# Foundations of Computing
## Lecture 21

Arkady Yerukhimovich

April 9, 2024

# Outline

1. Lecture 20 Review
2. A Review of $\mathcal{P}$ and $\mathcal{NP}$
3. Polynomial-Time Reductions
4. $\mathcal{NP}$-Completeness
5. $\mathcal{NP}$-Completeness Using Reductions

# Lecture 20 Review

- Verifying vs. Deciding
- The Complexity Class $\mathcal{NP}$

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

# Outline

1. Lecture 20 Review
2. A Review of $\mathcal{P}$ and $\mathcal{NP}$
3. Polynomial-Time Reductions
4. $\mathcal{NP}$-Completeness
5. $\mathcal{NP}$-Completeness Using Reductions

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case
- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case
- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:
  - For $x \in L$, there exists $w \in \{0,1\}^{poly(|x|)}$ s.t. $V(x,w) = 1$

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case
- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:
  - For $x \in L$, there exists $w \in \{0,1\}^{poly(|x|)}$ s.t. $V(x,w) = 1$
  - For $x \notin L$, for all $w \in \{0,1\}^{poly(|x|)}$, $V(x,w) = 0$

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case

- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:
  - For $x \in L$, there exists $w \in \{0,1\}^{poly(|x|)}$ s.t. $V(x,w) = 1$
  - For $x \notin L$, for all $w \in \{0,1\}^{poly(|x|)}$, $V(x,w) = 0$
  - $w$ is a witness to $x \in L$

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case
- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:
  - For $x \in L$, there exists $w \in \{0,1\}^{poly(|x|)}$ s.t. $V(x,w) = 1$
  - For $x \notin L$, for all $w \in \{0,1\}^{poly(|x|)}$, $V(x,w) = 0$
  - $w$ is a witness to $x \in L$

## Why Do We Study These?

# $\mathcal{P}$ and $\mathcal{NP}$

- $L \in \mathcal{P}$ if there is a poly-time DTM $M$ that decides $L$:
  - For $x \in L$, $M(x)$ halts and outputs 1
  - For $x \notin L$, $M(x)$ halts and outputs 0
  - Runtime of $M$ is $O(poly(|x|))$ for all $x$ – worst case
- $L \in \mathcal{NP}$ if there is a poly-time verifier DTM $V$:
  - For $x \in L$, there exists $w \in \{0,1\}^{poly(|x|)}$ s.t. $V(x, w) = 1$
  - For $x \notin L$, for all $w \in \{0,1\}^{poly(|x|)}$, $V(x, w) = 0$
  - $w$ is a witness to $x \in L$

**Why Do We Study These?**

Both $\mathcal{P}$ and $\mathcal{NP}$ contain many useful languages

# Why are $\mathcal{P}$ and $\mathcal{NP}$ Interesting?

## $\mathcal{P}$

- $\mathcal{P}$ captures the class of efficiently decidable languages

# Why are $\mathcal{P}$ and $\mathcal{NP}$ Interesting?

## $\mathcal{P}$

- $\mathcal{P}$ captures the class of efficiently decidable languages
- Can determine membership in $L$ for all inputs

# Why are $\mathcal{P}$ and $\mathcal{NP}$ Interesting?

## $\mathcal{P}$

- $\mathcal{P}$ captures the class of efficiently decidable languages
- Can determine membership in $L$ for all inputs

## $\mathcal{NP}$

- $\mathcal{NP}$ captures the class of problems where there exists a short proof that $x \in L$

# Why are $\mathcal{P}$ and $\mathcal{NP}$ Interesting?

## $\mathcal{P}$

- $\mathcal{P}$ captures the class of efficiently decidable languages
- Can determine membership in $L$ for all inputs

## $\mathcal{NP}$

- $\mathcal{NP}$ captures the class of problems where there exists a short proof that $x \in L$
- Can prove $x \in L$ for all inputs, but can't prove that a string not in $L$ is in the language

# Why are $\mathcal{P}$ and $\mathcal{NP}$ Interesting?

## $\mathcal{P}$

- $\mathcal{P}$ captures the class of efficiently decidable languages
- Can determine membership in $L$ for all inputs

## $\mathcal{NP}$

- $\mathcal{NP}$ captures the class of problems where there exists a short proof that $x \in L$
- Can prove $x \in L$ for all inputs, but can't prove that a string not in $L$ is in the language

