

CSIE 2136 Algorithm Design and Analysis, Fall 2022



# NP Completeness - I

---

Hsu-Chun Hsiao

# Annoucement

- HW4
  - Due Time (hand-written): 2022/12/20 14:20
  - Due Time (programming): 2022/12/30 23:59
  - Don't get spooked by the length; many of them are installation guides and tool/library manuals
- Final exam: 2022/12/22
  - Location TBA
  - Closed book
  - Scope: all topics taught in ADA, focusing on those after the midterm

# To solve, or not to solve?

Design an algorithm to solve the computational problem efficiently

Prove that the problem is too hard to solve (no easier than solving NP-complete problems!)



# We want to avoid...



**“I can’t find an efficient algorithm, I guess I’m just too dumb.”**



**“I can’t find an efficient algorithm, because no such algorithm is possible!”**

# A better way



**“I can’t find an efficient algorithm, but neither can all these famous people.”**

Origin: Michael Garey and David S. Johnson “Computers and Intractability: A Guide to the Theory of NP-Completeness” (1979)  
Cartoon: <https://www.ac.tuwien.ac.at/people/szeider/cartoon/>

# 2-week Agenda

- NP-Completeness Overview

- Warm up: graph coloring
- Complexity classes
- Decision problems
- Reduction
- Appendix: P-time solving vs. verification

- Proving NP-Completeness

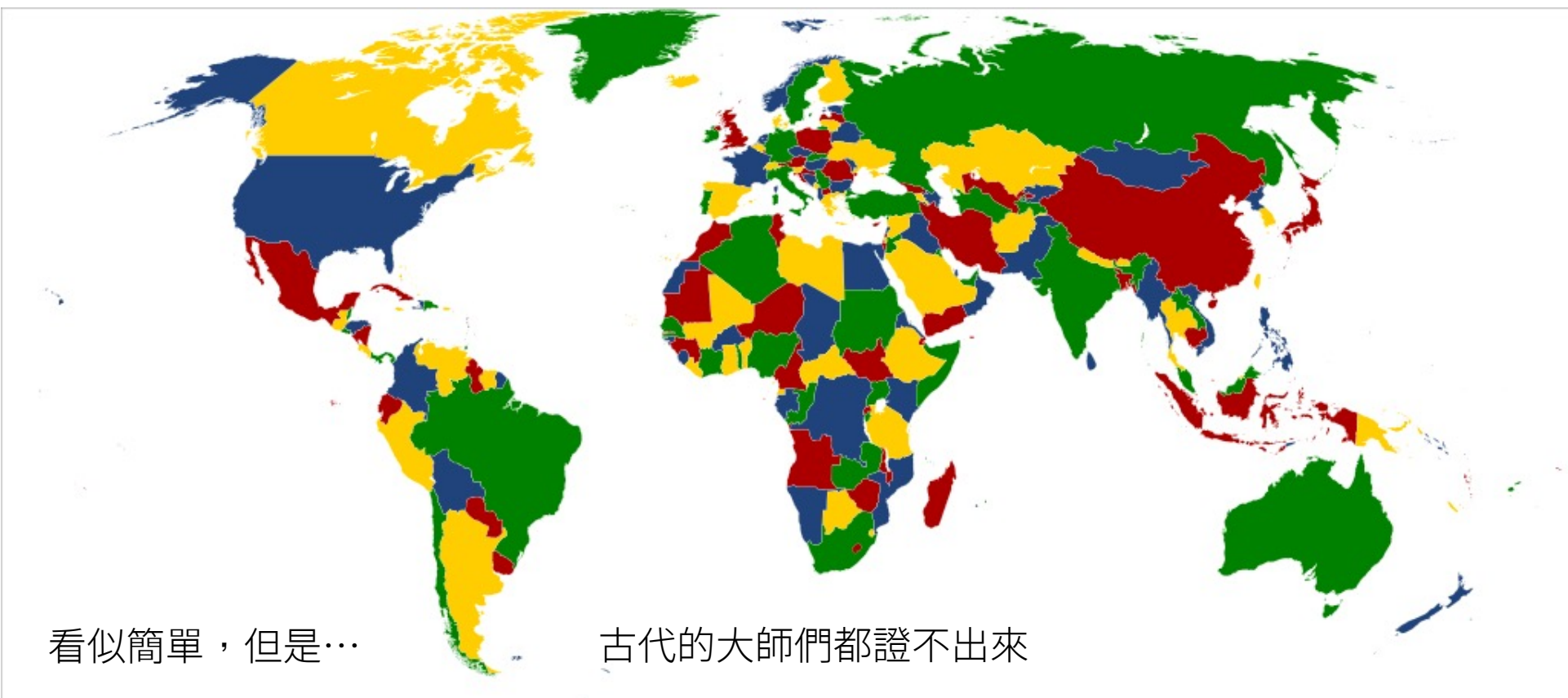
- Formula satisfiability
- 3-CNF satisfiability
- Clique
- Vertex cover
- Independent set
- Traveling salesman
- Hamiltonian cycle
- ...

Next Topic: Handling NP-Completeness

# Warm up: Graph Coloring

# 四色問題 (Four Color Problem)

- 只能使用四種顏色，使地圖上每兩個鄰接區域的顏色都不一樣



看似簡單，但是…

古代的大師們都證不出來



# 一百多年後的四色定理

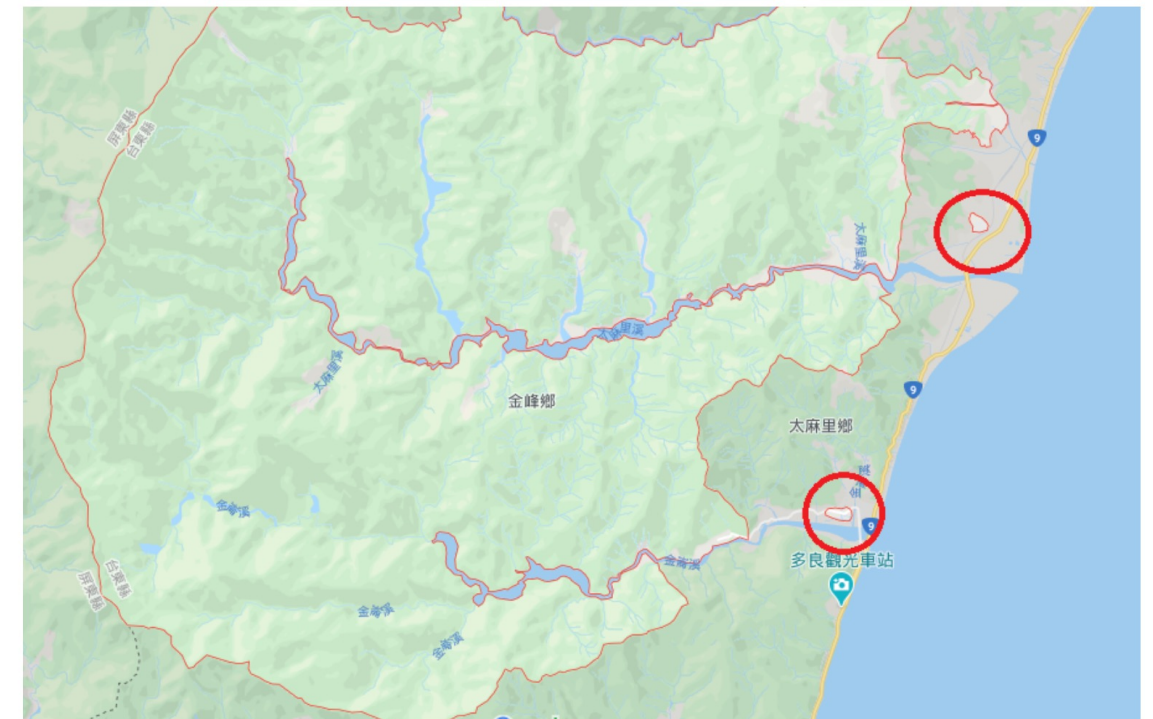


- Finally proven (with the help of computers) by Kenneth Appel and Wolfgang Haken in 1976
  - Their algorithm runs in  $O(n^2)$  time
- First major theorem proved by a computer
- Open problems remain...
  - Linear time algorithms to find a solution
  - Concise, human-checkable, mathematical proofs



# 沒那麼簡單之二：飛地

就像台東縣金峰鄉的這兩個村里，離金峰鄉的位置有段距離，並且自成一格的出現在別的鄉鎮：



[storycircle571.com/2020/02/10/例外還意外？那些台灣境內的飛地/](http://storycircle571.com/2020/02/10/例外還意外？那些台灣境內的飛地/)

Q: Can you color this map using only 2 colors?  
Using only 3 colors? Using only 4 colors?

2 colors: No

3 colors: Yes

4 colors: Yes

Q: Given a map, can you efficiently determine whether  $k$  colors is enough?

$k = 1$ : trivial

$k = 2$ : trivial

$k = 3$ : the planar 3-colorability problem is NP-complete.

$k \geq 4$ : always possible because of the four-color theorem



## K 色問題 (k-colorability)

Given a graph  $G$ , can we color the vertices with  $k$  colors such that no adjacent vertices have the same color?

Q: Can we efficiently solve the k-colorability problem when  $k = 1$ ?  $k = 2$ ?  
 $k \geq 3$ ?

$k = 1$ : trivial

$k = 2$ : trivial

$k \geq 3$ : NP complete

# Complexity Classes

# What is a hard problem

- A problem is **hard** if it has no known efficient algorithms to solve it
  - Efficient = tractable = **polynomial running time** (in the size of input)
  - K-colorability, Hamiltonian, knapsack, ...
- How to know whether a problem is hard?
  - Idea: 已知問題 A 很難。若能證明問題 B 至少跟 A 一樣難，那麼問題 B 也很難。
  - Decide which **complexity classes** the problem belongs to via **polynomial-time reduction**

# Complexity classes

- A complexity class is defined as a set of problems of related resource-based complexity
  - Resource = time, memory, communication, ...
- This lecture focuses on decision problems and time resource



# Complexity classes

- **Class P**: class of problems that can be solved in  $O(n^k)$ 
  - $O(n^k)$  means polynomial in the input size,  $n$  ( $k$  is a constant)
  - Problems in P are considered tractable
- **Class NP**: class of problems that can be verified in  $O(n^k)$ 
  - NP = Non-deterministic Polynomial

# Complexity classes

- **Class P**: class of problems that can be **solved** in  $O(n^k)$ 
  - $O(n^k)$  means polynomial in the input size,  $n$  ( $k$  is a constant)
  - Problems in P are considered **tractable**
- **Class NP**: class of problems that can be **verified** in  $O(n^k)$ 
  - NP = **Non-deterministic** Polynomial

Q: Is  $P \subseteq NP$ ?

$P \subseteq NP$ : Yes, because a problem solvable in polynomial time is verifiable in polynomial time as well.

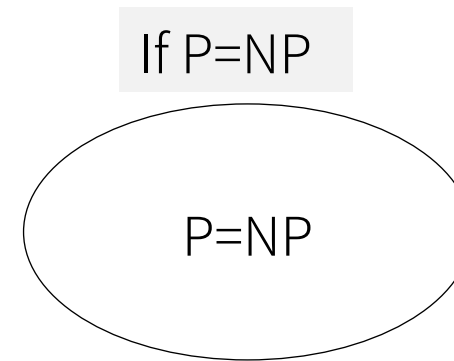
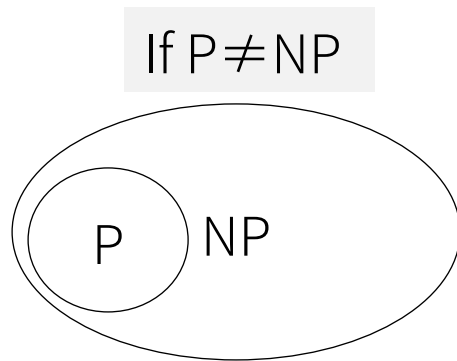
# Complexity classes

Q: Is  $NP \subseteq P$ ?

$NP \subseteq P$ : We don't know. If so,  $P = NP$ .

Whether  $P = NP$  is a question worth \$1,000,000 (Millennium Prize).

Widespread belief in  $P \neq NP$ .



