Part I

Reductions Continued
Propositional Formulas

**Definition**

Consider a set of boolean variables $x_1, x_2, \ldots, x_n$.

1. A *literal* is either a boolean variable $x_i$ or its negation $\neg x_i$.
2. A *clause* is a disjunction of literals. For example, $x_1 \lor x_2 \lor \neg x_4$ is a clause.
3. A *formula in conjunctive normal form (CNF)* is a propositional formula which is a conjunction of clauses
   
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5\] is a CNF formula.

4. A formula $\varphi$ is a 3CNF:
   A CNF formula such that every clause has exactly 3 literals.
   
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_1)\] is a 3CNF formula, but
   
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5\] is not.
Consider a set of boolean variables $x_1, x_2, \ldots, x_n$.

1. A **literal** is either a boolean variable $x_i$ or its negation $\neg x_i$.

2. A **clause** is a disjunction of literals.
   For example, $x_1 \lor x_2 \lor \neg x_4$ is a clause.

3. A **formula in conjunctive normal form (CNF)** is propositional formula which is a conjunction of clauses
   
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5\]
   is a **CNF** formula.

4. A formula $\varphi$ is a **3CNF**:
   A **CNF** formula such that every clause has **exactly** 3 literals.
   
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_1)\]
   is a **3CNF** formula, but
   \[(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5\]
   is not.
Satisfiability

SAT

**Instance**: A CNF formula $\varphi$.
**Question**: Is there a truth assignment to the variable of $\varphi$ such that $\varphi$ evaluates to true?

3SAT

**Instance**: A 3CNF formula $\varphi$.
**Question**: Is there a truth assignment to the variable of $\varphi$ such that $\varphi$ evaluates to true?
Satisfiability

**SAT**

Given a **CNF** formula \( \varphi \), is there a truth assignment to variables such that \( \varphi \) evaluates to true?

**Example**

1. \((x_1 \lor x_2 \lor \lnot x_4) \land (x_2 \lor \lnot x_3) \land x_5\) is satisfiable; take \(x_1, x_2, \ldots, x_5\) to be all true

2. \((x_1 \lor \lnot x_2) \land (\lnot x_1 \lor x_2) \land (\lnot x_1 \lor \lnot x_2) \land (x_1 \lor x_2)\) is not satisfiable.

**3SAT**

Given a **3CNF** formula \( \varphi \), is there a truth assignment to variables such that \( \varphi \) evaluates to true?

(More on **2SAT** in a bit...)
Importance of **SAT** and **3SAT**

1. **SAT** and **3SAT** are basic constraint satisfaction problems.
2. Many different problems can reduced to them because of the simple yet powerful expressively of logical constraints.
3. Arise naturally in many applications involving hardware and software verification and correctness.
4. As we will see, it is a fundamental problem in theory of **NP-Completeness**.
\textbf{SAT} \ \leq_p \ \textbf{3SAT}

**How \textbf{SAT} is different from \textbf{3SAT}?**

In \textbf{SAT} clauses might have arbitrary length: \(1, 2, 3, \ldots\) variables:

\[(x \lor y \lor z \lor w \lor u) \land (\neg x \lor \neg y \lor \neg z \lor w \lor u) \land (\neg x)\]

In \textbf{3SAT} every clause must have \textit{exactly} 3 different literals.

To reduce from an instance of \textbf{SAT} to an instance of \textbf{3SAT}, we must make all clauses to have exactly 3 variables...

**Basic idea**

1. Pad short clauses so they have 3 literals.
2. Break long clauses into shorter clauses.
3. Repeat the above till we have a 3CNF.
How SAT is different from 3SAT?

In SAT clauses might have arbitrary length: 1, 2, 3, ... variables:

\[(x \lor y \lor z \lor w \lor u) \land (\neg x \lor \neg y \lor \neg z \lor w \lor u) \land (\neg x)\]

In 3SAT every clause must have exactly 3 different literals.

To reduce from an instance of SAT to an instance of 3SAT, we must make all clauses to have exactly 3 variables...

Basic idea

1. Pad short clauses so they have 3 literals.
2. Break long clauses into shorter clauses.
3. Repeat the above till we have a 3CNF.
3SAT $\leq_p$ SAT

1. 3SAT $\leq_p$ SAT.

