NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
EXAMINATION FOR
Semester 1, 2009/2010

CS 5201 - FOUNDATION IN THEORETICAL CS

Nov/Dec 2009 Time Allowed: 3 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **four**(**4**) long questions and comprises **seven** (**7**) 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.

Algorithm (3 + 4 + 3 = 10 marks)

Tidiness and organisation of your answers count! Namely, we will deduct your marks even your answer is correct if it is not readable or not comprehensible.

1. Given a graph $G = (V, E)$, and a graph $G' = (V', E')$ as an induced subgraph of G if V' is a subset of V and E' is a subset of E. Given an input of an undirected graph G with n vertices and m edges, and with an integer y, find the maximum induced subgraph H of G such that the degree of each vertex in H is greater than or equal to y. You algorithm should tell if such a subgraph H exists, if so, compute H as an output in $O(m + n)$ time and memory usage. Describe the data structure you used and justify the correctness and time complexity.

2. There are n amoebas lining up in a horizontal line next to each other. Each of them has a weight of w_i, $i = 1$ to n. There is a big evil virus coming and these amoebas have to be united to fight against the virus. The goal is to merge every amoeba into one big amoeba. Each time, every two neighboring amoebas (an amoeba cannot merge with another which is not immediately next to it, and they will not swap their orders) can perform a fusion and become a new amoeba with its weight equal to the sum of the two weights. During each fusion, the two will produce an amount of heat same as their combined weight.

   For example, three amoebas with weights 3, 2 and 1. They can perform fusions by two different ways. The first way: the first amoeba merges with the second one, becoming an amoeba with weight 5 and releases 5 units of energy. Then this new amoeba performs a final fusion with the third one and becomes a final amoeba with weight 6 and release 6 units of energy. In total, the whole process releases 11 units of energy. However, the second way is to merge the second and third amoebas first and it ends up with only producing 9 units of energy in total.

   However, they cannot produce too much energy. Otherwise, the evil virus will notice and destroy them before they all merge into one. So given a number k, and the all the weights of the n amoebas $\{w_i \mid i = 1..n\}$, can you decide if there is a way to perform all the fusions and release a total energy which is less than or equal to k?

   Is this an NP-complete problem? If so, prove it with the given list of NPC problems below. Otherwise, give a polynomial time algorithm and give it's time complexity and a brief justification for that.

3. Description of the **LONGEST CIRCUIT PROBLEM** (LCP): Given a graph $G = (V, E)$, a length function $l(e)$ for each $e \in E$ and a number k. Is there a simple cycle c in G such that the sum of the lengths of all edges in c is greater than or equal to k?

   Prove that LCP is NP-complete.

   List of NP-complete problems:(All the problem belows are proven to be NP-complete)

   3-Satisfiability (3-SAT): Given a set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$. Is there a satisfying truth assignment for C?

   Subgraph Isomorphism problem (SIP): Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, does G_1 contains a subgraph isomorphic to G_2, that is, two subsets $V' \subseteq V_1$ and $E' \subseteq E_2$ such that $|V'| = |V_2|, |E'| = |E_2$, and $\exists$ a one-to-one function $f : V_2 \to V'$ satisfying $\{u, v\} \in E_2$ iff $\{f(u), f(v)\} \in E'$?

   Hamiltonian circuit (HC): Given a graph $G = (V, E)$, does G contains a Hamiltonian circuit?

   Vertex Cover (VC): Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is there a vertex cover of size k or less for G, i.e., a subset $V' \subseteq V$ with $|V'| \leq k$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to V'?

   Travelling Salesman Problem (TSP): Given a complete graph K_n with n vertices and the cost for each edge. Does there exist a simple cycle that traverses every vertex with a total cost less than or equal to a number k?

B: Theory of Computation (10 marks)

(A) Let the alphabet Σ be $\{0, 1, 2\}$.

1. Determine the largest number n such that there is a non-deterministic finite automaton A with exactly 4 states such that for exactly n words $\sigma \in \Sigma^*$ there is an accepting run of A on σ; note that the automaton A is not required to have a rejecting run on words it rejects, it just might have no run at all on such words.

2. Give the automaton for the number n chosen.

3. Explain in few sentences why there is no automaton with exactly 4 states accepting exactly m words where $n < m < \infty$.

(B) Consider grammars over the terminal alphabet $\{0, 1, 2\}$, the non-terminal alphabet $\{S, A, B, C\}$ with start symbol S and a set Π of production rules where every production rule in Π is of the form $X \to YZ$ with $X \in \{S, A, B, C\}$ and $Y, Z \in \{S, A, B, C, 0, 1, 2\}$.

