

CMSC 28100

Introduction to Complexity Theory

Autumn 2025

Instructor: William Hoza



Course Review

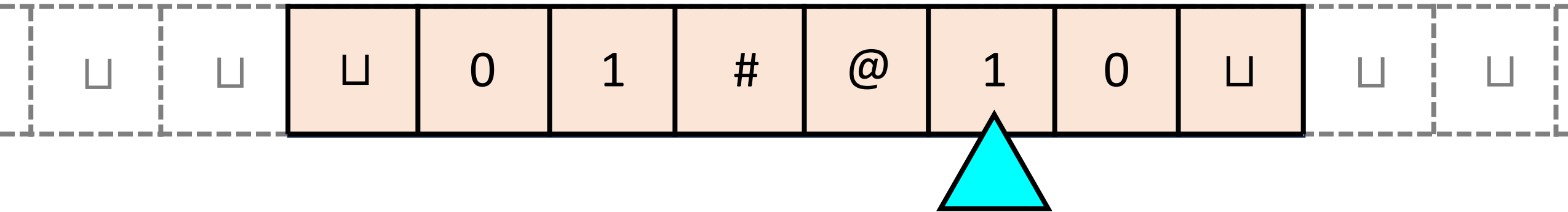
Which problems
can be solved
through computation?

Strings and languages

- Σ^* is the set of all strings over the alphabet Σ (of any finite length)
- A binary **language** is a subset $Y \subseteq \{0, 1\}^*$
- Corresponding **problem**: Given $w \in \{0, 1\}^*$, figure out whether $w \in Y$
 - To study other types of problems, we can often formulate a **closely related** language
 - E.g., Exercise 14: Searching for large cliques

Which problems
can be solved
through computation?

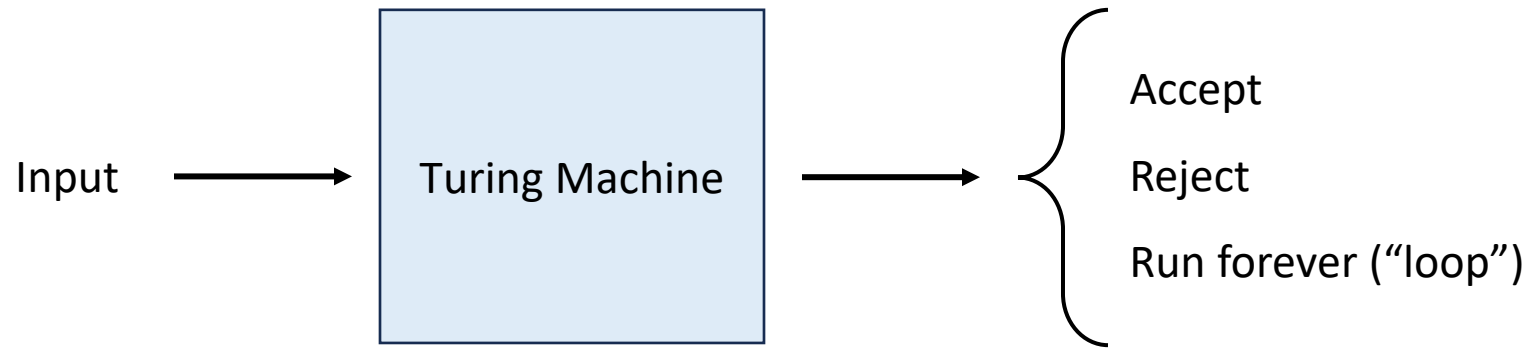
Turing machines



- **Tape** grows when necessary
- **Head** can read from tape / write to tape
- Internal **state**
- “Local evolution” described by **transition function** $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$

Which problems
can be solved
through computation?

Deciding a language



- Let M be a Turing machine and let $Y \subseteq \{0, 1\}^*$
- We say that M **decides** Y if M accepts every $w \in Y$ and M rejects every $w \in \{0, 1\}^* \setminus Y$

The Church-Turing Thesis

- Let $Y \subseteq \{0, 1\}^*$

Church-Turing Thesis:

The problem of deciding whether a given string is in Y
can be “solved through computation” if and only if
there is a Turing machine that decides Y .

← Intuitive notion
← Mathematically
precise notion

The Physical Church-Turing Thesis

- Let $Y \subseteq \{0, 1\}^*$

Physical Church-Turing Thesis:

It is physically possible to build a device that decides Y if and only if there is a Turing machine that decides Y .

Code as data

- A Turing machine M represents an algorithm
- At the same time, M can be encoded as a **string** $\langle M \rangle$
- This string $\langle M \rangle$ could be the input or output of a different algorithm!

Universal Turing machines

Theorem: There exists a Turing machine U such that for every Turing machine M and every $w \in \{0, 1\}^*$:

- If M accepts w , then U accepts $\langle M, w \rangle$.
- If M rejects w , then U rejects $\langle M, w \rangle$.
- If M loops on input w , then U loops on $\langle M, w \rangle$.

