# NP-Hard and NP-Complete Problems

- NP-Hard and NP-Complete Problems
- A first NP-Complete Problem: circuit-sat($C$)
- The Cook-Levin Theorem: sat is NP-Complete
- NP-Complete Problems from known Reductions
- NP-Complete ness of dir-ham-cycle and ham-cycle
- tsp is NP-Complete
- subset-sum is NP-Complete
- partition is NP-Complete

Imdad ullah Khan

# NP-Hard and NP-Complete Problems

A problem $X$ is NP-Hard, if every problem in NP is polynomial time reducible to $X$

$$\forall\, Y \in \mathrm{NP}, \quad Y \ \leq_p \ X$$

A problem $X \in \mathrm{NP}$ is NP-Complete, if every problem in NP is polynomial time reducible to $X$

$$X \in \mathrm{NP} \quad \text{AND} \quad \forall\, Y \in \mathrm{NP}, \quad Y \ \leq_p \ X$$

These problems are at least as hard as any problem in NP

Let NPC be the (sub)class of NP-Complete problems

▷ It is the set of hardest problems in NP

If any NP-complete problem can be solved in poly time, then all problems in NP can be, and thus $\mathrm{P} = \mathrm{NP}$

# Proving NP-COMPLETE Problems

A problem $X$ is NP-COMPLETE, if

1. $X \in \mathrm{NP}$
2. $\forall\ Y \in \mathrm{NP}\ Y \leq_p X$

To prove $X$ NP-COMPLETE, reduce an NP-COMPLETE problem $Z$ to $X$

If $Z$ is NP-COMPLETE, and

1. $X \in \mathrm{NP}$
2. $Z \leq_p X$

then $X$ is NP-COMPLETE

1. $X \in \mathrm{NP}$ is explicitly proved
2. $\forall\ Y \in \mathrm{NP},\quad Y \leq_p X$ follows by transitivity

   $\forall\ Y \in \mathrm{NP},\quad Y \leq_p Z$ is true as $Z$ is NP-COMPLETE

   $[Y \leq_p Z \ \wedge\ Z \leq_p X] \implies Y \leq_p X$

# A first NP-Complete Problem

To prove $X$ NP-Complete, reduce an NP-Complete problem $Z$ to $X$

Where to begin? we need a first NP-Complete Problem

**Theorem (The Cook-Levin theorem)**

SAT($f$) *is* NP-Complete

- Proved by Stephen Cook (1971) and earlier by Leonid Levin (but became known later)
- Levin proved six NP-Complete problems (in addition to other results)

We will prove this theorem, but first we prove that the CIRCUIT-SAT($C$) problem is NP-Complete and reduce it to the SAT($f$) problem

# CIRCUIT-SAT($C$) is NP-COMPLETE

**CIRCUIT-SAT is NP-COMPLETE**

First we prove that it is polynomial time verifiable

**CIRCUIT-SAT($C$) $\in$ NP**

The instance $C$ is encoded as a DAG

1. A certificate can be assignment of Boolean values to input wires
2. Verifier finds topological order of $C$ and evaluate output of each gate (node)
3. If the value of sink node is 1, the verifier outputs **Yes**, otherwise **No**

Runtime is clearly polynomial

- Topological sort takes time polynomial in size of input graph
- So is linear scan of vertices to evaluate their value constant time on each

# CIRCUIT-SAT$(C)$ is NP-COMPLETE

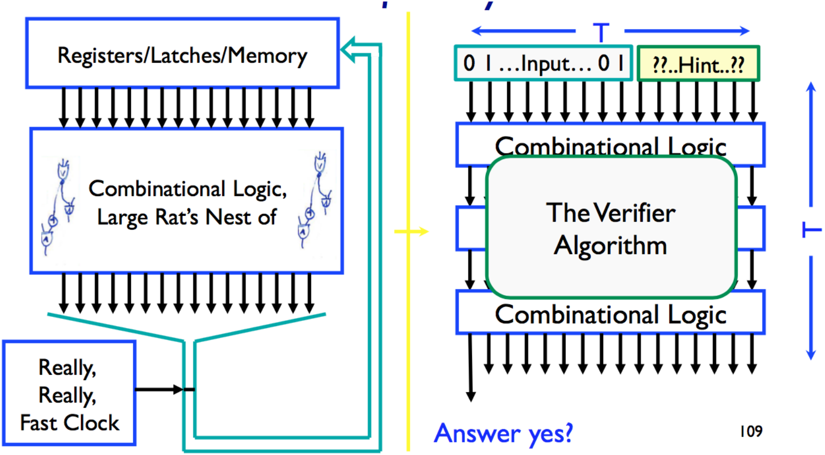
CIRCUIT-SAT$(C)$ is NP-HARD $\quad \forall\, X \in \mathrm{NP},\ X \leq_p$ CIRCUIT-SAT$(C)$

