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:

- connect CFGs/PDAs to context-sensitive grammars and bounded-tape machines;
- 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. From CFGs and PDAs to CSGs and LBAs
2. The Standard Turing Machine
3. TM Variants and Robustness
4. Decidability and the Church–Turing Thesis
5. Undecidability and Reductions
6. The Chomsky Hierarchy
7. Computational Complexity: P and NP
8. 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

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

An intermediate step: change access before removing the bound

Replace the stack by a read–write tape whose usable length is bounded by the input length. A linear bounded automaton (LBA) can revisit and mark input cells, while still using only linearly bounded workspace.

Important distinction

The issue is not simply "three blocks" or "a stack matches only two things." A stack can check arbitrarily many *nested* obligations. The arrangement of dependencies, not just their number, matters.

Context-Sensitive Grammars: Rewriting in Context

Use $G = (V, T, S, P)$, with finite disjoint variable and terminal sets. A context-sensitive production has the form

$$\alpha A \beta \to \alpha \gamma \beta, \qquad A \in V, \quad \gamma \in (V \cup T)^+, \quad \alpha, \beta \in (V \cup T)^*.$$

It replaces A only in the specified surrounding context, which remains unchanged. For example, $aB \to abB$ applies to a B preceded by a.

Equivalent language-level definition

Noncontracting grammars use rules $u \to v$ where u contains a variable and $|u| \leq |v|$. They generate exactly the same language family as context-sensitive grammars, though individual rule forms differ. The length-preserving rule $CB \to BC$ illustrates the distinction.

We allow $S \to \varepsilon$ only when S never occurs on a right-hand side. A **CSL** is a language generated by such a grammar. The equivalence and this empty-word convention are discussed in [G].

Linear Bounded Automata: A Precise Convention

An LBA is a tape machine confined between two protected endmarkers:

$$B = (Q, \Sigma, \Gamma, \delta, q_0, \vdash, \dashv, F), \qquad \delta : (Q \setminus F) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

Here Q, Σ, Γ are finite, $q_0 \in Q$, $F \subseteq Q$, and $\Sigma \subseteq \Gamma$. The distinct markers lie in $\Gamma \setminus \Sigma$. The initial tape is $\vdash w \dashv$, with the head at the left marker.

- The $|w|$ interior cells may be overwritten, but the head never moves outside the marked interval.
- Markers may neither be overwritten nor written into interior cells; at $\vdash$ only a right move is allowed, at $\dashv$ only a left move.
- Reaching F halts and accepts; a nonfinal configuration with no move halts and rejects. A branch may also run forever.

Acceptance is existential over branches. A *deterministic* LBA has at most one successor per state/symbol pair. Set-valued δ is essential for the general CSL correspondence.

Worked LBA: Three Equal Blocks

Goal: recognize $L = \{a^n b^n c^n : n \geq 1\}$ using only input cells.

1. Check $w \in a^+ b^+ c^+$; reject otherwise, including ε.
2. From the left, mark the first unmarked a as X.
3. Continue right to the first unmarked b and mark it Y; then find and mark the first unmarked c as Z. Reject if a required match is missing.
4. Return to $\vdash$ and repeat. When no a remains, accept exactly when no unmarked b or c remains.

Snapshots after complete rounds, not individual transitions

$$\vdash aabbcc \dashv \Longrightarrow \vdash XaYbZc \dashv \Longrightarrow \vdash XXYYZZ \dashv .$$

After r successful rounds on $a^i b^j c^k$, the interior is $X^r a^{i-r} Y^r b^{j-r} Z^r c^{k-r}$. Thus acceptance forces $i = j = k$.

Every round marks three new cells; each sweep is bounded. This particular LBA is deterministic and halts on every input.

The Bridge Theorem and the Next Step

CSG–LBA correspondence [G]

A language is context-sensitive iff some *nondeterministic* LBA accepts it, with the stated empty-word convention.

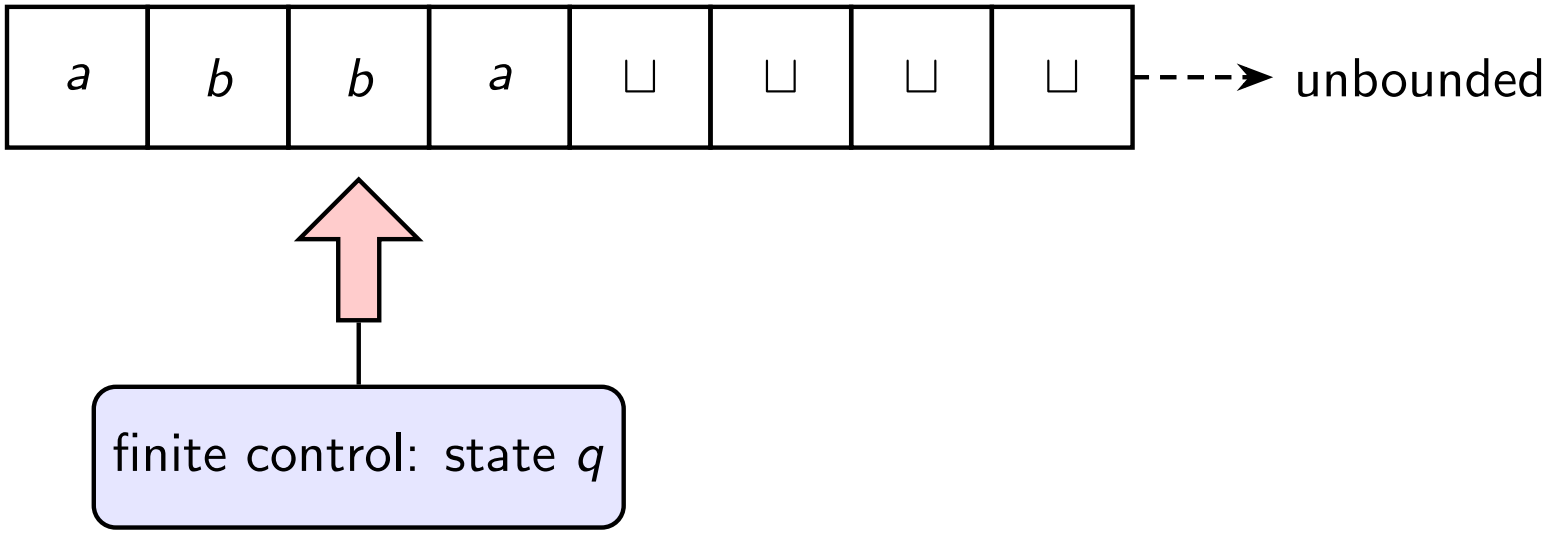
Allowing $c|w|$ work cells for a fixed constant c is equivalent to the input-cell convention: encode a fixed number of tracks in each tape cell using a larger finite tape alphabet. Handle the empty input separately.

