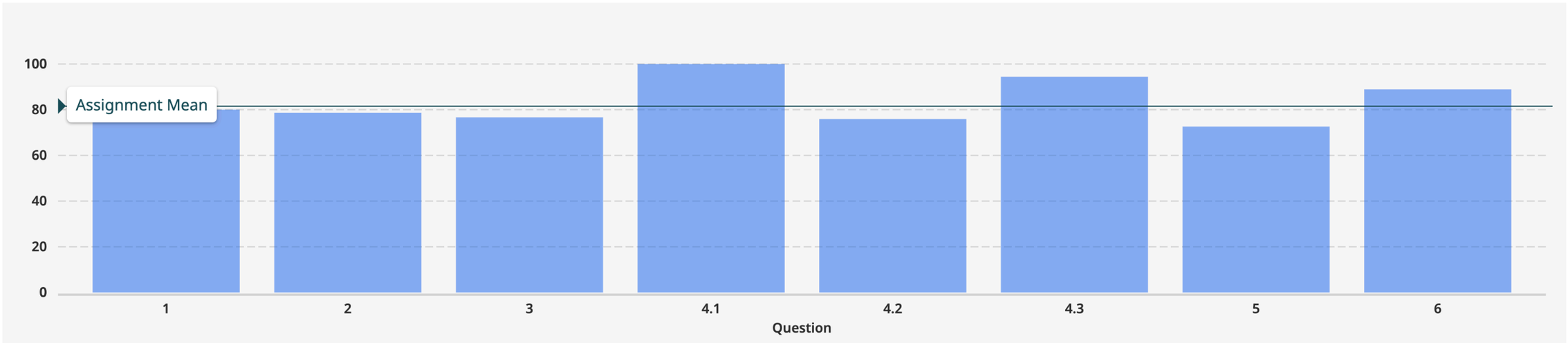
CSCI 361 Lecture 17:

Complexity Theory

Shikha Singh

Announcements & Logistics

- Hand in **Exercise # 13**, pick up **Exercise # 14**
- **Midterm 2 feedback** returned
    - Mean and median~ 81.4%
- Everyone did great on the undecidability & PL question
- Please go through feedback and let me know if you have questions



Next Week

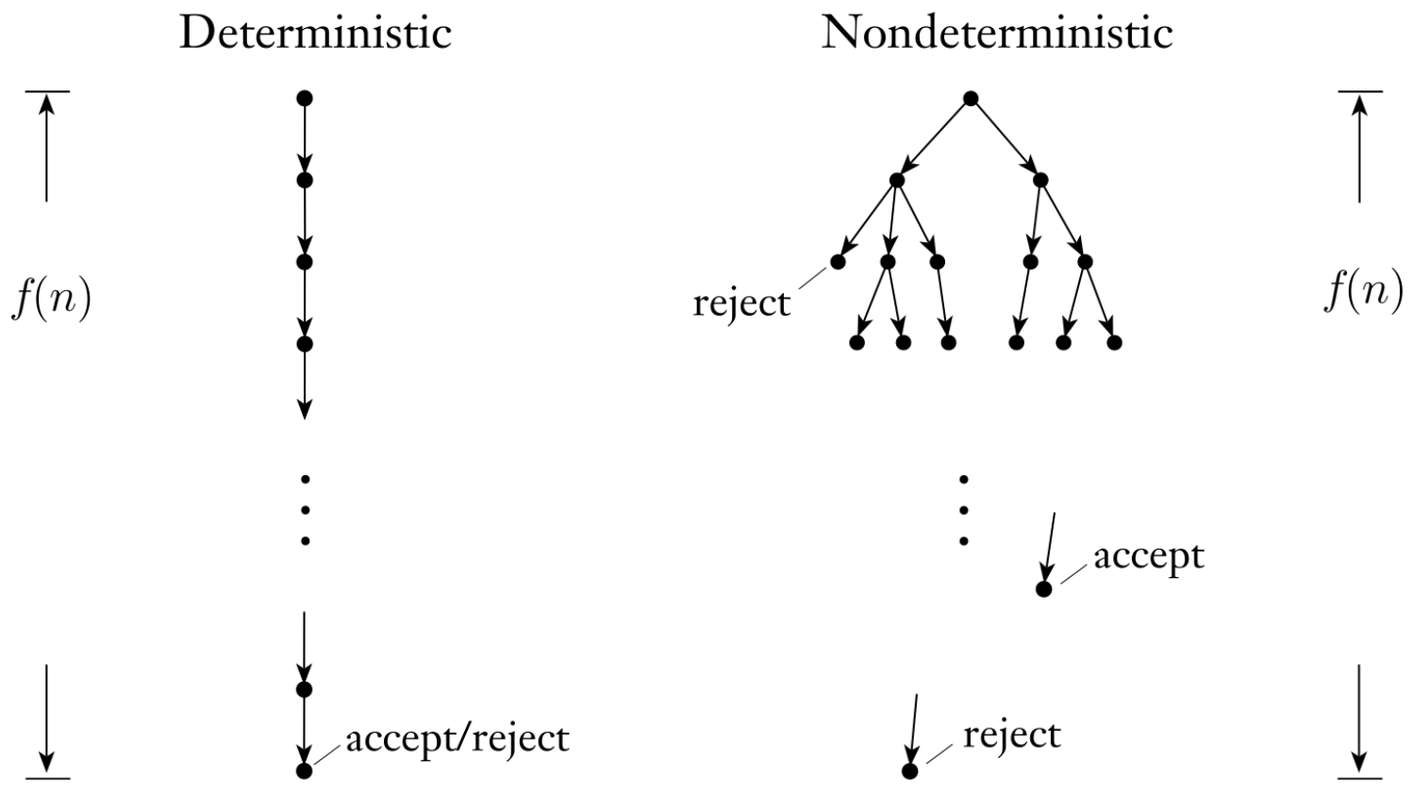
- I am traveling for a week long seminar on algorithms with predictions
- Sam McCauley will lecture on my behalf in CSCI 361
- Assignment 9 will be due on Thurs 10 pm
    - Can **work in pairs** (submit one PDF on Gradescope)
    - Opportunity to ask Sam questions last 10-15 mins of lecture
    - Will hold on-demand Zoom help hours: email me to set it up

Today

- Define the classes P and NP
    - Examples of problems in each
- Define NP hardness and NP completeness

