Certifiers

Definition
An algorithm $C(\cdot, \cdot)$ is a certifier for problem $X$ if for every $s \in X$ there is some string $t$ such that $C(s, t) =$ "yes", and conversely, if for some $s$ and $t$, $C(s, t) =$ "yes" then $s \in X$.

The string $t$ is called a certificate or proof for $s$.

Definition (Efficient Certifier.)
A certifier $C$ is an efficient certifier for problem $X$ if there is a polynomial $p(\cdot)$ such that for every string $s$, we have that

1. $|t| \leq p(|s|)$,
2. $C(s, t) =$ "yes",
3. and $C$ runs in polynomial time.

NP-Complete Problems

Definition
A problem $X$ is said to be NP-Complete if

- $X \in \text{NP}$, and
- (Hardness) For any $Y \in \text{NP}$, $Y \leq_p X$. 

Part I

NP Completeness
Proposition
Suppose $X$ is NP-Complete. Then $X$ can be solved in polynomial time if and only if $P = NP$.

Proof.
\[
\Rightarrow \quad \text{Suppose } X \text{ can be solved in polynomial time}
\]
\[
\quad \text{Let } Y \in \text{NP}. \text{ We know } Y \leq_p X.
\]
\[
\quad \text{We showed that if } Y \leq_p X \text{ and } X \text{ can be solved in polynomial time, then } Y \text{ can be solved in polynomial time.}
\]
\[
\quad \text{Thus, every problem } Y \in \text{NP} \text{ is such that } Y \in P; \text{ NP} \subseteq P.
\]
\[
\quad \text{Since } P \subseteq \text{NP}, \text{ we have } P = \text{NP}.
\]
\[
\Leftarrow \quad \text{Since } P = \text{NP}, \text{ and } X \in \text{NP}, \text{ we have a polynomial time algorithm for } X.
\]

Consequences of proving NP-Completeness
If $X$ is NP-Complete
\[
\quad \text{Since we believe } P \neq \text{NP},
\]
\[
\quad \text{and solving } X \text{ implies } P = \text{NP}.
\]
\[
\quad X \text{ is unlikely to be efficiently solvable.}
\]

At the very least, many smart people before you have failed to find an efficient algorithm for $X$.
(This is proof by mob opinion — take with a grain of salt.)
Circuits

Definition
A circuit is a directed acyclic graph with

1. **Input** vertices (without incoming edges) labelled with 0, 1 or a distinct variable.
2. Every other vertex is labelled ∨, ∧ or ¬.
3. Single node **output** vertex with no outgoing edges.

Claim
**CSAT is in NP.**

- **Certificate:** Assignment to input variables.
- **Certifier:** Evaluate the value of each gate in a topological sort of DAG and check the output gate value.

Cook-Levin Theorem

Definition (Circuit Satisfaction (**CSAT**).)
Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

Theorem (Cook-Levin)
**CSAT is NP-Complete.**

Need to show

1. **CSAT** is in **NP**.
2. every **NP** problem **X** reduces to **CSAT**.

CSAT is NP-hard: Idea

Need to show that every **NP** problem **X** reduces to **CSAT**.

What does it mean that **X** ∈ **NP**?

**X** ∈ **NP** implies that there are polynomials **p()** and **q()** and certifier/verifier program **C** such that for every string **s** the following is true:

- If **s** is a YES instance (**s** ∈ **X**) then there is a **proof** **t** of length **p(|s|)** such that **C(s, t)** says YES.
- If **s** is a NO instance (**s** ∉ **X**) then for every string **t** of length at **p(|s|)**, **C(s, t)** says NO.
- **C(s, t)** runs in time **q(|s| + |t|)** time (hence polynomial time).
Reducing $X$ to CSAT

$X$ is in $\text{NP}$ means we have access to $p(), q(), C(\cdot, \cdot)$.

What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
How are $p()$ and $q()$ given? As numbers.
Example: if 3 is given then $p(n) = n^3$.

Thus an $\text{NP}$ problem is essentially a three tuple $\langle p, q, C \rangle$ where $C$ is either a program or a TM.

