NATIONAL UNIVERSITY OF SINGAPORE

CS 4232 – Theory of Computation
Solutions

(Semester 1: AY 2019/2020)

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 TWENTYONE (21) 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. 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:		

Construct a context-free grammar for the language L of all palindromes of odd length over the alphabet $\{0, 1, 2\}$ in Chomsky Normal Form. Use at most eight nonterminals. Explain the solution.

A palindrome is a word which is equal to its mirror image like 0102010 and 1102011. A grammar is in Chomsky Normal Form when every rule is either of the form $A \to BC$ or $A \to d$ where A, B, C are nontermimals and d is a terimal symbol from $\{0, 1, 2\}$.

Solution. A standard context-free grammar is

$$(\{S\}, \{0, 1, 2\}, \{S \to 0S0|1S1|2S2|0|1|2\}, S).$$

However, for bringing the grammar into Chomsky Normal Form, one needs to replace in rules which do not lead to single terminals all $0, 1, 2$ by corresponding non-terminals X, Y, Z and break the then resulting rules $S \to XSX|YSY|ZSZ$ into two parts, So the resulting grammar is $(\{S, U, V, W, X, Y, Z\}, \{0, 1, 2\}, P, S)$ with the rules in P being $S \to XU|YV|ZW|0|1|2$, $U \to SX$, $V \to SY$, $W \to SZ$, $X \to 0$, $Y \to 1$, $Z \to 2$.

Bring the grammar $(\{S\}, \{0, 1, 2\}, \{S \to 0S1|0S11|00S1|2\}, S)$ into the normal form for linear grammars where all rules are of one of the forms $A \to Bc$, $A \to cB$, $A \to c$ where A, B are nonterminals and c is a terminal. Use the version for linear languages of the algorithm of Cocke, Kasami and Younger to check whether the word 0021111 is generated by the grammar when it is brought into this linear normal form.

Solution. The normal form mainly requires the introduction of new nonterminals in order to break the long rules on the right side. Now $(\{S, T, U\}, \{0, 1, 2\}, P, S)$ is the new grammar and $S \to 0T|2$, $T \to S1|U1$, $U \to 0S|S1$ are the rules in P. The following pyramid is made:

```
                  SU
               ST    T
            TU    TU    -
         SU    SU    -     -
      -     ST    T     -     -
   -     U     TU    -     -     -
-     -     S     -     -     -     -
0     0     2     1     1     1     1
```

In this pyramid, "-" stands for empty set and a list and one or more letters stand for the set consisting of the corresponding nonterminals. The bottom has the digits of the input word. The working shows that the grammar can generate 0021111.

Provide a regular expression of the set of all decimal numbers which do not have leading zeroes and which are odd multples of 25, that is, of the form $(2k+1)$ times 25 for some $k \in \mathbb{N}$.

The regular expression can use union ($\cup$), concatenation ($\cdot$), Kleene star (*) and brackets and finite sets of numbers (like $\{2, 3, 5, 7, 11\}$). For example,

$$\{3, 5, 8\}^* \cdot (\{888\} \cup (\{533\} \cdot \{3\}^*))$$

is the set of all numbers where all digits are 3 or 5 or 8 and where the number either ends with the digits 888 or ends with one digit 5 followed by two or more digits 3.

Solution. The odd multiples of 25 are $25, 75, 125, 175, 225, 275, \ldots$ and the solution is the following expression:

$$\{25, 75\} \cup (\{1, 2, 3, 4, 5, 6, 7, 8, 9\} \cdot \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^* \cdot \{25, 75\}).$$

So one has to make sure that the number ends with either 25 or 75 and before this is either ε or a decimal number not starting with 0.

Construct for the alphabet $\{0,1,2,3,4,5,6,7,8,9\}$ a deterministic finite automaton (dfa) which recognises the set of all words which satisfy the following: After the last 0 or from the beginning if the word has no 0, the sum of the digits is 2 modulo 7.

For example, 9902, 11 and 12345012345011 should be accepted; furthermore, 110, 2021 and 1234567 should be rejected.

Solution. The states are $\{0,1,2,3,4,5,6\}$ and the start state is 0. The transition function $\delta(q,d)$ is for state q and digit d defined as

$$\delta(q,d) = \begin{cases} 0 & \text{if } d = 0; \\ (q+d) \bmod 7 & \text{if } d \neq 0. \end{cases}$$

The accepting state is 2. Instead of the formula, one can also give the following table.

q	type	$\delta(q,a)$ for $a=0$	1	2	3	4	5	6	7	8	9
0	rej	0	1	2	3	4	5	6	0	1	2
1	rej	0	2	3	4	5	6	0	1	2	3
2	acc	0	3	4	5	6	0	1	2	3	4
3	rej	0	4	5	6	0	1	2	3	4	5
4	rej	0	5	6	0	1	2	3	4	5	6
5	rej	0	6	0	1	2	3	4	5	6	0
6	rej	0	0	1	2	3	4	5	6	0	1

Consider the following grammar in Greibach Normal Form:

$$(\{S, T\}, \{0, 1, 2\}, \{S \to 0S|2S|1|1T|1SS, T \to 0T|2T|0|2\}).$$

Let L be the language generated by this grammar. Which of the following statements is true:

- ☐ L is regular;
- ☐ L is linear but not regular;
- ☐ L is context-free but not linear;
- ☐ L is context-sensitive but not context-free;
- ☐ L is recursively enumerable but not context-sensitive?

