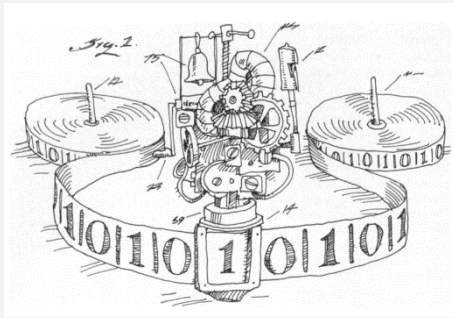
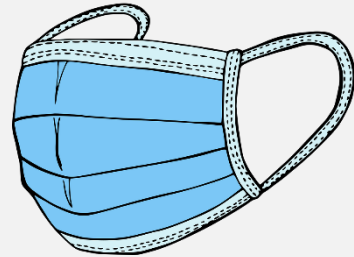
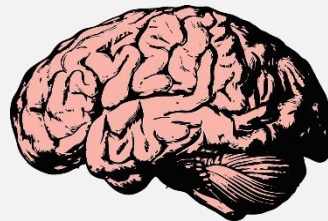
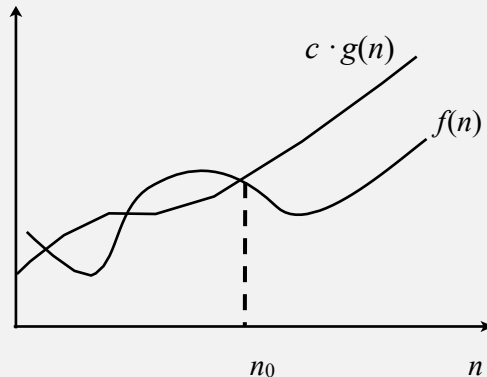


COMP 3602

Intro to Theory of Computation



Lecture 17

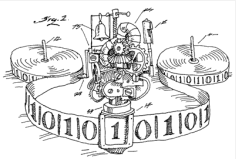


Announcements



- Computability drill due Mar 12
 - One more drill (on complexity) after that
- Two more tests left:
 - Oral exam on NP-completeness: Fri Mar 20?
 - 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?



Scaling down to complexity

“efficient”, “small”, “easy” $\equiv$ polynomial

- Decidable
 - Turing-recognizable
- For some decidable R
- $$L = \{x \mid \exists y R(x, y)\}$$

– P

– NP

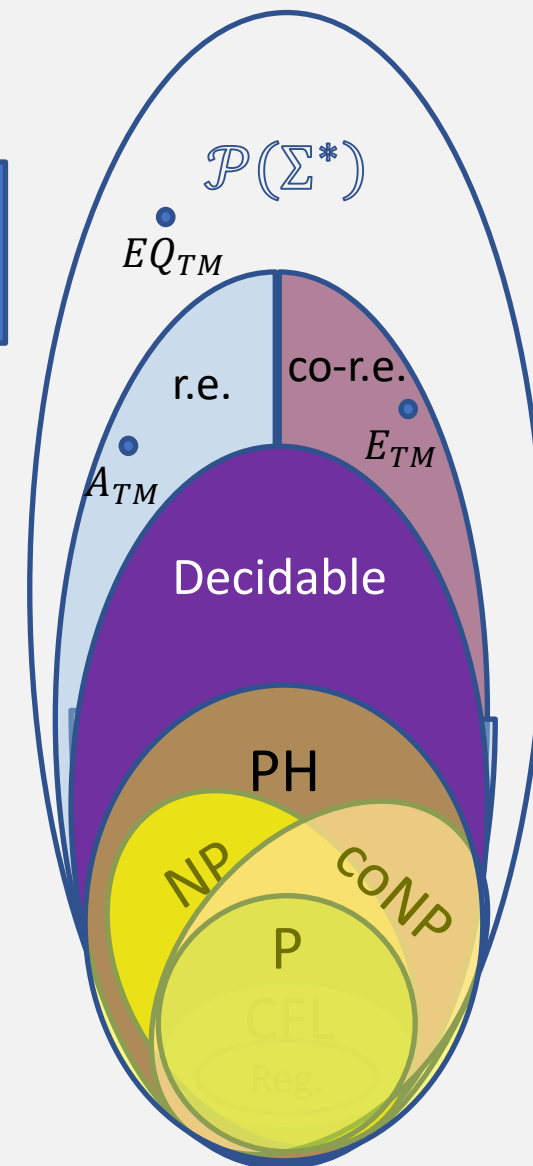
For some **polytime** decidable R ,
some constants c, d

$$L = \{x \mid \exists y \mid y \mid \leq cn^d \wedge R(x, y)\}$$

- Co-r.e
- Arithmetic hierarchy

– CoNP

– Polytime hierarchy (PH)





