NATIONAL UNIVERSITY OF SINGAPORE

CS 4232 – Theory of Computation
(Semester 1: AY 2015/2016)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number. Do not write your name.
2. This assessment paper consists of TEN (10) questions and comprises TWELVE (12) printed pages.
3. Students are required to answer **ALL** questions.
4. Students should answer the questions in the space provided.
5. This is a **CLOSED BOOK** assessment.
6. It is permitted to use calculators, provided that all memory and programs are erased prior to the assessment; no other material or devices are permitted.
7. Every question is worth FIVE (5) marks. The maximum possible marks are 50.

STUDENT NO: ______________________________

This portion is for examiner's use only

Question	Marks	Remarks	Question	Marks	Remarks
Q01:			Q06:		
Q02:			Q07:		
Q03:			Q08:		
Q04:			Q09:		
Q05:			Q10:		
			Total:		

For this question, it is *permitted* that a non-deterministic finite automaton has multiple start states. Construct for the alphabet $\{0,1,2\}$ a non-deterministic finite automaton which recognises

$$L = \{w \in \{0,1,2\}^* : digitsum(w) \leq 4\} \cup (\{0,1,2\}^* \cdot \{11\})$$

with up to eight states. Here $digitsum(w)$ is the sum of the digits occurring in w, so $digitsum(00112211)$ is 8.

Solution. The finite automaton is given by the following table.

state	succ at 0	succ at 1	succ at 2	type
s	s	t	u	start, acc
t	t	u	v	acc
u	u	v	w	acc
v	v	w	–	acc
w	w	–	–	acc
x	x	x, y	x	start, rej
y	–	z	–	rej
z	–	z	–	acc

Let $\Sigma = \{0,1,2\}$ and consider the language L of all words $w \in \Sigma^*$ which contain one symbol from Σ at least three times. So L is given by the regular expression

$$(\Sigma^* \cdot 0 \cdot \Sigma^* \cdot 0 \cdot \Sigma^* \cdot 0 \cdot \Sigma^*) \cup (\Sigma^* \cdot 1 \cdot \Sigma^* \cdot 1 \cdot \Sigma^* \cdot 1 \cdot \Sigma^*) \cup (\Sigma^* \cdot 2 \cdot \Sigma^* \cdot 2 \cdot \Sigma^* \cdot 2 \cdot \Sigma^*)$$

and L can be recognised by a deterministic finite automaton. Determine the number of states which the minimal deterministic finite automaton for L has.

Solution. The number of states is 28. One accepting state. Furthermore, for each $i,j,k \in \{0,1,2\}$, the values i,j,k can be recovered from $L_{0^i1^j2^k}$. The reason is that $L_{0^i1^j2^k} \cap 0^* = 0^{3-i}0^*$, $L_{0^i1^j2^k} \cap 1^* = 1^{3-j}1^*$, $L_{0^i1^j2^k} \cap 2^* = 2^{3-k}2^*$. Thus there are 27 different derivatives which each correspond to a different rejecting state in the deterministic finite automaton. So a deterministic finite automaton needs at least 28 states.

One can also see that when the number of zeroes seen so far i, the number of ones seen so far is j and the number of twos seen so far is k and $i,j,k < 3$ then the derivative is $L_{0^i1^j2^k}$. Furthermore, if one symbol is seen at least three times then the derivative is $\{0,1,2\}^*$. Thus the overall number of derivatives is 28 and the bound 28 from above is exact.

Construct a context-free grammar for the language

$$L = \{0^n 1^n 2^m 3^m : n, m \geq 1\}$$

and give the grammar in Greibach Normal form. In Greibach Normal Form, each left side of a rule is a single non-terminal and each right side of a rule is a terminal followed by some (possibly none) non-terminals. Some special regulations apply for grammars for languages which contain ε.