## $\mathcal{NP}$-Completeness

There are problems in $\mathcal{NP}$ that are as hard as any other problem in $\mathcal{NP}$

# Outline

# Mapping Reductions

## Mapping Reduction

Language $A$ is mapping reducible to language $B$ ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

# Mapping Reductions

## Mapping Reduction

Language $A$ is mapping reducible to language $B$ ($A \leq_m B$) if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

## Poly-time Mapping Reduction

Language $A$ is poly-time mapping reducible to language $B$ ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

# Mapping Reductions

## Mapping Reduction

Language $A$ is mapping reducible to language $B$ ($A \leq_m B$) if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

## Poly-time Mapping Reduction

Language $A$ is poly-time mapping reducible to language $B$ ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

- Poly-time reductions give an efficient way to convert membership testing in $A$ to membership testing in $B$

# Mapping Reductions

## Mapping Reduction

Language $A$ is mapping reducible to language $B$ ($A \leq_m B$) if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

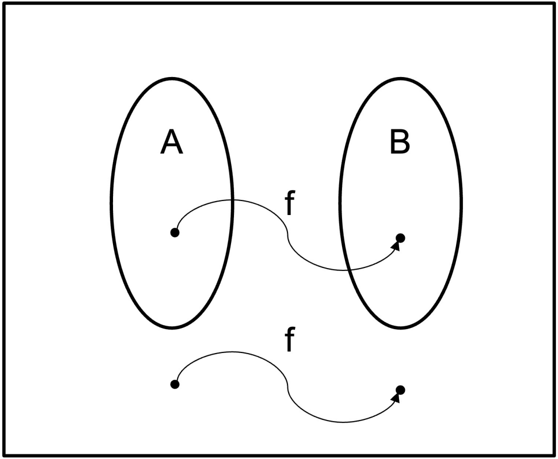
## Poly-time Mapping Reduction

Language $A$ is poly-time mapping reducible to language $B$ ($A \leq_P B$) if there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$, where for every $x$,

$$x \in A \iff f(x) \in B$$

- Poly-time reductions give an efficient way to convert membership testing in $A$ to membership testing in $B$
- If $B$ has a poly-time solution so does $A$

# Poly-time Mapping Reductions




$f$ runs in time $poly(|x|)$ on all inputs $x$

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$
- Can construct $M'$ deciding $A$:

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$
- Can construct $M'$ deciding $A$:
  $M'$ = On input $x$: Q: Is $x \in A$?
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever $M$ outputs

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$
- Can construct $M'$ deciding $A$:
  $M'$ = On input $x$:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever $M$ outputs
     - If $x \in A$, $f(x) \in B$ so $M$ accepts

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$
- Can construct $M'$ deciding $A$:
  $M' =$ On input $x$:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever $M$ outputs
  - If $x \in A$, $f(x) \in B$ so $M$ accepts
  - If $x \notin A$, $f(x) \notin B$, so $M$ rejects

# Why Poly-Time Reductions

**Theorem**

If $A \leq_P B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$

Proof:

- Let $M$ be the poly-time TM deciding $B$
- Let $f$ be the poly-time reduction from $A$ to $B$
- Can construct $M'$ deciding $A$:
  $M' =$ On input $x$:
  1. Compute $f(x)$
  2. Run $M(f(x))$ and output whatever $M$ outputs
  - If $x \in A$, $f(x) \in B$ so $M$ accepts
  - If $x \notin A$, $f(x) \notin B$, so $M$ rejects
  - Since both $f$ and $M$ are poly-time, $M(f(x))$ is also poly-time

# Using Poly-Time Reductions to Prove Hardness

**Theorem**

If $A \leq_P B$ and $A \notin \mathcal{P}$, then $B \notin \mathcal{P}$

1. Assume $B \in P \rightarrow \exists M$ that decides $B$

$M'(x)$

$M(f(x))$

# Outline

# $\mathcal{NP}$-Completeness

## Definition

A language $B$ is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

# $\mathcal{NP}$-Completeness

## Definition

A language $B$ is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- $B$ is "as hard" as any language in $\mathcal{NP}$

# $\mathcal{NP}$-Completeness

## Definition

A language $B$ is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- $B$ is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

# $\mathcal{NP}$-Completeness

## Definition

A language $B$ is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- $B$ is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