DOI:10.1145/3460351

**Advances in algorithms, machine learning, and hardware can help tackle many NP-hard problems once thought impossible.**

BY LANCE FORTNOW

# Fifty Years of P vs. NP and the Possibility of the Impossible

ON MAY 4, 1971, computer scientist/mathematician Steve Cook introduced the P vs. NP problem to the world in his paper, “The Complexity of Theorem Proving Procedures.” More than 50 years later, the world is still trying to solve it. In fact, I addressed the subject 12 years ago in a *Communications* article

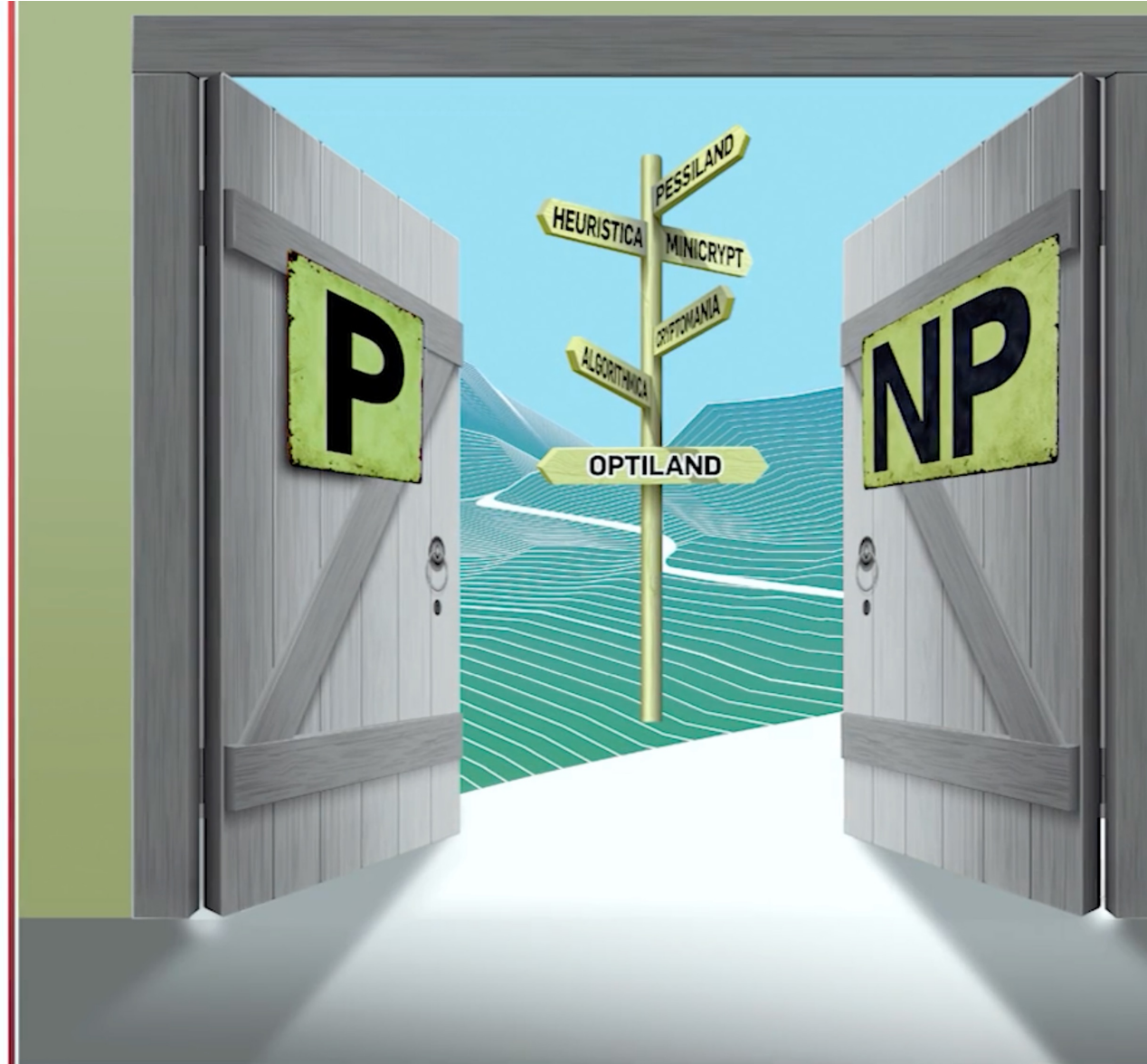
most importantly, the rise of data science and machine learning. In 2009, the top 10 companies by market cap included a single Big Tech company: Microsoft. As of September 2020, the first seven are Apple, Microsoft, Amazon, Alphabet (Google), Alibaba, Facebook, and Tencent.<sup>38</sup> The number of computer science (CS) graduates in the U.S. more than tripled<sup>8</sup> and does not come close to meeting demand.

Rather than simply revise or update the 2009 survey, I have chosen to view advances in computing, optimization, and machine learning through a P vs. NP lens. I look at how these advances bring us closer to a world in which  $P = NP$ , the limitations still presented by P vs. NP, and the new opportunities of study which have been created. In particular, I look at how we are heading toward a world I call “Optiland,” where we can almost miraculously gain many of the advantages of  $P = NP$  while avoiding some of the disadvantages, such as breaking cryptography.

As an open mathematical problem, P vs. NP remains one of the most important; it is listed on the Clay Mathematical Institute’s Millennium Problems<sup>21</sup> (the organization offers a million-dollar bounty for the solution). I close the article by describing some new theoretical computer science results that, while not getting us closer to solving the P vs. NP question, show us that thinking

## » key insights

- The P vs. NP problem turned 50 in 2021 and its resolution remains far out of reach. Dramatic advances in algorithms and hardware have allowed us to tackle many NP-complete problems



about P vs. NP still drives much of the | groups, but nothing you do seems to | we can efficiently solve the Clique p

Fifty Years of P vs. NP and the Possibility of the Impossible  
By Lance Fortnow Communications of the ACM, January 2022, Vol. 65 No. 1, Pages 76-85 10.1145/3460351  
<https://cacm.acm.org/magazines/2022/1/257448-fifty-years-of-p-vs-np-and-the-possibility-of-the-impossible/fulltext>

# Optiland

Russell Impagliazzo, in a classic 1995 paper,<sup>20</sup> described five worlds with varying degrees of possibilities for the P vs. NP problem:

- **Algorithmica:**  $P = NP$  or something "morally equivalent," such as fast probabilistic algorithms for NP.
- **Heuristica:** NP problems are hard in the worst case but easy on average.
- **Pessiland:** We can easily create hard NP problems, but not hard NP problems where we know the solution. This is the worst of all possible worlds, since we can neither solve hard problems on average nor do we get any apparent cryptographic advantage from the difficulty of these problems.
- **Minicrypt:** Cryptographic one-way functions exist, but we do not have public-key cryptography.
- **Cryptomania:** Public-key cryptography is possible—that is, two parties can exchange secret messages over open channels.

These worlds are purposely not formally defined but rather suggest the unknown possibilities given our knowledge of the P vs. NP problem. The general belief, though not universal, is that we live in Cryptomania.

Impagliazzo draws upon a "you can't have it all" from P vs. NP theory. You can either solve hard NP problems or have cryptography, but you can't have both (you can have neither). Perhaps, though, we are heading to a de facto *Optiland*. Advances in machine learning and optimization in both software and hardware are allowing us to make progress on problems long thought difficult or impossible—from voice recognition to protein folding—and yet, for the most part, our cryptographic protocols remain secure.

# Complexity classes

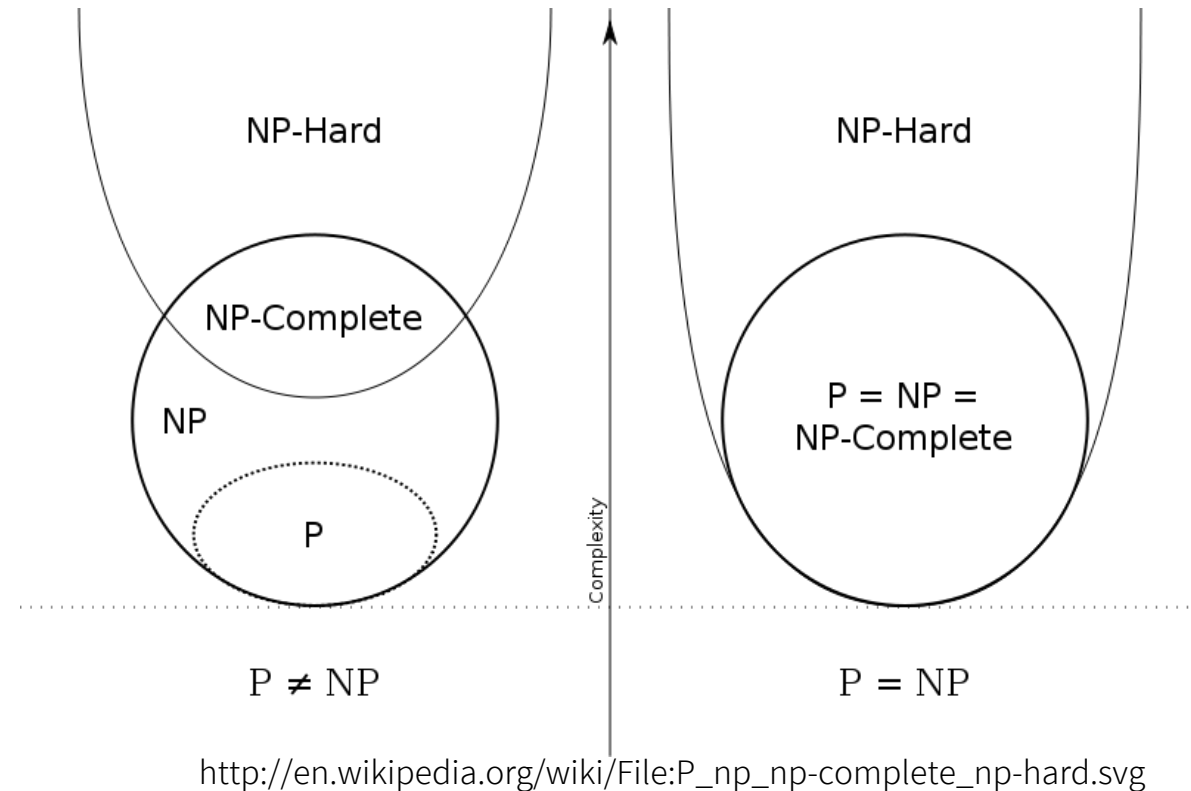
**Class P:** class of problems that can be solved in  $O(n^k)$

**Class NP:** class of problems that can be verified in  $O(n^k)$

**Class NP-hard:** a class of problems that are "at least as hard as the hardest problems" in NP

**Class NP-complete (NPC):** class of problems in both NP and NP-hard

Hardness relationship can be determined via polynomial-time reduction

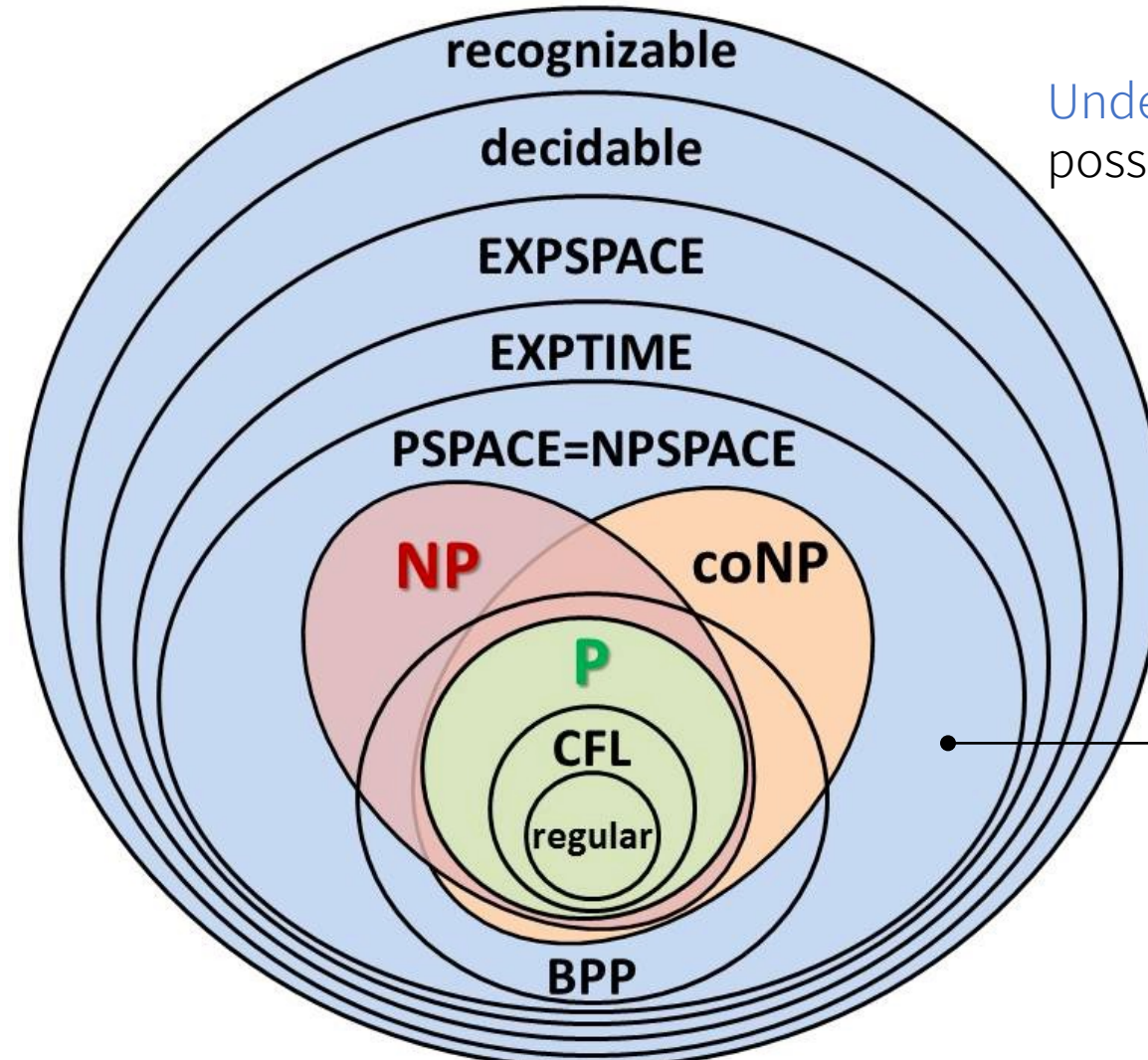




# The importance of class NPC

- Solving one NPC problem can lead to solving all NP problems
  - NPC problems are the hardest in NP
  - This is why it's called NP-“complete”
- Showing that a problem is “too hard to solve” by proving it is no easier than a NPC problem
  - NPC problems can be good reference points to judge whether an unknown problem is solvable in polynomial time

# More complexity classes



Undecidability. No algorithm possible, such as Halting problem.

