

NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
EXAMINATION FOR
Semester 2, 2009/2010
CS 5201 - FOUNDATION IN THEORETICAL CS

April/May 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.

Algorithm ((6+2+6+6) = 20 marks)

3. Algorithms (6+2+6+6) = 20 marks

(A) Jack Straws: Design an Algorithm

(6 marks) In the game of *Jack Straws*, a number of plastic or wooden “straws” are dumped on the table and players try to remove them one-by-one without disturbing the other straws. In the question here, we are concerned with the following, related problem: First n straws are dumped on the table. Next, for every pair of straws, we want to determine if the straws are connected by a path of touching straws. In other words, given a list of the endpoints for $n > 1$ straws (as if they were dumped on a large piece of graph paper), determine *all* the pairs of straws that are connected. Note that touching is connecting, but also two straws can be connected indirectly via other connected straws. Give an algorithm to compute all the pairs of straws that are connected.

Here is some additional information that you can use:

- i) As a basic step you need to determine whether two straws directly touch (intersect), i.e., whether straw \overline{ab} with end points a and b intersects with straw \overline{cd} (with end points c and d).
- ii) To get full credit also explain how the elementary step i) can be done. If you cannot find an algorithm for this step, assume an $O(n^2)$ algorithm and explain how to find all pairs of straws that are connected.

(B) Jack Straws: Algorithm Complexity

(2 marks) Analyse and explain the worst-case computational complexity of your algorithm under (A) in terms of n .

(C) Weighing Problem

(6 marks) Find an optimal set of n weights w_1, w_2, \dots, w_n so that it would be possible to weigh on a balance scale any *integer load* in the largest possible range from 1 to W , provided weights can be put only on the free cup of the scale. For example, for $n = 1$, we have to use $w_1 = 1$ to balance load 1. For $n = 2$, we simply add $w_2 = 2$ to balance the first previously unattainable load of 2. The weights $\{1, 2\}$ can balance every integral load up to their sum 3.

- Explain what the optimal set of weights is, given n . For example, for $n = 2$ the optimal weights are $w_1 = 1$ and $w_2 = 2$.
- Given n , what is the upper bound W on the load range that can be measured, i.e., $1, \dots, W$? Express W in terms of n . Give an argument why the optimal weights cover every integer load between 1 and W .

(D) Open Intervals

(6 marks) You have a list of n open intervals $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ on the real line. (An open interval (a, b) comprises all the points strictly between its endpoints a and b , i.e., $(a, b) = \{x | a < x < b\}$.) Find the maximal number of these intervals that have a common point. For example, for the intervals $(1, 4), (0, 3), (-1.5, 2), (3.6, 5)$, this maximal number is 3. Design an algorithm for this problem with a better than quadratic time efficiency.

B: Theory of Computation (10 Marks: 4+3+3)

- (A) 1. Is there an infinite context-free language A such that there is no infinite regular language R with $R \subseteq A$? Answer with “Yes, there is such a language A ” or “No, there is no such language A .”
2. Prove your answer to part 1.
3. Is there an infinite context-free language B such that there is no infinite regular language R with $B \cap R$ and $B - R$ both being infinite? Answer with “Yes, there is such a language B ” or “No, there is no such language B .”
4. Prove your answer to part 3.
- (B) Construct a deterministic finite automaton which accepts exactly $\frac{n(n-1)(n-2)}{6} + \frac{n(n-1)}{2} + 1$ many members of $\{0, 1\}^n$ for every n .
- (C) Let A, B be subsets of the natural numbers. Define $A \leq_f B$ iff there is a recursive function f such that (a) for every y there are only finitely many x with $f(x) = y$ and (b) for every x it holds that $x \in A \Leftrightarrow f(x) \in B$. Furthermore, $A \equiv_f B$ iff $A \leq_f B \wedge B \leq_f A$.
- Determine the unique n such that there are n sets A_1, A_2, \dots, A_n such that every recursive set B satisfies $B \equiv_f A_m$ for exactly one $m \in \{1, 2, \dots, n\}$:
1. State the value for n explicitly;
 2. Give the sets A_1, A_2, \dots, A_n ;
 3. Prove that every recursive set B satisfies $B \equiv_f A_m$ for exactly one $m \in \{1, 2, \dots, n\}$.

C: Principles of Programming Languages (10 marks)

(Q 1) (a). Consider a Prolog-like logic program of the form:

$$\begin{aligned} \text{append}([], YS, ZS) & \quad :- \quad YS = ZS \\ \text{append}([X|XS], YS, [X|ZS]) & \quad :- \quad \text{append}(XS, YS, ZS) \end{aligned}$$

Note that $[X|XS]$ denotes a list pattern where the head of list is X , while the tail of list is XS . One possible invocation of this method is the following predicate call, $\text{append}(XS, YS, [1, 2, 3])$, which would return pairs of lists of the form (XS, YS) which when catenated yield the input list $[1, 2, 3]$. That is, we would expect the above predicate call to return a set of four solutions for (XS, YS) , namely :

$$\{([], [1, 2, 3]), ([1], [2, 3]), ([1, 2], [3]), ([1, 2, 3], [])\}$$

Write an equivalent method in functional-type or procedural-style that would return all ways of splitting a list. Such a function may have the following type signature below. Clearly define all methods that you would decide to use. [2 marks]

$$\text{split} :: [t] \rightarrow [[t], [t]]$$

(b). Briefly discuss the execution mechanism used by logic programming, and how it compares with the implementation techniques used in other programming languages, such as procedural languages. [2 marks]

