Formal Languages and Automata Theory IV

Rana Barua

Visiting Scientist
IAI, TCG CREST, Kolkata

1 Turing Machines

Turing machine (TM) was introduced by Alan Turing as a model of "any possible computation". It consists of an infinite tape with cells, a finite control and a tape-head scanning the content of a cell at any given instant. Depending on the state of the finite control and the letter being scanned by the tape-head, the the TM in one move can print a symbol in place of the scanned letter, move the tape-head one cell to the right or left and enters, perhaps, a new state. Initially, the input string w is placed in consecutive cells with all other cells occupied by the **blank** symbol B.The TM starts at an initial state scanning the first letter of the input string w. If after finitely many moves, the TM enters an accepting state then the TM accepts the input w. Formally,

Definition 1. *A Turing machine $\mathcal{M}$ is a 7-tuple $(\Sigma, \Gamma, Q, q_0, \delta, B, F)$, where*

1. *$\Sigma \subseteq \Gamma$: is the **input alphabet** not containing B,*
2. *Γ: is the **tape alphabet**,*
3. *Q: is the finite set of **states**,*
4. *$q_0 \in Q$: is the **initial state**,*
5. *$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, is the **transition function**,*
6. *$B \in \Gamma$: is the **blank symbol**,*
7. *$F \subseteq Q$: is the set of **accepting** or **final states**.*

Instantaneous Description(ID): Gives the configuration of a TM at any given instant. It should give information about

1. the state of the Turing machine
2. the contents of the tape and
3. the position of the tape-head.

Hence, it is encoded by

$$w_1 q w_2$$

with the TM in state q scanning the first letter of w_2. Here w_1 is the portion of the string to the left of the tape-head starting from the first non-blank symbol, while w_2 is the portion of the string staring from the symbol being scanned by the tape-head till the last non-blank symbol. If the first/last non-blank symbol is to the right/left of the tape-head, then w_1/w_2 is λ/B.
For instance, the configuration

$\leftarrow \mid B \mid a_1 \mid B \mid a_2 \mid a_3 \mid B \rightarrow$
$- - - - - - - - - - \uparrow - - - - - - - - - -$
$. \qquad\qquad\qquad\qquad\qquad |q|$

is encoded by $a_1 B a_2 q a_3$.
Initial configuration or ID: Clearly, the starting configuration/ID is $q_0 w$, where w is the input string.

Final or accepting ID: If a configuration contains an accepting state then the ID is an accepting or final ID. For IDs I, J, if I yields J in one move of $\mathcal{M}$ then we write

$$I \vdash_{\mathcal{M}} J.$$

If ID I yields ID J in 0 or more steps then we write $I \vdash^*_{\mathcal{M}} J$.(If $\mathcal{M}$ is clear from the context we omit $\mathcal{M}$.) Thus $I \vdash^* J$ if $I = J$ or there is a sequence of ID's $I_0, I_1, \ldots, I_n$ such that $I_0 = I, I_n = J$ and for each $k < n, I_k \vdash I_{k+1}$.

A string $w \in \Sigma^*$ is accepted by $\mathcal{M}$, if starting from the initial configuration, in finitely many moves, $\mathcal{M}$ enters an accepting state.
The **language accepted by** $\mathcal{M}$ is

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^* : q_0 w \vdash^* I, \text{ for some accepting ID I}\}.$$

Thus $\mathcal{L}(\mathcal{M})$ is the set of all strings over Σ accepted by $\mathcal{M}$. Such languages are called **Turing-recognizable** or **recursively enumerable (re).**

Remark 1. We may w.l.g. assume that a TM $\mathcal{M}$ halts whenever it enters an accepting state. This may be done as follows. Add a new "halting state" q_h. Modify the transition function such that from any accepting state $\mathcal{M}$ enters the state q_h. Thus whenever $\mathcal{M}$ enters an accepting state, it then immediately enters q_h and halts i.e. it has no next move.

