Intro to Theory of Computation

CS 464

LECTURE 14

Last time

- Turing Machine Variants
- Church-Turing Thesis

Today

- Universal TM
- Decidable languages
- Designing deciders

Homework 5 due
Homework 6 out

Sofya Raskhodnikova

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"

I-clicker problem (frequency: AC)

The language corresponding to Hilbert's 10th problem is

A. not Turing-recongnizable

B. Turing-recognizable, but not decidable

C. decidable

D. regular

E. More than one choice above works.

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 464

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}

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 \ldots \langle \delta \rangle$$

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

accept state

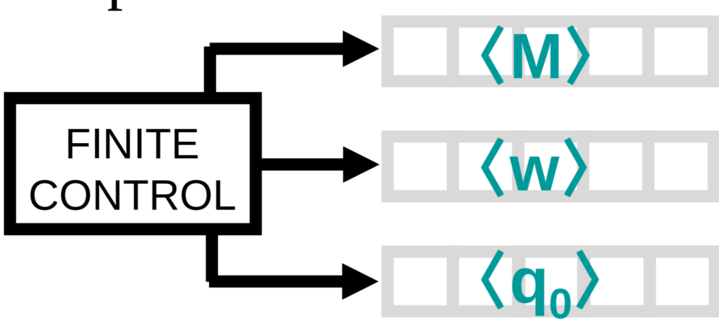
blank symbol

$$\delta : \langle (p,a), (q,b,L) \rangle = 0^p 1 0^a 1 0^q 1 0^b 1 0$$

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 any string **w** as input and simulates the computation of **M** on **w**.



encoding of M

tape contents

state

$$\langle w\rangle = \langle w_1,\ldots,w_n\rangle = 0^{w1}10^{w2}1\ldots0^{wn}$$

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:

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

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

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

Theorem. A_{DFA} is decidable.

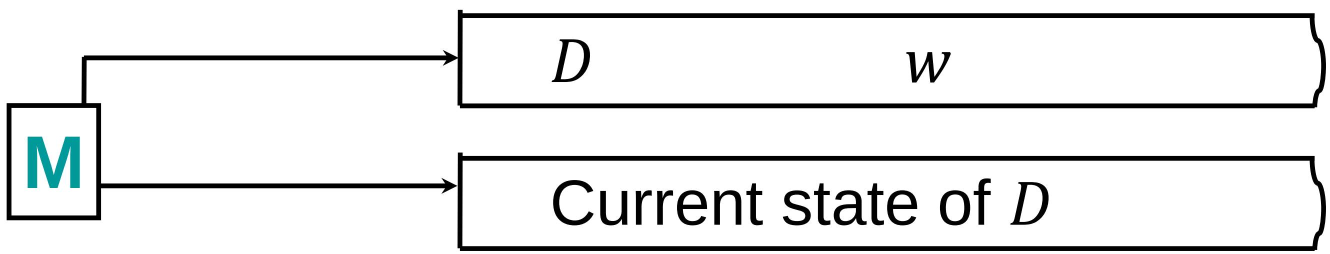
Proof: The following TM M decides A_{DFA}.

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.)

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.

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

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

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

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

More decidable languages

$\mathbf{E_{DFA}}$ = { $\langle D \rangle$ | D is a DFA that recognizes the empty language }

$\mathbf{EQ_{DFA}}$ = { $\langle D_1, D_2 \rangle$ | D_1, D_2 are DFAs and $L(D_1) = L(D_2)$ }

$\mathbf{E_{CFG}}$ = { $\langle G \rangle$ | G is a CFG that generates the empty language}

Theorem. E_{DFA} is decidable.

E_{DFA} = { ⟨*D*⟩ | *D* is a DFA that recognizes ∅.}

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

M = `` On input ⟨*D*⟩, where *D* is a DFA:

1. **Use BFS to determine if an accepting state of D is reachable from from its start state.**
2. **Accept if not. O.w. reject.”**

Theorem. EQ_{DFA} is decidable.

EQ_{DFA} = { $\langle D_1, D_2\rangle$| D_1, D_2 are DFAs & $L(D_1) = L(D_2)$}

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

M = `` On input $\langle D_1, D_2\rangle$, where D_1, D_2 are DFAs:

1. **Construct a DFA D that recognizes the set difference of $L(D_1)$ and $L(D_2)$.** **(on the board)**
2. **Run the decider for E_{DFA} on <D>.**
3. **If it accepts, accept. O.w. reject."**

Theorem. E_{CFG} is decidable.

E_{CFG} **= { ⟨*G*⟩ | *G* is a CFG that recognizes ∅.}**

Proof: The following TM M decides E_{CFG}.

M = `` On input ⟨*G*⟩, where *G* is a CFG:

1. **Mark all terminals in G.**
2. **Repeat until no new variable is marked:**
3. **Mark any variable A where G has a rule A→ ⋯ and each variable/terminal on the RHS is already marked.**
4. **Accept if the start variable is unmarked. O.w. reject.”**

Exercises

- Prove that the following language is decidable:

R_{DFA} = { ⟨D,w⟩ | **D is a DFA that rejects string w** }

- Formulate the following problem as a language and prove that it is decidable:

Given a PDA and a string, determine if the PDA accepts the string.

A_{PDA} = { ⟨P,w⟩ | **P is a PDA that accepts string w** }

Can a TM just simulate P on w, accept if it accepts and reject o.w.?

I-clicker problem (frequency: AC)

A decider for A_{PDA} can, on input <P, w>

A. simulate P on w, accept if it accepts and reject o.w.

B. convert P to an equivalent CFG G and then run a decider for A_{CFG}, accept if it accepts and reject o.w.

C. convert P to an equivalent CFG G and then run a decider for A_{CFG}, accept if it rejects and accept o.w.

D. None of the above.

E. More than one choice above works.