CS5371
Theory of Computation

Lecture 12: Computability III
(Decidable Languages relating to DFA, NFA, and CFG)

Objectives

- Recall that decidable languages are languages that can be decided by TM (that means, the corresponding TM will accept or reject correctly, never loops)
- In this lecture, we investigate some decidable languages that are related to DFA, NFA, and CFG
  - Testing Acceptance, Emptiness, or Equality
- Also, we show how TM can simulate CFG

Acceptance by DFA

Let A_{DFA} be the language

$\{ \langle B, w \rangle \mid B$ is a DFA that accepts $w \}$

where $\langle B, w \rangle$ denotes the encoding of B followed by w

For example, if D is a DFA accepting even length strings, then,

$\langle D, 01 \rangle$, $\langle D, 0000 \rangle$ are strings in A_{DFA} ,

but $\langle D, 1 \rangle$, $\langle D, 000 \rangle$ are not

Acceptance by DFA (2)

Theorem 1: A_{DFA} is a decidable language

Proof: We construct a TM $D_{A_{DFA}}$ that decides A_{DFA} as follows:

$D_{A_{DFA}}$ = "On input $\langle B, w \rangle$

1. Simulate B on input w
2. If the simulation ends in an accept state, accept. Else, reject "

Acceptance by DFA (3)

Q1: How can $D_{A_{DFA}}$ perform the above steps??

- It uses 3 tapes; initially, Tape 1 stores the input ⟨B, w⟩, the other two all blanks
- Then, $D_{A_{DFA}}$ copies w into Tape 2, and write the start state of B in Tape 3
- Usage: Tape head of Tape 2 points to next char in w for B to read, Tape 3 stores current state
- Based on Tapes 2 and 3, $D_{A_{DFA}}$ moves back and forth Tape 1 to know how B performs each transition, and update the 3 tapes accordingly

Acceptance by DFA (4)

Q2: Why is $D_{A_{DFA}}$ a decider for A_{DFA}?

- For any input ⟨B, w⟩, it can simulate B so that each transition in B takes finite number of steps
- To know which state B is at after reading w, there are only |w| transitions in B
- Thus, it takes finite number of steps to know if B accepts w or not. So, $D_{A_{DFA}}$ can decide (no infinite loop) whether to accept or reject ⟨B, w⟩

Acceptance by NFA

Let A_{NFA} be the language

$\{ \langle B, w \rangle \mid B$ is an NFA that accepts $w \}$

Theorem 2: A_{NFA} is a decidable language

Acceptance by NFA (2)

Proof:

[Solution 1] We can use the same idea when we simulate NTM by TM, so that we give a TM that decides A_{NFA}. Precisely, we need to try every possible branch of computation, but only of length up to $|Q||w| + |Q|$ (why??)

Acceptance by NFA (3)

[Solution 2 (easier)] We re-use $D_{A_{DFA}}$ to give a TM $D_{A_{NFA}}$ that decides A_{NFA}:

$D_{A_{NFA}}$ = "On input $\langle B, w \rangle$

1. Convert B to an equivalent DFA C
2. Run $D_{A_{DFA}}$ on $\langle C, w \rangle$
3. If $D_{A_{DFA}}$ accepts, accept. Else, reject"

Acceptance by NFA (4)

Q1: How can D_{ANFA} perform the above steps??

- It uses 5 tapes; initially, Tape 1 stores the input ⟨B, w⟩, Tape 2 stores the encoding of D_{ADFA}, the other three all blanks
- Then, D_{ANFA} converts B to C and store C in Tape 3
- It then consults D_{ADFA} in Tape 2, to know how D_{ADFA} simulates C running on w
- Tapes 4 and 5 can be used to store the current state of C, and the next char for C to read, so that D_{ANFA} can simulate D_{ADFA} to simulate C

Acceptance by NFA (5)

Q2: Why is $D_{A_{NFA}}$ a decider for A_{NFA}?

