Turing Machines, Undecidability, and Complexity

Foundations of Computability and Intractability

Computer Science Theory

Slide Set 7

Learning Objectives

By the end of this slide set, you should be able to:

- define a Turing machine (TM), configuration, recognizer, and decider;
- design a TM using implementation-level and high-level descriptions;
- explain why common TM variants have the same computational power;
- distinguish decidable, recognizable, and unrecognizable languages;
- prove undecidability using diagonalization and mapping reductions;
- apply Rice's theorem correctly; and
- distinguish computability from complexity, especially P, NP, and NP-completeness.

Road Map

1. The Standard Turing Machine
2. TM Variants and Robustness
3. Decidability and the Church–Turing Thesis
4. Undecidability and Reductions
5. The Chomsky Hierarchy
6. Computational Complexity: P and NP
7. Review and Exercises

Why Go Beyond Pushdown Automata?

A PDA has unbounded memory, but it can access that memory only in LIFO order. This restriction prevents a PDA from recognizing some simple-looking languages, such as

$$\{a^n b^n c^n \mid n \geq 0\} \qquad \text{and} \qquad \{ww \mid w \in \{0,1\}^*\}.$$

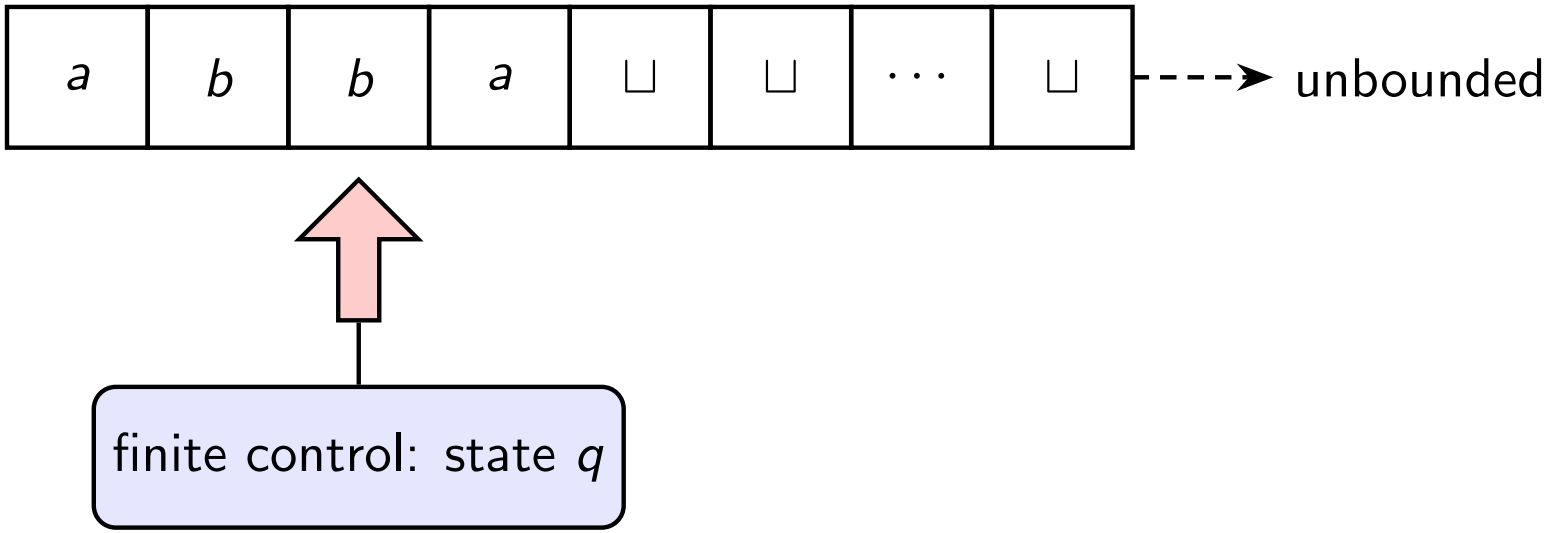
Turing's key idea

Replace the stack by an unbounded read–write tape. The head may revisit and overwrite previously scanned cells, so the machine can organize its memory rather than merely push and pop.

