SAT and 3SAT

SAT: Given a propositional formula, is it satisfiable?

Useful to write formula in conjunctive normal form:

- conjunction of clauses \( x_1 \lor x_2 \lor \ldots \lor x_k \) (variable or negation)

Can we polynomially reduce SAT \( \leq_P \) CNF-SAT? 

CNF conversion exponential. Polynomial reduction possible.

- k-SAT: at most (some def: exactly) \( k \) literals per clause
  - in particular: 3SAT (exactly 3 literals per clause)

How about 2SAT? 

\[
(x_1 \lor x_3) \land (\overline{x_1} \lor x_2) \land \ldots \land (x_4 \lor \overline{x_2})
\]

2SAT is polynomial-time

NP-completeness

Def. A language \( B \) is NP-complete iff

1. \( B \) is in NP
2. for any \( A \) in NP, we have \( A \leq_P B \)

(2) means \( B \) is NP-hard (at least as hard as any problem in NP); \( B \) itself need not be in NP

In NP (1) + NP-hard (2) means NP-complete.

- If \( B \) is NP-complete and \( B \leq_P C \), then \( C \) is NP-complete
- Reduce known NP-complete problem \( B \) to target \( C \)
- Reduce target problem \( C \) from NP-complete problem \( B \)

- If \( B \) is NP-complete and \( B \in P \), then \( P = NP \)

All NP-complete problems are polynomially reducible to one another

(The hardest problems in NP)

Review: Polynomial-time Reductions

Def. Language \( A \) is polynomial-time (mapping) reducible to language \( B \) \( (A \leq_P B) \) if a polynomial-time computable function \( f : \Sigma^* \rightarrow \Sigma^* \) exists, where for all \( w \),

\[
w \in A \iff f(w) \in B
\]

Use membership testing (solution) for \( B \) to decide \( A \) efficiently

- Reduction (function) goes one-way
- Equivalence proof goes both ways

YES maps to YES also need NO mapped to NO

Theorem. If \( A \leq_P B \) and \( B \in P \), then \( A \in P \)

3SAT \( \leq_P \) CLIQUE

Construct an instance of CLIQUE from a formula

- for \( k \) clauses, construct \( (G, k) \) \( (k\text{-CLIQUE}) \)

Construction: \( k \) groups of 3 nodes, one group for each clause.

- no connection between nodes from one clause
  - can have \( k\text{-CLIQUE} \) only by choosing one node per clause equivalent to satisfying each clause
- no connection between any \( x_i \) and \( \overline{x_i} \)
  - cannot choose \( x_i \) and \( \overline{x_i} \) at the same time equivalent to maintaining consistency
  - connect all other nodes

Cook-Levin Theorem: SAT is NP-complete

SAT: the prototypical NP-complete problem

Stephen Cook, 1971: proof
Richard Karp, 1972: 21 NP-complete problems
Leonid Levin (USSR), 1970s: 6 “universal search problems”

SAT clearly in NP: guess satisfying assignment, verify in poly-time

Proof idea:
Reduce any polynomial time NTM decider to a SAT problem.

Concretely: let \( A \in NP \) and \( \mathcal{M} \) a NTM that decides \( A \) in time \( n^k \).
7.4 NP-COMPLETENESS

PROOF IDEA
Showing that SAT is in NP is easy, and we do so shortly. The hard part of the proof is showing that any language in NP is reducible to SAT.

A Tableau of Configurations

Since time $n^k$, tape length at most $n^k$

Encoding Computation History as SAT

Each cell is a variable: $x_{i,j,s} = 1$ iff $cell[i,j] = s$
$s \in \{Q \cup \Gamma \cup \{\#\}\}$ (state or symbol)

Expressing the constraints

- One true variable per cell (unique cell contents) $O(n^{2k})$
- Encode starting configuration $O(n^k)$
- Encode valid moves (transition relation) using 3x2 windows $O(n^{2k})$ constant size per cell
- Some state in some line is accepting $O(n^{2k})$

Claim: If top row is start configuration, and every window is legal, then every row of the tableau is a configuration that legally follows the preceding one.

This shows we have indeed achieved a reduction.

3SAT is NP-complete

Proof options

1) Do nondeterministic TM conversion directly to 3SAT
   formula “almost” in CNF, except “windows” (constant size)

2) Converting arbitrary formula to 3SAT (Tseitin transform)
   new variable for each subformula
   
   $x \leftrightarrow \neg A \quad (\neg x \lor \neg A) \land (x \lor A)$
   
   $x \leftrightarrow A \lor B \quad (x \to A \lor B) \land (A \lor B \to x)$
   
   $= (\neg x \lor A \lor B) \land (\neg A \lor x) \land (\neg B \lor x)$
   
   $x \leftrightarrow A \land B \quad (x \to A \land B) \land (A \land B \to x)$
   
   $= (\neg x \lor A) \land (\neg x \lor B) \land (\neg A \lor \neg B \lor x)$

   In both cases, convert clause with $n$ literals to three:

   $x_1 \lor \ldots \lor x_n$ equisatisfiable with

   $(x_1 \lor x_2 \lor \neg 1) \land (\neg 1 \lor x_1 \lor \neg 2) \land \ldots \land (\neg n-1 \lor x_{n-1} \lor \neg n)$