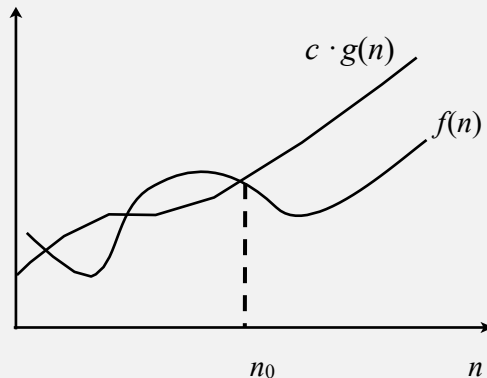
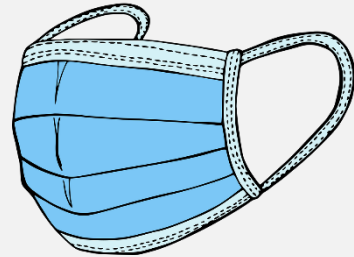
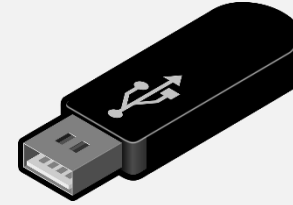
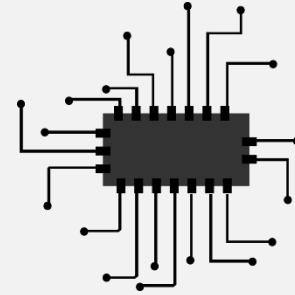
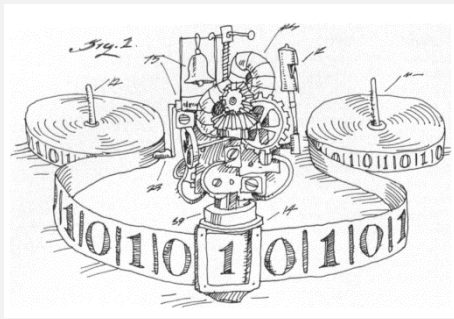


COMP 3602

Intro to Theory of Computation

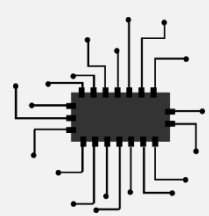
Lecture 17

Announcements



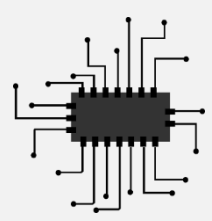
- Computability drill due Mar 12
 - One more drill (on complexity) after that
- Two more tests left:
 - Oral exam on NP-completeness: Mon Mar 23?
 - Practice problems are posted
 - Written exam on all the material we covered in the course.
 - Mar 27th?
- Presentations in class:
 - Mar 30 and Apr 1
 - All slides due Sun Mar 29th

Questions/comments?



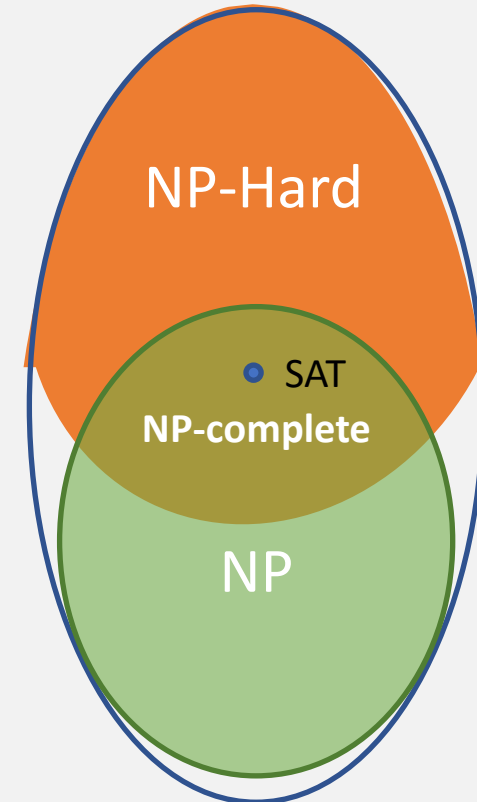
First NP-complete problem: SAT

- Propositional formula (recursive definition):
 - variables $x_1 \dots x_n$, x, y, z , etc are formulas
 - If φ, ψ are formulas, then so are $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$
- A truth assignment is an assignment of values TRUE/FALSE to variables
 - equivalently, 1 for TRUE, 0 for FALSE.
 - An assignment is satisfying if it makes a formula true.
 - A formula is satisfiable if there is a satisfying assignment for it.
- SAT: $\{\varphi \mid \varphi \text{ is a satisfiable propositional formula}\}$
 - Alternatively, as a computational problem:
 - Input: a propositional formula φ
 - Output: accept iff φ is satisfiable