Solution. The grammar is $(\{S, T, U, V, W\}, \{0, 1, 2, 3\}, P, S)$ where P contains the rules $S \to 0TVU|0VU$, $T \to 0TV|0V$, $U \to 2UW|2W$, $V \to 1$, $W \to 3$. V and W are placeholders for 1 and 3, respectively, occurring at positions in the right side of a rule where non-terminals have to be. The first rule for S is splitting the two parts which produce $\{0^n 1^n : n \geq 1\}$ and $\{2^m 3^m : m \geq 1\}$ which are generated from the non-terminals T and U; as the first 0 of the part for $\{0^n 1^n : n \geq 1\}$ has to be processed by this rule, the rule is not $S \to TU$ but rather $S \to 0TVU|0VU$. The rules for T and U are standard.

Prove that the language

$$L = \{0^n 1^m 2^k : n = 0 \text{ or } m = k\}$$

is not regular. For the proof, use one of the the following versions of pumping lemma: (a) Traditional Pumping Lemma, (b) Block Pumping Lemma, (c) Jaffe's Matching Pumping Lemma. Say which of (a), (b) and (c) is used, explain what this version of the pumping lemma says and use this pumping lemma to prove that L is not regular.

Solution. The Traditional Pumping Lemma does not prove that L is not regular, as L satisfies this pumping lemma. Therefore the choice must be (b) or (c). Both choices can be used to disprove that L is regular. Here the proof is given for choice (b).

The Block Pumping Lemma says that if L is regular then there is a constant k such that given a splitting $u_0 u_1 \ldots u_k u_{k+1}$ of a word in L, there are i, j with $1 \leq i \leq j \leq k$ such that

$$u_0 \ldots u_{i-1}(u_i \ldots u_j)^* u_{j+1} \ldots u_{k+1} \subseteq L.$$

The proof method is to show that L does not satisfy this Block Pumping Lemma for any constant k.

So assume by way of contradiction that k is the constant of the Block Pumping Lemma. Then one considers the word $0^k 1^k 2^k$ split into a way that $u_0 = 0^k$, $u_1, u_2, \ldots, u_k$ are 1 each and $u_{k+1} = 2^k$. If one now considers the word

$$u_0 \ldots u_{i-1} u_{j+1} \ldots u_{k+1}$$

which is created by omitting the pump then it is of the form $0^k 1^h 2^k$ for some $h < k$. This word is not in L, as the number of 0 is greater than 0 and the number of 1 and number of 2 are not equal. Thus L does not satisfy the Block Pumping Lemma for any given constant and so L cannot be regular.

Show that the deterministic context-free language $\{0^n 1^m : m \leq n\}$ can be recognised by a deterministic pushdown automaton which accepts by state but not by a deterministic pushdown automaton which accepts by empty stack.

Solution. A pushdown automaton which accepts by state can get stuck in the case that the word to be checked is not in the given language. This permits to make a smaller pushdown automaton. It has the states s, t which are both accepting and the stack symbols S, T and the following transition function δ: $\delta(s, 0, S) = \{(s, ST)\}$, $\delta(s, 0, T) = \{(s, TT)\}$, $\delta(s, 1, T) = \{(t, \varepsilon)\}$, $\delta(t, 1, T) = \{(t, \varepsilon)\}$. In summary, the pushdown automaton in state s reads 0 and pushes symbols T; in state t it reads 1 and pulls the symbols T. Once the T are exhausted, the automaton gets stuck; this enforces that only words of the form $0^n 1^m$ with $m \leq n$ are accepted.

If the pushdown automaton would be deterministic and accept by empty stack, then during the whole phase of reading the 0 the stack has to be empty as 0^n is in the language; then the information on how many 0 are there is not recorded or only visible from the state which has only finitely many choices; thus it cannot be compared with how many 1 follow. Therefore the pushdown automaton cannot accept by empty stack.

A linear language is a context-free language where on the right side of each rule $A \to r$, the word r contains at most one non-terminal. Linear grammars have a normal form where the rules $A \to r$ are either of the form $A \to c$, $A \to Bc$, $A \to cB$ or $S \to \varepsilon$ for non-terminals A, B and terminals c; if the rule $S \to \varepsilon$ is in the grammar, then the start symbol S does not occur on any right side of a rule.

(a) Explain the special case of the algorithm of Cocke, Kasami and Younger for linear languages in normal form and why the runtime is in $O(n^2)$ which is better than the usual $O(n^3)$ case.