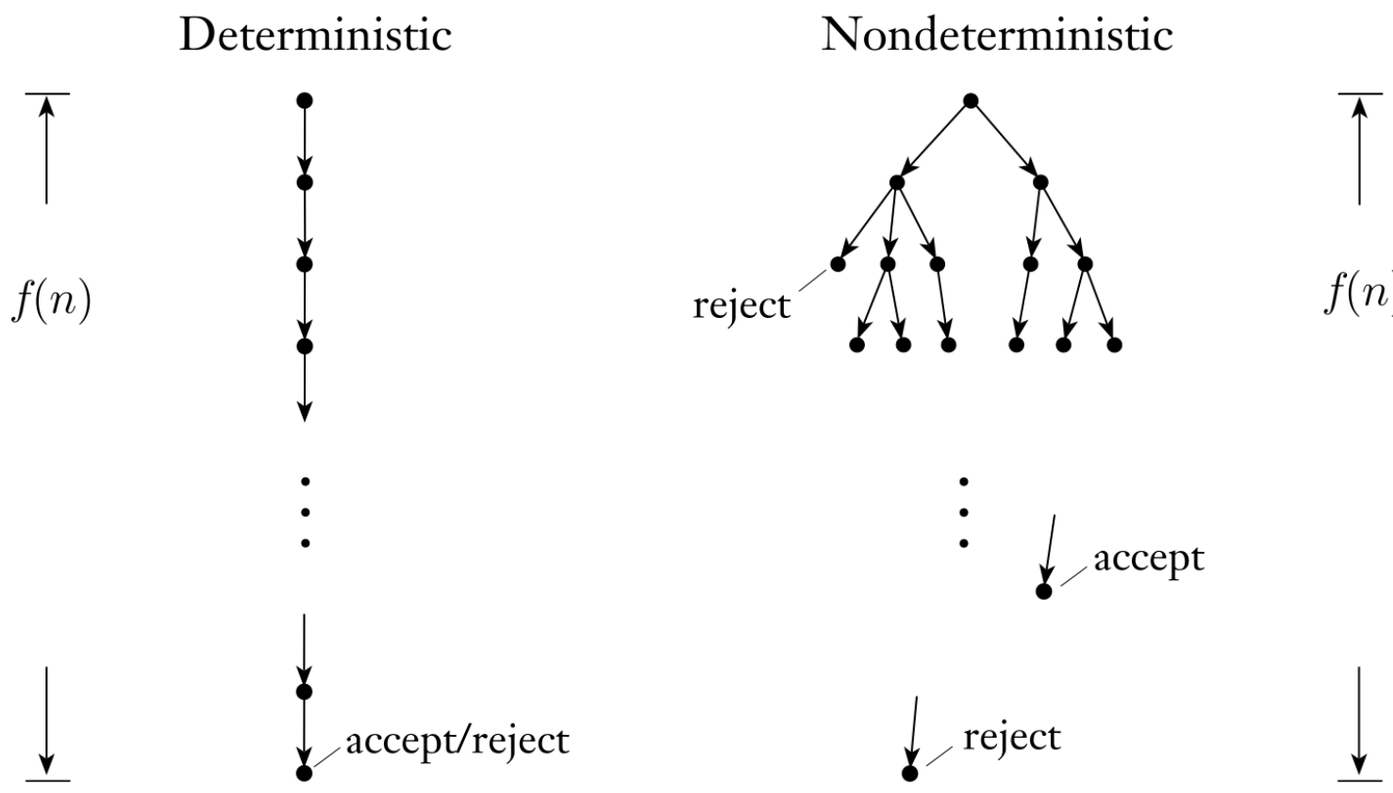
NTM Running Time

- **Definition.** Let M be a non-deterministic TM that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes on any branch of its computation on any input of length n.



Exponential Blow Up

- **Theorem.** Every $t(n)$-time non-deterministic TM has an equivalent $2^{O(t(n))}$-time deterministic TM, where $t(n) \geq n$.



- **Takeaway:** NTM is not polynomially-equivalent to a DTM.

Complexity Class P

Definition (Time). Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, TIME$(t(n))$, is

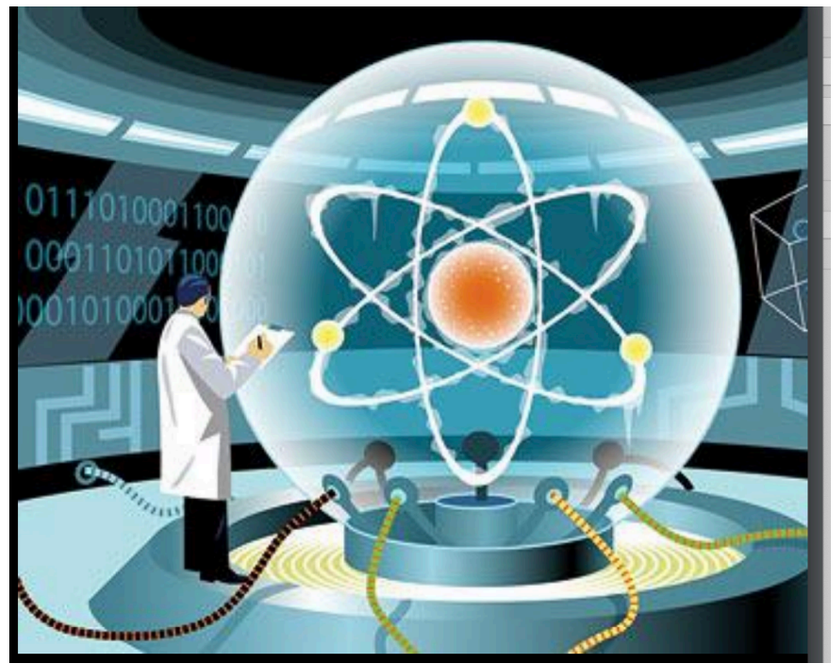
$$\text{TIME}(t(n)) = \{L \mid L \text{ is decided by a DTM in } O(t(n)) \text{ steps}\}$$

Definition (Class P). **P** is the class of languages that are decidable in polynomial time on a single-tape (deterministic) Turing machine. That is,

$$\text{P} = \cup_k \text{TIME}(n^k)$$

Extended Church Turing Thesis

Everyone's intuitive notion of
efficient algorithms
= polynomial-time algorithms



- Much more controversial:
  - Is $O(n^{10})$ efficient?
  - Randomized algorithms/ quantum algorithms can do much better

Extended Church Turing Thesis

Everyone's intuitive notion of
efficient algorithms
= polynomial-time algorithms

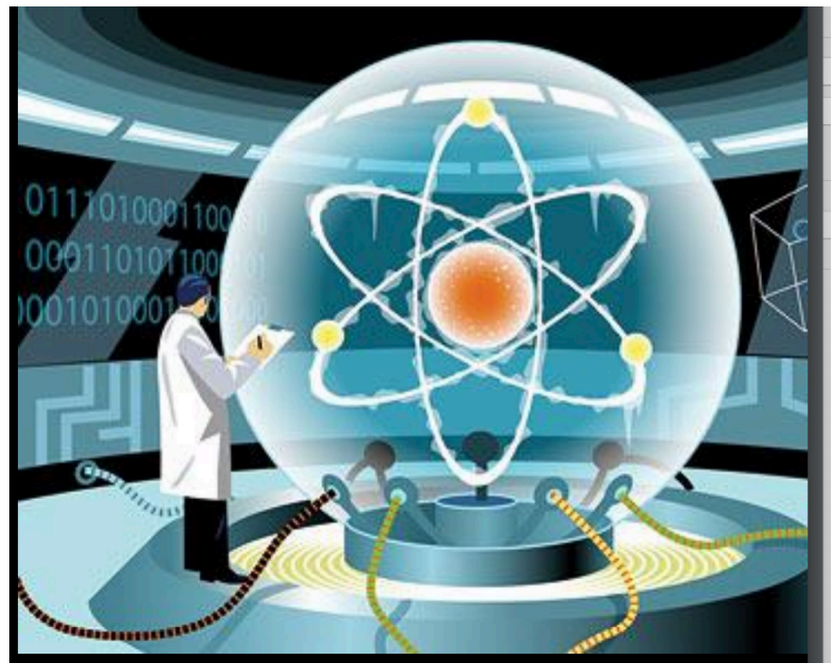


Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Problems in **P**

- Studied extensively in CSCI 256, but will use "language terminology"
- Examples in the book:
  - PATH $= \{\langle G, s, t\rangle \mid$ Given graph G and nodes s, t there is a path from $s \to t\}$
  - RELPRIME $= \{\langle x, y\rangle \mid x, y$ are relatively prime $\}$
  - $A_{CFG} = \{\langle G, w\rangle \mid G$ is a CFG and $w \in L(G)\}$
    - Parsing problem for CFGs
- Let's look at the last one: discuss a common parsing algorithm
- One-off example of a dynamic program

Chomsky Normal Form

- Algorithm described in book: CYK Parsing Algorithm (by John **C**ocke, Daniel **Y**ounger, and Tadao **K**asami)
- Assumes G is in CNF:
  - All rules are of the form $A \to BC,\ A \to b$
    - Additionally allow $S \to \varepsilon$
- Converting a grammar to CNF incurs constant-factor blow up in size

CYK Parsing Algorithm

- Let the input $w = w_1 \ldots w_n$. Goal: Does there exists a derivation $S \rightarrow \cdots \rightarrow w_n$ using the rules of G
- table[i, j] = variables of G that generate substring $w_i w_{i+1} \ldots w_j$
  - How do we find out if w is in $L(G)$?
  - Check if $S \in$ table $[1 , n]$
- Base case?
  - Handle $w = \varepsilon$ by checking if $s \rightarrow \varepsilon$
  - Fill out the diagonal: table $[i, i] = A$ if $A \rightarrow w_i$

CYK Parsing Algorithm

- Next step: all substrings of length 2
  - for $i = 1, \ldots, n-1$
    - For each rule $A \rightarrow BC$, if table$[i,\ i]$ contains B and $[i+1,\ i+1]$ contains C, then add A to $[i,\ i+1]$
- Substring of length 3 and so on,
  - Need a "split" point k such that if $w[i,k]$ is generated by B and $w[k+1,\ j]$ is generated by C and $A \rightarrow BC$, add A to table$[i,j]$

CYK Parsing Algorithm

$D =$ "On input $w = w_1 \cdots w_n$:

1. For $w = \varepsilon$, if $S \to \varepsilon$ is a rule, *accept*; else, *reject*. [[$w = \varepsilon$ case]]
2. For $i = 1$ to n: [[examine each substring of length 1]]
3. For each variable A:
4. Test whether $A \to \mathtt{b}$ is a rule, where $\mathtt{b} = w_i$.
5. If so, place A in $table(i, i)$.
6. For $l = 2$ to n: [[l is the length of the substring]]
7. For $i = 1$ to $n - l + 1$: [[i is the start position of the substring]]
8. Let $j = i + l - 1$. [[j is the end position of the substring]]
9. For $k = i$ to $j - 1$: [[k is the split position]]
10. For each rule $A \to BC$:
11. If $table(i, k)$ contains B and $table(k + 1, j)$ contains C, put A in $table(i, j)$.
12. If S is in $table(1, n)$, *accept*; else, *reject*."