Universal Turing machines



- If you are trying to build a computer from scratch, **your job is to build a universal Turing machine**
- A universal Turing machine **can be “programmed” to do anything** that is computationally possible

Undecidability

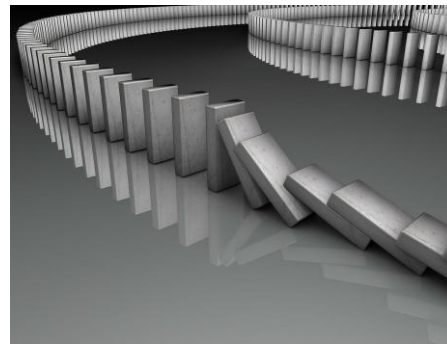
- $\text{SELF-REJECTORS} = \{\langle M \rangle : M \text{ rejects } \langle M \rangle\}$



Theorem: SELF-REJECTORS is undecidable

- We can rule out **all possible algorithms!**
- Proof technique: “Diagonalization”

Reductions



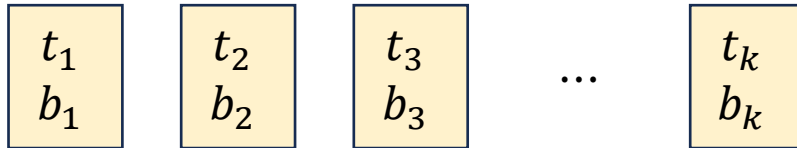
- To prove that Y_{NEW} is **undecidable**:
 - Identify some known undecidable language Y_{OLD}
 - Assume for the sake of contradiction that Y_{NEW} is **decidable**
 - Design an algorithm that decides Y_{OLD} , using hypothetical device deciding Y_{NEW}
- Example: $\text{HALT} = \{\langle M, w \rangle : M \text{ halts on } w\}$

Theorem: HALT is undecidable

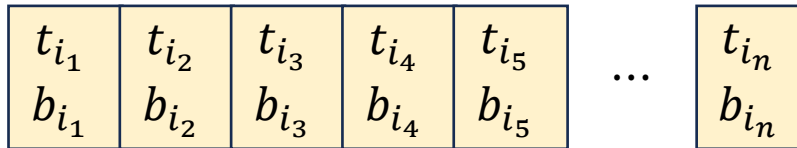


Post's Correspondence Problem

- Input: A set of “dominos”



- Goal: Determine whether it is possible to generate a “match”



in which the sequence of top symbols equals the sequence of bottom symbols

Theorem: PCP is undecidable

Asymptotic analysis

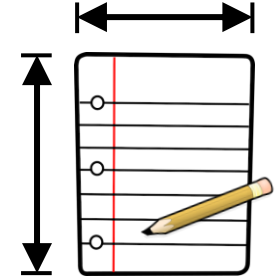
Notation	In words	Analogy
T is $o(f)$	$T(n)$ grows more slowly than $f(n)$	$<$
T is $O(f)$	$T(n)$ is at most $c \cdot f(n)$	\leq
T is $\Theta(f)$	$T(n)$ and $f(n)$ grow at the same rate	$=$
T is $\Omega(f)$	$T(n)$ is at least $c \cdot f(n)$	\geq
T is $\omega(f)$	$T(n)$ grows more quickly than $f(n)$	$>$

Polynomial-time computation



- We mainly focus on the distinction between **polynomial-time** algorithms and **exponential-time** algorithms
- We proved that $n^k = o(2^n)$ for every constant k
- Exponential-time algorithms are **almost worthless**
- Polynomial-time algorithms are typically **usable**

Complexity classes



- A **complexity class** is a set of binary languages
- $Y \in \mathbf{P}$ if Y can be decided by a polynomial-time TM
- $Y \in \mathbf{PSPACE}$ if Y can be decided by a polynomial-space TM
- $Y \in \mathbf{EXP}$ if Y can be decided by a TM with time complexity $2^{\text{poly}(n)}$

The Time Hierarchy Theorem

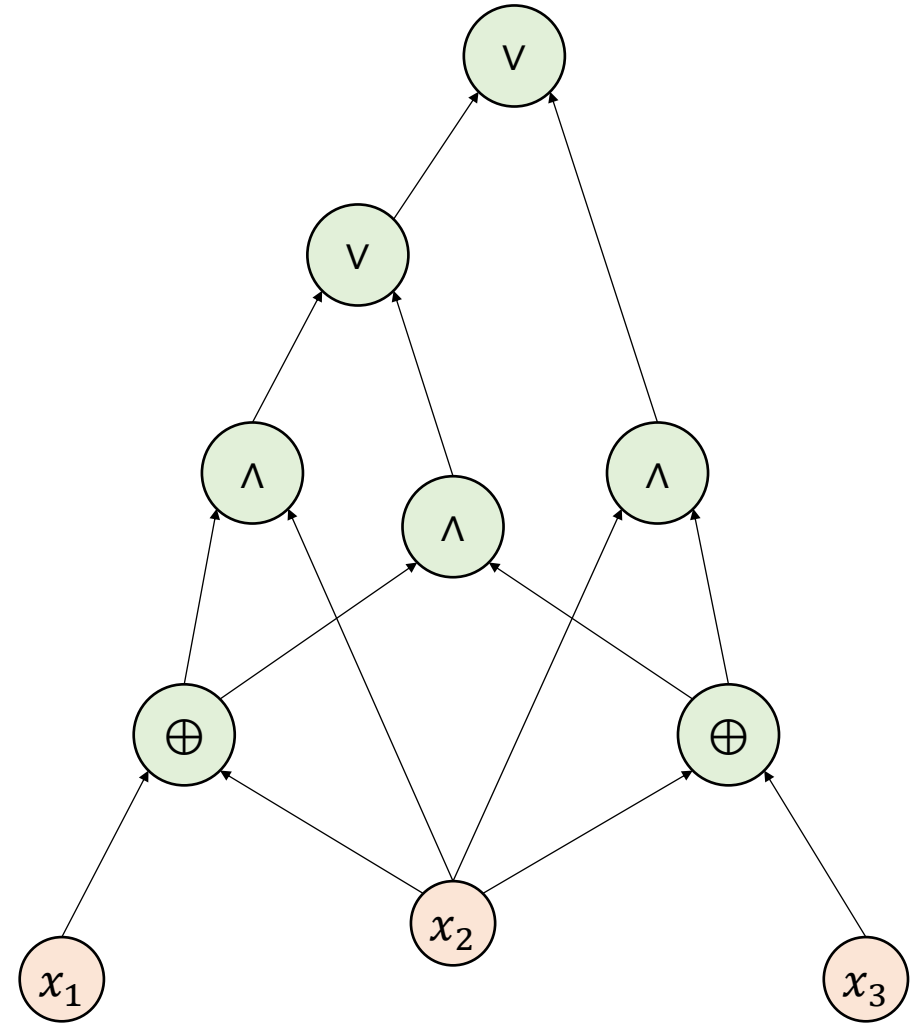
- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be any “reasonable” (time-constructible) function