(b) Assume that L is linear and let $L^{mi} = \{w^{mi} : w \in L\}$, where w^{mi} is the mirror image of w, so $(0012)^{mi} = 2100$. Construct a $O(n^2)$ deterministic membership test for $L \cdot L^{mi}$ and explain the algorithm, using the one from case (a) as a subroutine.

Solution. (a) The algorithm of Cocke, Kasami and Younger constructs on an input word $a_1 a_2 \ldots a_n$ for each pair i, j the set $E_{i,j}$ of non-terminals A such that $A \Rightarrow^* a_i a_{i+1} \ldots a_j$. Here $E_{k,k}$ are for given k all the A which have the rule $A \rightarrow a_k$ in the grammar; for $E_{i,j}$ with $j > i$ one can search search over a parameter k on how to split $a_i a_{i+1} \ldots a_j$ into two parts and then to let $E_{i,j}$ to be all A for which there is a rule $A \rightarrow BC$ with $B \in E_{i,k}$ and $C \in E_{k+1,j}$. This causes the algorithm to compute $O(n^2)$ values in $O(n)$ times each from those below. This gives an overall time complexity of $O(n^3)$.

In the case of a linear grammar, however, the local search can be reduced to checking out whether, in the normal form of linear grammars, to let $E_{i,j}$ to be the set of all A such that there is a non-terminal B and either a rule $A \rightarrow a_i B$ with $B \in E_{i+1,j}$ or a rule $A \rightarrow B a_j$ with $B \in E_{i,j-1}$. So one has only a constant amount of search for computing each $E_{i,j}$ and the overall algorithm is $O(n^2)$.

(b) If L is linear then also L^{mi} is linear; the linear grammar for L^{mi} is obtained from the one for L by replacing each rule $A \rightarrow u$ by $A \rightarrow u^{mi}$; also this new grammar satisfies that each u on the right side has at most one non-terminal and therefore L^{mi} is a linear language. The algorithm of Cocke, Kasami and Younger for L and L^{mi} permits (for grammars in the corresponding normal form) to compute in $O(n^2)$ for a given word $a_1 a_2 \ldots a_n$, which parts $a_i a_{i+1} \ldots a_j$ of the word are in L and which are in L^{mi}. Now $a_1 a_2 \ldots a_n$ is in $L \cdot L^{mi}$ iff either $\varepsilon \in L$ and $a_1 a_2 \ldots a_n \in L^{mi}$ or $a_1 a_2 \ldots a_n \in L$ and $\varepsilon \in L^{mi}$ or there is a $k \in \{1, 2, \ldots, n-1\}$ such that $a_1 a_2 \ldots a_k \in L$ and $a_{k+1} a_{k+2} \ldots a_n \in L^{mi}$. This test can be done in linear time, given the tables for L and L^{mi} and thus the overall complexity is $O(n^2)$.

Which of the following three questions about context-free grammars is decidable:

(a) For a language L given by a grammar in Chomsky Normal Form, is $L = \{0,1,2\}^*$?

(b) For a language L given by a grammar in Chomsky Normal Form, is $\{0\}^* \subseteq L$?

(c) For a language L given by a grammar in Chomsky Normal Form, is $\{0,1,2\}^* - L$ infinite?

Choose among the questions **(a)**, **(b)**, **(c)** the one which can be answered and explain the algorithm to answer the question.

Solution. The right choice is (b). From the Chomsky Normal Form, one can compute the pumping constant k. Now for each word 0^h with $h \geq k$, the pumps together have at most length k and when 0^h is in L so is $0^{h+k!}$. Thus it is sufficient to check whether $\varepsilon, 0, 0^2, 0^3, \ldots, 0^{k+k!}$ are in L. These tests can be done, as one can apply the algorithm of Cocke, Kasami and Younger to check the membership in the given grammar. If now all of $\varepsilon, 0, 0^2, 0^3, \ldots, 0^{k+k!}$ are in L then $\{0\}^* \subseteq L$ else $\{0\}^* \not\subseteq L$.

