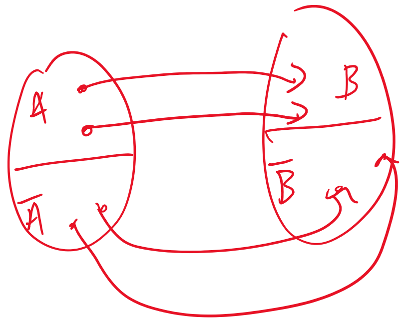
NP Completeness/Polynomial time reductions

05 April 2021 08:51

Def 7.28: $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if there is some poly time DTM M that takes input w, writes $f(w)$ on tape and halts.

Def 7.29: Language A is polynomial time reducible to language B, denoted $A \leq_p B$ if $\exists$ poly-time computable function f such that, $\forall w \in \Sigma^*$,

$$w \in A \iff f(w) \in B$$



f is called the polynomial time reduction from A to B

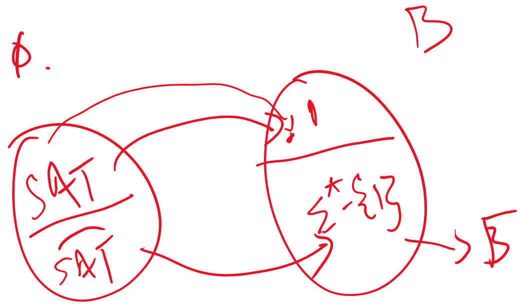
The Goal: A way to decide A efficiently

(1) $A \longrightarrow B$ (transform A to B)

(2) Decide B

We will show $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

It is important to have the restriction on the reduction. If not, we could test all the 2^n assignments possible, of a SAT instance ϕ.



$$f(\phi) = \begin{cases} 1 & \text{if satisfiable} \\ 0 & \text{otherwise} \end{cases}$$

We have an easy reduction from SAT to $\{1\}$ if we don't impose restrictions on the power of the reduction function.

<u>Theorem 7.31</u>: $A \leq_p B$ and $B \in P \Rightarrow A \in P$.

<u>Proof</u>: Suppose there is a poly time algorithm M for B. We have the following decider for A.

Alg for A: On input w $\rightarrow$ Poly time reduction from A to B.

$n = |w|$ $\quad n^{k_1}$ (1) Compute $f(w)$.

(2) Run M on $f(w)$. Accept iff M accepts $f(w)$. $\Rightarrow |f(w)| \leq n^{k_1}$

$\rightarrow n^{k_2}$

$n^{k_1} + |f(w)|^{k_2}$

Correctness: Easy

Time: (1) and (2) are both poly time. $\leq n^{k_1} + n^{k_1 k_2}$

$A \leq_p B,\ B \in P \Rightarrow A \in P$.

<u>Other results</u>

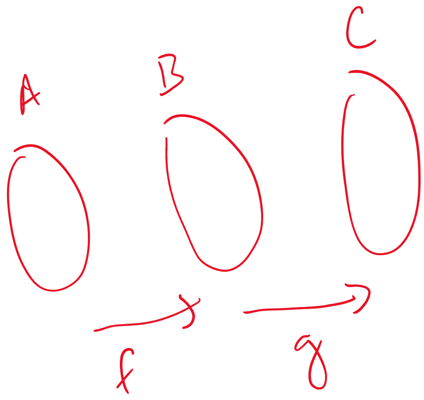
Other results

(1) $A \leq_p B$ and $B \in NP \implies A \in NP$

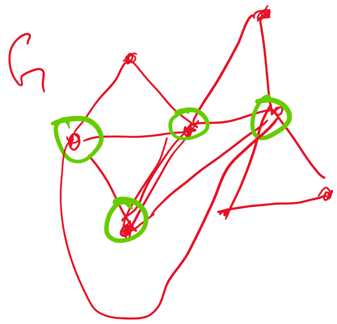
(2) $A \leq_p B$ and $A \notin P \implies B \notin P$

(3) $A \leq_p B$ and $B \leq_p C \implies A \leq_p C$

(4) $A \leq_p B \implies \overline{A} \leq_p \overline{B}$.



Theorem 7.32: 3-SAT $\leq_p$ CLIQUE.



3-SAT = $\{\langle\phi\rangle \mid \phi$ is a 3-CNF formula, ϕ is satisfiable$\}$

