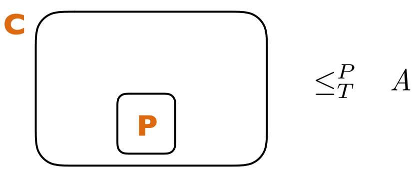
15-251
Great Ideas in
Theoretical Computer Science

Lecture 18:
NP-completeness continued

March 22nd, 2018

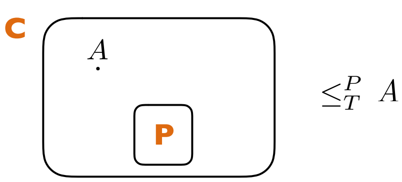
Summary so far

- How do you identify *intractable* problems?
  (problems not in **P**) e.g. *SAT, TSP, Subset-Sum, …*
- Poly-time reductions $A \leq_T^P B$ are useful to compare hardness of problems.
- Evidence for intractability of A:
  Show $L \leq_T^P A$, for **all** $L \in$ **C**, for a large class **C**.

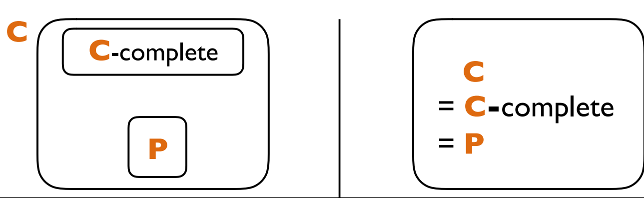


Summary so far

- Definitions of **C**-hard, **C**-complete.

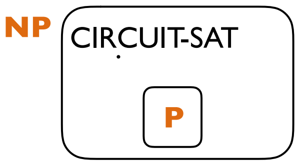


2 possible worlds



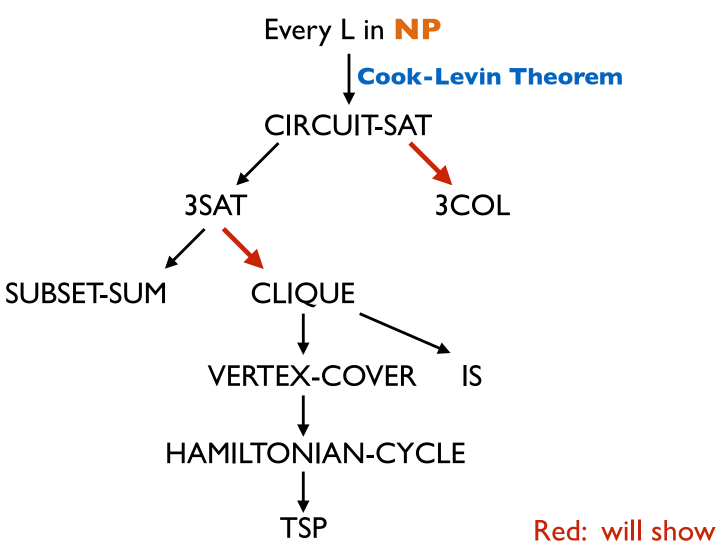
Summary so far

- The complexity class **NP** (take **C** = **NP**)
- **NP**-hardness, **NP**-completeness
- Cook-Levin Theorem: CIRCUIT-SAT is **NP**-complete



$\le_T^P$ CIRCUIT-SAT

- <u>Many</u> other languages are **NP**-complete.
- The **P** vs **NP** question

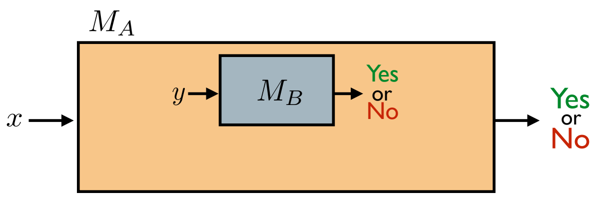


First:
An important note about reductions

Cook reduction

Cook reductions: poly-time Turing reductions

$$\boxed{A \leq_T^P B}$$



"You can solve A in poly-time using a blackbox that solves B."