Definition 2. *A language is said to be* ***decidable*** *or* ***recursive*** *if it is accepted by a Turing machine that* ***halts on every input****.*

Simple examples:

Example 1. We shall construct a TM $\mathcal{M}$ that accepts all palindromes over the alphabet $\{0, 1\}$. We first given an informal description of $\mathcal{M}$. Given an input, $\mathcal{M}$ reads the first letter of the input and "marks" it. If it is a 0 (respectively 1) it marks it with an X (respectively Y). It then moves right skipping over the 0's and 1's until it finds the *last* unmarked letter. If it matches with the letter already marked, then it replaces it with an X or a Y as the case may be. It then moves left skipping over the 0's and 1's until it finds the first unmarked letter of the input. $\mathcal{M}$ then repeats the process. If after marking the last unmarked letter, $\mathcal{M}$ does not find any unmarked letter, then the input is an even palindrome and $\mathcal{M}$ accepts. If after marking the first unmarked symbol, $\mathcal{M}$ does not find any unmarked letter, then the input is an odd palindrome and $\mathcal{M}$ accepts.

Exercise 1. Give a formal definition of the transition function of $\mathcal{M}$.
Hence describe, formally, an accepting computation on input 10101

Remark 2. The TM constructed above halts on every inputs and hence the set of palindromes over $\{0, 1\}$ is recursive.

Exercise 2. Construct a Turing machine that accepts

$$\{ww : w \in \{0, 1\}^*\}.$$

1.1 Variants of TMs:

1. Multi-tracks single-tape TM.
2. Single-tape one-way infinite.
3. Multi-tape TM

1. **Multi-track TM:** In a k-track TM $\mathcal{M}$, at any given instant, the tape-head would be scanning the contents of all the cells in the tracks. Thus, $\mathcal{M}$ would be reading a k-tuple of symbols . Depending on the k-tuple being read, and the state of the finite control, $\mathcal{M}$ would print a k-tuple of symbols–one for each track–move the tape-head one position to the right or left and enters a, possibly, new state. The input string is placed onto the first track with all other cells being blank. Thus if $a \in \Sigma$ is identified with the k-tuple $(a, B, \ldots, B)$ and the blank is identified with $(B, \ldots, B)$, then a muti-track TM is simply a single-tape TM whose tape alphabet is Γ^k with blank being identified with $(B, \ldots, B)$.
2. Exercise: Show that a two-way infinite tape TM can be simulated by a one-way infinite single tape TM.
3. **Multi-Tape TM:** In a $k-$tape TM $\mathcal{M}$, there are k tapes and for each tape there is a tape-head scanning the content of a cell at each instant. Depending on the letters being scanned by the tape-heads and the state of the finite-control, in one move $\mathcal{M}$

1. prints a letter in each of the cell being scanned
2. moves the tape-heads–independent of each other– one cell to the right or left, and
3. enters a possibly new state.

The input string is placed on the first tape with the corresponding tape-head scanning the first cell and all other cells are blank. The TM $\mathcal{M}$ starts in the initial state q_0 and if after finitely many moves, $\mathcal{M}$ enters an accepting state, the input string is accepted by $\mathcal{M}$. The language accepted by $\mathcal{M}$ is

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma \mid \mathcal{M} \text{ accepts w}\}.$$

Thus the transition-function of a k-tape TM is a function of the form

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

Theorem 1. *Every multi-tape TM $\mathcal{M}$ can be simulated by a single tape TM $\hat{\mathcal{M}}$.*
Consequently, $\mathcal{L}$ is re iff it is accepted by a multi-tape TM.

Proof. Let $\mathcal{M}$ be a k-tape TM. We shall construct a single-tape TM $\hat{\mathcal{M}}$ with $2k$ tracks–2 tracks for each tape– to simulate $\mathcal{M}$. The upper track contains the contents of the corresponding tape, while the lower track contains a single marker that indicates the position of the corresponding tape-head. To simulate a move of $\mathcal{M}$, $\hat{\mathcal{M}}$ starts from the cell containing the leftmost marker in the state of $\mathcal{M}$.

$\hat{\mathcal{M}}$ then makes a sweep from left to right visiting all the cells containing a marker and also noting the letters above the markers. A part of the finite control counts the number of markers visited as well as the letters being "read" by the markers. When $\hat{\mathcal{M}}$ has visited all the cells containing a marker, it has enough information to simulate a move of $\mathcal{M}$. $\hat{\mathcal{M}}$ then makes a leftward journey updating the letters above the markers and also moving the marker either to the left or to the right in order to simulate the move of $\mathcal{M}$. $\hat{\mathcal{M}}$ then enters the state to which $\mathcal{M}$ enters. $\hat{\mathcal{M}}$ accepts whenever $\mathcal{M}$ accepts.□

Exercise 3. Show that to simulate n moves of $\mathcal{M}$, $\hat{\mathcal{M}}$ requires $O(n^2)$ moves.

