Circuit Satisfiability

- **Circuit**: Consists of one or more **logic gates** connected by **wires**
- **Logic gates**:

<table>
<thead>
<tr>
<th>A</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A NAND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A NOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Circuit Satisfiability

- **Circuit**: Consists of one or more logic gates connected by wires
Circuit Satisfiability

- **Circuit**: alternative definition: A circuit is a directed acyclic graph with:
  - sources (no edge entering), labelled with variables or 0 or 1 — inputs
  - one sink (no edge leaving) — output
  - internal nodes

- A circuit computes an output (in the obvious way) when values are given for the input variables.
Circuit-SAT (Circuit Satisfiability) Problem

- **Input**: A circuit composed of AND, OR, and NOT gates.
- **Question**: Is it satisfiable?
Circuit-SAT is NP-Complete

**Theorem.** Circuit SAT is NP-complete.

- **Proof.**
  1. Circuit SAT is in NP.

  2. This is the first NP-completeness proof so we must prove that

     for every $X$ in $\text{NP}$, $X \leq_p \text{Circuit-SAT}$

     i.e. for every $X$ in $\text{NP}$, there is an algorithm that maps any input $x$ for $X$ to a circuit $C$ s.t. $x$ is a YES input iff $C$ is satisfiable.
Circuit SAT is in NP.

- Verifier Algorithm A:
  - Input:
    - a boolean circuit C
    - Certificate: an assignment of a boolean value to each of the source nodes in C
  - Output:
    - Result of verifying certificate: YES/NO
  - Algorithm A:
    - Checks the output of the entire circuit
      - output=1, algorithm A outputs 1
      - output=0, algorithm A outputs 0
Transforming an algorithm to a circuit

- Any algorithm that takes a **fixed** number of bits as input and produces a yes/no answer can be represented by a circuit
  - Circuit output: 1
  - Input to the circuit: the inputs for which the algorithm outputs yes
- If the number of steps of A is polynomial, the circuit has polynomial size
- Write a program for algorithm A. Compile it. Assemble . . . At the hardware level, A is implemented by \( \land \), \( \lor \), \( \neg \) gates. We get a circuit C.
- Inputs to C: bits of x (known), bits of y (variables)
- Internal nodes of circuit: memory locations after each time step of algorithm A
Circuit-SAT is NP-Complete

- Circuit-SAT is NP-complete: for every $X$ in NP, $X \leq_p$ Circuit SAT
  - for every $X$ in NP, there is an algorithm that maps any input $x$ for $X$ to a circuit $C$ s.t. $x$ is a YES input iff $C$ is satisfiable.
- Idea:
  - $X \leq_p$ Circuit SAT
  - Given input $x$ to $X$, we want to solve the problem using circuit-SAT to decide whether $X(x)$ return YES/NO
  - $X \in$ NP: $\exists$ an efficient verification algorithm $A$ for $X$. $A$ takes two inputs $x, y$ ($y =$ certificate or “guess”) and outputs YES/NO.
  - To answer whether $x \in X$ for some specific input $x$ of length $n$, we can equivalently answer

  Is there a $y$ of length $p(n)$ so that $A(x, y)$ output yes

  - Property of $A$: $x$ is a YES instance for $X$ iff $\exists$ $y$ (of poly size) s.t. $A(x, y)$ outputs YES
  - Convert algorithm $A$ with known input $x$ and unknown input $y$ to a circuit $C$ with input variables = bits of $y$ such that $C$ is satisfiable iff $\exists$ $y$ s.t. $A(x, y)$ outputs YES
  - Because size($y$) is polynomial and $A$ runs in polynomial time, the circuit has polynomial size.
Sequence of configurations produced by A when running on input x and certificate y
Example

- **Input**: a graph $G$
  - A graph with $n$ nodes: specified by $\text{choose}(n,2)$ bits
    - 1 bit for each pair:
      - 1 if there is an edge and 0 otherwise
- **Question**: does it contain a two-node independent set
- **Verifier Algorithm**:
  - **Input**:
    - Graph $G$ with $\text{choose}(n, 2)$ bits
    - The certifier $y$ with $n$ bits: each bit corresponds to a node:
      - 1 if the node belongs to the independent set and 0 otherwise
  - The certifier checks
    - At least two of the bits in $y$ are set to 1
    - No two bits in $t$ are set to 1 if they form the two ends of an edge
Example

Have both ends of some edge been chosen?

Have at least two nodes been chosen?
\[
\begin{align*}
\text{Ind. Set} \leq \text{P} & \quad \text{Vertex Cover} \leq \text{P} \\
\text{Circuit SAT} \leq \text{P} & \quad \\
\text{3-SAT} \leq \text{P} & \quad \text{directed hamiltonian cycle} \leq \text{P} \\
\text{Directed ham. cycle} \leq \text{P} & \quad \text{Subset Sum} \\
\text{Undirected ham. cycle} \leq \text{P} & \quad \\
\text{Ham. cycle} \leq \text{P} & \quad \text{TSP} \\
\text{Subset Sum} & \quad \text{Ham. path} \leq \text{P}
\end{align*}
\]
Circuit-SAT \leq_p 3-SAT

Theorem. 3-SAT is NP-complete.

