NP-Completeness

Algorithm Design & Analysis

Fall 2018
Outline
Recap

• $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
Recap

• $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.

• Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$
Recap

• $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.

• Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$

• Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$

• Strategies for reductions
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$
- Strategies for reductions
  - Direct equivalence ($\text{INDSET} \equiv_p \text{VERTEXCOVER}$)
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$
- Strategies for reductions
  - Direct equivalence ($INDSET \equiv_p VERTEXCOVER$)
  - Special Case: ($VERTEXCOVER \leq_p SETCOVER$)
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$.
- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$.
- Strategies for reductions:
  - Direct equivalence ($INDSET \equiv_p VERTEXCOVER$).
  - Special Case: ($VERTEXCOVER \leq_p SETCOVER$).
  - Gadget Building: ($3SAT \leq_p INDSET$).
Recap

• $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.

• Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$

• Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$

• Strategies for reductions
  • Direct equivalence ($\text{INDSET} \equiv_p \text{VERTEXCOVER}$)
  • Special Case: ($\text{VERTEXCOVER} \leq_p \text{SETCOVER}$)
  • Gadget Building: ($\text{3SAT} \leq_p \text{INDSET}$)

• Problem Characteristics
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.

- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$

- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$

- Strategies for reductions
  - Direct equivalence ($INDSET \equiv_p VERTEXCOVER$)
  - Special Case: ($VERTEXCOVER \leq_p SETCOVER$)
  - Gadget Building: ($3SAT \leq_p INDSET$)

- Problem Characteristics
  - Decision Problems: Output is YES/NO
Recap

- $X \leq_p Y$ if problem $X$ can be solved in polynomial time by some algorithm that is allowed to solve instances of problem $Y$ in constant time.
- Polynomial Equivalence: $X \equiv_p Y$ if $X \leq_p Y$ and $Y \leq_p X$
- Transitivity: If $X \leq_p Y$ and $Y \leq_p Z$ then $X \leq_p Z$
- Strategies for reductions
  - Direct equivalence ($\text{INDSET} \equiv_p \text{VERTEXCOVER}$)
  - Special Case: ($\text{VERTEXCOVER} \leq_p \text{SETCOVER}$)
  - Gadget Building: ($\text{3SAT} \leq_p \text{INDSET}$)
- Problem Characteristics
  - Decision Problems: Output is YES/NO
  - Certifiability: If answer is YES, there's a "short proof"
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
- An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()

- An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$
  - $s \in X$ if and only if there is some string $t_s$ such that $C(s, t_s)$ returns "yes".
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
- An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$
  - $s \in X$ if and only if there is some string $t_s$ such that $C(s, t_s)$ returns "yes".
- A certifier $C(s, t)$ for decision problem $X$ is a polynomial-time certifier if
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
- An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$
  - $s \in X$ if and only if there is some string $t_s$ such that $C(s, t_s)$ returns "yes".
- A certifier $C(s, t)$ for decision problem $X$ is a polynomial-time certifier if
  - $|t_s| \leq p(|s|)$ for some polynomial $p()$ ($t_s$ is not too big!)
Recap: Decision Problems and Certifiers

- Algorithm $A$ solves decision problem $X$ in polynomial time if $A(s)$ executes at most $O(p(|s|))$ operations, for some polynomial $p()$
- An algorithm $C(s, t)$, where $s$ and $t$ are strings, is a certifier for decision problem $X$ if for every $s$
  - $s \in X$ if and only if there is some string $t_s$ such that $C(s, t_s)$ returns "yes".
- A certifier $C(s, t)$ for decision problem $X$ is a polynomial-time certifier if
  - $|t_s| \leq p(|s|)$ for some polynomial $p()$ ($t_s$ is not too big!)
  - $C(s, t)$ runs in time $q(|s|)$ for some polynomial $q(x)$ ($C()$ is efficient)
A Complexity Hierarchy

- $P = \{X : \text{There is a poly-time algorithm } A() \text{ that decides } X\}$
- $NP = \{X : \text{There is a poly-time certifier } C(s, t) \text{ for } X\}$
A Complexity Hierarchy

- $P = \{X : \text{There is a poly-time algorithm } A() \text{ that decides } X\}$
- $NP = \{X : \text{There is a poly-time certifier } C(s, t) \text{ for } X\}$
- Claim: $P \subseteq NP$: $C(s, t) = A(s)$; just let $t_s = \epsilon$
- $\text{EXP} = \{X : \text{Some exp-time algorithm } A() \text{ that decides } X\}$
- Claim: $NP \subseteq \text{EXP}$
- $P \subset \text{EXP}$: Consequence of the Time Hierarchy Theorem
- Big Question: Is $P = NP$?
- Consensus view is "no"
- Most fundamental problem in computer science
- Clay Foundation offers $1,000,000$ prize for the answer
A Complexity Hierarchy

