Intractable Problems

Time-Bounded Turing Machines

Classes **P** and **NP**

Polynomial-Time Reductions

$O(1)$ constant time
$O(\log_2(n))$ logarithm
$O(n)$ linear
$O(n^2)$ quadratic
$O(n^k)$ polynomial
$O(k^n)$ exponential
$O(n!)$ factorial

k = constant
n = จำนวน input

Time-Bounded TM's

- A Turing machine that, given an input of length n, always halts within T(n) moves is said to be *T(n)-time bounded*.
  - The TM can be multitape.
  - Sometimes, it can be nondeterministic.
- The deterministic, multitape case corresponds roughly to "an O(T(n)) running-time algorithm."

บทคำนวณได้ในเวลา Polytime.

The class **P**

- If a DTM M is T(n)-time bounded for some polynomial T(n), then we say M is *polynomial-time* ("*polytime*") bounded.
- And L(M) is said to be in the class **P**.
- Important point: when we talk of **P**, it doesn't matter whether we mean "by a computer" or "by a TM" (next slide).

Polynomial Equivalence of Computers and TM's

- A multitape TM can simulate a computer that runs for time $O(T(n))$ in at most $O(T^2(n))$ of its own steps.
- If $T(n)$ is a polynomial, so is $T^2(n)$.

Examples of Problems in **P**

- Is w in L(G), for a given CFG G?
  - Input = w.
  - Use CYK algorithm, which is $O(n^3)$.
- Is there a path from node x to node y in graph G?
  - Input = x, y, and G.
  - Use Dijkstra's algorithm, which is O(n log n) on a graph of n nodes and arcs.

Running Times Between Polynomials

- You might worry that something like O(n log n) is not a polynomial.
- However, to be in **P**, a problem only needs an algorithm that runs in time less than some polynomial.
- Surely O(n log n) is less than the polynomial $O(n^2)$.

A Tricky Case: Knapsack

- The *Knapsack Problem* is: given positive integers i_1, i_2 ,…, i_n, can we divide them into two sets with equal sums?
- Perhaps we can solve this problem in polytime by a dynamic-programming algorithm:
  - Maintain a table of all the differences we can achieve by partitioning the first j integers.

The Class **NP**

หาคำตอบ. ใช้เวลาจากการทำ poly time

- The running time of a nondeterministic TM is the maximum number of steps taken along any branch.
- If that time bound is polynomial, the NTM is said to be *polynomial-time bounded*.

Non-terministic

Check ได้. ตรวจคำตอบ ถูก-ไม่ถูก ใช้เวลา ภายใน poly time

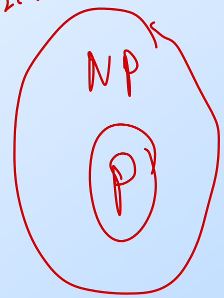
- And its language/problem is said to be in the class **NP**.

Example: **NP**

- The Knapsack Problem is definitely in **NP**, even using the conventional binary representation of integers.
- Use nondeterminism to guess one of the subsets.
- Sum the two subsets and compare.

Deterministic Non-Deterministic.

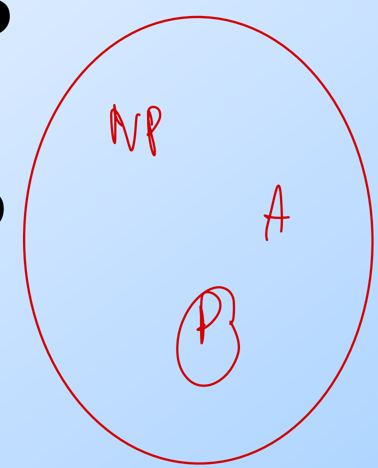
P Versus **NP**



- Originally a curiosity of Computer Science, mathematicians now recognize as one of the most important open problems the question **P** = **NP**?
- There are thousands of problems that are in **NP** but appear not to be in **P**.
- But no proof that they aren’t really in **P**.

ปัญหา A → NP-Complete ; ถ้าแปลงปัญหา NP (B) $\xrightarrow{\text{1เชิง}}$ A ได้ในเวลา Poly time

Complete Problems



- One way to address the **P** = **NP** question is to identify *complete problems* for NP.
- An *NP-complete problem* has the property that if it is in **P**, then every problem in **NP** is also in **P**.
- Defined formally via “polytime reductions.” Satisfiability, TSP(traveling salesman Problems)

Knapsack

NP-Hard.