Time Hierarchy Theorem: There is a language $Y \in \text{TIME}(T^4)$
such that $Y \notin \text{TIME}(o(T))$

- Consequence: $P \neq EXP$

Boolean circuits

- A “circuit” is a network of binary logic gates applied to Boolean variables



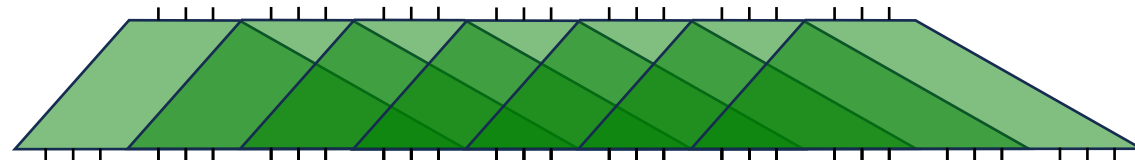
Circuit complexity

- DNF/CNF representation: Every function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ can be computed by a circuit of size $O(2^n \cdot n)$
- Exercise 17: There exists a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ with circuit complexity $\Omega(2^n/n)$

Turing machine \Rightarrow circuits

- Suppose Y can be decided by a Turing machine with time complexity $T(n)$ and space complexity $S(n)$

Theorem: For each n , there is a circuit of size $O(T(n) \cdot S(n))$ that decides Y restricted to inputs of length n



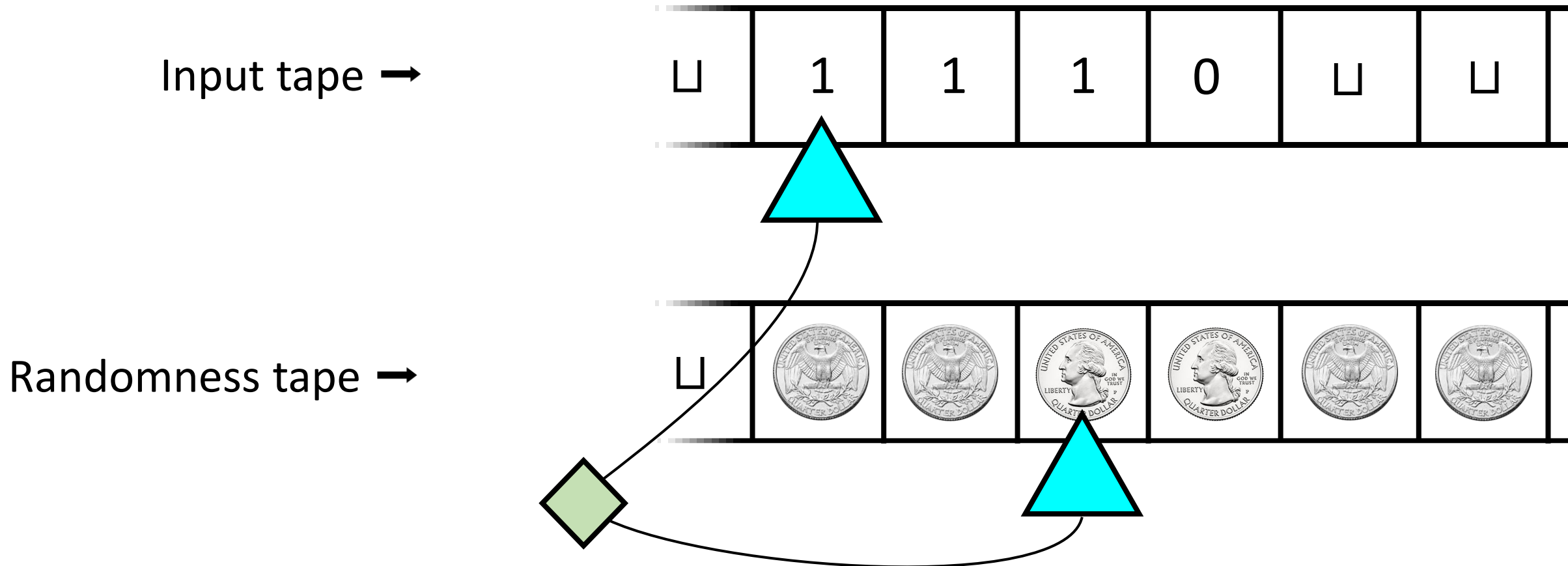
Circuits \Leftrightarrow Computing with advice



- $Y \in \text{PSIZE}$ if for every n , there is a poly-size circuit that decides Y restricted to inputs of length n
- $Y \in \text{P/poly}$ if Y can be decided in polynomial time given a polynomial amount of **advice** that depends only on the **length** of the input

