Lecture 12
Last time:
• Turing Machines and Variants
Today
• Turing Machine Variants
• Church-Turing Thesis
• Universal Turing Machine
• Decidable languages

Sofya Raskhodnikova
TMs are equivalent to **multitape TMs**

(last time)

TMs are equivalent to **nondeterministic TMs**

(last time)

TMs are equivalent to **doubly unbounded TMs**
A TM with doubly unbounded tape is like an ordinary TM but

- Its tape is infinite on the left and on the right.

Initially, only the input is written on the tape and the head is on the first nonblack symbol.
Exercise

A. TM can simulate a doubly unbounded TM $U$
   A. by marking the leftmost "investigated" square and using a nondeterministic step every time $U$ moves to the left of it.
   
   B. by using 2 tapes: one for input + squares to the right; the other for squares to the left of the input.
   
   C. by using each square of the tape to keep two characters from $U$’s tape alphabet (2 tracks on the tape).
   
   D. None of the above.
   
   E. More than one choice above works.
TMs are equivalent to multitape TMs (last time)

TMs are equivalent to nondeterministic TMs (last time)

TMs are equivalent to doubly unbounded TMs

TMs are equivalent to enumerators
TM variant: enumerator

- Starts with a blank tape
- Prints strings

$L(E) = \text{set of strings that } E \text{ eventually prints.}$

Enumerator $E$ enumerates language $L(E)$.

May never terminate even if the language is finite.
May print the same string many times.
Theorem. A language is Turing-recognizable ⇔ some enumerator enumerates it.

Proof:

⇐ Start with an enumerator E that enumerates A. Give a TM that recognizes A.

⇒ Start with a TM that recognizes A. Give an enumerator E that enumerates A.
TMs are equivalent to multitape TMs 
(last time)

TMs are equivalent to nondeterministic TMs 
(last time)

TMs are equivalent to double unbounded TMs 
(on the board)

TMs are equivalent to enumerators. 
(on the board)

TMs are equivalent to FIFO automata. 
(HW problem)

TMs are equivalent to primitive recursive functions.

TMs are equivalent to cellular automata.
L is recognized by a program for some computer*

\[ \uparrow \]

L is recognized by a TM

**History**

- 23 Hilbert’s problems (1900)
  - stated at International Congress of Mathematicians
  - 10th problem: Give a procedure for determining if a polynomial in \( k \) variables has an integral root.

* The computer must be “reasonable”
The Church-Turing Thesis is consistent with all known “reasonable” computers.

R1: 1101001...
R2: 1011001...
  ...
RAM: #1011#1101101#1011001#...#

Programs for a computer have instructions like:
ADD R1, R2, R3; LOAD R1, R2; STORE R1,R2; MUL R1, R2, R2; BRANCH R1, X;...
Programming languages

- Programming languages like Java, Python, Scheme, C, … are equivalent to TMs
- We call such languages **Turing-complete**

**Corollary.** If two programming languages are Turing-complete, then they can recognize exactly the same set of languages.
Since TMs and programming languages are equivalent, we can think of TMs as programs.

Since programs are strings, we can consider languages whose elements are programs.
Can we encode a Turing Machine as a string of 0s and 1s?

- $\langle O \rangle$ denotes an encoding of object $O$ as a string

states $Q=\{0,1,\ldots,n-1\}$

tape symbols $\Gamma=\{0,1\ldots,m-1\}$
(first $k$ are input symbols)

$\delta : \langle (p,a), (q,b,L) \rangle = 0^p 10^a 10^q 10^b 10$
A universal Turing Machine

- Since TMs and programming languages are equivalent, we can think of TMs as programs.
- Since programs are strings, we can consider languages whose elements are programs.
- \( \langle M \rangle \) denotes an encoding of a TM M as a string

**Theorem.** We can make a **Universal TM**, a TM that takes any TM description \( \langle M \rangle \) and any string \( w \) as input and simulates the computation of \( M \) on \( w \).
Similarly, we can encode DFAs, NFAs, regular expressions, PDAs, CFGs, etc into strings of 0s and 1s.

We can define the following languages:

\[ A_{\text{DFA}} = \{ \langle D, w \rangle \mid D \text{ is a DFA that accepts string } w \} \]

\[ A_{\text{NFA}} = \{ \langle N, w \rangle \mid N \text{ is an NFA that accepts string } w \} \]

\[ A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \]
Theorem. $A_{DFA}$ is decidable.

Proof: The following TM $M$ decides $A_{DFA}$.

$M = \langle D, w \rangle$, where $D$ is a DFA and $w$ is a string:

1. Check if input (to $M$) is legal, reject if not. (This step is assumed to be the first step of every algorithm.)
2. Simulate $D$ on $w$.
3. Accept if $D$ ends in an accept state. O.w. reject.”

Corollary. $A_{NFA}$ is decidable.

(1. Convert input NFA $N$ to an equivalent DFA $D$.)