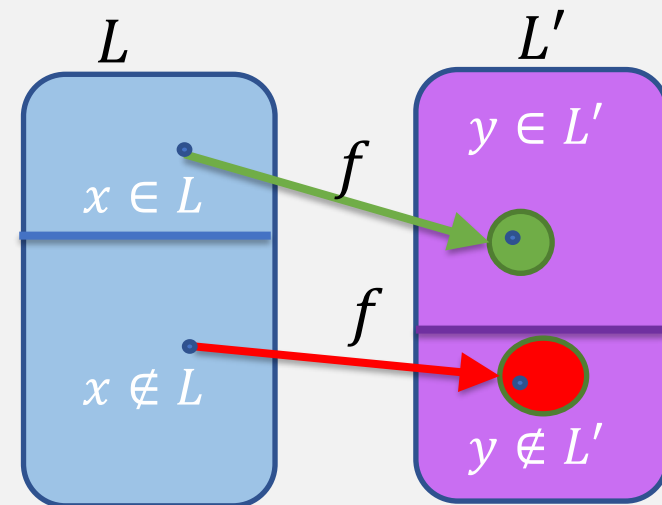
Polytime reductions

- Recall: L is **mapping-reducible** to L' , written $L \leq_m L'$ iff there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that $\forall x \in \Sigma^*, x \in L \Leftrightarrow f(x) \in L'$
 - But computable functions can just solve NP, EXP and so on problems!
 - Need a weaker reduction than the languages we want to compare.

Definition: L is **polytime mapping-reducible** to L' , written $L \leq_p L'$ iff there exists a **polytime** computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that

$$\forall x \in \Sigma^*, x \in L \Leftrightarrow f(x) \in L'$$

- If $L \leq_p L'$, then L is at most as hard as L'
 - $\leq_p$ is “hardness greater than”
 - If $L \leq_p L'$, and L is “hard”, then L' must be “hard”
- If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$
- If $L_1 \leq_p L_2$, then $\overline{L_1} \leq_p \overline{L_2}$





Clique and Independent set

- Let $G = (V, E)$ be an undirected graph.
- A “clique” is a complete subgraph of G
 - $S \subseteq V$ and $\forall u \in V \forall v \in V (u \in S \wedge v \in S \rightarrow E(u, v))$
- An “independent set” is a set of vertices with no edges between any pair
 - $S \subseteq V$ and $\forall u \in V \forall v \in V (u \in S \wedge v \in S \rightarrow \neg E(u, v))$
- CLIQUE: $\{\langle G, k \rangle \mid \text{graph } G \text{ has a clique } S \text{ st } |S| \geq k\}$
 - $\{\langle G, k \rangle \mid \exists S \subseteq V |S| \geq k \wedge \forall u \in S \forall v \in S E(u, v)\}$
- IndSet: $\{\langle G, k \rangle \mid \text{graph } G \text{ has an independent set } S \text{ st } |S| \geq k\}$
 - $\{\langle G, k \rangle \mid \exists S \subseteq V |S| \geq k \wedge \forall u \in S \forall v \in S \neg E(u, v)\}$
- Both CLIQUE and IndSet are in NP
 - Can check that S is a clique (independent set) in time $O(|S|^2)$
 - If have random access to E , otherwise at most $O(|E|^2 |S|^2)$