- Unlike the PDA, the LBA can revisit and rewrite all its usable cells.
- Unlike a general TM, it cannot use arbitrarily many fresh cells on one fixed input.
- A fixed input gives finitely many configurations, so searching their graph decides LBA acceptance. The appendix makes this precise.

Do not infer that deterministic and nondeterministic LBAs are equivalent: that remains an open problem [V].

Next: remove the linear space restriction. A Turing machine may use any finite amount of tape during a finite computation—but even this model cannot decide every language.

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.
- A left move at the leftmost cell leaves the head there.

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 state set, containing distinct halting states q_{accept} and q_{reject};
- Σ is a finite input alphabet and $\sqcup \notin \Sigma$;
- Γ is a finite 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.

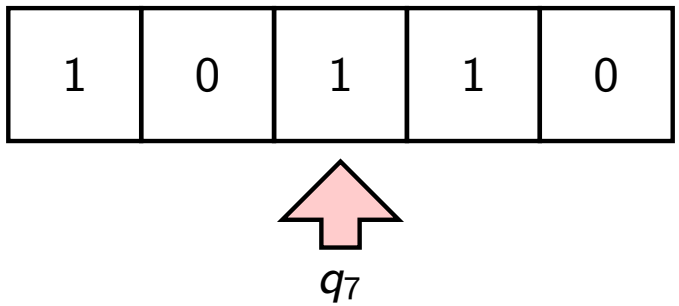
The transition function is total on nonhalting states; entering either halting state ends the computation immediately. Linz often presents an equivalent final-state/partial-transition convention. The convention may change how halting is written, but not which languages can be recognized.

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. If $v = \varepsilon$, the head scans the first implicit blank; $q_0\varepsilon$ starts on a blank cell.



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 decorated first-cell symbol lets later sweeps locate the left end; returning there is a tape operation, not a free reset of the head.

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. If none remains, go to step 5.
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

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

On 0^i1^j, after r successful rounds the tape is $X^r0^{i-r}Y^r1^{j-r}$. Acceptance therefore forces $i = j$; conversely, equal counts complete every round. Every bounded sweep terminates, and each round marks new symbols. The empty word accepts.

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

Show effective simulations in both directions. For recognizers, preserve the accepted language; for deciders, also ensure halting on every input. Enumerators require a separate interface-equivalence theorem.

Nondeterministic Turing Machines

An NTM may have several legal moves:

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

It **accepts** an input if at least one computation branch accepts. It rejects only if all branches halt and reject; a blocked nonaccepting branch counts as rejection. 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. For an NTM decider, the search also terminates if no branch accepts. The finite-branching argument justifying this is in the appendix.

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.

Encodings and Effective Simulation

Fix a finite alphabet, say binary, and a computable, unambiguous format for descriptions of machines, strings, and pairs. Write $\langle M, w \rangle$ for the resulting encoded pair, not for a new input symbol.

- The simulator can decode a finite transition table and execute one simulated step effectively.
- A machine description can itself be input data. Thus $\langle M, \langle M \rangle \rangle$ is a legitimate encoded instance.
- Malformed descriptions are rejected by the standard machine acceptance/halting languages in this set. Their complements therefore include those malformed strings.

Computing a string versus deciding a language

A TM with a designated output convention may compute a partial function $f : \Sigma^* \rightharpoonup \Sigma^*$. A decider instead computes a total yes/no membership function. Encoding a problem makes it precise; it does not guarantee that a TM can solve it.

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. Alternate one step of each simulation; an infinite run in one must not prevent the other from progressing. All complements use the same universe Σ^*.

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

A Useful Spectrum of Examples

Problem	Classification	Reason
A_{DFA}	decidable	simulate the DFA for exactly $\lvert w\rvert$ steps
A_{LBA}	decidable	explore the finite configuration graph for the given input
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

Vocabulary check

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

Here $A_{\mathrm{DFA}} = \{\langle D, w\rangle : D \text{ accepts } w\}$; A_{LBA} is defined analogously.

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 (reject malformed input):

> *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. if M rejects, enters an explicit infinite loop. If M never halts, N simply remains in the simulation.

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

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

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

Constructing code is not running it

The translator writes M, w into a finite simulation program and halts; it need not know M's outcome. Map malformed inputs to a fixed looping machine paired with ε, so f is total.

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.

A property outside Rice's scope may still be undecidable; the theorem simply does not settle it. A proof sketch appears in the appendix.