## Theorem

If $B$ is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

# $\mathcal{NP}$-Completeness

## Definition

A language $B$ is $\mathcal{NP}$-complete if

- $B \in \mathcal{NP}$
- For every language $A \in \mathcal{NP}$, $A \leq_P B$

- $B$ is "as hard" as any language in $\mathcal{NP}$
- To study hardness of $\mathcal{NP}$, enough to study hardness of some $\mathcal{NP}$-complete problem

## Theorem

If $B$ is $\mathcal{NP}$-complete and $B \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$

## Theorem

If $B$ is $\mathcal{NP}$-complete and $B \leq_P C$ for $C \in \mathcal{NP}$, then $C$ is $\mathcal{NP}$-complete

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
   - Need to design reduction $f$ from $A$ to $SAT$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
    - Need to design reduction $f$ from $A$ to $SAT$
    - $f$ takes an input $x$ and produces formula $\phi$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
    - Need to design reduction $f$ from $A$ to $SAT$
    - $f$ takes an input $x$ and produces formula $\phi$
        - If $x \in A$ then $\phi$ is satisfiable

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
    - Need to design reduction $f$ from $A$ to $SAT$
    - $f$ takes an input $x$ and produces formula $\phi$
        - If $x \in A$ then $\phi$ is satisfiable
        - If $x \notin A$ then $\phi$ is not satisfiable

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
   - Need to design reduction $f$ from $A$ to $SAT$
   - $f$ takes an input $x$ and produces formula $\phi$
     - If $x \in A$ then $\phi$ is satisfiable
     - If $x \notin A$ then $\phi$ is not satisfiable
   - Idea: Let $\phi$ be a formula simulating $\mathcal{NP}$ machine for $A$ on input $x$

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
   - Need to design reduction $f$ from $A$ to $SAT$
   - $f$ takes an input $x$ and produces formula $\phi$
     - If $x \in A$ then $\phi$ is satisfiable
     - If $x \notin A$ then $\phi$ is not satisfiable
   - Idea: Let $\phi$ be a formula simulating $\mathcal{NP}$ machine for $A$ on input $x$
     - That is, $\phi$ corresponds to the Boolean logic done by this machine

# SAT is $\mathcal{NP}$-Complete

## SAT Problem

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Proof Idea:

1. SAT$\in \mathcal{NP}$
2. For each $A \in \mathcal{NP}$, $A \leq_P SAT$
   - Need to design reduction $f$ from $A$ to $SAT$
   - $f$ takes an input $x$ and produces formula $\phi$
     - If $x \in A$ then $\phi$ is satisfiable
     - If $x \notin A$ then $\phi$ is not satisfiable
   - Idea: Let $\phi$ be a formula simulating $\mathcal{NP}$ machine for $A$ on input $x$
     - That is, $\phi$ corresponds to the Boolean logic done by this machine
     - Since any computation can be represented as a Boolean computation, this is always possible

# Execution of Turing Machine $M$ deciding $A$

start → poly(|x|)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # | $q_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $\sqcup$ | $\cdots$ | $\sqcup$ | # |
| # | | | | | | | | | # |
| # | | | | | | | | | # |
| | | | | | | | | | |
| # | | | | | | | | | # |

Table: Tableau of configurations of $M$

# Execution of Turing Machine $M$ deciding $A$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # | $q_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $\sqcup$ | $\cdots$ | $\sqcup$ | # |
| # | | | | | | | | | # |
| # | | | | | | | | | # |
| | | | | | | | | | |
| # | | | | | | | | | # |

Table: Tableau of configurations of $M$

- Every row is a configuration of $M$

# Execution of Turing Machine $M$ deciding $A$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # | $q_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $\sqcup$ | $\cdots$ | $\sqcup$ | # |
| # | | | | | | | | | # |
| # | | | | | | | | | # |
| | | | | | | | | | |
| # | | | | | | | | | # |

Table: Tableau of configurations of $M$

- Every row is a configuration of $M$
- Two consecutive rows represent a valid transition if they follow rules of $M$

# Execution of Turing Machine $M$ deciding $A$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\#$ | $q_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $\sqcup$ | $\cdots$ | $\sqcup$ | $\#$ |
| $\#$ | $x_1$ | $q_1$ | $x_2$ | | | | | | $\#$ |
| $\#$ | | | | | | | | | $\#$ |
| | | | | | | | | | |
| $\#$ | | | | | | | | | $\#$ |

Table: Tableau of configurations of $M$

- Every row is a configuration of $M$
- Two consecutive rows represent a valid transition if they follow rules of $M$
- Every cell contains $\#$, or a state $q \in Q$, or a tape symbol $\in \Gamma$

