UMC-205 Assignment-5

Aditya Gupta
SR No: 22205

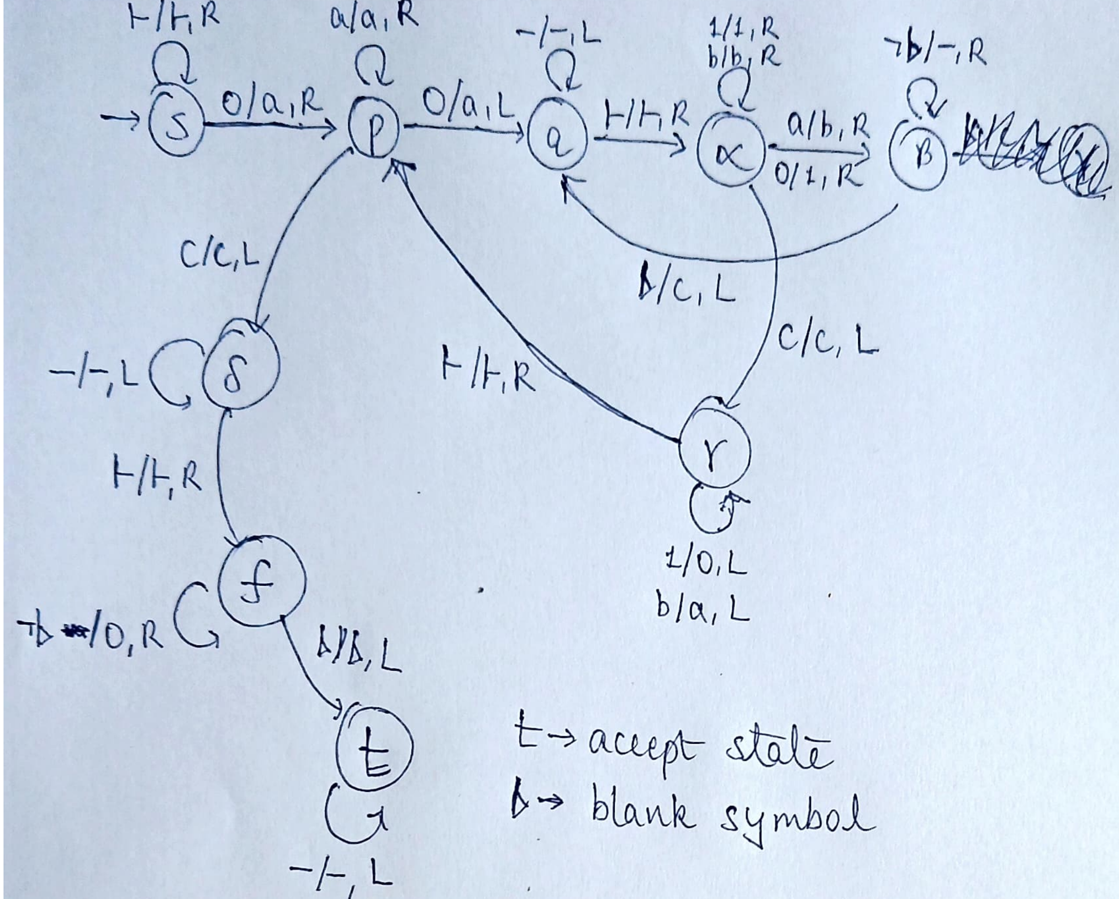
Question 1

We are given an input of n 0's. To compute the square, we need to write the n 0's for n times. We do this by keeping track of the number of times we have written the n 0's. We do this for n-1 times, since we already have n 0's to start with. The turing machine is shown in the next page.

In this machine, we mark the first n 0's to represent our current condition. If a 0 is marked as 'a', then it means that we have written or are writing the n 0's corresponding to it. If the 0 is still 0, it means we have not reached it yet.

While writing the new 0's, we replace every 'a' by 'b' and 0 by 1 in the original n 0's to signify how many of those are written in this set. We keep writing a 'c' on the blank tape at the right for every element in the first n 0's. Once we have written the n set, we replace the 'b' by 'a' and the 0 by 1 in the original n 0's. Then we move to the left end point.

We repeat this by finding the next 0 in the first n 0's, marking it as 'a' and writing the n 0's again. We do this n-1 times. Once this is done, we will have a total of n^2 symbols on the tape. So, we just replace all of them by 0 and halt.



t → accept state
ƀ → blank symbol

Question 2

We have the question "Given a Turing Machine M and a state q of M, does M ever enter state q on some input". This question is undecidable.

We proceed by contradiction. Assume such a turing machine exists, which can decide if a turing machine M will reach the state q on some input. Call this turing machine M'. Now, let the state q be the accept state t of M. Thus M' can decide if M reaches the state t on some input. In other words, M' can decide if the language of M is non empty. We know from class that this problem is undecidable. Thus, our assumption that M' exists is false. Hence, the question is undecidable.

