CS304: Automata and Formal Languages

Lec 23

Encoding and Variations of TMs

Rachit Nimavat

October 13, 2025

Outline

What is a Turing machine (TM)?

Definition

A Turing machine (TM) M is a 6-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, H)$, where,

1. Q: A finite set (set of states).
2. Σ: A finite set (input alphabet). Σ excludes $\triangleright, \Box, \leftarrow, \rightarrow$.
3. Γ: A finite set (tape alphabet).
   $\Sigma \cup \{\triangleright, \Box\} \subseteq \Gamma$. Γ excludes $\leftarrow, \rightarrow$.
4. $\delta : (Q - H) \times \Gamma$ to $Q \times (\Gamma \cup \{\leftarrow, \rightarrow\})$ is the transition function such that the tape head never falls off or erases $\triangleright$ symbol
   ▷ Time (computation)
5. q_0: The start state (belongs to Q).
6. $H = \{q_{\text{acc}}, q_{\text{rej}}\}$: The set of halting states (subset of Q).

Some notes on the Turing machine

Symbols

- $\triangleright$: Left end symbol
- $\square$: Blank symbol
- $\leftarrow, \rightarrow$: Left and right movement symbols
- Σ : Represents input/output/special symbols
- Γ : Represents symbols that can be present on the tape

Transition

- M never falls off the left end of the tape i.e.,
  when the current symbol is $\triangleright$, the tape head has to move right
- M stops when it reaches an accept or a reject state i.e.,
  δ is not defined for states in H

Construct TM for accepting $L = \{a^n b^n c^n \mid n \geq 1\}$

Problem

- Construct a Turing machine to accept all strings from the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Construct TM for accepting $L = \{a^n b^n c^n \mid n \geq 1\}$

Problem

- Construct a Turing machine to accept all strings from the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Solution

Language $A = \{abc, aabbcc, aaabbbccc, \ldots\}$

- No DFA can accept this language. $\triangleright$ Use pumping lemma
- No CFG can accept this language. $\triangleright$ Use pumping lemma
- A TM accepts this language.

Construct TM for accepting $L = \{a^n b^n c^n \mid n \geq 1\}$

Problem

- Construct a Turing machine to accept all strings from the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Solution (continued)

State	Tape								
q_0	$\triangleright$	a	a	b	b	c	c	$\square$	$\cdots$
q_2	$\triangleright$	x	a	b	b	c	c	$\square$	$\cdots$
q_3	$\triangleright$	x	a	y	b	c	c	$\square$	$\cdots$
q_4	$\triangleright$	x	a	y	b	z	c	$\square$	$\cdots$
q_2	$\triangleright$	x	x	y	b	z	c	$\square$	$\cdots$
q_3	$\triangleright$	x	x	y	y	z	c	$\square$	$\cdots$
q_4	$\triangleright$	x	x	y	y	z	z	$\square$	$\cdots$
q_5	$\triangleright$	x	x	y	y	z	z	$\square$	$\cdots$
q_{acc}	$\triangleright$	x	x	y	y	z	z	$\square$	$\cdots$

Construct TM for accepting $L = \{a^n b^n c^n \mid n \geq 1\}$

Problem

- Construct a Turing machine to accept all strings from the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Solution (continued)

- $\Gamma = \Sigma \cup \{\triangleright, \square, x, y, z\}$
- Cells with "$-$" means that the TM terminates in q_{rej} state

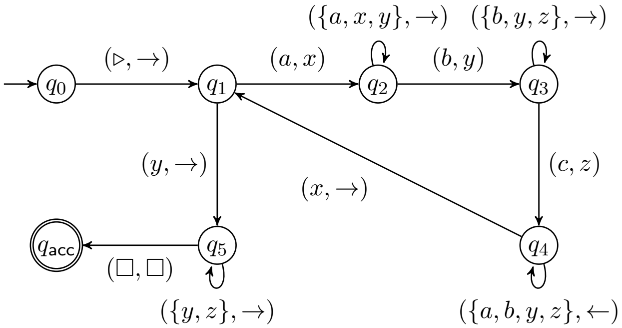
	Current symbol (Γ)							
St.	$\triangleright$	a	b	c	x	y	z	$\square$
q_0	$(q_1, \rightarrow)$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
q_1	$-$	(q_2, x)	$-$	$-$	$-$	$(q_5, \rightarrow)$	$-$	$-$
q_2	$-$	$(q_2, \rightarrow)$	(q_3, y)	$-$	$(q_2, \rightarrow)$	$(q_2, \rightarrow)$	$-$	$-$
q_3	$-$	$-$	$(q_3, \rightarrow)$	(q_4, z)	$-$	$(q_3, \rightarrow)$	$(q_3, \rightarrow)$	$-$
q_4	$-$	$(q_4, \leftarrow)$	$(q_4, \leftarrow)$	$-$	$(q_1, \rightarrow)$	$(q_4, \leftarrow)$	$(q_4, \leftarrow)$	$-$
q_5	$-$	$-$	$-$	$-$	$-$	$(q_5, \rightarrow)$	$(q_5, \rightarrow)$	$(q_{\text{acc}}, \square)$

Construct TM for accepting $L = \{a^n b^n c^n \mid n \geq 1\}$

Problem

- Construct a Turing machine to accept all strings from the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Solution (continued)



Why So Complicated?!