Theorem: $\text{PSIZE} = \text{P/poly}$

Randomized Turing machines



The complexity class NP

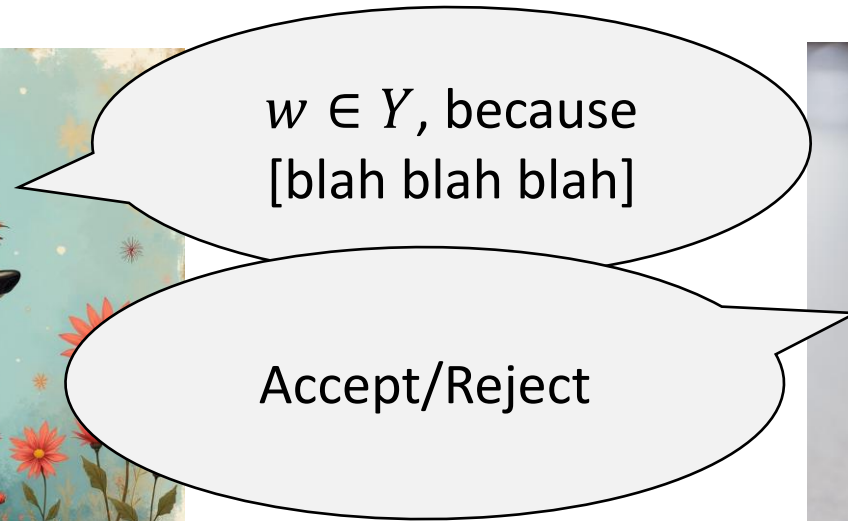


- A language Y is in NP if there is a polynomial-time randomized Turing machine M such that:
 - For every $w \in Y$, we have $\Pr[M \text{ accepts } w] \neq 0$
 - For every $w \notin Y$, we have $\Pr[M \text{ accepts } w] = 0$
- Equivalent: Every $w \in Y$ has a **certificate** of membership, and certificates can be **verified** in (deterministic) polynomial time

Prover and verifier

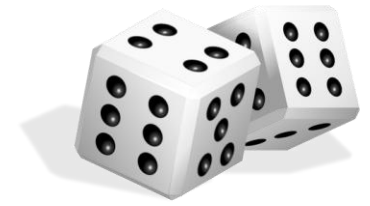


Prover
(Computationally
Unbounded)



Verifier
(Polynomial Time)

The complexity class BPP

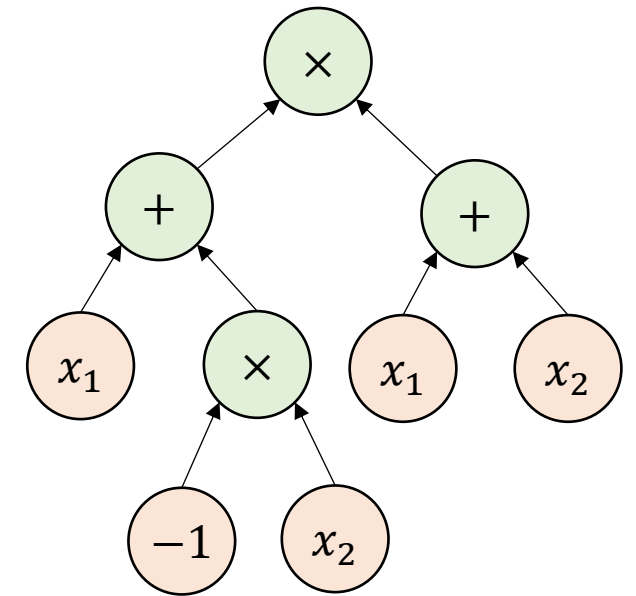


- A language Y is in **BPP** if there is a polynomial-time **randomized** Turing machine M such that:
 - For every $w \in Y$, we have $\Pr[M \text{ accepts } w] \geq 2/3$
 - For every $w \notin Y$, we have $\Pr[M \text{ rejects } w] \geq 2/3$
- **Amplification Lemma:** We can replace $2/3$ with $1 - 1/2^{n^k}$