# Execution of Turing Machine $M$ deciding $A$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\#$ | $q_0$ | $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $\sqcup$ | $\cdots$ | $\sqcup$ | $\#$ |
| $\#$ | | | | | | | | | $\#$ |
| $\#$ | | | | | | | | | $\#$ |
| | | | | | | | | | |
| $\#$ | | | | | | | | | $\#$ |

Table: Tableau of configurations of $M$

- Every row is a configuration of $M$
- Two consecutive rows represent a valid transition if they follow rules of $M$
- Every cell contains $\#$, or a state $q \in Q$, or a tape symbol $\in \Gamma$
- $M$ accepts $x$ if a row of this tableau is in $q_{accept}$

# SAT is $\mathcal{NP}$-Complete

Given input $x$ that we want to check if $x \in A$
We need to build a formula $\phi$ that checks the following four things:

# SAT is $\mathcal{NP}$-Complete

Given input $x$ that we want to check if $x \in A$

We need to build a formula $\phi$ that checks the following four things:

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$

# SAT is $\mathcal{NP}$-Complete

Given input $x$ that we want to check if $x \in A$

We need to build a formula $\phi$ that checks the following four things:

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$
2. Top row is the start configuration (on input $x$)

# SAT is $\mathcal{NP}$-Complete

Given input $x$ that we want to check if $x \in A$
We need to build a formula $\phi$ that checks the following four things:

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$
2. Top row is the start configuration (on input $x$)
3. Some row is in $q_{accept}$

# SAT is $\mathcal{NP}$-Complete

Given input $x$ that we want to check if $x \in A$

We need to build a formula $\phi$ that checks the following four things:

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$
2. Top row is the start configuration (on input $x$)
3. Some row is in $q_{accept}$
4. Every pair of adjacent rows represents a valid transition of $M$

# SAT is $\mathcal{NP}$-Complete

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$

# SAT is $\mathcal{NP}$-Complete

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$
   - For $1 \leq i, j \leq n^k$, and $s \in C$, let $x_{i,j,s} = 1$ if $cell[i,j] = s$

# SAT is $\mathcal{NP}$-Complete

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$

- For $1 \leq i, j \leq n^k$, and $s \in C$, let $x_{i,j,s} = 1$ if $cell[i,j] = s$
- The following equation $\phi_{i,j}^{cell}$ guarantees that a cell has a valid value

cell[i,j] ≠ s or cell[i,j] ≠ t

$$\phi_{i,j}^{cell} = \underbrace{\left(\bigvee_{s \in C} x_{i,j,s}\right)}_{\text{cell } i,j \text{ has at least 1 value}} \wedge \underbrace{\left(\bigwedge_{s,t \in C, s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})\right)}_{\text{cell } i,j \text{ has at most 1 value}}$$

# SAT is $\mathcal{NP}$-Complete

1. Every cell contains a valid character in $C = Q \bigcup \Gamma \bigcup \{\#\}$

- For $1 \leq i, j \leq n^k$, and $s \in C$, let $x_{i,j,s} = 1$ if $cell[i,j] = s$
- The following equation $\phi_{i,j}^{cell}$ guarantees that a cell has a valid value

$$\phi_{i,j}^{cell} = \underbrace{\left( \bigvee_{s \in C} x_{i,j,s} \right)}_{\text{cell } i,j \text{ has at least 1 value}} \wedge \underbrace{\left( \bigwedge_{s,t \in C, s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right)}_{\text{cell } i,j \text{ has at most 1 value}}$$

- Now, we just take the AND over all $n^{2k}$ cells in the tableau

# SAT is $\mathcal{NP}$-Complete

2. Top row is the start configuration on input x

# SAT is $\mathcal{NP}$-Complete

2. Top row is the start configuration
   - Define a formula $\phi_{start}$ that checks that all the cells in the top row are correct

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \cdots \wedge x_{1,n^k,\#}$$

