Lecture 19 - P vs. NP - Search Problems

Announcements:

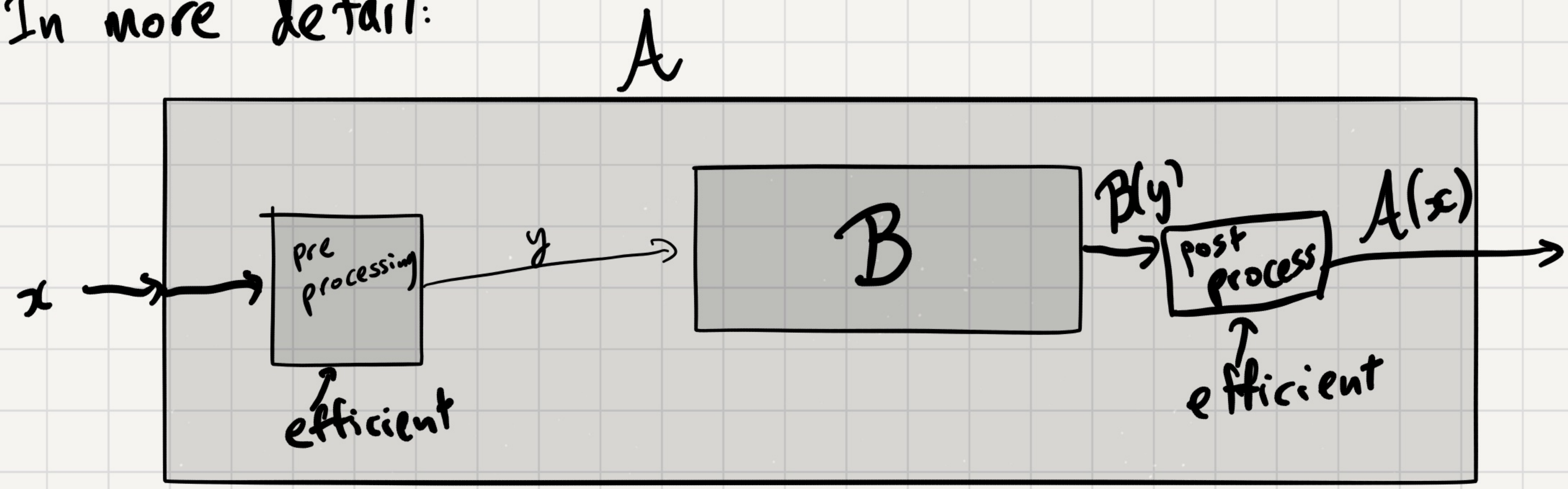
- Project will be released this week.
  start figuring out project teams. (up to 3 per team)
- New section: Mondays 2PM
  Annamira & Adnaan.

} see Piazza for more details

Recap: Reductions

"a problem A reduces to a problem B if any subroutine to solve B can be used to solve A"

In more detail:



(*) an efficient alg for B $\Rightarrow$ an efficient alg for A.

A reduction = pre-processing + post-processing.

(*) $\neg\exists$ an efficient alg for B $\Leftarrow$ $\neg\exists$ an efficient alg for A