Example: chess and Go

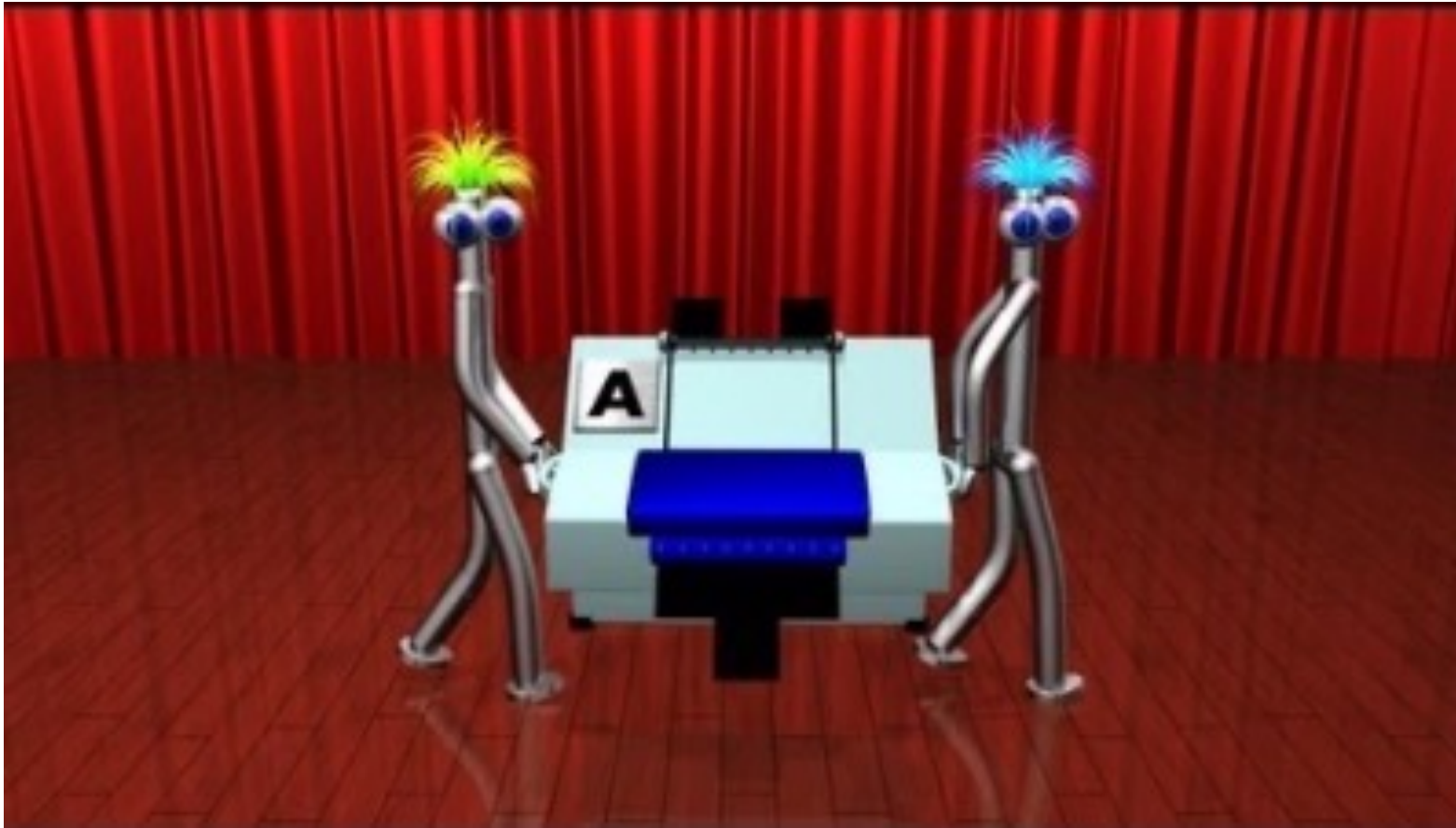


## Halting Problem

Given a program and an input, determine whether the program will finish running, or continue to run forever.

# Theorem: Halting problem is undecidable

Proof (animated version)



# Theorem: Halting problem is undecidable

## Proof

- Let's show that the halting problem is **undecidable** by contradiction
- Suppose  $h$  can determine whether a program  $p$  halts on input  $x$ 
  - $h(p, x) = \text{return } (p \text{ halts on input } x)$  [Eq. 1]
- Define  $g(p) = \text{if } h(p, p) \text{ is } 0 \text{ then return } 0 \text{ else } \textit{HANG}$
- $\Rightarrow g(g) = \text{if } h(g, g) \text{ is } 0 \text{ then return } 0 \text{ else } \textit{HANG}$  [Eq. 2]
- Consider whether  $g$  halts on  $g$ , both cases contradict the assumption:
  - $g$  halts on  $g$ : then  $h(g, g) = 1$  [Eq. 1], which would make  $g(g)$  hang [Eq. 2]
  - $g$  does not halt on  $g$ : then  $h(g, g) = 0$  [Eq. 1], which would make  $g(g)$  halt [Eq. 2]

# Decision Problems

## Decision problems

The answer to the problem is simply “yes” or “no” (or “1” or “0”)



- Examples
  - **PRIME**: Given a natural number  $x$ , is  $x$  a prime?
  - **COLOR**: Given a graph  $G = (V, E)$  and a value  $k$ , can we color the vertices with  $k$  colors such that no adjacent vertices have the same color?
  - **MST**: Given a graph  $G = (V, E)$  and a bound  $K$ , is there a spanning tree with a cost at most  $K$ ?
  - **KNAPSACK**: Given a knapsack of capacity  $C$ , a set of objects with weights and values, and a target value  $V$ , is there a way to fill the knapsack with at least  $V$  value?

## Optimization problems

Each feasible solution to the problem has an associated value, and we wish to find a feasible solution with the best value (i.e., maximum or minimum)

- Examples
  - **MST-OPT**: Given a graph  $G = (V, E)$ , find the minimum spanning tree of  $G$
  - **KNAPSACK-OPT**: Given a knapsack of capacity  $C$  and a set of objects with weights and values, fill the knapsack so as to maximize the total value

# Converting an optimization problem to a decision problem

- Every optimization problem has a decision version that is **no harder than\*** the optimization problem.
- In fact, they have **equal\*** computational difficulty.

\* In the sense that they are in the same **complexity class**

$P_{opt}$ : given a graph and  $(s, t)$ , find the length of the shortest path between  $s$  and  $t$

$P_{dec}$ : given a graph,  $(s, t)$ , and  $k$ , determine whether there is a path between  $s$  and  $t$  whose length  $\leq k$

Q: Suppose algorithm  $A_{opt}$  solves problem  $P_{opt}$ , and algorithm  $A_{dec}$  solves problem  $P_{dec}$ . Can we use  $A_{opt}$  to solve  $P_{dec}$ ? Can we use  $A_{dec}$  to solve  $P_{opt}$ ?

Using  $A_{opt}$  to solve  $P_{dec}$ : check if the optimal value  $\leq k$  (in the minimization problem)

Using  $A_{dec}$  to solve  $P_{opt}$ : apply binary search on the value range, linear to the value's binary representation

# Converting an optimization problem to a decision problem

- Every optimization problem has a decision version that is **no harder than**\* the optimization problem.
- In fact, they have **equal**\* computational difficulty.

\* In the sense that they are in the same **complexity class**

Q: To convert a maximization problem to a related decision problem, we can impose a \_\_\_\_\_ bound on the value to be optimized. Vice versa.

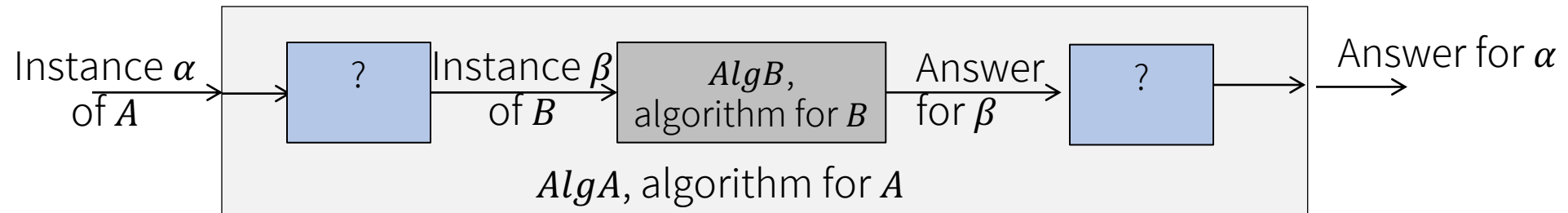
Lower



# Reduction (歸約)

# Reduction

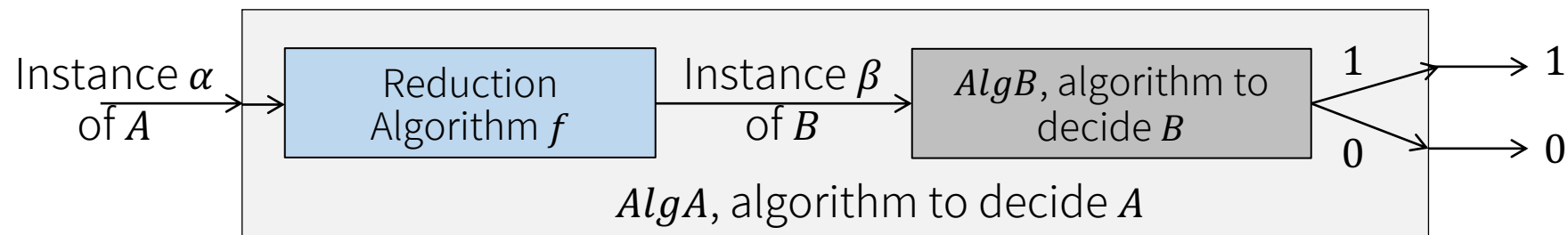
- General concept: Problem  $A$  reduces to problem  $B$  if we can use an algorithm that solves  $B$  to help solve  $A$



Complexity of the Algorithm for  $A$ ?

# Reduction of decision problems

- A **reduction**  $f$  is an algorithm for **transforming every instance** of a problem  $A$  into an instance of another problem  $B$ , and, for all  $\alpha$ ,  $AlgA(\alpha) = 1$  **if and only if**  $AlgB(f(\alpha)) = 1$ 
  - Thus, we can use  $AlgB$  to construct  $AlgA$  for solving problem  $A$
- For ease of understanding, try replacing  $A$  and  $B$  with simple yet concrete problems
  - Example:  $A$  is “Can 2 divide  $x$ ?”, and  $B$  is “Can  $y$  divide  $x$ ?”



Q: Suppose

Problem A is “Can 2 divide  $x$ ?”

Problem B is “Can  $y$  divide  $x$ ?”

$x$  and  $y$  are both nature numbers.

Answer the following questions:

(1) What are Problem A’s instances?

(2) What are Problem B’s instances?

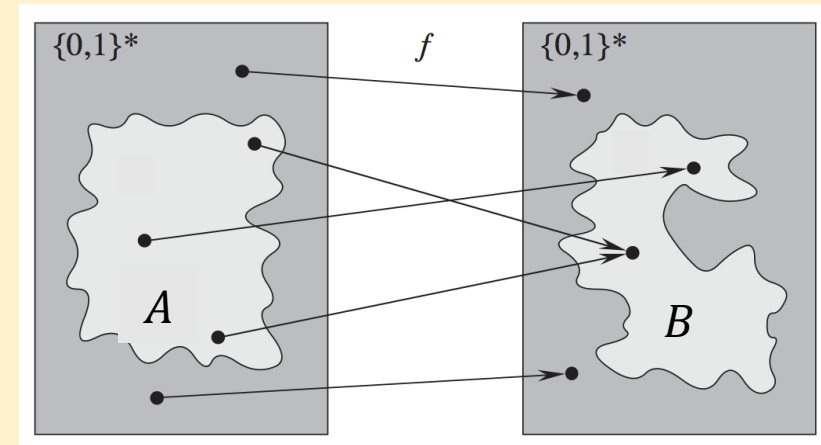
(3) Show that A can be reduced to B by constructing a reduction  $f$

(1)  $\alpha \in \{1, 2, 3, \dots\}$

(2)  $\beta \in \{(1,1), (1,2), \dots, (2,1), (2,2), \dots\}$

(3)  $f(\alpha) = (\alpha, 2)$ , this can be done in  $O(1)$ .

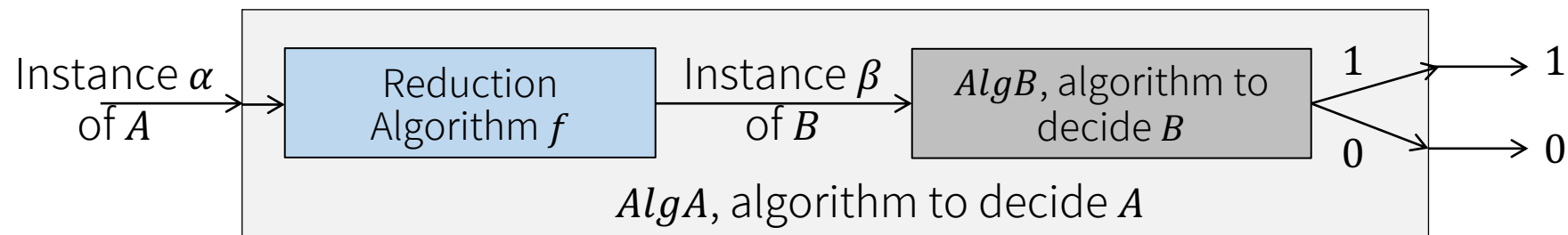
Also, trivially  $AlgB(f(\alpha)) = 1$  iff  $AlgA(\alpha) = 1$ .