Important distinction

More memory access increases *computational power*; it does not guarantee that every problem becomes solvable. Some languages remain undecidable or even unrecognizable.

A Standard Single-Tape Turing Machine



- The input occupies the leftmost cells; all remaining cells contain the blank $\sqcup$.
- In one step the machine reads, changes state, writes, and moves one cell left or right.
- We use the common one-way-infinite tape convention. Two-way-infinite tapes are equivalent.

Formal Definition: Convention Used Here

We use the Sipser-style halting-state convention throughout:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where

- Q is a finite set of states; $q_{\text{accept}} \neq q_{\text{reject}}$;
- Σ is the input alphabet and $\sqcup \notin \Sigma$;
- Γ is the tape alphabet, with $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$;
- $q_0 \in Q$ is the start state; and
- $\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

Meaning of a transition

$\delta(q, a) = (p, b, R)$ means: in state q scanning a, enter p, write b, and move one cell right.

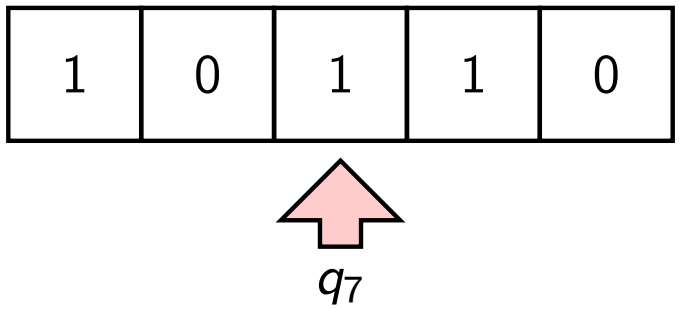
Linz often presents an equivalent final-state/partial-transition convention. The convention may change how halting is written, but not which languages can be recognized.

Configurations (Instantaneous Descriptions)

A configuration records the current state, tape contents, and head position. We write

$$uqv,$$

where the head scans the first symbol of v, u lies to its left, and q is the current state. Trailing blanks are normally suppressed.



The picture represents $10q_7110$. If $\delta(q_7, 1) = (q_5, 0, R)$, then

$$10q_7110 \vdash 100q_510.$$

The initial configuration on w is q_0w. A configuration containing q_{accept} or q_{reject} is halting.

How to Describe a TM Without Drowning in Transitions

Following the standard textbook hierarchy, use the right level of detail:

1. **Formal description:** give all states and transitions. Best for small machines.
2. **Implementation description:** describe head motions and tape operations in prose.
3. **High-level description:** describe algorithmic stages, treating the tape as structured storage.

Design pattern: crossing off symbols

Repeatedly mark one symbol in each block, sweep back to the left, and verify that all blocks finish together. New tape symbols such as X, Y, Z act as bookkeeping marks.

A complete description must say

what happens on malformed input, when the machine accepts, and when it rejects. A decider must halt on every input.

Worked Example: Deciding $\{0^n1^n \mid n \geq 0\}$

On input w:

1. First verify that $w \in 0^*1^*$; reject otherwise.
2. Sweep from the left to find the first unmarked 0; change it to X.
3. Sweep right to find the first unmarked 1; change it to Y. If none exists, reject.
4. Return to the left end and repeat.
5. When no unmarked 0 remains, accept iff no unmarked 1 remains.

Tape snapshots on input 0011

$$0011 \implies X0Y1 \implies XXYY \implies \text{accept}.$$

Each round marks one 0 and one 1, so the procedure halts after finitely many sweeps.

Extension Example: Deciding $\{a^n b^n c^n \mid n \geq 0\}$

Use markers X, Y, Z and repeat:

1. verify initially that the input has the form $a^* b^* c^*$;
2. mark the leftmost unmarked a as X;
3. move right and mark the leftmost unmarked b as Y;
4. move right and mark the leftmost unmarked c as Z;
5. return to the left end and begin the next round.

Reject if a required matching symbol is missing. When no unmarked a remains, accept exactly when no unmarked b or c remains.

Why a TM succeeds where a PDA fails

The TM can revisit all three regions and preserve several kinds of marks simultaneously; a single-stack PDA is restricted to LIFO access.

Robustness of the TM Model

Many changes make programs easier to describe without changing which languages are recognizable.

