# CS304: Automata and Formal Languages

## Lec 24

Universal TMs

Rachit Nimavat

October 16, 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. $\Sigma$: A finite set (input alphabet). $\Sigma$ excludes $\triangleright, \Box, \leftarrow, \rightarrow$.
3. $\Gamma$: A finite set (tape alphabet).
   $\Sigma \cup \{\triangleright, \Box\} \subseteq \Gamma$. $\Gamma$ 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
   $\triangleright$ 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
- $\Sigma$ : Represents input/output/special symbols
- $\Gamma$ : 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.,
  $\delta$ is not defined for states in $H$

# 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 $\{q_1, \dots, q_k\}$ for some $k \in \mathsf{N}$
- $q_1$ 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 $\{1, \dots, k\} \cup \Sigma \cup \Gamma \cup \{\leftarrow, \rightarrow, \#\}$.

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 $n$.

# Recognizers and Recognizability

Consider a TM $M$ and a language $L$ over $\Sigma$.

- $M$ 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 $L$.
- Recall, **Recursively Enumerable (RE)** languages are the same as **Recognizable** languages.

# Deciders and Decidability

Consider a TM $M$ and a language $L$ over $\Sigma$.

- $M$ 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 $L$.
- **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.

# Universal Turing machine (UTM)

**Definition**

- A Universal Turing machine (UTM) $MU$ can simulate the execution of any Turing machine $M$ on any input $w$.

**Working of UTM $U(\langle M, w \rangle)$**

- Halt iff $M$ halts on input $w$.
- If $M$ is a deciding/semideciding machine, then
  - If $M$ accepts, accept.
  - If $M$ rejects, reject.
- If $M$ computes a function, then $U(\langle M, w \rangle)$ must equal $M(w)$.

# Universal TM Exists

Consider a TM $M$. Build a Universal TM $U$ that takes input <M>, w where <M> is the encoding of TM $M$ and $w \in \Sigma^*$ is an input to $M$.
Convenient to assume $U$ has $3$ tapes.

> **Program Tape** contains <M>. **M-tape** simulates the tape of $M$. **State Tape** to keep track of $M$'s state.
>
> Initially, $U$ has <M> on Program Tape, $w$ on M-tape and $q_1$ (start state of $M$) on State Tape.
>
> Now, $U$ simulates $M$ on input $w$ step-by-step.

**Conclusion:** This Universal TM is a general purpose computer. All it needs is a **program/algorithm** that tells it what/how to compute.

# Repercussions of Universal TMs

Birth of the **Stored-Program Computer**

**Computability Theory:** Allows us to reason about the limits of what is computable. We can now ask universal questions about all possible computations, not just the computations of a single machine.

Universal TM strengthens **Church-Turing Thesis** since it can compute anything that any TM can compute.

# Halting Problem

Can Universal TM compute everything? **NO!** There are limits to what is computable.
We'll see its impossible to determine if an arbitrary TM $M$ halts on an arbitrary input $w$.
This is called the **Halting Problem**.

Question 1 (Halting Problem)

*Does a TM* Halts *exist, that, on input* $(\langle \mathrm{M} \rangle, w)$ *accepts if* $M$ *halts on input* $w$ *and rejects if* $M$ *loops on input* $w$*?*

Assume Halts exists. We'll show a contradiction by building a paradoxical TM Paradox using Halts as a subroutine.

**Input:** single input: code $\langle \mathrm{M} \rangle$ of a TM $M$.

**Operation:** Run Halts on input $(\langle \mathrm{M} \rangle, \langle \mathrm{M} \rangle)$.

**Result:**

If Halts accepts, then Paradox deliberately runs infinite loop.

If Halts rejects, then Paradox halts and accepts.

# Halting Problem : Continued

What happens when we run Paradox on input $\langle\text{Paradox}\rangle$?

- If Halts accepts input $(\langle\text{Paradox}\rangle, \langle\text{Paradox}\rangle)$, then Paradox loops forever on input $\langle\text{Paradox}\rangle$. **contradiction!**
- If Halts rejects input $(\langle\text{Paradox}\rangle, \langle\text{Paradox}\rangle)$, then Paradox halts and accepts on input $\langle\text{Paradox}\rangle$. **contradiction!**

**Conclusion:** No TM can solve the Halting Problem. It is an **uncomputable problem**.

# Russel Paradox

Let $R = \{S \mid S \notin S\}$ be the set of all sets that do not contain themselves as a member.

**Does $R$ contain itself as a member?**

If $R \in R$, then by definition of $R$, $R \notin R$. **contradiction!**

If $R \notin R$, then by definition of $R$, $R \in R$. **contradiction!**

Helped mathematicians realize that **naive set theory** is inconsistent. Led to development of more robust axiomatic set theories.