The Formal Chomsky Hierarchy

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

As families over finite alphabets, the containments are proper:

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

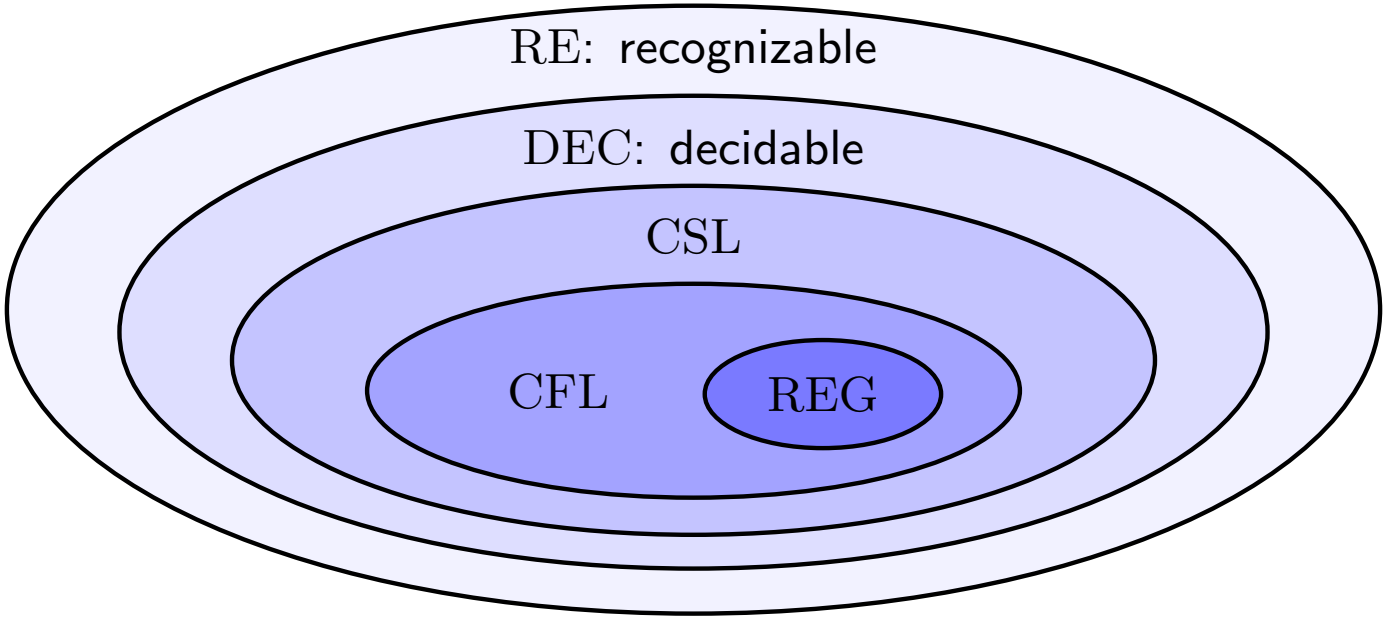
Example separators include $\{a^n b^n\}$ (CFL but not regular) and $\{a^n b^n c^n\}$ (context-sensitive but not context-free).

Type 1 permits the usual start-symbol $S \to \varepsilon$ 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:

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



$\mathrm{CSL} \subseteq \mathrm{DEC}$ follows from finite configuration search. Strictness requires a space-hierarchy/diagonalization argument; it does not follow merely because TMs have more available tape.

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.

DFA membership and directed graph reachability are examples in P. The appendix gives further examples and closure proofs.

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.

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: a simple undirected graph G and nonnegative integer k. Certificate: k *distinct* vertex identifiers from G. Reject $k > |V(G)|$; verify every pair is joined by an edge. The certificate has polynomial length even when k is encoded in binary.

Mnemonic

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

V must halt in time polynomial in $|x| + |c|$ on every pair. Checking one proposed certificate is different from finding a successful one.

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:

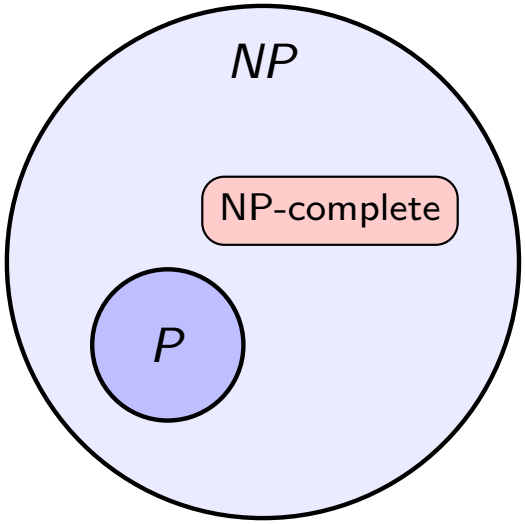
1. show $B \in NP$ by giving a verifier;
2. choose a known NP-complete problem A; and
3. construct and prove a polynomial reduction $A \leq_p B$.

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

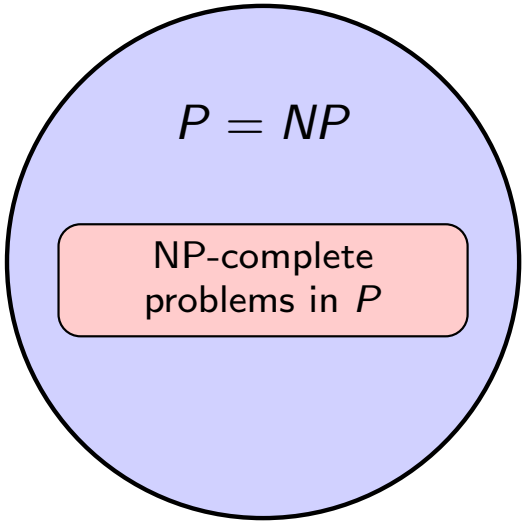
Anchor theorem: Cook–Levin and its 3SAT corollary

SAT (satisfiable Boolean formulas) and 3SAT (satisfiable conjunctions of three-literal clauses) are NP-complete. Thus a polynomial-time algorithm for either would imply $P = NP$. Proof sketches and a worked reduction to CLIQUE are in the appendix.