Variant	Computational power	Possible efficiency effect
stay-put move S	unchanged	constant-factor overhead
two-way-infinite tape	unchanged	small simulation overhead
multiple tapes/heads	unchanged	may be much faster
nondeterminism	unchanged for computability	central to complexity
enumerator	characterizes recognizable languages	different interface

Simulation principle

To prove two models equivalent, show how each machine in one model can be simulated by a machine in the other while preserving acceptance and rejection behavior.

Multitape Turing Machines

A k-tape TM has k tapes and k independently moving heads:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k.$$

Theorem

Every multitape TM has an equivalent single-tape TM.

Proof sketch. Encode the k virtual tapes as delimited regions on one tape and decorate each region's scanned symbol to record its virtual head. One simulated step uses sweeps to read all head symbols, update them, and move the head markers.
If a multitape machine runs in time $t(n)$, the standard simulation takes $O(t(n)^2)$ time after accounting for the single tape's repeated sweeps.

Interpretation

Same computability, potentially different running time. Model robustness does not mean equal speed.

Nondeterministic Turing Machines

An NTM may have several legal moves:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

It **accepts** an input if at least one computation branch accepts. It rejects only if all branches reject. A nondeterministic decider has no infinite branch.

Theorem

Every NTM has an equivalent deterministic TM.

Proof sketch. A DTM explores the NTM's computation tree breadth-first. Breadth-first search eventually reaches any accepting node; depth-first search may become trapped on one infinite branch.

Takeaway

Nondeterminism does not enlarge the class of computable languages, but it may change the resources needed. This distinction motivates P versus NP.

Universal Machines and Enumerators

Universal Turing machine

A universal TM U receives $\langle M, w \rangle$ and simulates machine M on input w. Both programs and data are encoded as strings—the conceptual basis of stored-program computing.

Enumerator

An enumerator is a TM with a printer. Starting from a blank tape, it prints strings over time; repetition and arbitrary order are allowed.

Enumeration theorem

A language is Turing-recognizable iff some enumerator enumerates it.

Proof idea. A recognizer can be dovetailed over all inputs to produce an enumerator; conversely, compare the enumerator's outputs with the requested input.

The Church–Turing Thesis

Church's λ-calculus and Turing's machines gave equivalent formal accounts of effective computation. Many other reasonable models were later shown equivalent as well.

Church–Turing thesis

Every function that is effectively calculable by an algorithm is Turing-computable.

It is a *thesis*, not a mathematical theorem, because "effective procedure" is an informal notion rather than a previously defined mathematical object.

Do not overstate it

The thesis identifies algorithmic computability with TM computability. Claims about what every physically possible device can compute are stronger, separate formulations.

Recognizers and Deciders

A TM on an input may accept, reject, or run forever.

Turing-recognizable / recursively enumerable

L is recognizable if some TM accepts every $w \in L$ and accepts no $w \notin L$. On a nonmember, the machine may reject or loop.

Turing-decidable / recursive

L is decidable if some TM halts on every input and accepts exactly the strings in L.

$$\text{decidable} \implies \text{recognizable}, \qquad \text{but not conversely.}$$

Recognizability and Complementation

Define

$$\mathrm{RE} = \{L \mid L \text{ is recognizable}\}, \qquad \mathrm{coRE} = \{L \mid \overline{L} \in \mathrm{RE}\}.$$

Fundamental theorem

A language L is decidable iff both L and $\overline{L}$ are recognizable.

Proof sketch. If L is decidable, swap the accept and reject states to decide $\overline{L}$. Conversely, run recognizers for L and $\overline{L}$ in dovetailing fashion. Exactly one must eventually accept, giving a halting yes/no procedure.

$$\mathrm{DEC} = \mathrm{RE} \cap \mathrm{coRE}.$$

A Useful Spectrum of Examples

Problem	Classification	Reason
A_{DFA}	decidable	simulate the DFA for exactly $\lvert w \rvert$ steps
A_{TM}	recognizable, undecidable	simulate M on w; diagonalization rules out a decider
$\overline{A_{\mathrm{TM}}}$	unrecognizable	otherwise A_{TM} and its complement would both be recognizable
Some languages over $\{0, 1\}$	unrecognizable	there are uncountably many languages but only countably many TMs

Vocabulary check

"Undecidable" does not mean "unrecognizable." The language A_{TM} is the standard counterexample.

