Zoom University

~~BU~~ CS 332 – Theory of Computation

Lecture 13:

- Mid-Semester Feedback
- Enumerators
- Decidable Languages

Reading:

Sipser Ch 4.1

HW 4 due Wednesday (2 PM)

HW 5 out due Monday 3/23

Mark Bun

March 16, 2020

What aspects of the course help you learn best?

- Examples in class
- Reviewing past homeworks/exams in class
- Textbook
- Posting materials online
- Lecture, generally
- Office hours
- In-depth problem-solving in discussion section
- Top Hat questions
- Piazza discussions / instructor response

Most frequent

less frequent

What in the class so far has hindered your learning?

- Pace of information transmission / workload
- Criteria for formality of proofs on homework and exams
- Poor handwriting Is this acceptable?
- Questions in class not fully answered
- Lack of organization in discussion
- Broad concepts

- "Bureaucratic descriptions"
- "All materials concluded"

What specific changes can we make to improve your learning?

- More examples
- Post solutions / other materials online — Thanks COVID!
- Discussion solutions
- More Top Hat questions — Trying!
- Go slower
- More guidelines for how to solve each type of problem
- Looser grading — Discussion. Giving as much formative feedback as possible.
- Midterm too long
- More detailed slides

Do you understand what is expected from you in this class?

Starting w/ HW 5: specific exercises we think are important

- Reading the book before vs. after class
- Need to do every problem in the book to succeed?
- Lack of coordination between readings and lectures
- “I have to attend lectures, read the material in the book, do some practice problems and then attempt the homework”

  Most important part of class!
- Exam grading critical over formatting vs. looser standards on homework
  - Give a lot of constructive comments, including about formatting arguments
  - Generally, we don't take off points for formatting

How can you improve your own learning?

- Read the book Yes!
- Solve more practice problems
- Review HW solutions
- Come to office hours
- Time management
- Open mind to more abstract ways of thinking

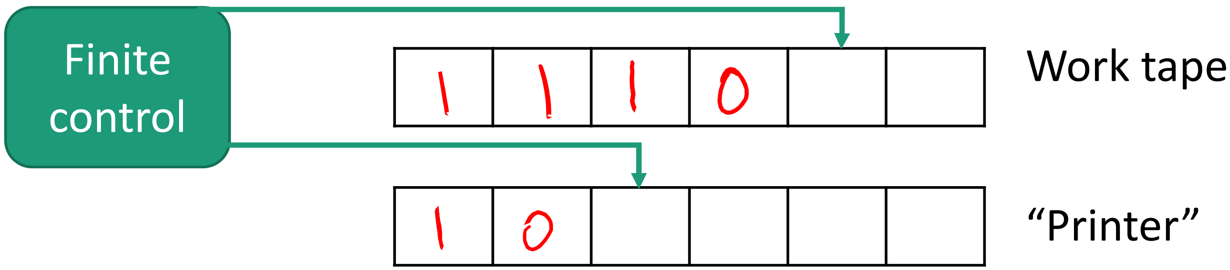
Enumerators

TMs are equivalent to…

- TMs with "stay put"
- TMs with 2-way infinite tapes
- Multi-tape TMs
- Nondeterministic TMs
- Random access TMs
- Enumerators
- Finite automata with access to an unbounded queue = 2-stack PDAs
- Primitive recursive functions
- Cellular automata
- "Turing-complete" programming languages (C, Python, Java…)

…

Enumerators



- Starts with two blank tapes
- Prints strings to printer

$L(E) = \{\text{strings eventually printed by } E\}$

- May never terminate (even if language is finite)
- May print the same string many times

Enumerator Example

Enumerator

1
4
9
16
25
⋮

TM
(on input 16)

1. Initialize $c = 1$
2. Repeat forever:
   - Calculate $s = c^2$ (in binary)
   - Send s to printer
   - Increment c

What language does this enumerator enumerate?

$\{x \mid x$ is a binary number representing a perfect square$\}$

Enumerable = Turing-Recognizable

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

$\Leftarrow$ Start with an enumerator E for A and give a TM

On input w:

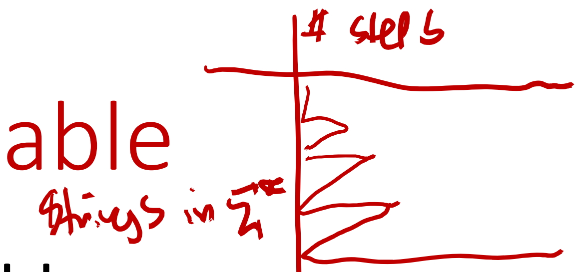
1. Run E, producing $s_1, s_2, s_3, \ldots$
2. If w appears in the sequence of enumerated strings, accept

why does this work?