- $P = \{X : \text{There is a poly-time algorithm } A() \text{ that decides } X\}$
- $NP = \{X : \text{There is a poly-time certifier } C(s, t) \text{ for } X\}$
- Claim: $P \subseteq NP$: $C(s, t) = A(s)$; just let $t_s = \epsilon$
- $EXP = \{X : \text{Some exp-time algorithm } A() \text{ that decides } X\}$
A Complexity Hierarchy

• \( P = \{ X : \) There is a poly-time algorithm \( A() \) that decides \( X \}\}
• \( NP = \{ X : \) There is a poly-time certifier \( C(s, t) \) for \( X \}\}
• Claim: \( P \subseteq NP: C(s, t) = A(s); \) just let \( t_s = \epsilon \)
• \( EXP = \{ X : \) Some exp-time algorithm \( A() \) that decides \( X \}\}
• Claim: \( NP \subseteq EXP \)
A Complexity Hierarchy

- $P = \{ X : \text{There is a poly-time algorithm } A() \text{ that decides } X \}$
- $NP = \{ X : \text{There is a poly-time certifier } C(s, t) \text{ for } X \}$
- Claim: $P \subseteq NP$: $C(s, t) = A(s)$; just let $t_s = \epsilon$
- $EXP = \{ X : \text{Some exp-time algorithm } A() \text{ that decides } X \}$
- Claim: $NP \subseteq EXP$
- $P \nsubseteq EXP$: Consequence of the Time Hierarchy Theorem
A Complexity Hierarchy

- \( P = \{ X : \) There is a poly-time algorithm \( A() \) that decides \( X \}\)
- \( NP = \{ X : \) There is a poly-time certifier \( C(s, t) \) for \( X \}\)
- Claim: \( P \subseteq NP: C(s, t) = A(s); \) just let \( t_s = \epsilon \)
- \( EXP = \{ X : \) Some exp-time algorithm \( A() \) that decides \( X \}\)
- Claim: \( NP \subseteq EXP \)
- \( P \subsetneq EXP: \) Consequence of the Time Hierarchy Theorem
- **Big Question:** Is \( P = NP \)?
A Complexity Hierarchy

- \( P = \{ X : \text{There is a poly-time algorithm } A() \text{ that decides } X \} \)
- \( NP = \{ X : \text{There is a poly-time certifier } C(s, t) \text{ for } X \} \)
- Claim: \( P \subseteq NP: \ C(s, t) = A(s); \text{ just let } t_s = \epsilon \)
- \( EXP = \{ X : \text{Some exp-time algorithm } A() \text{ that decides } X \} \)
- Claim: \( NP \subseteq EXP \)
- \( P \subsetneq EXP: \text{Consequence of the Time Hierarchy Theorem} \)
- **Big Question:** Is \( P = NP \)?
  - Consensus view is "no"
A Complexity Hierarchy

- $P = \{X : \text{There is a poly-time algorithm } A() \text{ that decides } X\}$
- $NP = \{X : \text{There is a poly-time certifier } C(s, t) \text{ for } X\}$
- Claim: $P \subseteq NP$: $C(s, t) = A(s)$; just let $t_s = \epsilon$
- $EXP = \{X : \text{Some exp-time algorithm } A() \text{ that decides } X\}$
- Claim: $NP \subseteq EXP$
- $P \subset EXP$: Consequence of the Time Hierarchy Theorem
- Big Question: Is $P = NP$?
  - Consensus view is "no"
  - Most fundamental problem in computer science

The Clay Foundation offers $1,000,000 prize for the answer.
A Complexity Hierarchy

- $P = \{X : \text{There is a poly-time algorithm } A() \text{ that decides } X\}$
- $NP = \{X : \text{There is a poly-time certifier } C(s, t) \text{ for } X\}$
- Claim: $P \subseteq NP$: $C(s, t) = A(s)$; just let $t_s = \epsilon$
- $EXP = \{X : \text{Some exp-time algorithm } A() \text{ that decides } X\}$
- Claim: $NP \subseteq EXP$
- $P \subsetneq EXP$: Consequence of the Time Hierarchy Theorem
- **Big Question:** Is $P = NP$?
  - Consensus view is "no"
  - Most fundamental problem in computer science
  - Clay Foundation offers $1,000,000$ prize for the answer
NP-Completeness Defined

Are there "hardest" problems in NP?
NP-Completeness Defined

Are there "hardest" problems in NP?

