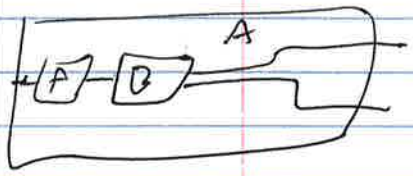
lastlw

Satisfiability

Lets review what we did last time. We began our discussion on NP-completeness. To prove ~~show~~ some problem B is NP-complete you should:

1. Prove $B \in NP$ by showing it is verifable in polynomial time.

2. prove B is NP-hard. How? $A \leq_p B$
   - Choose some known A which is NP-complete
   - give some reduction f computable in polytime such that $x \in A \Leftrightarrow f(x) \in B$. This can be done by showing $x \in A$ (x is good) $\Rightarrow f(x) \in B$ (f(x) is good) and $x \notin A$ (x is bad) $\Rightarrow f(x) \notin B$ (f(x) is bad.



In order to prove a problem is NP-complete, this depends on some other known NP-complete problem existing. Cook and independently Levin did this. They showed SAT is NP-complete without a reduction. That is, $\forall A \in NP$, $A \leq_p SAT$. Note, this is for every problem in NP. What is SAT?

a variable is one of $x_1, \ldots, x_2$
a literal is a variable or its negation $x_1, \ldots x_2, \neg x_1, \ldots \neg x_2$
a clause is an OR of several literals
a formula in CNF form is an AND of several clauses.

$(x) \wedge (\neg x)$ is unsatisfiable

might be.

$(x \vee y \vee z) \wedge (x \vee z \vee w) \wedge \ldots$

SAT is extremely universal. Most constraint problems can be rade to look like SAT. each clause is a constraint, every constraint must be satisfied, but they can be satisfied in a number of ways.

Lets say you have to feed everyone. I want either a burger or gyro or a cheeseburger. My buddy only wants a cheeseburger. Each of us is a constraint where b, g, c are variables if you order those or not. Our SAT formula is like $(b \vee g \vee c) \wedge (c)$

truth tables of AND, OR, NOT

The formula $(x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge \cdots \wedge (x_n \vee \neg y_n) \wedge (\neg x_n \vee y_n)$ is satisfiable only when $x_1 = y_1, \cdots, y_n = x_n$. A SAT formula for string equality!

To be clear, an assignment is a selection of variables $x_i \in \{0, 1\}$. a assignment is satisfying if under boolean arithmetic, the assignment equals 1

$$SAT = \{ \phi \mid \phi \text{ is a formula in CNF form and is satisfiable} \}.$$

Recall Cook and Levin proved $\forall L \in NP \; L \leq_p SAT$. So if $SAT \in P \Rightarrow NP \subseteq P \Rightarrow P = NP$. SAT is like an elected representative of the entire class of NP. This is also why we don't believe there exists a polynomial time algorithm for SAT.

$$KSAT = \{ \phi \mid \phi \text{ is a formula in kCNF, satisfiable, each clause has at most k literals} \}.$$

We prove 3SAT is NP-complete by reduction.
first we show $3SAT \in NP$. our witness is simply the assignment of variables for the problem instance. Substitution of these and computing can be done in polynomial time
$V(\phi, [1,0,\ldots])$ check if $\phi(1,0,\ldots) = 1$ or not.
(x_1 x_2 ...)

Now we prove $SAT \leq_p 3SAT$. for φ a general SAT formula, we convert it to a 3SAT instance such that
φ is satisfiable ($\varphi \in SAT$) $\Leftrightarrow$ $f(\varphi)$ is satisfiable ($f(\varphi) \in 3SAT$)
we describe our reduction f as follows. On input φ every SAT formula has some max clause size k. If $k \leq 3$ don't mess with φ, so $f(\varphi) = \varphi$. Clearly $\varphi \in SAT \Leftrightarrow \varphi \in 3SAT$.
Now suppose φ has max clause size $k > 3$. we convert a clause of size $k > 3$ to a pair of clauses, one of size $k-1$, and the other of size 3. we add variable z as follows

$$(x_1 \vee x_2 \vee \ldots \vee x_{k-2} \vee x_{k-1} \vee x_k) =$$

$$(x_1 \vee x_2 \vee \ldots \vee x_{k-2} \vee z) \wedge (x_{k-1} \vee x_k \vee \bar{z})$$

where each x_i is a literal. Note if the k clause is true, atleast one of the literals is true, so there is a selection of z to make the two clauses true. If the k clause is always false, the two clauses are also always false for any selection of z. Note it is important this conversion does not change the satisfiability of φ. repeat this process, adding dummy variables until φ only has clauses of size ≤ 3.

- Note since this doesn't alter satisfiability, $\varphi \in SAT \Leftrightarrow f(\varphi) \in 3SAT$.
- This reduction f occurs in polynomial time.

We conclude $SAT \leq_p 3SAT$ and so 3SAT is NP-complete.