Note that  $\{f(\alpha) | AlgA(\alpha) = 1\}$  is a subset of  $\{\beta | AlgB(\beta) = 1\}$

# Polynomial-time reduction

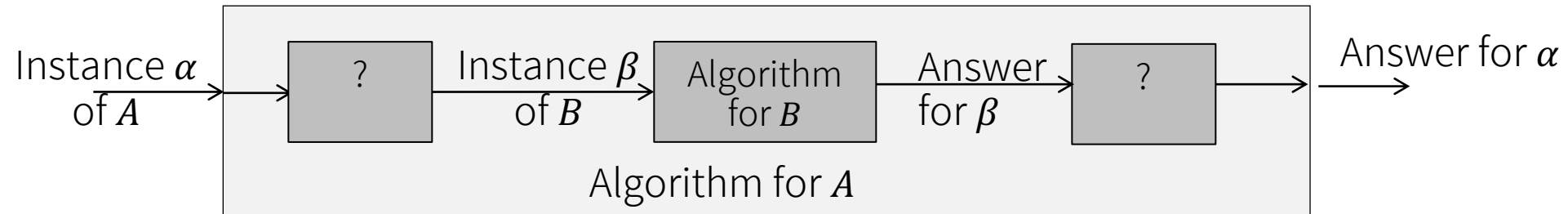
- A polynomial-time reduction ( $A \leq_p B$ ) is a polynomial-time algorithm for transforming every instance of a problem  $A$  into an instance of another problem  $B$ 
  - Can help determine the hardness relationship between problems (within a polynomial-time factor)
  - $A \leq_p B$  implies  $A$  is no harder than  $B$ ; equivalently,  $B$  is at least as hard as  $A$



# Applications of reduction

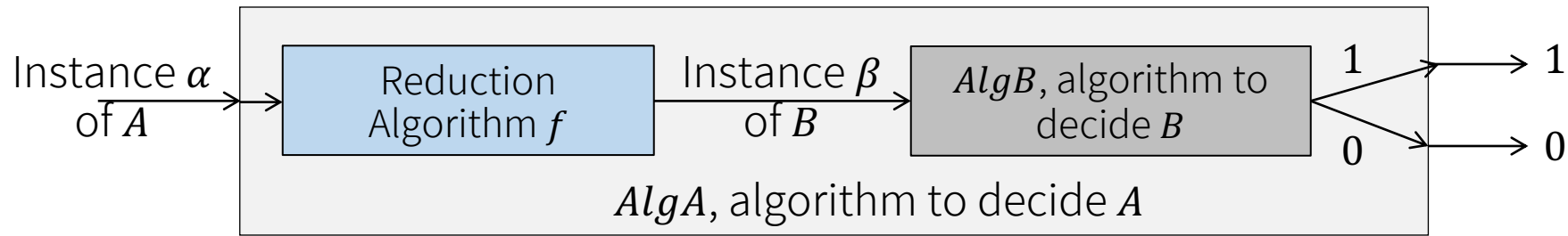
1. **Designing algorithms:** if we can reduce  $A$  to  $B$ , then we can solve  $A$  using a given algorithm for  $B$ .
2. **Classifying problems:** if  $A \leq_p B$  and  $B \leq_p A$ , then  $A$  and  $B$  have the same hardness level.
3. **Proving limits:** if  $A \leq_p B$  and  $A$  is hard, then so is  $B$ .
  - Use frequently in NP-completeness proofs.
  - 目的是證明  $B$  為 intractable，而非設計可用的演算法！

# Applications of reduction: designing algorithms



- Example #1:
  - A = all-pairs shortest path problem
  - B = single-source shortest path problem
- Example #2:
  - A = finding arbitrage opportunity
  - B = detecting negative cycles
- Any other examples?

# Applications of reduction: proving limits



- A common usage in NP-completeness proofs
  - Known:  $A$  is NPC
  - Goal: prove that  $B$  is at least as hard as  $A$  (i.e., NP-hard)
  - Approach: show that  $A \leq_p B$  by constructing a polynomial-time reduction algorithm to convert every  $\alpha$  to a  $\beta$



# Practice: Show $\text{PARTITION} \leq_p \text{KNAPSACK}$

## KNAPSACK:

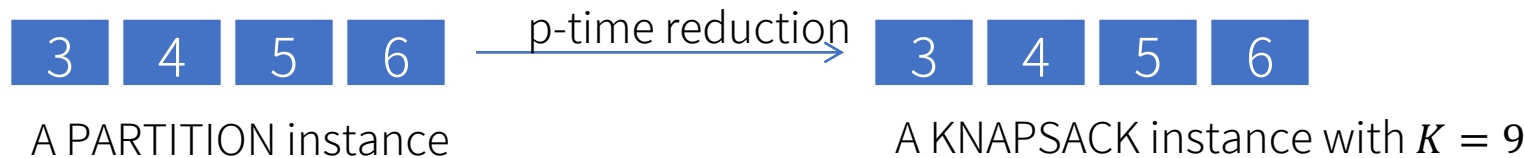
Given a set  $\{a_1, \dots, a_n\}$  of non-negative integers, and an integer  $K$ , decide if there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = K$ .

## PARTITION:

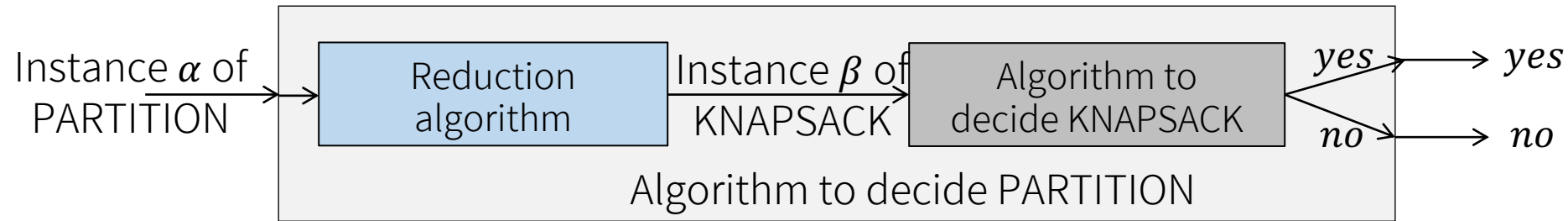
Given a set of non-negative integers  $\{a_1, \dots, a_n\}$ , decide if there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$

If we can solve KNAPSACK, how can we use that to solve PARTITION?

Polynomial-time reduction: Set  $K = \frac{1}{2} \sum_{i \in 1, \dots, n} a_i$



# Practice: Show $\text{PARTITION} \leq_p \text{KNAPSACK}$



If we can solve KNAPSACK, how can we use that to solve PARTITION?

Polynomial-time reduction: Set  $K = \frac{1}{2} \sum_{i \in 1, \dots, n} a_i$

Correctness proof: show that  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes **if and only if**  $\text{PARTITION}(\{a_1, \dots, a_n\})$  returns yes

# Practice: Show $\text{PARTITION} \leq_p \text{KNAPSACK}$

Polynomial-time reduction: Set  $K = \frac{1}{2} \sum_{i \in 1, \dots, n} a_i$

Correctness proof: show that  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes **if and only if**  $\text{PARTITION}(\{a_1, \dots, a_n\})$  returns yes

(1) If  $\text{PARTITION}(\{a_1, \dots, a_n\})$  returns yes  $\rightarrow$   $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes

- If  $\text{PARTITION}(\{a_1, \dots, a_n\})$  returns yes, then there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = \sum_{i \notin P} a_i = K$ .
- Thus,  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  will return yes

(2) If  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes  $\rightarrow$   $\text{PARTITION}(\{a_1, \dots, a_n\})$  returns yes

- If  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes, then there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = K$ .
- The rest (not in  $P$ ) must sum up to  $K$  as well. Thus,  $\text{PARTITION}(\{a_1, \dots, a_n\})$  will return yes

# Practice: Show $\text{KNAPSACK} \leq_p \text{PARTITION}$

## KNAPSACK:

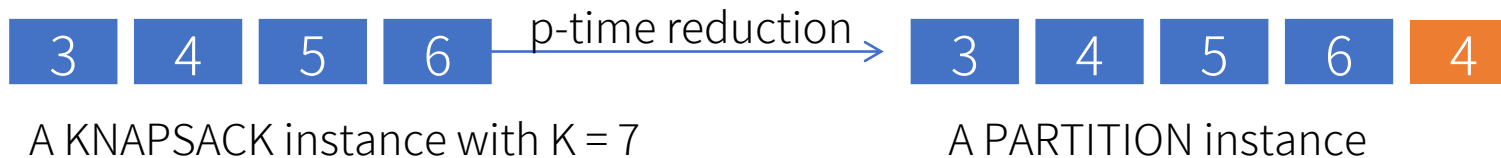
Given a set  $\{a_1, \dots, a_n\}$  of non-negative integers, and an integer  $K$ , decide if there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = K$ .

## PARTITION:

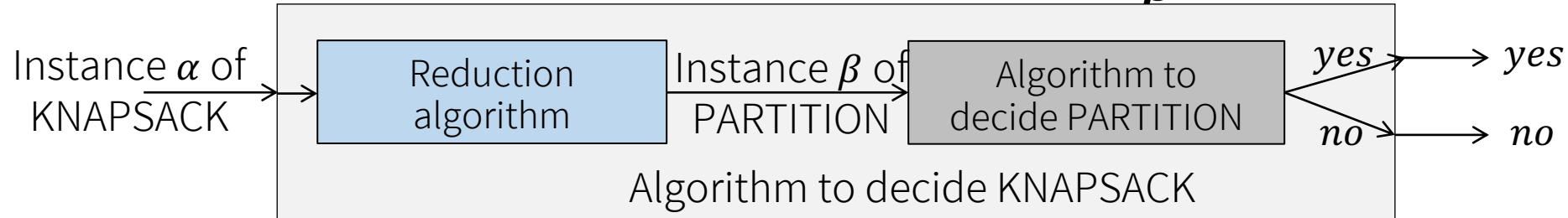
Given a set of non-negative integers  $\{a_1, \dots, a_n\}$ , decide if there is a subset  $P \subseteq [1, n]$  such that  $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$ .

If we can solve PARTITION, how can we use that to solve KNAPSACK?

Polynomial-time reduction: Add  $a_{n+1} = |H - 2K|$ , where  $H = \sum_{i \in 1, \dots, n} a_i$



# Practice: Show $\text{KNAPSACK} \leq_p \text{PARTITION}$



If we can solve PARTITION, how can we use that to solve KNAPSACK?

Polynomial-time reduction: Add  $a_{n+1} = |H - 2K|$ , where  $H = \sum_{i \in 1, \dots, n} a_i$

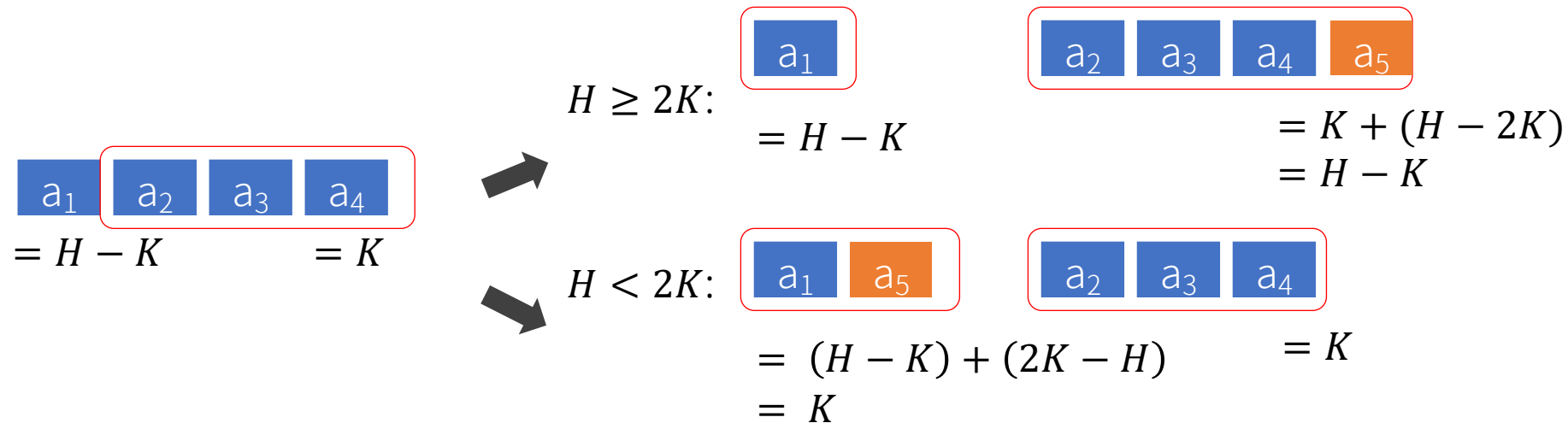
Correctness proof: show that  $\text{PARTITION}(\{a_1, \dots, a_n, a_{n+1}\})$  returns yes **if and only if**  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes

# Practice: Show $\text{KNAPSACK} \leq_p \text{PARTITION}$

Polynomial-time reduction: Add  $a_{n+1} = |H - 2K|$ , where  $H = \sum_{i \in 1, \dots, n} a_i$

Correctness proof: show that  $\text{PARTITION}(\{a_1, \dots, a_n, a_{n+1}\})$  returns yes **if and only if**  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes

(1) If  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes  $\rightarrow \text{PARTITION}(\{a_1, \dots, a_n, a_{n+1}\})$  returns yes

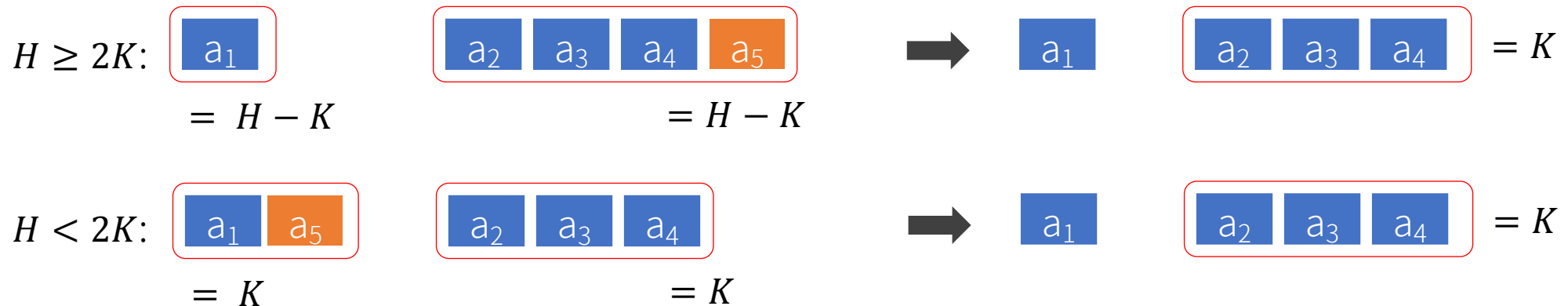


# Practice: Show $\text{KNAPSACK} \leq_p \text{PARTITION}$

Polynomial-time reduction: Add  $a_{n+1} = |H - 2K|$ , where  $H = \sum_{i \in 1, \dots, n} a_i$