Not obvious, but since $NP \subseteq EXP$, at least possible
Are there "hardest" problems in NP?

Not obvious, but since $NP \subseteq EXP$, at least possible

**Definition**

A decision problem $X$ is *NP-Complete* if
NP-Completeness Defined

Are there "hardest" problems in NP?

Not obvious, but since $NP \subseteq EXP$, at least possible

Definition

A decision problem $X$ is $NP$-Complete if

- $X \in NP$
Are there "hardest" problems in NP?

Not obvious, but since $NP \subseteq EXP$, at least possible

**Definition**

A decision problem $X$ is *NP-Complete* if

- $X \in NP$
- For every $Y \in NP$, $Y \leq_p X$
Are there "hardest" problems in NP?
Not obvious, but since $NP \subseteq EXP$, at least possible

Definition
A decision problem $X$ is $NP$-Complete if
- $X \in NP$
- For every $Y \in NP$, $Y \leq_p X$

Are there any such problems?
**NP-Completeness Defined**

Are there "hardest" problems in NP?
Not obvious, but since $NP \subseteq EXP$, at least possible

**Definition**
A decision problem $X$ is *NP-Complete* if
- $X \in NP$
- For every $Y \in NP$, $Y \leq_p X$

Are there *any* such problems?

Surprisingly, *thousands* of problems have been shown to be NP-Complete
Why is Definition Important?

Theorem

Let $Y$ be any NP-Complete problem. Then $Y \in P$ if and only if $P = NP$.

Proof.

$P = NP \implies Y \in P$ 

• Clear: $Y \in NP$, so $Y \in P$.

$Y \in P \implies P = NP$ 

• Let $X \in NP$. Then $X \leq_P Y$, since $Y$ is NP-Complete.

• But if $Y \in P$ then $X \in P$.

• Thus $NP \subseteq P$. But $P \subseteq NP$, so $P = NP$. 

Why is Definition Important?

**Theorem**

Let $Y$ be any *NP-Complete problem*. Then $Y \in P$ if and only if $P = NP$.
Theorem
Let $Y$ be any \textit{NP-Complete} problem. Then $Y \in P$ if and only if $P = NP$

Proof.
Why is Definition Important?

**Theorem**

Let $Y$ be any NP-Complete problem. Then $Y \in P$ if and only if $P = NP$

**Proof.**

$P = NP \Rightarrow Y \in P$
Why is Definition Important?

**Theorem**

Let $Y$ be any NP-Complete problem. Then $Y \in P$ if and only if $P = NP$

**Proof.**

$P = NP \Rightarrow Y \in P$

- Clear: $Y \in NP$, so $Y \in P$
Why is Definition Important?

Theorem
Let $Y$ be any NP-Complete problem. Then $Y \in P$ if and only if $P = NP$

Proof.

$P = NP \Rightarrow Y \in P$

- Clear: $Y \in NP$, so $Y \in P$

$Y \in P \Rightarrow P = NP$
**Why is Definition Important?**

*Theorem*

Let $Y$ be any *NP-Complete* problem. Then $Y \in P$ if and only if $P = NP$

*Proof.*

$P = NP \Rightarrow Y \in P$

- Clear: $Y \in NP$, so $Y \in P$

$Y \in P \Rightarrow P = NP$

- Let $X \in NP$. Then $X \leq_p Y$, since $Y$ is *NP-Complete*
Why is Definition Important?

Theorem
Let $Y$ be any $NP$-Complete problem. Then $Y \in P$ if and only if $P = NP$

Proof.

$P = NP \Rightarrow Y \in P$

- Clear: $Y \in NP$, so $Y \in P$

$Y \in P \Rightarrow P = NP$

- Let $X \in NP$. Then $X \leq_p Y$, since $Y$ is $NP$-Complete
- But if $Y \in P$ then $X \in P$. 
Why is Definition Important?