$x_{i,j,c} = 1$ if $cell[i,j] = c$

$\#\ q_0\ x_1\ x_2\ x_n\ \sqcup\ \sqcup\ \#$

# SAT is $\mathcal{NP}$-Complete

3. Some row is in $q_{accept}$

# SAT is $\mathcal{NP}$-Complete

3. Some row is in $q_{accept}$
   - Define a formula $\phi_{accept}$ that checks that some row contains $q_{accept}$

$$\phi_{accept} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{accept}}$$

# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$

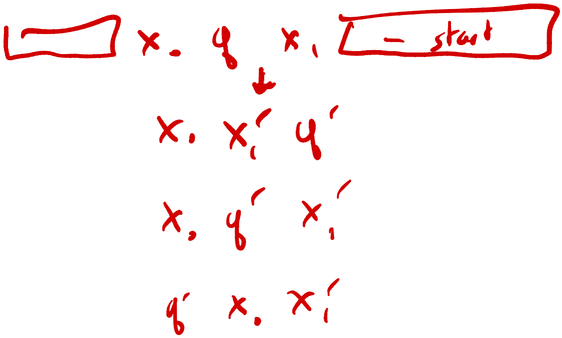
# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$

- We need to define what is a valid move between two configurations

# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$
   - We need to define what is a valid move between two configurations
   - If the control head is not next to some cell, that cell will not change



# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$

- We need to define what is a valid move between two configurations
- If the control head is not next to some cell, that cell will not change
- For cells after control head, can write to the cell and move left or right (depending on $M$)

# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$

- We need to define what is a valid move between two configurations
- If the control head is not next to some cell, that cell will not change
- For cells after control head, can write to the cell and move left or right (depending on $M$)
- Every $2 \times 3$ cell window can be checked to follow these rules

# SAT is $\mathcal{NP}$-Complete

4. Every pair of adjacent rows represents a valid transition of $M$
   - We need to define what is a valid move between two configurations
   - If the control head is not next to some cell, that cell will not change
   - For cells after control head, can write to the cell and move left or right (depending on $M$)
   - Every $2 \times 3$ cell window can be checked to follow these rules
   - Now just take the $\wedge$ over all possible 6-cell windows

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)

$x_{i,j,s} = 1$ if cell[i, j] = s

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells
- $\phi_{cell}$ has fixed size for each cell, so $O(n^{2k})$ total

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells
- $\phi_{cell}$ has fixed size for each cell, so $O(n^{2k})$ total
- $\phi_{start}$ has fixed size for each cell in top row, so $O(n^k)$ total

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells
- $\phi_{cell}$ has fixed size for each cell, so $O(n^{2k})$ total
- $\phi_{start}$ has fixed size for each cell in top row, so $O(n^k)$ total
- $\phi_{move}$ and $\phi_{accept}$ have fixed size for each cell, so $O(n^{2k})$ total

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells
- $\phi_{cell}$ has fixed size for each cell, so $O(n^{2k})$ total
- $\phi_{start}$ has fixed size for each cell in top row, so $O(n^k)$ total
- $\phi_{move}$ and $\phi_{accept}$ have fixed size for each cell, so $O(n^{2k})$ total
- Summing up, we see $|\phi| = O(n^{2k})$

# SAT is $\mathcal{NP}$-Complete

- Finally, need to check that this reduction runs in poly time (in $n = |x|$)
- Recall that the tableau has size $n^k \times n^k$, so $n^{2k}$ cells
- $\phi_{cell}$ has fixed size for each cell, so $O(n^{2k})$ total
- $\phi_{start}$ has fixed size for each cell in top row, so $O(n^k)$ total
- $\phi_{move}$ and $\phi_{accept}$ have fixed size for each cell, so $O(n^{2k})$ total
- Summing up, we see $|\phi| = O(n^{2k})$
- Since $k = O(1)$, this is polynomial in $n$

# Outline

1. Lecture 20 Review
2. A Review of $\mathcal{P}$ and $\mathcal{NP}$
3. Polynomial-Time Reductions
4. $\mathcal{NP}$-Completeness
5. **$\mathcal{NP}$-Completeness Using Reductions**

$SAT \le_p A$

$A \in NP$

$\Rightarrow$

A is NP-complete

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

## 3-CNF formulas

- A literal is a (possibly negates) Boolean variable – $x$ or $\overline{x}$

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

## 3-CNF formulas

- A literal is a (possibly negates) Boolean variable – $x$ or $\overline{x}$
- A clause is several literals connected with $\vee$'s – $x_1 \vee \overline{x_2} \vee x_3$