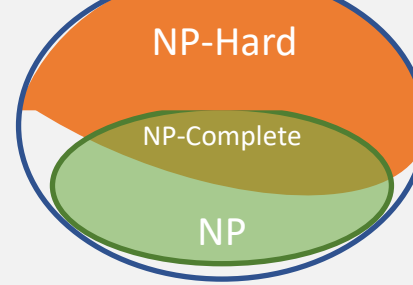
Cook-Levin theorem

- SAT: $\{\varphi \mid \varphi \text{ is a satisfiable propositional formula}\}$
- $NP = \bigcup_{d \in \mathbb{N}} NTime(n^d)$
- Theorem: SAT is NP-complete.
 - SAT is in NP: can evaluate a formula in polytime on a given assignment
 - Let's show hardness: that for every $L \in NP$, $L \leq_p SAT$
 - This is where (non-deterministic) Turing machines really shine!
 - Main idea: make $Check_A$ a propositional formula.
 - The size of it will depend on the number of steps M takes on w .
 - Here is where the polynomial time bound comes in.



Lemma: SAT is NP-hard.

- Show $\forall L \in NP, L \leq_p SAT$
- Let M_L be a non-deterministic TM deciding L in time $< cn^d$ for some constants c, d , where n is the input length
 - To simplify the proof, let's use one-way infinite tape
- Consider a computation of M_L on some input w of length n .
 - Any configuration of M_L on w has length at most cn^d
 - There are at most cn^d configurations on any branch of computation.
 - Write a (branch of) computation of M_L as a table
 - Called *tableau of computation*
 - Where a tape cell (mostly) corresponds to a column
 - Every cell of the tableau can be computed from only the three cells above, above-right and above-left.



Eg: NTM M checks if w contains 1.

- $(q_0, 0) \rightarrow (q_0, 0, R)$
- $(q_0, 1) \rightarrow (q_0, 1, R)$
- $(q_0, 1) \rightarrow (q_a, 1, R)$
- $(q_0, _) \rightarrow (q_r, _, R)$

Let $w=011$

Lemma: SAT is NP-hard.

- Show $\forall L \in NP, L \leq_p SAT$

- Consider a $cn^d \times cn^d$ tableau of computation of M_L on w , $|w|=n$

- Write a propositional $Check_A$ formula:

- Variables: $x_{i,j,s}$, true if cell (i,j) contains symbol s , for $1 \leq i,j \leq cn^d, s \in \Gamma \cup Q$
- Formula will be an AND of several parts: $\varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{cell} \wedge \varphi_{move}$
- Want: a satisfying assignment to the formula encoding an accepting tableau of M_L on w

1. Check that the first line in the tableau encodes initial configuration followed by blanks (φ_{start})

$$\varphi_{start} = x_{1,1,q_0} \wedge x_{1,2,w_1} \wedge \dots \wedge x_{1,n+1,w_n} \wedge x_{1,n+2,-} \dots \wedge x_{1,cn^d,-}$$

2. Check that there is an accept state (φ_{accept})

$$\varphi_{accept} = \bigvee_{1 \leq i,j \leq cn^d} x_{i,j,q_a}$$

3. Ensure exactly one symbol is in each cell (φ_{cell})

4. Check that all transitions are correct (φ_{move})

- The value in a cell (i,j) can be computed just from cells $(i-1,j-1), (i-1,j), (i-1,j+1)$
 - Called the “window”
- To check if each step was legal, check that every window is legal.
 - By doing “or” of all possibilities to see if they are correct (large but finite for each window)

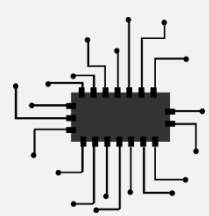
Eg: NTM M checks if w contains 1.