**Theorem**
Let Y be any NP-Complete problem. Then $Y \in P$ if and only if $P = NP$

**Proof.**

$P = NP \Rightarrow Y \in P$

- Clear: $Y \in NP$, so $Y \in P$

$Y \in P \Rightarrow P = NP$

- Let $X \in NP$. Then $X \leq_p Y$, since $Y$ is NP-Complete
- But if $Y \in P$ then $X \in P$.
- Thus $NP \subseteq P$. But $P \subseteq NP$, so $P = NP$
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete:

1. **From Definition**
   - Show that $Y \in NP$
   - Show that for all $X \in NP$, $X \leq_p Y$

2. **Reduction**
   - Show that $Y \in NP$
   - Show that $Z \leq_p Y$ for some NP-Complete problem $Z$

So if $X \in NP$, $X \leq_p Z$ and $Z \leq_p Y$, so $X \leq_p Y$.

Can't use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

From Definition
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

*From Definition*

1. Show that $Y \in NP$

Can't use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

*From Definition*

- Show that $Y \in NP$
- Show that for all $X \in NP$, $X \leq_p Y$

Can’t use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

From Definition

- Show that $Y \in NP$
- Show that for all $X \in NP$, $X \leq_p Y$

Can’t use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

**From Definition**

- Show that $Y \in NP$
- Show that for all $X \in NP$, $X \leq_p Y$

**Reduction**

- Show that $Y \in NP$

Can’t use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

*From Definition*

- Show that $Y \in NP$
- Show that for all $X \in NP$, $X \leq_p Y$

*Reduction*

- Show that $Y \in NP$
- Show that $Z \leq_p Y$ for some NP-Complete problem $Z$

Can't use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem $Y$ is NP-Complete

*From Definition*

- Show that $Y \in NP$
- Show that for all $X \in NP$, $X \leq_p Y$

*Reduction*

- Show that $Y \in NP$
- Show that $Z \leq_p Y$ for some NP-Complete problem $Z$
- So if $X \in NP$, $X \leq_p Z$ and $Z \leq_p Y$, so $X \leq_p Y$

Can't use second method until we use first method!
Establishing NP-Completeness

There are two ways to show a problem \( Y \) is NP-Complete

*From Definition*

- Show that \( Y \in NP \)
- Show that for all \( X \in NP \), \( X \leq_p Y \)

*Reduction*

- Show that \( Y \in NP \)
- Show that \( Z \leq_p Y \) for some for some NP-Complete problem \( Z \)
- So if \( X \in NP \), \( X \leq_p Z \) and \( Z \leq_p Y \), so \( X \leq_p Y \)

Can’t use second method until we use first method!
Circuit Satisfiability : A First NP-Complete Problem

We will show the following

• \textsc{CircuitSat} is NP-Complete
• \textsc{CircuitSat} \leq_p \textsc{Atmost3Sat}, so \textsc{Atmost3Sat} is NP-Complete (\textsc{Atmost3Sat} \in NP)
• \textsc{Atmost3Sat} \leq_p \textsc{3Sat}, so \textsc{3Sat} is NP-Complete (\textsc{3Sat} \in NP)

This will show that \textsc{Indset}, \textsc{VertexCover}, \textsc{Setcover} are NP-Complete, since

• They are all in NP, and
• \textsc{3Sat} \leq_p \textsc{Indset} \leq_p \textsc{VertexCover} \leq_p \textsc{Setcover}

From these, an avalanche of NP-Complete problems will follow
We will show the following:

- CIRCUITSAT is NP-Complete
We will show the following

- CIRCUITSAT is NP-Complete
- CIRCUITSAT $\leq_p$ ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT $\in$ NP)
We will show the following

- CIRCUITSAT is NP-Complete
- CIRCUITSAT \( \leq_p \) ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT \( \in \) NP)
- ATMOST3SAT \( \leq_p \) 3SAT, so 3SAT is NP-Complete (3SAT \( \in \) NP)

From these, an avalanche of NP-Complete problems will follow
Circuit Satisfiability: A First NP-Complete Problem

We will show the following

- CIRCUITSAT is NP-Complete
- CIRCUITSAT $\leq_p$ ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT $\in$ NP)
- ATMOST3SAT $\leq_p$ 3SAT, so 3SAT is NP-Complete (3SAT $\in$ NP)
- This will show that INDSET, VERTEXCOVER, SETCOVER are NP-Complete, since
Circuit Satisfiability : A First NP-Complete Problem

We will show the following

• CIRCUITSAT is NP-Complete
• CIRCUITSAT $\leq_p$ ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT $\in$ NP)
• ATMOST3SAT $\leq_p$ 3SAT, so 3SAT is NP-Complete (3SAT $\in$ NP)
• This will show that INDSET, VERTEXCOVER, SETCOVER are NP-Complete, since
  • They are all in NP, and
We will show the following