Turing Machine is a very low-level model of computation.

It is designed to be as simple as possible while still being able to perform any computation that can be done by a computer.

The complexity of the examples arises from the need to explicitly manage the tape and head movements, which are abstracted away in higher-level models like programming languages.

The goal is to understand the fundamental capabilities and limitations of computation, which requires working with this minimalistic model.

We will **not focus on designing Turing Machines for specific tasks**, but rather on **understanding what can and cannot be computed in principle**.

How are TM's different from DFA's and PDA's?

Feature	DFA	PDA	TM
Memory size	Finite	Infinite	Infinite
Halts?	✓	✓	✓, ✗
Input scanning	Left-to-right	Left-to-right	Arbitrary
#Passes	1 pass	1 pass	Any
Halting	End of input	End of input	Accept state
Computing power	Least	Medium	Highest
Language recognizer?	✓	✓	✓
Function calculator?	✗	✗	✓
Decide RL's?	✓	✓	✓
Decide CFL's?	✗	✓	✓
Decide REL's?	✗	✗	✓

Encoding TMs

How to represent a TM M as a natural number?

Every TM M can be converted to a TM with:

- States are $\{q_1, \ldots, q_k\}$ for some $k \in \mathbf{N}$
- q_1 is start States
- q_2 is unique accept state
- q_3 is unique reject state

To encode TM, only need to write transitions

Encode as hence the encoding is a string over $\{1, \ldots, k\} \cup \Sigma \cup \Gamma \cup \{\leftarrow, \rightarrow, \#\}$.

Can also convert this encoding to a natural number denoted by $< M >$ (hides the specifics about the convention used to get the number).

Each TM encoded by a unique natural number, and each natural number encodes a TM (not necessarily a valid TM). Convention: treat invalid TMs as rejecting all inputs.

In this course, we use $< M >$ to denote the encoding of TM M as a natural number and by M_n to denote the TM encoded by natural number n.

Terminology

Consider a TM M.

M **accepts** a string w if M halts in an accept state on input w.

M **rejects** a string w if M halts in a reject state on input w.

M **loops** on a string w if M neither accepts nor rejects w (i.e., it runs forever).

M **does not accept** w if M either rejects or loops on w.

M **does not reject** w if M either accepts or loops on w.

M **halts** on w if M either accepts or rejects w.

Recognizers and Recognizability

Consider a TM M and a language L over Σ.

- M is a **recognizer** for L if for all $w \in \Sigma^*$, $w \in L$ if and only if (written, iff) M accepts w.
- L is **recognizable** if M is a recognizer for L.
- Can you use a recognizer M to verify that a string w is in L?
- Can you use a recognizer M to verify that a string w is not in L?
- Recall, **Recursively Enumerable (RE)** languages are the same as **Recognizable** languages.

Deciders and Decidability

Consider a TM M and a language L over Σ.

- M is a **decider** for L if for all $w \in \Sigma^*$ (i) M halts on input w; and (ii) $w \in L$ iff M accepts w.
- L is **decidable** if M is a decider for L.
- Can you use a decider M to verify that a string w is in L?
- Can you use a decider M to verify that a string w is not in L?
- **Recursive (R)** languages are the same as **Decidable** languages.

R and RE

Every decider is also a recognizer.

This means $R \subseteq RE$

Huge **open question**: Is $R = RE$?

Most believe $R \neq RE$. Just confirming answers should be easier than solving them.

Multitape TMs

A Multitape Turing Machine is a variation of the standard TM that has multiple tapes, each with its own independent read/write head.

- Formally, a k-tape Turing Machine has k tapes and k heads.
- The transition function is modified to read symbols from all k tapes and write symbols to all k tapes in a single step.
- The machine can move each head independently (left, right, or stay).
- Computationally equivalent to standard single-tape TM. Having 2 infinite RAMs is same as having 1 infinite RAM.
- More intuitive for designing algorithms
- Many other equivalent variants of TMs exist. (See, Chapter 8 of ALC.)