- $(q_0, 0) \rightarrow (q_0, 0, R)$
- $(q_0, 1) \rightarrow (q_0, 1, R)$
- $(q_0, 1) \rightarrow (q_a, 1, R)$
- $(q_0, -) \rightarrow (q_r, -, R)$

Let $w=011$

q_0	0	1	1	-
0	q_0	1	1	-
0	1	q_0	1	-
0	1	1	q_a	-
0	1	1	q_a	-

The resulting $\varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{cell} \wedge \varphi_{move}$ is satisfiable iff there exists an accepting tableau of M_L on w



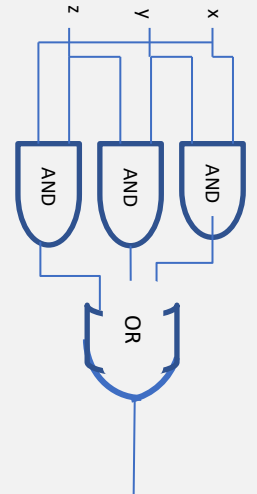
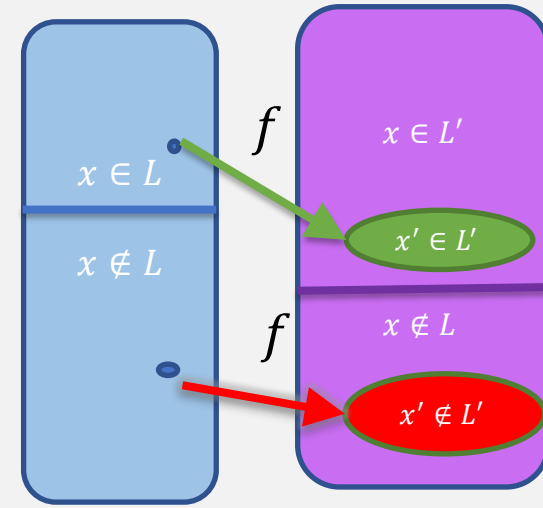
Variants of SAT

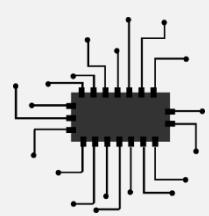
- Last time: **SAT is NP-complete**
 - In fact, with a little bit more work you can show that this proof gives us that **CNF-SAT** is NP-complete: $\{\varphi \mid \varphi \text{ is a satisfiable CNF formula}\}$
- For some classes of formulas, satisfiability problem is easy! The following are in P:
 - **2SAT** = $\{\varphi \mid \varphi \text{ is a satisfiable 2CNF formula}\}$
 - **HornSAT** = $\{\varphi \mid \varphi \text{ is a satisfiable CNF formula with at most one positive literal per clause}\}$
 - **XOR-SAT** = $\{\varphi \mid \varphi \text{ is a satisfiable AND of XORs of literals}\}$
- Other restrictions of SAT are as hard as the general case.
 - **3SAT** = $\{\varphi \mid \varphi \text{ is a satisfiable 3CNF formula}\}$ is NP-complete.
 - **Exact3SAT** = $\{\varphi \mid \varphi \text{ is a satisfiable 3CNF formula with exactly 3 literals per clause}\}$ is NP-complete.
 - Exercise: show that $3SAT \leq_p \text{Exact3SAT}$

- $3SAT = \{\varphi \mid \varphi \text{ is a satisfiable 3CNF formula}\}$
- Recursive definition of a propositional formula:
 - A variable or a constant TRUE/FALSE is a propositional formula
 - If A, B are propositional formulas, then so are $\neg A, A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B$

Theorem: 3SAT is NP-complete.

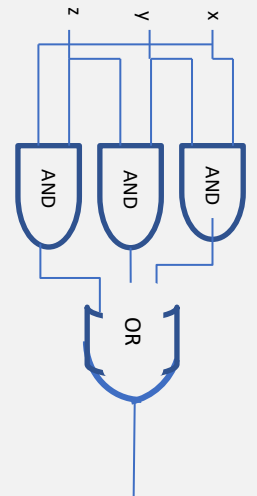
- First, 3SAT is in NP: given a potential satisfying assignment y for a formula φ , can evaluate $\varphi(y)$ in polynomial time.
- Now, to prove that 3SAT is NP-hard, let's show $SAT \leq_p 3SAT$
 - Construct a polytime-computable function f that transforms an arbitrary propositional formula φ into a 3CNF formula φ' so that φ is satisfiable iff φ' is satisfiable.
 - $f(\varphi) = \varphi'$ such that $\forall \varphi \quad \varphi \in SAT \leftrightarrow f(\varphi) \in 3SAT$
 - Idea: use the recursive definition of propositional formulas
 - For each connective, make a 3CNF formula stating that its value is the respective function of its parts (that it “computes correctly”)
 - The whole formula is an AND of these for all connectives
 - AND with “the outermost connective evaluates to true”
 - This is called Tseitin transformation
 - Also works for Boolean circuits! Widely used in hardware verification.





