NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2007/2008

# CS 5201 - FOUNDATION IN THEORETICAL CS

April 2008 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. Please write your Matriculation number in **ALL** the answer books.

**1. Theory of Computation ((1+2+3)+ (2+2)) = 10 marks**

**(A)** Consider the alphabet $\Sigma = \{0, 1\}$. We say that a language $L \subseteq \Sigma^*$ *is-blind-to* two strings $x$ and $y$ (where $x \in \Sigma^*$ and $y \in \Sigma^*$) if and only if

$$\forall z \in \Sigma^* \; xz \in L \Leftrightarrow yz \in L$$

Thus, a language $L \subseteq \Sigma^*$ defines an is-blind-to relation $\equiv_L$ where we write $x \equiv_L y$ if and only if $L$ *is-blind-to* strings $x, y$.

(i) Show that $\equiv_L$ is an equivalence relation (reflexive, symmetric, and transitive).

(ii) If the $\equiv_L$ relation of a language $L$ has finitely many equivalence classes, which of the following claims are true? Explain your answer.

- $L$ must be regular.
- $L$ must be non-regular.
- $L$ may be regular or non-regular.

(iii) Consider the language $L1 = \{x \mid x \in \Sigma^* \text{ and } x \text{ ends with } 01\}$ where $\Sigma = \{0, 1\}$. How many equivalence classes does the $\equiv_{L1}$ relation of language $L1$ contain?

**(B)** A regular expression captures a collection of strings over an alphabet. For example $(0+1+2+3)^*$ is a regular expression over the alphabet $\{0, 1, 2, 3\}$. However, we can also interpret the representation of a regular expression as a string over an enriched alphabet. For example, $(0 + 1 + 2 + 3)^*$ can be seen as a string over the alphabet $\{0, 1, 2, 3, (, ), +, ^*\}$.

(i) Write a context-free grammar which accepts all strings representing regular expressions over the alphabet $\Sigma = \{0, 1, 2, 3\}$. Note that $\Sigma$ does not contain any of the symbols conventionally used for regular expression construction such as:

- $+$ denoting union of regular languages,
- $\cdot$ denoting concatenation of regular languages,
- $^*$ denoting the Kleene star operation,
- (, ) denoting open/closed paranthenses,
- $\epsilon$ denoting the empty string.

(ii) Since any regular language is also context-free, context-free grammars can be used to describe regular languages. Is the language accepted by your constructed context-free grammar a regular language as well? Give justifications for your answer.

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

**(A)** Recall that a formula is in disjunctive normal form if and only if it is a disjunction over conjunctive terms of literals. For example, $(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ is in disjunctive normal form.

(i) Write the formula $(x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3)$ in disjunctive normal form.

(ii) Consider a formula $\phi$ using the variables $x_1, x_2, x_3, x_4, x_5, x_6, x_7$. The formula $\phi(x_1, x_2, \ldots, x_7)$ is true if and only if exactly 2 or exactly 4 of these variables are true. How many conjunctive terms does $\phi$ have, when written in disjunctive normal form (without any redundant terms)?

(iii) Give a polynomial-time algorithm which takes in a formula $\phi$ in disjunctive normal form and produces the conjunctive normal form representation of $\neg\phi$.

**(B)** Check the satisfiability of the following systems of clauses using resolution.

(i) $x_1 \vee x_2 \vee x_3$, $\neg x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_3$, $\neg x_2 \vee \neg x_3$.

(ii) $x_1 \vee x_2$, $\neg x_1 \vee \neg x_3$, $\neg x_2 \vee \neg x_3$, $x_3$.

**(C)** Determine which two of the following three formulas can be unified and write down the unifier. Here $u, v, w, x, y, z$ are variables, $p$ is a predicate, $f, g$ are functions and $0, 1, 2$ are constants.

- $p(1, f(f(g(0,1),0),v),w)$
- $p(1, g(0,1), u)$
- $p(1, f(x, g(0,y)), z)$

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

For any matrix $A$, we denote $r(A)$ and $c(A)$ be its number of rows and columns, respectively. Consider two matrices $A$ and $B$. Let $\texttt{multiply}(A, B)$ be a procedure to compute $A \times B$. Note that the multiplication is allowed if $c(A) = r(B)$. The running cost of $\texttt{multiply}(A, B)$ is $r(A) * c(A) * c(B)$.