Why Unrecognizable Languages Must Exist

1. Every TM has a finite description, so it can be encoded as a finite string.
2. The set of finite strings is countable; therefore the set of TMs is countable.
3. The set $\{0,1\}^*$ is countable, but its power set $\mathcal{P}(\{0,1\}^*)$—the set of all languages over $\{0,1\}$—is uncountable.

Hence there are strictly more languages than TMs, so some languages have no recognizer.

Cantor's diagonal idea

For any proposed list $L_1, L_2, \ldots$ and list of strings $w_1, w_2, \ldots$, define L_D so that $w_i \in L_D$ iff $w_i \notin L_i$. Then L_D differs from every L_i.

This is an existence argument; the next theorem gives a concrete undecidable language.

The Acceptance Problem A_{TM}

$$A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}.$$

First observation: A_{TM} is recognizable

On input $\langle M, w\rangle$, simulate M on w. Accept if M accepts. If M rejects, reject; if M loops, the simulator also loops.

Theorem

A_{TM} is undecidable.

The obstacle is not our inability to simulate M. It is our inability to determine in every case whether a simulation that has not halted will ever accept.

Diagonal Proof That A_{TM} Is Undecidable

Assume a decider H for A_{TM} exists. Construct D:

> *On input $\langle M \rangle$, run H on $\langle M, \langle M \rangle \rangle$. If H accepts, reject; if H rejects, accept.*

Now run D on its own encoding $\langle D \rangle$:

$$\begin{aligned} D \text{ accepts } \langle D \rangle &\iff H \text{ rejects } \langle D, \langle D \rangle \rangle \\ &\iff D \text{ does not accept } \langle D \rangle. \end{aligned}$$

This contradiction shows that H cannot exist.

Source of the contradiction

Self-reference alone is harmless; the crucial step is diagonal *negation*: D deliberately does the opposite of the alleged decider's prediction.

Mapping Reducibility

A language A is mapping reducible to B, written $A \leq_m B$, if there is a total computable function f such that

$$x \in A \iff f(x) \in B.$$

Think of f as a yes/no-preserving translator from instances of A to instances of B.

Transfer principles

- If $A \leq_m B$ and B is decidable, then A is decidable.
- If $A \leq_m B$ and A is undecidable, then B is undecidable.
- If $A \leq_m B$ and B is recognizable, then A is recognizable.

Direction matters

To prove a new problem B hard, reduce a known hard problem A *to* B.

The Halting Problem $\mathrm{HALT}_{\mathrm{TM}}$

$$\mathrm{HALT}_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ halts on } w\}.$$

Reduction from A_{TM}. Given $\langle M, w\rangle$, construct a TM N that, on any input:

1. simulates M on w;
2. halts and accepts if M accepts w;
3. loops if M rejects or loops on w.

Output $f(\langle M, w\rangle) = \langle N, \epsilon\rangle$. Then

$$\langle M, w\rangle \in A_{\mathrm{TM}} \iff \langle N, \epsilon\rangle \in \mathrm{HALT}_{\mathrm{TM}}.$$

Thus $A_{\mathrm{TM}} \leq_m \mathrm{HALT}_{\mathrm{TM}}$. Because A_{TM} is undecidable, $\mathrm{HALT}_{\mathrm{TM}}$ is undecidable.

Reduction Proof Template

To prove a target language B undecidable:

1. Choose a known undecidable language A.
2. Assume, for contradiction, that a decider for B exists.
3. Transform any instance x of A into an instance $f(x)$ of B.
4. Prove both directions: $x \in A \iff f(x) \in B$.
5. Conclude that the assumed decider for B would decide A.

Common failed proof

Showing $B \leq_m A$ only says that B is no harder than A; it does not prove B undecidable.

Also check that f is computable and handles malformed encodings.

Rice's Theorem

Theorem

Every nontrivial semantic property of the language recognized by a TM is undecidable.

"Nontrivial" means that some recognizable language has the property and another does not. "Semantic" means that the property depends only on $L(M)$, not on the machine's syntax.