Tseitin transformation.

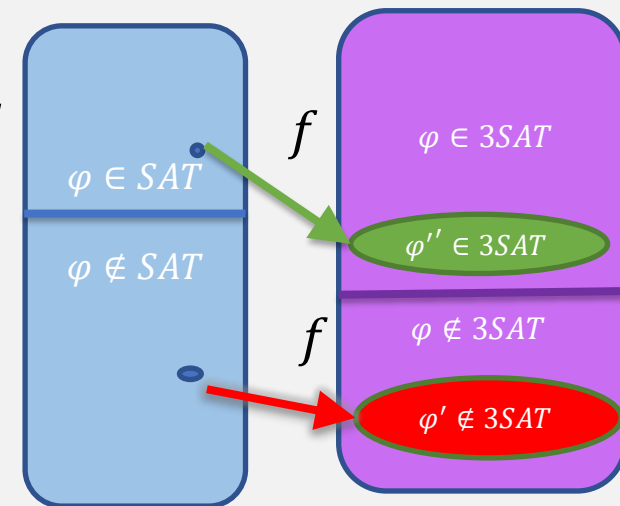
- Recursive definition of a propositional formula:
 - *A variable or a constant; if A, B are propositional formulas, then so are $\neg A, A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B$*
- Tseitin transformation:
 - A polytime-computable function f that transforms an arbitrary propositional formula φ into a 3CNF formula φ' so that φ is satisfiable iff φ' is satisfiable, defined as follows:
 - For every connective introduce a variable coding its value
 - Eg, for each subformula C which is not a variable or a constant, introduce a variable z_C
 - For each variable/constant, leave them as coding their own value.
 - Make a 3CNF formula φ_C stating that z_C is computed correctly
 - Eg if C is of the form $A \rightarrow B$, with z_A, z_B coding values of A, B then want $(z_A \rightarrow z_B) \leftrightarrow z_C$
 - Converting $(z_A \rightarrow z_B) \leftrightarrow z_C$ to CNF, get $\varphi_C = (\neg z_A \vee z_B \vee \neg z_C) \wedge (z_A \vee z_C) \wedge (\neg z_B \vee z_C)$
 - Finally, state that the outermost connective has value true.
 - Eg if C was an outermost connective, add $\wedge z_C$.
 - The full formula φ' is $z_{C_{out}} \wedge \bigwedge_C \varphi_C$



- To prove that 3SAT is NP-hard, let's show $SAT \leq_p 3SAT$

- Construct a polytime-computable function f that transforms an arbitrary propositional formula φ into a 3CNF formula φ' so that φ is satisfiable iff φ' is satisfiable.

- Let $f(\varphi) = \varphi'$ where φ' is a Tseitin transformation of φ



- Correctness proof:

- $\Rightarrow$ If $a_1 \dots a_n$ was a satisfying assignment for φ , a satisfying assignment for φ' is $a_1 \dots a_n$ together with correct values for all z_C
- $\Leftarrow$ A satisfying assignment to φ' contains in particular a satisfying assignment to variables of φ : this is the assignment to its first n variables.

- Finally, the construction is polytime: recursively unwind φ from the outermost connective.

Proving NP-completeness: template.

Goal: show that a language L' is NP-complete.

1. Show that $L' \in NP$

- Explain what potential solutions are and how to check them efficiently.
- That is, explain what y and R are in $L = \{x | \exists y |y| < poly(|x|) \wedge R(x, y)\}$,
- and why y is short and R is polytime computable.

2. Show that L' is NP-hard

- Pick a problem L we have showed to be NP-complete already.

- Show $L \leq_p L'$

a) Construct a polytime reduction f from L to L' ,

- that is, describe how to get $x' = f(x)$ for each x

» **Important: f cannot determine whether $x \in L$!**

b) Show correctness: two directions.

i. $\Rightarrow$ Given a solution y to x , construct a solution y' to $f(x)$

ii. $\Leftarrow$ Given a solution y' to $f(x)$, reconstruct a solution y to x

» **This applies only to $x' = f(x)$, not arbitrary element of L' !**

c) Explain briefly why f is polynomial-time computable