$w \in A$: E an enumerator $\Rightarrow$ $w = s_i$ for some i $\Rightarrow$ accept

$w \notin A$: w never appears in 1st $\Rightarrow$ TM never halts

Enumerable = Turing-Recognizable



Theorem: A language is Turing-recognizable ⇔ some enumerator enumerates it

⇒ Start with a TM M for A and give an enumerator

Idea: If $w \in A$, then $\exists\, i \in \mathbb{N}$ s.t. $M(w)$ accepts after running for i steps

Enumerator: Fix $s_1, s_2, s_3, \ldots$ of all strings over Σ^*

For $i = 1, 2, 3, \ldots$:

- Run M on s_1 for i steps, run M on s_2 for i steps, . . . run M on s_i for i steps
- Print every $s \in \{s_1, \ldots, s_i\}$ on which M accepted

Decidable Languages

1928 – The *Entscheidungsproblem*

The "Decision Problem"

Is there an algorithm which takes as input a formula (in first-order logic) and decides whether it's logically valid?

"mathematical statement"   "true mathematical statement"

Meta-computational problem: Is ∃ possible to automate mathematicians?

For us: problems about DFAs, CFGs, TMs

Questions about regular languages

Design a TM which takes as input a DFA D and a string w, and determines whether D accepts w

How should the input to this TM be represented?

Let $D = (Q, \Sigma, \delta, q_0, F)$. List each component of the tuple separated by ;

- Represent Q by ,-separated binary strings 0, 1, 10, 11
- Represent Σ by ,-separated binary strings 0, 1
- Represent $\delta : Q \times \Sigma \rightarrow Q$ by a ,-separated list of triples (p, a, q), … $0 \xrightarrow{0} 1 \rightsquigarrow (0, 0, 1)$

Denote the encoding of D, w by $\langle D, w \rangle$

Representation independence

Computability (i.e., decidability and recognizability) is not affected by the choice of encoding

Why? A TM can always convert between different encodings

For now, we can take $\langle \quad \rangle$ to mean "any reasonable encoding"

A "universal" algorithm for recognizing regular languages

$A_{\text{DFA}} = \{\langle D, w\rangle \mid \text{DFA } D \text{ accepts } w\}$

Theorem: A_{DFA} is decidable

"High-level description"

Proof: Define a 3-tape TM M on input $\langle D, w\rangle$:

1. Check if $\langle D, w\rangle$ is a valid encoding (reject if not)
   (implicit at the beginning of all deciders; feel free to omit)
2. Simulate D on w, i.e.,
   - Tape 2: Maintain w and head location of D
   - Tape 3: Maintain state of D, update according to δ
3. Accept iff D ends in an accept state

Other decidable languages

$A_{\text{DFA}} = \{\langle D, w\rangle \mid \text{DFA } D \text{ accepts } w\}$

$A_{\text{NFA}} = \{\langle N, w\rangle \mid \text{NFA } N \text{ accepts } w\}$

$A_{\text{REX}} = \{\langle R, w\rangle \mid \text{regular expression } R \text{ generates } w\}$

$A_{\text{CFG}} = \{\langle G, w\rangle \mid \text{CFG } G \text{ generates } w\}$

CFG Generation

Theorem: $A_{\text{CFG}} = \{\langle G, w\rangle \mid \text{CFG } G \text{ generates } w\}$ is Turing-recognizable

Proof idea: Define a TM M recognizing A_{CFG}

On input $\langle G, w\rangle$:

1. Enumerate all strings that can be generated from G

   (i.e., all length-1 derivations, all length-2 derivations, …)

2. If any of these strings equal w, accept

CFG Generation

Theorem: $A_{\text{CFG}} = \{\langle G, w\rangle \mid \text{CFG } G \text{ generates } w\}$ is decidable

Chomsky Normal Form for CFGs:

- Can have a rule $S \to \varepsilon$
- All remaining rules of the form $A \to BC$ or $A \to a$
- Cannot have S on the RHS of any rule

Lemma: Any CFG can be converted into an equivalent CFG in Chomsky Normal Form

Lemma: If G is in Chomsky Normal Form, any nonempty string w that can be derived from G has a derivation with at most $2|w| - 1$ steps

CFG Generation

Theorem: $A_{\mathrm{CFG}} = \{\langle G, w\rangle \mid \mathrm{CFG}\ G \text{ generates } w\}$ is decidable

Proof idea: Define a TM M recognizing A_{CFG}

On input $\langle G, w\rangle$:

1. Convert G into Chomsky Normal Form
2. Enumerate all strings derivable in $\leq 2|w| - 1$ steps
3. If any of these strings equal w, accept

Context Free Languages are Decidable

Theorem: Every CFL L is decidable

Proof: Let G be a CFG generating L. The following TM decides L.