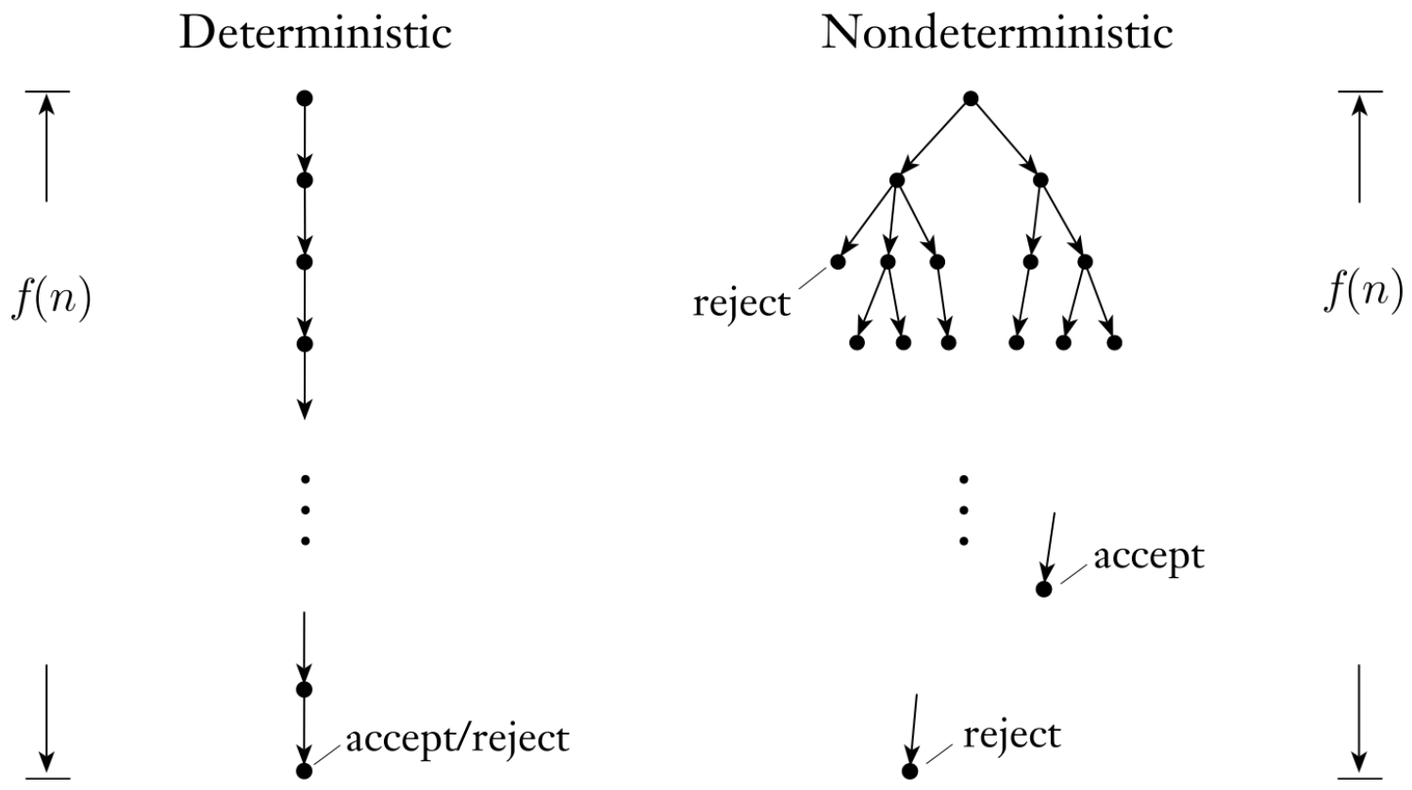
CYK Parsing is in **P**

- Running time of CYK parsing is $O(n^3)$
- Thus, verifying if a given CFG generates a given string is in **P**

Towards NP

- **Definition.** Let $t : \mathbb{N} \to \mathbb{N}$ be a function. The time complexity class, NTIME$(t(n))$, is

$$\text{NTIME}(t(n)) = \{L \mid L \text{ is decided by an NTM in } O(t(n)) \text{ steps}\}$$



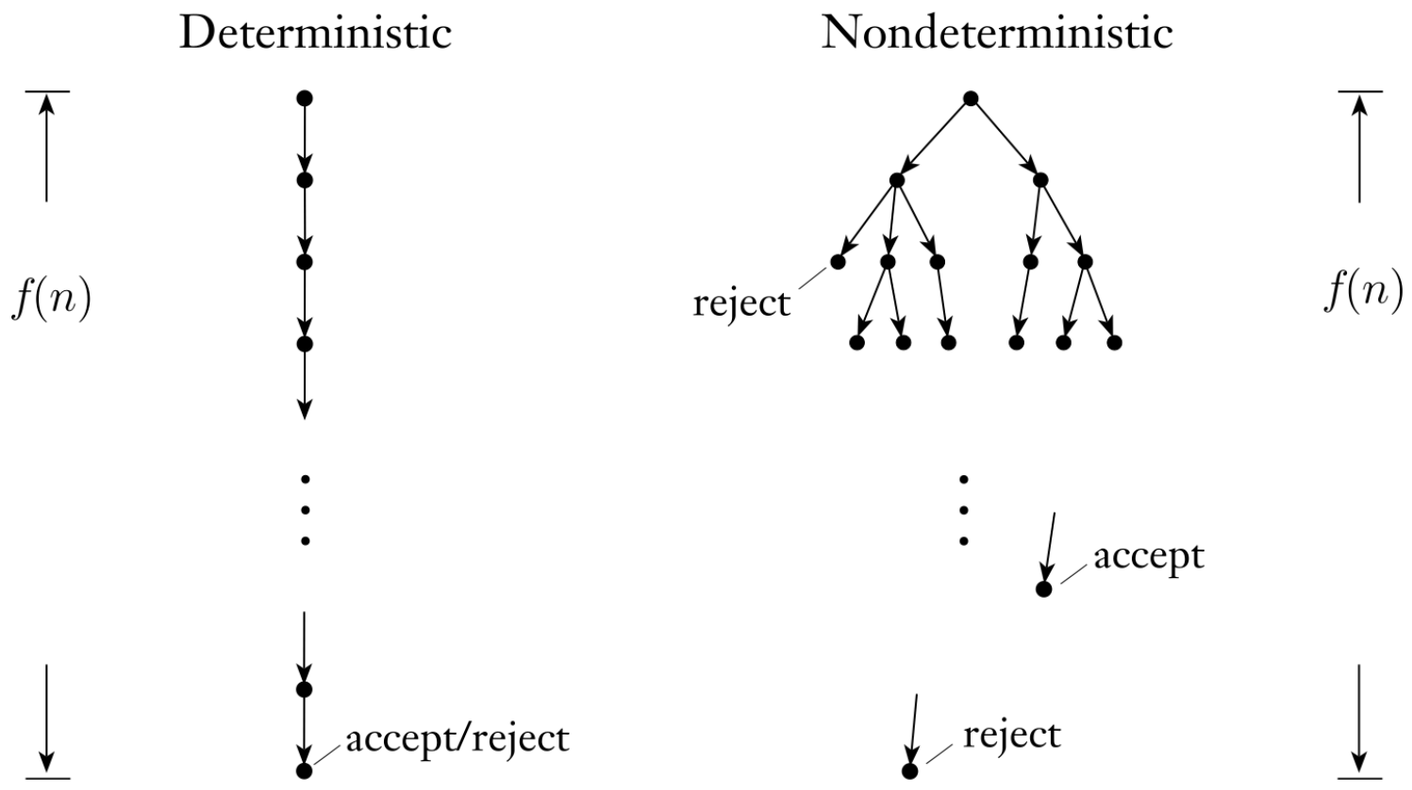
Complexity Class NP: Definition I

Definition. **NP** is the class of languages that are decidable in polynomial time on non-deterministic Turing machine. That is,

