Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lecture 0

Instructor: Goutam Biswas *Autumn Semester 2023*

1 Turing Machine

A *Turing Machine (TM)* is a primitive computing device which can compute anything that is *computable*. Its data structure is a string over a finite set of characters (alphabet). Its program (at a state) uses a *read-write head (cursor)* to read a character from its current position, write a character in the same position. After that the cursor may be shifted one position to left or right or may stay stationary on the string. The machine may enter a new state of the program.

Mathematically a Turing machine M is specified by 4-tuple of data, $M = (Q, \Sigma, \delta, s)$.

(a) Q is a finite set of states.

(b) Σ is the finite alphabet of the strings. It includes two special symbols, '$\sqcup$' (blank) and '$\triangleright$' (left end-marker).

(c) The program is represented by the function

$$\delta : Q \times \Sigma \to (Q \cup \{h, Y, N\}) \times \{\leftarrow, \rightarrow, -\},$$

where 'h' is halting state, 'Y' is accepting halt state, 'N' is rejecting halt state; '$\leftarrow$', head moves left, '$\rightarrow$', head moves right, '$-$', head is stationary.

(d) s is the start state of the TM.

Sometime people talk about two alphabets, an *input alphabet* (Σ) and a *tape alphabet* (Γ), where $\Sigma \subset \Gamma$.

The program δ of the machine specifies its dynamics. $\delta(p, \sigma) = (q, \gamma, D)$: at the current state p, if M reads σ at the current position of the head, it overwrites σ by γ (γ may be same as σ), the head moves according to $D \in \{\leftarrow, \rightarrow, -\}$ and the machine goes to a new state q (q may be same as p).

There is a special requirement in δ for the symbol '$\triangleright$'. $\delta(p, \triangleright) = (q, \triangleright, \rightarrow)$, for all $p \in Q$. This stops the machine to go beyond the leftmost end of the

string. If the machine goes beyond the right end of the input, it sees a *blank* ($\sqcup$) and it may be overwritten and the string increases in length.

Input to the machine is "$\triangleright x$", where $x \in (\Sigma \setminus \{\triangleright, \sqcup\})^*$. The start state is s and the initial head position is '$\triangleright$'.
A configuration of a machine is (p, x, u), where the machine is in state p, the current position of the head is at the last symbol of x, u is the string to the right of the head (last symbol of u should not be '$\sqcup$').

We define a *binary relation* '$\rightarrow_M$' over the space of configurations of M. The TM M goes in one step from the configuration (p, x, u) to (q, y, v) and we write $(p, x, u) \rightarrow_M (q, y, v)$ when $\delta(p, a) = (q, b, D)$. Let $x = x'a$, one of the following is the situation.

- If $a = \triangleright$, then $b = \triangleright$ and $D = \rightarrow$.
  If $u = \varepsilon$, then $y = x\sqcup$ and $v = \varepsilon$,
  If $u = cu'$, then $y = xc$ and $v = u'$.
- If $a \neq \triangleright$ and $D = \rightarrow$.
  If $u = \varepsilon$, then $y = x'b\sqcup$ and $v = \varepsilon$,
  If $u = cu'$, then $y = x'bc$ and $v = u'$.
- If $a \neq \triangleright$ and $D = \leftarrow$,
  then $y = x'$ and $v = bu$.
- If $a \neq \triangleright$ and $D = -$,
  then $y = x'b$ and $v = u$.

If there are $k + 1 \geq 1$ configurations such that

$$(p, x, u) = (p_0, x_0, u_0) \rightarrow_M (p_1, x_1, u_1) \rightarrow_M \cdots \rightarrow_M (p_k, x_k, u_k) = (q, y, v),$$

we say that starting from the configuration (p, x, u), the TM reaches the configuration (q, y, v) in $k \geq 0$ steps and denote it as

$$(p, x, u) \rightarrow_M{}^* (q, y, v) \text{ or simply } (p, x, u) \rightarrow^* (q, y, v).$$

The start configuration of the TM $M = (Q, \Sigma, \delta, s)$ is $(s, \triangleright, x)$ where $x \in (\Sigma \setminus \{\triangleright, \sqcup\})^*$. There are three possible halting configurations of our brand of machine: $(h, u, v), (Y, u, v)$ and (N, u, v). But a TM may not halt at all.