ปัญหา A → NP-hard ; ถ้าสามารถแปลงปัญหา NP-Complete B → A ได้ใน poly time

Complete Problems – Intuition

NP-Hard

NP-Complete

NP

P

- A complete problem for a class embodies every problem in the class, even if it does not appear so.
- Compare: PCP embodies every TM computation, even though it does not appear to do so. Decision Problem
- Strange but true: Knapsack embodies every polytime NTM computation.

การลดรูป.

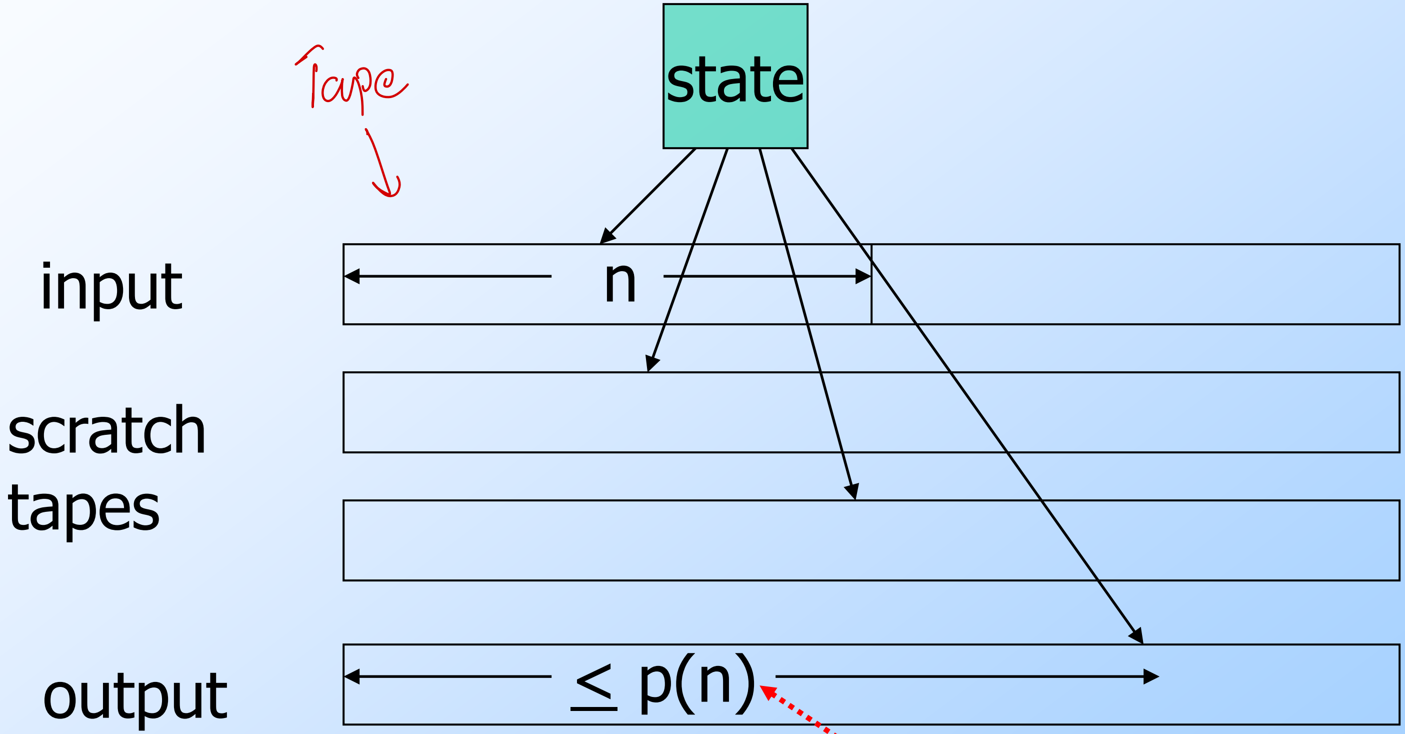
Polytime Reductions

- Goal: find a way to show problem L to be NP-complete by reducing every language/problem in **NP** to L in such a way that if we had a deterministic polytime algorithm for L, then we could construct a deterministic polytime algorithm for any problem in **NP**.

Polytime Reductions – (2)

- We need the notion of a *polytime transducer* – a TM that:
  1. Takes an input of length n.
  2. Operates deterministically for some polynomial time p(n).
  3. Produces an output on a separate *output tape*.
- Note: output length is at most p(n).