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 [CMI]. Every NP language is decidable: enumerate the finitely many certificates of permitted length and run the verifier on each. This may require far more than polynomial time. The issue is 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 $P = NP$?

Checkpoint Exercises III: Bounded Space and Encodings

1. On *aabbccc*, what is the tape after two rounds of the LBA's matching routine? Why must the final check reject?
2. An LBA has 4 states, 3 interior cells, 2 possible symbols per interior cell, and two fixed endmarkers. Give a configuration upper bound when the head may also scan either marker.
3. One branch of an NLBA repeats a configuration. Must the input be rejected? Contrast this with a deterministic run.
4. Why does deciding membership for each fixed LBA input not decide whether its language is empty?
5. A proposed reduction simulates $M(w)$ to completion *before* producing its output. What can go wrong?

Write down the quantifiers: a statement about *one* input is not automatically a statement about *all* input lengths.

Summary

- CSGs correspond to nondeterministic LBAs; bounded-input configuration search makes every CSL decidable, but not necessarily easy to decide.
- 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.

Appendix Guide

The core sequence retains the CSG/LBA bridge, TM design, diagonalization, mapping reductions, Rice's theorem, and the definitions of P and NP. Select appendix material according to course depth:

- CSG examples, finite configuration graphs, and LBA emptiness;
- a complete transition table, multitape simulation, and termination;
- checkpoint answers and five solved construction/proof challenges;
- Rice's theorem, TM equivalence, and different kinds of reduction;
- closure of P, Cook–Levin, SAT-to-3SAT, and 3SAT-to-CLIQUE.

The six relocated enrichment frames are retained in full. Attempt each challenge before reading its solution; the source guide distinguishes edition-specific textbook problems from newly phrased course practice. Check the acceptance mode, encoding, quantifiers, and reduction direction.

Separate Counts Are Not the Same as One Shared Count

Compare two four-block languages:

Context-free: two independent adjacent matches

$L_1 = \{a^n b^n c^m d^m : n, m \geq 0\}$ has the CFG

$$S \to AB, \qquad A \to aAb \mid \varepsilon, \qquad B \to cBd \mid \varepsilon.$$

A PDA finishes matching a's against b's, then reuses its stack for the c's and d's.

Context-sensitive but not context-free: one count shared four ways

$L_2 = \{a^n b^n c^n d^n : n \geq 0\}$ is checked by four-way marking on an LBA. If it were a CFL, the homomorphism deleting d and retaining a, b, c would make $\{a^n b^n c^n : n \geq 0\}$ a CFL, a contradiction.

Here we use CFL closure under homomorphism from Slide Set 5. For L_2, accept ε separately before checking nonempty blocks.

A Noncontracting Grammar for Three Equal Blocks

For $\{a^n b^n c^n : n \geq 1\}$, use variables S, B, C and rules

$$\begin{aligned} S &\to aSBC \mid aBC, & CB &\to BC, \\ aB &\to ab, & bB &\to bb, \\ bC &\to bc, & cC &\to cc. \end{aligned}$$

All rules are noncontracting; the swap $CB \to BC$ is not itself in context-preserving form. The language-equivalence theorem relates the two grammar definitions.

Deriving *aabbcc*

$$\begin{aligned} S &\Rightarrow aSBC \Rightarrow aaBCBC \Rightarrow aaBBCC \\ &\Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcC \Rightarrow aabbcc. \end{aligned}$$

Each S expansion supplies one a, B, C. Swaps preserve these counts; terminalization preserves them too. A terminal derivation must have all b's before all c's: a B stranded to the right of a terminal c cannot be converted. Conversely, sort $(BC)^n$ to $B^n C^n$ before terminalizing to obtain every $a^n b^n c^n$.

How Many Configurations Can an LBA Have?

Fix an LBA B and an input w of length n. In our convention, there are n writable cells and two fixed marker cells. Let g be the number of symbols allowed in an interior cell. A configuration specifies:

$$\underbrace{|Q|}_{\text{state}} \times \underbrace{(n+2)}_{\text{head position}} \times \underbrace{g^n}_{\text{interior tape contents}} =: C.$$

This is an upper bound; many configurations may be unreachable. If markers and their head positions are excluded by another convention, the often-quoted bound is $|Q|ng^n$ instead.

Deterministic halting on one given input

Simulate the unique run, stopping if it halts. If $C+1$ configurations have been visited without halting, some repeats. Determinism forces the same future, so this run loops forever. Alternatively, store visited configurations and detect repetition directly.

For $|Q| = 4, n = 3, g = 2$, the bound is $4 \cdot 5 \cdot 8 = 160$. It also handles $n = 0$: the head can still scan either marker.

For an NLBA, Search a Graph—Not One Branch

For fixed (B, w), create a vertex for each reachable configuration and an edge for each legal step. Treat accepting and blocked configurations as terminal. The graph is finite even if some runs are infinite.

Question about this input	Finite-graph test
Is w accepted?	Is an accepting configuration reachable?
Does some branch halt?	Is any terminal configuration reachable?
Do all branches halt?	Is the reachable nonterminal subgraph acyclic?

A reachable cycle can support an infinite branch *and* have an exit leading to acceptance. Thus one repeated configuration on one branch is not a rejection certificate for an NLBA.