Non-deterministic TM: In a non-deterministic TM $\mathcal{M}$, at each instant, the TM –depending on the letter being scanned and the state of the finite control–has several choices for the next move. Thus the transition function is a function of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

The equation

$$\delta(q, a) = \{(p_1, a_1, D_1), \ldots, (p_k, a_k, D_k)\}$$

means that the TM in state q, reading the letter a, has the choice of printing a_i in place of a, enter the state p_i and move the tape-head in the direction D_i, where i is one of $1, 2, \ldots, k$.

A string w is accepted by $\mathcal{M}$ if, starting from the initial configuration, $\mathcal{M}$ has a choice of moves that will lead to an accepting configuration.

We shall now show that a non-deterministic TM is no more powerful that a deterministic one.

Theorem 2. *Every non-deterministic Turing machine $\mathcal{M}$ can be simulated by a deterministic Turing machine $\hat{\mathcal{M}}$.*

Consequently, a language $\mathcal{L}$ is re iff it is accepted by non-deterministic Turing machine.

Proof. Let us view $\mathcal{M}$'s computation on w as a tree where each node is labelled by a configuration, while the children of a node, labelled by I, are labelled by the all the possible configurations that $\mathcal{M}$ can enter in one move from the configuration I. The root is labelled by the initial configuration. The TM $\hat{\mathcal{M}}$ explores the tree by using the breath-first search. As soon as $\hat{\mathcal{M}}$ finds an accepting configuration, it halts and accepts.

Thus, $\hat{\mathcal{M}}$ consists of three tapes. The first tape is the input tape, while the second tape is the simulation or computation tape. The third tape is an address tape. Suppose each node of the tree has at most k children. Then each node is assigned an address which is a string over the alphabet $\{1, 2, \ldots, k\}$. For instance 231 is assigned to the node we arrive at by starting at the root, then going to its 2nd child, then going to that node's 3rd child and finally going to that node's 1st child. The TM $\hat{\mathcal{M}}$ behaves as follows. Initially, the first tape contains the input w, while the other tapes are blank.

1. Copy the input w onto the second tape.
2. Simulate $\mathcal{M}$ on w as dictated by the address on the 3rd tape, aborting if the address is invalid. Accept the input string if the configuration on the node visited is accepting.
3. Replace the string on the 3rd tape with the lexicographically next string and go to step 2.

Clearly, $\hat{\mathcal{M}}$ simulates $\mathcal{M}$. $\square$

Exercise 4. Show that to simulate n moves of $\mathcal{M}$, $\hat{\mathcal{M}}$ requires about $O(2^n)$ steps.

1.2 Properties of re and recursive languages.

Theorem 3. *A language $\mathcal{L} \subseteq \Sigma^*$ is recursive iff $\mathcal{L}$ and $\mathcal{L}^C$ are both re.*

Proof. Suppose $\mathcal{L}$ is recursive. Fix a TM $\mathcal{M}$ that accepts $\mathcal{L}$ and halts on all inputs. Clearly $\mathcal{L}$ is re. Let $\mathcal{M}'$ be the TM obtained from $\mathcal{M}$ by interchanging the accepting and rejecting states.Then clearly, $\mathcal{M}'$ accepts $\mathcal{L}^C$.

Conversely, suppose both $\mathcal{L}$ and $\mathcal{L}^C$ are re. Fix single-tape TMs $\mathcal{M}_1, \mathcal{M}_2$ accepting $\mathcal{L}, \mathcal{L}^C$ respectively. W.l.g. assume that both $\mathcal{M}_1$ and $\mathcal{M}_2$ halt whenever they enter an accepting state i.e. both $\mathcal{M}_1, \mathcal{M}_2$ halt on acceptance. Construct a two-tape TM $\hat{\mathcal{M}}$ as follows.