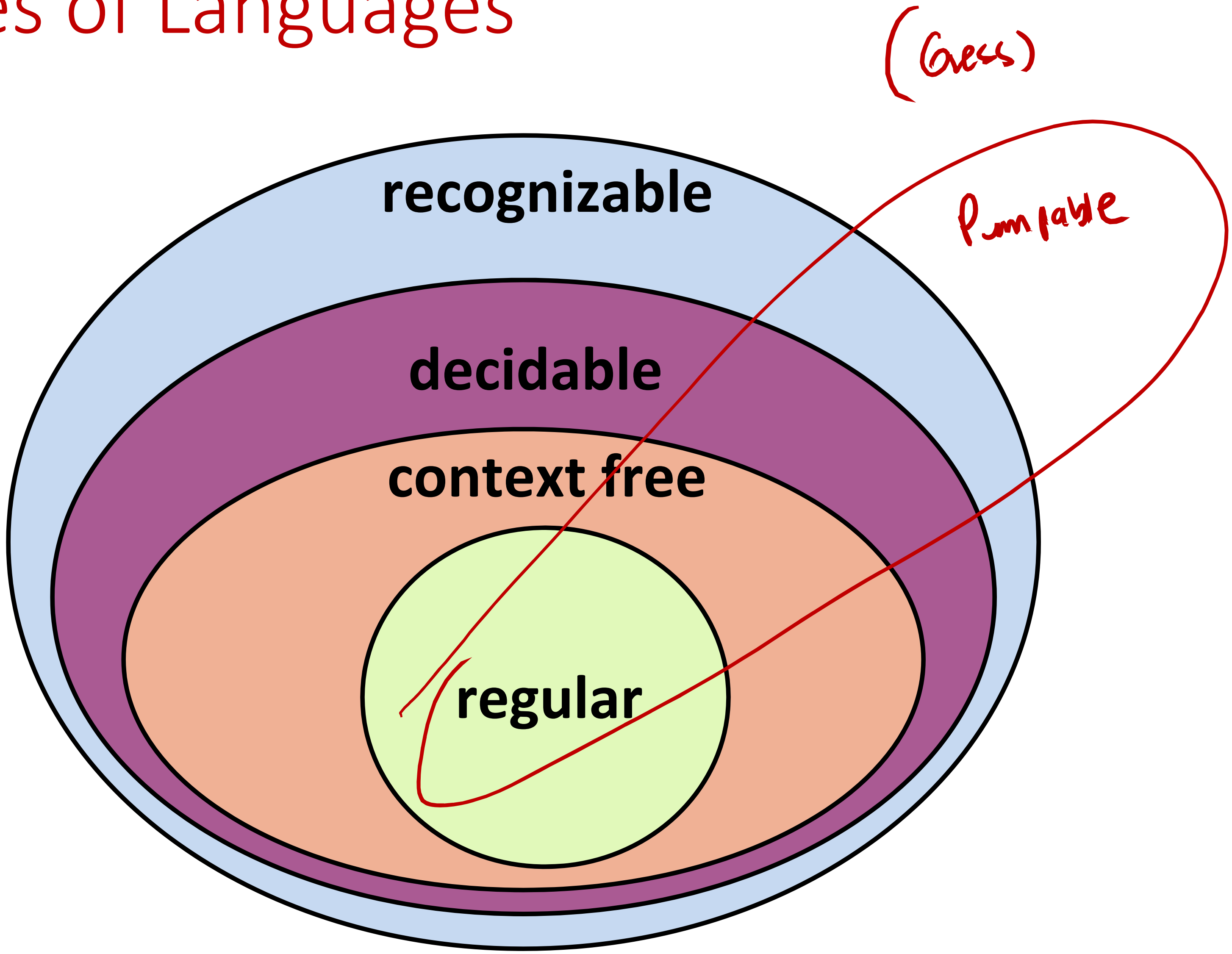
On input w:

1. Run the decider for A_{CFG} on input $\langle G, w \rangle$
2. Accept if the decider accepts; reject otherwise

Decider accepts $\iff$ $\langle G, w\rangle \in A_{CFG}$

$\iff$ w generated by G

Classes of Languages



More Examples

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

$D =$ 

$D \in E_{DFA}$

$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG that recognizes the empty language}\}$

