain its best and worst case time complexity.

Algorithm 1 A mysterious algorithm

```
1: mystery (int a[], int n)
2: int i = 0;
3: while i < n − 1 do
4:    if a[i] > a[i + 1] then
5:       SWAP(a[i],a[i + 1])
6:       i = 0;
7:    else
8:       i = i + 1;
9:    end if
10: end while
```

(Footnote: the function SWAP(x, y) exchanges the two values, i.e. $t = x; x = y; y = t;$.)

(B) Give an algorithm to compute the number of connected components in a graph in $O(n + e)$ time, where n and e are the numbers of vertices and edges respectively.

(C) Give a polynomial time algorithm to compute the boundary of the union of n circular disks where the center of every disk lies on a common line (see Figure 1 for an example). Give and explain the worst-case time complexity of your algorithm (in terms of n). In order to get full credit, your algorithm has to run in optimal time.

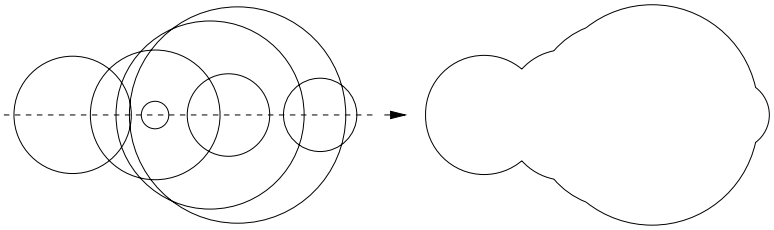

Figure 1: Computing the boundary of the union of n disks.

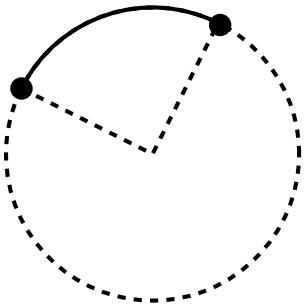

Figure 2: An example of an arc.

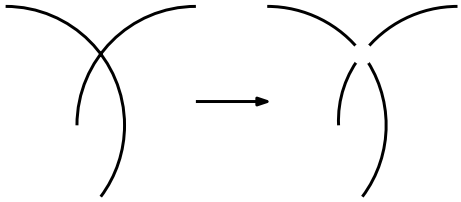

Figure 3: The function *intersect* breaks two arcs into smaller pieces according to the intersection.

Here are some facts and hints that you can use:

- A disk, d, with center z and radius r is $d = \{x \in \mathbb{R}^2 \mid \|z - x\| \leq r\}$.
- The union of n disks, $d_1, d_2, ..., d_n$ is just $\bigcup_{i=1}^{n} d_i$.
- You can assume a data type called *arc*. An arc is any connected portion (one component) of the circumference of a circle. A simple example of its implementation contains the center and radius of a circle, and the starting and ending angles of the arc, see Figure 2.
- Every circle is initially given by being decomposed into 4 arcs, namely, cutting the circle by vertical and horizontal lines that pass through the center.
- You can assume a function *intersect()* that takes two arcs as inputs, and outputs the intersections and breaks down the arcs into smaller arcs, see Figure 3.
- The boundary of the union can be decomposed by vertical slabs such that each slab has the arcs belonging to only one disk. (see Figure 4)

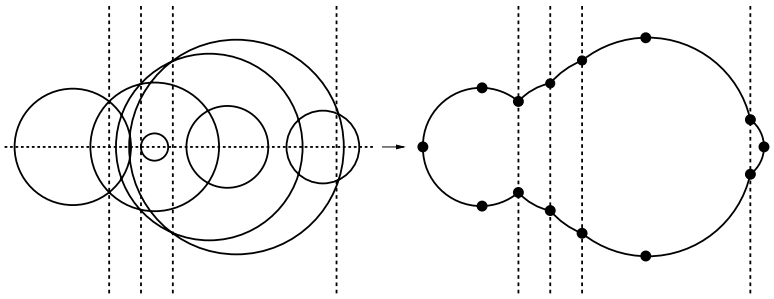

Figure 4: Each vertical slab has only the arcs of one disk.

(D) How can you modify the algorithm developed in part **(C)** to compute the boundary of the union of n disks if the boundary of every disk passes through a common point (Figure 5)?

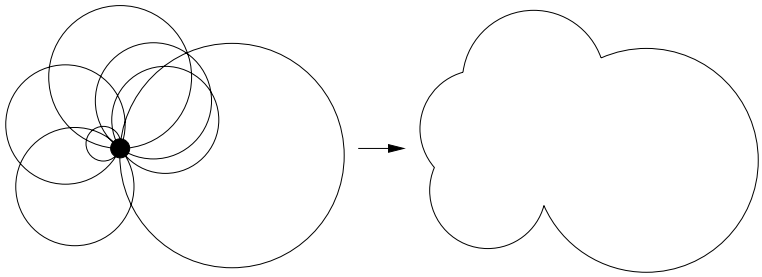

Figure 5: A different case when all the circles pass through a common point.

4. Principles of Programming Languages

An abstract machine is a useful concept in programming languages. The environment E is a map from variable identifiers to store entities. A store entity can be a definite value (e.g., `2`, `"any string"`, `true`), or a partial value (e.g., `tuple(2 X Y)`, `[a b X]`, where `X` and `Y` here are variables).

A semantic statement is a pair of statement and environment. An execution state is a pair of a stack of semantic statements (called also semantic stack) and an assignment store σ that contains entities, that is, $([(< \mathit{statement} >, E)], \sigma)$. The stack gets updated with every function call.

We say that a procedure/function is doing an iterative computation if and only if the associated semantic stack has constant size (regardless their parameters).

Let us consider the functions `F`, `G` and `H` given below.

```
fun {F X N}
   if N==0 then 1
   else X*{F X N-1}
   end
end
```

```
local
  fun {G1 X N A}
    if N==0 then A
    else {G1 X N-1 X*A}
    end
  end
in fun {G X N} {G1 X N 1} end end
```

```
local
  fun {H1 X N A}
    if N==0 then A
    else
      if (N mod 2) == 0 then
           {H1 X*X (N div 2) A}
      else {H1 X    N-1      X*A}
      end
    end
  end
in fun {H X N} {H1 X N 1} end end
```

Note that `N mod 2` above returns `0` if and only `N` is divisible by `2`, and `N div 2` is the quotient from dividing `N` by `2`. The `local..in` statements above are used to define `G1` and `H1` as private/local functions for `G` and `H`, respectively.

(A) Does function `F` perform an iterative computation? Explain your answer.

(B) Prove that `F` is equivalent to `G`. Does function `G` perform an iterative computation? Explain.

(C) Prove that `G` is equivalent to `H`. Compare function `G` and `H` in terms of time complexity.