- CIRCUITSAT is NP-Complete
- CIRCUITSAT $\leq_p$ ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT $\in$ NP)
- ATMOST3SAT $\leq_p$ 3SAT, so 3SAT is NP-Complete (3SAT $\in$ NP)
- This will show that INDSET, VERTEXCOVER, SETCOVER are NP-Complete, since
  - They are all in NP, and
  - 3SAT $\leq_p$ INDSET $\leq_p$ VERTEXCOVER $\leq_p$ SETCOVER
Circuit Satisfiability: A First NP-Complete Problem

We will show the following

- CIRCUITSAT is NP-Complete
- CIRCUITSAT \leq_p ATMOST3SAT, so ATMOST3SAT is NP-Complete (ATMOST3SAT \in NP)
- ATMOST3SAT \leq_p 3SAT, so 3SAT is NP-Complete (3SAT \in NP)
- This will show that INDSET, VERTEXCOVER, SETCOVER are NP-Complete, since
  - They are all in NP, and
  - 3SAT \leq_p INDSET \leq_p VERTEXCOVER \leq_p SETCOVER
- From these, an avalanche of NP-Complete problems will follow
A Note on Alphabets and Strings

An alphabet $\Sigma$ is just a finite set. The inputs to decision problems are strings over some alphabet $\Sigma$. We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$).

A language $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$. So a decision problem $X$ is just a language over $\Sigma$. It doesn't really matter which alphabet we choose.

If $|\Sigma| = k$, any $\sigma \in \Sigma$ can be encoded with $c = \lceil \log k \rceil$ bits.

So if $s \in \Sigma^*$ has length $n$, its bit-encoding has length $cn$ over $\{0, 1\}$.

So alphabet choice impacts efficiency by only a constant factor.
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set

The inputs to decision problems are strings over some alphabet $\Sigma$
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set

The inputs to decision problems are strings over some alphabet $\Sigma$

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$)
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set.

The inputs to decision problems are strings over some alphabet $\Sigma$.

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$).

A *language* $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$.
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set.

The inputs to decision problems are strings over some alphabet $\Sigma$.

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$).

A language $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$.

So a decision problem $X$ is just a language over $\Sigma$. 
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set

The inputs to decision problems are strings over some alphabet $\Sigma$

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$)

A language $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$

So a decision problem $X$ is just a language over $\Sigma$

It doesn’t really matter which alphabet we choose
A Note on Alphabets and Strings

An *alphabet* $\Sigma$ is (just) a finite set

The inputs to decision problems are strings over some alphabet $\Sigma$

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$)

A *language* $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$

So a decision problem $X$ is just a language over $\Sigma$

It doesn’t really matter which alphabet we choose

If $|\Sigma| = k$, any $\sigma \in \Sigma$ can be encoded with $c = \lceil \log k \rceil$ bits
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set.

The inputs to decision problems are strings over some alphabet $\Sigma$.

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$).

A language $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$.

So a decision problem $X$ is just a language over $\Sigma$.

It doesn’t really matter which alphabet we choose.

If $|\Sigma| = k$, any $\sigma \in \Sigma$ can be encoded with $c = \lceil \log k \rceil$ bits.

So if $s \in \Sigma^*$ has length $n$, its bit-encoding has length $cn$ over $\{0, 1\}$.
A Note on Alphabets and Strings

An alphabet $\Sigma$ is (just) a finite set.

The inputs to decision problems are strings over some alphabet $\Sigma$.

We write $\Sigma^*$ for the set of all finite strings over $\Sigma$ (including $\epsilon$).

A language $X$ over $\Sigma$ is just some $X \subseteq \Sigma^*$.

So a decision problem $X$ is just a language over $\Sigma$.

It doesn’t really matter which alphabet we choose.

If $|\Sigma| = k$, any $\sigma \in \Sigma$ can be encoded with $c = \lceil \log k \rceil$ bits.

So if $s \in \Sigma^*$ has length $n$, its bit-encoding has length $cn$ over $\{0, 1\}$.

So alphabet choice impacts efficiency by only a constant factor.
Boolean Circuits: An Example

We will assume that $\Sigma = \{0, 1\}$ in what follows.
Boolean Circuits: An Example

We will assume that $\Sigma = \{0, 1\}$ in what follows.