- Covered: "Is $L(M) = \varnothing$?", "Is $L(M)$ regular?", "Is $L(M)$ finite?", "Does $101 \in L(M)$?"
- Not covered: "Does M have five states?", "Does M ever move left?", or directly "Does M halt on w?"

Scope

Rice's theorem proves undecidability, not automatically unrecognizability, and it concerns properties of the recognized language rather than arbitrary execution behavior.

The Formal Chomsky Hierarchy

Type	Grammar restriction	Language family	Machine model
3	right/left linear	regular	finite automaton
2	context-free rules	context-free	pushdown automaton
1	noncontracting rules	context-sensitive	linear bounded automaton
0	unrestricted	recognizable (RE)	Turing machine

For standard alphabets, the containments are proper:

$$\mathrm{REG} \subsetneq \mathrm{CFL} \subsetneq \mathrm{CSL} \subsetneq \mathrm{RE}.$$

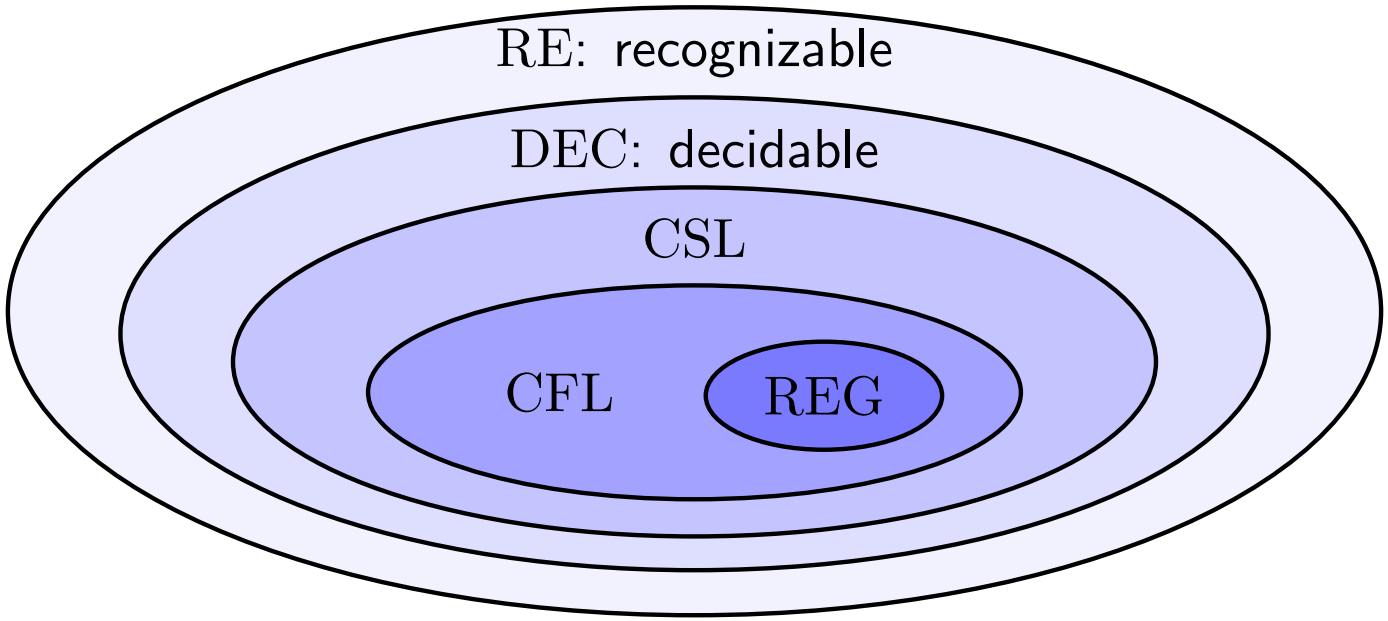
Example separators include $\{a^n b^n\}$ (CFL but not regular) and $\{a^n b^n c^n\}$ (context-sensitive but not context-free).

Type 1 permits the usual start-symbol $S \to \epsilon$ exception when S never appears on a right-hand side.

Where Do Decidable Languages Fit?