Consequence

$A_{\mathrm{LBA}} = \{\langle B, w\rangle : B \text{ accepts } w\}$ is decidable. Hence every CSL is decidable. The external graph-search decider need not itself obey the LBA's linear space restriction.

The Catch: LBA Emptiness Is Undecidable

$$E_{\mathrm{LBA}} = \{\langle B \rangle : L(B) = \emptyset\}.$$

Finite configuration search settles one (B, w), not every possible input length. There is no fixed global finite graph covering all inputs.

Computation-history reduction [S20]

Given $\langle M, w \rangle$, construct an LBA B whose input is a proposed finite accepting computation history of M on w. Check the initial configuration, each consecutive transition, and the accepting final configuration; reject malformed histories.

These checks compare finite fields by sweeps and marks within the supplied history. Fixed extra tracks can be packed into the tape alphabet. Thus B can even be deterministic and halt on every history it checks.

There is an accepted history iff M accepts w, so

$$A_{\mathrm{TM}} \leq_m \overline{E_{\mathrm{LBA}}}.$$

A decider for emptiness, with its answer reversed, would decide A_{TM}. Malformed source encodings map to a fixed empty-language LBA. This does not contradict decidable membership for every individual CSL.

Optional Complexity: LBA Acceptance and PSPACE

PSPACE consists of decision languages solvable by a deterministic TM in space polynomial in the encoded input length.

Uniform acceptance problem [B21]

A_{LBA} is PSPACE-complete: the input contains *both* the LBA description and its word, with the tape bound enforced by the model. This is not a claim about every individual fixed CSL.

Why polynomial space suffices: one configuration and a counter up to C take polynomial space. Guess a path of length less than C; Savitch's theorem converts this nondeterministic polynomial-space test to a deterministic polynomial-space one.

Why it is hard: a polynomial-space decider can be simulated on an LBA input padded to supply its polynomially many tape cells. The padding and machine description are constructible in polynomial time.

Deterministic-LBA halting is PSPACE-complete as well: for hardness, modify this simulation to halt on acceptance and loop on rejection. A large configuration bound alone proves neither PSPACE-hardness nor an unconditional exponential-time lower bound.

A Complete Small TM: Scan and Flip Bits

Let $Q = \{q_s, q_{\text{accept}}, q_{\text{reject}}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$, and start in q_s. These are *all* transitions on nonhalting states:

scanned symbol	0	1	$\sqcup$
$\delta(q_s, \cdot)$	$(q_s, 1, R)$	$(q_s, 0, R)$	$(q_{\text{accept}}, \sqcup, R)$

On 01, the configurations are

$$q_s 01 \;\vdash\; 1 q_s 1 \;\vdash\; 10 q_s \sqcup \;\vdash\; 10 \sqcup q_{\text{accept}} \sqcup .$$

Two questions about the same machine

As a *recognizer*, it accepts $\{0, 1\}^*$, including ε. As a *string transformer*, with the output taken before the first blank, it computes bitwise complementation: $01 \mapsto 10$.

For $0 \le r \le |w|$, after r steps exactly the first r bits have been flipped. The first blank is reached after $|w|$ steps; one further step accepts. An unreachable reject state is allowed.

Multitape Turing Machines

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

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k.$$

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 the multitape machine runs in time $t(n) \geq n$, with k fixed, 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.

Why a Halting NTM Has a Finite Computation Tree

Fix one input and unfold its computations into a rooted tree: a node is a computation prefix, and its children are the finitely many next moves.

König's lemma: the relevant form

Every infinite, finitely branching rooted tree has an infinite branch.

Proof idea: if the root has infinitely many descendants, one of its finitely many children must have infinitely many descendants. Choose such a child repeatedly; the choices build an infinite branch.

Therefore, if *every* computation branch halts, the tree must be finite. Breadth-first simulation eventually exhausts it, accepting if any branch accepts and otherwise rejecting.

Where the hypotheses matter

Infinitely many finite branches need not have bounded depth in an *infinitely branching* tree. TMs avoid that issue because their transition descriptions have finite branching. For a recognizer with an infinite branch, breadth-first search need not terminate on a nonmember.

A finite tree for each input establishes termination, not a polynomial running-time bound.

Solutions I: TMs and Decidability

1. Simulate $(q, a) \mapsto (p, b, S)$ by writing b and moving right to a helper state; preserve the encountered symbol, move left, and enter p. Moving right first also works at the left boundary.
2. For $w\#w$, check that exactly one $\#$ occurs. Mark the next unmarked bit in the left half, remember it in finite control, and match the next unmarked bit on the right. Reject mismatches or unequal lengths. Accept when both halves are exhausted, including the input $\#$. Each round marks new cells, so the procedure halts.
3. A DFA consumes exactly one symbol per step, so its run finishes after $|w|$ steps. A simulated TM may continue forever.
4. Alternate steps of the two recognizers. Accept if the L recognizer accepts; reject if the $\overline{L}$ recognizer accepts. Exactly one event must eventually occur.
5. No. A_{TM} is recognizable but undecidable.

Solutions II: Reductions and Complexity

1. Use $A_{\mathrm{TM}} \leq_m B$: a hypothetical decider for B would decide the known undecidable source.
2. "At least five strings" is semantic and nontrivial: $\emptyset$ fails and Σ^* satisfies it. "At least five states" is syntactic and can be checked directly from the description.
3. Swap the two halting outcomes of a polynomial-time decider; its running time is unchanged.
4. It is not known whether NP is closed under complement; this is equivalent to $NP = \mathrm{coNP}$. Here $\mathrm{coNP} = \{L : \overline{L} \in NP\}$. Do not confuse this with P's known complement closure.
5. Prove both $B \in NP$ and NP-hardness. For the latter, reduce a known NP-complete language *to* B in polynomial time.
6. Every $A \in NP$ reduces to 3SAT in polynomial time. Composing that reduction with a polynomial-time 3SAT decider puts A in P. Together with $P \subseteq NP$, this gives $P = NP$.