![Diagram of a Boolean circuit with three inputs, two additional sources with assigned truth values, and one output.](image)

**Figure 8.4** A circuit with three inputs, two additional sources that have assigned truth values, and one output.
Boolean Circuits

Definition
A boolean circuit is a DAG in which

• Sources represent input bits
• Sinks represent output bits
• Other bits represent boolean operations (∧, ∨, ¬)

Theorem (We won’t prove this....)
Let A be a poly-time algorithm that takes n input bits and produces 1 output bit. Then there is a boolean circuit C such that
• C can be produced from A in poly-time (and hence is of poly-size)
• C produces a 1 if and only if A does
**Boolean Circuits**

*Definition*

A *boolean circuit* is a DAG in which

- Sources represent input bits
Boolean Circuits

Definition

A boolean circuit is a DAG in which

- Sources represent input bits
- Sinks represent output bits

Theorem (We won’t prove this....)

Let A be a poly-time algorithm that takes n input bits and produces 1 output bit. Then there is a boolean circuit C such that

- C can be produced from A in poly-time (and hence is of poly-size)
- C produces a 1 if and only if A does
Boolean Circuits

Definition

A boolean circuit is a DAG in which

- Sources represent input bits
- Sinks represent output bits
- Other bits represent boolean operations ($\wedge, \vee, \neg$)

Theorem (We won’t prove this....)

Let A be a poly-time algorithm that takes n input bits and produces 1 output bit. Then there is a boolean circuit C such that

- C can be produced from A in poly-time (and hence is of poly-size)
- C produces a 1 if and only if A does
Boolean Circuits

Definition

A boolean circuit is a DAG in which

• Sources represent input bits
• Sinks represent output bits
• Other bits represent boolean operations ($\land, \lor, \neg$)

Theorem (We won’t prove this....)

Let A be a poly-time algorithm that takes n input bits and produces 1 output bit. Then there is a boolean circuit C such that
Boolean Circuits

Definition

A boolean circuit is a DAG in which

- Sources represent input bits
- Sinks represent output bits
- Other bits represent boolean operations ($\land, \lor, \neg$)

Theorem (We won’t prove this....)

Let A be a poly-time algorithm that takes n input bits and produces 1 output bit. Then there is a boolean circuit C such that

- C can be produced from A in poly-time (and hence is of poly-size)
**Boolean Circuits**

**Definition**
A boolean circuit is a DAG in which
- Sources represent input bits
- Sinks represent output bits
- Other bits represent boolean operations ($\wedge, \vee, \neg$)

**Theorem (We won’t prove this....)**
Let $A$ be a poly-time algorithm that takes $n$ input bits and produces 1 output bit. Then there is a boolean circuit $C$ such that
- $C$ can be produced from $A$ in poly-time (and hence is of poly-size)
- $C$ produces a 1 if and only if $A$ does
Definition (CIRCUITSAT)

Given a boolean circuit $C$ with $n$ input bits (some of which may be fixed), is there an assignment of values to the unfixed input bits such that $C$ returns 1 (true/yes)?
**Definition (CIRCUITSAT)**

Given a boolean circuit $C$ with $n$ input bits (some of which may be fixed), is there an assignment of values to the unfixed input bits such that $C$ returns 1 (true/yes)?

**Theorem**

*CIRCUITSAT is NP-complete*
Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.
CIRCUITSAT is NP-Complete
CIRCUITSAT is NP-Complete

Proof.
CIRCUITSAT is NP-Complete

Proof.
Need to show that, for any $X \in NP$, $X \leq_p CIRCUITSAT$
CIRCUITSAT is NP-Complete

Proof.
Need to show that, for any $X \in NP$, $X \leq_p CIRCUITSAT$

- Since $X \in NP$, $X$ has a poly-time certifier $C(s, t)$
CIRCUITSAT is NP-Complete

Proof.

Need to show that, for any $X \in NP$, $X \leq_p CIRCUITSAT$

- Since $X \in NP$, $X$ has a poly-time certifier $C(s, t)$
- A string $s$ is in $X$ if and only if some $t_s$ of length $p(|s|)$ makes $C(s, t_s)$ return "yes" (that is, 1).
CIRCUITSAT is NP-Complete

Proof.
Need to show that, for any $X \in NP$, $X \leq_p CIRCUITSAT$

- Since $X \in NP$, $X$ has a poly-time certifier $C(s, t)$
- A string $s$ is in $X$ if and only if some $t_s$ of length $p(|s|)$ makes $C(s, t_s)$ return "yes" (that is, 1).
- So view $C(s, t)$ as an algorithm that takes at most $|s| + p(|s|)$ bits and outputs 1 bit
CIRCUITSAT is NP-Complete

Proof.