$A \to B$

$A \leq B$

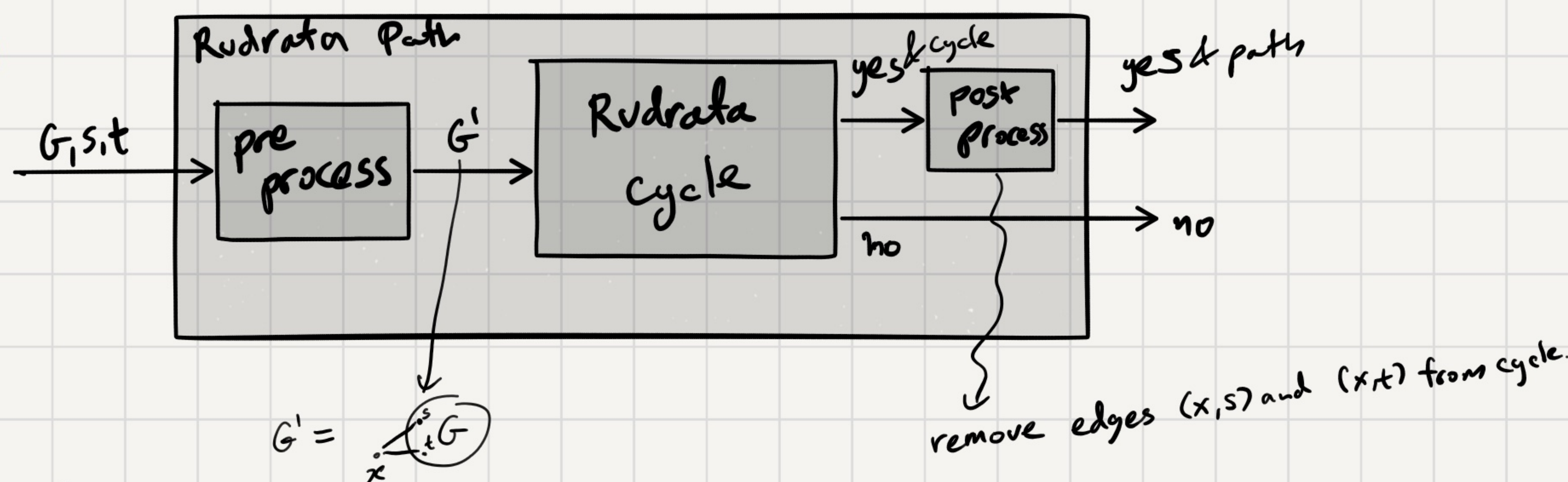
Recap: Rudrata Cycle & Rudrata Path

Rudrata Cycle Problem: Given an undirected graph $G=(V,E)$
Is there a cycle that visit each vertex exactly once?

a.k.a Hamiltonian Cycle

Rudrata st-path Problem: Given an undirected graph $G=(V,E)$, nodes s & t.
Is there a path from s to t that visits each vertex exactly once?

RudPath ≤ RudCycle



To prove correctness:

