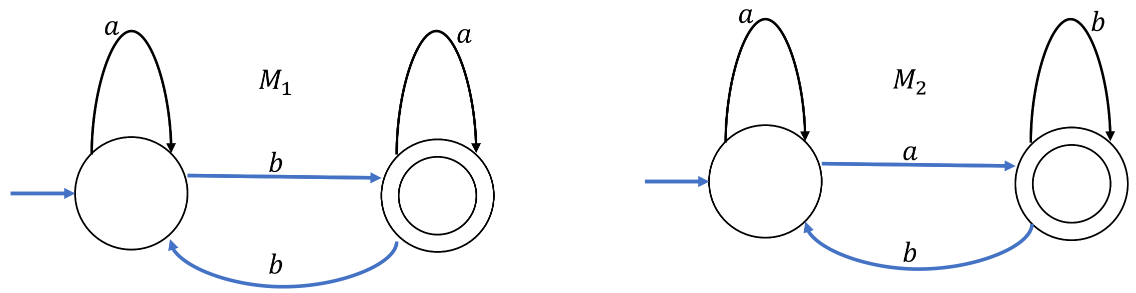
Theory of Computation '23 Endsem
Time : 120 minutes, Marks : 70

Name : **Roll :** **Marks :**

Beware: Solutions are to be written in the space provided. NO EXTRA SHEET. Meaningless rambles fetch negative credits. For Problem 2-6, 1 point for writing 'I don't know'

Problem 1. (10 points) Recall

$$A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an NFA and } w \text{is a string and } w \in L(N)\}$$
$$EQ_{DFA,REX} = \{\langle D, R\rangle \mid D \text{ is a DFA and } R \text{ is a regular expression and } L(D) = L(R)\}$$



Let R be the regular expression $a^\star b(a^\star b)^\star$ State True or False with a **single** sentence argument.

1. $\langle M_1, aba\rangle \in A_{NFA}$
   True. aba **is accepted by** M_1
2. $\langle M_1, bab\rangle \in A_{NFA}$
   False. bab **not accepted by** M_1
3. $\langle M_1, M_2\rangle \in EQ_{DFA,REX}$
   False. M_2 **is not a reg-ex**
4. $\langle M_1, R\rangle \in EQ_{DFA,REX}$
   False. $bb \in L(R)$ **but** $bb \notin L(M_1)$
5. $\langle M_2, R\rangle \in EQ_{DFA,REX}$
   False. M_2 **is not a DFA OR** $bb \in L(R)$ **but** $bb \notin L(M_2)$

Rubric : +1 for correct answer , +1 for reason

Problem 2. Prove without using Rice's Theorem that the following language is undecidable

$$B_{TM} = \{\langle M \rangle \mid M \text{ is a deterministic Turing Machine with input alphabets } a, b \text{ such that } L(M) \subseteq b(a+b)^\star, L(M) \neq \emptyset\}$$

Solution. We can reduce either A_{TM} or $HALT_{TM}$ to B_{TM}. **+1** I will show reduction from A_{TM}. Assume there is a decider for B_{TM}, D. **+1**

We will show how to construct a decider R for A_{TM}. **+1**

Decider $R\langle M, w\rangle$ **:** **+1**

1. Construct a new TM M_w which does the following on any input x **+1**
   - If $x \neq b$ then *Reject* **+2** Else
   - Run M on w and *Accept* if M accepts w **+2**
2. Run $D\langle M_w\rangle$. *Accept* if D accepts else *Reject* **+2**

Why this works (not required for full score) : The language of M_w is $\{b\}$ if and only if M accepts w. Hence the decider D for B_{TM} can separate the two scenarios and decide A_{TM}. **Remark :**

a. Some of you might have shown a mapping reduction instead of proving contradiction. That also works. There is also a possiblity of a general reduction and not a mapping reduction. All of those are fine.

b. A few of you have argued that since the subsets of $b(a+b)^\star$ is an uncountable set, B_{TM} is unrecognizable and hence undecidable. This is a *wrong* solution. The point is, B_{TM} itself is not a subset of $b(a+b)^\star$. Rather, it is a language which contains Turing Machines that accept only such languages (which might potentially be a much much smaller set than all possible subsets of $b(a+b)^\star$. We have given a small credit for this attempt.