Recall that a homomorphism $h : \{0,1,2\}^* \to \{3,4,5\}^*$ is a mapping from words to words satisfying $h(v \cdot w) = h(v) \cdot h(w)$ for all v, w; thus h is known when one knows $h(0), h(1), h(2)$. What is the number of homomorphisms $h : \{0,1,2\}^* \to \{3,4,5\}^*$ satisfying $h(012) = 333343333$ and $h(0112) = 3333433343333$?

Solution. $h(1)$ must contain one 4, because $h(0112)$ has two 4 and $h(012)$ has one 4. Thus $h(1) = 3^i 4 3^j$ for some i, j. As $h(11)$ has the subword 43334, one knows that $i + j = 3$. Now choosing $h(0) = 3^{4-i}$ and $h(2) = 3^{4-j}$ gives that $h(012) = 333343333$ and $h(0112) = 3333433343333$. Choosing i to be any value from $0, 1, 2, 3$ and $j = 3 - i$ gives the following four solutions:

$(i = 0, j = 3)$ $h(0) = 3333$, $h(1) = 4333$, $h(2) = 3$;

$(i = 1, j = 2)$ $h(0) = 333$, $h(1) = 3433$, $h(2) = 33$;

$(i = 2, j = 1)$ $h(0) = 33$, $h(1) = 3343$, $h(2) = 333$;

$(i = 3, j = 0)$ $h(0) = 3$, $h(1) = 3334$, $h(2) = 3333$.

There are no other solutions, so there are four homomorphisms h.

Assume that φ_e is the partial recursive function computed by the e-th register machine and that W_e is the range of φ_e. Consider the index set

$$E = \{e : W_e \text{ has at least 5 elements}\}.$$

Is this set (a) recursive, (b) recursively enumerable but not recursive, (c) not recursively enumerable? Choose the right answer and prove why this choice applies.

Solution. The right answer is (b). By the Theorem of Rice, the index set cannot be recursive, as the only recursive index sets are $\emptyset$ and $\mathbb{N}$. Let $F = \{(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5) : y_1 < y_2 < y_3 < y_4 < y_5\}$; this set of tuples is recursively enumerable, indeed even recursive. Note that if φ_e satisfies one of these conditions iff there are x_1, x_2, x_3, x_4, x_5 such that φ_e is defined on these five inputs and $\varphi_e(x_1) < \varphi_e(x_2) < \varphi_e(x_3) < \varphi_e(x_4) < \varphi_e(x_5)$. Whenever a function φ_e takes five or more different values then one can bring the inputs into an order such that the values are strictly increasing. Thus E is the set of all e such that φ_e matches some list of conditions layed out in F. So, by the Theorem of Rice, E is recursively enumerable. Alternatively, one can see this by considering the function

$$f(e, x_1, x_2, x_3, x_4, x_5) = \begin{cases} e & \text{if } \varphi_e \text{ is defined on } x_1, x_2, x_3, x_4, x_5 \\ & \text{and } \varphi_e(x_1) < \varphi_e(x_2) < \varphi_e(x_3) < \\ & \varphi_e(x_4) < \varphi_e(x_5); \\ undefined & \text{otherwise.} \end{cases}$$

The range of this function f is E.

Let $\log(x) = \min\{y \in \mathbb{N} : 2^y \geq x\}$, so $\log(0) = \log(1) = 0$, $\log(2) = 1$, $\log(3) = \log(4) = 2$, $\log(5) = \log(8) = 3$. Write a register machine program which computes the logarithm of the input according to this definition. The registers and constants can be added and subtracted and compared. The possible register values are natural numbers (including 0). The program can have conditional and unconditional jump instructions ("Goto", "If condition Then Goto"). The Return-statement identifies the value of the function.

Solution. For input R_1 and initial value $R_3 = 1$, the program counts how often one has to double R_3 until it is greater or equal R_1. R_2 is the register used for this counting.

```
Line 1: Function Log($R_1$);
Line 2: $R_2 = 0$;
Line 3: $R_3 = 1$;
Line 4: If $R_3 \geq R_1$ Then Goto Line 8;
Line 5: $R_3 = R_3 + R_3$;
Line 6: $R_2 = R_2 + 1$;
Line 7: Goto Line 4;
Line 8: Return($R_2$).
```

END OF PAPER