□

Note that since Cook-Levin showed us $\forall L \in NP,\ L \leq_p SAT$, and we showed $3SAT \in NP,\ SAT \leq_p 3SAT$. This implies $\forall L \in NP\ \ L \leq_p 3SAT$. 3SAT is NP-complete without having to repeat the entire ~~reduction~~ proof. A simple reduction suffices. It is possible to repeat this reduction for 4SAT, 5SAT, ... kSAT for any $k \geq 3$. ~~what~~

What about 2SAT? Actually 2SAT $\in P$ (!!!!!) so if ~~2SAT~~ SAT $\leq_p$ 2SAT, SAT $\in P$ and $NP = P$ something we dont believe should happen.

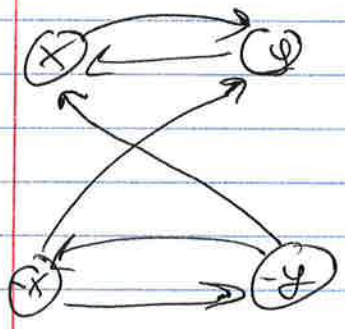
STOC 2021 incident $3SAT \leq_p 2SAT$

Recall $(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$. So every ~~clause~~ clause of size two ~~is~~ is an implication

$(a \vee b)$	$\neg a \Rightarrow b$
$(\neg a \vee b)$	$a \Rightarrow b$
$(a \vee \neg b)$	$\neg a \Rightarrow \neg b$
$(\neg a \vee \neg b)$	$a \Rightarrow \neg b$.

Create a graph, ~~two~~ two vertices for each literal, ~~one~~ two edges for each clause. if $(a \vee b)$ a clause, add edge $\neg a \to b$, $\neg b \to$ ~~b~~ a. recall implication is transitive, and a formula is unsatisfiable if $\forall x$ $x \Rightarrow \ldots \Rightarrow \neg x$. or $\neg x \Rightarrow \ldots \Rightarrow x$, so $\exists$ a path in our graph from x to $\neg x$ or from $\neg x$ to x.

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



$\neg x \to y$ if $x = 0 \overset{(3)}{\Rightarrow} y = 1$

$\neg y \to x$ $y = 1 \Rightarrow \neg y = 0$

$\neg x \to \neg y$ $\neg y = 0 \overset{(2)}{\Rightarrow} x = 1$

$y \to x$

$x \to y$

$\neg y \to \neg x$

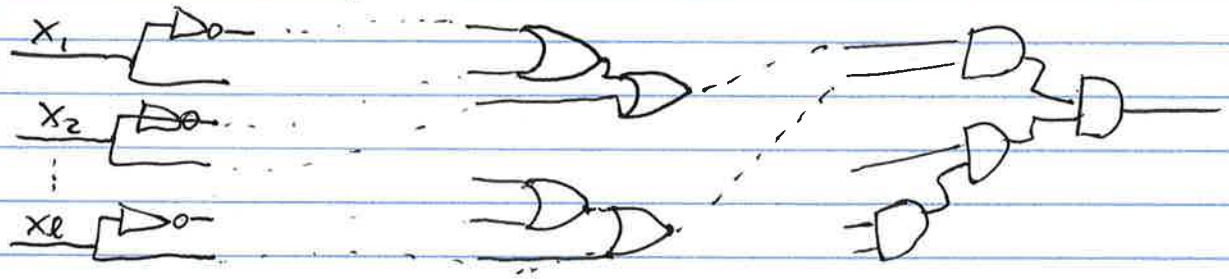
$\{0,1\}^n \to \{0,1\}$

circuitSAT = $\{\langle C \rangle \mid C$ a boolean circuit with AND/OR/NOT gates $\exists$ input to bring output to 1$\}$

we prove circuitSAT is NP-complete.

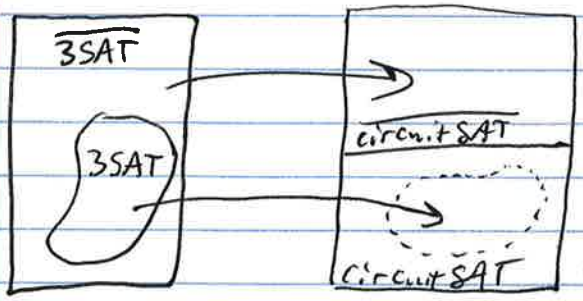
- first we show circuitSAT $\in$ NP verifier V takes as input $\langle C \rangle$ and a witness of n bits, and runs $\langle C \rangle$ on the inputs in the size of the input, obviously polytime. (increasing depth of circuit increases input)

- Now we show 3SAT $\leq_p$ circuitSAT.
let φ be a 3SAT formula. we create a boolean circuit for variables $x_1 \cdots x_\ell$, add input wires $x_1 \cdots x_\ell$. for any not-ed literals, add one not gate on the next layer. For each clause, add a subcircuit OR-ing the appropriate three. Then add an "AND" tree to AND the clauses toger



- if $\varphi \in$ 3SAT, this circuit $C = f(\varphi) \in$ circuitSAT
- if $\varphi \notin$ 3SAT this circuit is also unsatisfiable.
- computing f, construction of this circuit obviously takes polynomial time.

we conclude 3SAT $\leq_p$ circuitSAT so circuitSAT is NP-complete.



although the reduction is correct, the complexity of boolean circuits seems far greater and less restrictive. Nevertheless, if circuitSAT $\in$ P, it would solve many greater things, but it would also accidentally solve 3SAT for us.