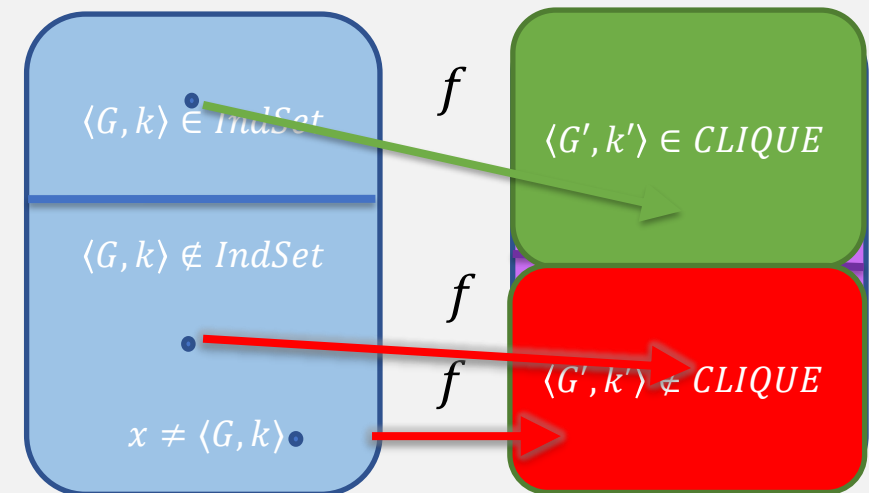


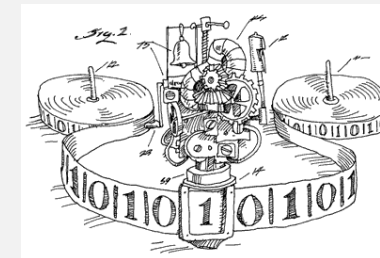
$\text{IndSet} \leq_p \text{CLIQUE}$

- **CLIQUE**: $\{\langle G, k \rangle \mid \text{graph } G \text{ has a clique } S \text{ such that } |S| \geq k\}$
 - $\{\langle G, k \rangle \mid \exists S \subseteq V \ |S| \geq k \wedge \forall u \in S \forall v \in S \ E(u, v)\}$
- **IndSet**: $\{\langle G, k \rangle \mid \text{graph } G \text{ has an independent set } S \text{ st } |S| \geq k\}$
 - $\{\langle G, k \rangle \mid \exists S \subseteq V \ |S| \geq k \wedge \forall u \in S \forall v \in S \ \neg E(u, v)\}$
- **Claim**: $\text{IndSet} \leq_p \text{CLIQUE}$
 - Construct a polytime computable function f
$$f(x) = \begin{cases} \langle G', k' \rangle & \text{if } x = \langle G, k \rangle \\ x & \text{otherwise} \end{cases}$$
 - Where $k' = k$ and $G' = (V, \bar{E})$
 - That is, G' is a complement of G .

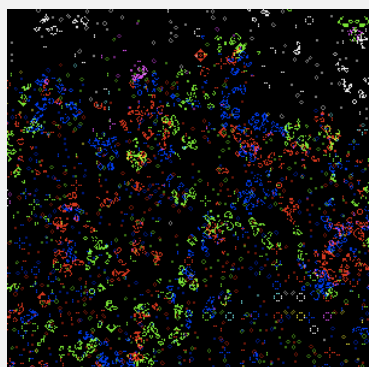
We will often ignore the “garbage” case and just say what f does on correctly formed instances

- Write eg $f(\langle G, k \rangle) = \langle G', k' \rangle$ and say what $\langle G', k' \rangle$ is in terms of $\langle G, k \rangle$





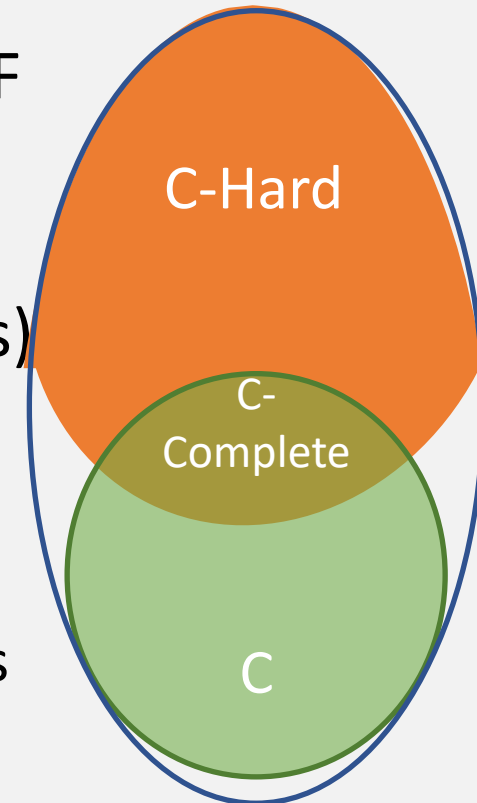
Some languages are as hard as anything in their complexity class!





