

## Week 10 at a glance

For Monday, Definition 7.1 (page 276).

For Wednesday, Definition 7.7 (page 279).

For Friday: skim through examples in Chapter 7.

### We will be learning and practicing to:

- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
  - Use mapping reduction to deduce the complexity of a language by comparing to the complexity of another.
    - \* Use appropriate reduction (e.g. mapping, Turing, polynomial-time) to deduce the complexity of a language by comparing to the complexity of another.
    - \* Use polynomial-time reduction to prove NP-completeness
  - Classify the computational complexity of a set of strings by determining whether it is decidable or undecidable and recognizable or unrecognizable.
    - \* Distinguish between computability and complexity
    - \* Articulate motivating questions of complexity
    - \* Define NP-completeness
    - \* Give examples of PTIME-decidable, NPTIME-decidable, and NP-complete problems
  - Describe several variants of Turing machines and informally explain why they are equally expressive.
    - \* Define nondeterministic Turing machines
    - \* Use high-level descriptions to define and trace machines (Turing machines and enumerators)

### TODO:

Student Evaluations of Teaching forms: Evaluations are open for completion anytime BEFORE 8AM on Saturday, December 7. Access your SETs from the Evaluations site

<https://academicaffairs.ucsd.edu/Modules/Evals>

You will separately evaluate each of your listed instructors for each enrolled course.

Homework 6 submitted via Gradescope (<https://www.gradescope.com/>), due Tuesday 12/3/2024

## Summary from Week 9

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

To prove the existence of a Turing machine that decides / recognizes some language, it's enough to construct an example using any of the equally expressive models.

But: some of the **performance** properties of these models are not equivalent.

## Monday: Church-Turing Thesis and Complexity

In practice, computers (and Turing machines) don't have infinite tape, and we can't afford to wait unboundedly long for an answer. "Decidable" isn't good enough - we want "Efficiently decidable".

For a given algorithm working on a given input, how long do we need to wait for an answer? How does the running time depend on the input in the worst-case? average-case? We expect to have to spend more time on computations with larger inputs.

A language is **recognizable** if \_\_\_\_\_

A language is **decidable** if \_\_\_\_\_

A language is **efficiently decidable** if \_\_\_\_\_

A function is **computable** if \_\_\_\_\_

A function is **efficiently computable** if \_\_\_\_\_

Definition (Sipser 7.1): For  $M$  a deterministic decider, its **running time** is the function  $f : \mathbb{N} \rightarrow \mathbb{N}$  given by

$$f(n) = \max \text{ number of steps } M \text{ takes before halting, over all inputs of length } n$$

Definition (Sipser 7.7): For each function  $t(n)$ , the **time complexity class**  $TIME(t(n))$ , is defined by

$$TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$$

An example of an element of  $TIME(1)$  is

An example of an element of  $TIME(n)$  is

Note:  $TIME(1) \subseteq TIME(n) \subseteq TIME(n^2)$

Definition (Sipser 7.12) :  $P$  is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_k TIME(n^k)$$

Theorem (Sipser 7.8): Let  $t(n)$  be a function with  $t(n) \geq n$ . Then every  $t(n)$  time deterministic multitape Turing machine has an equivalent  $O(t^2(n))$  time deterministic 1-tape Turing machine.

Definitions (Sipser 7.1, 7.7, 7.12): For  $M$  a deterministic decider, its **running time** is the function  $f : \mathbb{N} \rightarrow \mathbb{N}$  given by

$$f(n) = \max \text{ number of steps } M \text{ takes before halting, over all inputs of length } n$$

For each function  $t(n)$ , the **time complexity class**  $TIME(t(n))$ , is defined by

$$TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$$

$P$  is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_k TIME(n^k)$$

Definition (Sipser 7.9): For  $N$  a nondeterministic decider. The **running time** of  $N$  is the function  $f : \mathbb{N} \rightarrow \mathbb{N}$  given by

$$f(n) = \max \text{ number of steps } N \text{ takes on any branch before halting, over all inputs of length } n$$

Definition (Sipser 7.21): For each function  $t(n)$ , the **nondeterministic time complexity class**  $NTIME(t(n))$ , is defined by

$$NTIME(t(n)) = \{L \mid L \text{ is decidable by a nondeterministic Turing machine with running time in } O(t(n))\}$$

$$NP = \bigcup_k NTIME(n^k)$$

**True or False:**  $TIME(n^2) \subseteq NTIME(n^2)$

**True or False:**  $NTIME(n^2) \subseteq TIME(n^2)$

**Every problem in NP is decidable with an exponential-time algorithm**

Nondeterministic approach: guess a possible solution, verify that it works.