Let $x \in \Sigma \setminus \{\triangleright, \sqcup\}$, $x \in L(M)$ if $(s, \triangleright, x) \rightarrow_M{}^* (Y, x_1, u_1)$, $x \notin L(M)$ if $(s, \triangleright, x) \rightarrow_M{}^* (N, x_1, u_1)$.

We also have to fix up a convention about the value of the function computed by M i.e. $(s, \triangleright, x) \rightarrow_M{}^* (h, x_1, u')$. We may have $f(x) = M(x) = x'u'$, where $x_1 = \triangleright x'$ and there is no blanks ($\sqcup$) at the end of u'.

Example 1. $M = (\{s\}, \{0, 1, \triangleright, \sqcup\}, \delta, s)$ is a very simple TM. The state transition table is as follows.

$p \in Q$	$\sigma \in \Sigma$	$\delta(p, \sigma) = (q, \gamma, D)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(s, 1, \rightarrow)$
s	1	$(s, 0, \rightarrow)$
s	$\sqcup$	$(h, \sqcup, \rightarrow)$

It finds the 1's complement of a binary string.

Exercise 1. Design a TM that does not halt on any input. Let the input alphabet be $\{0,1\}$.
Exercise 2. Design a TM that takes input $x \in \{0,1\}*$. It halts only after getting four 1's.
Exercise 3. What does the TM $M = (\{s, q_0, q_1\}, \{0, 1, \triangleright, \sqcup\}, \delta, s)$ compute on input $\triangleright x$, where $x \in \{0,1\}^+$?

$p \in Q$	$\sigma \in \Sigma$	$\delta(p,\sigma) = (q,\gamma,D)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s	1	$(s, 1, \rightarrow)$
s	$\sqcup$	$(q_0, \sqcup, \leftarrow)$
q_0	0	$(q_0, 0, \leftarrow)$
q_0	1	$(q_1, 1, \leftarrow)$
q_0	$\triangleright$	$(h, \triangleright, \rightarrow)$
q_1	0	$(q_1, 1, \leftarrow)$
q_1	1	$(q_1, 0, \leftarrow)$
q_1	$\triangleright$	$(h, \triangleright, \rightarrow)$

Exercise 4. What does the TM $M = (\{s, q_1, q_2, q_3\}, \{0, 1, \triangleright, \sqcup\}, \delta, s)$ compute on input $\triangleright x$, where $x \in \{0,1\}^+$?

$p \in Q$	$\sigma \in \Sigma$	$\delta(p,\sigma) = (q,\gamma,D)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s	$\sqcup$	$(N, \sqcup, \leftarrow)$
s	1	$(q_1, 1, \rightarrow)$
q_1	1	$(q_1, 1, \rightarrow)$
q_1	0	$(q_2, 0, \rightarrow)$
q_1	$\sqcup$	$(N, \sqcup, \leftarrow)$
q_2	0	$(s, 0, \rightarrow)$
q_2	1	$(q_3, 1, \rightarrow)$
q_2	$\sqcup$	$(N, \sqcup, \leftarrow)$
q_3	0	$(q_3, 0, \rightarrow)$
q_3	1	$(q_3, 1, \rightarrow)$
q_3	$\sqcup$	$(Y, \sqcup, \leftarrow)$

Exercise 5. Design a TM that takes the input $\triangleright \sqcup^k x$, where $k \geq 0$, $x \in \{0,1\}^+$ and produces the output $\triangleright \sqcup^{k+1} x$.
Exercise 6. Design a TM that takes the input $\triangleright x$, where $x \in \{0,1\}^+$ and produces the output $\triangleright y$, where $y = 2 \times x$. Interpret x as an unsigned binary numeral.
Exercise 7. Design a TM that takes the input $\triangleright x$, where $x \in \{0,1\}^*$. It accepts x if $x = 0^n 1^n$, $n \geq 0$. Otherwise it rejects x.
Exercise 8. What is the estimate of the number of steps (state transitions) of the machine for an input $0^n 1^n$?