2. Because...
   A 3SAT instance is also an instance of SAT.
Claim

\(\textbf{SAT} \leq_p \textbf{3SAT} \).

Given \(\varphi\) a \textbf{SAT} formula we create a \textbf{3SAT} formula \(\varphi'\) such that

1. \(\varphi\) is satisfiable iff \(\varphi'\) is satisfiable.
2. \(\varphi'\) can be constructed from \(\varphi\) in time polynomial in \(|\varphi|\).

Idea: if a clause of \(\varphi\) is not of length 3, replace it with several clauses of length exactly 3.
Claim

$\text{SAT} \leq_p \text{3SAT}$.

Given $\varphi$ a SAT formula we create a 3SAT formula $\varphi'$ such that

1. $\varphi$ is satisfiable iff $\varphi'$ is satisfiable.
2. $\varphi'$ can be constructed from $\varphi$ in time polynomial in $|\varphi|$.

Idea: if a clause of $\varphi$ is not of length 3, replace it with several clauses of length exactly 3.
Claim

$\text{SAT} \leq_P \text{3SAT}$.

Given $\varphi$ a \text{SAT} formula we create a \text{3SAT} formula $\varphi'$ such that

1. $\varphi$ is satisfiable iff $\varphi'$ is satisfiable.
2. $\varphi'$ can be constructed from $\varphi$ in time polynomial in $|\varphi|$.

\text{Idea:} if a clause of $\varphi$ is not of length 3, replace it with several clauses of length exactly 3.
Reduction Ideas

Challenge: Some of the clauses in $\varphi$ may have less or more than 3 literals. For each clause with $< 3$ or $> 3$ literals, we will construct a set of logically equivalent clauses.

1. Case clause with one literal: Let $c$ be a clause with a single literal (i.e., $c = \ell$). Let $u, v$ be new variables. Consider

$$c' = (\ell \lor u \lor v) \land (\ell \lor u \lor \neg v) \land (\ell \lor \neg u \lor v) \land (\ell \lor \neg u \lor \neg v).$$

Observe that $c'$ is satisfiable iff $c$ is satisfiable.
SAT \leq_p 3SAT

A clause with two literals

Reduction Ideas: 2 and more literals

Case clause with 2 literals: Let \( c = \ell_1 \lor \ell_2 \). Let \( u \) be a new variable. Consider

\[
    c' = (\ell_1 \lor \ell_2 \lor u) \land (\ell_1 \lor \ell_2 \lor \neg u).
\]

Again \( c \) is satisfiable iff \( c' \) is satisfiable
Breaking a clause

Lemma

For any boolean formulas $X$ and $Y$ and $z$ a new boolean variable. Then

$$X \lor Y \text{ is satisfiable}$$

if and only if, $z$ can be assigned a value such that

$$(X \lor z) \land (Y \lor \neg z) \text{ is satisfiable}$$

(with the same assignment to the variables appearing in $X$ and $Y$).
Let $c = \ell_1 \lor \cdots \lor \ell_k$. Let $u_1, \ldots, u_{k-3}$ be new variables. Consider

$$c' = (\ell_1 \lor \ell_2 \lor u_1) \land (\ell_3 \lor \neg u_1 \lor u_2)$$

$$\land (\ell_4 \lor \neg u_2 \lor u_3) \land$$

$$\cdots \land (\ell_{k-2} \lor \neg u_{k-4} \lor u_{k-3}) \land (\ell_{k-1} \lor \ell_k \lor \neg u_{k-3}).$$

### Claim

$c$ is satisfiable if and only if $c'$ is satisfiable.