G has an s-t-Rudrata Path ⇒ G' has a Rudrata Cycle

G has no s-t-Rudrata Path ⇒ G' has no Rudrata Cycle

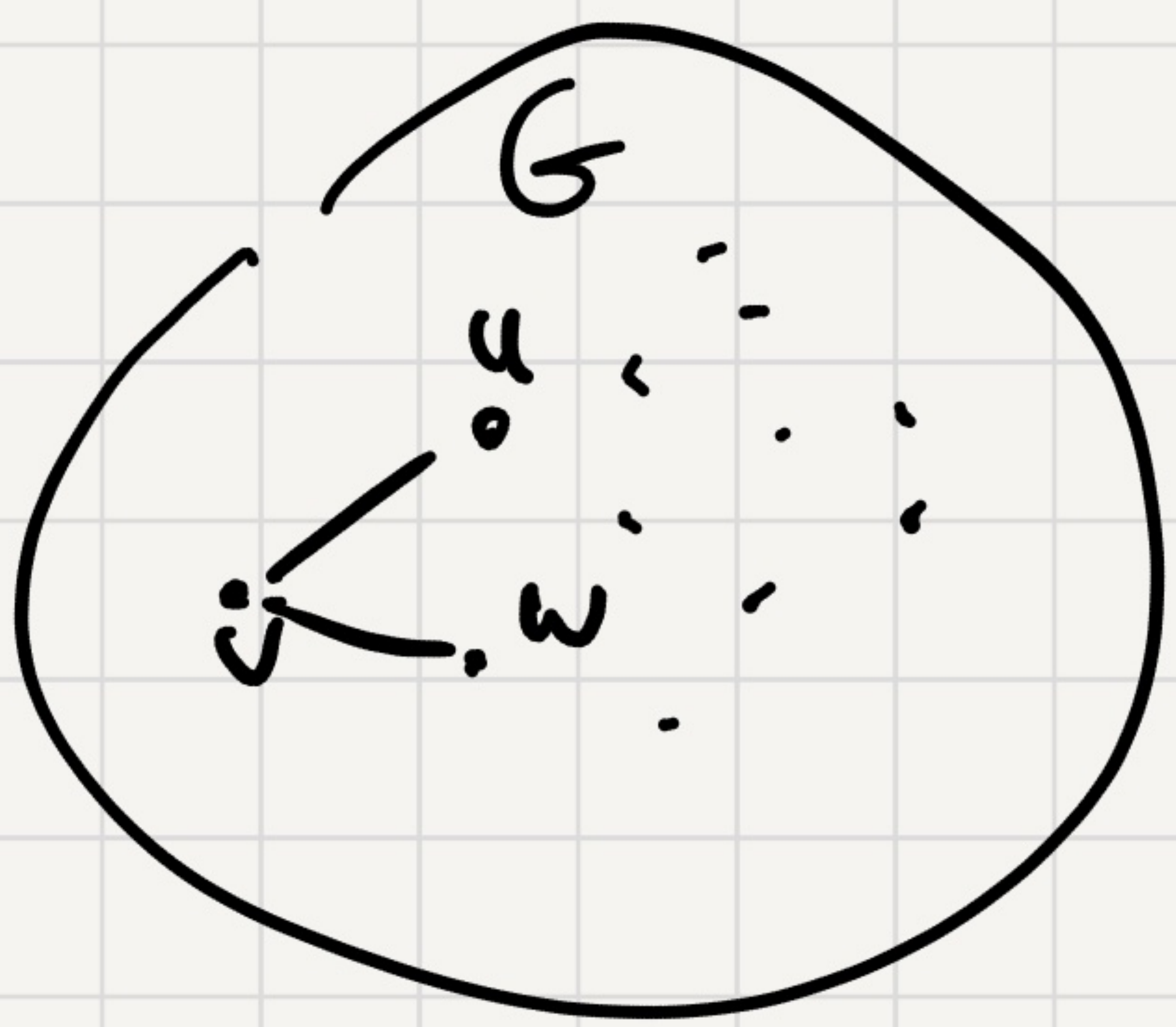
Exercise from last time

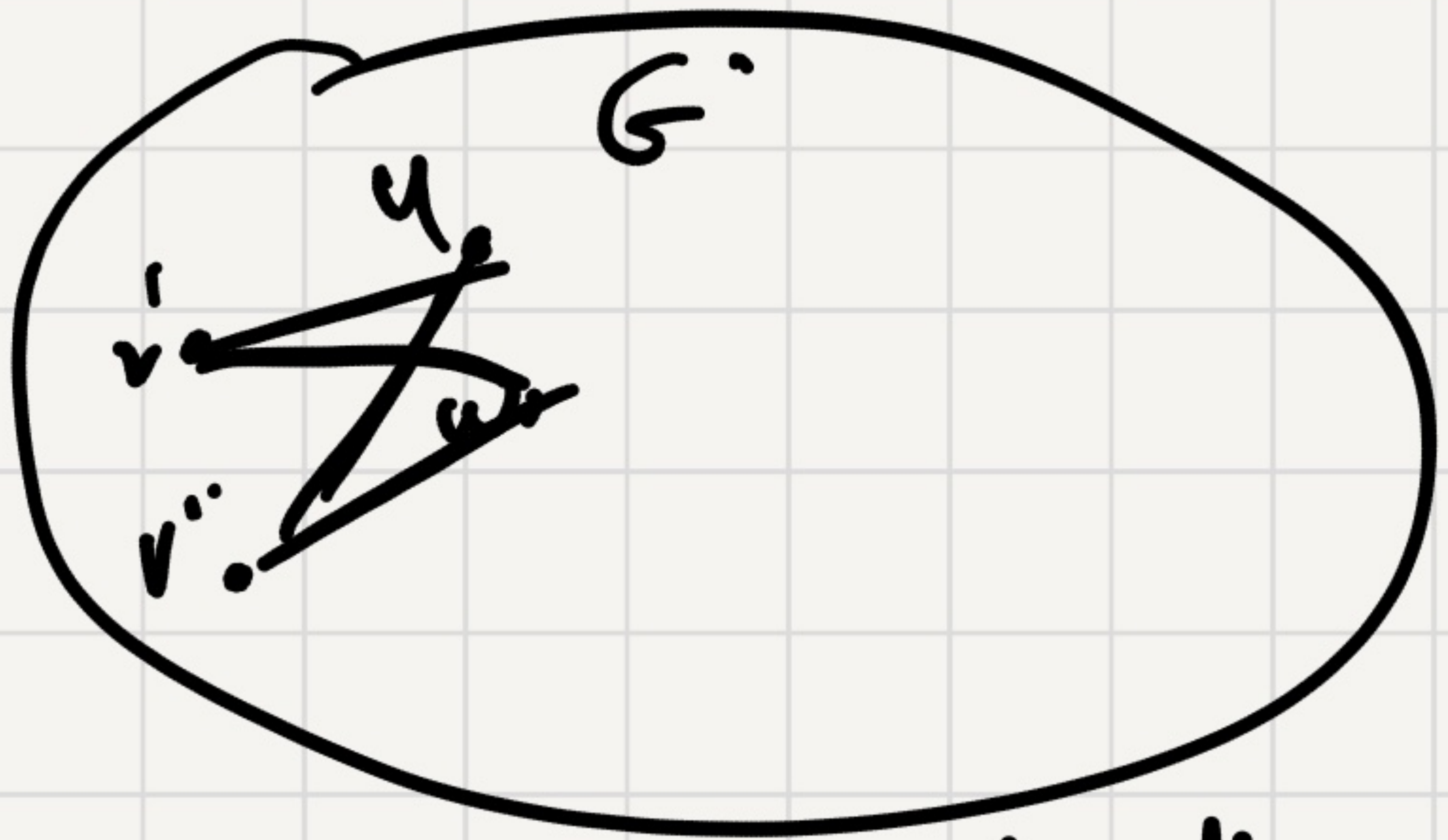
RudCycle $\leq$ RudPath?