Brute-force (worst-case exponential time) approach: iterate over all possible solutions, for each one, check if it works.

## Wednesday: P and NP

### Examples in $P$

*Can't use nondeterminism; Can use multiple tapes; Often need to be "more clever" than naïve / brute force approach*

$$PATH = \{\langle G, s, t \rangle \mid G \text{ is digraph with } n \text{ nodes there is path from } s \text{ to } t\}$$

Use breadth first search to show in  $P$

$$RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime integers}\}$$

Use Euclidean Algorithm to show in  $P$

$$L(G) = \{w \mid w \text{ is generated by } G\}$$

(where  $G$  is a context-free grammar). Use dynamic programming to show in  $P$ .

### Examples in $NP$

*"Verifiable" i.e. NP, Can be decided by a nondeterministic TM in polynomial time, best known deterministic solution may be brute-force, solution can be verified by a deterministic TM in polynomial time.*

$$HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is digraph with } n \text{ nodes, there is path from } s \text{ to } t \text{ that goes through every node exactly once}\}$$

$$VERTEX - COVER = \{\langle G, k \rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-node vertex cover}\}$$

$$CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-clique}\}$$

$$SAT = \{\langle X \rangle \mid X \text{ is a satisfiable Boolean formula with } n \text{ variables}\}$$

| Problems in $P$                           | Problems in $NP$   |
|---|--------------------|
| (Membership in any) regular language      | Any problem in $P$ |
| (Membership in any) context-free language |                    |
| $A_{DFA}$                                 | $SAT$              |
| $E_{DFA}$                                 | $CLIQUE$           |
| $EQ_{DFA}$                                | $VERTEX - COVER$   |
| $PATH$                                    | $HAMPATH$          |
| $RELPRIME$                                | $\dots$            |
| $\dots$                                   |                    |

Notice:  $NP \subseteq \{L \mid L \text{ is decidable}\}$  so  $A_{TM} \notin NP$

Million-dollar question: Is  $P = NP$ ?

One approach to trying to answer it is to look for *hardest* problems in  $NP$  and then (1) if we can show that there are efficient algorithms for them, then we can get efficient algorithms for all problems in  $NP$  so  $P = NP$ , or (2) these problems might be good candidates for showing that there are problems in  $NP$  for which there are no efficient algorithms.

Definition (Sipser 7.29) Language  $A$  is **polynomial-time mapping reducible** to language  $B$ , written  $A \leq_P B$ , means there is a polynomial-time computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that for every  $x \in \Sigma^*$

$$x \in A \quad \text{iff} \quad f(x) \in B.$$

The function  $f$  is called the polynomial time reduction of  $A$  to  $B$ .

**Theorem** (Sipser 7.31): If  $A \leq_P B$  and  $B \in P$  then  $A \in P$ .

Proof:

Definition (Sipser 7.34; based in Stephen Cook and Leonid Levin's work in the 1970s): A language  $B$  is **NP-complete** means (1)  $B$  is in NP **and** (2) every language  $A$  in NP is polynomial time reducible to  $B$ .