**(A)** The following pseudocodes compute $A_1 \times A_2 \times \ldots \times A_k$. Suppose $r(A_i) = i$ and $c(A_i) = i + 1$ for $i = 1, 2, \ldots, k$. Which of the following pseudocodes is more efficient? Explain your answer.

(i)

```
$X = A_1$;
For $i = 2$ to $k$
    $X = \texttt{multiply}(X, A_i)$;
return $X$;
```

(ii)

```
$X = A_k$;
For $i = k - 1$ downto 1
    $X = \texttt{multiply}(A_i, X)$;
return $X$;
```

**(B)** Given a set of matrices $S = \{A_1, A_2, \ldots, A_k\}$. We say $S$ is valid if we can form a multiplication formula $A_{j_1} \times A_{j_2} \times \ldots \times A_{j_k}$ such that $\{j_1, \ldots, j_k\} = \{1, 2, \ldots, k\}$ and $c(A_{j_i}) = r(A_{j_{i+1}})$ for $i = 1, 2, \ldots, k - 1$. Can you give an efficient algorithm to check if $S$ is valid? What is the running time of your algorithm?

**(C)** Let $f[1], \ldots, f[n]$ be a set of user-defined integers. Consider the recursive equation $V(i, j)$ with $1 \le i \le j \le n$ such that $V(i, i) = f[i]$ and

$$V(i,j) = \max \begin{cases} V(i, \lfloor \frac{i+j}{2} \rfloor) + f[i] \\ V(\lfloor \frac{i+j}{2} \rfloor + 1, j) + f[j] \\ f[i] + f[j] \end{cases}$$

Suppose we want to compute $V(1, n)$. Which of the following approaches will lead to a more efficient algorithm?

- Dynamic programming.
- Encoding of the above recursive equation as a recursive procedure.

Explain your answer in details.

**4. Principles of Programming Languages (1+1+4+4) = 10 marks**

Lazy evaluation is a feature of several modern programming languages that allows expressions to be computed in a *demand-driven* fashion. Lazy evaluation also allows the representation of infinite data structures, such as *streams*, which are also known as *lazy lists.* A stream is a possibly infinite sequence, represented by a pair betwen two expressions: the *head*, and the *tail.* The expression representing the tail is usually recursive and may loop infinitely on an eager evaluation platform, such as Java or C, where expressions are evaluated as soon as they are bound to a variable. However, in a lazy evaluation setting, stream elements will be produced only when there is demand for them; thus, as long as the demand is finite, the computation will be finite too.

In a language such as Haskell, we may specify the stream of all natural numbers starting from 1 using the following expression:

```
naturals = 1:(map (1+) naturals)
```

Here, the colon operator represents the list constructor. The expression `(1+)` represents the increment operator (an alternative Haskell syntax would be `\ x->x+1`), and `map` is the higher-order function that maps an operator to each element of a list, returning the list of results. An alternative, and possibly more familiar, definition of the stream of natural numbers might be:

```
naturalsFrom n = n:(naturalsFrom (n+1)) naturals = naturalsFrom 1
```

Now the expression

```
take 10 naturals
```

will evaluate to the list `[1,2,3,4,5,6,7,8,9,10]`. Since this expression demands a finite number of elements in the stream, the computation does not run into infinite loop.

**(A)** Describe informally what would be the internal representation of `naturals` produced by a compiler/interpreter for a language that allows lazy evaluation.

**(B)** Using one of the two programming styles given above, provide a Haskell definition for the infinite stream containing all the powers of 2 (do not worry about minor syntactic errors).

**(C)** Devise a systematic translation scheme for lazy stream expressions into a language like Java. You may describe your translation mechanism informally. Apply your translation mechanism to the definition of `naturalsFrom` and provide the resulting Java code (do not worry about minor syntax errors in your Java code).

**(D)** The following recursive Java method prints a solution to the Towers of Hanoi puzzle.

```
void hanoi ( int nDisks, int source, int destination, int aux ) {
    if ( nDisks > 0 ) {
        hanoi ( nDisks-1, source, aux, destination ) ;
        System.out.println ( "Move a disk from " + source +
                             " to " + destination + "." ) ;
        hanoi ( nDisks-1, aux, destination, source ) ;
    }
}
```

For instance, the call `hanoi(3,1,2,3)` prints out a sequence of 7 disk moves that would take a pile of 3 disks from pole 1 to pole 2, using pole 3 as an auxilliary.

Write a non-recursive version of this method.

END OF PAPER