The question is can we do it better. The answer is positive, but the TM will be slightly more complicated and instead of giving the precise description of the machine we provide the outline of the algorithm. But before that an example.
Example 2. Input: $\triangleright$ 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

(a) We first check whether there is any 0 after 1. This will take $O(n)$ steps.

(b) Repeat the following steps as long as there is some 0 or 1 is left in the string.

(i) Check whether the parity of 0 and 1 (together) is even. *reject* if odd. This also takes $O(n)$ steps.

(ii) Cross off alternate 0's starting with the first one. Also cross off alternate 1's starting with the first one. This reduces the number of 0's and 1's to half in each step.

$\triangleright$ 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$\triangleright$ $\not\emptyset$ 0 $\not\emptyset$ 0 $\not\emptyset$ 0 $\not\emptyset$ 0 $\not1$ 1 $\not1$ 1 $\not1$ 1 $\not1$ 1
$\triangleright$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ 0 $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ 0 $\not1$ $\not1$ $\not1$ 1 $\not1$ $\not1$ $\not1$ 1
$\triangleright$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ 0 $\not1$ $\not1$ $\not1$ $\not1$ $\not1$ $\not1$ 1
$\triangleright$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not\emptyset$ $\not1$ $\not1$ $\not1$ $\not1$ $\not1$ $\not1$ $\not1$
Each pass takes $O(\log n)$ steps.

(c) If there is no 1 or 0 on the tape, *accept*, otherwise *reject*.

So the number of steps are $O(n) + O(n \log n) = O(n \log n)$.

Can we do even better. The answer is **negative** on our single string TM. If the number of steps required to decide a language L on a single tape TM is $o(n \log n)$, then L is a *regular language*. But our language $L = \{0^n 1^n : n \geq 0\}$ is known to be non-regular (context-free) language. So $O(n \log n)$ is the best on this model.

Definition 1. Let the language $L \subseteq (\Sigma \setminus \{\sqcup, \triangleright\})^*$. If there is a TM M, that on input $\triangleright x$ ($x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$) halts at state Y if $x \in L$, and halts at state N if $x \notin L$, then the language L is called *Turing decidable* or *recursive* language.

A language $L \subseteq (\Sigma \setminus \{\sqcup, \triangleright\})^*$ is *Turing recognizable* or *recursively enumerable* if there is a TM M that on input $\triangleright x$ halts at state Y if $x \in L$, but either halts at state N or may not halt at all if $x \notin L$.

From the definition it is clear that the set of *recursive* languages is a subset of *recursively enumerable* languages. In fact it is a proper subset. The language

$$L_H = \{< M, d >: \ M \text{ is a Turing machine which halts on input } d\}.$$

is *recursively enumerable* (*Turing recognizable*) but not *recursive* (not *Turing decidable*).

2 Turing Machine with Multiple Tapes/Strings

There are different variants of a Turing machine. In terms of absolute computing power all these models are equivalent. But in terms of resource restricted computation their powers are different (at least it seems to be).

Definition 2. A k-tape TM, $M = (Q, \Sigma, \delta, s)$, $k \geq 1$, has k independent strings, each equipped with its independently movable read/write head. Actions of each head are controlled by the function (program) δ.

$$\delta : Q \times \Sigma^k \to (Q \cup \{h, Y, N\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k.$$

$\delta(p, \sigma_1, \cdots, \sigma_k) = (q, (\gamma_1, D_1), \cdots, (\gamma_k, D_k))$, where p is the current state, $\sigma_1, \cdots, \sigma_k$ are input read from k tapes, q is the next state, $\gamma_1, \cdots, \gamma_k$ are symbols written on k tapes, and $D_1, \cdots, D_k$ are movements of head on k tapes.