Need to show that, for any $X \in NP$, $X \leq_p CIRCUITSAT$

- Since $X \in NP$, $X$ has a poly-time certifier $C(s, t)$
- A string $s$ is in $X$ if and only if some $t_s$ of length $p(|s|)$ makes $C(s, t_s)$ return "yes" (that is, 1).
- So view $C(s, t)$ as an algorithm that takes at most $|s| + p(|s|)$ bits and outputs 1 bit
- $C(s, t)$ can be converted into a boolean circuit $C$ with $|s|$ fixed bits; other $p(|s|)$ bits represent $t_s$
CIRCUITSAT is NP-Complete

Proof.

Need to show that, for any $X \in NP, X \leq_p CIRCUITSAT$

- Since $X \in NP$, $X$ has a poly-time certifier $C(s, t)$
- A string $s$ is in $X$ if and only if some $t_s$ of length $p(|s|)$ makes $C(s, t_s)$ return "yes" (that is, 1).
- So view $C(s, t)$ as an algorithm that takes at most $|s| + p(|s|)$ bits and outputs 1 bit
- $C(s, t)$ can be converted into a boolean circuit $C$ with $|s|$ fixed bits; other $p(|s|)$ bits represent $t_s$
CIRCUITSAT is NP-Complete

Proof.
Need to show that, for any \( X \in NP \), \( X \leq_p CIRCUITSAT \)

- Since \( X \in NP \), \( X \) has a poly-time certifier \( C(s, t) \)
- A string \( s \) is in \( X \) if and only if some \( t_s \) of length \( p(|s|) \) makes \( C(s, t_s) \) return "yes" (that is, 1).
- So view \( C(s, t) \) as an algorithm that takes at most \(|s| + p(|s|)\) bits and outputs 1 bit
- \( C(s, t) \) can be converted into a boolean circuit \( C \) with \(|s|\) fixed bits; other \( p(|s|) \) bits represent \( t_s \)
- \( C \) is satisfiable if and only if there’s some setting of \( t_s \) bits that makes \( C(s, t_s) \) true.
CIRCUITSAT is NP-Complete

Proof.

Need to show that, for any \( X \in NP \), \( X \leq_p CIRCUITSAT \)

- Since \( X \in NP \), \( X \) has a poly-time certifier \( C(s, t) \)
- A string \( s \) is in \( X \) if and only if some \( t_s \) of length \( p(|s|) \) makes \( C(s, t_s) \) return "yes" (that is, 1).
- So view \( C(s, t) \) as an algorithm that takes at most \(|s| + p(|s|)\) bits and outputs 1 bit
- \( C(s, t) \) can be converted into a boolean circuit \( C \) with \(|s|\) fixed bits; other \( p(|s|) \) bits represent \( t_s \)
- \( C \) is satisfiable if and only if there's some setting of \( t_s \) bits that makes \( C(s, t_s) \) true.
- Thus \( X \) has been poly-time reduced to an instance of CIRCUITSAT
From CIRCUITSAT to ATMOST3SAT

Definition
Let $\Phi$ be a CNF expression with at most 3 literals per clause. ATMOST3SAT is the problem of deciding whether $\Phi$ is satisfiable.
Definition
Let $\Phi$ be a CNF expression with at most 3 literals per clause. ATMOST3SAT is the problem of deciding whether $\Phi$ is satisfiable.

Theorem
ATMOST3SAT is NP-complete.
From \textit{CIRCUITSAT} to \textit{ATMOST3SAT}

\textbf{Definition}
Let $\Phi$ be a CNF expression with at most 3 literals per clause. \textit{ATMOST3SAT} is the problem of deciding whether $\Phi$ is satisfiable.

\textbf{Theorem}
\textit{ATMOST3SAT} is \textit{NP}-complete.

\textbf{Proof.}
Note that \textit{ATMOST3SAT} is in \textit{NP}. We show that \textit{CIRCUITSAT} $\leq_p$ \textit{ATMOST3SAT}.
From CIRCUITSAT to ATMOST3SAT

**Definition**
Let $\Phi$ be a CNF expression with at most 3 literals per clause. ATMOST3SAT is the problem of deciding whether $\Phi$ is satisfiable.