- For any input ⟨B, w⟩, it convert B into the equivalent DFA C in finite number of steps
- Then, it consults $D_{A_{DFA}}$ which takes finite number of steps to know if C accepts w or not. Thus, $D_{A_{NFA}}$ can decide (no infinite loop) whether to accept or reject ⟨B, w⟩

Acceptance by Regular Expression (RE)

Let A_{RE} be the language

$\{ \langle R, w \rangle \mid R$ is an RE that generates $w \}$

Theorem 3: A_{RE} is a decidable language

Acceptance by RE (2)

Proof: W give a TM $D_{A_{RE}}$ that decides A_{RE}:

$D_{A_{RE}}$ = "On input $\langle R, w \rangle$

1. Convert R to an equivalent NFA A
2. Run $D_{A_{NFA}}$ on $\langle A, w \rangle$
3. If $D_{A_{NFA}}$ accepts, accept. Else, reject"

Emptiness Test for DFA

Let E_{DFA} be the language

$\{ \langle B \rangle \mid B$ is a DFA and $L(B) = \{ \} \}$

Theorem 4: E_{DFA} is a decidable language

Observation: DFA accepts no string if and only if we cannot reach any accept state from the start state by following transition arrows

Emptiness Test for DFA (2)

Proof: We use similar idea as we test if a graph G is connected. Precisely, we give a TM $D_{E_{DFA}}$ that decides E_{DFA} as follows:

$D_{E_{DFA}}$ = "On input $\langle B \rangle$

1. Mark the start state of B
2. Repeat until no new states are marked
   - Mark any state that has a transition coming into it from a marked state
3. If no accept state of B is marked, accept. Else, reject"

Equality Test for DFA

Let EQ_{DFA} be the language
{ ⟨A,B⟩ | A and B are DFAs and L(A) = L(B) }

Theorem 5: EQ_{DFA} is a decidable language

Hint: Let C be a DFA that accepts strings that is in L(A) but not in L(B), and also strings that is in L(B) but not in L(A). Then, L(C) ={ } if and only if L(A) = L(B)

Equality Test for DFA (2)

Proof: Based on the hint, we give a TM $D_{EQ_{DFA}}$ that decides EQ_{DFA} as follows:

$D_{EQ_{DFA}}$ = "On input $\langle A,B \rangle$

1. Construct C (how?)
2. Run $D_{E_{DFA}}$ (Emptiness-Tester for DFA) on $\langle C \rangle$
3. If $D_{E_{DFA}}$ accepts, accept. Else, reject"

Acceptance by CFG

Let A_{CFG} be the language

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

Theorem 6: A_{CFG} is a decidable language

Hint: We need to avoid testing infinite derivations... If G is in Chomsky normal form, any derivation of w takes exactly $2|w| - 1$ derivation steps

Acceptance by CFG (2)

Proof: Based on the hint, we give a TM $D_{A_{CFG}}$ that decides A_{CFG} as follows:

$D_{A_{CFG}}$ = "On input ⟨G, w⟩

1. Convert G into G' = (V,T,R,S) in CNF
2. Generate all derivations of G' with $2|w|-1$ derivation steps (Note: # of such derivations is less than $(4|V||T|)^{2|w|-1}$. That is, a finite number)
3. If any derivation generates w, accept. Else, reject"

Emptiness Test for CFG

Let E_{CFG} be the language

$\{ \langle G \rangle \mid G$ is a CFG and $L(G) = \{ \} \}$

Theorem 7: E_{CFG} is a decidable language

Observation: Suppose that we can mark all the variables in G that can generate a string of terminals. Then, $L(G) = \{ \}$ if the start variable is not marked

Emptiness Test for CFG (2)

Proof: We use similar idea as we test if a graph G is connected. Precisely, we give a TM $D_{E_{CFG}}$ that decides E_{CFG} as follows:

$D_{E_{CFG}}$ = "On input $\langle G \rangle$

1. Mark all terminals of G
2. Repeat until no new variable is marked
   - Mark variable A if G has a rule $A \rightarrow U_1U_2...U_k$ and all U_i's are marked
3. If the start variable is not marked, accept. Else, reject"

Equality Test for CFG?

Let EQ_{CFG} be the language

$\{ \langle A,B \rangle \mid A \text{ and } B \text{ are CFGs and } L(A) = L(B) \}$

Is EQ_{CFG} is a decidable language?

Unfortunately, no...

Note that we cannot apply similar trick as we prove EQ_{DFA} is decidable

We shall show EQ_{CFG} is undecidable later...

TM can simulate CFG

- Previously (a long time ago), we have shown that given a DFA, we can always find a CFG that decides the same language

- How about, if we are given a CFG, can we find a TM that decides the same language?
  - The answer is YES!

