

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. 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.

Review Quiz 9 on PrairieLearn (<http://us.prairielearn.com>), due 3/12/2025

Homework 6 submitted via Gradescope (<https://www.gradescope.com/>), due 3/13/2025

Project submitted via Gradescope (<https://www.gradescope.com/>), due 3/19/2025

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 there is a TM that accepts all and only strings in this language.

A language is **decidable** if there is a TM that accepts all and only strings in the language and rejects all and only strings not in language.

A language is **efficiently decidable** if there is a TM that is a decider and whose running time is polynomial and recognizes the language

* A function is **computable** if there is a TM that, for each input, halts within finitely many steps, with output of function of type.

A function is **efficiently computable** if there is TM that halts for each input with polynomial running time and for each input, output gives the value of the function.

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

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

languages aka computational problems solvable in no worse than $t(n)$ time.

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

$$\Sigma^*, \emptyset, \{1w \mid w \in \Sigma^*\}$$

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

$$\{w1 \mid w \in \Sigma^*\}, \text{ any set in } O(1)$$

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

Time hierarchy

TM recognizing Σ^*



TM recognizing \emptyset



TM recognizing $\{w \in \Sigma^* \mid |w| = 3\}$

TM recognizing $\{1w \mid w \in \Sigma^*\}$

TM recognizing $\{w1 \mid w \in \Sigma^*\}$

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

deterministic alg gives
us a "nondeterministic" alg

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

???

nondeterministic alg
↓
not too big a cost
deterministic alg

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

nondeterministic

Examples in P

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

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

Recall:
E_{DFA}

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

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

Recall:
A_{CFG}

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

Examples in NP

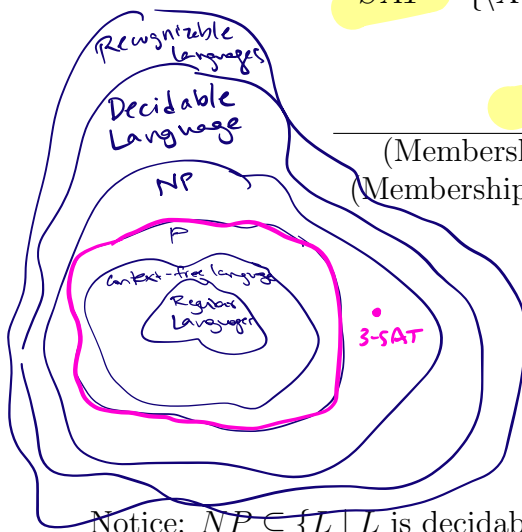
"Verifiable" i.e. NP, Can be decided by a nondeterministic TM in polynomial time, best known deterministic solution may be brute-force, solution can be verified by a deterministic TM in polynomial time.

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

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

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

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



Problems in P

(Membership in any) regular language
(Membership in any) context-free language

A_{DFA}
 E_{DFA}
 EQ_{DFA}
 $PATH$
 $RELPRIME$
...

Problems in NP

Any problem in P

SAT
 $CLIQUE$
 $VERTEX - COVER$
 $HAMPATH$
...

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

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

How do we get \$1M?

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 $x \in \Sigma^*$

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

translation property

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 $A \in P$.

Review ch 5

Proof: Let A and B be arbitrary languages

Assume ① $A \leq_P B$ and ② $B \in P$

WTS $A \in P$.

By assumption ② there is a TM with polynomial running time which decides B , call it M_B . By assumption ① there's a TM F so that, for each input x , $F(x)$ is calculated after polynomially many steps and $x \in A$ iff $F(x) \in B$.

Build

$M_A =$ "On input x

1. Calculate $F(x)$ polynomial time
2. Run M_B on $F(x)$ polynomial (polynomial) still polynomial
3. If accepts, accept ^{et}
4. If rejects, reject ^{et}

Claim: $L(M_A) = A$, M_A is a decider, running time M_A is polynomial.