$$\text{NP} = \cup_k \text{NTIME}(n^k)$$

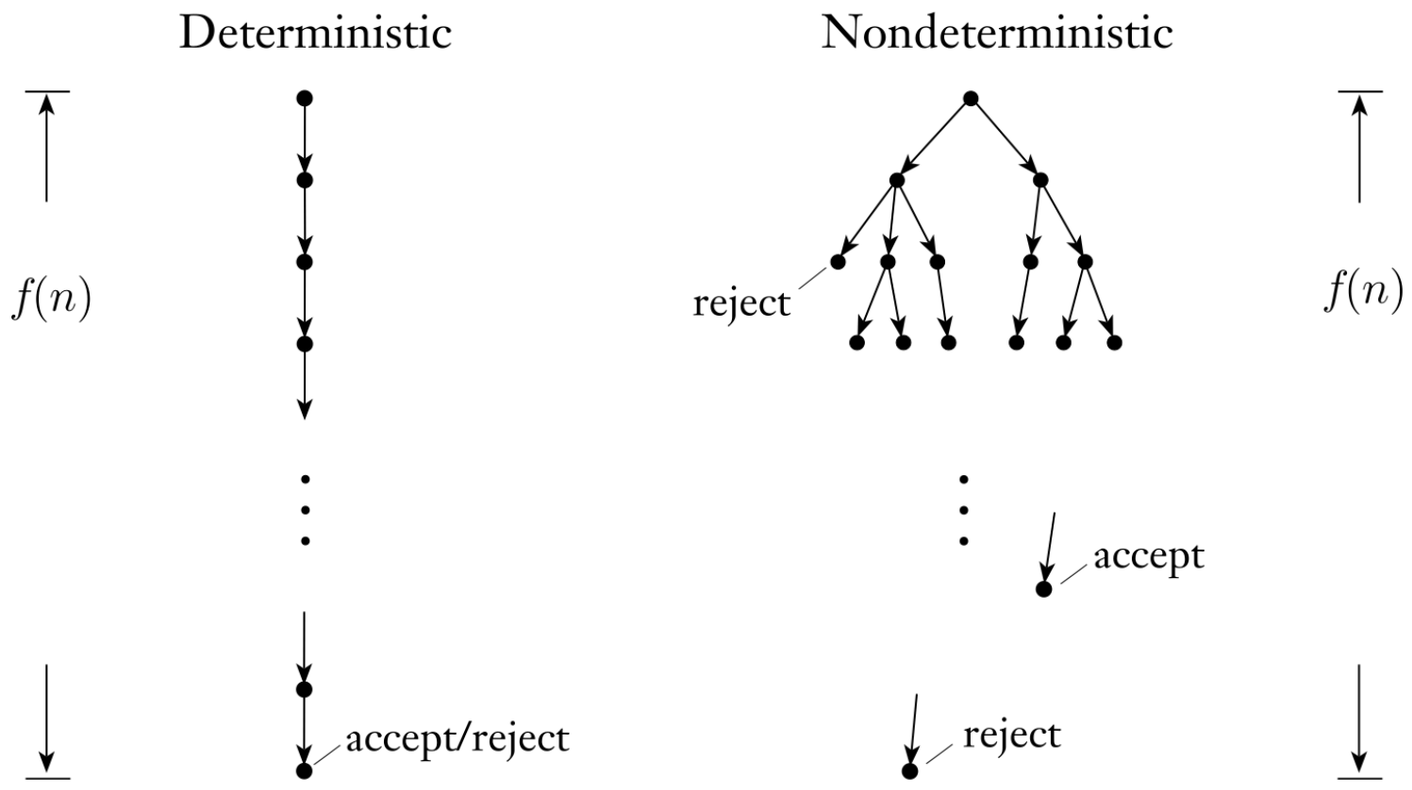
Subtlety in Definition of wrt NTM

- NTM guesses correspond to the transition function:
  $\delta : Q \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{L, R\})$
- Thus, at each step the "guess" is a choice among a constant c options



Subtlety in Definition of wrt NTM

- When thinking of algorithms run on NTM, we often guess the entire acceptance path (from root to leaf) of length $f(n)$ at once
- Then follow analyze the deterministic TM following such a path



Complexity Class NP: Definition 2

(Algorithms analog.) NP is the class of languages that have "polynomial-time verifiers"

Definition. A verifier for a language A is an algorithm V such that

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$$

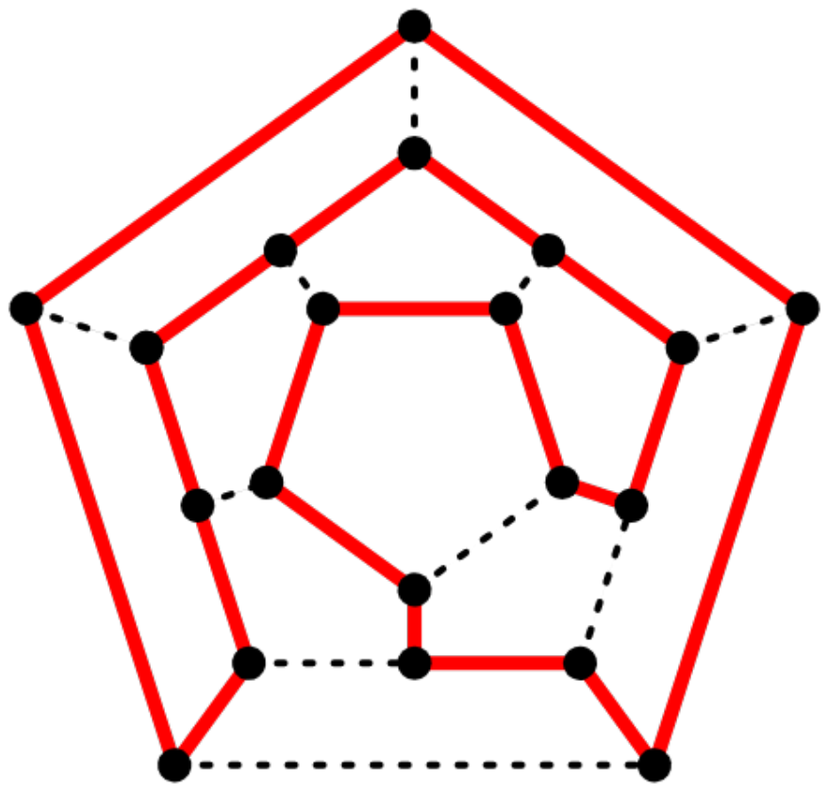
- For each $w \in A$, there exists a string c s.t. V accepts $\langle w, c \rangle$ iff $w \in A$
- A polynomial-time verifier V runs in polynomial time in $|w|$
- Here c is a **certificate**: polynomial-length string, $|c| = \text{poly}(|w|)$
- Eg.
  HAMPATH $= \{\langle G, s, t \rangle \mid G$ is a directed graph with a Hamiltonian path from s to $t\}$

Hamiltonian Path

- **Decision problem**: Given a directed graph G and two vertices $s, t \in V(G)$, does there exist a path from s to t that visits each vertex in G exactly once

- Stated in terms of a language:

  - HAMPATH $= \{\langle G, s, t\rangle \mid G$ is a directed graph wt a Hampath from s to $t\}$



HAMPATH in NP

- Non-deterministic Turing machine?
- Guess step: Guess a path from s to t in G
- Check step: Deterministically check if path is a valid Hampath

HAMPATH in NP: Verifier

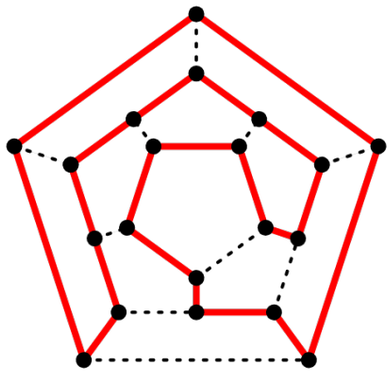
- For each "yes" instance $\langle G, s, t \rangle$, need a certificate string c s.t. a poly-time verifier V accepts $\langle w, c \rangle$ iff $w \in A$
- Following is a polynomial-time verifier:
  - On input $\langle \langle G, s, t \rangle, c \rangle$,
    - Check if c is a valid permutation of the nodes of G and starts with s and ends with t; reject if not
    - Check if each adjacent pair of nodes are an edge in G; reject if not; if all checks pass, c represents a valid Hamiltonian path from s to t in G and so accept

Equivalent Definitions

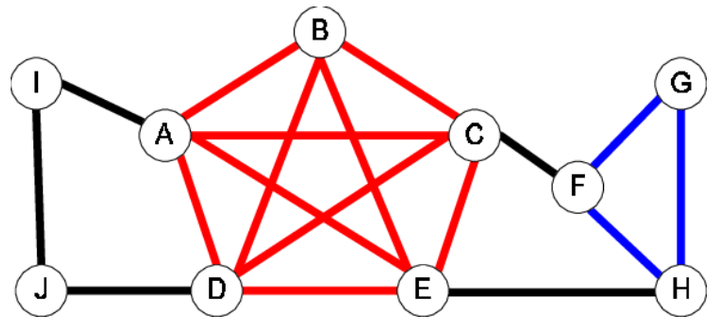
- **Theorem.** A language can be decided by a NTM in polynomial time if and only if it has a polynomial time verifier.
- Proof outline.
  - Suppose it can be decided by a NTM, what is the certificate that an input $w \in L$?
  - Suppose it has a polynomial-time verifier, what should a NTM "guess" to show $w \in L$

- **Takeaway.** NP is the "one-sided" analog of ***efficiently*** Turing decidable.