Given G

Pick some vertex $v \Rightarrow$ duplicate it v', v''

and both v', v'' have same neighbors as v





Claim: G has RudCycle $\Leftrightarrow$ G' has a RudPath from v' to v''.

σ: v — . . . . . — v

$\Downarrow$ $\Uparrow$

σ' v' — . . . . . — v''

Search, Decision & Optimization Problems

So far we talked mainly about optimization problems.

For Example: 1. Find shortest path from s to t.

2. Find Best prefix-free Encoding.

3. Find Maximum Flow.

Search Problems:

Examples

1. Given G, s, t & Budget B, find a path of length $\leq B$ from s to t (if one exists)
2. Given $f_1, \dots, f_m$ & Budget B, find a prefix-free tree of cost $\leq B$ (if one exists)
3. Given G find a Rudrata Cycle (if one exists)

Decision Problems:

Given G, s, t & Budget B, is there a path of length $\leq B$ from s to t?

Search Problems

Def'n: A language L is a subset $\{0,1\}^*$.

Def'n: A binary relation R is a subset $R \subseteq \{0,1\}^* \times \{0,1\}^*$.

We say that a binary relation is "efficiently verifiable" if given (x,y) there exists an efficient algorithm that decides whether $(x,y) \overset{?}{\in} R$.

- $(x,y) \in R \Rightarrow |y| \le poly(|x|)$ (witnesses are $poly(|x|)$-length).
- Runtime of algorithm (verifier): at most $poly(|x|)$.

Example: $R = \{ (G, C) \mid$ G is an undirected graph, C is a cycle in G that visits every vertex once. $\}$

(G: input)

Def'n: $L(R) = \{ x : \exists y, (x,y) \in R \}$ Language assoc. with R.

Decide(R) = Given x decide whether $\exists y : (x,y) \in R$.

Search(R) = Given x find $y : (x,y) \in R$ (if one exists).

<u>Observe:</u> If R is "efficiently verifiable" then Decide(R) can be solved $2^{poly(|x|)}$ time.

<u>Proof:</u> Given x:

1. For every possible $y \in \{0,1\}^{poly(|x|)}$:

   check whether $(x,y) \overset{?}{\in} R$. (poly-time).

   $\xrightarrow{\text{yes}}$ accept $\quad \xrightarrow{\text{no}}$ continue.

2. Reject.

$$2^{poly(|x|)} \cdot poly(|x|) \text{ time.}$$

P, NP

$P = \{ L(R) \text{ s.t. Decide}(R) \text{ can be solved efficiently} \}$

$NP = \{ L(R) \text{ s.t. } R \text{ is efficiently verifiable} \}.$

non-deterministic

$R = \{ ((\underbrace{G, s, t, B}_{x}), \underbrace{P}_{y}) : P \text{ is a path from } s \text{ to } t \text{ with length} \leq B \}.$

Decide(R) efficiently? Yes, Dijkstra $L(R) \in P$.

$R = \{ (G, C) : C \text{ is a Rudrata Cycle in } G \}$

$L(R) \in NP \qquad L(R) \overset{?}{\in} P.$

$P \subseteq NP. \qquad L \qquad R_L = \{ (x, 1) : x \in L \}.$

Does $P = NP$?