Correctness proof: show that  $\text{PARTITION}(\{a_1, \dots, a_n, a_{n+1}\})$  returns yes **if and only if**  $\text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes

(2) If  $\text{PARTITION}(\{a_1, \dots, a_n, a_{n+1}\})$  returns yes  $\rightarrow \text{KNAPSACK}(\{a_1, \dots, a_n\}, K)$  returns yes

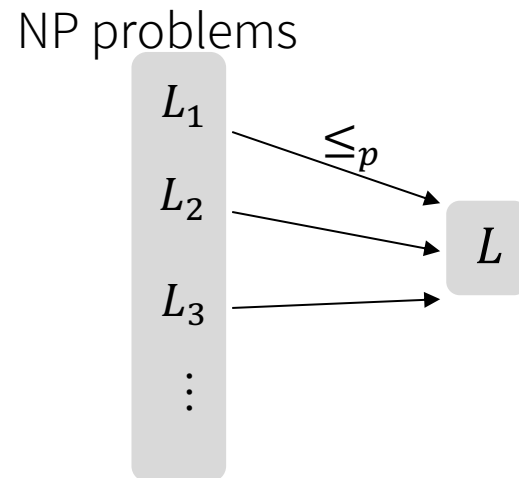


# Proving NP-completeness



# Proving NP-Completeness

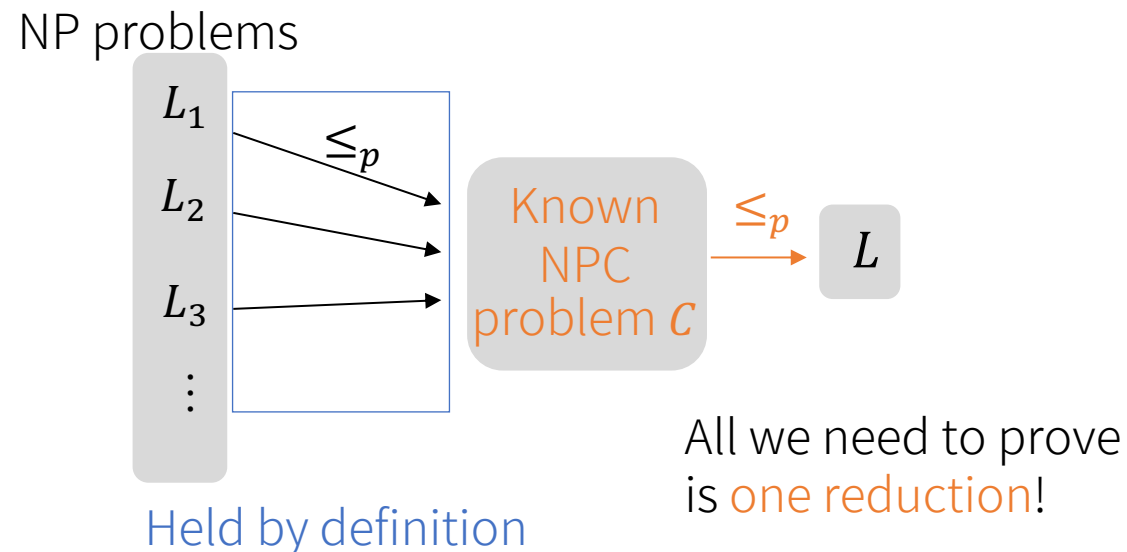
- Class NP-Complete (NPC): class of decision problems in both NP and NP-hard
- In other words, a decision problem  $L$  is NP-complete if
  1.  $L \in \text{NP}$
  2.  $L \in \text{NP-Hard}$  (that is,  $L' \leq_p L$  for every  $L' \in \text{NP}$ )



Q: How to prove NP-Hard (every NP problem reduces to  $L$ )?

# Proving NP-Completeness

- Class NP-Complete (NPC): class of decision problems in both NP and NP-hard
- In other words, a decision problem  $L$  is NP-complete if
  1.  $L \in \text{NP}$
  2.  $L \in \text{NP-Hard}$  (that is,  $L' \leq_p L$  for every  $L' \in \text{NP}$ )
- The second part can be simplified into showing  $C \leq_p L$  for a known NPC problem  $C$
- Note, if  $A \leq_p B$  and  $B \leq_p C$ , then  $A \leq_p C$

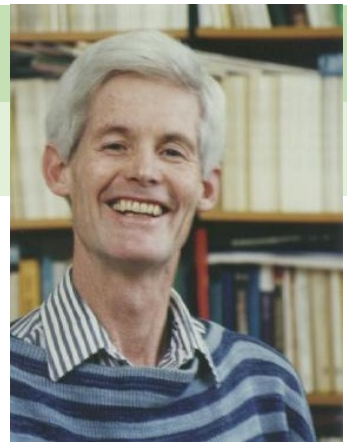


# The first NP-Complete problem: Circuit-Satisfiability Problem

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

## Cook's Theorem

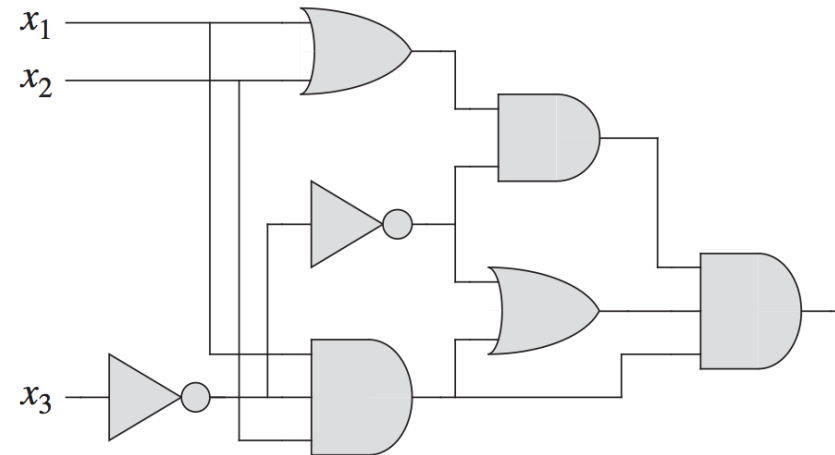
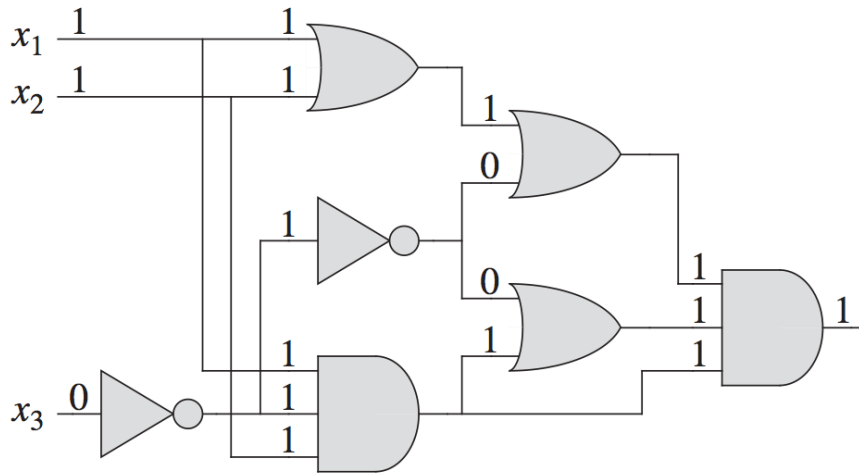
The Circuit-Satisfiability Problem (CIRCUIT-SAT) is NP-complete



Stephen Cook  
Turing Award (1982)

# The first NP-Complete problem: Circuit-Satisfiability Problem

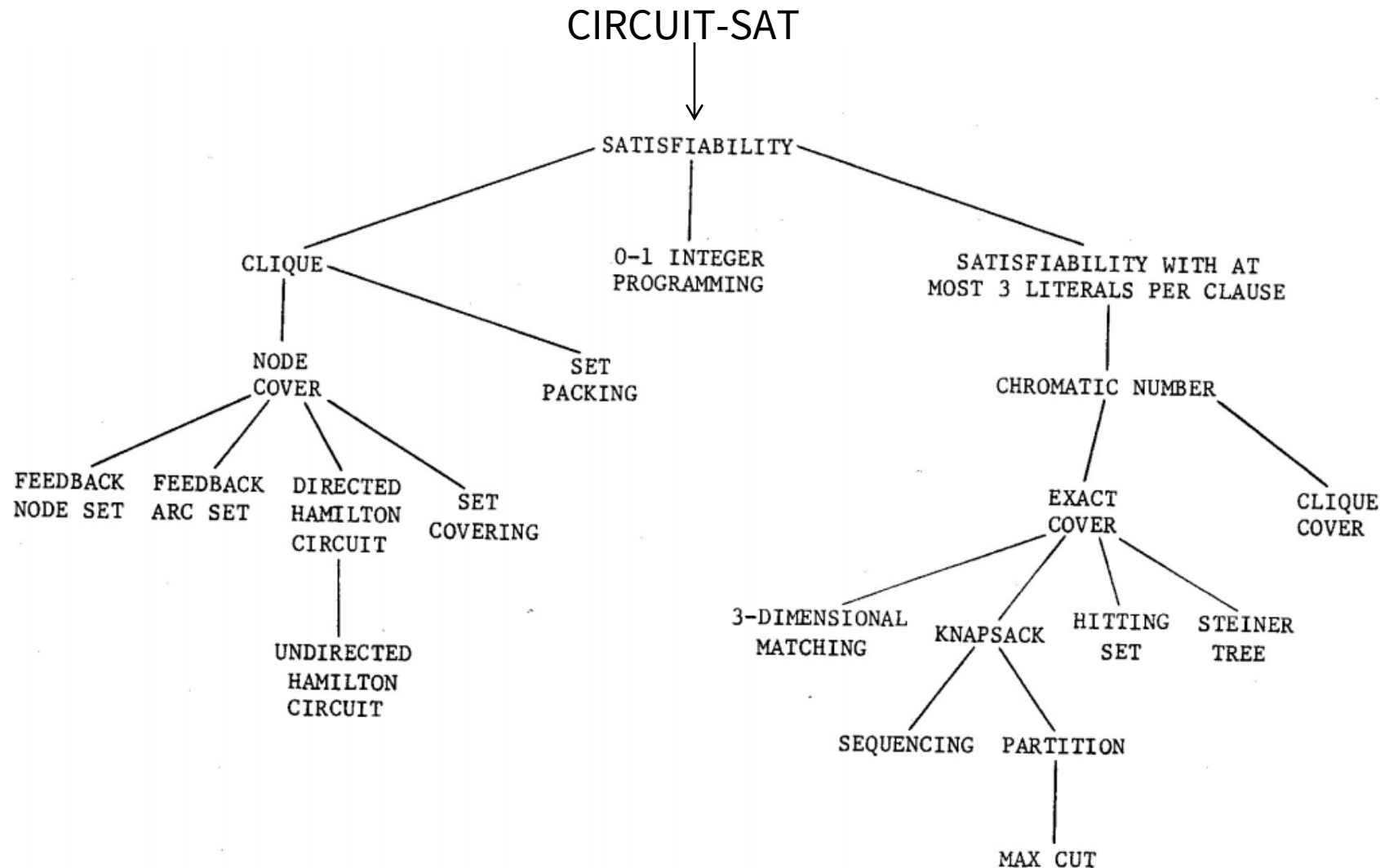
- Given a Boolean combinational circuit composed of AND, OR, and NOT gates, is it **satisfiable**?
  - Satisfiable = there exists an assignment s.t. the circuit outputs 1



# Karp's 21 NP-complete problems



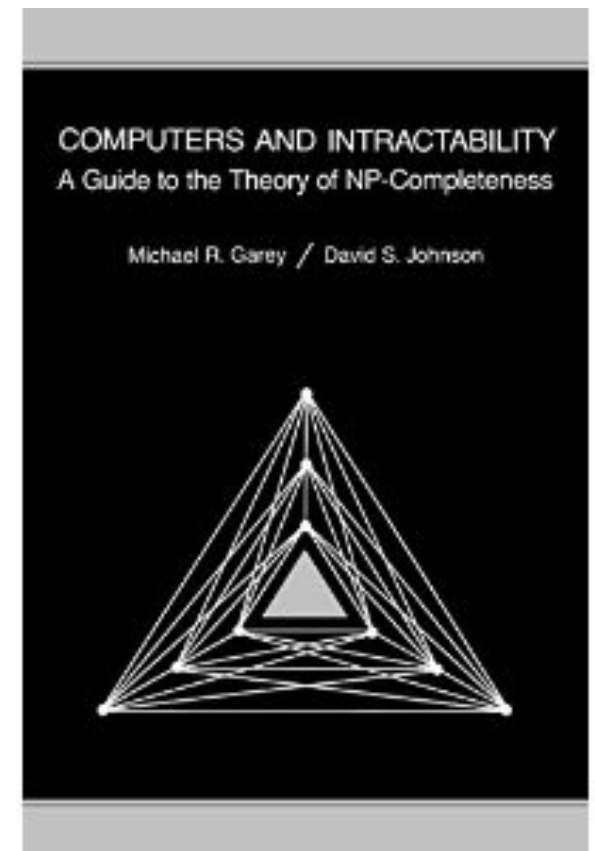
Richard M. Karp  
Turing Award (1985)



# More NP-complete problems

- “Computers and Intractability” by Garey and Johnson includes more than 300 NP-complete problems
  - Most cited book in CS