Question 3

a. We are given that L is a regular language and K is an arbitrary language. The language L/K is defined as:

$$L/K = \{x \mid \exists y \in K, xy \in L\}$$

We need to prove that L/K is regular.

Since L is regular, there exists a DFA $A = (Q, s, \delta, F)$ that accepts L. Using this, we will construct a DFA $A' = (Q, s, \delta, F')$ that accepts L/K. The states, start state and transition function of A' will be the same as A. The set of final states F' will be defined as:

$$F' = \{q \mid \hat{\delta}(q, x) \in F \text{ for some } x \in K\}$$

We will now prove that A' accepts L/K. Let $x \in L/K$. This means that there exists $y \in K$ such that $xy \in L$. Since A accepts L, there exists a path from s to a final state $q \in F$ on input xy. Let $\hat{\delta}(s, x) = q'$. Then $\hat{\delta}(q', y) = q$. Since $y \in K$, $q' \in F'$. Hence $L/K \subseteq L(A')$.

Now, let $x \in L(A')$. This means that there exists a path from s to a final state $q' \in F'$ on input x. Since $q' \in F'$, there exists $y \in K$ such that $\hat{\delta}(q', y) = q \in F$. Thus $xy \in L$. By the definition of L/K, $x \in L/K$. Hence $L(A') \subseteq L/K$.

$\therefore$ $L(A') = L/K$ and L/K is regular.

b. We have shown above that the language L/K is regular. However, this does not imply that we can construct an automaton for it using the DFA for L and turing machine for K.

Consider the construction for the DFA A' above. To check if a state q is in F', we need to check if:

$$\{x \in \Sigma^* \mid \hat{\delta}(q, x) \in F\} \cap K \neq \phi$$

Let the language $\{x \in \Sigma^* \mid \hat{\delta}(q, x) \in F\}$ be L. Since we can make a DFA for it, it is regular. Since it is regular, we can make a turing machine for it. Hence L is recursively enumerable. We are also given a turing machine for K, which implies K is also recursively enumerable. Let the turing machines for L and K be M_L and M_K respectively.

Since both L and K are recursively enumerable, their intersection is also recursively enumerable, as we can make a turing machine M' by using the encoding of M_L and M_K to simulate both of them on any input. M' would accept if the input string is accepted by both M_L and M_K. Thus, $L \cap K$ is recursively enumerable.

Now, the question reduces to asking if the turing machine M' for the recursively enumerable language $L \cap K$ accepts any string. We know that this question is undecidable. Hence, we cannot construct an automaton for L/K using the DFA for L and turing machine for K.

Question 4

a. TOTAL = $\{M \mid M$ halts on all inputs$\}$

We have to show that this language is not recursively enumerable. We use Rice's theorem to prove this. Rice's theorem states that any non monotone property of recursively enumerable languages is not recursively enumerable. We will use this to show that TOTAL is not recursively enumerable.

Consider the turing machines M_1 and M_2 such that M_1 rejects every string and M_2 accepts strings of the form $M\#x$ if M halts on input x. The languages of M_1 and M_2 are ϕ and HP respectively. Since we have a turing machine for these, the languages are recursively enumerable.

Now, consider the TOTAL property for these two languages. Clearly, M_1 halts on all inputs but M_2 does not (since HP is not recursive). Thus $M_1 \in TOTAL$ but $M_2 \notin TOTAL$. However, $L(M_1) \subseteq L(M_2)$. Thus, the TOTAL property is non monotone. By Rice's theorem, TOTAL is not recursively enumerable.

b. $\overline{TOTAL} = \{M \mid M$ does not halt on some input$\}$

We have to show that this language is not recursively enumerable. Once again, we use Rice's theorem to prove this. Rice's theorem states that any non monotone property of recursively enumerable languages is not recursively enumerable. We will use this to show that $\overline{TOTAL}$ is not recursively enumerable.

Consider the turing machines M_1 and M_2 such that M_1 accepts strings of the form $M\#x$ if M halts on input x and M_2 accepts every string. The languages of M_1 and M_2 are HP and Σ^* respectively. Since we have a turing machine for these, the languages are recursively enumerable.

Now, consider the $\overline{TOTAL}$ property for these two languages. Clearly, M_1 does not halt on some input (since HP is not recursive) but M_2 halts on all inputs. Thus $M_1 \in \overline{TOTAL}$ but $M_2 \notin \overline{TOTAL}$. However, $L(M_1) \subseteq L(M_2)$. Thus, the $\overline{TOTAL}$ property is non monotone. By Rice's theorem, $\overline{TOTAL}$ is not recursively enumerable.