NP Completeness

Def: A problem A is <u>NP-Hard</u> if $\forall B \in NP$

$$B \to A. \quad (B \le A)$$

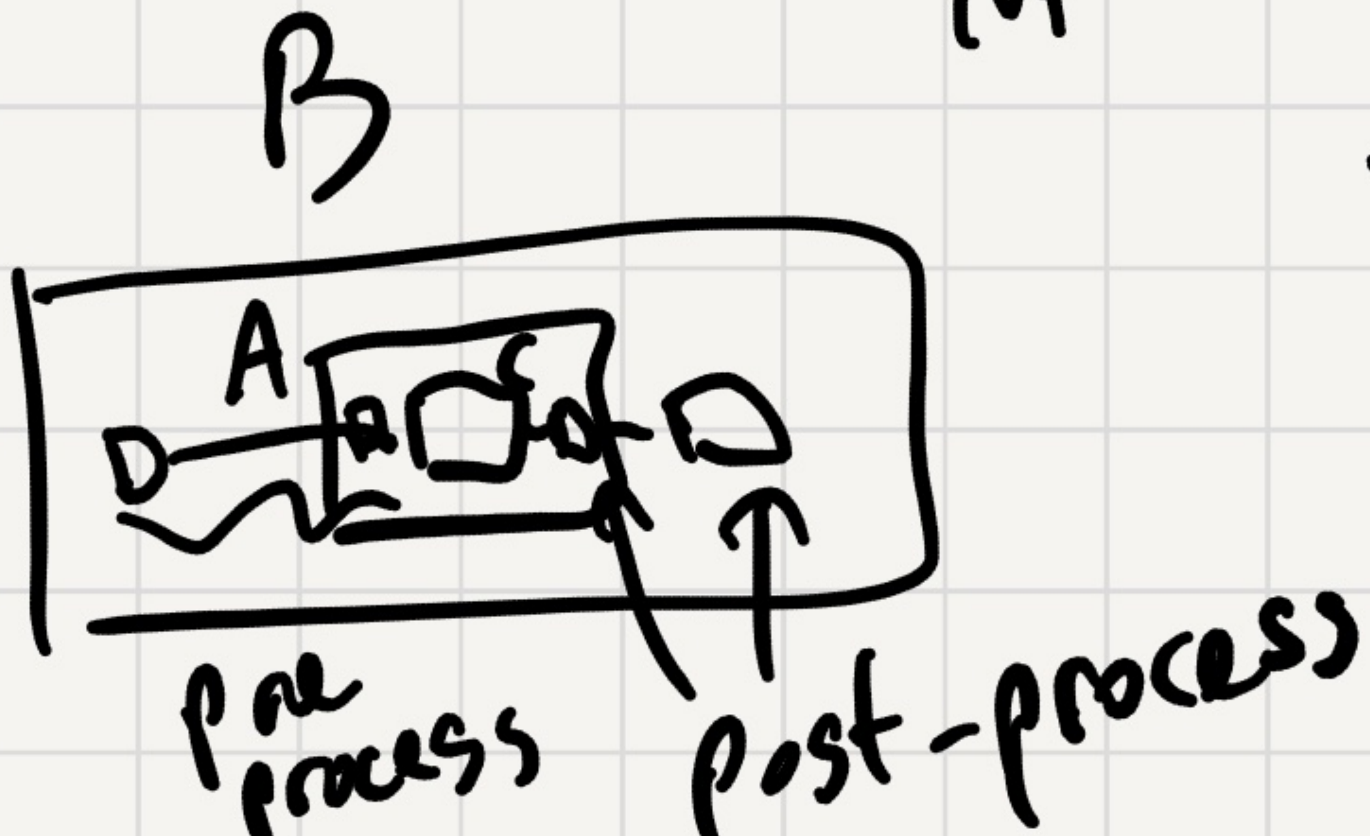
Def: A problem A is <u>NP-complete</u> if A is NP-hard & $A \in NP$

There exist NP-complete problems!

$$\left.\begin{aligned} &\underset{\substack{\in \\ NP}}{B} \to A \text{ is NPC} \\ &A \to C \end{aligned}\right\} \quad C \text{ is NPC.}$$