Many Examples of Problems in NP

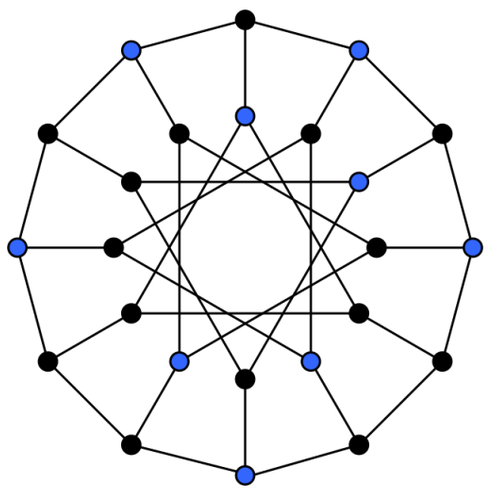


CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size $k\}$

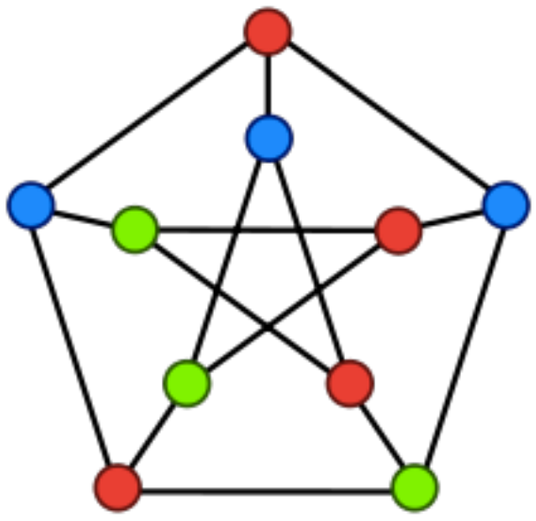


HAMPATH $= \{\langle G, s, t\rangle \mid \exists$ a Hamiltonian path from s to t in $G\}$

INDSET $= \{\langle G, k\rangle \mid G$ has an independent set of size $k\}$



SUBSET-SUM $= \{\langle S, t\rangle \mid S$ has a subset that sums to $t\}$



3COLOR $= \{\langle G\rangle \mid G$ has a valid 3-coloring$\}$

Hardest Problem in NP

- **SAT.** Consider a Boolean formulae ϕ with **variables** or their negations connected by AND ($\wedge$) and OR ($\vee$) and NOT. Here a **literal** is a variable or its negation. The language is defined as follows:

$$\text{SAT} = \{\phi \mid \phi \text{ has a satisfying assignment}\}$$

- E.g. $\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

  $\phi \in$ SAT because it has a satisfying assignment

  $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$

- **Question.** Is SAT in NP?

Hardest Problem in NP

- **SAT.** Consider a Boolean formulae ϕ with **variables** or their negations connected by AND ($\wedge$) and OR ($\vee$) and NOT. Here a **literal** is a variable or its negation. The language is defined as follows:

$$\text{SAT} = \{\phi \mid \phi \text{ has a satisfying assignment}\}$$

- E.g. $\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

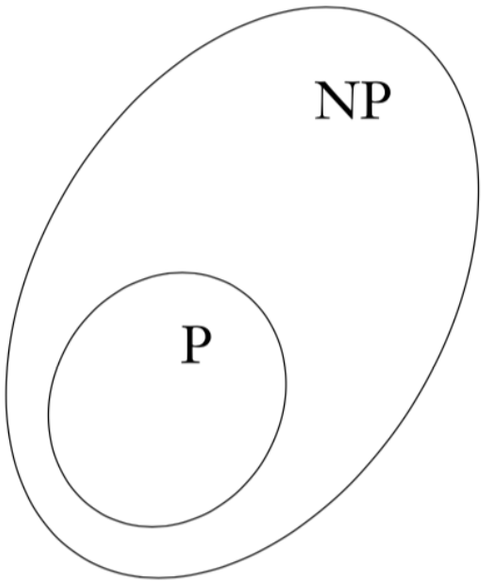
  $\phi \in$ SAT because it has a satisfying assignment

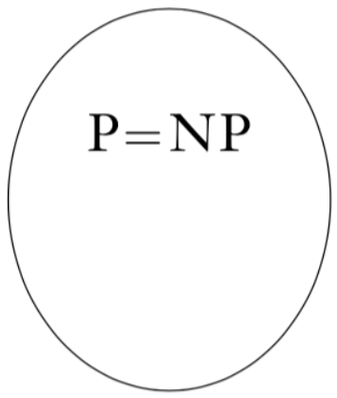
  $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$

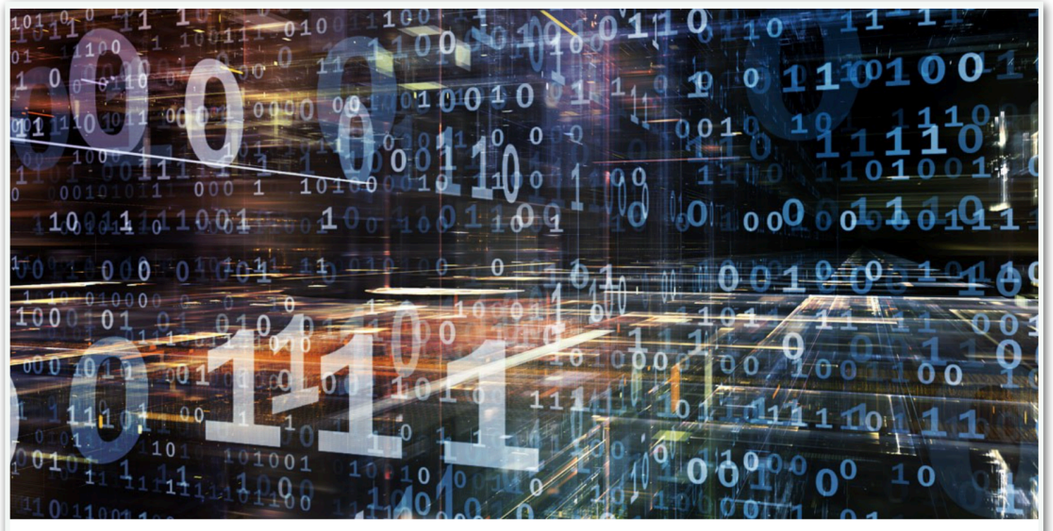
- **Theorem. (Cook Levin Theorem)** SAT $\in$ P $\iff$ P = NP

What We Don't Know: P = NP?