Solutions III: Bounded Space and Encodings

1. The interior is $XXYYZZc$. The final scan finds an unmatched c and rejects; seeing no unmarked a alone is not enough.
2. $4(3+2)2^3 = 160$ configurations at most, including head positions on the two endmarkers.
3. Not necessarily: another branch may accept, or a later choice may exit the cycle. In a deterministic run, repeating a nonhalting configuration forces the same infinite future.
4. Emptiness asks whether *any* word of *any* length is accepted. Testing words one by one can find a witness to nonemptiness but cannot generally certify its absence.
5. If $M(w)$ loops, that translator never outputs anything, so it is not a total mapping reduction. Write a program containing the simulation instead; the translator itself must always terminate.

In all three topics, distinguish the finite description of a process from the potentially infinite behavior of the process it describes.

Challenge Sources and Study Route

These prompts are paraphrased, with worked arguments written for this course. Exercise numbers depend on the edition.

Challenge	Source / status
1. Ordered enumeration and decidability	Sipser, 3rd ed., Problem 3.18
2. An infinite decidable subset	Sipser, 3rd ed., Problem 3.19
3. An effective list of deciders is incomplete	Sipser, 3rd ed., Problem 4.30
4. A head that may move right or stay	Sipser, 2nd ed., Problem 3.13
5. Find a SAT witness using a decision procedure	Additional course practice; no numbered exercise claimed

University sources verify the numbered prompts [I22, T07]. Linz's construction viewpoint supports Challenge 4; Sipser's enumeration and diagonalization viewpoint guides Challenges 1–3.

Questions to ask before checking the solution

What guarantees progress? Does the argument handle finite and empty languages? Does it prove that a decider exists, or give a uniform algorithm to obtain one from the supplied description?

Challenge 1: Ordered Enumeration and Decidability

Sipser 3rd ed., Problem 3.18. Show that L is decidable iff some enumerator prints its words in increasing *standard string order*. Here this means length first, then dictionary order (shortlex):

$$\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$$

Decider $\Rightarrow$ ordered enumerator

Visit all strings in shortlex order. Run the decider on each and print exactly those it accepts. Every test halts, so every word of L is eventually printed, in order.

Ordered enumerator $\Rightarrow$ existence of a decider

If L is infinite, on input x simulate the enumerator until it prints x (accept) or a word after x (reject). There are only finitely many words at or before x, so one event must occur.

If L is finite, it is decidable by a finite membership table, including the empty-table case. This is a separate case, not the same simulation.

A finite-language enumerator may run silently forever after its last output. The theorem does not supply an end-of-enumeration test.

A Subtle Point: Existence Is Not Uniform Construction

Challenge 1 does *not* give an algorithm that converts every ordered enumerator into a decider for its language.

Why such a general converter cannot exist

From $\langle M, w \rangle$, effectively build an enumerator E that simulates $M(w)$ and prints ε only if that simulation accepts. It prints nothing otherwise. Its language is either $\{\varepsilon\}$ or $\emptyset$, and in either case its outputs are ordered.

A supposed converter would produce a decider D_E. Running $D_E(\varepsilon)$ would then decide whether M accepts w, contradicting the undecidability of A_{TM}.

Both possible languages individually have easy deciders; the impossible task is choosing the correct one *effectively from this description*.

Under the additional promise that the enumerated language is infinite, the simulation from Challenge 1 *is* a uniform construction. This distinction also explains why infinitely many decidable fixed-input cases need not yield a decision procedure for a uniform problem.

Challenge 2: An Infinite Decidable Subset

Sipser 3rd ed., Problem 3.19. Prove that every infinite recognizable language L contains an infinite decidable subset B.

Construction: retain only new length records

Run an enumerator for L. Retain its first output; thereafter retain an output only when its length is strictly greater than the length of the last retained word. Let B be the set of retained words.

Subset and infinitude. Every retained word belongs to L. Over a finite alphabet, only finitely many words have bounded length. Since L is infinite, longer outputs keep appearing; hence B is infinite.

Decider for B. On input x, simulate the retention process. Accept if it retains x; reject once it retains a word longer than x. Retained lengths are strictly increasing and unbounded, so this halts.

Do not test against every output of the original enumerator

Only retained words belong to B. A discarded output equal to x does not justify acceptance. Also, this result does not make L decidable: an undecidable language may contain an easy infinite subset.

Challenge 3: No Effective List Covers All Decidable Languages

Sipser 3rd ed., Problem 4.30. Let A be a recognizable set of TM descriptions over a fixed input alphabet, each describing a decider. Show that some decidable language is not decided by any machine whose description belongs to A.

Diagonal construction when A is infinite

Enumerate its distinct descriptions as $\langle M_0 \rangle, \langle M_1 \rangle, \ldots$, suppressing repeats. List all input words $w_0, w_1, \ldots$ in shortlex order. Define a machine D:

1. On w_i, enumerate until M_i appears.
2. Run M_i on w_i and reverse its accept/reject answer.

The enumeration reaches its ith distinct description because A is infinite; M_i halts because it is a decider. Thus D is a decider, but $L(D) \neq L(M_i)$ for every i, witnessed by w_i.

If A is finite, its machines decide only finitely many languages; choose a singleton language not among them.

All TM descriptions can be listed, but no effective list containing *only deciders* can represent every decidable language.

Challenge 4: Remove Left Moves, Lose Language Power

Sipser 2nd ed., Problem 3.13. A deterministic TM may write, move right, or stay, but never move left. Show it recognizes only regular languages, even though its tape is unbounded.