C is in NP

$$B \to A \to C$$
$$\Downarrow$$
$$B \longrightarrow C$$

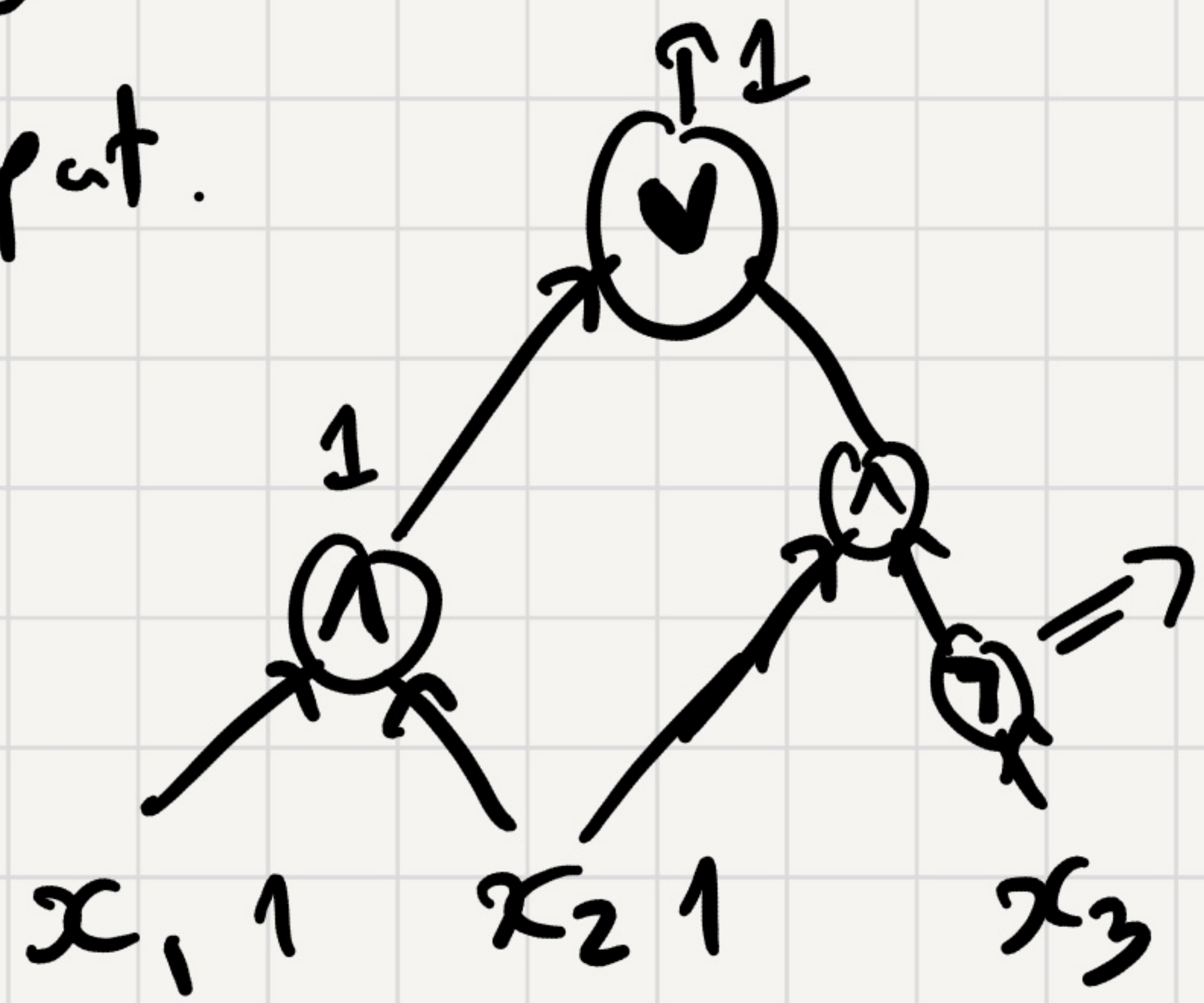


$B \leq A \leq C$

$\Rightarrow B \leq C.$

<u>Def'n:</u> A circuit is a directed acyclic graph with input nodes marked by $x_1, \dots, x_n$ & gates: OR gate, AND gate, NOT gate. 1 output.