On input w, $\hat{\mathcal{M}}$ copies w onto the second tape and then runs $\mathcal{M}_1$ on tape-1 and $\mathcal{M}_2$ on tape-2. $\hat{\mathcal{M}}$ accepts whenever $\mathcal{M}_1$ accepts and halts when one of them halts. Clearly, $\hat{\mathcal{M}}$ accepts $\mathcal{L}$ and for any input w either $w \in \mathcal{L}$ or $w \in \mathcal{L}^C$ and hence one of $\mathcal{M}_1, \mathcal{M}_2$ halts on input w. So $\hat{\mathcal{M}}$ halts on any input. Hence, $\mathcal{L}$ is recursive. $\square$

Theorem 4. *Recursive/re languages are closed under finite $\bigcup$ and $\bigcap$.*
Recursive languages are closed under complementation.

Proof. We shall prove closure under finite $\bigcup$ for re. The remaining proofs are similar.

Let $\mathcal{L}_1, \mathcal{L}_2$ be re languages. Fix single-tape TMs $\mathcal{M}_1, \mathcal{M}_2$ accepting $\mathcal{L}_1, \mathcal{L}_2$ respectively. We shall construct a two-tape TM $\hat{\mathcal{M}}$ as follows.

On input w, $\hat{\mathcal{M}}$ copies w onto the second tape and then runs $\mathcal{M}_1$ on tape-1 and $\mathcal{M}_2$ on tape-2. $\hat{\mathcal{M}}$ accepts whenever one of $\mathcal{M}_1, \mathcal{M}_2$ accepts. Clearly, $\hat{\mathcal{M}}$ accepts $\mathcal{L}_1 \bigcup \mathcal{L}_2$. $\square$

1.3 Undecidability

Encoding TMs: Let $\mathcal{M} =< \{0,1\}, \mathit{\Gamma}, Q, q_1, \delta, B, F >$ be a Turing machine over the alphabet $\{0,1\}$. Let

$$Q = \{q_1, q_2, \ldots, q_n\},$$

where q_1 is the initial state and q_2 is the only accepting state. Let

$$\mathit{\Gamma} = \{X_1, X_2, X_3, \ldots, X_k\},$$

for some integer k, where $X_1 = 0, X_2 = 1$ and $X_3 = B$. Let $D_0 = L, D_1 = R$. Then each move or transition of a TM $\mathcal{M}$ is given by an equation of the form

$$\delta(q_i, X_j) = (q_k, X_l, D_m), \tag{1}$$

where $1 \leq i, k \leq n, 1 \leq j, l \leq k$ and $m = 1, 2$. This can be encoded by the binary string

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

The collection of these binary encodings completely describes δ and can be combined to give an encoding of the TM $\mathcal{M}$ as follows.
Let $C_1, C_2, \ldots, C_m$ be the encodings of all the transitions of $\mathcal{M}$. Then a code of the entire TM $\mathcal{M}$ is

$$C_1 11 C_2 11 \ldots 11 C_m$$

and is denoted by $< \mathcal{M} >$.

This also gives an enumeration of all TM's over the alphabet $\{0,1\}$ as follows.
Let $w_1, w_2, \ldots$ be the canonical enumeration of all binary strings. Thus $w_1 = 0, w_2 = 1, w_3 = 00, w_4 = 01, w_5 = 10$ and so on. Then the ith Turing machine, denoted by $\mathcal{M}_i$, is the Turing machine over $\{0,1\}$ whose code is w_i. If w_i is not a code of a TM, then $\mathcal{M}_i$ is a fixed TM $\mathcal{M}^*$ such that $\mathcal{L}(\mathcal{M}^*) = \phi$. Note that $< \mathcal{M}_i >= w_i$, if w_i is a code of a TM.

Exercise 5. Find an algorithm $\mathcal{A}$ which when given an input a binary string w outputs an integer i such tha $w = w_i$.

What is the complexity of your algorithm?