1. M R Garey, D S Johnson  
[Computers and Intractability: A Guide to the Theory of NP-Completeness](#) 1979  
8562
2. T H Cormen, C E Leiserson, R L Rivest  
[Introduction to Algorithms](#) 1991  
7126

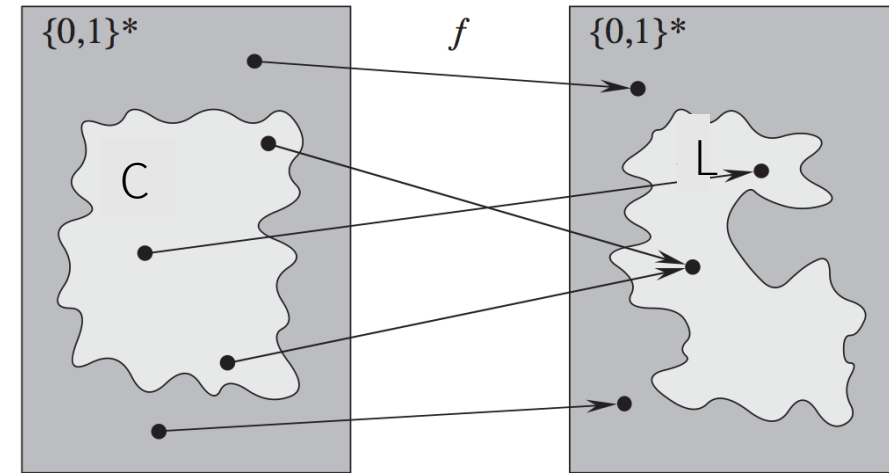


# Proving NP-Completeness

$L \in \text{NP-Complete} \Leftrightarrow L \in \text{NP} \text{ and } L \in \text{NP-hard}$

Step-by-step approach for proving  $L$  in NPC:

1. Prove  $L \in \text{NP}$
2. Prove  $L \in \text{NP-hard}$  ( $C \leq_p L$ )
  - ① Select a **known NPC problem  $C$**
  - ② **Construct a reduction  $f$**  transforming every instance of  $C$  to an instance of  $L$
  - ③ Prove that  $x$  in  $C$  **if and only if**  $f(x)$  in  $L$  for all  $x$  in  $\{0,1\}^*$
  - ④ Prove that  $f$  is a **polynomial-time transformation**



# Fun examples



# 差一點差很多之 complexity class

猜猜以下每組中哪一個 in P (if  $P \neq NP$ ) ?

Shortest simple path

vs.

Longest simple path

Eulerian tour

vs.

Hamiltonian cycle

LCS with two input  
sequences

vs.

LCS with arbitrary  
input sequences

Degree-constrained  
spanning tree

vs.

Minimum spanning  
tree

# How about these games?

Sudoku

4	1						3	7
6	2						9	8
		5	9		2	1		
		2	7		3	4		
				5				
		1	4		6	9		
		4	3		8	6		
2	5						4	3
9	6						1	2

Candy Crush



MineSweeper



# 別解問題の計算量と完全性、およびパズルへの応用

八登 崇之

東京大学大学院理学系研究科  
情報科学専攻  
yato@is.s.u-tokyo.ac.jp

瀬田 剛広

東京大学大学院情報理工学系研究科  
コンピュータ科学専攻  
seta@is.s.u-tokyo.ac.jp

## 概要

問題  $\Pi$  に対する別解問題 (ASP) というのは、 $\Pi$  のインスタンス  $x$  とそれに対する 1 つの解  $s$  が与えられた時に、 $x$  の  $s$  以外の解を求める問題のことである。(新しい問題クラスとしての ASP の概念は Ueda と Nagao による。) 本論文では  $n$  個の解が与えられた時にもう 1 つの解を求める問題 ( $n$ -ASP) について考察する。特に、対応する解への変換も多項式時間で行えるような多項式時間 parsimonious 還元に関する完全性である ASP 完全性について考察する。

応用として、本論文では 3 つの有名なパズル、スリザーリンクとカックロ (クロスサム) とナンバープレイス (数独) についてその ASP 完全性 (NP 完全性を含意する) を示す。

## Complexity and Completeness of Finding Another Solution and Its Application to Puzzles

Takayuki YATO

Department of Information Science  
Graduate School of Science  
The University of Tokyo  
yato@is.s.u-tokyo.ac.jp

Takahiro SETA

Department of Computer Science  
Graduate School of Information Science and Technology  
The University of Tokyo  
seta@is.s.u-tokyo.ac.jp

## Abstract

The Another Solution Problem (ASP) of a problem  $\Pi$  is the following problem: for a given instance  $x$  of  $\Pi$  and a solution  $s$  to it, find a solution to  $x$  other than  $s$ . (The notion of ASP as a new class of problems was first introduced by Ueda and Nagao.) In this paper we consider  $n$ -ASP, the problem to find another solution when  $n$  solutions are given. In particular we consider ASP-completeness, the completeness with respect to the parsimonious reductions which allow polynomial-time transformation of solutions.

As an application, we prove the ASP-completeness (which implies NP-completeness) of three popular puzzles: Slither Link, Cross Sum, and Number Place.

# Bejeweled, Candy Crush and other match-three games are (NP-)Hard!

## What is this all about?

This is an implementation of the reduction provided in the paper [Bejeweled, Candy Crush and other Match-Three Games are \(NP\)-Hard](#) which has been accepted for presentation at the [2014 IEEE Conference on Computational Intelligence and Games \(CIG 2014\)](#). To find more about what NP-Hard means you can read [this blog post](#) or the corresponding [page on Wikipedia](#).

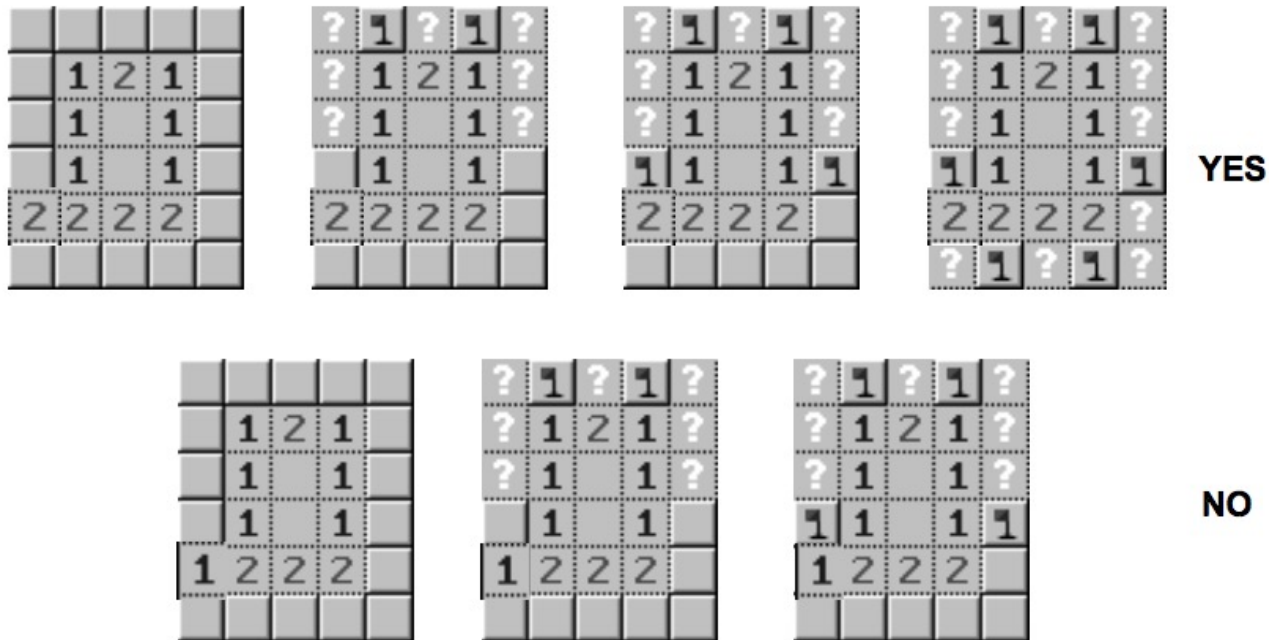
## About the authors

We are an Italian group of three people: [Luciano Gualà](#), [Stefano Leucci](#), and [Emanuele Natale](#). We had the weird idea to spend our weekends proving that Candy Crush Saga is NP-Hard. We also thought that it was nice to put online an implementation of our hardness reduction... so here it is!



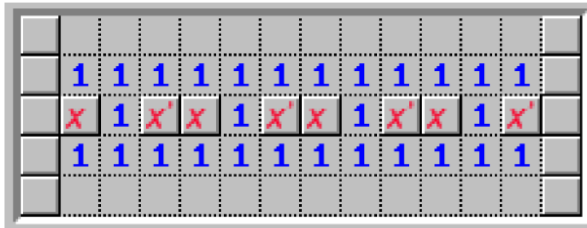
# Minesweeper consistency problem is also NP-complete

- Minesweeper consistency problem: Given a state of what purports to be a Minesweeper game, is it logically consistent?