Example: Identity testing

- **Given:** An arithmetic formula F
- **Goal:** Figure out whether $F \equiv 0$

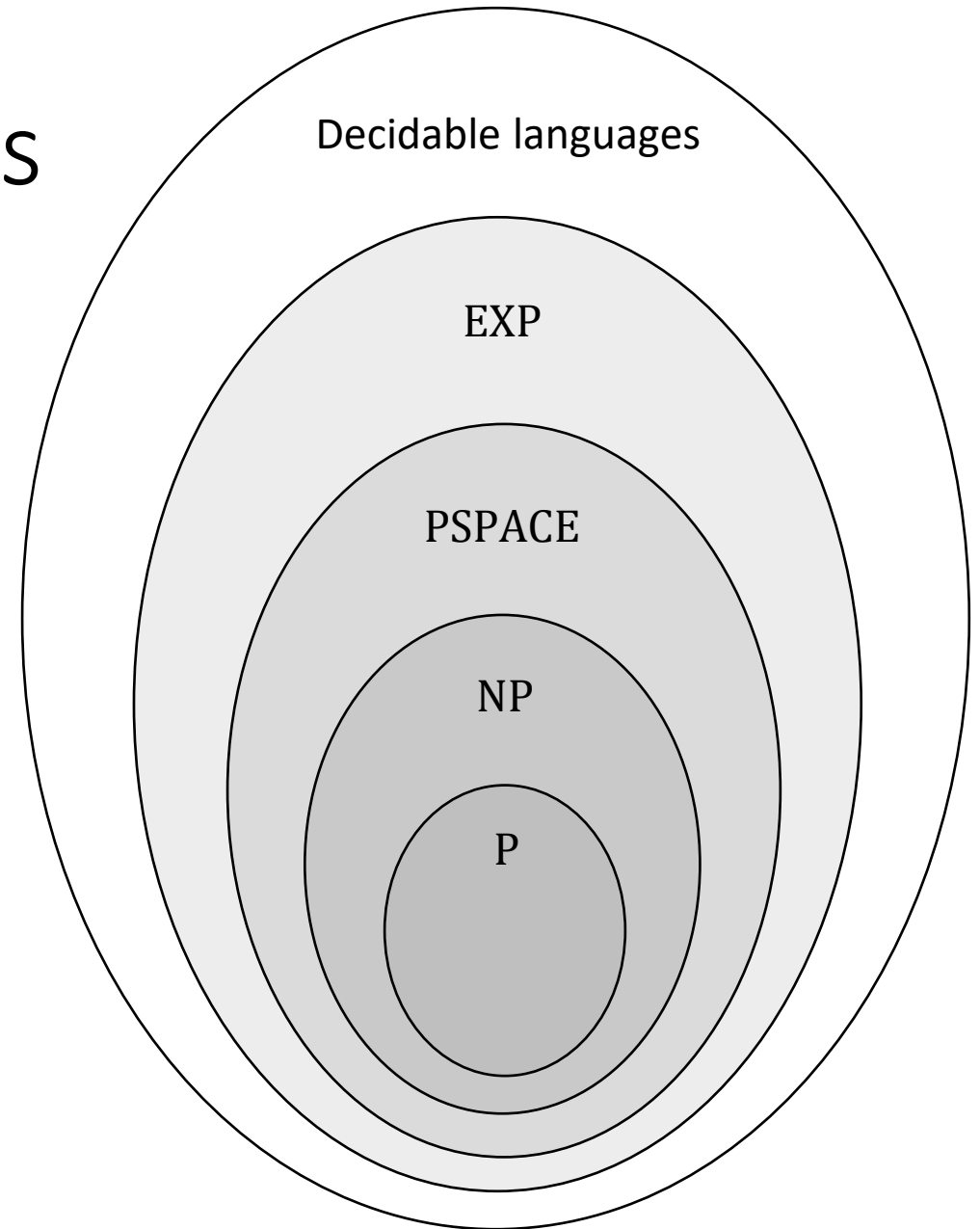
Theorem: IDENTICALLY-ZERO \in BPP



- **Algorithm idea:** Pick \vec{x} at random and check whether $F(\vec{x}) = 0$

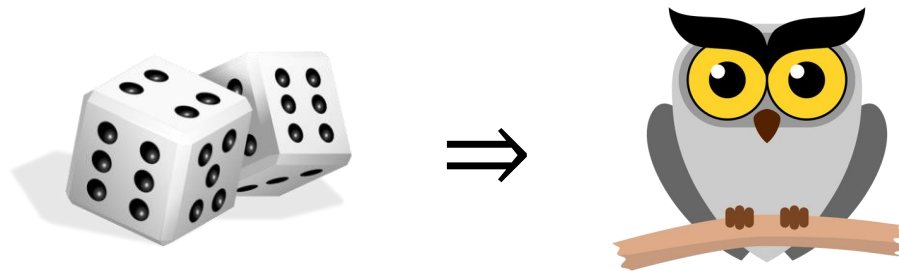
Complexity class inclusions

- $P \subseteq BPP$ and $P \subseteq NP$
- $BPP \subseteq PSPACE$ and $NP \subseteq PSPACE$
by brute force
- $PSPACE \subseteq EXP$ because we can't
repeat a configuration

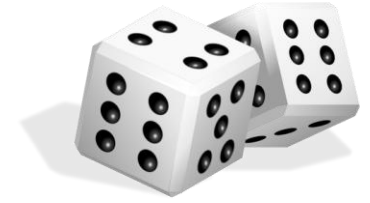


Adleman's theorem

Adleman's Theorem: $BPP \subseteq P/poly$



P vs. BPP



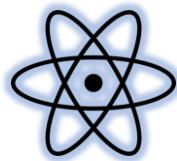
- To prove that certain problems are **intractable**, we need a mathematical **model** of tractability
- P and BPP are both reasonable models of tractability
- Which should we use?
- **Conjecture:** **P = BPP**, so it's a moot point

Extended Church-Turing Thesis

Extended Church-Turing Thesis:

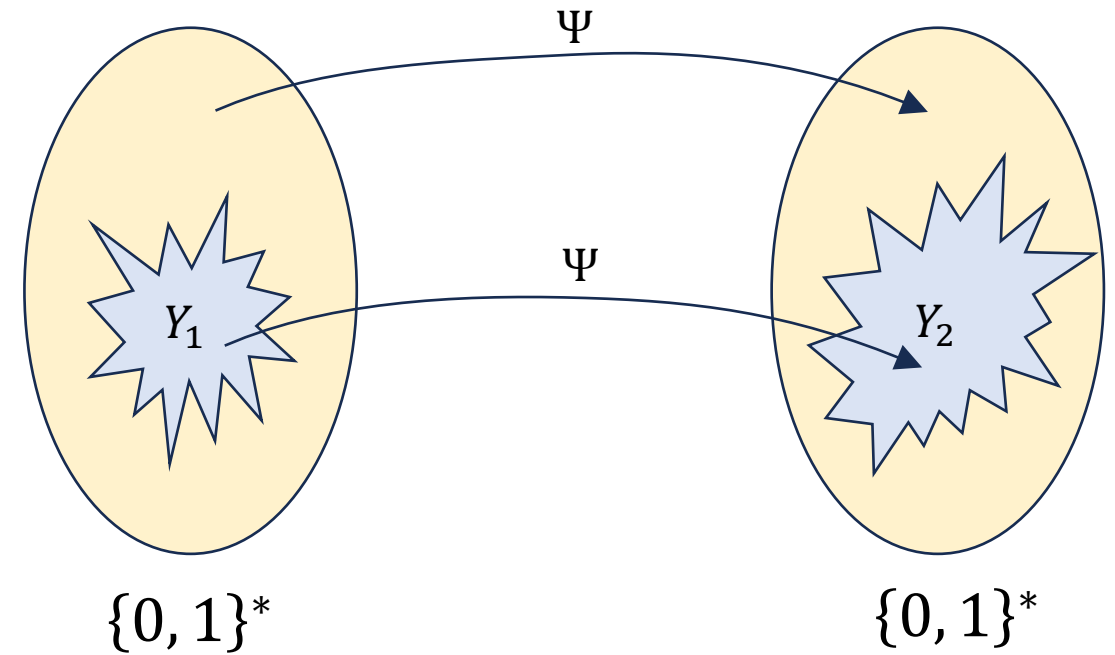
For every $Y \subseteq \{0, 1\}^*$, it is physically possible to build a device that decides Y in polynomial time if and only if $Y \in P$.