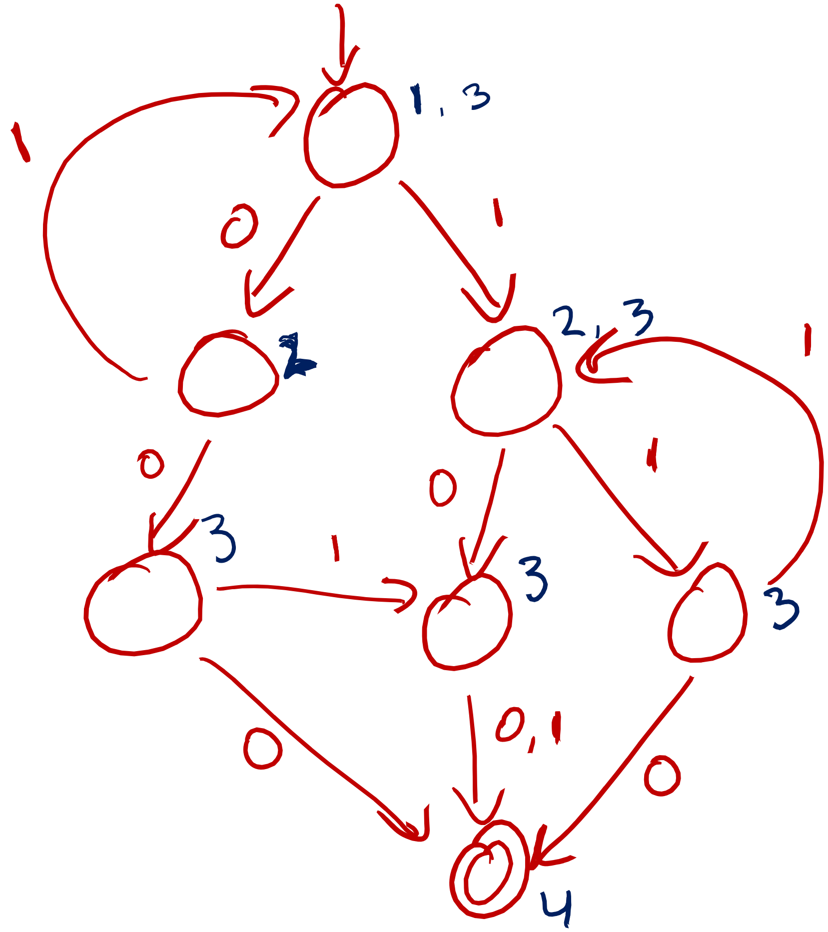
Decidability of E_{DFA}

Theorem: $E_{\text{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA that recognizes } \emptyset\}$ is decidable

Proof: The following TM decides E_{DFA}

On input $\langle D \rangle$, where D is a DFA with n states:

1. Perform n steps of breadth-first search on state diagram of D to determine if an accept state is reachable from the start state
2. Accept if an accept state not reachable; reject otherwise


1, 3
0
1
2, 3
0
1
3
3
3
0, 1
0
4

Decidability of E_{CFG}

Theorem: $E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG that recognizes } \emptyset\}$ is decidable

Proof: The following TM decides E_{CFG}

On input $\langle G \rangle$, where G is a CFG with n states:

1. Mark all terminal symbols in G
2. Repeat until no new variable is marked:
   Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \dots U_k$ and every symbol $U_1, \dots, U_k$ is marked
3. Accept if the start variable is unmarked; else reject

New Deciders from Old

$EQ_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle \mid D_1, D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

Theorem: EQ_{DFA} is decidable

Proof: The following TM decides EQ_{DFA}

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

1. Construct a DFA D that recognizes the **symmetric difference** $L(D_1) \triangle L(D_2)$

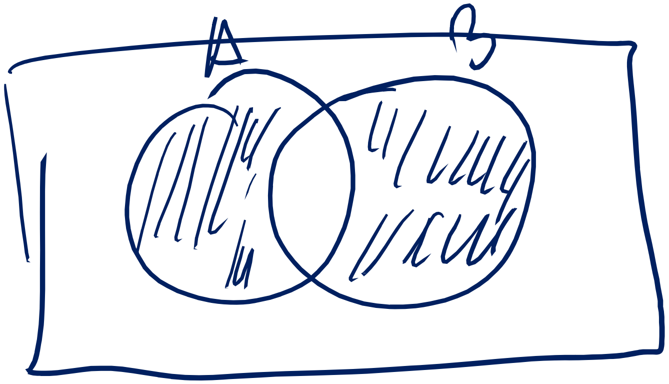
   $= \{ w \mid w \text{ is in exactly one of } L(D_1) \text{ or } L(D_2)\}$

   $L(D_1) = L(D_2) \iff L(D_1) \triangle L(D_2) = \emptyset$

2. Run the decider for E_{DFA} on $\langle D\rangle$ and return its output

Symmetric Difference

$$A \triangle B = \{w \mid w \in A \text{ or } w \in B \text{ but not both}\}$$



$$A \triangle B = (A \setminus B) \cup (B \setminus A)$$
$$= (A \cap \overline{B}) \cup (B \cap \overline{A})$$

Using closure constructions can construct a DFA recognizing $L(D_1) \triangle L(D_2)$