"Decidable/recursive" is not a separate Chomsky grammar type. It refines the region between context-sensitive and recognizable languages:

$$\mathrm{REG} \subsetneq \mathrm{CFL} \subsetneq \mathrm{CSL} \subsetneq \mathrm{DEC} \subsetneq \mathrm{RE}.$$



From Computability to Complexity

Complexity theory studies the resources used by *deciders*, usually as a function of input length n and in the worst case.

Time complexity

$\mathrm{TIME}(t(n))$ is the class of languages decidable by a deterministic TM in $O(t(n))$ steps.

The class P

$$P = \bigcup_{k \geq 1} \mathrm{TIME}(n^k).$$

Thus P contains languages decidable in polynomial time on a deterministic TM.

P is a robust mathematical model of efficient computation, but "polynomial" is not identical to "fast in practice": constants, exponents, input sizes, and machine models still matter.

Examples and Closure Properties of P

Typical problems in P include:

- DFA membership and graph reachability;
- shortest paths and minimum spanning tree;
- primality testing; and
- linear programming.

Useful closure facts

P is closed under union, intersection, concatenation, and complement.

Complement closure is immediate: run the polynomial-time decider and exchange accept with reject. Complexity claims require an explicit encoding because n is the length of that encoding.

The Class *NP*: Verifiers and Certificates

A language L lies in *NP* iff there are a polynomial-time verifier V and polynomial p such that

$$x \in L \iff \exists c,\ |c| \leq p(|x|) \text{ and } V(x, c) = 1.$$

Equivalently, *NP* is the class decidable in polynomial time by an NTM.

Example: CLIQUE

Instance: graph G and integer k. Certificate: a list of k vertices. Verify in polynomial time that every pair of listed vertices is connected by an edge.

Mnemonic

NP means **nondeterministic polynomial time**, not "non-polynomial." We know $P \subseteq NP$.

Polynomial-Time Reductions and NP-Completeness

$A \leq_p B$ means there is a polynomial-time computable f satisfying $x \in A \iff f(x) \in B$.

- B is **NP-hard** if every $A \in NP$ reduces to B.
- B is **NP-complete** if $B \in NP$ and B is NP-hard.

Standard proof strategy

To prove a new problem B NP-complete:

1. show $B \in NP$ by giving a verifier;
2. choose a known NP-complete problem A; and
3. construct and prove a polynomial reduction $A \leq_p B$.

Again, reducing B to a known hard problem does not establish that B is hard.

Cook–Levin Theorem: Why SAT Is NP-Complete

Theorem

SAT, the set of satisfiable Boolean formulas, is NP-complete.

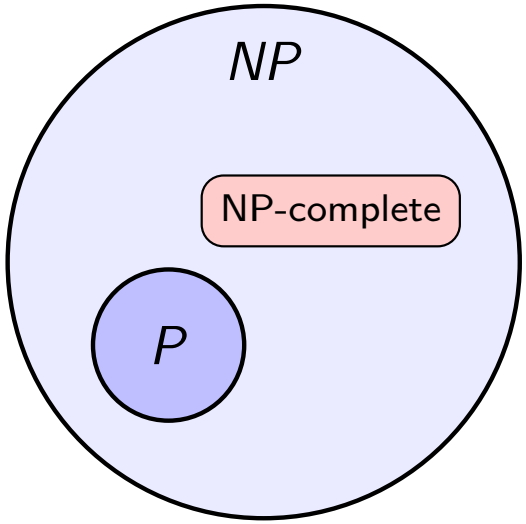
Membership in *NP*. A truth assignment is a polynomial-size certificate and can be checked by evaluating the formula.

NP-hardness proof idea. For any polynomial-time NTM computation, construct a Boolean formula describing a computation tableau. Variables encode the state, tape symbol, and head location at each time and tape position; local clauses enforce a legal start, transition sequence, and accepting state.