- The Extended Church-Turing Thesis is probably false because of quantum computing



Mapping reductions

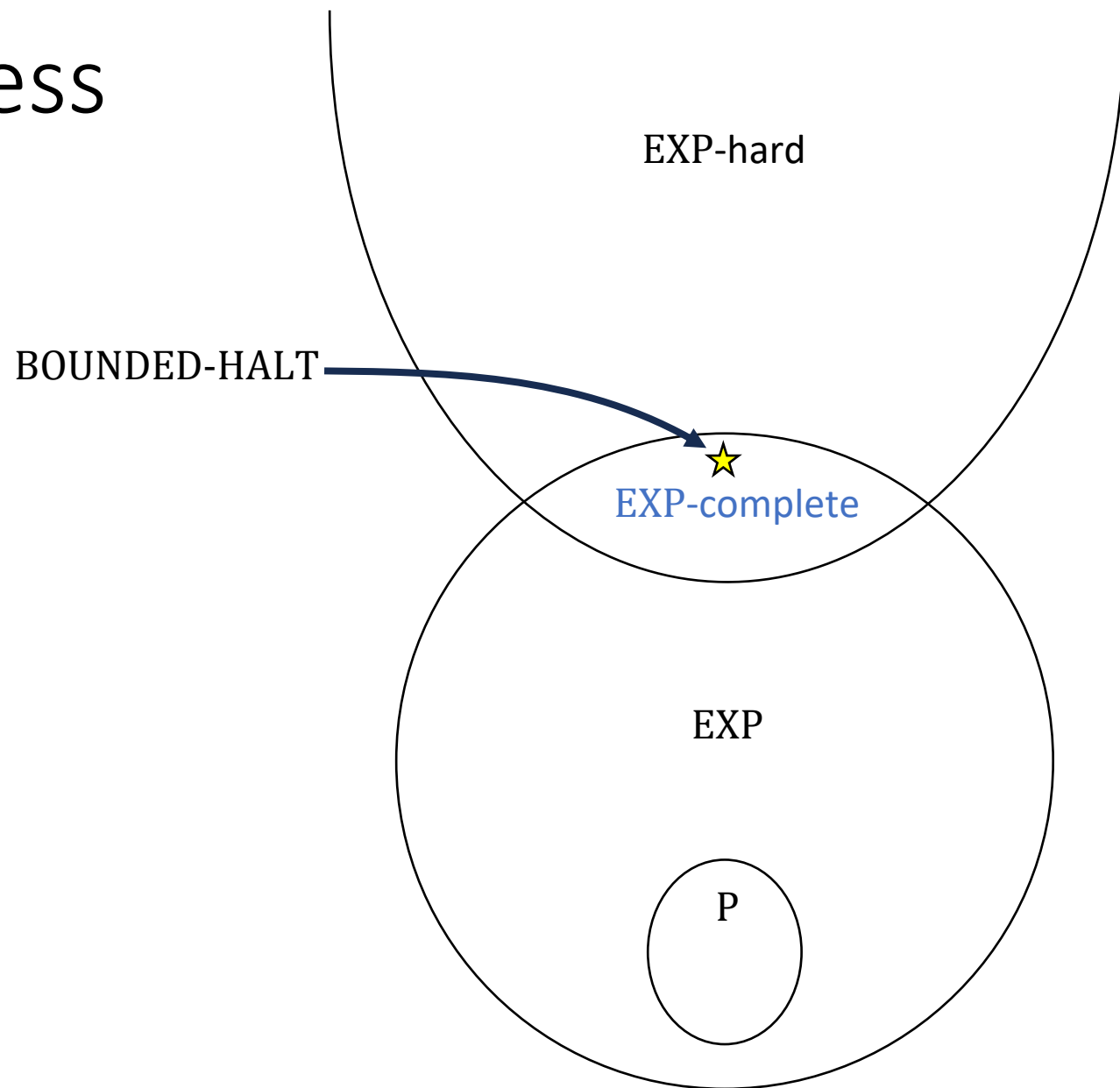
- Let $Y_1, Y_2 \subseteq \{0, 1\}^*$
- **Definition:** $Y_1 \leq_P Y_2$ if there is a poly-time computable function $\Psi: \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that
 - $w \in Y_1 \Rightarrow \Psi(w) \in Y_2$
 - $w \notin Y_1 \Rightarrow \Psi(w) \notin Y_2$