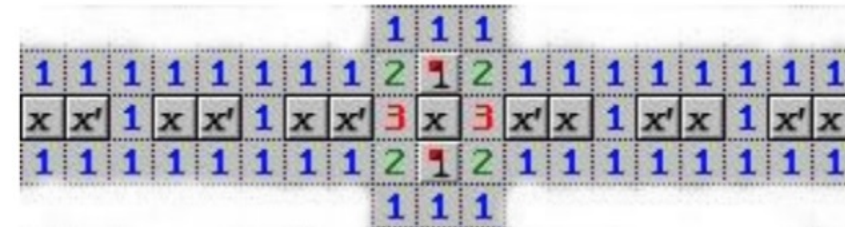




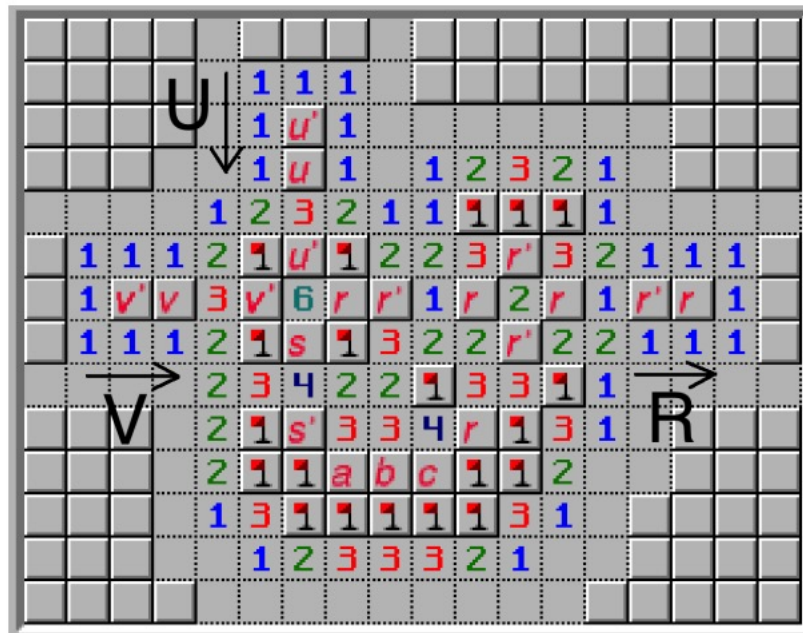
Wire



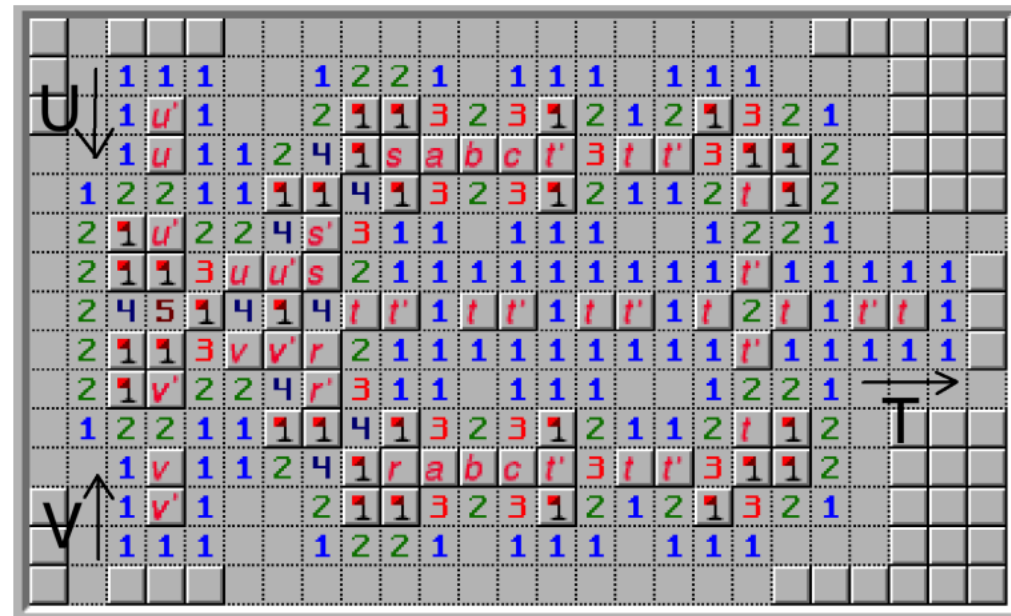
NOT Gate



OR Gate



AND Gate



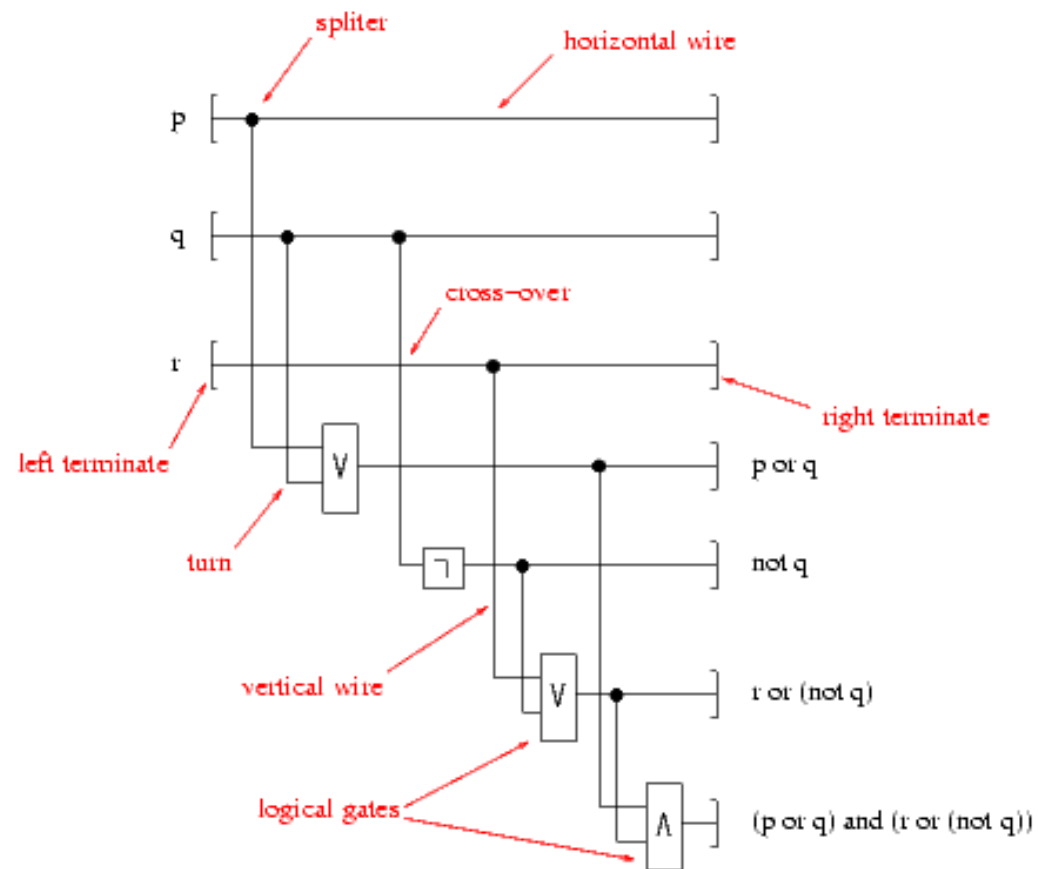


Figure 1 First step of translation

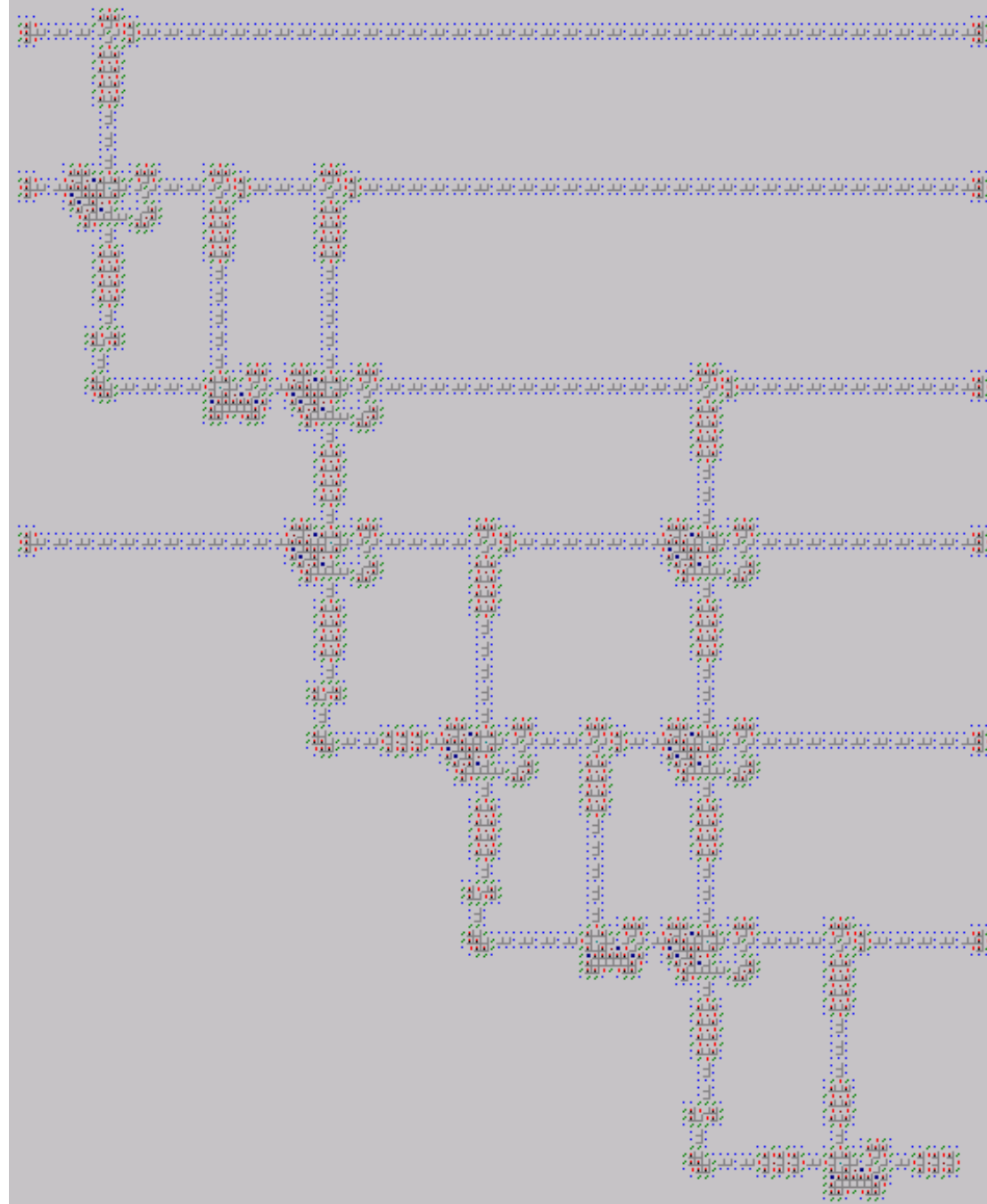


Figure 2 The minesweeper configuration corresponding to figure 1

**True/False:** If A can be reduced to B in polynomial time ( $A \leq_p B$ ), then A is no harder than B

**True/False:** if  $A \leq_p B$  and  $B \leq_p C$ , then  $A \leq_p C$

**True/False:** NP-Complete problems can be reduced to each other in polynomial time.

**True/False:** if  $A \leq_p B$  and B is in NPC, then A is in NPC too.

**True/False:** if  $A \leq_p B$  and A is in NPC, then B is in NPC too.



# Appendix: Polynomial-time Solving & Polynomial-time Verification

Chap. 34.1, 34.2

# Abstract problems

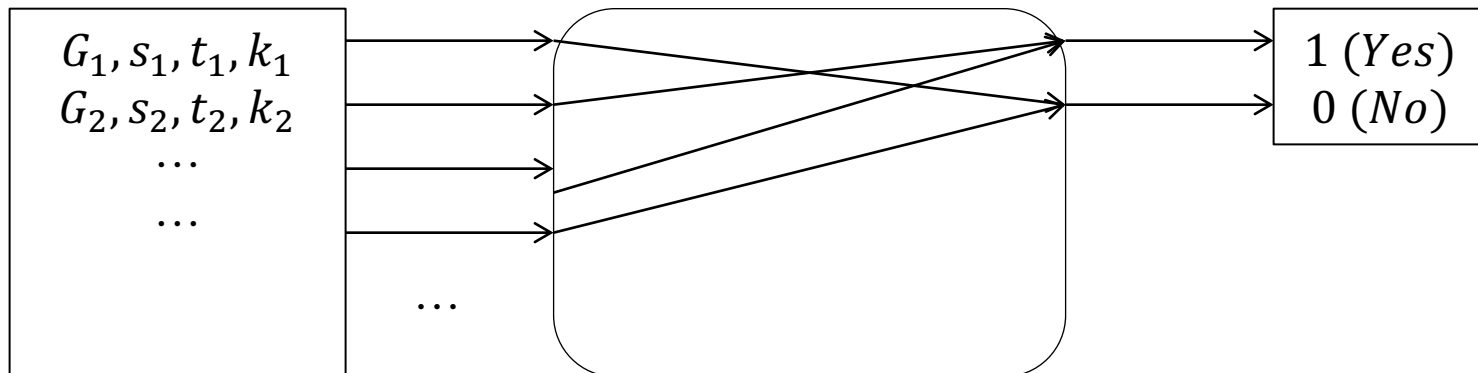
- $I$ : the set of problem instances
- $S$ : the set of problem solutions
- $Q$ : an abstract problem, defined as a **binary relation** on  $I$  and  $S$

Example of a decision problem, SHORTEST-PATH:

$I: \langle G, src, dest, k \rangle$   
所有可能的圖、起訖  
點、長度上限

$Q$ : SHORTEST-PATH  
是否找得到最短路徑長度  $\leq k$ ?

$S: \{0,1\}$



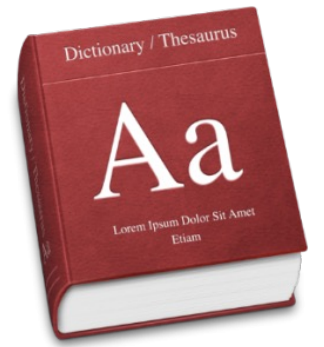
# Encoding of problem instances

- Convert an abstract problem instance into a binary string that is fed to a computer program



- A concrete problem is **polynomial-time solvable** if there exists an algorithm that solves any concrete instance of length  $n$  in time  $O(n^k)$  for some constant  $k$ 
  - Solvable = can produce a solution

# Formal-language theory



- An **Alphabet**  $\Sigma$  = a finite set of symbols
- A **Language**  $L$  over  $\Sigma$  = any set of strings made up from alphabet  $\Sigma$
- Notations
  - $\varepsilon$ : empty string
  - $\phi$ : empty language
  - $\Sigma^*$ : the language of all strings over  $\Sigma$

Q: In English, what is the alphabet  $\Sigma$  and the language  $L$ ?

- $\Sigma = \{a, b, c, \dots, z\}$
- $L =$  a set of English words

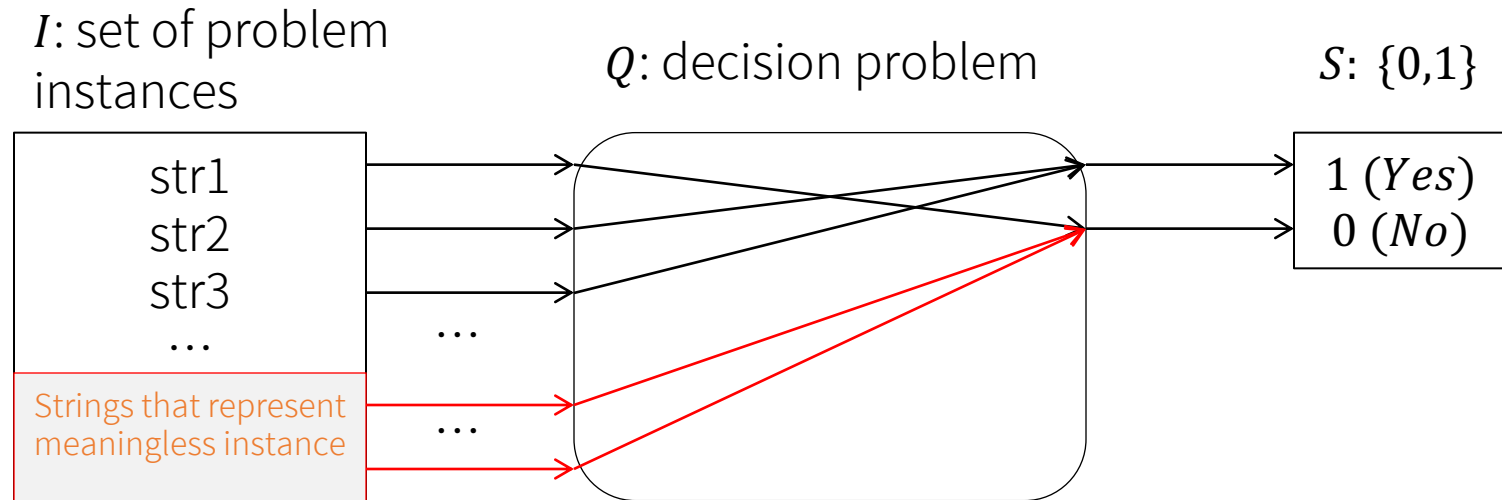
Q: In the language of binary representation of prime numbers, what is the alphabet  $\Sigma$  and the language  $L$ ?

- $\Sigma = \{0,1\}$
- $L = \{10, 11, 101, 111, \dots\}$

Q: If  $\Sigma = \{0,1\}$ , what is  $\Sigma^*$ ?