- Every problem in **P** is also in **NP** but what about the reverse?
- A problem that can be verified quickly, can it also be decided quickly?
- Is **P = NP** ?
  - Million Dollar Question







P vs NP and the $1M Millennium Prize Problems

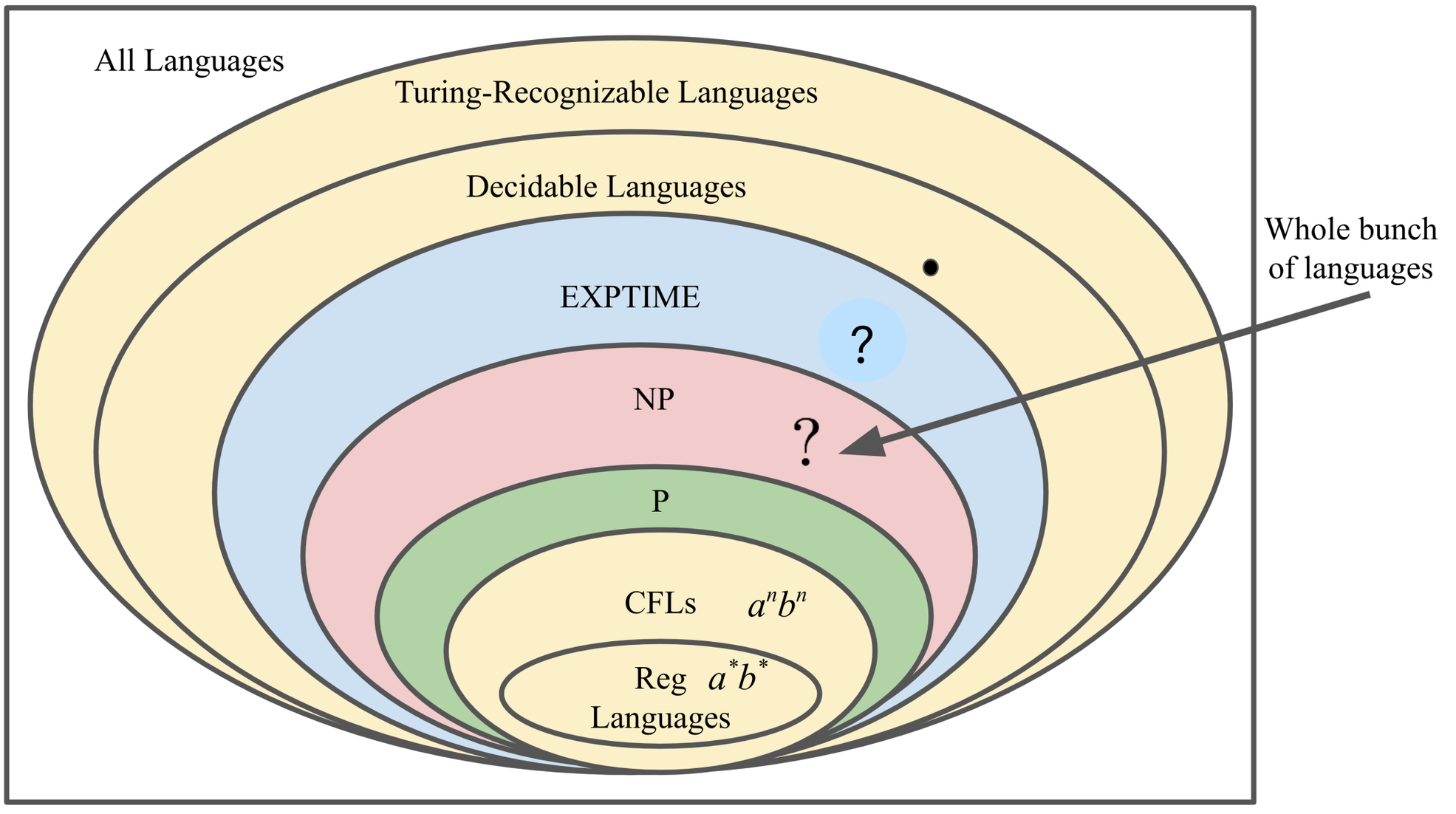
What's the most difficult way to earn $1M US Dollars?

What We Know

- P: class of languages whose membership can be ***decided*** quickly
- NP: class of languages whose membership can be ***verified*** quickly
- **Theorem.** $P \subseteq NP$
- **Theorem.** $NP \subseteq \bigcup_k \text{TIME}(2^{n^k}) = \text{EXPTIME}$
- We don't know whether either of these containment are **strict**

Classifying Problems

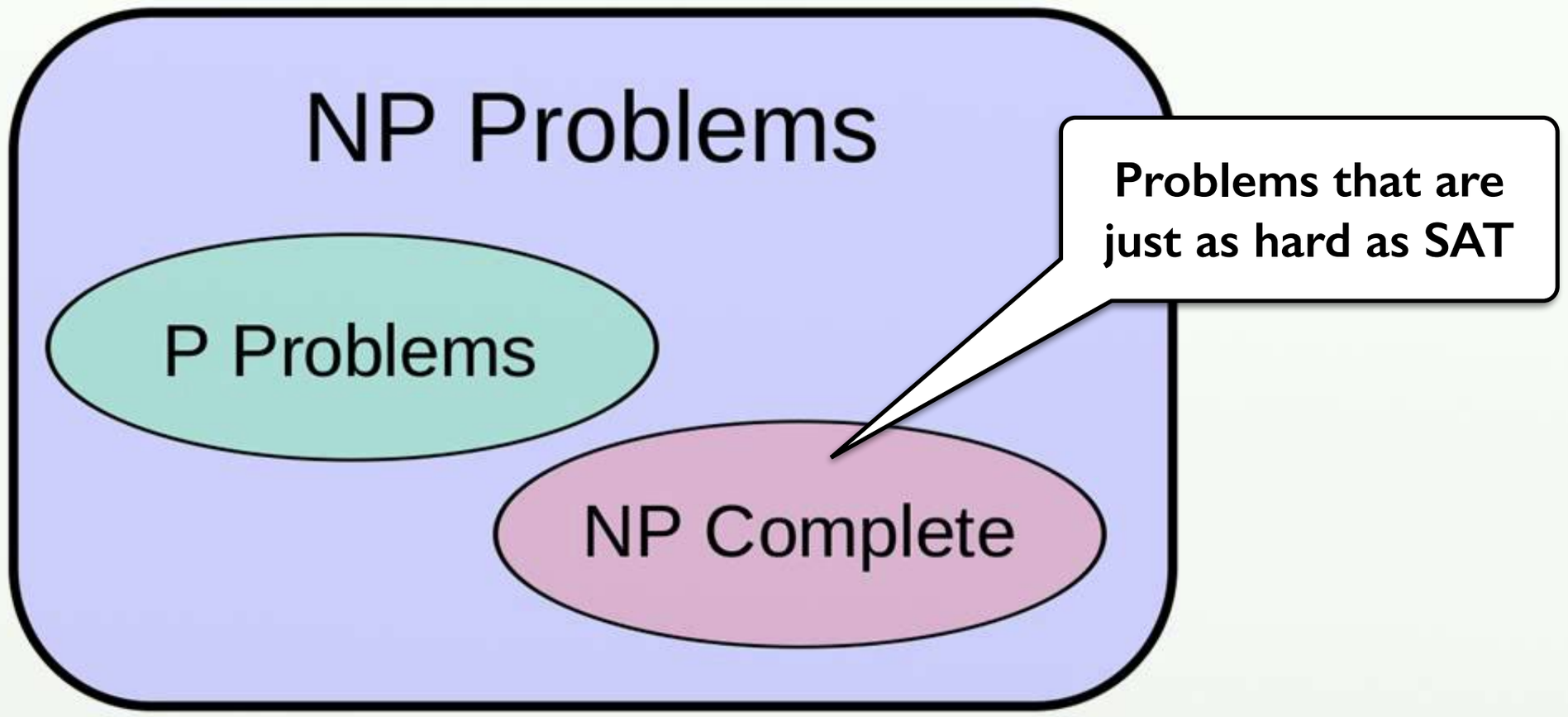
- So far were definitively able to classify problems based on difficulty
- Unable to prove a lot of problems are impossible to solve efficiently



Classifying Problems

- Instead, we say problem X is likely very hard to solve by saying, if a polynomial-time algorithm was found for X, then **P = NP** (*which we believe is not possible*)
- **Theorem. (Cook Levin Theorem)** SAT $\in$ P $\iff$ P = NP