EXP-completeness

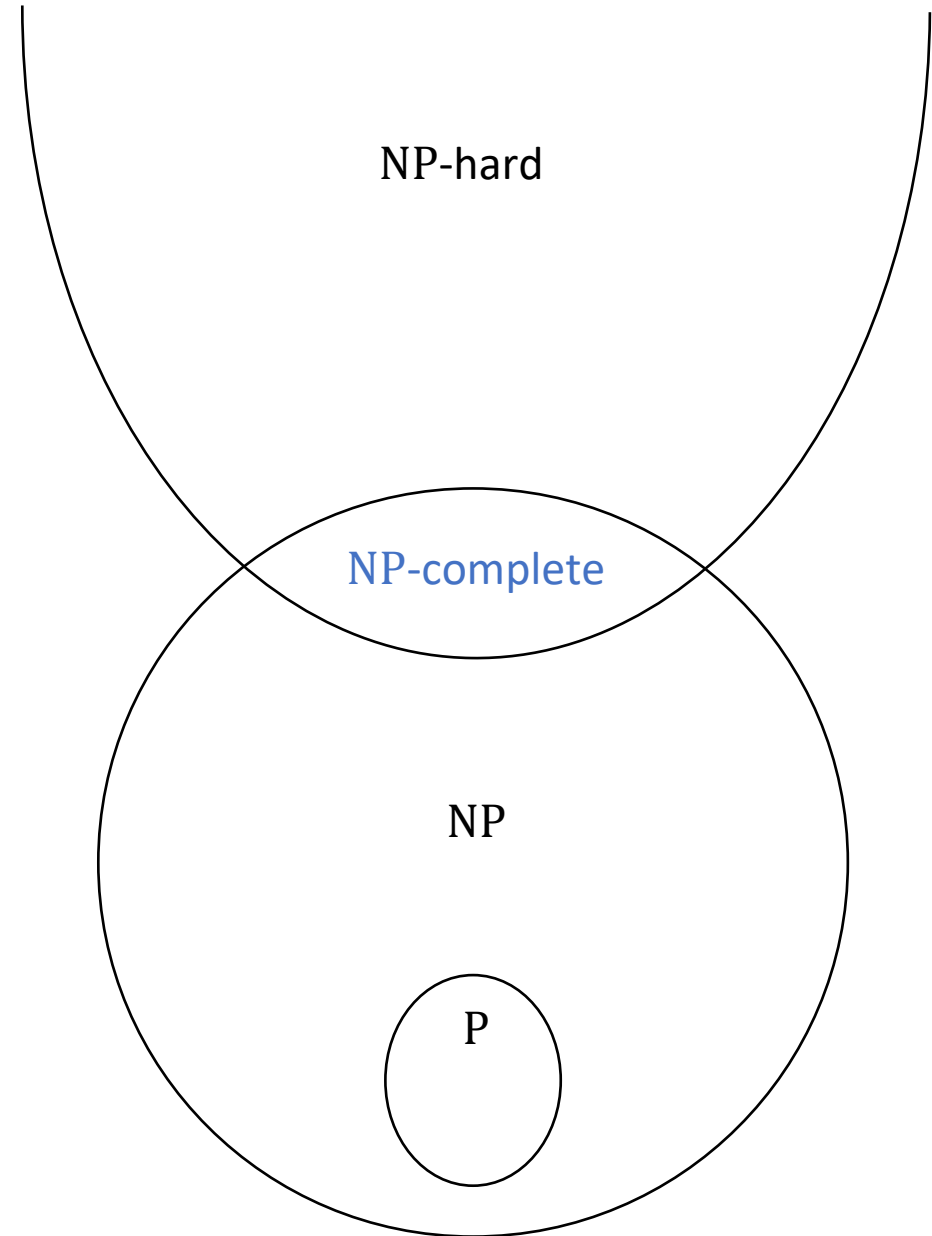
- We say that Y is **EXP-hard** if $Z \leq_P Y$ for **every** $Z \in \text{EXP}$
- We say that Y is **EXP-complete** if Y is EXP-hard and $Y \in \text{EXP}$
- EXP-complete languages are not in P
- Example: BOUNDED-HALT is EXP-complete

EXP-completeness



NP-completeness

- A language Y is **NP-complete** if $Y \in \text{NP}$ and $Z \leq_p Y$ for every $Z \in \text{NP}$

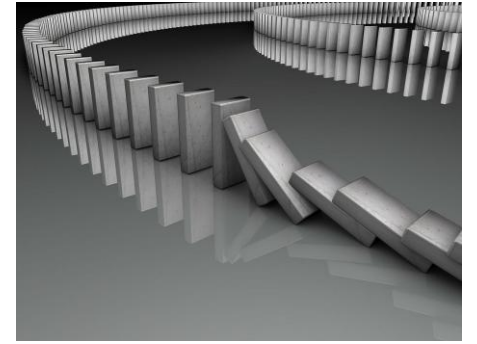


Circuit satisfiability

- $\text{CIRCUIT-SAT} = \{\langle C \rangle : C \text{ is a satisfiable circuit}\}$

Theorem: CIRCUIT-SAT is NP-complete

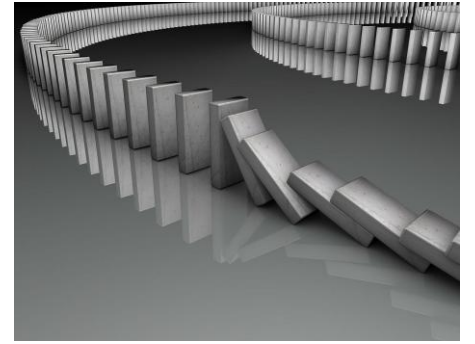
The Cook-Levin Theorem



- **Definition:** A k -CNF formula is an AND of ORs of at most k literals
- **Definition:** k -SAT = $\{\langle \phi \rangle : \phi \text{ is a satisfiable } k\text{-CNF formula}\}$

