REDUCING ONE (DECISION) PROBLEM TO ANOTHER

A

input for A

Transform to input/problem B

input problem B

TR

solve B

output of B

output of A

NO

YES

solve A using B : TR + TB

is L.I.S.(A) \( \geq k \) ?

copy A \( \rightarrow \) B

sort B

is L.C.S.(A,B) \( \geq k \) ?

YES

YES

NO

NO
Lower bound for one (decision) problem via another

This is just one way of solving D

- Transform to input/problem C
  
  \[ T_R = o(f(n)) \]

- Solve C
  
  \[ T_C = \Omega(f(n)) \]

Solve D: known \( \Omega(f(n)) = T_R + T_C \) \( \Rightarrow T_C = \Omega(f(n)) \)

- Quick transformation \( \Rightarrow \) C is at least as difficult as D

(First solution for C \( \Rightarrow \) fast solution for D)

e.g. D = sorting, C = convex hull.

D might have another (easier) solution too, w/o using C
NPC REDUCTION for one (decision) problem via another C (from) D 

~Like establishing an almost certain lower bound for C.

input for D → Transform to input/problem C → solve C

$T_R = \text{polynomial}$

output of C → YES

NO

output of D → YES

NO

solve D: known NPC = $T_R + T_c = \text{poly-time} + T_c$

$\Rightarrow T_c$ is NP-hard

& if C is in NP then C is NPC

(or $T_c$ is polynomial $\Rightarrow P=NP$)

(plus you rule and/or destroy the world)
\( A \leq_p B \) : in polynomial time, A can be _reduced_ to B

\( \implies B \) is at least as hard as A.

Solving B in poly-time \( \Rightarrow \) solving A in poly-time

If \( \{\text{any problem in } NPC\} \leq_p B \) then B is NPC.

(assuming B is in NP, otherwise it is just NP-hard)

Notice if A & B are NPC then A \( \leq_p B \) & B \( \leq_p A \).

\( \sim A \cong_p B \)
Thousands of NPC Problems

Reductions/transformations between them resemble a di-graph.

\[ \subseteq \text{a DAG? No.} \]

- Strongly connected components?
  - Must be 1 precisely.

**Fact**

\( \exists \) at least one problem \( A \) s.t.

\( A \geq_p \\{ \text{everything} \} \)

& \( \{ \text{everything} \} \geq_p \ldots \geq_p \ldots \geq_p A \)
Circuit SAT (satisfiability)
The first NPC problem.

Given a circuit, can the output ever be 1?

1 - in NP ... intuitive
2 - every problem can be described as a circuit

\[ \text{everything} \leq_p \text{circuit-SAT} \]

...and we don't know how to solve this in polynomial time. (not in P)
Boolean SAT \( (x_1 \lor x_2 \lor \overline{x_3}) \land ((x_1 \leftrightarrow x_5) \lor (x_4 \rightarrow \overline{x_3})) \rightarrow ? \rightarrow 1 \)

Given a circuit-SAT instance, transform (quickly) into a Boolean SAT:

\[
\begin{align*}
\text{by solving the Boolean SAT you get an answer for circuit SAT} \\
W \land (W \leftrightarrow (Y_4 \land Y_2 \land Y_3)) \\
\land (Y_4 \leftrightarrow (Y_1 \lor Y_2)) \\
\land (Y_1 \leftrightarrow (C_1 \lor C_2)) \\
\land (Y_2 \leftrightarrow \overline{C_3}) \\
\land (Y_3 \leftrightarrow (C_4 \land C_5))
\end{align*}
\]
3-SAT

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_4 \lor x_5) \land (x_2 \lor \overline{x_4} \lor x_{13}) \land (x_2 \lor \overline{x_3} \lor x_3) \ldots\]

\(\text{n clauses, each with 3 literals}\)
\[(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})\] k clauses

3-SAT \rightarrow boolean SAT \rightarrow Circuit SAT

construct graph (quickly)

for every clause=triple \((x \lor y \lor z)\) make a triple of vertices, linked together
\((a \lor b \lor c) \land (b \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor c \lor d) \land (a \lor \bar{b} \lor \bar{d})\) 

\(k\) clauses

3-SAT \(\rightarrow\) boolean SAT \(\rightarrow\) circuit SAT

construct graph (quickly)

link every vertex to any other vertex that has the same variable but negated
(a \lor b \lor c) \\
\land (b \lor \bar{e} \lor \bar{d}) \\
\land (\bar{a} \lor c \lor d) \\
\land (a \lor \bar{b} \lor \bar{d}) \\
\text{\{\hspace{1cm} k \text{ clauses} \hspace{1cm}\}}

\text{construct\hspace{1cm}graph\hspace{1cm}(quickly)}

3-SAT \rightarrow \text{boolean SAT} \rightarrow \text{circuit SAT}

\text{Ask: does this graph have k independent vertices?}
\((a \lor b \lor c) \land (b \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor c \lor d) \land (a \lor \bar{b} \lor \bar{d})\) 

\(k\) clauses

\[ \text{construct graph (quickly)} \]

\[ \text{3-SAT} \rightarrow \text{boolean SAT} \rightarrow \text{circuit SAT} \]

Ask: does this graph have \(k\) independent vertices?

To find \(k\) independent, you must use each triple, because each triple can only have 1 independent vertex.

As soon as you select a vertex in a triple, that blocks you from using its negation anywhere else.
\[(a \lor b \lor c) \land (b \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor c \lor d) \land (a \lor \bar{b} \lor \bar{d})\] for \(k\) clauses.

Construct graph (quickly).

If you can't find \(k\) independent vertices, then some triple had no selectable vertex... i.e. all 3 were blocked by selections in other triples. That corresponds to an unsatisfiable clause in 3-SAT.

Ask: does this graph have \(k\) independent vertices?

If yes, then 3-SAT is \(\checkmark\)
If no, then 3-SAT is \(\times\)
\[
\{ (a \lor b \lor c), \\
\land (b \lor \overline{c} \lor \overline{d}), \\
\land (\overline{a} \lor c \lor d), \\
\land (a \lor \overline{b} \lor \overline{d}) \}
\]

k clauses

3-SAT \rightarrow \text{boolean SAT} \rightarrow \text{Circuit SAT}

\text{independent set}

Ask: does this graph have k independent vertices?

If yes, then 3-SAT is \text{\checkmark}
If no, then 3-SAT is \text{\times}

\text{construct graph} \quad \text{(quickly)}

\leq 1 \text{ per triplet}
CLIQUE in a graph: subset of $V$ s.t. all pairs of vertices share edges.

independent set size $k$ \[\iff\] clique size $k$

$G$ \[\xrightarrow{\text{transform}}\] complement $(G)$

Asking if a graph has a clique of size $k$ : NPC
OTHER HARD PROBLEMS
- knapsack: given item types, w/ size & value, fill a bag w/ max value (multiples ok)
- subset sum: does a subset of given integers sum to t?
- partition: split set of integers in 2 groups with equal sums
- planar SAT: SAT without wire crossings
- set cover: given some sets, select min # sets s.t. all elements are present
- hitting set: given sets, select min # elements s.t. all sets are represented
- longest path: visiting each vertex once.
- Steiner tree: ~MST on selected subset of a graph
- traveling salesman: min-cost simple cycle on weighted complete graph
OTHER HARD PROBLEMS

... & some are even harder

Tetris  Minesweeper  Lemmings

Mario Bros  Pac-man

Prince of Persia  Portal  Doom

etc