1. Find a set Π of production rules such that the resulting grammar generates exactly the language $\{10011001\}$.

2. Is every language generated by a grammar of this form context-free?

3. Give a context-free language not generated by a grammar of this form.

(C) 1. Let f be a total function from the natural numbers to the natural numbers and assume that $\{(x, y) : f(x) \neq y\}$ is recursively enumerable. Can it be concluded that f is recursive? ("YES, f is always recursive" versus "NO, f might be nonrecursive")

2. Prove your answer of Question (C) 1.

3. Let $A_0, A_1, A_2, \ldots$ be a uniformly recursively enumerable sequence of sets, that is, the set $\{(e, x) : x \in A_e\}$ is recursively enumerable. Can it be concluded that $B = A_0 \cap A_1 \cap A_2 \cap \ldots$ is also recursively enumerable? ("YES, B is always recursively enumerable" versus "NO, $A_0, A_1, A_2, \ldots$ can be chosen such that B is not recursively enumerable")

4. Prove your answer of Question (C) 3.

C: Principles of Programming Languages (37 marks)

Consider the following programming language SeqLa ("Sequential Language"), written in Backus-Naur Form:

$$\begin{array}{lcl} E & ::= & n \mid x \mid (E + E) \mid (E - E) \mid (E * E) \\ B & ::= & (!B) \mid (E < E) \\ C & ::= & \texttt{int}\ x \mid \texttt{read}\ x \mid \texttt{print}\ E \mid x = E \mid \\ & & \texttt{if}\ B\ \{P\}\ \texttt{else}\ \{P\} \mid \texttt{while}\ B\ \{P\} \\ P & ::= & \epsilon \mid C; P \end{array}$$

SeqLa programs, denoted by the non-terminal symbol P in the grammar above, consist of sequences of commands, denoted by the non-terminal symbol C. The symbol n denotes non-positive integers such as 4711, and the symbol x denotes variable names that start with a lower-case letter. The symbol ϵ denotes an empty program.

As an example, consider the following example SeqLa program:

```
int n;
int limit;
n = 0;
read limit;
while (n < limit) {
   n = (n + 1);
   print n;
};
```

This program declares first an integer variable `n` whose scope is the rest of the sequence (including the while-loop). The variable `n` is assigned the value `0` first. Then the program reads an integer value from a user input device and assigns the variable `limit` to the value. In the while-loop, the variable `n` is incremented and printed. If the user supplies the value 5, the program prints:

```
1 2 3 4 5
```

1. (3 marks) Implement the factorial function in SeqLa. Your program should read a positive integer n and print $n!$ as result.

2. (6 marks) In order to find out which variables are used in a given arithmetic expression, we can define a function *usedVariables* as follows:

$$\begin{array}{rcl} \mathit{usedVariables}(n) & = & \emptyset \\ \mathit{usedVariables}(x) & = & \{x\} \\ \mathit{usedVariables}(E_1 + E_2) & = & \mathit{usedVariables}(E_1) \cup \mathit{usedVariables}(E_1) \\ \mathit{usedVariables}(E_1 - E_2) & = & \mathit{usedVariables}(E_1) \cup \mathit{usedVariables}(E_1) \\ \mathit{usedVariables}(E_1 * E_2) & = & \mathit{usedVariables}(E_1) \cup \mathit{usedVariables}(E_1) \end{array}$$

   Now, we are interested in all variables that are *declared* in a given SeqLa *program*. Define the function *declaredVariables* that computes the set of all variables that are declared using `int` in a given SeqLa program P. You may use a helper function $\mathit{declaredVariables}_C$ to find the declared variables of a command.

$$
\begin{aligned}
\mathit{declaredVariables}(\epsilon) &= \ldots \\
\mathit{declaredVariables}(C; P) &= \ldots \\
\mathit{declaredVariables}_C(\texttt{int } x) &= \ldots \\
\mathit{declaredVariables}_C(\texttt{read } x) &= \ldots \\
\mathit{declaredVariables}_C(\texttt{print } E) &= \ldots \\
\mathit{declaredVariables}_C(\texttt{if } B \ \{P_1\} \ \texttt{else} \ \{P_2\}) &= \ldots \\
\mathit{declaredVariables}_C(\texttt{while } B \ \{P\}) &= \ldots
\end{aligned}
$$

3. (10 marks) In SeqLa, the scope of an integer variable declaration is the rest of the sequence, in which the declaration appears (including possibly nested conditionals and while-loops). In order to define a type checker that makes sure that every variable that is used in a given program has been declared, we need to keep track of declared variables as we examine a given program. For this, we use the predicate *well-typed*(P, V), which holds if all variables used but not declared in P are included in V.