1. 3-SAT is in NP. (easy, details omitted)

2. Circuit SAT \leq_p 3-SAT
   
   - Assume we have a polynomial time algorithm for 3-SAT. Make a polynomial time algorithm for Circuit SAT.
   - **Input:** A circuit C
   - **Output:** Is C satisfiable?
     
     - construct a 3-SAT formula $\phi$ such that
       
       C is satisfiable iff $\phi$ is satisfiable

     - run the 3-SAT algorithm
     - return its answer
Circuit-SAT $\leq_p$ 3-SAT

Convert circuit $C$ to formula $\varphi$

The obvious way:

Circuit Size: Superpolynomial
Circuit-SAT $\leq_P$ 3-SAT

Convert circuit $C$ to formula $\varphi$

- make a variable $x_u$ for each node $u$ in the circuit
- Output: a boolean formula in CNF, each clause contains at most 3 variables
- See example in the next slide
Circuit-SAT $\leq_p$ 3-SAT

- Convert circuit C to formula $\phi$
  - **Input:** a circuit C
  - **Output:** a boolean formula in CNF
- make a variable $x_u$ for each node $u$ in the circuit
  - See example in the next slide
  - each clause contains at most 3 variables
- Turn clauses of at **most 3** literals into clauses of **exactly** 3 literals.
  - Add dummy variables:
    - Case 1: There is 1 variable in the clause, e.g., $y$
      - Change $y$ with: $(y \lor p \lor q) \land (y \lor p \lor \neg q) \land (y \lor \neg p \lor q) \land (y \lor \neg p \lor \neg q)$
    - Case 2: There are 2 variables in clause, e.g., $(x \lor y)$
      - Change $(x \lor y)$ with: $(x \lor y \lor p) \land (x \lor y \lor \neg p)$
- Final formula: $\phi = \land$ of all clauses $\land x_{output}$
\[ x_u \equiv x_v \lor x_w \]

\[ (\neg x_u \lor x_v \lor x_w) \land (x_u \lor \neg x_v) \land (x_u \lor \neg x_w) \]

\[ x_u \equiv x_v \land x_w \]

\[ (\neg x_u \lor x_v) \land (\neg x_u \lor x_w) \land (x_u \lor \neg x_v \lor \neg x_w) \]

\[ x_u \equiv \neg x_v \]

\[ (x_u \lor x_v) \land (\neg x_u \lor \neg x_v) \]

\[ a \equiv b \text{ means } (\neg a \lor b) \land (a \lor \neg b) \]
Circuit-SAT $\leq_p$ 3-SAT: Runtime of reduction

**Claim.** $\varphi$ has polynomial size and can be computed in polynomial time.
Circuit-SAT $\leq_p$ 3-SAT: Runtime of reduction

- Convert circuit $C$ to formula $\varphi$
  - **Input:** a circuit $C$
  - **Output:** a boolean formula in CNF
- make a variable $x_u$ for each node $u$ in the circuit
  - See example in the next slide
  - each clause contains at most 3 variables
- Turn clauses of at **most 3** literals into clauses of **exactly** 3 literals.
  - Add dummy variables:
    - Case 1: There is 1 variable in the clause, e.g., $y$
      - Change $y$ with: $(y \lor p \lor q) \land (y \lor p \lor \neg q) \land (y \lor \neg p \lor q) \land (y \lor \neg p \lor \neg q)$
    - Case 2: There are 2 variables in clause, e.g., $(x \lor y)$
      - Change $(x \lor y)$ with: $(x \lor y \lor p) \land (x \lor y \lor \neg p)$
- **Final formula:** $\varphi = \land$ of all clauses $\land x_{output}$

\text{At most one variable and 3 clause per node $\rightarrow O(n)$}

\text{expanded each clause by $\leq 4$ clauses $\rightarrow O(n)$}

\text{The size of the resulting formula is polynomial in the length of the original formula}
Circuit-SAT $\leq_p$ 3-SAT: Correctness of reduction

- **Claim.** C is satisfiable if and only if $\phi$ is satisfiable
  - $\Rightarrow$ Suppose C is satisfiable. Then assigning True/False to variables of $\phi$ according to C’s computation will satisfy $\phi$
Circuit-SAT $\leq_p$ 3-SAT: Correctness of reduction

- **Claim.** C is satisfiable if and only if $\phi$ is satisfiable
  - $\Leftarrow$ Suppose $\phi$ is satisfiable. Then there is an assignment of True/False to the variables (original inputs + new variables for circuit nodes) that makes $\phi$ True. For circuit C, use the same values for the input variables. By construction, the variables for the circuit nodes capture the evaluation of C. And $x_{\text{output}} = 1$ (True). Therefore C is satisfiable.