A *configuration* is a $2k+1$-tuple: $(p, x_1, u_1, \cdots, x_k, u_k)$, where p is the current state, the i^{th} string has $x_i u_i$. The i^{th} head is at the rightmost symbol of x_i. The *start configuration* is $(s, \triangleright, x, \triangleright, \varepsilon, \cdots, \triangleright, \varepsilon)$, the actual input is x on the first tape. Other tapes are empty except the left-end sentinel '$\triangleright$'.

An input x is *accepted* i.e. $x \in L(M)$, if

$$(s, \triangleright, x, \triangleright, \varepsilon, \cdots, \triangleright, \varepsilon) \to_M^* (Y, x_1, u_1, \cdots, x_k, u_k).$$

An input x is *rejected* i.e. $x \notin L(M)$ if

$$(s, \triangleright, x, \triangleright, \varepsilon, \cdots, \triangleright, \varepsilon) \to_M^* (N, x_1, u_1, \cdots, x_k, u_k).$$

For computation of a function we need to fix-up a convention where the output of $M(x)$ will be.

Exercise 9. Show that $L = \{0^n 1^n : n \geq 0\}$ can be decided in $O(n)$ steps by a 2-tape TM.

3 Nondeterministic Turing Machine

A *Nondeterministic Turing machine (NTM)* is a not so realistic model of computation. It is also equivalent to a TM in terms of the absolute power of computation. But in terms of efficiency it appears to be exponentially more powerful. Unfortunately it is unknown whether this exponential speedup is intrinsic to this model or due to lack of understanding of the nature of *nondeterminism*. This gives rise to the famous question of **P** verses **NP**.

An NTM $N = (Q, \Sigma, \Delta, s)$ has same three data Q, Σ and s. But $\Delta \subseteq (Q \times \Sigma) \times (Q \cup \{h, Y, N\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\})$ i.e. Δ is a *relation*.

Given a pair of current state and input, the machine may have zero, one or more number of next state, output and cursor movement triples. A *configuration* of an NTM is similar to a *configuration* of a TM. But for a TM, from a given configuration there is at most one (TM may be incompletely specified) next configuration, and a computation of TM forms a finite or infinite (non-terminating) sequence of configurations. But in an NTM a computation forms a *tree of configurations* with the *start configuration* at the *root*.

An input $x \in (\Sigma \backslash \{\triangleright, \sqcup\})^*$ is *accepted* by an NTM N if there exists a sequence of configurations starting from $(s, \triangleright, x)$ reaches a configuration (Y, y, u). Other sequences starting from $(s, \triangleright, x)$ may reach (N, z, v) or even may not halt.

A language $L \subseteq (\Sigma \setminus \{\triangleright, \sqcup\})^*$ is *decided* by an NTM N, if for any $x \in (\Sigma \setminus \{\triangleright, \sqcup\})^*$, $x \in L$ if and only if $(s, \triangleright, x) \to_N^* (Y, y, u)$.
This essentially says that if $x \in L$, there is a sequence of nondeterministic choices that leads to an *accept halt*. But if $x \notin L$, no sequence of choice can leads to *accept halt*.

References

[CHP] Computational Complexity by *Christos H Papadimitriou*, Pub. Addision-Wesley, 1994, ISBN 0-201-53082-1.

[MS] *Theory of Computation* by Michael Sipser, (3rd. ed.) Pub. Cengage Learning, 2013, ISBN 978-81-315-2529-6.

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.

[DCK1] *Theory of Computation* by Dexter C Kozen, Pub. Springer, 2006, ISBN 978-81-8128-696-3.

[RMK] R M Karp, *Reducibility Among Combinatorial Problems*, in R E Miller and J W Thatcher, ed. *Complexity of Computer Communications*, pp 85-103, Plenum, 1972.

[NPMJF] Nicholas Pippenger, And Michael J Fischer, *Relation Among Complexity Measures*, in JACM 26, 2 (April 1979), pp 361-381.

[WJS] Walter J Savitch, *Relationships between nondeterministic and deterministic tape complexities*, Journal of Computer and System Sciences 4 (2), pp 177-192 1970.