22 April 2019 — The Cook-Levin Theorem (3SAT is NP-Complete)

Multi-tape Turing machine:

$T < \infty$ tapes (infinite sequences of symbols in Σ)

finite state set Q (s = starting state, t = terminal state)

T read-write heads each occupying a position on one tape.

Transition function $\delta : Q \times \Sigma^T \longrightarrow Q \times \Sigma^T \times \{-1, 0, 1\}^T$

Decision problem (a.k.a. "language") $L \subseteq \Sigma^*$ belongs to NP if and only if $\exists$ verifier V, an algorithm (TURING MACHINE) with two inputs x, y, s.t. $\forall x \in \Sigma^*$

$$x \in L \iff \exists y \; |y| \leq \text{poly}(|x|) \text{ s.t. } V(x,y) = 1,$$

and such that V's worst case running time is $O(\text{poly}(|x|))$.

Cook-Levin Theorem. Every language L in NP is polynomial time Karp reducible to 3SAT. $\equiv$ 3SAT is NP-Hard.

(This implies 3SAT is NP-Complete because 3SAT $\in$ NP is an exercise.)

Proof step 1. CNF-SAT $\leq_p$ 3SAT.

In CNF SAT we have a Boolean formula which is a (AND) conjunction of clauses. Each clause is a (OR) disjunction of literals. Is it satisfiable?

Exactly like 3SAT without the limitation of 3 literals per clause.

Gadget: If we have a disjunction with $n+m$ literals we can introduce an extra variable and replace the clause with 2 clauses, having $n+1$ and $m+1$ variables.

$a_1 \vee a_2 \vee \cdots \vee a_n \vee b_1 \vee b_2 \vee \cdots \vee b_m$ (n+m)

$\hookrightarrow (a_1 \vee a_2 \vee \cdots \vee a_n \vee z) \wedge (\bar{z} \vee b_1 \vee b_2 \vee \cdots \vee b_m)$ (n+1, m+1)

Note: A truth assignment of $a_1, \ldots, a_n, b_1, \ldots, b_m$ satisfies the 1st clause $\iff \exists z$ s.t. second pair of clauses is satisfied.

Note: This reduces clause size if $n, m > 1$ i.e. $m+n \geq 4$. For 3CNF clauses their size can't be further reduced by this trick.

If we iteratively apply this trick to break down a clause of size k into 3CNF clauses we get how many?

(A) $3k$ (B) $2k+2$ (C) $k-2$ (D) $2k-2$

$l_1 \vee l_2 \vee l_3 \vee l_4 \vee \cdots \vee l_{k-2} \vee l_{k-1} \vee l_k$

$(l_1 \vee l_2 \vee z) \quad (\bar{z} \vee l_3 \vee l_4 \vee \cdots \vee l_k)$ +1

$(\bar{z} \vee l_3 \vee z') \quad (\bar{z}' \vee l_4 \vee \cdots \vee l_k)$

$(\bar{z}' \vee l_4 \vee z'') \quad (\bar{z}'' \vee l_5 \vee \cdots \vee l_k)$

k-4 of these

$(\bar{z}'' \vee l_{k-2} \vee z''') \quad (\bar{z}''' \vee l_{k-1} \vee l_k)$ +1

"constraint satisfaction problem"

Proof step 2. CSP $\leq_p$ CNF-SAT

In CSP we are given variables $x_1, \ldots, x_n$
finite sets $X_1, \ldots, X_n$
and constraints of the form

$$(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \neq (y_{i_1}, \ldots, y_{i_k})$$

← k-tuple of variables → ← k-tuple of values in $X_{i_1} \times X_{i_2} \times \cdots \times X_{i_k}$

Does there exist an assignment of values to variables s.t. each x_i is assigned a value in X_i and each constraint is satisfied?

E.g. CNF-SAT is a special case of CSP. $X_i = \{T, F\}$ $\forall i$

IS $\leq_p$ CSP

Given (G, k) set up a CSP with variables $x_1, \ldots, x_k$ representing the elements of the indep set.

$$X_1 = \ldots = X_k = V(G).$$

Constraints $\forall i \neq j$ $\quad x_i \neq x_j$ && $(x_i, x_j) \notin E(G)$.

The CSP $\leq_p$ CNF-SAT reduction ~~takes~~ a CSP problem and forms Boolean variables z_{ij} $\forall\, i \in [n],\ j \in X_i$.

Interpretation: $z_{ij} = \text{TRUE}$ represents choosing value $x_i = j$.

Clauses: (i) Each x_i must be assigned exactly one value in X_i

$$\left(\bigvee_{j \in X_i} z_{ij}\right) \wedge \left(\overline{z_{i1}} \vee \overline{z_{i2}}\right) \wedge \left(\overline{z_{i1}} \vee \overline{z_{i3}}\right) \wedge \ldots$$

x_i gets at least one value in X_i — x_i doesn't get 2 different values.

(ii) Each CSP clause must be satisfied.

$$(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \neq (y_{i_1}, \ldots, y_{i_k})$$

$$\equiv \overline{z_{i_1, y_{i_1}}} \vee \overline{z_{i_2, y_{i_2}}} \vee \cdots \vee \overline{z_{i_k, y_{i_k}}}$$

Proof step 3. For every $L \in NP$, $L \leq_p CSP$.

Trick in this proof is to encode the verifier's accepting computation as a set of CSP variables.

Recall: $\exists$ a Turing machine V and a polynomial $P(n)$ s.t. $\forall x \in \Sigma^*$, $x \in L \iff \exists y$ s.t. V accepts x, y in at most $P(|x|)$ steps.

Make a bunch of variables that encode everything that happens during an computation that accepts x. $n = |x|$.

$\tilde{\sigma}(j,k)$ = symbol being read on tape j at time k.

$\sigma(i,j,k)$ = symbol in position i on tape j at time k.

$0 \leq i \leq P(n)$, $1 \leq j \leq T$, $0 \leq k \leq P(n)$.

$q(k)$ = machine's state at time k

$\pi(j,k)$ = position of j^{th} read-write head at time k.

Constraints. [1.] The initial configuration corresponds to input x paired with some $y \in \Sigma^*$

$$\sigma(i,1,0) = \begin{cases} x_i & \text{if } i \leq n \\ \_ & \text{if } i > n \end{cases}$$

Tape 1 contains x

$\sigma(i,2,0)$ unconstrained

Tape 2 contains any y.

$\sigma(i,j,0) = \_$ if $j > 2$.

Tape 3,...T initially blank.

[2] The computation proceeds according to δ_V, the verifier's transition rule.

$$\left(\sigma(i,j,k) = \sigma \;\&\&\; \pi(j,k) = i\right) \Longrightarrow \tilde{\sigma}(j,k) = \sigma$$

$$\left(\tilde{\sigma}(1,k) = \sigma_1, \tilde{\sigma}(2,k) = \sigma_2, \ldots \tilde{\sigma}(t,k) = \sigma_t\right) \Longrightarrow$$

A bunch of variables at time $k+1$ have values determined by δ_V

[3] The computation accepts (x,y): $\sigma(0,3,P(n)) = '1'$.