Question 5

a. A VALCOMP string is:

$$\begin{matrix}\vdash & b & a & a & b & b & \flat\\ s & - & - & - & - & - & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & s & - & - & - & - & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & - & p & - & - & - & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & - & - & p & - & - & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & - & - & - & - & p & - & -\end{matrix}\#$$

$$\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & - & - & - & - & p & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & \flat\\ - & - & - & - & - & - & - & p\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & b & c\\ - & - & - & - & - & - & q & -\end{matrix}\#\begin{matrix}\vdash & b & a & a & b & c & c\\ - & - & - & - & - & t & - & -\end{matrix}$$

b. The set of matching triples is:

- No change:

$$\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & -\;-\;- \end{pmatrix}$$

- $(s,\vdash)\to(s,\vdash,R)$

$$\begin{pmatrix} \vdash a_1\,a_2 & & \vdash a_1\,a_2\\ s\;-\;- & , & -\;s\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & s\;-\;- \end{pmatrix}$$

- $(s,b)\to(p,b,R)$

$$\begin{pmatrix} b\,a_1\,a_2 & & b\,a_1\,a_2\\ s\;-\;- & , & -\;p\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & p\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,b\,a_2 & & a_1\,b\,a_2\\ -\;s\;- & , & -\;-\;p \end{pmatrix},\begin{pmatrix} a_1\,a_2\,b & & a_1\,a_2\,b\\ -\;-\;s & , & -\;-\;- \end{pmatrix}$$

- $(p,x)\to(p,x,R), x\in\Gamma\setminus\{\flat\}$

$$\begin{pmatrix} x\,a_1\,a_2 & & x\,a_1\,a_2\\ p\;-\;- & , & -\;p\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & p\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,x\,a_2 & & a_1\,x\,a_2\\ -\;p\;- & , & -\;-\;p \end{pmatrix},\begin{pmatrix} a_1\,a_2\,x & & a_1\,a_2\,x\\ -\;-\;p & , & -\;-\;- \end{pmatrix}$$

- $(p,\flat)\to(q,c,L)$

$$\begin{pmatrix} \flat\,a_1\,a_2 & & c\,a_1\,a_2\\ p\;-\;- & , & -\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & -\;-\;q \end{pmatrix},\begin{pmatrix} a_1\,\flat\,a_2 & & a_1\,c\,a_2\\ -\;p\;- & , & q\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,\flat & & a_1\,a_2\,c\\ -\;-\;p & , & -\;q\;- \end{pmatrix}$$

- $(q,b)\to(t,c,L)$

$$\begin{pmatrix} b\,a_1\,a_2 & & c\,a_1\,a_2\\ q\;-\;- & , & -\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & -\;-\;t \end{pmatrix},\begin{pmatrix} a_1\,b\,a_2 & & a_1\,c\,a_2\\ -\;q\;- & , & t\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,b & & a_1\,a_2\,c\\ -\;-\;q & , & -\;t\;- \end{pmatrix}$$

- $(t,y)\to(t,y,R), y\in\Gamma$

$$\begin{pmatrix} y\,a_1\,a_2 & & y\,a_1\,a_2\\ t\;-\;- & , & -\;t\;- \end{pmatrix},\begin{pmatrix} a_1\,a_2\,a_3 & & a_1\,a_2\,a_3\\ -\;-\;- & , & t\;-\;- \end{pmatrix},\begin{pmatrix} a_1\,y\,a_2 & & a_1\,y\,a_2\\ -\;t\;- & , & -\;-\;t \end{pmatrix},\begin{pmatrix} a_1\,a_2\,y & & a_1\,a_2\,y\\ -\;-\;t & , & -\;-\;- \end{pmatrix}$$

- $(s, a) \to (r, a, L)$

$$\begin{pmatrix} a_1\,a\,a_2 & a_1\;a\;a_2 \\ -\;s\;-, & r\;-\;- \end{pmatrix}, \begin{pmatrix} a\,a_1\,a_2 & a\;a_1\;a_2 \\ s\;-\;-, & -\;-\;- \end{pmatrix}, \begin{pmatrix} a_1\,a_2\,a & a_1\;a_2\;a \\ -\;-\;s, & -\;r\;- \end{pmatrix}, \begin{pmatrix} a_1\,a_2\,a_3 & a_1\;a_2\;a_3 \\ -\;-\;-, & -\;-\;r \end{pmatrix}$$

- $(r, z) \to (r, z, R), z \in \Gamma$

$$\begin{pmatrix} z\,a_1\,a_2 & z\;a_1\;a_2 \\ r\;-\;-, & -\;r\;- \end{pmatrix}, \begin{pmatrix} a_1\,a_2\,a_3 & a_1\;a_2\;a_3 \\ -\;-\;-, & r\;-\;- \end{pmatrix}, \begin{pmatrix} a_1\,z\,a_2 & a_1\;z\;a_2 \\ -\;r\;-, & -\;-\;r \end{pmatrix}, \begin{pmatrix} a_1\,a_2\,z & a_1\;a_2\;z \\ -\;-\;r, & -\;-\;- \end{pmatrix}$$

c. We prove this by induction on length of the valid configurations.