The Cook-Levin Theorem: 3-SAT is NP-complete

More NP-complete problems



- CLIQUE
- SUBSET-SUM
- KNAPSACK
- 3-COLORABLE (Exercise 26)

The P vs. NP problem

- We **conjecture** that $P \neq NP$: Solving and verifying are different
- A proof that $P = NP$ would **change the world**
 - *Assuming the proof gives us truly **practical** algorithms
- We could solve countless important problems in polynomial time 😊
- Hackers could break our encryption schemes in polynomial time 😞

Approximation algorithms

- We can find a “99% optimal” solution to a given instance of the **knapsack** problem in polynomial time 😊
- If $P \neq NP$, then we cannot even find a “1% optimal” solution to a given instance of the **clique** problem in polynomial time 😞

The complexity class $\text{NP} \cap \text{coNP}$

- $\text{FACTOR} = \{\langle K, R \rangle : K \text{ has a prime factor } p \leq R\}$

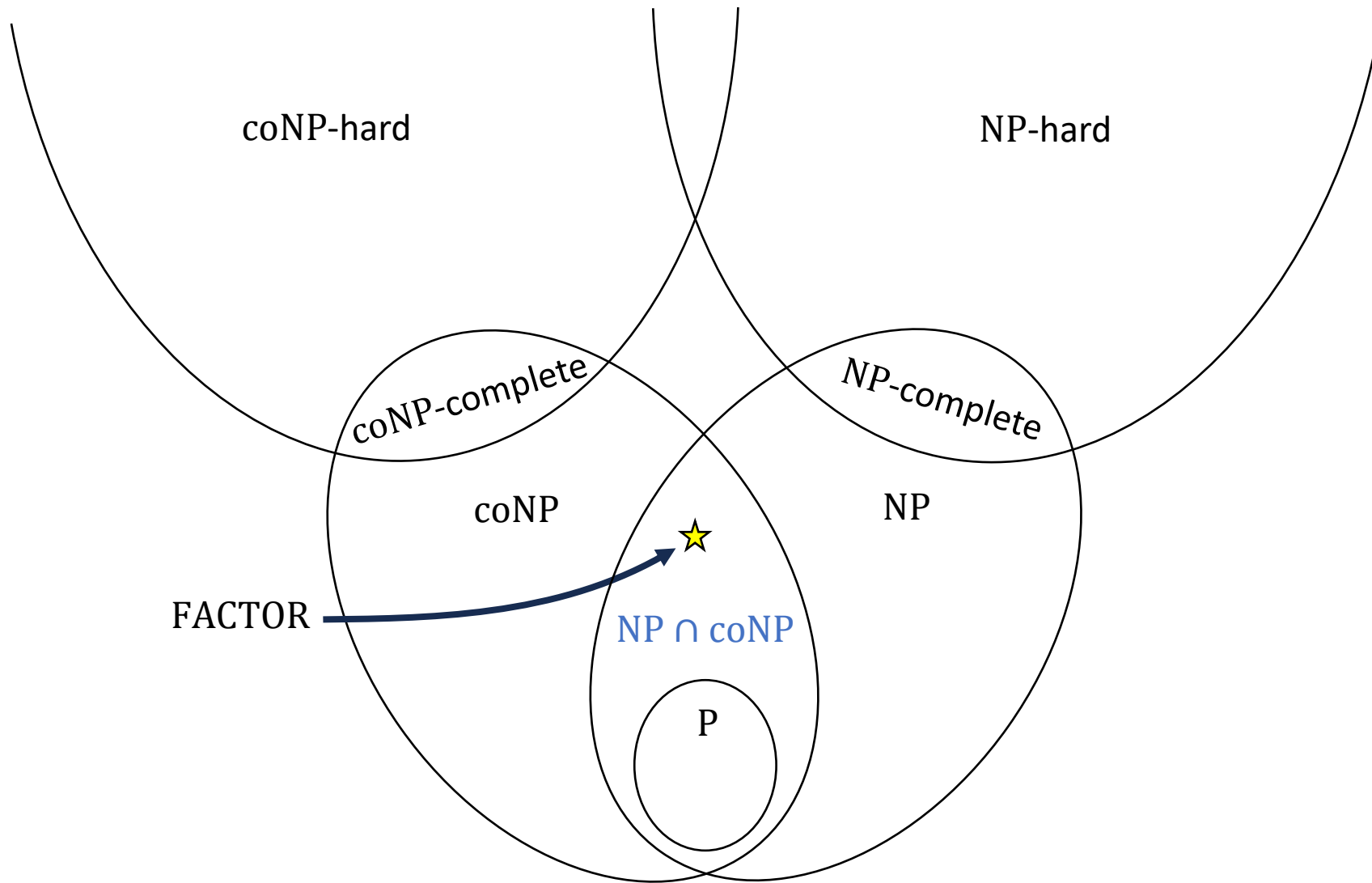
Theorem: $\text{FACTOR} \in \text{NP} \cap \text{coNP}$



I'll give you the prime factorization of K to prove that K **has** a small factor...
...and I'll give you the prime factorization of K' to prove that K' does **not** have a small factor!



- Consequence: FACTOR is probably **not** NP-complete



Thank you!

- Teaching you has been a privilege
- I hope you've enjoyed taking the course as much as I've enjoyed teaching it
- Please fill out the College Course Feedback Form using My.UChicago (deadline is December 14)

Three big lessons

1. To study **all possible algorithms**, make a mathematical **model** of computation
 2. Computation has severe, unavoidable **limitations**
 3. Computer science is **deep / profound / sublime**
 - What is the nature of the physical universe?
 - What is the nature of the human condition?
- **See you at office hours and the final exam!**