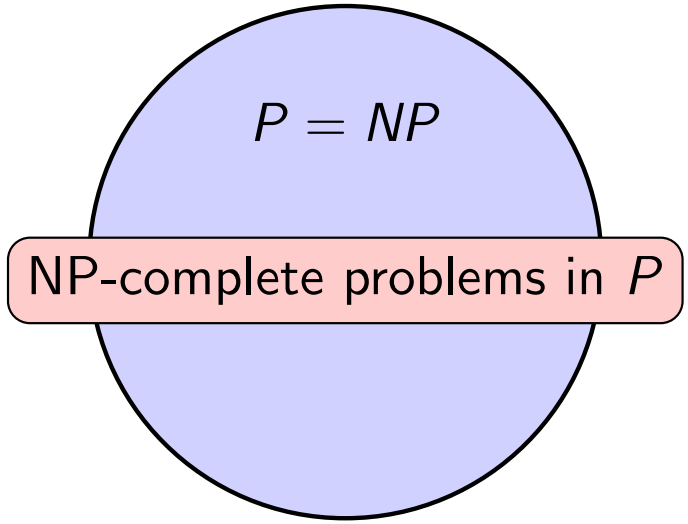
Consequence

A polynomial-time algorithm for SAT—or for any NP-complete problem—would imply $P = NP$.

The P versus NP Question



If $P \neq NP$, no NP-complete language is in P.



If $P = NP$, all NP problems have polynomial-time algorithms.

The question remains open. NP-complete problems are decidable; their suspected difficulty is about polynomial-time efficiency, not computability.

Common Conceptual Traps

Misconception	Correction
"Undecidable means the program is slow."	No algorithm halts with the correct answer on every input.
"Recognizable means decidable."	A recognizer may loop on nonmembers.
"Nondeterministic means random."	Acceptance is existential over computation branches.
"NP means not polynomial."	NP means nondeterministic polynomial time; $P \subseteq NP$.
"NP-complete means undecidable."	NP-complete problems are decidable, often by exponential-time search.
"A more convenient TM is more powerful."	Multitape and nondeterministic TMs have the same computability power.

Checkpoint Exercises I: TMs and Decidability

1. Does adding a stay-put move S increase a TM's language-recognition power? Explain a simulation.
2. Give a high-level decider for $\{w\#w \mid w \in \{0,1\}^*\}$.
3. Contrast A_{DFA} with A_{TM}. Why does direct simulation decide one but only recognize the other?
4. Suppose both L and $\overline{L}$ have recognizers. Construct a decider for L.
5. Is every undecidable language unrecognizable? Give a counterexample.

Checkpoint Exercises II: Reductions and Complexity

1. To prove a new problem B undecidable using A_{TM}, which direction should the reduction go?
2. Is "$L(M)$ contains at least five strings" covered by Rice's theorem? Is "M has at least five states"?
3. Prove that P is closed under complement.
4. Is NP known to be closed under complement?
5. What two obligations must be met to prove that a problem is NP-complete?
6. Why would a polynomial-time algorithm for $3\mathrm{SAT}$ imply $P = NP$?

Hints and Short Answers

1. Simulate S using an extra state and two moves, for example R then L.
2. Repeatedly mark the next unmatched symbol before $\#$ and compare it with the corresponding symbol after $\#$; reject malformed inputs or mismatches.
3. A DFA always halts after $|w|$ transitions; a TM simulation may run forever.
4. Dovetail the two recognizers and halt when one accepts.
5. No: A_{TM} is recognizable but undecidable.
6. Use $A_{\mathrm{TM}} \leq_m B$. The language-size property is semantic and nontrivial; state count is syntactic.
7. Swap accept/reject in a polynomial-time decider. Whether $NP = \mathrm{coNP}$ is open.
8. Show membership in NP and NP-hardness. Since $3\mathrm{SAT}$ is NP-complete, every language in NP reduces to it in polynomial time.

Summary

- TMs provide a robust formal model of algorithms and unrestricted computation.
- Deciders halt on all inputs; recognizers may loop on nonmembers.
- A_{TM} is recognizable but undecidable; diagonalization and reductions expose this limit.
- Mapping reductions transfer decidability and recognizability results in a precise direction.
- Rice's theorem covers every nontrivial semantic property of a TM's recognized language.
- Complexity asks how many resources decidable problems require; P versus NP remains open.

References and Suggested Reading

Peter Linz and Susan H. Rodger,

An Introduction to Formal Languages and Automata, 7th ed.,

Jones & Bartlett Learning, 2023.

Michael Sipser,

Introduction to the Theory of Computation, 3rd ed.,

Cengage Learning, 2013.

Recommended emphasis: Linz for machine construction and formal-language progression; Sipser for recognizability, diagonalization, reductions, and the verifier view of *NP*.