Reducing $X$ to CSAT

How do we reduce $X$ to $\text{CSAT}$? Need an algorithm $\mathcal{A}$ that
- takes $s$ (and $\langle p, q, C \rangle$) and creates a circuit $G$ in polynomial time in $|s|$ (note that $\langle p, q, C \rangle$ are fixed).
- $G$ is satisfiable if and only if there is a proof $t$ such that $C(s, t)$ says YES

Simple but Big Idea: Programs are essentially the same as Circuits!
- Convert $C(s, t)$ into a circuit $G$ with $t$ as unknown inputs (rest is known including $s$)
- We know that $|t| = p(|s|)$ so express boolean string $t$ as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
- Asking if there is a proof $t$ that makes $C(s, t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make $G$ evaluate to true/YES.

Example: Independent Set

Problem: Does $G = (V, E)$ have an Independent Set of size $\geq k$?
- Certificate: Set $S \subseteq V$.
- Certifier: Check $|S| \geq k$ and no pair of vertices in $S$ is connected by an edge.

Formally, why is Independent Set in $\text{NP}$?
Example: Independent Set

Formally why is Independent Set in \( \text{NP} \)?

- **Input:** 
  
  \(< n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k >\) encodes \(< G, k >\).

- \( n \) is number of vertices in \( G \)
- \( y_{i,j} \) is a bit which is 1 if edge \((i, j)\) is in \( G \) and 0 otherwise (adjacency matrix representation)
- \( k \) is size of independent set.

- **Certificate:** \( t = t_1 t_2 \ldots t_n \). Interpretation is that \( t_i \) is 1 if vertex \( i \) is in the independent set, 0 otherwise.

Certifier for Independent Set

Certifier \( C(s, t) \) for Independent Set:

- if \((t_1 + t_2 + \ldots + t_n < k)\) then
  - return \( \text{NO} \)
- else
  - for each \((i, j)\) do
    - if \((t_i \land t_j \land y_{i,j})\) then
      - return \( \text{NO} \)
  - return \( \text{YES} \)

Example: Independent Set

A certifier circuit for Independent Set

![Diagram of a certifier circuit for Independent Set]

Programs, Turing Machines and Circuits

Consider “program” \( A \) that takes \( f(|s|) \) steps on input string \( s \).

**Question:** What computer is the program running on and what does step mean?

Real computers difficult to reason with mathematically because

- instruction set is too rich
- pointers and control flow jumps in one step
- assumption that pointer to code fits in one word

Turing Machines

- simpler model of computation to reason with
- can simulate real computers with \textit{polynomial} slow down
- all moves are \textit{local} (head moves only one cell)
Certifiers that at TMs

Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine $M$

Problem: Given $M$, input $s$, $p$, $q$ decide if there is a proof $t$ of length $p(|s|)$ such that $M$ on $s$, $t$ will halt in $q(|s|)$ time and say YES.

There is an algorithm $A$ that can reduce above problem to $CSAT$ mechanically as follows.

1. $A$ first computes $p(|s|)$ and $q(|s|)$.
2. Knows that $M$ can use at most $q(|s|)$ memory/tape cells.
3. Knows that $M$ can run for at most $q(|s|)$ time.
4. Simulates the evolution of the state of $M$ and memory over time using a big circuit.

NP-Hardness of Circuit Satisfaction

Key Ideas in reduction:

1. Use TMs as the code for certifier for simplicity.
2. Since $p()$ and $q()$ are known to $A$, it can set up all required memory and time steps in advance.
3. Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time.

Note: Above reduction can be done to SAT as well. Reduction to SAT was the original proof of Steve Cook.

SAT is NP-Complete

- We have seen that $SAT \in NP$.
- To show NP-Hardness, we will reduce Circuit Satisfiability (CSAT) to SAT.

Instance of CSAT (we label each node):

Output: $\neg a \land (b \land c) \lor (d \land e)$

Inputs: $x_0, x_1, x_2, x_3, x_4$
Converting a circuit into a CNF formula

Label the nodes

(A) Input circuit
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node

(B) Label the nodes.
(C) Introduce var for each node.

Converting a circuit into a CNF formula

Write a sub-formula for each variable that is true if the var is computed correctly.

XXX

(D) Write a sub-formula for each variable that is true if the var is computed correctly.

Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula

| $x_k$ | $x_k = x_i \land x_e$ | $x_k = x_i \land x_k$ |
| $x_k = x_i \land x_k$ | $x_j = x_g \land x_h$ | $x_j = x_g \land x_h$ |
| $x_i = \neg x_f$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
| $x_h = x_d \lor x_e$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
| $x_g = x_b \lor x_c$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
| $x_f = x_a \land x_e$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
| $x_d = 0$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
| $x_a = 1$ | $x_i = \neg x_f$ | $x_i = \neg x_i \lor \neg x_f$ |
Converting a circuit into a CNF formula

Take the conjunction of all the CNF sub-formulas

\[ x_k \land (\neg x_k \lor x_i) \land (\neg x_k \lor x_j) \land (x_k \lor v) \land (x_k \lor x_f) \land \ldots \]

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.

Reduction: \( \text{CSAT} \leq_{P} \text{SAT} \)

Continued...

- Case \( \lor \): So \( x_v = x_u \lor x_w \). In SAT formula generated, add clauses \((x_v \lor \neg x_u), (x_v \lor \neg x_w)\), and \((\neg x_v \lor x_u \lor x_w)\). Again, observe that

\[
\begin{align*}
(x_v = x_u \lor x_w) \text{ is true } & \iff (x_v \lor \neg x_u), \\
(x_v \lor \neg x_u) \text{ all true.} & \iff (\neg x_v \lor x_u \lor x_w)
\end{align*}
\]

Reduction: \( \text{CSAT} \leq_{P} \text{SAT} \)

Continued...

- Case \( \land \): So \( x_v = x_u \land x_w \). In SAT formula generated, add clauses \( (\neg x_v \lor x_u), (\neg x_v \lor x_w)\), and \((x_v \lor \neg x_u \lor \neg x_w)\). Again, observe that

\[
\begin{align*}
x_v = x_u \land x_w \text{ is true } & \iff (\neg x_v \lor x_u), \\
(\neg x_v \lor x_u) \text{ all true.} & \iff (x_v \lor \neg x_u \lor \neg x_w)
\end{align*}
\]
Reduction: \( \text{CSAT} \leq_p \text{SAT} \)

Continued...

- If \( v \) is an input gate with a fixed value then we do the following.
  - If \( x_v = 1 \) add clause \( x_v \).
  - If \( x_v = 0 \) add clause \( \neg x_v \).
- Add the clause \( x_v \) where \( v \) is the variable for the output gate.

Correctness of Reduction

Need to show circuit \( C \) is satisfiable iff \( \varphi_C \) is satisfiable

\[ \Rightarrow \]
- Consider a satisfying assignment \( a \) for \( C \)
  1. Find values of all gates in \( C \) under \( a \)
  2. Give value of gate \( v \) to variable \( x_v \); call this assignment \( a' \)
  3. \( a' \) satisfies \( \varphi_C \) (exercise)

\[ \Leftarrow \]
- Consider a satisfying assignment \( a \) for \( \varphi_C \)
  1. Let \( a' \) be the restriction of \( a \) to only the input variables
  2. Value of gate \( v \) under \( a' \) is the same as value of \( x_v \) in \( a \)
  3. Thus, \( a' \) satisfies \( C \)

Theorem

\( \text{SAT} \) is \text{NP-Complete}.\

Proving that a problem \( X \) is \text{NP-Complete}

To prove \( X \) is \text{NP-Complete}, show

- \( X \) is in \text{NP}.
  1. Certificate/proof of polynomial size in input
  2. Polynomial time certifier \( C(s, t) \)
- Reduction from a known \text{NP-Complete} problem such as \( \text{CSAT} \) or \( \text{SAT} \) to \( X \)

\( \text{SAT} \leq_p X \) implies that every \text{NP} problem \( Y \leq_p X \). Why?

Transitivity of reductions:

\( Y \leq_p \text{SAT} \) and \( \text{SAT} \leq_p X \) and hence \( Y \leq_p X \).

NP-Completeness via Reductions

- \( \text{CSAT} \) is \text{NP-Complete}.
- \( \text{CSAT} \leq_p \text{SAT} \) and \( \text{SAT} \) is in \text{NP} and hence \( \text{SAT} \) is \text{NP-Complete}.
- \( \text{SAT} \leq_p 3\text{-SAT} \) and hence 3-SAT is \text{NP-Complete}.
- 3-SAT \( \leq_p \text{Independent Set} \) (which is in \text{NP}) and hence \text{Independent Set} is \text{NP-Complete}.
- \text{Vertex Cover} is \text{NP-Complete}.
- \text{Clique} is \text{NP-Complete}.

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be \text{NP-Complete}.

A surprisingly frequent phenomenon!