Intro to Theory of Computation

CS 332

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…

TMs are equivalent to multitape TMs

(last time)

TMs are equivalent to nondeterministic TMs

(last time)

TMs are equivalent to doubly unbounded TMs

(last time)

TMs are equivalent to enumerators

TM variant: enumerator

- Starts with a blank tape
- Prints strings

L(E) = set of strings that E eventually prints.

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

May never terminate even if the language is finite.

May print the same string many times.

TMs vs. enumerators

Theorem. A language is Turing-recognizable $\Leftrightarrow$ some enumerator enumerates it.

Proof:

$\Leftarrow$ Start with an enumerator E that enumerates A.

Give a TM that recognizes A.

TMs vs. enumerators

Theorem. A language is Turing-recognizable $\Leftrightarrow$ some enumerator enumerates it.

Proof:

$\Rightarrow$ Start with a TM M that recognizes A.
Give an enumerator E that enumerates A.
Let $s_1, s_2, \ldots$ be all strings in Σ^* in string order.

TMs are equivalent to…

TMs are equivalent to multitape TMs

(last time)

TMs are equivalent to nondeterministic TMs

(last time)

TMs are equivalent to double unbounded TMs

(last time)

TMs are equivalent to enumerators.

(on the board)

TMs are equivalent to 2-stack PDA.

(HW problem)

TMs are equivalent to primitive recursive functions.

TMs are equivalent to cellular automata.

The Church-Turing Thesis (1936)

L is recognized by a program for some computer*

$\updownarrow$

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.

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.

CS 332

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

start state

reject state

states Q={0,1,…,n-1}

$$0^n10^m10^k10^s10^t10^r10^u1\ldots \langle \delta \rangle$$

tape symbols Γ={0,1…,m-1}
(first k are input symbols)

accept state

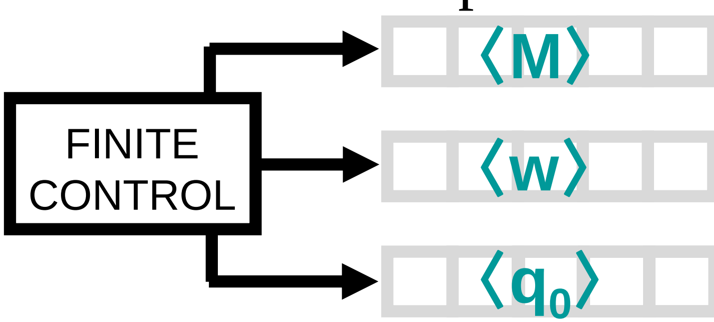
blank symbol

$$\delta : \langle (p,a), (q,b,L) \rangle = 0^p10^a10^q10^b10$$

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.
- **⟨M⟩** denotes an encoding of a TM M as a string

Theorem. We can make a **Universal TM,** a TM that takes any TM description **⟨M⟩** and a description of any string **w** as input and simulates the computation of **M** on **w**.



encoding of M

tape contents

state

$$\langle w \rangle = \langle w_1, \ldots, w_n \rangle = 0^{w_1}10^{w_2}1\ldots0^{w_n}$$

Encodings of DFAs, NFAs, CFGs, *etc*

- Similarly, we can encode DFAs, NFAs, regular expressions, PDAs, CFGs, etc into strings of 0s and 1s.
- We can define the following languages:

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

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

$A_{CFG} = \{ \langle G,w \rangle \mid G \text{ is a CFG that generates string } w \}$

CS 332

Theorem. A_{DFA} is decidable.

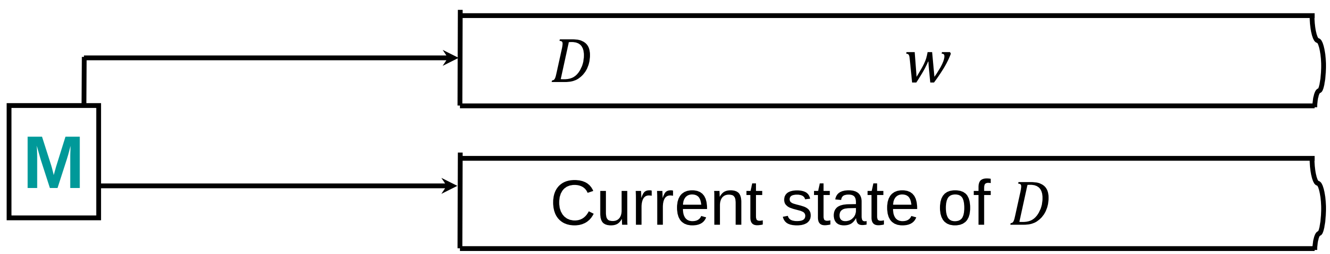
Proof: The following TM M decides A_{DFA}.

M = `` On input $\langle D, w\rangle$, 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.)

CS 332

Theorem. A_{CFG} is decidable.

A_{CFG} = { ⟨G,w⟩ | G is a CFG that generates string w }

Chomsky Normal Form for CFGs

- Can have a rule $S \to \varepsilon$.
- All remaining rules are of the form

$$A \to BC \qquad\qquad A, B, C \in V$$

$$A \to a \qquad\qquad a \in \Sigma$$

- Cannot have S on the RHS of any rule.

Lemma. Any CFG can be converted into an equivalent CFG in Chomsky normal form. (Proof in Sipser.)

Lemma. If G is in Chomsky normal form, any derivation of string w of length n in G has $2n - 1$ steps.

Chomsky Normal Form for CFGs

Lemma. If G is in Chomsky normal form, any derivation of string w of length n in G has $2n - 1$ steps.

Proof idea:

- Only rules of the form $A \rightarrow BC$ increase the number of symbols: need to apply rules of this form $n - 1$ times.
- Only rules of the form $A \rightarrow a$ replace variables with terminals: need to apply rules of this form n times.

CS 332

Theorem. A_{CFG} is decidable.

A_{CFG} = { ⟨G,w⟩ | G is a CFG that generates string w }

Proof: **The following TM M decides A_{CFG}.**

M = `` On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. **Convert G to Chomsky normal form.**
2. **Let $n = |w|$.**
3. **Test all derivations with $2n - 1$ steps.**
4. **Accept if any derived w. O.w. reject."**

Examples of decidable languages

A_{DFA} = { $\langle D,w \rangle$ | D is a DFA that accepts string w }

A_{NFA} = { $\langle N,w \rangle$ | N is an NFA that accepts string w }

A_{CFG} = { $\langle G,w \rangle$ | G is a CFG that generates string w }