**Theorem**
ATMOST3SAT is NP-complete.

**Proof.**
Note that ATMOST3SAT is in NP. We show that CIRCUITSAT $\leq_p$ ATMOST3SAT.

- Let $C$ be a boolean circuit. We’ll build $\Phi_C$ such that $\Phi_C$ is satisfiable if and only if $C$ is satisfiable.
Proof that CIRCUITSAT $\leq_p$ ATMOST3SAT

Proof.
Proof that $\text{CIRCUITSAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\overline{v})$ otherwise
Proof that \( \text{CIRCUITSAT} \leq_p \text{ATMOST3SAT} \)

**Proof.**

- For each fixed bit source \( v \), create clause \( (v) \) if value is 1 and \( (\overline{v}) \) otherwise.
- For output bit (sink) \( v_{\text{final}} \), create clause \( (v_{\text{final}}) \) to force output bit to be 1.
Proof that $\text{CIRCUITSAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\overline{v})$ otherwise
- For output bit (sink) $v_{final}$, create clause $(v_{final})$ to force output bit to be 1
- For each internal node $v$
Proof that $\text{CIRCUITSAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\overline{v})$ otherwise.
- For output bit (sink) $v_{\text{final}}$, create clause $(v_{\text{final}})$ to force output bit to be 1.
- For each internal node $v$:
  - If $v$ is a $\neg$ gate from $u$, create clauses $(v \lor u) \land (\overline{v} \lor \overline{u})$. 

If $\Phi$ is satisfiable, $(v_{\text{final}})$ has value 1, and all fixed source variables received their correct values, and all other source variables received values that make $C$ produce 1.

If $C$ is satisfiable, $\Phi$ is satisfiable.
Proof that $\text{CIRCUIT SAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\overline{v})$ otherwise
- For output bit (sink) $v_{final}$, create clause $(v_{final})$ to force output bit to be 1
- For each internal node $v$
  - If $v$ is a $\neg$ gate from $u$, create clauses $(v \lor u) \land (\overline{v} \lor \overline{u})$
  - If $v$ is a $\lor$ gate from $u$ and $w$, create clauses $(v \lor \overline{u}) \land (v \lor \overline{w}) \land (\overline{v} \lor u \lor w)$
Proof that $\text{CIRCUITSAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\bar{v})$ otherwise
- For output bit (sink) $v_{final}$, create clause $(v_{final})$ to force output bit to be 1
- For each internal node $v$
  - If $v$ is a $\neg$ gate from $u$, create clauses $(v \lor u) \land (\bar{v} \lor \bar{u})$
  - If $v$ is a $\lor$ gate from $u$ and $w$, create clauses $(v \lor \bar{u}) \land (v \lor \bar{w}) \land (\bar{v} \lor u \lor w)$
  - If $v$ is a $\land$ gate from $u$ and $w$, create clauses $(\bar{v} \lor u) \land (\bar{v} \lor w) \land (v \lor \bar{u} \lor \bar{w})$
Proof that CIRCUITSAT $\leq_p$ ATMOST3SAT

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\bar{v})$ otherwise
- For output bit (sink) $v_{final}$, create clause $(v_{final})$ to force output bit to be 1
- For each internal node $v$
  - If $v$ is a $\neg$ gate from $u$, create clauses $(v \lor u) \land (\bar{v} \lor \bar{u})$
  - If $v$ is a $\lor$ gate from $u$ and $w$, create clauses $(v \lor \bar{u}) \land (v \lor \bar{w}) \land (\bar{v} \lor u \lor w)$
  - If $v$ is a $\land$ gate from $u$ and $w$, create clauses $(\bar{v} \lor u) \land (\bar{v} \lor w) \land (v \lor \bar{u} \lor \bar{w})$
  - If $\Phi$ is satisfiable, $(v_{final})$ has value 1, and all fixed source variables received their correct values, and all other source variables received values that make $C$ produce 1
Proof that $\text{CIRCUITSAT} \leq_p \text{ATMOST3SAT}$

Proof.

- For each fixed bit source $v$, create clause $(v)$ if value is 1 and $(\bar{v})$ otherwise
- For output bit (sink) $v_{final}$, create clause $(v_{final})$ to force output bit to be 1
- For each internal node $v$
  - If $v$ is a $\neg$ gate from $u$, create clauses $(v \lor u) \land (\bar{v} \lor \bar{u})$
  - If $v$ is a $\lor$ gate from $u$ and $w$, create clauses $(v \lor \bar{u}) \land (v \lor \bar{w}) \land (\bar{v} \lor u \lor w)$
  - If $v$ is a $\land$ gate from $u$ and $w$, create clauses $(\bar{v} \lor u) \land (\bar{v} \lor w) \land (v \lor \bar{u} \lor \bar{w})$
- If $\Phi$ is satisfiable, $(v_{final})$ has value 1, and all fixed source variables received their correct values, and all other source variables received values that make $C$ produce 1
- If $C$ is satisfiable, $\Phi$ is satisfiable