Polytime Transducer

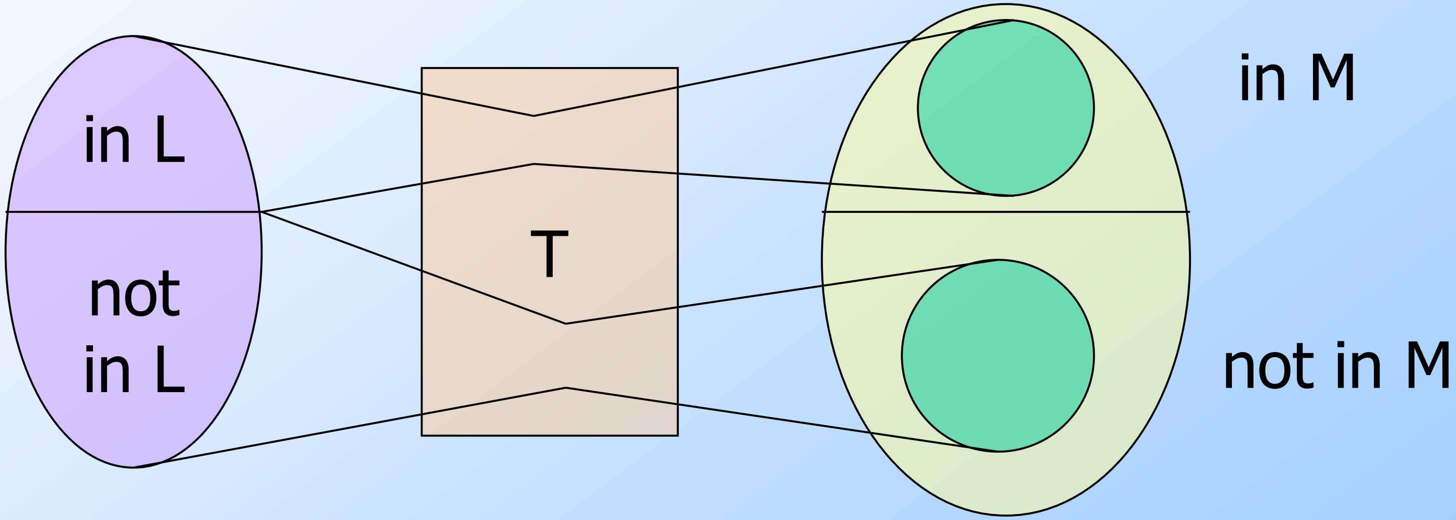


Remember: important requirement is that *time* ≤ p(n).

Polytime Reductions – (3)

- Let L and M be langauges.
- Say L is *polytime reducible* to M if there is a polytime transducer T such that for every input w to T, the output x = T(w) is in M if and only if w is in L.