Explain why this choice applies. What is the deterministic time complexity to check whether a word w is in the language L generated by this grammar?

Solution. One can see that S can generate an arbitrary number of 0 and 2. Furthermore, whenever S makes a 1, either S disappears or S becomes a T which can only make $0, 2$ or two new copies of S replace the old S. Thus an invariant is that the number of S plus the number of 1 generated is always odd and indeed S generates all those words in $\{0, 1, 2\}^*$ which contain an odd number of 1. This is equivalent to saying that the ternary number represented by the word is odd as a natural number. The language of these numbers is regular and this can be checked by a dfa in linear time.

Is the language $L = \{0^n 1^m 2^n 3^m : n, m > 1\}$ context-free? If so, provide a context-free grammar; if not, state a suitable pumping lemma and use it to show that it is not context-free.

Solution. The language is not context-free. One uses the standard pumping lemma for context-free languages which says that there is a constant k such that for every word u longer than k, one can split u into $vwxyz$ such that $|wxy| \leq k$ and $vw^h xy^h z \in L$ for all $h \in \mathbb{N}$.

Given the pumping constant k, one chooses $n, m \geq 2k$ and $u = 0^n 1^m 2^n 3^m$. Pumping can due to the bound on the length of wxy only affect two neighbouring letters, say 1 and 2. When pumping up the length of 1^m or 2^n increases while those of 0^n and 3^m remain constant, hence the word $vwwxyyz$ cannot be in L. Similarly when pumping up parts of 0^n and 1^m, the number of 2 and 3 remains unchanged and when pumping up parts of 2^n and 3^m, the number of 0 and 1 remains unchanged. Thus in all cases, pumping goes out of L and L does not satisfy the context-free pumping lemma. Thus L is not context-free.

Given a homomorphism and a deterministic context-free set L, is $h(L)$ deterministic context-free?

☐ Yes, ☐ No.

Explain the answer.

Note that a set is deterministic context-free iff a deterministic pushdown automaton recognises it.

Solution. The answer is "No", there are choices of L and h with $h(L)$ not being deterministic context-free. Consider the language $L = \{0^n 2 0^n 1 0^m 1, 0^n 1 0^m 2 0^m 1 : n, m \geq 1\}$. A deterministic pushdown automaton will first put all 0 onto the stack until the first nonzero symbol a comes.

If $a = 2$, it will then pull the 0 from the stack and compare the number with the pushed once and see whether the bottom symbol of the stack is reached exactly at the time the 1 follows. Afterwards it will verify that there are zeroes followed by a 1 in the input and then go into an accepting state and stop.

If $a = 1$, it will empty the stack up to the bottom symbol, push back the bottom symbol onto the stack and then push the 0 onto the stack until it sees the 2. Then it will compare the 0 which follow until it reaches the 1 and if they exactly remove all archived symbols but not the bottom symbol then it will go into an accepting state after the 1 and stop.

Now let $h(0) = 0, h(1) = 1, h(2) = 1$. Now a pushdown automaton for $h(L)$ cannot know whether it should stack up the first run of zeroes and then compare with the second run of zeroes or stack up the second run of zeroes and compare with the third run of zeroes. Since the memory management is contrary in both cases for the middle run of zeroes, there is no deterministic pushdown automaton which can handle this. Thus the deterministic context-free languages are not closed under the image of homomorphisms; even if those map each letter to exactly one letter.

Let $\varphi_0, \varphi_1, \ldots$ be an acceptable numbering of all partial-recursive functions and let

$$I = \{e : \varphi_e \text{ takes some value } y \text{ on at least } y+1 \text{ inputs}\}.$$

Which of the following statements is true:

- ☐ I is not an index set;
- ☐ I is a decidable index set;
- ☐ I is a recursively enumerable index set which is not decidable;
- ☐ I is an index set but not recursively enumerable?

Explain the answer. Recall that an index set is a subset $J \subseteq \mathbb{N}$ such that whenever $\varphi_d = \varphi_e$ then either both $d, e \notin J$ or both $d, e \in J$. When using the Theorem of Rice, please state the version of it which applies.

Solution. The set is an index set, as the definition only refers to the function φ_e computed by index e, but has no other dependence on e. Thus all indices d, e which compute the same function either both satisfy the condition to go into I or both do not satisfy the condition to go into I. As there are everywhere undefined functions, their indices go not into I while the index of every constant function goes into I. Thus the set is undecidable, as the first Theorem of Rice states that the only decidable index sets are $\emptyset$ and $\mathbb{N}$. The second Theorem of Rice states that an index set is recursively enumerable iff there is an enumeration of tuples such that $e \in I$ iff there is one tuple $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ such that $\varphi_e(x_1) = y_1$ and $\varphi_e(x_2) = y_2$ and $\ldots$ and $\varphi_e(x_n) = y_n$. In the case of the given I, the set of tuples S is the set of all $(x_1, y, x_2, y, \ldots, x_y, y, x_{y+1}, y)$ where $x_1, x_2, \ldots, x_y, x_{y+1}$ is a list of $y+1$ pairwise different inputs. Now $e \in I$ iff there is an y such that φ_e takes the value y on $y+1$ inputs iff φ_e agrees on the x's of such a tuple with the output y where the list of x's contains $y+1$ inputs. It is easy to see that S is recursively enumerable; indeed S is even decidable. Therefore the correct choice is

> "I is a recursively enumerable index set which is not decidable"

among the four choices for the set I.