Another way to see it — reduce size of clause by one:

$$c' = (\ell_1 \lor \ell_2 \ldots \lor \ell_{k-2} \lor u_{k-3}) \land (\ell_{k-1} \lor \ell_k \lor \neg u_{k-3}).$$
An Example

Example

\[ \varphi = (\neg x_1 \lor \neg x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4 \lor x_1) \land (x_1) . \]

Equivalent form:

\[ \psi = (\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor y_1) \land (x_4 \lor x_1 \lor \neg y_1) \land (x_1 \lor u \lor v) \land (x_1 \lor u \lor \neg v) \land (x_1 \lor \neg u \lor v) \land (x_1 \lor \neg u \lor \neg v) . \]
\[ \varphi = (\neg x_1 \lor \neg x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4 \lor x_1) \land (x_1). \]

Equivalent form:

\[ \psi = (\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor y_1) \land (x_4 \lor x_1 \lor \neg y_1) \land (x_1 \lor u \lor v) \land (x_1 \lor \neg u \lor \neg v) \land (x_1 \lor \neg u \lor \neg v). \]
Example

\[ \varphi = \left( \neg x_1 \lor \neg x_4 \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor x_4 \lor x_1 \right) \land (x_1) . \]

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor y_1 \right) \land \left( x_4 \lor x_1 \lor \neg y_1 \right) \land (x_1) \land (x_1) . \]
An Example

Example

\[ \varphi = (\neg x_1 \lor \neg x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4 \lor x_1) \land (x_1). \]

Equivalent form:

\[ \psi = (\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor y_1) \land (x_4 \lor x_1 \lor \neg y_1) \land (x_1 \lor u \lor v) \land (x_1 \lor \neg u \lor \neg v) \land (x_1 \lor \neg u \lor \neg v). \]
Overall Reduction Algorithm

Reduction from **SAT** to **3SAT**

**ReduceSATTo3SAT**(φ):

// φ: CNF formula.
for each clause $c$ of φ do
  if $c$ does not have exactly 3 literals then
    construct $c'$ as before
  else
    $c' = c$

$\psi$ is conjunction of all $c'$ constructed in loop
return Solver3SAT($\psi$)

**Correctness (informal)**

φ is satisfiable iff $\psi$ is satisfiable because for each clause $c$, the new 3CNF formula $c'$ is logically equivalent to $c$. 
What about 2SAT?

2SAT can be solved in polynomial time! (specifically, linear time!)

No known polynomial time reduction from SAT (or 3SAT) to 2SAT. If there was, then SAT and 3SAT would be solvable in polynomial time.

Why the reduction from 3SAT to 2SAT fails?

Consider a clause \((x \lor y \lor z)\). We need to reduce it to a collection of 2CNF clauses. Introduce a fake variable \(\alpha\), and rewrite this as

\[
(x \lor y \lor \alpha) \land (\neg \alpha \lor z) \quad \text{(bad! clause with 3 vars)}
\]

or

\[
(x \lor \alpha) \land (\neg \alpha \lor y \lor z) \quad \text{(bad! clause with 3 vars)}.
\]
What about \textbf{2SAT}?

A challenging exercise: Given a \textbf{2SAT} formula show to compute its satisfying assignment...

(Hint: Create a graph with two vertices for each variable (for a variable $x$ there would be two vertices with labels $x = 0$ and $x = 1$). For every \textbf{2CNF} clause add two directed edges in the graph. The edges are implication edges: They state that if you decide to assign a certain value to a variable, then you must assign a certain value to some other variable. Now compute the strong connected components in this graph, and continue from there...)
**Independent Set**

**Instance:** A graph $G$, integer $k$.

**Question:** Is there an independent set in $G$ of size $k$?
The reduction $3\text{SAT} \leq_P \text{Independent Set}$

**Input:** Given a $3\text{CNF}$ formula $\varphi$

**Goal:** Construct a graph $G_\varphi$ and number $k$ such that $G_\varphi$ has an independent set of size $k$ if and only if $\varphi$ is satisfiable.

$G_\varphi$ should be constructable in time polynomial in size of $\varphi$

Importance of reduction: Although $3\text{SAT}$ is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

Notice: We handle only $3\text{CNF}$ formulas – reduction would not work for other kinds of boolean formulas.
The reduction $3SAT \leq_P Independent\ Set$

**Input:** Given a 3CNF formula $\varphi$

**Goal:** Construct a graph $G_\varphi$ and number $k$ such that $G_\varphi$ has an independent set of size $k$ if and only if $\varphi$ is satisfiable.

$G_\varphi$ should be constructable in time polynomial in size of $\varphi$.