<u>Example:</u>



$\Rightarrow$ negation

size = # of gates

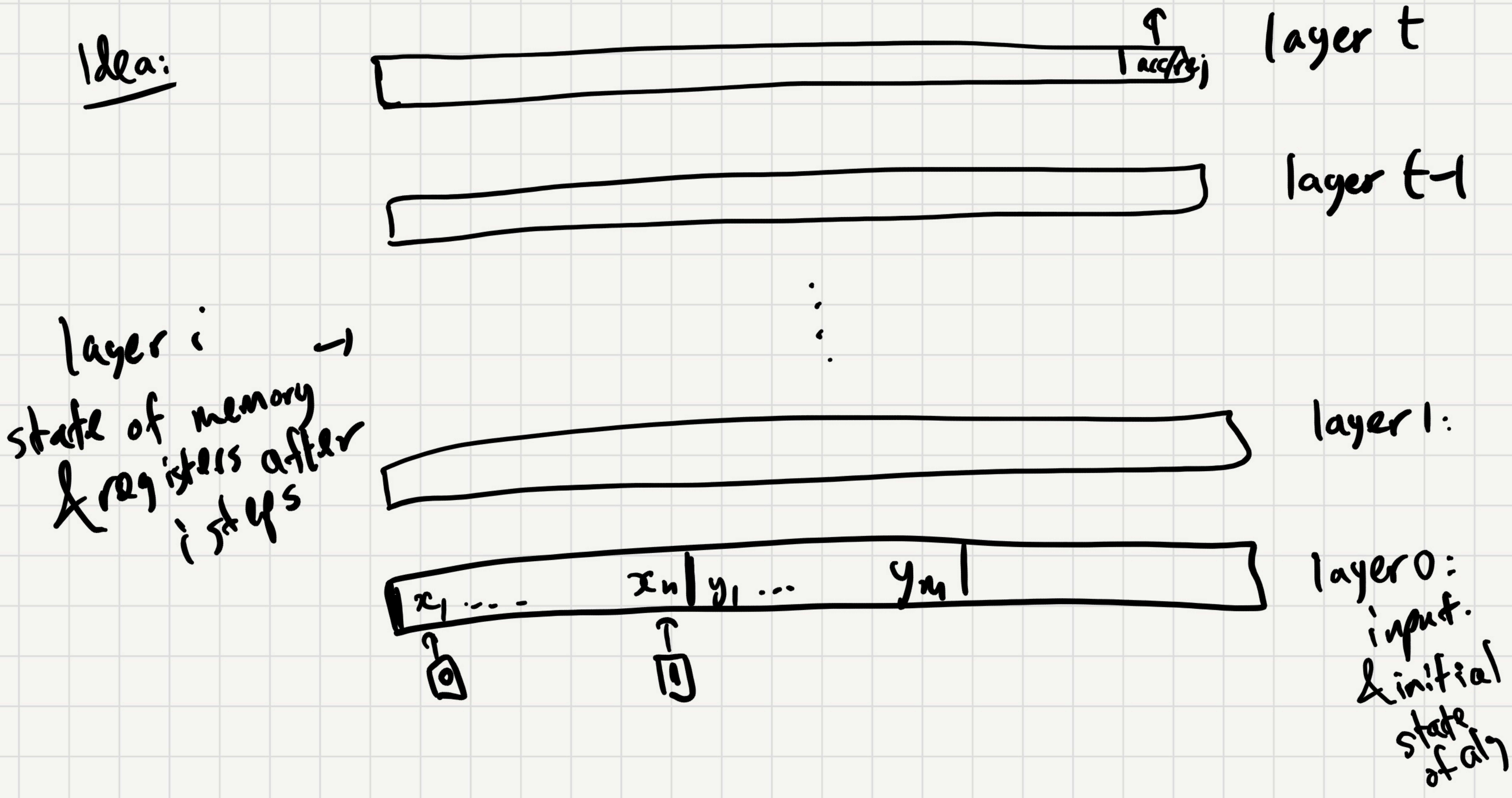
<u>Def'n:</u> CSAT (circuit satisfiability) Given circuit C on inputs $y_1, \dots, y_m$

Decide whether $\exists\, y \in \{0,1\}^m$ s.t. $C(y) = 1$.

"<u>Claim</u>": Suppose algorithm A runs on inputs of length n in time $\leq t$.

Then, there exists a circuit of size $O(t^2 \cdot n)$ that "simulates" A.