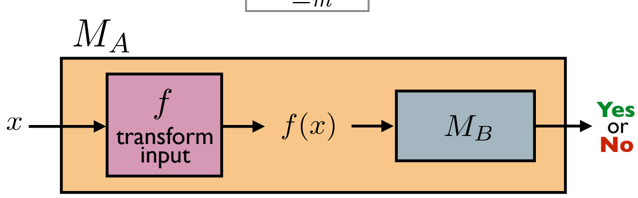
You can call the blackbox poly(|x|) times.

Karp reduction

NP-hardness is usually defined using Karp reductions.

Karp reduction (polynomial-time many-one reduction):

$$\boxed{A \leq_m^P B}$$



Make one call to M_B and directly use its answer as output.

<u>We must have:</u> $x \in A \implies f(x) \in B$

$x \notin A \implies f(x) \notin B$

Can define **NP**-hardness with respect to $\leq_T^P$.

(what some courses use for simplicity)

Can define **NP**-hardness with respect to $\leq_m^P$.

(what experts use)

These lead to different notions of **NP**-hardness.

3COL is NP-complete

3COL is **NP**-complete: High level steps

3COL is in **NP** (exercise).

We know CIRCUIT-SAT is **NP**-hard.
So it suffices to show CIRCUIT-SAT $\leq_m^P$ 3COL.

<u>We need to:</u>

CIRCUIT-SAT ≤ 3COL: The construction

<u>1. Define a map $f : \Sigma^* \rightarrow \Sigma^*$.</u>

If x is not $\langle C \rangle$ for a circuit C, map it to ϵ.

So assume x is a valid encoding of a circuit.

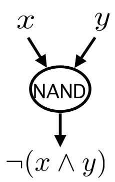
Circuit with AND, OR, NOT gates

↓

Circuit with only NAND gates
(in addition to input gates and constant gates)

CIRCUIT-SAT ≤ 3COL: The main gadget

Consider a NAND gate.



x and y represent some other gates.

$\neg(x \wedge y)$ becomes the input of another gate.

For each NAND gate, construct:

CIRCUIT-SAT ≤ 3COL: The main gadget

Claim:

A valid coloring of this "gadget" mimics the behaviour of the NAND gate.

Colors = {**0**, **1**, **n**}

WLOG:
- vertex **0** gets color **0**
- vertex **1** gets color **1**
- vertex **n** gets color **n**

CIRCUIT-SAT ≤ 3COL: The main gadget

A couple of observations:

Observation1:

vertices x, y

$x \wedge y$ and $\neg(x \wedge y)$

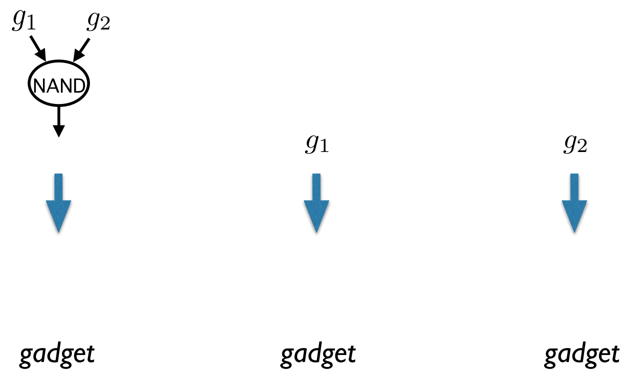
will not be assigned the color **n**.

Observation2:

$x \wedge y$ and $\neg(x \wedge y)$

will be assigned different colors.

CIRCUIT-SAT ≤ 3COL: Rest of construction



CIRCUIT-SAT ≤ 3COL: Why does it work?

Convince yourself that:

$w \in$ CIRCUIT-SAT $\implies f(w) \in$ 3COL

$w \notin$ CIRCUIT-SAT $\implies f(w) \notin$ 3COL

f is computable in polynomial time.