**Importance of reduction:** Although $3SAT$ is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

**Notice:** We handle only 3CNF formulas – reduction would not work for other kinds of boolean formulas.
**3SAT \( \leq_P \) Independent Set**

**The reduction** \( 3SAT \leq_P \) Independent Set

**Input:** Given a 3CNF formula \( \varphi \)

**Goal:** Construct a graph \( G_\varphi \) and number \( k \) such that \( G_\varphi \) has an independent set of size \( k \) if and only if \( \varphi \) is satisfiable.

\( G_\varphi \) should be constructable in time polynomial in size of \( \varphi \)

**Importance of reduction:** Although 3SAT is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

**Notice:** We handle only 3CNF formulas – reduction would not work for other kinds of boolean formulas.
Interpreting 3SAT

There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick $x_i$ and $\neg x_i$.

We will take the second view of 3SAT to construct the reduction.
Interpreting 3SAT

There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick $x_i$ and $\neg x_i$.

We will take the second view of 3SAT to construct the reduction.
There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick $x_i$ and $\neg x_i$

We will take the second view of 3SAT to construct the reduction.
Interpreting 3SAT

There are two ways to think about 3SAT

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick $x_i$ and $\neg x_i$

We will take the second view of 3SAT to construct the reduction.
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take $k$ to be the number of clauses

Figure: Graph for

$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take $k$ to be the number of clauses

Figure: Graph for

\[ \varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4) \]
The Reduction

1. \( G_\varphi \) will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take \( k \) to be the number of clauses

Figure: Graph for

\[ \varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4) \]
The Reduction

1. \( G_\varphi \) will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take \( k \) to be the number of clauses

Figure: Graph for
\[
\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)
\]
The Reduction

1. $G_\varphi$ will have one vertex for each literal in a clause.
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true.
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict.
4. Take $k$ to be the number of clauses.

Figure: Graph for

$$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$$
**Proposition**

φ is satisfiable if and only if $G_\varphi$ has an independent set of size $k$ ($= \text{number of clauses in } \varphi$).

**Proof.**

$\Rightarrow$ Let $a$ be the truth assignment satisfying $\varphi$

1. Pick one of the vertices, corresponding to true literals under $a$, from each triangle. This is an independent set of the appropriate size.
Correctness

Proposition

φ is satisfiable if and only if $G_\varphi$ has an independent set of size $k$ ($=$ number of clauses in $\varphi$).

Proof.

⇒ Let $a$ be the truth assignment satisfying $\varphi$

1 Pick one of the vertices, corresponding to true literals under $a$, from each triangle. This is an independent set of the appropriate size
Proposition

φ is satisfiable iff $G_\varphi$ has an independent set of size $k$ (i.e., number of clauses in φ).

Proof.

$\Leftarrow$ Let $S$ be an independent set of size $k$

1. $S$ must contain exactly one vertex from each clause
2. $S$ cannot contain vertices labeled by conflicting clauses
3. Thus, it is possible to obtain a truth assignment that makes in the literals in $S$ true; such an assignment satisfies one literal in every clause
Transitivity of Reductions

Lemma

\[ X \leq_P Y \text{ and } Y \leq_P Z \implies X \leq_P Z. \]

Note: \( X \leq_P Y \) does not imply that \( Y \leq_P X \) and hence it is very important to know the FROM and TO in a reduction.

To prove \( X \leq_P Y \) you need to show a reduction FROM \( X \) TO \( Y \) in other words show that an algorithm for \( Y \) implies an algorithm for \( X \).
Part II

Definition of NP
## Recap

### Problems

1. **Clique**
2. **Independent Set**
3. **Vertex Cover**

1. **Set Cover**
2. **SAT**
3. **3SAT**
Recap...

Problems
1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship
Independent Set $\leq_P$ Clique
### Recap

#### Problems

1. **Clique**
2. **Independent Set**
3. **Vertex Cover**

1. **Set Cover**
2. **SAT**
3. **3SAT**

#### Relationship

Independent Set $\leq_p$ Clique
Recap

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\leq_{P}$ Clique $\leq_{P}$ Independent Set
Recap ...