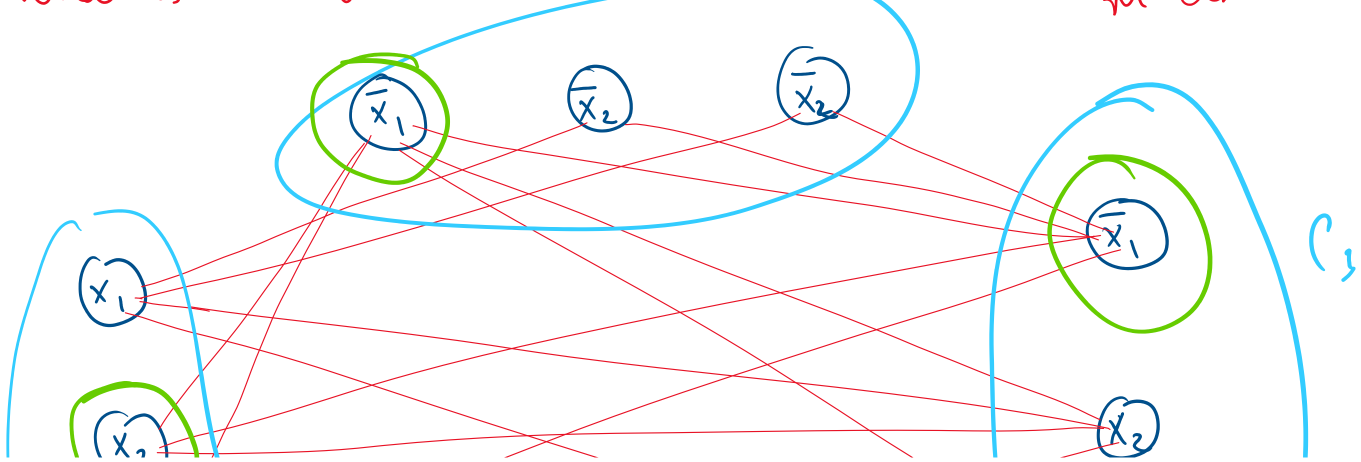
CLIQUE = $\{\langle G, k\rangle \mid G$ is an undirected graph with a k-clique$\}$

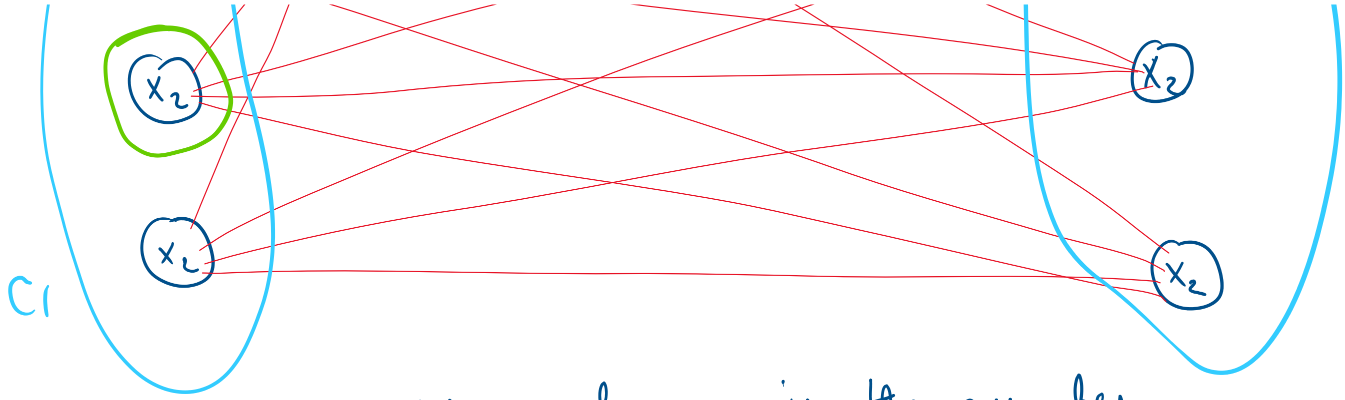
$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_k \vee b_k \vee c_k)$

Consider $\phi = (x_1 \vee x_2 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

What is the goal here?

ϕ = n vars
m clauses.





G has $3m$ vertices, where m is the number of clauses. G has 3 vertices for each clause in ϕ.

Edges: No edges between vertices of same clause.

No edge between x_i and $\bar{x}_i$, for any i.

This construction takes only m^2 time.

ϕ is satisfiable $\Leftrightarrow$ G has m clique

$\phi \in$ 3-SAT $\Leftrightarrow$ $\langle G, m \rangle \in$ CLIQUE.

($\Rightarrow$) $\phi \in$ 3-SAT $\Rightarrow$ $\exists$ an assignment which sets each clause to true

$\Rightarrow$ Every clause has at least one true literal

$\Rightarrow$ Choose one true literal from each clause. That is m

$\Rightarrow$ Choose one true literal
each clause. That is m
vertices.

$\Rightarrow$ These m vertices will be adjacent to each other.

(We cannot have both x_i & $\bar{x}_i$ set to true in the satisfying assignment)

Thus these m nodes form a clique.

So $\langle G, m \rangle \in$ CLIQUE.

($\Leftarrow$) G has m clique $\Rightarrow$ At most one vertex from each clause.

$\Rightarrow$ Exactly one vertex from each clause.

$\Rightarrow$ We cannot have x_i & $\bar{x}_i$ both in the clique (since there are no edges between them)

So we have m non-contradicting literals, one from each clause. We can assign true to all of them.

Each clause is true $\Rightarrow$ ϕ is satisfied.

It does not matter what we set to the unassigned variables.

Gadgets: Tools used to convert one problem into another. We will see more examples

---

NP- Completeness

Def 7.34: B is NP- complete if

(1) $B \in NP$ and

(2) $\forall A \in NP$, $A \leq_p B$.

We can think of NP- complete problems as the "hardest problems in NP".

Consequences: (1) Suppose B is an NP- complete language. If $B \in P$, then $\forall A \in NP$, we have $A \in P$. That is $P = NP$.

(2) If B is NP- complete, $C \in NP$ and $B \leq_p C$, then C is NP- complete.

By assumption, $C \in NP$. Condition (1)

$\forall A \in NP$, $\underbrace{A \leq_p B \text{ and } B \leq_p C}_{A \leq_p C}$

$A \leq_p C$. Condition (2).

Cook-Levin Theorem: SAT is NP-complete.