**Theorem** (Sipser 7.35): If  $B$  is NP-complete and  $B \in P$  then  $P = NP$ .

Proof:

## Friday: NP-Completeness

### NP-Complete Problems

**3SAT:** A literal is a Boolean variable (e.g.  $x$ ) or a negated Boolean variable (e.g.  $\bar{x}$ ). A Boolean formula is a **3cnf-formula** if it is a Boolean formula in conjunctive normal form (a conjunction of disjunctive clauses of literals) and each clause has three literals.

$$3SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \}$$

Example string in  $3SAT$

$$\langle (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z) \rangle$$

Example string not in  $3SAT$

$$\langle (x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \rangle$$

**Cook-Levin Theorem:**  $3SAT$  is  $NP$ -complete.

*Are there other NP-complete problems?* To prove that  $X$  is  $NP$ -complete

- *From scratch:* prove  $X$  is in  $NP$  and that all  $NP$  problems are polynomial-time reducible to  $X$ .
- *Using reduction:* prove  $X$  is in  $NP$  and that a known-to-be  $NP$ -complete problem is polynomial-time reducible to  $X$ .

**CLIQUE:** A  $k$ -**clique** in an undirected graph is a maximally connected subgraph with  $k$  nodes.

$$CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Example string in *CLIQUE*

Example string not in *CLIQUE*

Theorem (Sipser 7.32):

$$3SAT \leq_P CLIQUE$$

Given a Boolean formula in conjunctive normal form with  $k$  clauses and three literals per clause, we will map it to a graph so that the graph has a clique if the original formula is satisfiable and the graph does not have a clique if the original formula is not satisfiable.

The graph has  $3k$  vertices (one for each literal in each clause) and an edge between all vertices except

- vertices for two literals in the same clause
- vertices for literals that are negations of one another

Example:  $(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee z)$



| Model of Computation   | Class of Languages  |
|--|---|
| <p><b>Deterministic finite automata:</b> formal definition, how to design for a given language, how to describe language of a machine? <b>Nondeterministic finite automata:</b> formal definition, how to design for a given language, how to describe language of a machine? <b>Regular expressions:</b> formal definition, how to design for a given language, how to describe language of expression? <i>Also:</i> converting between different models.</p> | <p><b>Class of regular languages:</b> what are the closure properties of this class? which languages are not in the class? using <b>pumping lemma</b> to prove nonregularity.</p>                   |
| <p><b>Push-down automata:</b> formal definition, how to design for a given language, how to describe language of a machine? <b>Context-free grammars:</b> formal definition, how to design for a given language, how to describe language of a grammar?</p>  | <p><b>Class of context-free languages:</b> what are the closure properties of this class? which languages are not in the class?</p>   |
| <p>Turing machines that always halt in polynomial time</p> <p>Nondeterministic Turing machines that always halt in polynomial time</p>   | <p><math>P</math></p> <p><math>NP</math></p>  |
| <p><b>Deciders</b> (Turing machines that always halt): formal definition, how to design for a given language, how to describe language of a machine?</p>   | <p><b>Class of decidable languages:</b> what are the closure properties of this class? which languages are not in the class? using diagonalization and mapping reduction to show undecidability</p> |
| <p><b>Turing machines</b> formal definition, how to design for a given language, how to describe language of a machine?</p>  | <p><b>Class of recognizable languages:</b> what are the closure properties of this class? which languages are not in the class? using closure and mapping reduction to show unrecognizability</p>   |

**Given a language, prove it is regular**

*Strategy 1:* construct DFA recognizing the language and prove it works.

*Strategy 2:* construct NFA recognizing the language and prove it works.

*Strategy 3:* construct regular expression recognizing the language and prove it works.

*“Prove it works” means ...*

**Example:**  $L = \{w \in \{0,1\}^* \mid w \text{ has odd number of 1s or starts with } 0\}$

Using NFA

Using regular expressions

**Example:** Select all and only the options that result in a true statement: “To show a language  $A$  is not regular, we can...”

- a. Show  $A$  is finite
- b. Show there is a CFG generating  $A$
- c. Show  $A$  has no pumping length
- d. Show  $A$  is undecidable

**Example:** What is the language generated by the CFG with rules

$$S \rightarrow aSb \mid bY \mid Ya$$

$$Y \rightarrow bY \mid Ya \mid \varepsilon$$

**Example:** Prove that the language  $T = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) \text{ is infinite}\}$  is undecidable.

**Example:** Prove that the class of decidable languages is closed under concatenation.