(Q 2) (a). Briefly discuss the role of types in domain-specific languages. How would it help or hinder the development of languages that are tailored to some specific application domains? [1 mark]

(b). Let us consider a polymorphic higher-order language, like ML or Haskell. A simple example of such a function is the identity function:

$$\text{id } x = x$$

Based on this definition, we can infer the following polymorphic type :

$$\text{id} :: \forall t. t \rightarrow t$$

This type states that the id method would take as input value of any type t , and return as result a corresponding value of the same t type. Provide polymorphic types (possible of higher ranks) for the following method definitions, where possible. Note that (e_1, e_2, e_3) denotes a tuple of three items, while $(f e_1 e_2)$ denotes a function f applied to two arguments, e_1 and e_2 . [3 marks]

(i) $h_1 f g x = g (f x)$

(ii) $h_2 f x y = (f x, f y)$

(iii) $h_3 f = x = (f 3, f "h", f x)$

(iv) $h_4 f g z [] = z$
 $h_4 f g z (x : xs) = \text{if } (f x) \text{ then } g x (h_4 xs)$
 $\quad \quad \quad \text{else } (h_4 g z (x : xs) xs)$

(v) $h_5 p s n x = \text{if } (p x) \text{ then } s x (h_5 p s (n x))$
 $\quad \quad \quad \text{else } x$

(Q 3). "Loops may be viewed as a special-case of recursion". Discuss the merit of this claim. Explain why this is possible (or otherwise), and elaborate on language features that are required to support this view. Show a convincing example of this claim, or counter-claim. [2 marks]

D: Logic and AI (26 marks)

1. (General; 7 marks) In logic, we often use *models* to give meaning to formulas. A formula is usually considered *valid* iff it holds in every model.

We sometimes define proof procedures P that can be applied to a formula F . The application $P(F)$ can result in the following answers:

- True: this means that the formula is considered *proven*.
 - False: this means that the formula is considered *disproven*.
 - Don't know: the proof procedure P is not able to prove or disprove F .
- (a) (1 mark) A proof procedure is called *sound*, if it returns True only when the formula is valid. Give a trivial proof procedure, that is sound regardless of the definition of validity under consideration.
- (b) (1 mark) A proof procedure is called *complete*, if it returns True whenever the formula is valid. Give a trivial proof procedure, that is complete regardless of the definition of validity under consideration.
- (c) (2 marks) What mathematical structures can be considered as *models* for a formula in propositional logic where only the propositional variables in the set V occur?
- (d) (3 marks) Describe in words a sound and complete proof procedure for propositional logic, with respect to your models from the previous question.
2. (Propositional logic; 3 marks) In propositional logic, is the formula

$$(r \wedge s) \rightarrow (s \vee p)$$

equivalent to the formula

$$(\neg(s \vee p)) \vee r ?$$

Provide a proof for your answer.

3. (Predicate calculus; 5 marks) In proof procedures for predicate calculus, we are allowed to substitute any term for a universally quantified variable. For example, if (in some context of a proof) we know that the formula

$$\forall z.(\exists x, y.(S(x, y) \vee P(z)))$$

holds, we can substitute (replace) z by any term t , and the resulting formula still holds. Describe the problem that arises if t is $f(y)$. How is this problem usually handled? Illustrate your answer with the given example.

4. (Predicate logic; 4 marks) Prove or disprove the validity of the following formula in predicate logic, where S is a binary predicate.

$$(\forall x, y, z.(x = y \vee y = z \vee x = z)) \rightarrow (\forall u, v, w.((S(u, v) \wedge S(v, w)) \rightarrow S(u, w)))$$

5. (Hoare logic; 2 marks) In Hoare logic, we are comparing the “strength” of a formula as follows:

- A formula F is stronger than a formula G , if $F \rightarrow G$, but $\neg(G \rightarrow F)$.
- A formula F is weaker than a formula G , if $G \rightarrow F$, but $\neg(F \rightarrow G)$.

We are often looking for the weakest precondition for a program, and the strongest postcondition.

- (a) (1 mark) Give for any program P and postcondition F , the *strongest* precondition G such that $\{ G \} P \{ F \}$ is a valid Hoare triple.

(b) (1 mark) Give for any program P and precondition F , the *weakest* postcondition G such that $\{ F \} P \{ G \}$ is a valid Hoare triple.

6. (Hoare logic; 2 marks) Give the *weakest* precondition for the program

```
y := 10;  
z := y;  
x := 2 * x
```

and postcondition $x = y \cdot z$, where the variables x , y , and z denote integers. Simplify the formula as much as possible, using integer arithmetics.

7. (Hoare logic; 3 marks) Give the *strongest* postcondition for the program

```
if (y == x * x)  
then y := y - 1  
else x = x * x  
end
```

and precondition $x = 10$, where the variables x and y denote integers. Simplify the formula as much as possible, using integer arithmetics.