P(n): For c_1 and c_2 of length n, $c_1 \overset{1}{\Longrightarrow} c_2$ iff for each position in c_1, the triples of symbols in c_1 and the corresponding triples in c_2 are valid.

Base Case: For $n = 1$, the statement is trivially true as the "no change" trivial triple works.

Inductive step: Assume P(n) is true. We show P(n+1).

Let c_1 and c_2 be valid configurations of length n+1. Then $c_1 \overset{1}{\Longrightarrow} c_2$ in two ways:

- $c_1[1 \ldots n] \overset{1}{\Longrightarrow} c_2[1 \ldots n]$ and $c_1(n+1) = c_2(n+1)$

  In this case, we know that the triples of symbols in $c_1[1 \ldots n]$ and the corresponding triples in $c_2[1 \ldots n]$ are valid by the inductive hypothesis. Since $c_1(n+1) = c_2(n+1)$, the triples of the form $c(n-1)c(n)c(n+1)$ are also valid. Thus, the triples of symbols in c_1 and the corresponding triples in c_2 are valid.

- $c_1[1 \ldots n] = c_2[1 \ldots n]$ and $c_1(n+1) \overset{1}{\Longrightarrow} c_2(n+1)$

  In this case, we know that the triples of symbols in $c_1(n+1)$ and the corresponding triples in $c_2(n+1)$ are valid by equality. Since $c_1(n+1) \overset{1}{\Longrightarrow} c_2(n+1)$, the triples of the form $c(n-1)c(n)c(n+1)$ are also valid. Thus, the triples of symbols in c_1 and the corresponding triples in c_2 are valid.

Thus, P(n+1) is true. By induction, P(n) is true for all n. Hence, for any two valid configurations c_1 and c_2, $c_1 \overset{1}{\Longrightarrow} c_2$ iff for each position in c_1, the triples of symbols in c_1 and the corresponding triples in c_2 are valid.

Question 6

We have to show that it is undecidable whether the intersection of two given CFLs is a CFL.

We start by considering the ALT-VALCOMP language for a turing machine, which consists of words of the form $c_0\#c_1^R\#c_2\#c_3^R \ldots \#$ where $c_0\#c_1\#c_2\#c_3 \ldots \#$ is the encoding of a valid halting computation of the turing machine M on input x (ie, the string length must be finite and must end on an accept or reject state).

$$ALT - VALCOMP_M = \bigcup_{x \in \Sigma*} \{c_0\#c_1^R\#c_2\#c_3^R \ldots \# \mid c_0\#c_1\#c_2\#c_3 \ldots \# \in VALCOMP_{M,x}\}$$

We will show that the language ALT-VALCOMP can be formed by the intersection of two CFLs. To do this, consider two PDAs P_1 and P_2 such that:

- P_1 checks if each odd numbered configuration is followed by a valid configuration for the given turing machine M
- P_2 checks if each even numbered configuration is followed by a valid configuration for the given turing machine M

To perform the check, the PDA P_1 would first dump the entire odd position configuration into its stack. Then, it would start reading the stack and compare it to the even configuration which it reads next. This way, the last letter of the odd configuration can be compared to the last letter of the even configuration when the process starts. This goes on till the PDA reads the entire even string. If the configurations are valid, the PDA accepts. The PDA P_2 would do the same, but in the reverse order.

From this, we conclude that the language ALT-VALCOMP is the intersection of two CFLs (since PDAs are equivalent to CFLs). Now, assume we had a turing machine T to decide the original question, ie whether the intersection of two CFLs is a CFL. We can use this turing machine to decide whether ALT-VALCOMP is a CFL.

However, we know that ALT-VALCOMP is not a CFL if it is non empty. This can be shown using the Pumping Lemma for CFLs. Since both VALCOMP and ALT-VALCOMP require the $c_i's$ to be of equal length, pumping at any point would disrupt this property. However, this only applies to strings with non zero length. If the language is empty, it is a CFL trivially, since no strings would be present to pump through.

From this, we can conclude that deciding whether the ALT-VALCOMP language is a CFL is the same as deciding whether or not it is empty. The ALT-VALCOMP language is non empty if the turing machine M halts on some input x. Thus, the turing machine T can be used to decide whether or not the language of M is non empty. However, we know that this problem is undecidable. Hence, the turing machine T cannot exist. Thus, the original question is undecidable.