- $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

# Representation of a decision problem

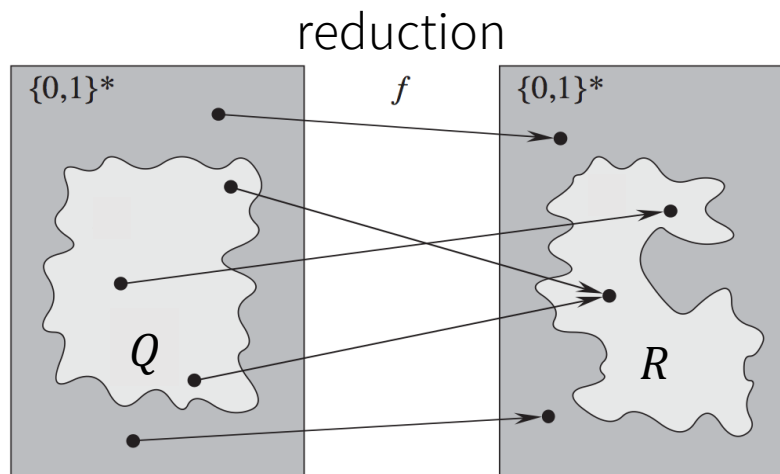


- $I$  = the set of problem instances =  $\Sigma^*$ , where  $\Sigma = \{0, 1\}$
- $Q$  = a decision problem = a language  $L$  over  $\Sigma = \{0, 1\}$  such that  $L = \{x \in \Sigma^* : Q(x) = 1\}$ 
  - 以答案為 1 的 instances 定義 decision problem  $Q$ !
  - $L = \{str2, str3\}$  in this simple example

## Decision problem

A decision problem  $Q$  can be defined as a language  $L$  over  $\Sigma = \{0, 1\}$  such that  $L = \{x \in \Sigma^*: Q(x) = 1\}$

- 以答案為 1 的 instances 定義 decision problem  $Q$
- $Q$  can be reduced to  $R$  means  $x \in Q \Leftrightarrow f(x) \in R$



# Definition of the complexity class $P$ in language-theoretic framework

- An algorithm  $A$  **accepts** a string  $x \in \{0,1\}^*$  if  $A(x) = 1$
- An algorithm  $A$  **rejects** a string  $x \in \{0,1\}^*$  if  $A(x) = 0$
- An algorithm  $A$  **accepts** a language  $L$  means...
  - $A$  accepts every string  $x \in L$
  - 對所有在  $L$  中的 string 都要能正確地輸出 YES
  - 對不在  $L$  中的 string 可能輸出 NO, 也可能無法判斷 (e.g., loop forever)
- An algorithm  $A$  **decides** a language  $L$  if  $A$  accepts  $L$  and  $A$  rejects every string  $x \notin L$ 
  - 對所有的string都要能正確地輸出YES/NO





# Definition of the complexity class P in language-theoretic framework

- **Class P:** a class of decision problems **solvable** in p-time
  - That is, given an instance  $x$  of a decision problem  $Q$ , its solution  $Q(x)$  (i.e., YES or NO) can be found in polynomial time
  - Hence, an alternative definition of class  $P$ :

## Definition of Class P

$P = \{L \subseteq \{0,1\}^* \mid \text{there exists an algorithm that decides } L \text{ in p-time}\}$

## Theorem 34.2

$P$  is also the class of language that can be accepted in polynomial time.  
 $P = \{L \subseteq \{0,1\}^* \mid L \text{ is accepted by a polynomial-time algorithm}\}$

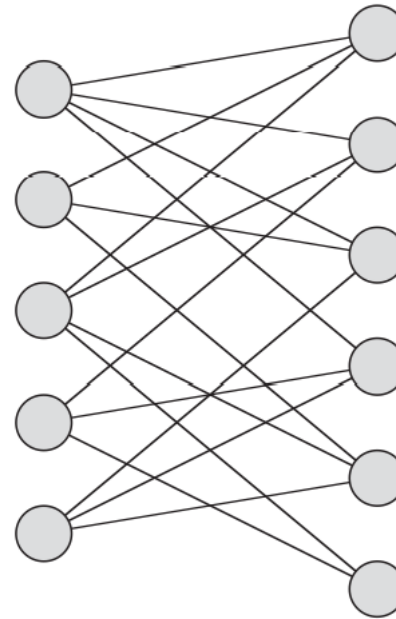
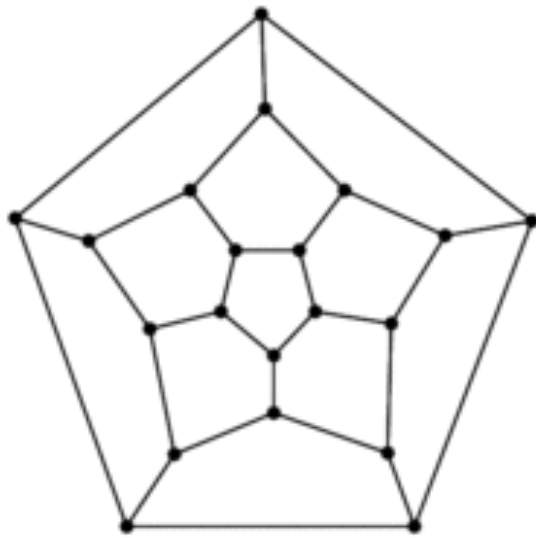
## Hamiltonian-Cycle Problem

**HAM-CYCLE** =  $\{\langle G \rangle \mid G \text{ has a Hamiltonian cycle}\}$

Q: Can you find a cycle visiting each vertex exactly once in the examples?

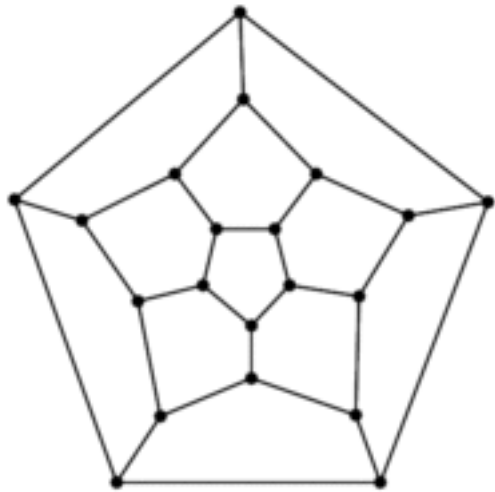
Q: Is this language decidable?

Q: Is this language decidable in polynomial time?



# Polynomial-time Verification: Hamiltonian-cycle problem

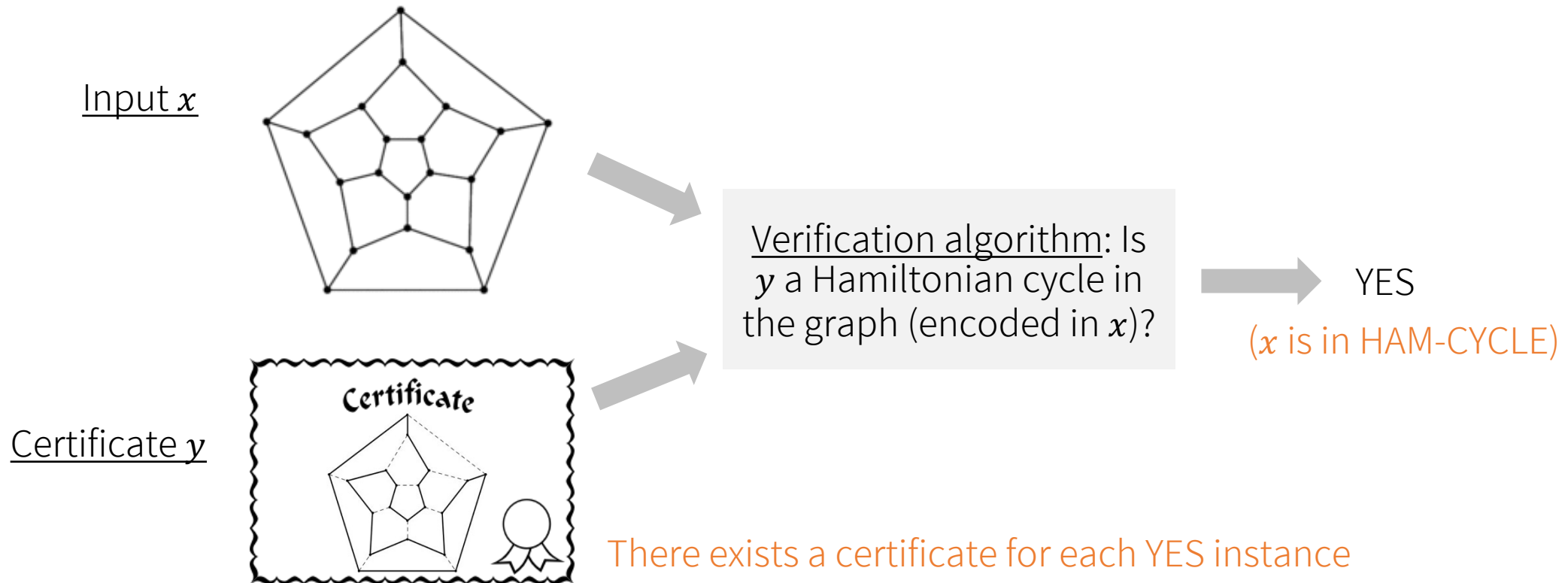
- Suppose a nice TA gives you a **certificate** – a vertex sequence that forms a Hamiltonian cycle in a given graph  $G$
- With this certificate, how much time does it take to **verify** that  $G$  indeed contains a Hamiltonian cycle?



# Verification algorithms

- **Verification algorithms** verify memberships in language with the help of certificates

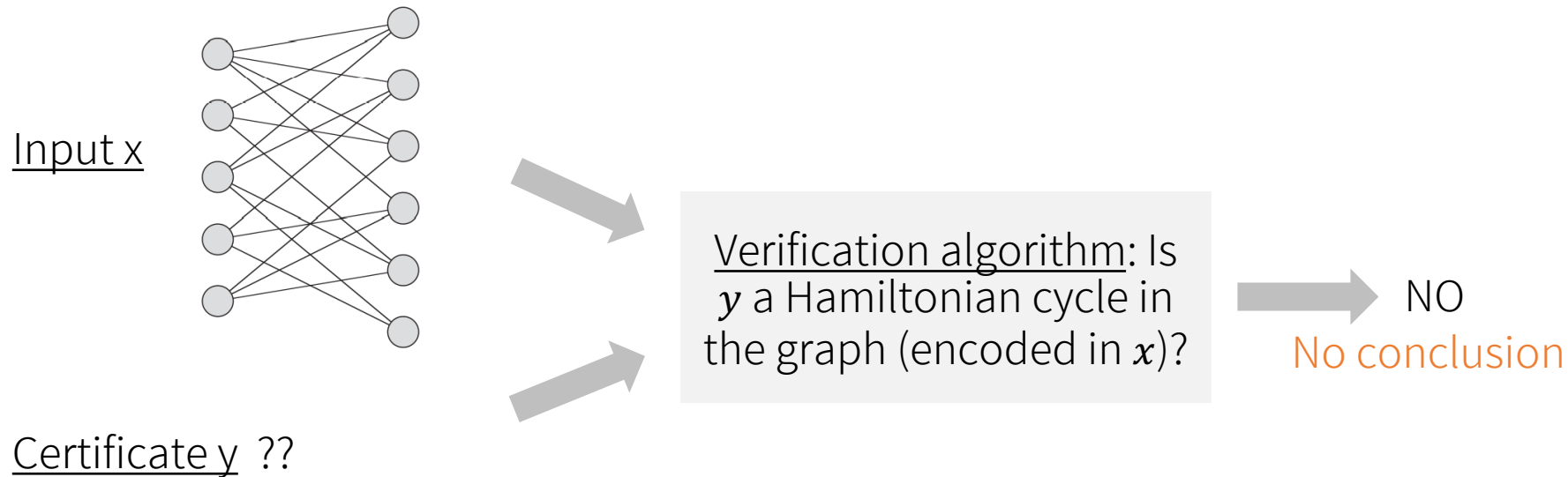
$\text{HAM-CYCLE} = \{\langle G \rangle \mid G \text{ has a Hamiltonian cycle}\}$



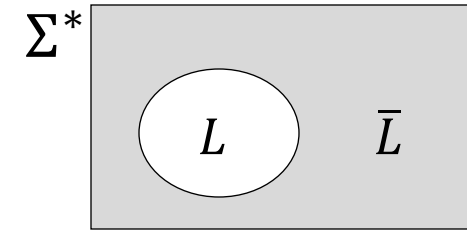
# Verification algorithms

- Verification algorithms verify memberships in language

$\text{HAM-CYCLE} = \{\langle G \rangle \mid G \text{ has a Hamiltonian cycle}\}$



# Co-NP

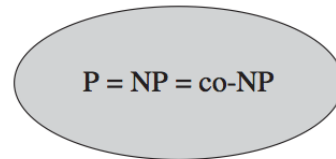


- **Class co-NP**: class of decision problems whose **complement** are in NP
  - Complement of  $L = \bar{L} = \Sigma^* - L$
- **HAM-CYCLE**: Given a graph  $G$ , is there a simple cycle that visits each vertex on  $G$  exactly once?
- **$\overline{\text{HAM-CYCLE}}$** : Given a graph  $G$ , does  $G$  contain no Hamiltonian cycle?
- It's still an open problem whether **HAM-CYCLE**  $\in$  co-NP
  - Equivalently, whether  **$\overline{\text{HAM-CYCLE}}$**   $\in$  NP
  - 是否能提供 certificate 證明 graph  $G$  沒有 Hamiltonian cycle?

# P, NP, and co-NP

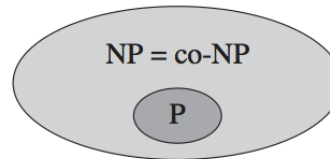
- Class P: class of decision problems solvable in  $O(n^k)$ 
  - $O(n^k)$  means polynomial in the input size ( $k$  is a constant)
- Class NP: class of decision problems verifiable in  $O(n^k)$
- Class co-NP: class of decision problems whose complement are in NP

If  $P = NP = \text{co-NP}$



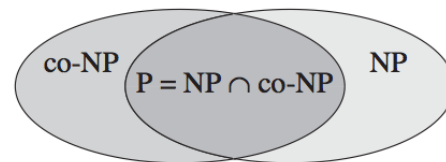
(a)

If  $P \neq NP$  and  $NP = \text{co-NP}$



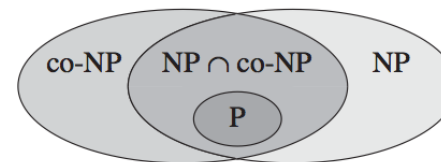
(b)

If  $P = NP \cap \text{co-NP}$



(c)

Most likely case!



(d)