Summarize processing of one cell

Starting with state q and scanned symbol a, simulate stay moves until the machine accepts, rejects, moves right in state p, or repeats a state/symbol pair. Repetition implies a local infinite loop; there are at most $|Q||\Gamma|$ such pairs.

Make these summaries the transitions of a DFA. Early acceptance goes to an accepting sink; rejection or local looping goes to a rejecting sink. A right move goes to p: abandoned tape cells can never be read again.

After input ends, the TM may traverse fresh blank cells. Summarize each blank-cell visit similarly; only finitely many entry states exist. Mark a DFA state final iff this blank-only continuation eventually accepts. Thus the DFA has exactly the same language. Conversely, a right-moving TM can simulate any DFA and decide at the first blank.

Adding stay moves to L/R changes no power; *replacing* left by stay does.

Challenge 5: Find a SAT Witness Using Yes/No Answers

Additional practice: given a polynomial-time SAT decider D, construct a polynomial-time algorithm that returns a satisfying assignment for a formula φ, or reports that none exists.

1. Ask $D(\varphi)$. If the answer is no, report unsatisfiable.
2. Maintain a partial assignment with a satisfying extension. For each unassigned variable x_i, ask whether φ together with the current assignments and $x_i = 0$ is satisfiable.
3. If yes, keep $x_i = 0$; otherwise set $x_i = 1$. Continue until every variable is assigned.

Why the invariant survives

Before each query, at least one extension exists. Such an extension sets x_i either to 0 or to 1. If the 0 branch has no extension, the 1 branch must have one. The final complete assignment therefore satisfies φ.

For m variables, at most $m + 1$ queries are made; appended assignment constraints keep every query polynomial in the original input size. This is an adaptive *search-to-decision* reduction, not a single mapping reduction between two decision languages.

A Reduction That Also Shows Unrecognizability

Let $E_{\mathrm{TM}} = \{\langle N \rangle : L(N) = \emptyset\}$. Given a valid $\langle M, w \rangle$, output a machine N that on input x simulates $M(w)$ and accepts x only if that simulation accepts.

$$L(N) = \begin{cases} \Sigma^*, & M \text{ accepts } w, \\ \emptyset, & M \text{ does not accept } w. \end{cases}$$

Map malformed source strings to a fixed machine with empty language. The map is total and satisfies

$$\overline{A_{\mathrm{TM}}} \leq_m E_{\mathrm{TM}}.$$

Since $\overline{A_{\mathrm{TM}}}$ is unrecognizable, E_{TM} is unrecognizable too: otherwise compute the map and run an E_{TM} recognizer.

But nonemptiness is recognizable

Dovetail N on all strings and accept when any simulation accepts. Also accept malformed descriptions, since these belong to $\overline{E_{\mathrm{TM}}}$ under our fixed encoding convention. Hence E_{TM} is co-recognizable but not recognizable.

Rice's Theorem: Why the General Pattern Works

Let $\mathcal{P}$ be a nontrivial property of recognizable languages. First suppose $\emptyset \notin \mathcal{P}$; choose a recognizable language $K \in \mathcal{P}$ with recognizer T.

Construct a gated recognizer

Given $\langle M, w \rangle$, output N whose action on x is: simulate $M(w)$; if it accepts, run $T(x)$ and copy its outcome. If the first simulation rejects, reject; if it loops, keep simulating.

$$L(N) = K \text{ if } M \text{ accepts } w; \qquad L(N) = \emptyset \text{ otherwise.}$$

Thus deciding whether $L(N) \in \mathcal{P}$ would decide A_{TM}. Invalid source encodings map to an empty-language machine.

If $\emptyset \in \mathcal{P}$, apply the argument to the complementary *property* among recognizable languages and reverse the decision. This is not taking the complement of K or claiming it is recognizable.

The proof uses both hypotheses: nontriviality supplies K, and semantic dependence makes the two language outcomes decisive.

TM Equivalence: Neither Side Is Recognizable

Define $\mathrm{EQ}_{\mathrm{TM}} = \{\langle M_1, M_2\rangle : L(M_1) = L(M_2)\}$. Fix machines E, U for $\emptyset, \Sigma^*$. From $\langle M, w\rangle$, construct N that accepts its input exactly when the simulation of $M(w)$ accepts. Thus

Behavior of $M(w)$	$L(N)$	$L(N) = L(E)$?	$L(N) = L(U)$?
accepts	Σ^*	no	yes
rejects or loops	$\emptyset$	yes	no

The two computable maps $\langle M, w\rangle \mapsto \langle N, E\rangle$ and $\langle M, w\rangle \mapsto \langle N, U\rangle$ respectively establish

$$\overline{A_{\mathrm{TM}}} \leq_m \mathrm{EQ}_{\mathrm{TM}}, \qquad \overline{A_{\mathrm{TM}}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}.$$

For malformed source strings, use $N = E$; both equivalences still hold. Since $\overline{A_{\mathrm{TM}}}$ is unrecognizable, neither target is recognizable.

Stronger than undecidability

There is no recognizer for all equivalent pairs, nor for all inequivalent pairs. Rice's theorem alone would not establish these two stronger claims. See Sipser's discussion of mapping reductions [2, MIT].

Mapping Reductions Are Not Oracle Reductions

Write $A \leq_T B$ if an algorithm decides A using a hypothetical *oracle* that correctly answers membership queries for B. Queries may be adaptive; the oracle need not be computable.

An oracle can answer a question whose answer we then reverse

Always $A \leq_T \overline{A}$: ask whether $x \in \overline{A}$ and flip the answer. A mapping reduction instead requires one total computable f with $x \in A \iff f(x) \in B$; it cannot reverse that equivalence.