Hardness and completeness

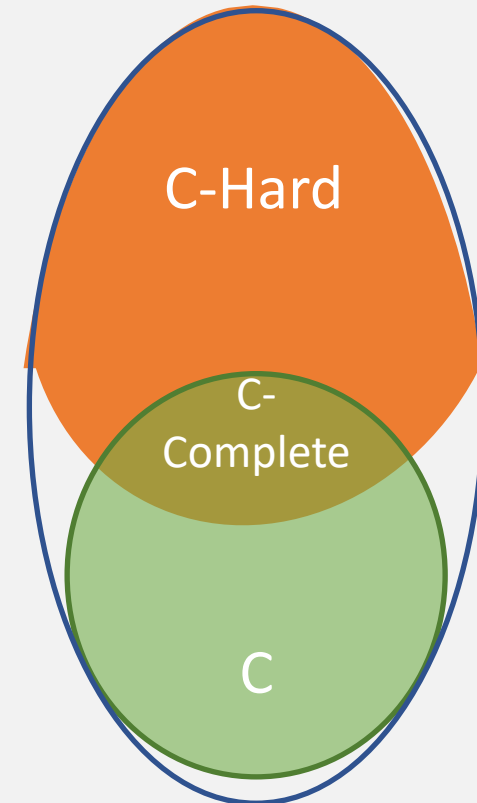
- *Definition:* A language L' is **hard** for a class C (*C-hard*) under F-reductions iff for every $L \in C$, there is a function f_L computable in F such that $\forall x \in \Sigma^* x \in L \leftrightarrow f_L(x) \in L'$
 - That is, for all $L \in C$, $L \leq_F L'$
- If L' is *C-hard* and $L' \in C$, then L' is **C-complete** (under F-reductions)
 - We do not want C to be computable in F, otherwise the reduction itself can decide L so everything in C would be complete under F-reductions
 - Once we get one complete language L' for a class C , to prove *C-hardness* for another language L'' enough to show $L' \leq_F L''$
 - The definition of “hard” applies computational problems other than languages,
 - but not the definition of complete.





Hardness and completeness

- *Definition:* A language L' is **hard** for a class C (*C-hard*) under F-reductions iff for every $L \in C$, there is a function f_L computable in F such that $\forall x \in \Sigma^*$ $x \in L \leftrightarrow f_L(x) \in L'$. That is, for all $L \in C$, $L \leq_F L'$
- If L' is C -hard and $L' \in C$, then L' is **C-complete** (under F-reductions).
- In particular, L' is **NP-hard** (under polytime reductions) iff for every $L \in \text{NP}$ there exists a polynomial-time computable function f_L such that $\forall x \quad x \in L \leftrightarrow f_L(x) \in L'$.
- If, also, $L' \in \text{NP}$, then L' is **NP-complete**.
 - That is, $L' \in \text{NP}$ and for all $L \in \text{NP}$, $L \leq_p L'$
 - Unless specified otherwise, NP hardness and completeness are under polytime reductions.



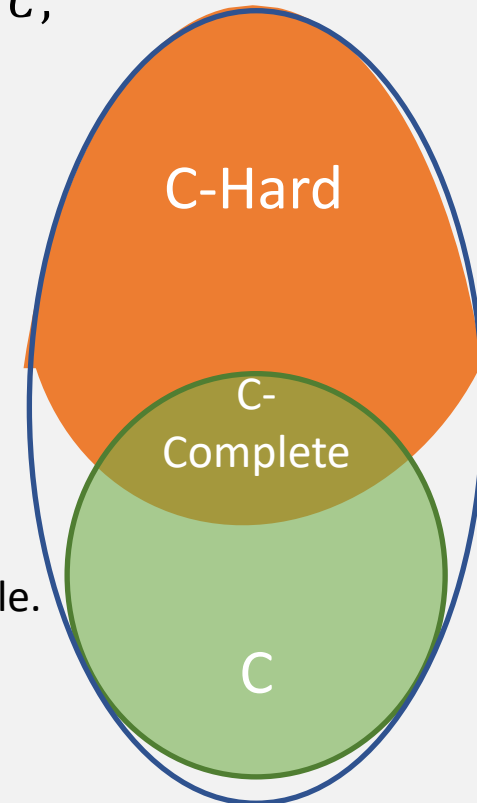


A_{TM} is r.e.-complete

- *Definition:* A language L' is **hard** for a class C under F-reductions iff for every $L \in C$, there is a function f_L computable in F such that $\forall x \in \Sigma^* x \in L \leftrightarrow f_L(x) \in L'$
 - If, additionally, $L' \in C$, then L is **C-complete** (under F-reductions).