A subset A of $\mathbb{N}$ is Diophantine iff there is a polynomial f with n variables for some n and with coefficients from $\mathbb{Z}$ such that, for all $x \in \mathbb{N}$,

$$x \in A \Leftrightarrow \exists y_1, \ldots, y_n \in \mathbb{N}\,[f(x, y_1, \ldots, y_n) = 0].$$

It is known that the set P of primes is Diophantine, that is, there is a polynomial p with

$$x \in P \Leftrightarrow \exists y_1, \ldots, y_9 \in \mathbb{N}\,[p(x, y_1, \ldots, y_9) = 0].$$

Use this polynomial p to define a polynomial q witnessing that the set Q given as

$$x \in Q \Leftrightarrow \exists v, w \in P\,[x = v + w \text{ and } v \text{ is odd and } w \text{ is odd}]$$

is Diophantine. Provide the polynomial q using p and explain how the construction works.

Solution. Given p, one knows that two numbers v', w' are primes iff there are $y_1, \ldots, y_9$ and $z_1, \ldots, z_9$ with $p(v', y_1, \ldots, y_9)$ and $p(w', z_1, \ldots, z_9)$ being 0. Furthermore, as one wants v' and w' to be odd, one uses $v' = v + 3$ and $w' = w + 3$ for variables v, w which are quantified over natural numbers and every prime greater equal 3 is odd. Furthermore, all three polynomials are 0 iff the sum of their squares is 0. So one gets the following polynomial q:

$$q(x, v, w, y_1, \ldots, y_9, z_1, \ldots, z_9) =$$
$$(x - v - w - 6)^2 + (p(v + 3, y_1, \ldots, y_9))^2 + (p(w + 3, z_1, \ldots, z_9))^2.$$

Now $x \in Q$ iff there are natural numbers $v, w, y_1, \ldots, y_9, z_1, \ldots, z_9$ which make this polynomial q to take the value 0 and these natural numbers must satisfy that $x = v + w + 6$ and $v + 3$ is a prime as witnessed by $y_1, \ldots, y_9$ and $w + 3$ is a prime as witnessed by $z_1, \ldots, z_9$. Note that one has to take two different sets of variables $y_1, \ldots, y_9$ and $z_1, \ldots, z_9$ for this, as it might be that one cannot choose the same values in p to witness that both $v + 3$ and $w + 3$ are primes. This completes the solution.

The polynomial p itself is quite complicated and therefore it is not the task of this question to find p itself; one such p, using more than the theoretically possible 9 auxiliary variables, can be found on Wikipedia on

`https://en.wikipedia.org/wiki/Formula_for_primes`

and note that Wikipedia more looks at a polynomial whose nonnegative values cover all the primes, but such a polynomial can be translated into another one witnessing the above. It is an open problem whether Q equals to the set of even numbers above 6, if so, $q(x, y) = x - 6 - 2y$ is a much easier solution. Using that $2 + 2 = 4$, Cristian Goldbach conjectured in 1742 that every even number greater equal four is the sum of two prime numbers.

Question 10 [5 marks]

Write a register machine program Count which counts how many digits in the decimal representation of x are the digit 8.

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. If Macros are used, they have to be given explicitly.

Solution. The register machine copies R_1 into R_2 and determines in each iteration the last digit R_4 of R_2 and the value $R_5 = R_2/10$. If R_4 is 8, the counter R_3 of the number of digits being 8 is incremented. Afterwards R_2 is updated to R_5 and the loop is repeated, provided that $R_2 > 0$.

Line 1: Function Count(R_1);
Line 2: $R_2 = R_1$; $R_3 = 0$; // R_2 is copy of input, R_3 counts the 8;
Line 3: $R_4 = R_2$; $R_5 = 0$; // Start to compute $R_4 = R_2\%10$, $R_5 = R_2/10$;
Line 4: If $R_4 < 10$ Then Goto Line 7;
Line 5: $R_4 = R_4 - 10$; $R_5 = R_5 + 1$; // Inner Loop of this computation.
Line 6: Goto Line 4;
Line 7: If $R_4 \neq 8$ Then Goto Line 9; // R_4 is last digit;
Line 8: $R_3 = R_3 + 1$; // Update R_3 in the case that $R_4 = 8$;
Line 9: $R_2 = R_5$; // Update $R_2 = R_2/10$, this value is in R_5;
Line 10: If $R_2 > 0$ Then Goto Line 3; // Continue if there are digits left;
Line 11: Return(R_3).

END OF PAPER