Use the definition of $X \in \mathrm{NP}$ critically

- There is a **poly-sized** certificate $S$ for every instance $\mathcal{I}$ of $X$ and a **poly-time** verifier $\mathcal{V}$ such that $\mathcal{V}(\mathcal{I}, S) =$ **Yes** iff $X(\mathcal{I}) =$ **Yes**
- $\mathcal{I}$ and $S$ have a binary encoding (in digital computers)
- $\mathcal{V}$ can be implemented in a digital computer, takes $\mathcal{I}$ and $S$ and outputs $\mathbf{1}/\mathbf{0}$ in **poly number** of clock cycles
- A computer has a configuration/state (values of all registers (memory), control registers, program counters etc.)
- State changes after each clock cycle according to instruction of $\mathcal{V}$ that are executed by a Boolean combinatorial circuit (the ALU)
- $\mathcal{V}$ outputs $\mathbf{1}/\mathbf{0}$ depending on the final state
- **Ignore the clock and replicate the circuit mapping states to next states**

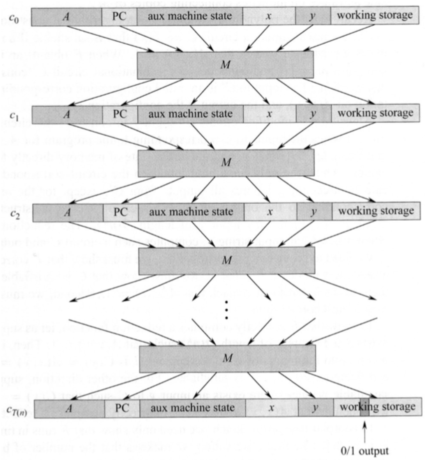
# CIRCUIT-SAT($C$) is NP-COMPLETE

CIRCUIT-SAT($C$) is NP-HARD  $\forall X \in$ NP, $X \leq_p$ CIRCUIT-SAT($C$)



# CIRCUIT-SAT($C$) is NP-COMPLETE

CIRCUIT-SAT($C$) is NP-HARD $\quad \forall X \in$ NP, $X \leq_p$ CIRCUIT-SAT($C$)



# CIRCUIT-SAT($C$) is NP-COMPLETE

**CIRCUIT-SAT($C$) is NP-HARD** $\forall X \in \text{NP}, \; X \leq_p$ CIRCUIT-SAT($C$)

Let $\mathcal{A}$ be an algorithm to decide the CIRCUIT-SAT($C$) problem

We use $\mathcal{A}$ to decide the problem $X \in \text{NP}$ on an instance $\mathcal{I}$

- Construct a circuit $C'$ from the digital implementation of $\mathcal{V}$ on input $\mathcal{I}$
- $\mathcal{A}(C') =$ **Yes** $\iff$ CIRCUIT-SAT($C'$) $=$ **Yes** means
  - there is an input for which $C'$ outputs **Yes**,
  - meaning there is a certificate $S$ (since $\mathcal{I}$ is hard-coded), on which the verifier $\mathcal{V}$ outputs **Yes**
- Since $\mathcal{V}(\mathcal{I}, S) =$ **Yes** $\iff$ $X(\mathcal{I}) =$ **Yes**, we get an answer for $X(\mathcal{I})$
- Number of stages in $C'$ is polynomial (equal to number of clock cycles, which is polynomial since $\mathcal{V}$ is polynomial time in $|\mathcal{I}|$)
- Number of gates in $C'$ is polynomial, hence constructing it takes poly time