In fact, $A_{\mathrm{TM}} \not\leq_m \overline{A_{\mathrm{TM}}}$. If such an f existed, then

$$x \notin A_{\mathrm{TM}} \iff f(x) \in A_{\mathrm{TM}}.$$

Computing $f(x)$ and running an A_{TM} recognizer would recognize $\overline{A_{\mathrm{TM}}}$, a contradiction.

Hence $A \leq_m B$ implies $A \leq_T B$, but the converse fails. The SAT-witness challenge likewise uses several adaptive decision queries; it is not a single mapping reduction.

State the reduction you are proving

Oracle access, a total computable translation, and a polynomial-time translation are different requirements. See [2], Chapter 6.

Examples and Closure Properties of P

P is a class of *decision languages*. Examples include:

- DFA membership and graph reachability;
- whether a graph with nonnegative binary-encoded integer weights has an s–t path of weight at most K;
- whether an undirected weighted graph has a spanning tree of total weight at most K.

Useful closure facts

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

Complement closure: run the decider and exchange accept with reject. For concatenation, try all $n+1$ splits $w = uv$ and run the two deciders; polynomially many polynomial-time tests remain polynomial.

Finding a shortest path or a minimum spanning tree is an associated function/optimization problem, also solvable in polynomial time. Complexity claims require an explicit encoding because n is the length of that encoding.

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 restriction 3SAT is also NP-complete. Fresh variables for subformulas convert SAT to clauses of at most three literals without exponential expansion; the next slide outlines this construction.

SAT to 3SAT: Avoiding Exponential Expansion

Regard a formula as a tree of two-input AND/OR and one-input NOT gates. Introduce a fresh variable z for each subformula's value. Enforce its relation to its children with constant-size clauses, e.g.

$$z \leftrightarrow (x \wedge y) \quad \Longleftrightarrow \quad (\neg z \vee x) \wedge (\neg z \vee y) \wedge (z \vee \neg x \vee \neg y).$$

For $z \leftrightarrow \neg x$, use $(z \vee x) \wedge (\neg z \vee \neg x)$; OR gates have analogous clauses. Add the unit clause asserting the root.

Why satisfiability is preserved

A satisfying assignment to the original variables extends by giving each new variable its subformula value. Conversely, the gate clauses force those values, and the root clause forces the original formula true.

Each clause has at most three literals. Repeating a literal pads short clauses to exactly three occurrences without changing their meaning. The number of variables and clauses is linear in the formula's tree size; writing their identifiers still takes only polynomial time [E].

The new formula is *equisatisfiable*, not logically equivalent over the enlarged variable set. This construction establishes SAT $\leq_p$ 3SAT.

A Concrete Polynomial Reduction: 3SAT to CLIQUE

Let $\varphi = C_1 \wedge \cdots \wedge C_m$ have three literal occurrences per clause. Construct a graph with one vertex for each occurrence.

1. Put no edges between occurrences belonging to the same clause.
2. Between different clauses, join two vertices exactly when their literals are not contradictory (x and $\neg x$).
3. Output (G, k) with $k = m$.

The two directions

A satisfying assignment supplies one true literal per clause. These m vertices are pairwise compatible, hence form a clique.

An m-clique uses one vertex from each clause. Its literals contain no contradictory pair, so assign them true and extend arbitrarily to other variables. Every clause then has a true literal.

There are $3m$ vertices and $O(m^2)$ candidate edges; comparing literal names also takes polynomial time in the formula encoding length. Since 3SAT is NP-complete and CLIQUE is in NP, CLIQUE is NP-complete. Malformed formulas map to a fixed no-instance: an empty graph with $k = 1$.

References: Numbered Challenge Sources

[S06] Michael Sipser, *Introduction to the Theory of Computation*, 2nd ed., Thomson Course Technology, 2006. Problem 3.13: TMs with right and stay-put moves only.

[I22] University of Illinois Urbana–Champaign, CS 475, Spring 2022, *Enumerators*. Identifies Sipser Problems 3.18, 3.19, and 4.30 and clarifies the standard (shortlex) order of strings.

[T07] Luca Trevisan, UC Berkeley CS 172, February 12, 2007, *Problem Set 4*. Identifies the second-edition right/stay-put machine problem.

Statements have been paraphrased and solutions developed for these slides. Use the edition-specific numbers in the challenge guide: exercise numbers can change between editions. The SAT-witness task and the uniform-conversion discussion are supplementary course material, not claimed numbered exercises.

References: CSGs, LBAs, and Bounded Space

[G] Jean Gallier, University of Pennsylvania, *Phrase-Structure Grammars and Context-Sensitive Grammars*, Chapter 8. Context-preserving/noncontracting grammars and LBA equivalence.

[V] VSB–Technical University of Ostrava, Introduction to Theoretical Informatics, *Turing Machines*. LBA restrictions and the determinism question.

[S20] Michael Sipser, MIT 18.404J, Fall 2020, Lecture 10: Computation History Method. LBA acceptance versus emptiness and history checking.

[B21] UC Berkeley CS 172, Spring 2021, Problem Set 9 Solutions. PSPACE-completeness of bounded-tape acceptance.

References: Computability and Complexity

[MIT] Michael Sipser, MIT 18.404J, Fall 2020, Lecture Notes. Turing machines, undecidability, reductions, and complexity; Lecture 15 includes 3SAT $\leq_p$ CLIQUE.

[E] Jason Eisner, Johns Hopkins University, *How to Convert a Formula to CNF*. Fresh-variable encodings and equisatisfiability.

[CMI] Clay Mathematics Institute, *P vs NP*. Problem statement and open status; consulted September 2026.

The configuration-graph and reduction proofs here spell out the model and encoding conventions used in this slide set.

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*.