Problem 3. $DOUBLE{-}SAT = \{\langle\varphi\rangle \mid \varphi$ is a Boolean formula satisfiable by at least two assignments $\}$

Prove that $DOUBLE - SAT$ in **NP**-Complete. **Solution.**

a. $DOUBLE - SAT$ is in NP. This is very easy using small certificates which are two different assignments. It can be verified in polynomial time if both are satisfying or not. **+2**

b. $DOUBLE - SAT$ is **NP**-hard. This can be done by reduction from SAT. Suppose φ is an instance of SAT. We create the following instance of $DOUBLE - SAT$: $\varphi' = \varphi \wedge (x_1 \vee x_2)$, where x_1, x_2 are Boolean variables that *do not occur* in φ. **+3**

   Suppose we have a YES instance φ. Then there are clearly two distinct satisfying assignments for φ' : just take the satisfying assignment for φ and make $x_1 = 1, x_2 = 0$ for one and $x_1 = 0, x_2 = 1$ for the other. Clearly φ' is a YES instance then. **+3**

   Suppose we have a YES instance φ'. Then clearly there is at least one satisfying assignment for φ since we can simply ignore the clause $x_1 \vee x_2$ and use the fact that these variables do not occur in φ. **+3**.

The above reduction is trivially polynomial time. **+1**.

Problem 4. Say that string x is a prefix of string y if a string z exists where $xz = y$, and say that x is a proper prefix of y if in addition $x \neq y$. A language is prefix-free if it doesn't contain a proper prefix of any of its members. Let

$$PrefixFreeREX = \{R \mid R \text{ is a regular expression where } L(R) \text{ is prefix-free }\}$$

Show that PrefixFreeREX is decidable. (Give a proper description of a Turing machine which decides this language) **Solution.**

We design a deterministic Turing Machine which decides this language

On any input $\langle R \rangle$

1. First check if R is a valid regular expression and *Reject* if not **We are not penalizing for missing this**
2. Convert R to an NFA N **+3**
3. Convert the $NFA\ N$ to a DFA D **+1**
4. Minimize the DFA D **+2**
5. Check the following two conditions
   - There exists a path between *two* different final states in the minimized DFA **+3**
   - There exists a cycle containing at least one final state **+3**
6. If any of the above condition is TRUE, *Reject* otherwise *Accept*

Remarks : Each of the above steps is absolutely necessary. Common mistakes are : converting R to DFA directly (no algorithm is known for that), checking conditions on DFA without minimizing (might give false negatives) etc.

Problem 5. Prove that if **P** = **NP** then every language in **P**, except $\emptyset$ and $\Sigma^\star$ is **NP**-complete.

We will show that any language $L \in$**P** is **NP**-complete.

1. $L \in NP$ clearly since $P \subset NP$ **+2**
2. We do a *polynomial time* reduction from any language $A \in$ **NP** to L. **+2, -1 for missing polynomial time**

   Now the proof. This has **+8** points and the evaluation is subjective depending on how much close to correctness your proof is. Please do not argue for score increments unless you have written a perfectly correct solution.

   We need two observation. Firstly, since $L \notin \{\Sigma^\star, \Phi$, there exists a string $x \in L$ and a string $x' \notin L$. Now, consider the language $A \in$ **NP**. By the assumed condition, A is also in P. Now we simply define a mapping $f : A \to L$ such that $f(y) = x$ if $y \in A$ and $f(y) = x'$ if $y \notin A$. The main observation is that, this function is computable in polynomial time. The reason simply is that A is decidable in polynomial time by the assumed condition.

Remark: Many of you have used a fact : problems in **P** are reducible to each other. This is an absolutely non-trivial fact and you do not get credit for this without a proof. For example, this implies every problem in **P** is reducible to sorting - clearly that is not true, is it?

Problem 6. Let B be the language of all palindromes over 0, 1 containing an equal number of 0s and 1s. Show that B is not context-free.

Solution.

1. Given any $p > 0$ **+1**, choose the string $s = 0^p 1^{2p} 0^p$ **+2**. Clearly $s \in B$.
2. Consider any split $s = uvxyz$ such that $|vxy| \leq p$ and at least one of $|v|, |y| > 0$. **+2**

   Now there could be two cases and we show that Pumping Lemma fails under both of them

   - Case 1 : vxy spans only the $1's$. Let $|vy| = m$. But then the string $uv^0xy^0z = 0^p1^{2p-m}0^p$. This is still a palindrome but does not contain equal number of 0's and 1's. **+3**
   - Case 2 : vxy contains 0's. But now, it can be observed that vxy can either span strictly the first half or the second half. In either case, uv^0xy^0z will contain a different number of 0's in the two halves. Hence, this will not be a palindrome and hence not in B (Note that in this case, number of 0's and 1's in the pumped down string can still be equal depending on the decomposition). **+4**