NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
EXAMINATION FOR
Semester 2, 2009/2010

# CS 5201 - FOUNDATION IN THEORETICAL CS

Nov/Dec 2010 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.

## A: Algorithms ($2 + 8 + 10 = 20$ points)

Note: You need to write your answer in a clean and organized way. Marks will not be given to answers that are not readable or not comprehensible. It is your job to show that your answer is correct. Otherwise no marks will be given **even if the examiner cannot show that your answer is wrong and cannot construct a counter-example. Namely, the burden of proof is on your side.** For example, if you claim P=NP without a proof and use that claim in your answer, no mark will be given even though the examiner cannot prove that your claim is wrong.

**Problem 1** You are given $n$ intervals $[x[i], y[i]]$ on the real line, where $x[i]$ and $y[i]$ are both real numbers and $x[i] \leq y[i]$ for $i$ from 1 through $n$. We say that a set $S$ of points on the real line *splits* all these intervals if for all $i$, there is at least one point $z$ in $S$ such that $z$ falls within the interval of $[x[i], y[i]]$ (i.e., $x[i] \leq z \leq y[i]$). Our goal is to find a set $S$ that can split all these intervals and where the size of $S$ is the smallest. We call such a set as a *minimum split*. If there are multiple such sets with the same smallest size (i.e., multiple minimum splits), finding one minimum split suffices.

For example, for the 3 intervals of $[1.2, 2.04]$, $[-1.2, 4.1]$, and $[5.6, 9.1]$, the set $S_1 = \{1.3, 0, 7.0\}$ splits them. Similarly, the set $S_2 = \{1.5, 7.0\}$ also splits them. The size of $S_2$ is the smallest among all sets that can split these 3 intervals, since a set with a single point (i.e., size of 1) can never split the above 3 intervals. Thus $S_2$ is a minimum split. There are other minimum splits (e.g., $\{1.6, 7.31\}$), and we only need to find one minimum split.

- (2 points) Prove that there exists a minimum split $S$ such that for each $z$ in $S$, $z = y[i]$ for some $i$ between 1 and $n$.
- (8 points) Design an algorithm with $o(n^2)$ (notice this is small-"o") worst-case time complexity to find a minimum split. You should first describe/illustrate your algorithm idea at a high-level, then write out all the steps in your algorithm, then briefly explain its time complexity, and finally **rigorously prove** that your algorithm is correct.

**Problem 2 (10 points)** You are given $m$ boxes where the $j$th box can accommodate at most $a[j]$ balls. There are $n$ balls to be put into the boxes. Each ball has a single color. There are total $k$ different colors among all the balls, and there are $n[i]$ balls with the $i$th color. (Obviously, $n[1] + n[2] + \ldots + n[k] = n$.) You may place any ball into any box except that a box is not allowed to have multiple balls of the same color and a box's capacity (i.e., $a[j]$) cannot be exceeded.

For example, assume that we have three boxes with $a[1] = 2$, $a[2] = 1$, and $a[3] = 2$. Assume that we have 1 red ball, 1 blue ball, and 2 green balls. One possible placement will be: Box 1 has a red ball, box 2 has a green ball, and box 3 has a blue ball and a green ball.

Design an algorithm to find a way to place all the balls into the boxes under the previous constraints, in the general case (i.e., under arbitrary input). If such a placement does not exist, your algorithm should output that no such placement exists. Your algorithm should have polynomial worst-case time complexity and space complexity, with respect to both $m$ and $n$. You should first describe/illustrate your algorithm idea at a high-level, then write out all the steps in your algorithm, then briefly explain its time and space complexity, and finally **rigorously prove** that your algorithm is correct.

## B: Theory of Computation

All questions below carry equal marks.

Q1. Let $\#_a(w)$ denote the number of $a$'s in the string $w$. Similarly, let $\#_b(w)$ denote the number of $b$'s in the string $w$. Give a DFA (deterministic finite automata) for the following language $L$ over the alphabet $\Sigma = \{a, b\}$.

$L = \{w \in \Sigma^* : \text{for each prefix of } u \text{ of } w,\ \#_a(u) - \#_b(u) \leq 3 \text{ and } \#_b(u) - \#_a(u) \leq 3\}$.

Q2. Give a CFG (context free grammar) for the language:

$L = \{a^i b^j : 3i \leq j \leq 5i\}$.

Q3. Let $w^R$ denote the reverse of string $w$. For example, $(abaab)^R = baaba$.

Show that $\{ww^Rw : w \in \{a, b\}^*\}$ is not context free.

Q4. For any language $A$, let $A^*$ denote the language $\{w_1 w_2 \dots w_n : w_1, w_2, \dots, w_n \in A\}$. Here $n$ can be 0.

Prove or disprove:

Suppose $L^*$ is a recursive language over the alphabet $\Sigma = \{a, b\}$. Then $L$ is a recursive language.

## C: Principles of Programming Languages

Consider a language called *Raster* for describing pages to be sent to a black-on-white raster printer. The printer uses a bitmap format, consisting of a two-dimensional $150 \times 85$ array called *outputPage* of boolean values.

An array value $outputPage[x][y]$ is true iff the corresponding pixel at position $(x, y)$ should be black.