TM can simulate CFG (2)

Theorem 8: Given a CFG G, we can construct a TM that decides the same language. In other words, every CFL is a decidable language

TM can simulate CFG (3)

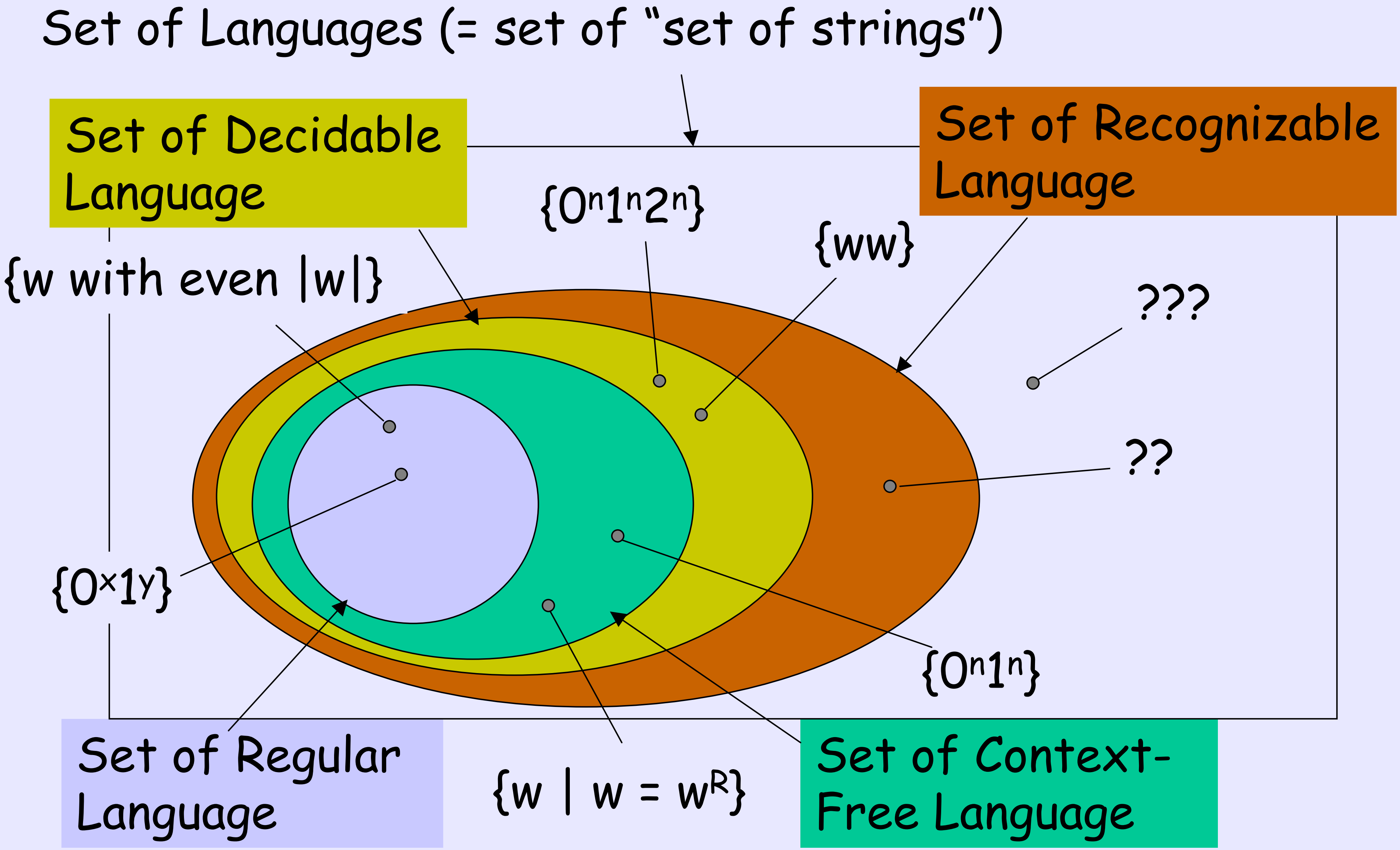
Proof: We find a TM M_G with ⟨G⟩ stored in it initially; M_G then performs as follows:

M_G = "On input ⟨w⟩

1. Run $D_{A_{CFG}}$ (Accept-by-CFG checker) on ⟨G, w⟩
2. If $D_{A_{CFG}}$ accepts ⟨G, w⟩, accept. Else, reject "

We can see that M_G decides the same language as G. This completes the proof

Language Hierarchy (revisited)



Next Time

- Undecidable Languages
  - Languages that CANNOT be decided by ANY Turing Machine
  - Example 1: Turing-recognizable, but not Turing-decidable
  - Example 2: Non-Turing recognizable (that is, even more difficult!!)