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

## 3-CNF formulas

- A literal is a (possibly negates) Boolean variable – $x$ or $\overline{x}$
- A clause is several literals connected with $\vee$'s – $x_1 \vee \overline{x_2} \vee x_3$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses connected by $\wedge$'s

$$(x_1 \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee x_5)$$

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

## 3-CNF formulas

- A literal is a (possibly negates) Boolean variable – $x$ or $\overline{x}$
- A clause is several literals connected with $\vee$'s – $x_1 \vee \overline{x_2} \vee x_3$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses connected by $\wedge$'s

$$(x_1 \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee x_5)$$

- A Boolean formula is a 3-CNF if all the clauses have exactly 3 literals

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee x_4 \vee x_2)$$

# The 3SAT Problem

- Recall that SAT asks if a Boolean formula has a satisfying assignment
- 3SAT asks the same question for 3-CNF formulas

## 3-CNF formulas

- A literal is a (possibly negates) Boolean variable – $x$ or $\overline{x}$
- A clause is several literals connected with $\vee$'s – $x_1 \vee \overline{x_2} \vee x_3$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses connected by $\wedge$'s

$$(x_1 \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee x_5)$$

- A Boolean formula is a 3-CNF if all the clauses have exactly 3 literals

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee x_4 \vee x_2)$$

## 3-SAT

$$\text{3SAT} = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3-CNF formula}\}$$

Can show that 3SAT is $\mathcal{NP}$-complete using similar proof to SAT

# 3SAT $\leq_P$ CLIQUE

- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where

# 3SAT $\leq_P$ CLIQUE

- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$

# 3SAT $\leq_P$ CLIQUE

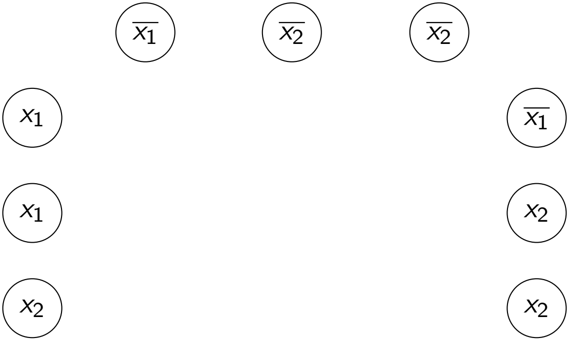
- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$
  - If $\phi$ is not satisfiable, $G$ has no clique of size $k$

# 3SAT $\leq_P$ CLIQUE

- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$
  - If $\phi$ is not satisfiable, $G$ has no clique of size $k$
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

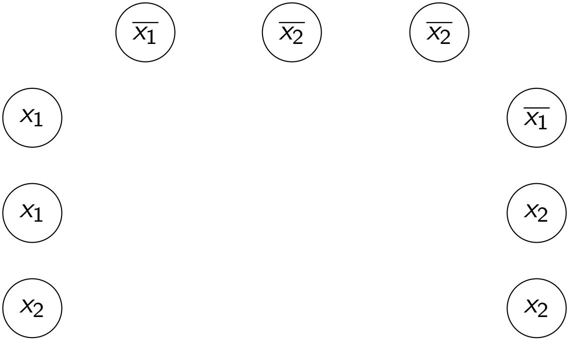
# 3SAT $\leq_P$ CLIQUE

- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$
  - If $\phi$ is not satisfiable, $G$ has no clique of size $k$
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$



# 3SAT $\leq_P$ CLIQUE

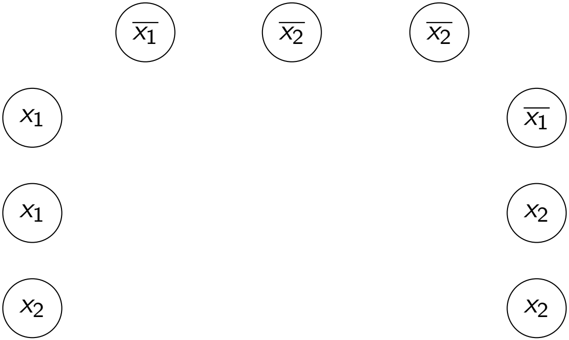
- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k\rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$
  - If $\phi$ is not satisfiable, $G$ has no clique of size $k$
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$




- If $\phi$ is satisfiable then $G$ has a $k$-clique

# 3SAT $\leq_P$ CLIQUE

- Need to show reduction $f$ from 3SAT formula $\phi$ to $\langle G, k \rangle$ where
  - If $\phi$ is satisfiable, $G$ has a clique of size $k$
  - If $\phi$ is not satisfiable, $G$ has no clique of size $k$
- Consider $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$




- If $\phi$ is satisfiable then $G$ has a $k$-clique
- If $G$ has a $k$-clique then $\phi$ is satisfiable