   For example, consider the following program:

   ```
   int x;
   x = 1;
   if (x < 2) { int y; y = 3; print (x + y); } else { };
   ```

   During program analysis, we will check the formula

   $$\textit{well-typed}(\ \texttt{int y; y = 3; print (x + y);}, \ \{\texttt{x}\})$$

   to verify that the variables that are used, but not declared in the program `int y; y = 3; print (x + y);` are included in {x}.

   In order to check that a given program P is well-typed, we check whether *well-typed*$(P, \emptyset)$ holds.

   Define the predicate *well-typed*.

4. (5 marks) Extend the Backus-Naur Form of SeqLa by integer arrays, resulting in a language that we shall call ASeqLa. In ASeqLa, you should be able to declare arrays whose size is computed by the program, and assign and access array fields, as shown in the following example:

   ```
   int i;
   x = 1;
   array A[(i + 4)];
   while (i < 6) { A[i-1] = i; i = (i+1); }
   while (! (1 < i)) { print (i + A[i-2]); i = (i-1);}
   ```

   Note that array variables are written using upper-case letters; in your Backus-Naur Form of SeqLa, use the non-terminal X for these array variables. The example above declares an array `A` of size 5, whose indices range from 0 to 4.

   In your Backus-Naur Form, *do not* allow assignment of array variables themselves, as in

   ```
   A = ...
   ```

5. (4 marks) Consider the language ASeqLa from Question 4. It is possible to write meaningless programs, which access array fields that do not exist. For example,

```
array C[10];
print C[10];
```

Here, the indices for `C` only range from 0 to 9, so `print C[10]` leads to a runtime error. Is it possible to implement an array bounds checker in a compiler for ASeqLa, which accepts exactly those programs whose array accesses are within the declared bounds? Briefly explain your answer.

6. (5 marks) Consider an implementation of the language ASeqLa from Question 4 using a compiler to the x86 machine code. In order to assign a memory location to a given variable (integer or array) at runtime, which one of the following possibilities is the best:

   - static allocation,
   - stack allocation or
   - heap allocation?

   Briefly explain your answer.

7. (4 marks) In ASeqLa from Question 4, we have excluded the possibility of assigning one array to another one, as in the following program.

   ```
   array A[10];
   array B[10];
   A = B;
   ```

   For two arrays `A` and `B` of sizes s_a and s_b, respectively, there are (at least) two different ways to give meaning to the assignment `A = B`:

   Copy semantics. Copy the first $min(s_a, s_b)$ fields of the array `B` to the array `A`.

   Reference semantics. Make the array variable `A` refer to the same array as the array variable `B`.

   - Give an example, where both versions can be executed without error, but the printed results are different.
   - Give an example, where copy semantics leads to an array bound error, but reference semantics leads to execution without error.

D: Logic and AI (25 marks)

PART A (15 MARKS)

Given the following short passage:

A person needs a PhD and prior teaching experience to teach in NUS. To get a PhD, the person must first pass his qualifying exams (QE), and after the QE, he has to pass his thesis exam. In addition, there are some who teach in NUS who are married to someone who teaches in another university.

Question 1. (5 MARKS)

Rewrite the passage in first-order logic including quantifiers.

Question 2. (4 MARKS)

Rewrite your answer in Question 1 in CNF.

Use resolution to answer the following two questions. In each case show any additional rules or facts that you have inserted.

Question 3. (3 MARKS)

Show that since Ted teaches in NUS, it is not possible that he failed his QE.

Question 4. (3 MARKS)

Show that it is possible that Bob, who teaches in NUS, is married to Alice who is teaching in another university NTU.

PART B (10 MARKS)

Question 1 (3 MARKS)

a. What are Horn clauses? Show an example of a Horn clause. (1 MARK)
b. Explain why Horn clauses important in forward chaining. (2 MARKS)

Question 2 (2 MARKS)

What do you understand by the terms "sound" and "complete"?

Question 3 (5 MARKS)

Show that, given a set of first-order-logic clauses containing universal and existential quantifiers, that the forward-chaining algorithm is both sound and complete. You may assume for this question that the clauses do not contain function symbols.