Poll

Which of the following are true?

- 3COL $\leq_m^P$ 2COL is known to be true.
- 3COL $\leq_m^P$ 2COL is known to be false.
- 3COL $\leq_m^P$ 2COL is open.
- 2COL $\leq_m^P$ 3COL is known to be true.
- 2COL $\leq_m^P$ 3COL is known to be false.
- 2COL $\leq_m^P$ 3COL is open.

CLIQUE is NP-complete

Want to show:

- CLIQUE is in **NP**.
- CLIQUE is **NP**-hard.

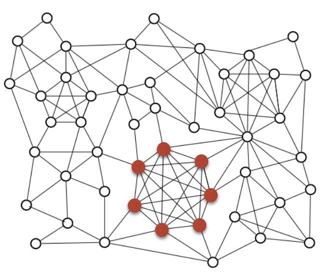
3SAT is **NP**-hard, so show 3SAT $\leq_m^P$ CLIQUE.

Definition of CLIQUE

CLIQUE

Input: $\langle G, c \rangle$ where G is a graph and c is a positive int.

Output: Yes iff G contains a clique of size c.



Definition of 3SAT

3SAT

Input: A Boolean formula in "conjunctive normal form" in which every clause has exactly 3 literals.

e.g.:

$$\underbrace{(x_1 \vee \neg x_2 \vee x_3)} \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

a clause (an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

Output: Yes iff the formula is satisfiable.

Aside: 3SAT is in **NP**

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

φ satisfiable

$\Longleftrightarrow$

can pick one literal from each clause and set them to True

$\Longleftrightarrow$

the sequence of literals picked does not contain both a variable and its negation.

What is a good proof that $\varphi \in$ 3SAT ?

CLIQUE is **NP**-complete: High level steps

CLIQUE is in **NP**. ✓

We know 3SAT is **NP**-hard.
So suffices to show 3SAT $\leq_m^P$ CLIQUE.

We need to:

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ 3SAT $\implies$ $f(w) \in$ CLIQUE
3. Show $w \notin$ 3SAT $\implies$ $f(w) \notin$ CLIQUE
4. Argue f is computable in polynomial time.

3SAT ≤ CLIQUE: Defining the map

1. Define a map $f : \Sigma^* \to \Sigma^*$.

not valid encoding of a 3SAT formula $\mapsto \epsilon$

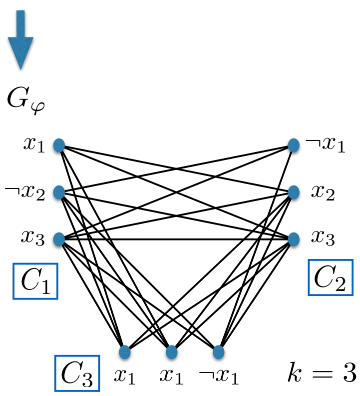
otherwise we have valid 3SAT formula φ (with m clauses).

$\varphi \mapsto \langle G, k \rangle$ (we set $k = m$)

Construction demonstrated with an example.

3SAT ≤ CLIQUE: Defining the map

$$\varphi = \underbrace{(x_1 \vee \neg x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(x_1 \vee x_1 \vee \neg x_1)}_{C_3}$$



3SAT ≤ CLIQUE: Why it works

If φ is satisfiable, then G_φ contains an m-clique:

φ is satisfiable $\Longrightarrow$

$\Longrightarrow$ G_φ contains an m-clique.

3SAT ≤ CLIQUE: Why it works

If G_φ contains an m-clique, then φ is satisfiable:

G_φ has a clique K of size m $\implies$

$\implies$ φ is satisfiable.

3SAT ≤ CLIQUE: Poly-time reduction?

Creation of G_φ is poly-time:

Creating the vertex set:

- there is just one vertex for each literal in each clause.
- scan input formula and create the vertex set.

Creating the edge set:

- there are at most O(m^2) possible edges.
- scan input formula to determine if an edge should be present.

□