P versus NP problem

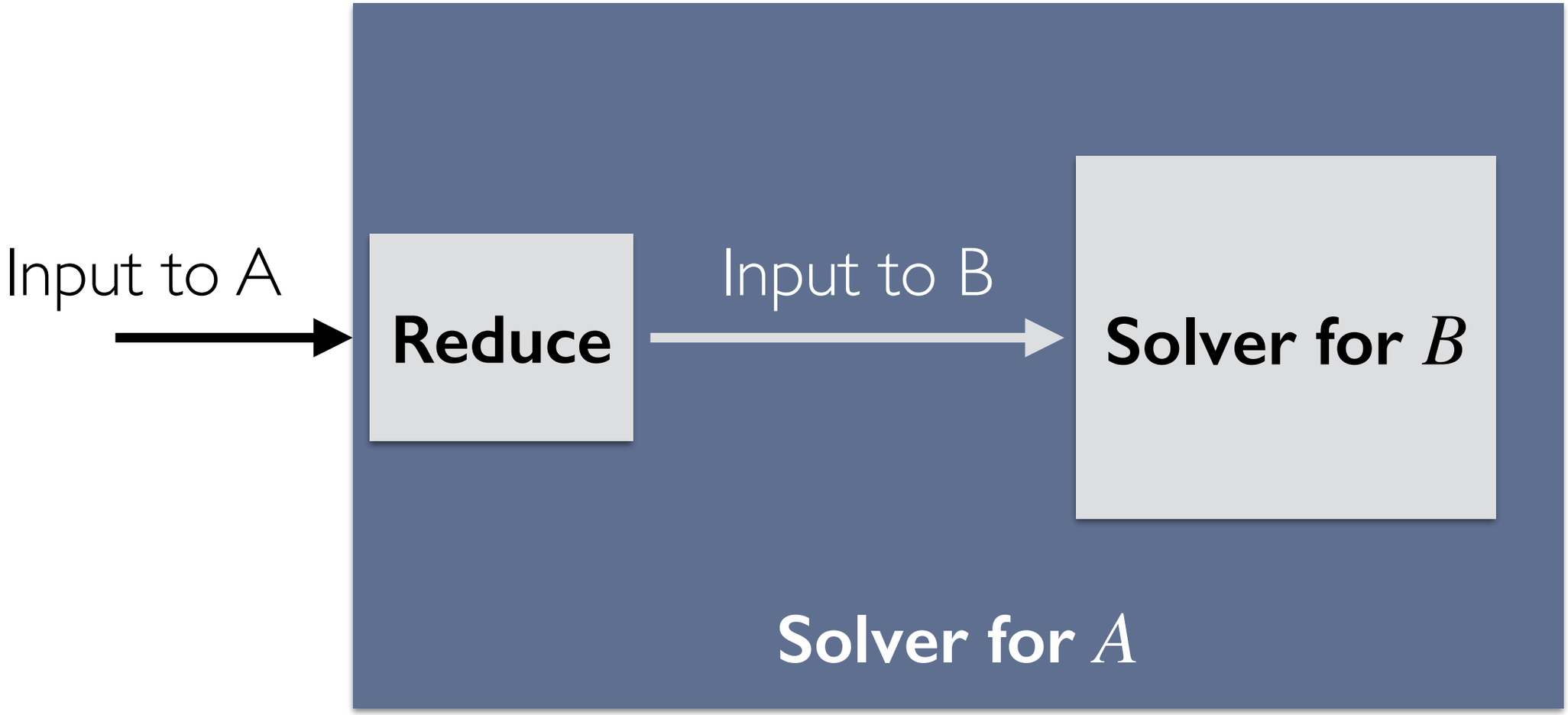


Recent CACM article by Lance Fortnow: https://www.cs.cmu.edu/~15326-f24/CACM-Fortnow.pdf

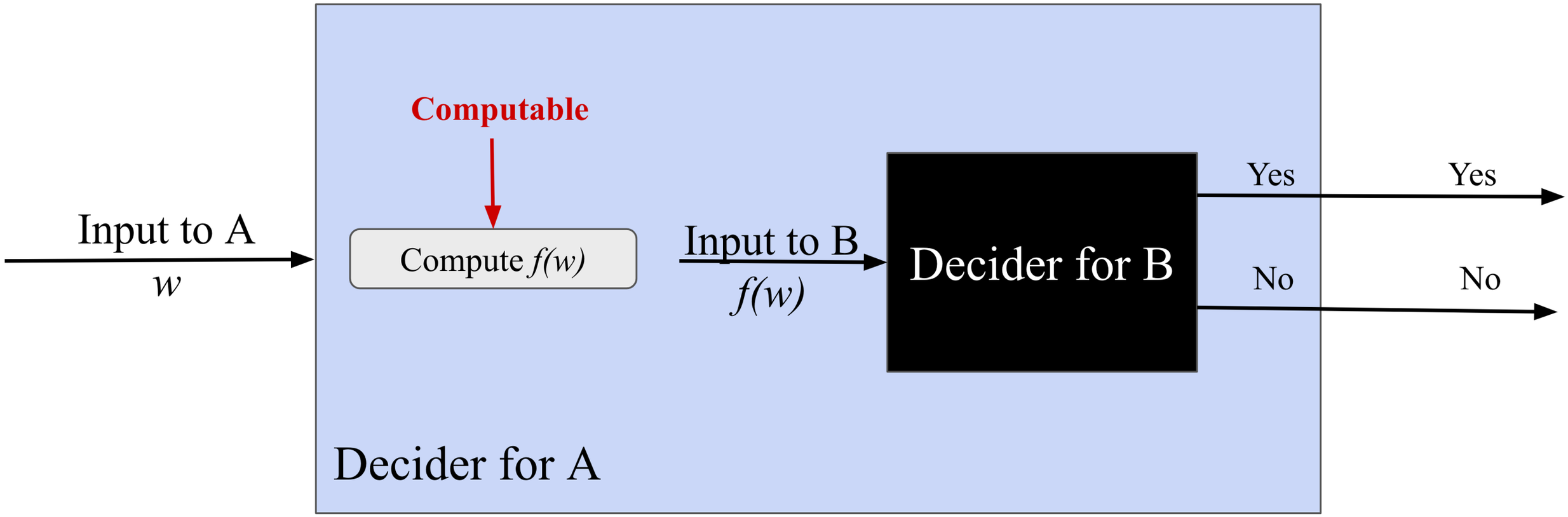
"*None of us truly understand the P versus NP problem, we have only begun to peel the layers around this increasingly complex question. Perhaps we will see a resolution of the P versus NP problem in the near future but I almost hope not.The P versus NP problem continues to inspire and boggle the mind and continued exploration of this problem will lead us to yet even new complexities in that truly mysterious process we call computation.*" - *Fortnow 2022*

Reductions

- The notion of reductions has been fundamental to classifying the relative difficulty of problems
- If a problem A reduces to problem B, then solving A is no harder than solving B