Picture of Polytime Reduction



NP-Complete Problems

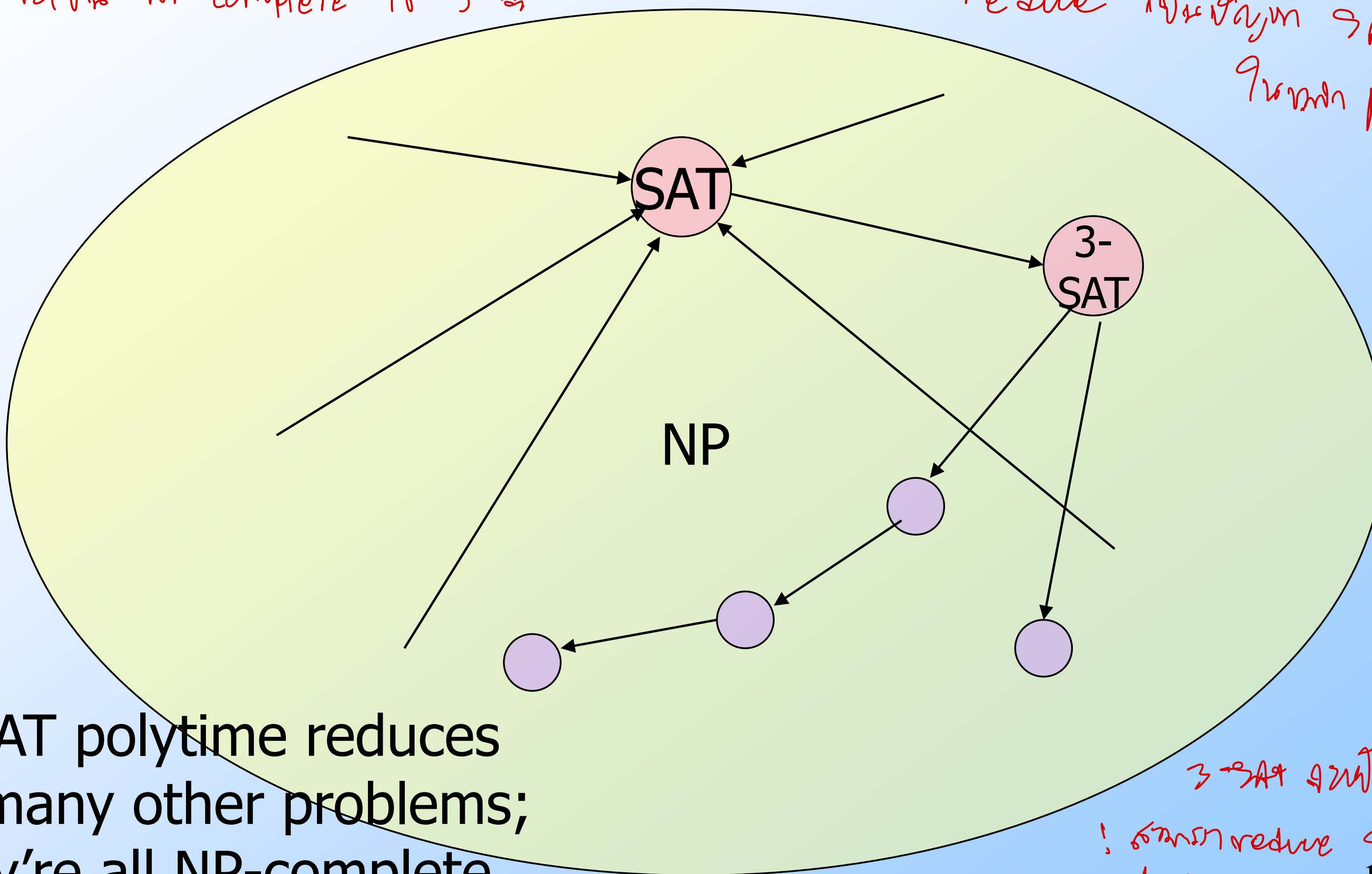
- A problem/language M is said to be *NP-complete* if for every language L in **NP**, there is a polytime reduction from L to M.
- Fundamental property: if M has a polytime algorithm, then L also has a polytime algorithm.
  - I.e., if M is in **P**, then every L in **NP** is also in **P**, or “**P** = **NP**.”

The Plan

All of **NP** polytime reduces to SAT, which is therefore NP-complete

SAT polytime reduces to 3-SAT

SAT จะเป็น NP-complete ได้ ; ปัญหาทุกปัญหาใน NP ต้องสามารถ Reduce เป็นปัญหา SAT ได้ ในเวลา poly time.



3-SAT polytime reduces to many other problems; they're all NP-complete

3-SAT จะเป็น NP-complete ; ต้องสามารถ reduce SAT → 3SAT ได้ในเวลา poly time.

The Satisfiability Problem

Cook's Theorem: An NP-Complete Problem

Restricted SAT: CSAT, 3SAT

Boolean Expressions

- Boolean, or propositional-logic expressions are built from variables and constants using the operators AND, OR, and NOT.
  - Constants are true and false, represented by 1 and 0, respectively.
  - We'll use concatenation (juxtaposition) for AND, + for OR, - for NOT, unlike the text.

Example: Boolean expression

OR NOT T

- (x+y)(-x + -y) is true only when variables x and y have opposite truth values.
- Note: parentheses can be used at will, and are needed to modify the precedence order NOT (highest), AND, OR.

The Satisfiability Problem (*SAT*)

- Study of boolean functions generally is concerned with the set of *truth assignments* (assignments of 0 or 1 to each of the variables) that make the function true.
- NP-completeness needs only a simpler question (SAT): does there exist a truth assignment making the function true?