Example: A_{TM} is complete for *r.e.* under *computable* reductions

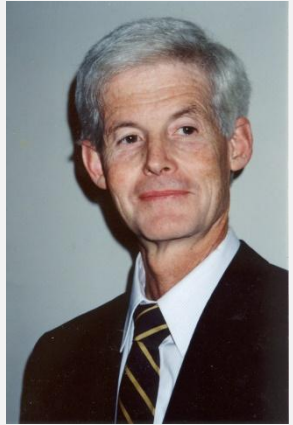
- That is, C is r.e., F is computable functions.
- Proof: We already did $A_{TM} \in r.e.$ It remains to show that A_{TM} is r.e.-hard. That is, $\forall L \in r.e.$ there is a computable f_L such that $\forall x, x \in L \leftrightarrow f_L(x) \in A_{TM}$
 - Since L in r.e., there is a TM M_L recognizing L .
 - Now, define $f_L(x) = \langle M_L, x \rangle$. Then $x \in L$ iff M_L accepts x , that is iff $\langle M_L, x \rangle \in A_{TM}$
 - Note: M_L is hardcoded in f_L : we get it from the $L \in r.e.$ assumption. Thus f_L is computable.
- Note: there can be languages neither in C nor C-hard
 - Eg: co-r.e. undecidable languages are neither in r.e. nor r.e.-hard.





NP-completeness

- 1971: Stephen Cook at the University of Toronto publishes “Complexity of theorem-proving procedures” where he proves that SAT is NP-complete.
- At the same time, Leonid Levin in Soviet Union worked on “perebor problem” (“перебор”), the problem of eliminating exhaustive search:
 - Are there problems that cannot be solved faster than by exhaustive search?
 - If so, $P \neq NP$
- The next year after Stephen Cook’s paper, Richard Karp in Berkeley showed that 21 problems people have been trying hard to solve are NP-complete
 - Decision versions of 0-1 integer programming, job sequencing, Hamiltonian path...
 - That got everybody paying attention to NP-completeness!



Stephen Cook



Leonid Levin

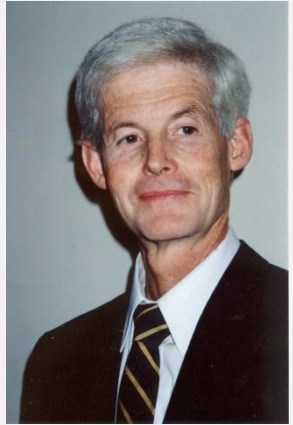


Richard Karp



NP-completeness

- 1971: Stephen Cook at the University of Toronto publishes “Complexity of theorem-proving procedures” where he proves that SAT is NP-complete.
- At the same time, Leonid Levin in Soviet Union worked on “perebor problem” (“перебор”), the problem of eliminating exhaustive search:
 - Are there problems that cannot be solved faster than by exhaustive search?
 - If so, $P \neq NP$
- The next year after Stephen Cook’s paper, Richard Karp in Berkeley showed that 21 problems people have been trying hard to solve are NP-complete
 - Decision versions of 0-1 integer programming, job sequencing, Hamiltonian path...
 - That got everybody paying attention to NP-completeness!



Stephen Cook

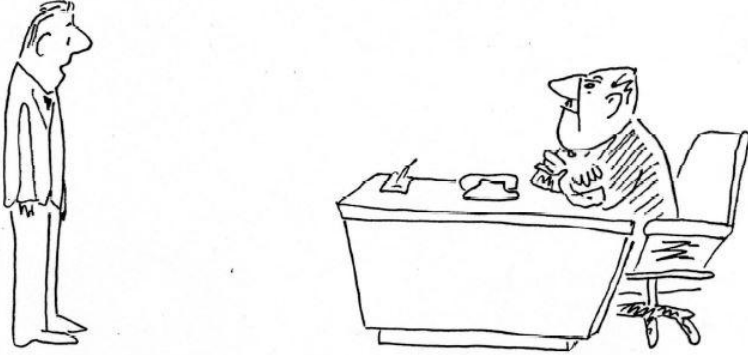


Leonid Levin



Richard Karp

David Johnson's cartoons



"I CAN'T SOLVE IT - I GUESS I'M JUST TOO DUMB."



"I CAN'T SOLVE IT - BECAUSE NO SOLUTION EXISTS!"