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 $P = NP$.

↑
NP-hard

Proof:

If B hardest and not too hard then all of NP is not too hard.

Let B be arbitrary

Assume ① B is NP complete and ② $B \in P$

WTS $P = NP$. Already know that $P \subseteq NP$,

WTS $NP \subseteq P$. Consider arbitrary $A \in NP$

and WTS $A \in P$.

By definition of NP-completeness, by ① $A \leq_P B$.

Our previous theorem, this is enough (when combined with ②) to give $A \in P$ \square

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.

ANDs of ORs

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

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.

in NP? \checkmark
all NP problems no harder than it?

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 .

Claim: If Y is NP-complete and $Y \leq_p X$
then, for each NP problem A
 $A \leq_p X$ aka X is NP-hard.

Garey Johnson

It would be enough to prove that polynomial time reduction is transitive: if $A \leq_p Y$ and $Y \leq_p X$ then $A \leq_p X$

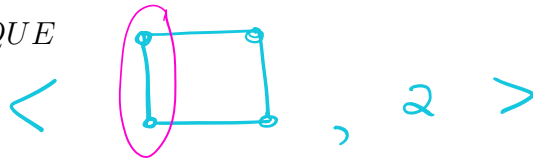
Claim: Polynomial-time and mapping reductions are both transitive.

Pf: Consider arbitrary A, Y, X and assume there's polynomial-time computable f_1 so that $\forall x (x \in A \text{ iff } f_1(x) \in Y)$ and polynomial-time computable f_2 so that $\forall x (x \in Y \text{ iff } f_2(x) \in X)$.
To prove that $A \leq_p X$ consider $f_2 \circ f_1$, well-defined efficiently computable where, for each x
 $x \in A$ iff $f_1(x) \in Y$ iff $f_2(f_1(x)) \in X$ iff $f_2 \circ f_1(x) \in X$

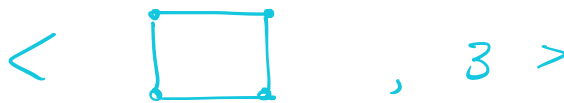
CLIQUE: A k -clique in an undirected graph is a maximally connected subgraph with k nodes.

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

Example string in *CLIQUE*



Example string not in *CLIQUE*



Theorem (Sipser 7.32):

$$3SAT \leq_P CLIQUE$$

Therefore *CLIQUE* is NP-complete

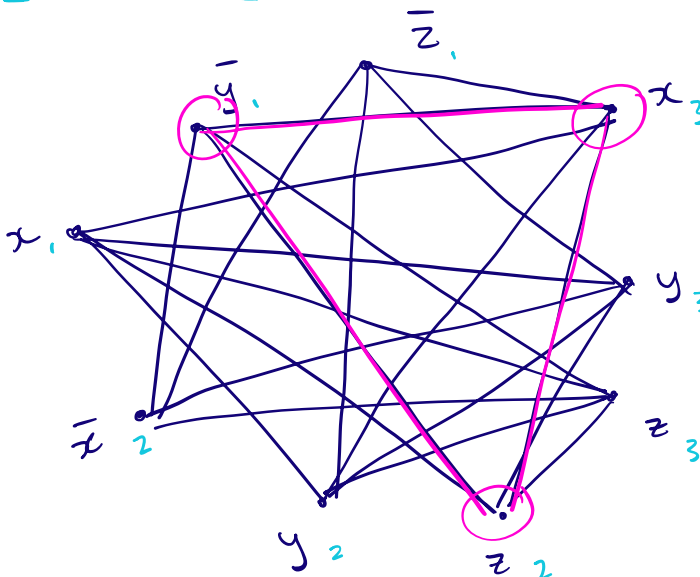
To build witnessing function need: for $x \in 3SAT$ map to graphs w/ clique. for $x \notin 3SAT$ map to graphs w/ no clique.

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