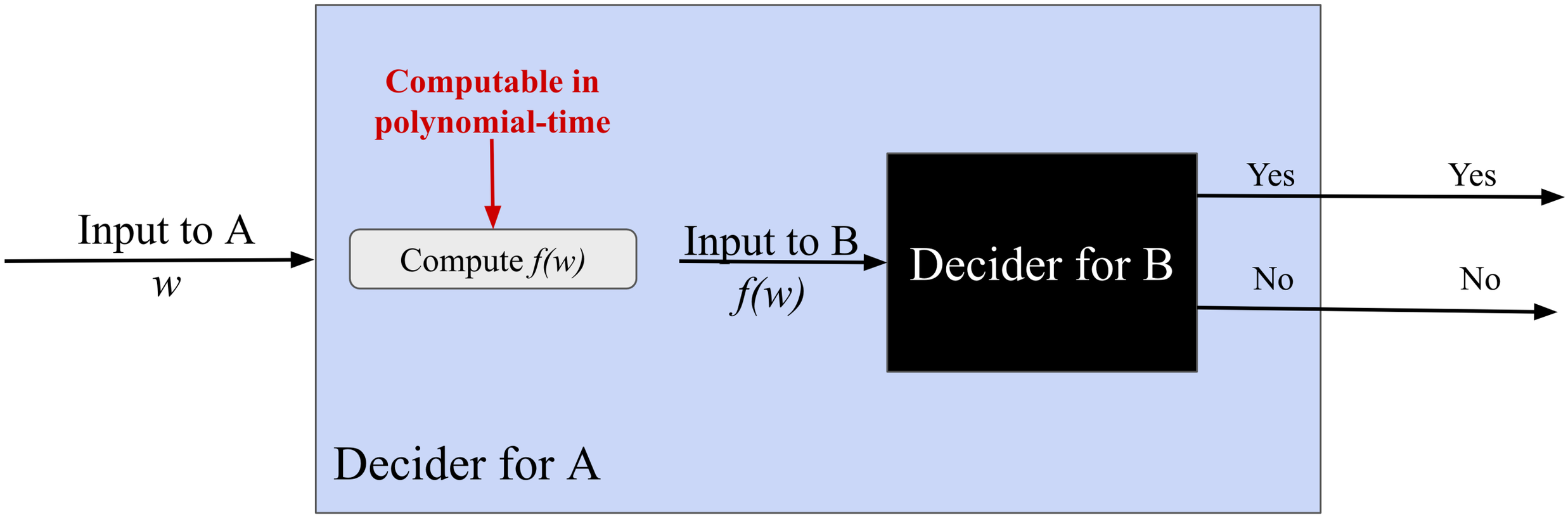


So Far: Mapping Reductions



$$A \leq_m B$$

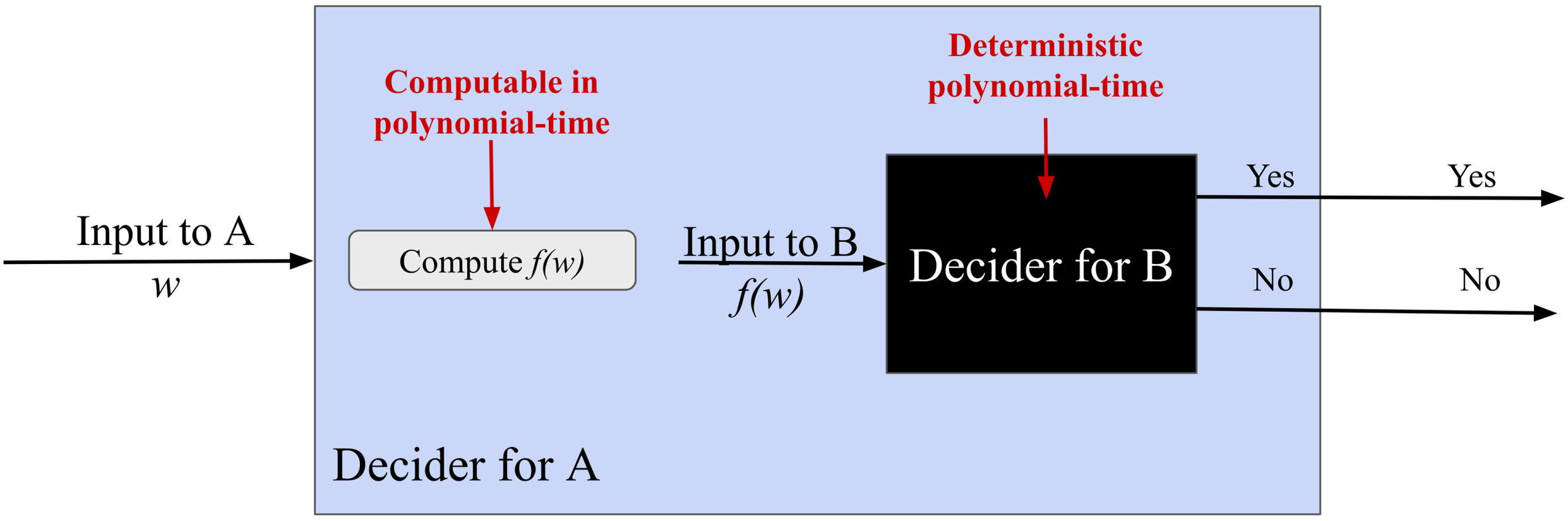
Now: Polynomial-Time Reductions



$$A \leq_p B$$

Using Reductions: Positive Use

- Using reductions to prove **membership** in P:
  - **Theorem.** If $A \leq_P B$ and $B \in \mathsf{P}$, then $A \in \mathsf{P}$.



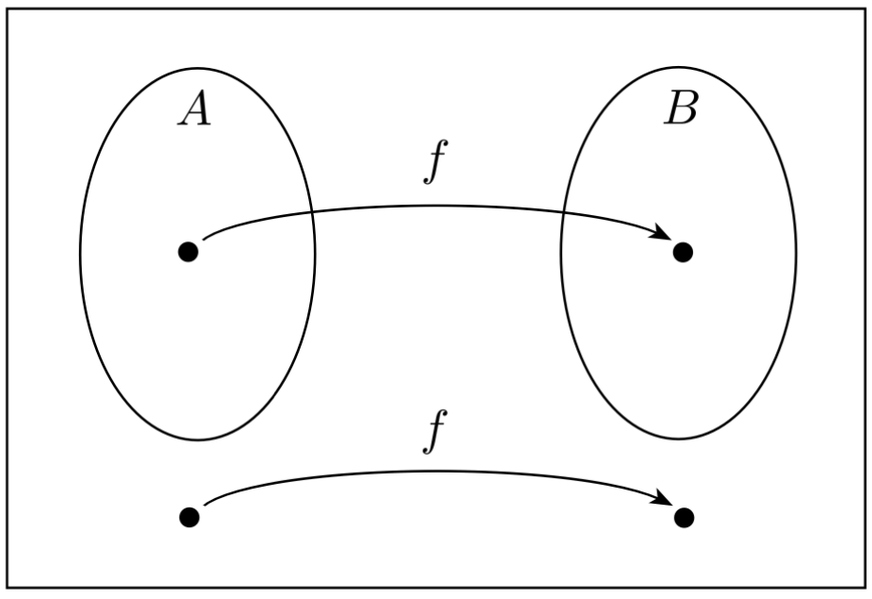
$$A \leq_p B$$

Polynomial-time Reductions

- Similar to mapping reductions but now we care about efficiency
- **Definition.** Language A is polynomial-time reducible to language B, denoted $A \leq_P B$, if there exists a ***polynomial-time computable*** function $f : \Sigma^* \to \Sigma^*$, such that

$$w \in A \iff f(w) \in B \quad \text{for every } w$$

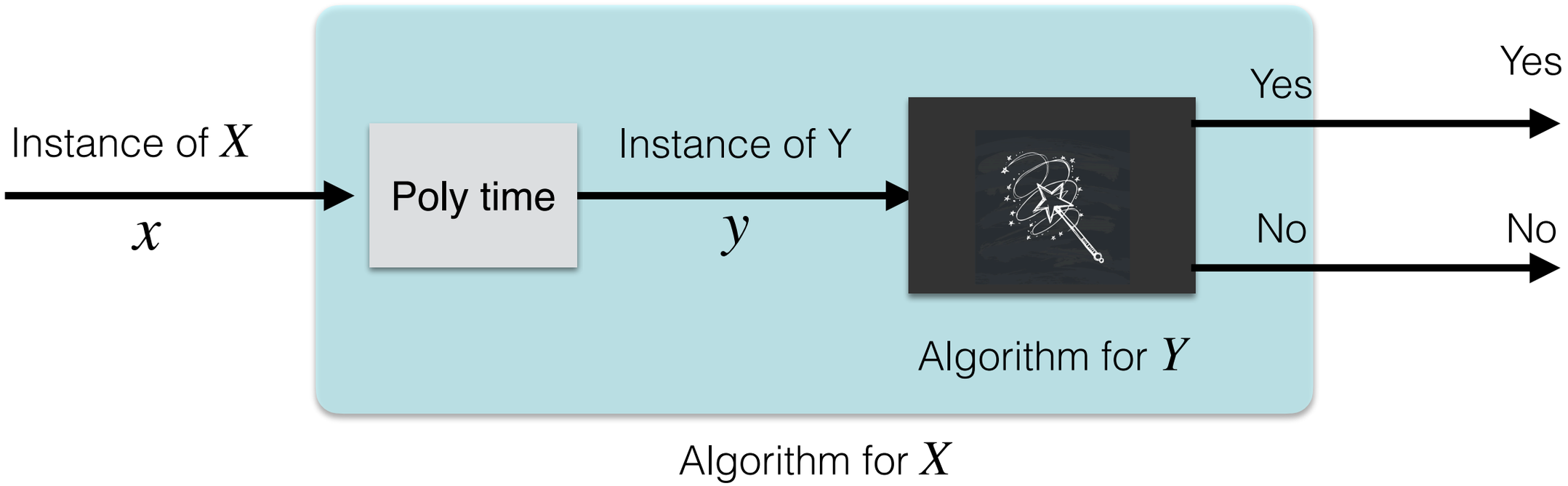
- The function f is the **polynomial-time reduction** from A to B



Polynomial-Time Reductions

Notation. $X \leq_p Y$

- Solving X is no harder than solving Y: if we have an algorithm for Y, we can use it + poly time reduction to solve X



Review of Reductions from CS256

Class Exercises:

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent: for any $x, y \in S$, $(x, y) \notin E$
    - **IND-SET Problem.** Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?
- Given a graph $G = (V, E)$, a vertex cover is a subset of vertices $T \subseteq V$ such that for every edge $e = (u, v) \in E$, either $u \in T$ or $v \in T$.
    - **VERTEX-COVER Problem.** Given a graph $G = (V, E)$ and an integer k, does G have a vertex cover of size at most k?
- A **clique** in an undirected graph is a subset of nodes such that every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.
    - **CLIQUE.** Given a graph G and a number k, does G contain a k-clique?

Questions. Show that **IND-SET** $\leq_p$ **CLIQUE** and **Vertex-Cover** $\leq_p$ **IND-SET**

Class Exercises:

Questions. Show that **IND-SET** $\leq_p$ **CLIQUE** and **Vertex-Cover** $\leq_p$ **IND-SET**