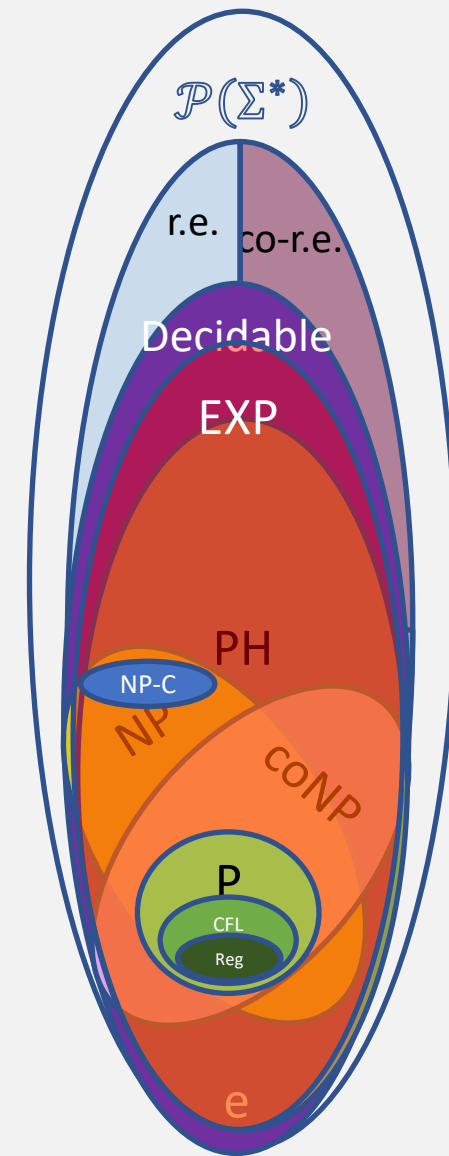
"I CAN'T SOLVE IT - BUT NEITHER CAN ALL THESE FAMOUS PEOPLE!"



WE MAY NOT BE ABLE TO SOLVE IT...
BUT WE SURE CAN GET CLOSE!

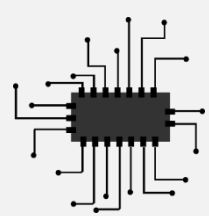
Between P and NP-complete

- Is everything in NP either in P or NP-complete?
 - No!
- Ladner's theorem:
 - If $P \neq NP$, then there are (infinitely many) problems between P and NP-complete.



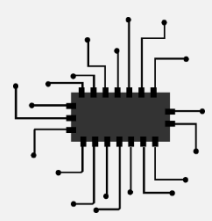
Graph Isomorphism

$$O(n^{\log^c n})$$



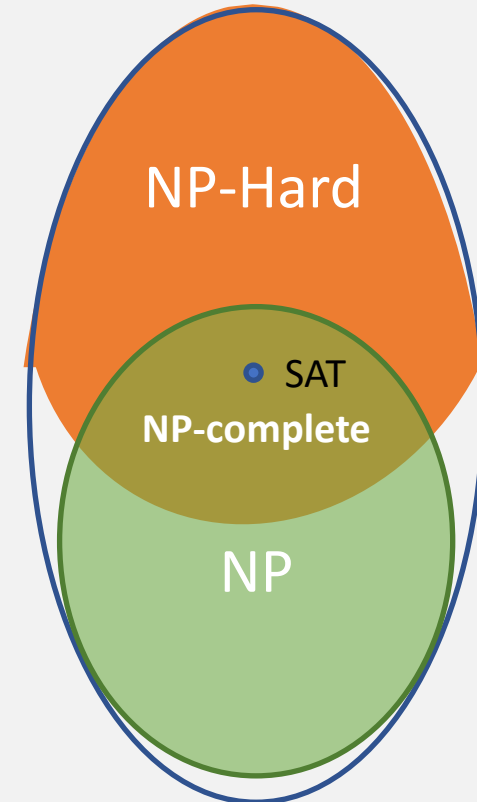
First NP-complete problem: SAT

- 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



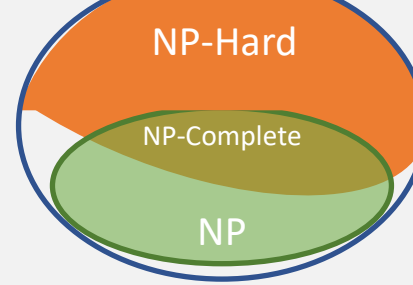
Cook-Levin theorem

- 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