Define the *diagonalization language* L_d by

$$\mathcal{L}_d = \{w_i : \mathcal{M}_i \text{ does not accept w}_\text{i}\}.$$

Theorem 5. *The language L_d is not recursively enumerable*

Proof. Suppose L_d is recursively enumerable. Then there is a TM, say $\mathcal{M}_{i^*}$, such that

$$\mathcal{L}(\mathcal{M}_{i^*}) = L_d.$$

Now consider the string w_{i^*}. If $w_{i^*} \in L_d$, then by our assumption, $\mathcal{M}_{i^*}$ accepts w_{i^*}. This implies that $w_{i^*} \notin L_d$, a contradiction. On the other hand, if $w_{i^*} \notin L_d$, then $\mathcal{M}_{i^*}$ does not accepts w_{i^*}. By definition of L_d, this means that $w_{i^*} \in L_d$, again a contradiction. Thus L_d cannot be re. □

Universal language: Let

$$\mathcal{L}_U = \{<\mathcal{M}> 111w : \mathcal{M} \text{ accepts w}\}.$$

We shall show that $\mathcal{L}_U$ is re but **not** recursive. The language $\mathcal{L}_U$ is called a universal language and the TM accepting $\mathcal{L}_U$ is called a universal TM.

Theorem 6. *The language $\mathcal{L}_U$ is recursively enumerable.*

Proof. We shall construct a 3-tape TM $\mathcal{M}_U$ that accepts $\mathcal{L}_U$. $\mathcal{M}_U$ works as follows.

1. $\mathcal{M}_U$ first checks that the input is of the form $<\mathcal{M}> 111w$. If not, then $\mathcal{M}_U$ rejects the input string.
2. $\mathcal{M}_U$ then copies w onto the second tape. Note that w is the string that follows the first block of 111.
3. The third tape contains a string of 0's with 0^i representing the state q_i. Initially, the third tape consists of 0 to represent the initial state q_1.
4. To simulate a move of $\mathcal{M}$, $\mathcal{M}_U$ searches within $<\mathcal{M}>$ a substring of the form $0^i10^j10^k10^l10^m$, where 0^i is the string on the 3rd tape and X_j is the symbol of $\mathcal{M}$ at the position on tape 2 scanned by $\mathcal{M}_U$. This represents the transition that $\mathcal{M}$ would next make. Thus $\mathcal{M}_U$ behaves as follows.
   (a) Change the contents of tape 3 to 0^k i.e. simulate the state change of $\mathcal{M}$
   (b) Replace X_j by X_l i.e. change the tape symbol of $\mathcal{M}$,
   (c) Shift the tape head on tape 2 one cell to the left if $m = 1$ or one cell to the right if $m = 2$. Thus $\mathcal{M}_U$ simulates the move of $\mathcal{M}$ to the left or to the right.
5. If no such substring is found, then $\mathcal{M}$ has no next move and hence halts in the simulated configuration. $\mathcal{M}_U$ does likewise.
6. If $\mathcal{M}$ enters the accepting state, then the contents of tape 3 is 00 and so $\mathcal{M}_U$ accepts.

In this way, $\mathcal{M}_U$ simulates $\mathcal{M}$ on w and accepts $<\mathcal{M}> 111w$ iff $\mathcal{M}$ accepts w. □

Theorem 7. *The universal language $\mathcal{L}_U$ is not recursive.*
Consequently, the class of recursive languages is strictly contained in the class of re languages.

Proof. Suppose $\mathcal{L}_U$ is recursive. Then $\mathcal{L}_U^C$ is re. Hence there is a TM $\mathcal{M}$ accepting $\mathcal{L}_U^C$. Using $\mathcal{M}$ we shall construct a TM $\mathcal{M}'$ that accepts $\mathcal{L}_d$. Given input w_i, $\mathcal{M}'$ first checks if w_i is a code of a TM. If it is not a code, then $\mathcal{M}_i = \mathcal{M}^*$ and hence $\mathcal{M}_i$ doe not accept w_i and so $\mathcal{M}'$ accepts w_i. Suppose w_i is a code, then it is a code of the ith TM $\mathcal{M}_i$. The TM $\mathcal{M}'$ then runs $\mathcal{M}$ on w_i111w_i. If $\mathcal{M}$ accepts, then $\mathcal{M}'$ accepts w_i. Note that $\mathcal{M}$ accepts w_i111w_i iff $<\mathcal{M}_i> 111w_i \notin \mathcal{L}_U$ iff $\mathcal{M}_i$ does not accepts w_i. Thus $\mathcal{M}_i$ accepts $\mathcal{L}_d$. This contradicts the fact that $\mathcal{L}_d$ is not re. Thus $\mathcal{L}_U$ can not be recursive.□