<u>Idea:</u>

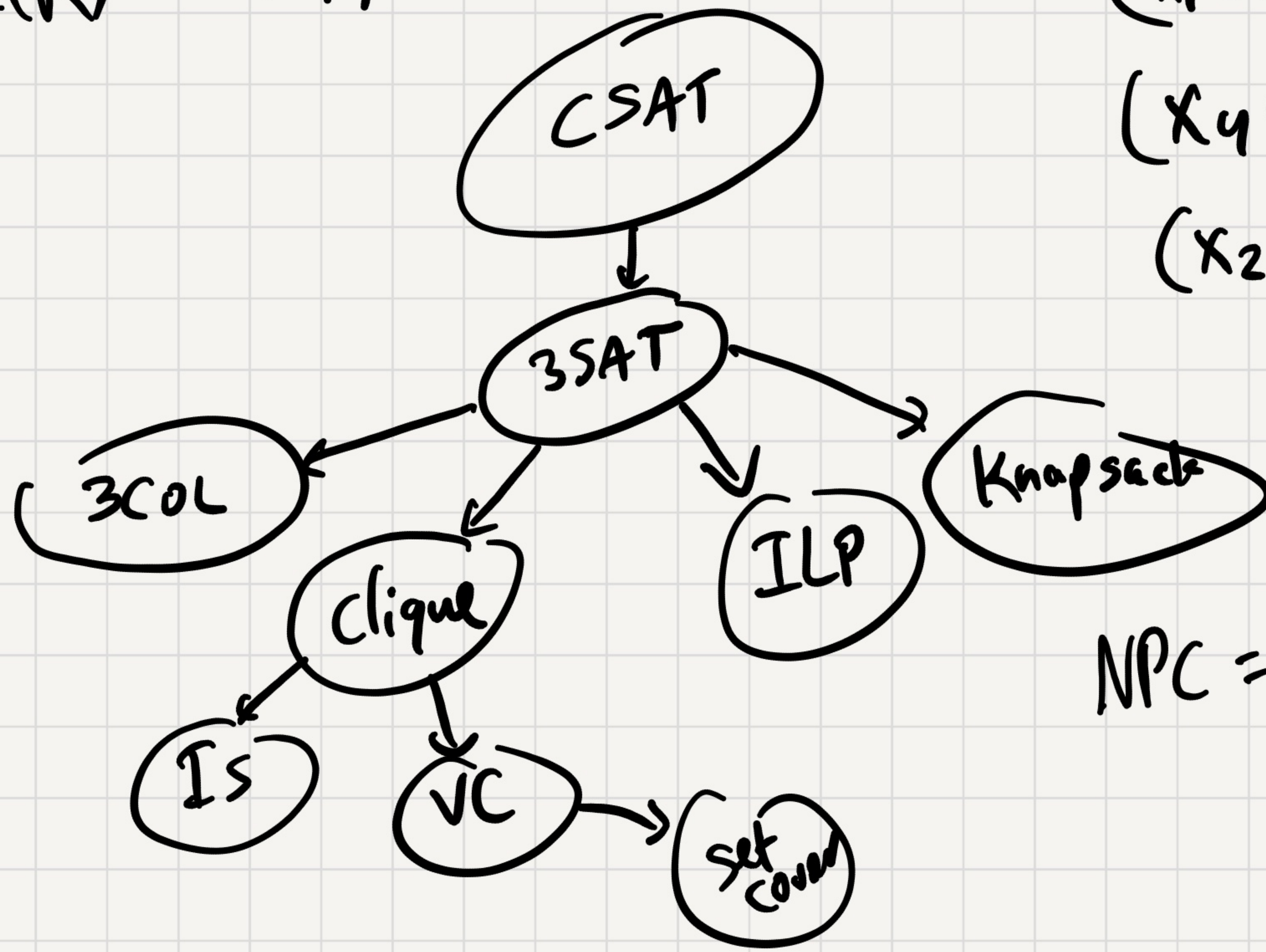


CSAT

NPC

$$R = \{(C, y) \; : \; C(y) = 1\}$$

$$L(R) = CSAT.$$



$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge$$
$$(x_4 \vee x_5 \vee x_7) \wedge$$
$$(x_2 \vee \overline{x_5} \vee \overline{x_8}) \wedge$$
$$\vdots$$

$$NPC = \underbrace{NPH} + NP.$$