Example: SAT

- $(x+y)$ and $(-x + -y)$ is satisfiable.
- There are, in fact, two satisfying truth assignments:
  1. x=0; y=1.
  2. x=1; y=0.
- x(-x) is not satisfiable. F

SAT as a Language/Problem

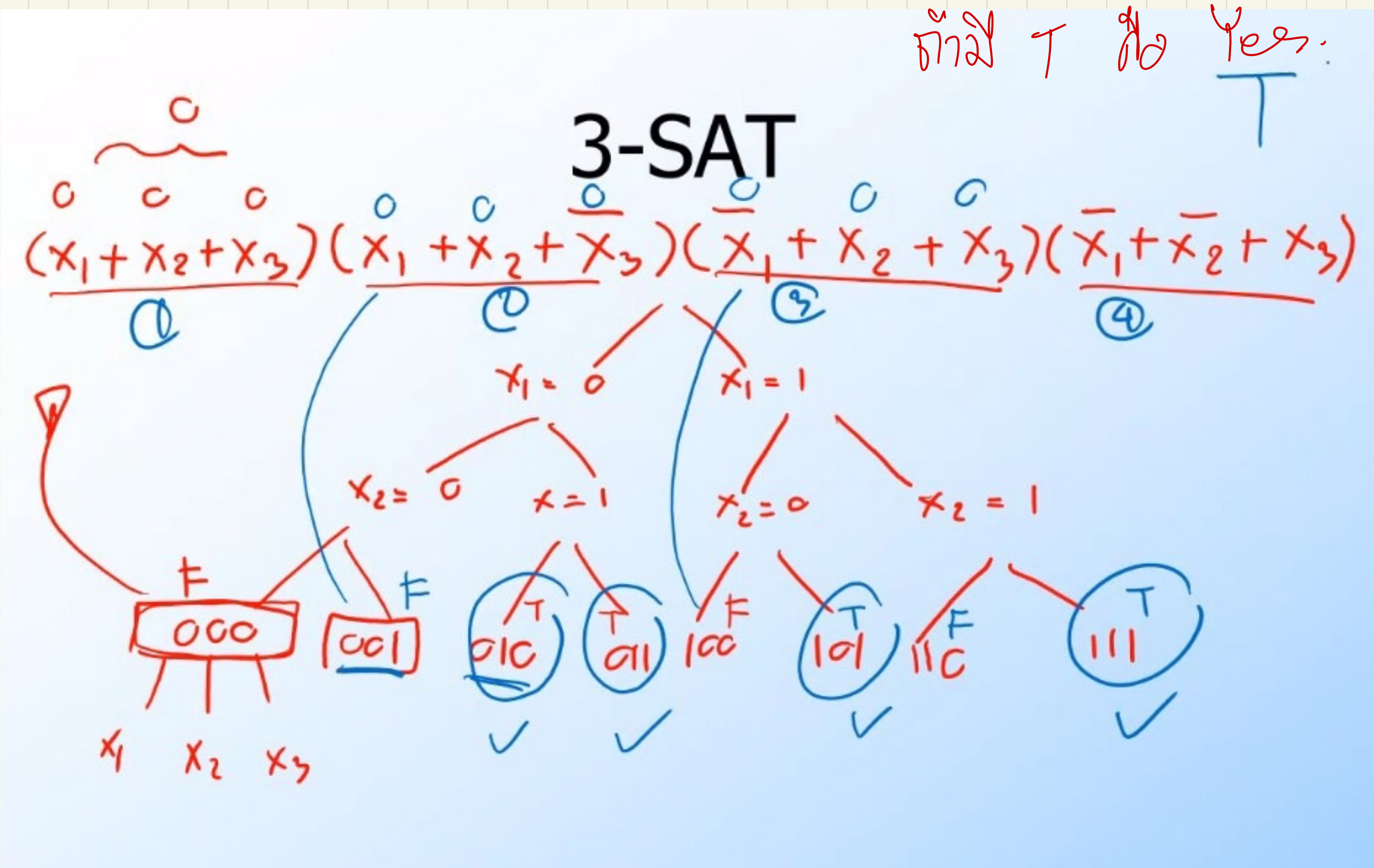
- An instance of SAT is a boolean function.
- Must be coded in a finite alphabet.
- Use special symbols (,), +, - as themselves.
- Represent the i-th variable by symbol x followed by integer i in binary.

Example: Encoding for SAT

- (x+y)(-x + -y) would be encoded by the string `(x1+x10)(-x1+-x10)`

SAT is in **NP**

- There is a multitape NTM that can decide if a Boolean formula of length n is satisfiable.
- The NTM takes $O(n^2)$ time along any path.
- Use nondeterminism to guess a truth assignment on a second tape.
- Replace all variables by guessed truth values.
- Evaluate the formula for this assignment.
- Accept if true.


ถ้ามี T คือ Yes.
3-SAT
(x1 + x2 + x3)(x1 + x2 + x̄3)(x̄1 + x2 + x3)(x̄1 + x̄2 + x3)
x1 = 0
x1 = 1
x2 = 0
x = 1
x2 = 0
x2 = 1
F
000
001
010
011
100
101
110
111
x1 x2 x3