Programs in the language Raster consist of possibly empty sequences of commands to print black points, black horizontal lines and black rectangles on the initially white page. Commands are separated by semicolon symbols. An example program would be:

```
point(0,4); line(2,3,5); rectangle(4,0,5,6)
```

The upper left corner of the corresponding *outputPage* looks as follows:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | • | • | | | | |
| 1 | | | | | • | • | | | | |
| 2 | | | | | • | • | | | | |
| 3 | | | • | • | • | • | • | | | |
| 4 | • | | | | • | • | | | | |
| 5 | | | | | • | • | | | | |
| 6 | | | | | • | • | | | | |
| 7 | | | | | | | | | | |
| | | | | | | | | | | |

Note that the `point` command gives the $(x, y)$ coordinate of a black point. The `line` command gives the $(x, y)$ coordinate of the starting point of the line, followed by the number of points in the line. The `rectangle` command gives the $(x, y)$ coordinates of the upper left corner, followed by the $(x, y)$ coordinates of the lower right corner. The following restrictions apply:

- All $(x, y)$ coordinates must lie within the size limits of *outputPage*.
- No lines or rectangles must extend beyond the size limits of *outputPage*.
- The second point (lower right corner) of a rectangle must not lie above or to the left of the first point (upper right corner).

Also note that several commands may affect the same pixel position. In this case, the pixel remains black, once one command makes it black.

1. (5 marks) Describe the language Raster in BNF notation. You may use a non-terminal symbol $n$ to denote integers.

2. (10 marks) The informal description of language Raster above mentions restrictions on the integers to be used in points, lines and rectangles. Formalize these restrictions by describing a typing function

   $$\mathit{well\text{-}typed} : \mathit{Raster} \rightarrow \mathit{boolean}$$

   which returns *true* iff the given program complies with the restrictions. Use a notation in your description that can serve as a basis for a computer implementation of the typing function.

3. (15 marks) Assume from now on, that the restrictions are all met, and therefore the given programs are well-typed.

   One approach to formally describe the semantics of Raster as a function is to see a program as a transformation function from a given page to a new page in which the commands have been executed:

   $$\mathit{interprete} : \mathit{Raster} \times \mathit{boolean}[150][85] \rightarrow \mathit{boolean}[150][85]$$

where *boolean*[150][85] are two-dimensional boolean arrays of pixels described above. Thus, in order to obtain the page described by a given program $R$, we can apply $interprete(R, white)$, where *white* is a $150 \times 85$ array containing only *false* boolean values.

Define the function *interprete* such that it can serve as a basis for a computer implementation of an interpreter.

**Important** Your function must be purely declarative, in a sense that the arrays are never destructively changed. The only operation on arrays that is allowed is the function

$$blacken : boolean[150][85] \times int \times int \to boolean[150][85]$$

which returns a new array that is the same as the given array, except that the boolean value at the given $(x, y)$ position is *true*, regardless of its value in the given array.

You may freely use recursion and helper functions in your solution. Do not use loops in which assignments to variables occur that denote arrays.

4. (15 marks) Consider an extension of Raster called *Raster++* that allows the definition of integer constants, as in the following example:

```
let a = 10 in
    rectangle(0,0,a,a), line(0,a,50),
    let b = 20 in
        point(b,a)
    end
end
```

To the right of the `=` symbol, only integer constants are allowed. The same constant symbol may be used in nested `let` declarations. In this case, any reference to the symbol refers to the innermost declaration. For example, the program

```
let c = 3 in
    let c = 77 in
        point(0,c)
    end
end
```

is equivalent to the program

```
point(0,77)
```

- Describe Raster++ in BNF.
- Describe a transformation of Raster++ to Raster that preserves the meaning of programs. For this, formally describe a function

  $$transform : Raster{+}{+} \to Raster$$

  Hint: Use environments of the form

  $$env : symbol \to int$$

  and an environment extension function

  $$\cdot[\cdot \leftarrow \cdot] : env \times symbol \times int \to env$$

  such that
  $e[x \leftarrow v](y) = v$ if $y = x$ and
  $e[x \leftarrow v](y) = e(y)$ if $y \neq x$.

## D: Logic and AI ( 3 + (2 + 2) + (2+1) ) = 10 marks

**(A)** We can represent a program as a logical formula. For example the function `f` given below

```
int f(x){ return x; }
```

can captured by the first order logic formula $\forall X eq(f(X), X)$ where $eq$ is the equality predicate and $f$ is a function symbol. Represent the following program as a first order logic formula.

```
int g(x, y, z){
1 if (x > y){
2   if (x > z){
3      return x;
4   else  return z;}
5 else return y;}}
```

**(B)** Given a program, a *program path* is a sequence of program statements (from beginning to the end of the program) which obeys the control flow restrictions of the programming language constructs. Thus, in the program of part (A) we know that $(1, 2, 3)$ is a legal program path, and so is $(1, 2, 4)$. However $(1, 2, 5)$ is not a legal program path because once the branch in line 2 is executed either 3 (if-then part) or line 4 (if-else part) must be executed as per the control flow restrictions of the if-then-else construct.

Program paths can be associated with a first order logic formula free from quantifiers. Let us call this the path formula. For example, the path $(1, 2, 3)$ in the program of part (A) is associated with the formula $x > y \wedge x > z$. In other words, such formula are free from universal quantifiers and any variable appearing in the formula is implicitly existentially quantified. Thus, the path formula for the path $(1, 2, 3)$ in the program of part (A) corresponds to the following first-order logic formula.

$$\exists x \exists y \exists z \;\; x > y \wedge x > z$$

Given a program and a path in the program

- is it possible for the path formula to be unsatisfiable?
- is it possible for the path formula to be valid?

Give detailed justification with example programs instead of simple yes/no answers.

**(C)** Given a program P, and path $\pi$ in the program can you suggest an automated procedure to compute the path formula of $\pi$ in $P$.

Show the working of your algorithm of your algorithm on the path $(1, 2, 3, 4, 5, 6, 7)$ of the following program.

```
1 input x, y, z;
2 if (y > 0){
3    z = y * 2;
4    x = y - 2;
5    x = x - 2;}
6 if (z == x){
7    output("How did I get here");
}
```