Problems
1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship
Independent Set $\approx_P$ Clique
## Recap

### Problems

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clique</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Independent Set</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Vertex Cover</td>
<td>3</td>
</tr>
</tbody>
</table>

### Relationship

- Independent Set $\approx_P$ Clique
- Independent Set $\leq_P$ Vertex Cover
## Recap

### Problems

1. **Clique**
2. **Independent Set**
3. **Vertex Cover**
4. **Set Cover**
5. **SAT**
6. **3SAT**

### Relationship

- Independent Set \( \approx_p \) Clique
- Independent Set \( \leq_p \) Vertex Cover \( \leq_p \) Independent Set
Recap ...

<table>
<thead>
<tr>
<th>Problems</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Clique</td>
<td>Independent Set $\approx_P$ Clique</td>
</tr>
<tr>
<td>2. Independent Set</td>
<td>Clique $\approx_P$ Vertex Cover</td>
</tr>
<tr>
<td>3. Vertex Cover</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set Cover $\approx_P$ SAT</td>
</tr>
<tr>
<td></td>
<td>SAT $\approx_P$ 3SAT</td>
</tr>
</tbody>
</table>
### Recap

#### Problems

| 1 | Clique       |
| 2 | Independent Set       |
| 3 | Vertex Cover       |

| 1 | Set Cover       |
| 2 | SAT       |
| 3 | 3SAT       |

#### Relationship

\[
\text{Vertex Cover} \approx_P \text{Independent Set} \approx_P \text{Clique}
\]
Recap

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover \(\approx_P\) Independent Set \(\approx_P\) Clique
3SAT \(\leq_P\) SAT
## Recap

### Problems

1. **Clique**  
2. **Independent Set**  
3. **Vertex Cover**

1. **Set Cover**  
2. **SAT**  
3. **3SAT**

### Relationship

- $\text{Vertex Cover} \approx_P \text{Independent Set} \approx_P \text{Clique}$
- $3\text{SAT} \leq_P \text{SAT} \leq_P 3\text{SAT}$
### Recap

#### Problems

- 1. Clique
- 2. Independent Set
- 3. Vertex Cover
- 1. Set Cover
- 2. SAT
- 3. 3SAT

#### Relationship

- $\text{Vertex Cover} \approx_P \text{Independent Set} \approx_P \text{Clique}$
- $\text{3SAT} \approx_P \text{SAT}$
### Recap

---

#### Problems

<table>
<thead>
<tr>
<th></th>
<th>Clique</th>
<th>Independent Set</th>
<th>Vertex Cover</th>
<th>Set Cover</th>
<th>SAT</th>
<th>3SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Relationship

- Vertex Cover $\approx_P$ Independent Set $\approx_P$ Clique
- $3\text{SAT} \approx_P SAT$
- $3\text{SAT} \leq_P$ Independent Set
Decision Problems

1. **Problem Instance:** Binary string $s$, with size $|s|$
2. **Problem:** A set $X$ of strings on which the answer should be "yes"; we call these YES instances of $X$. Strings not in $X$ are NO instances of $X$.

Definition

1. A is an **algorithm for problem** $X$ if $A(s) = \text{"yes"}$ iff $s \in X$.
2. A is said to have a **polynomial running time** if there is a polynomial $p(\cdot)$ such that for every string $s$, $A(s)$ terminates in at most $O(p(|s|))$ steps.
**Definition**

*Polynomial time* (denoted by $P$) is the class of all (decision) problems that have an algorithm that solves it in polynomial time.
**Definition**

*Polynomial time* (denoted by $\mathbb{P}$) is the class of all (decision) problems that have an algorithm that solves it in polynomial time.

**Example**

Problems in $\mathbb{P}$ include

1. Is there a shortest path from $s$ to $t$ of length $\leq k$ in $G$?
2. Is there a flow of value $\geq k$ in network $G$?
3. Is there an assignment to variables to satisfy given linear constraints?
Efficiency Hypothesis

A problem $X$ has an efficient algorithm iff $X \in P$, that is $X$ has a polynomial time algorithm.

Justifications:

1. Robustness of definition to variations in machines.
2. A sound theoretical definition.
3. Most known polynomial time algorithms for “natural” problems have small polynomial running times.
Problems with no known polynomial time algorithms

There are of course undecidable problems (no algorithm at all!) but many problems that we want to solve are of similar flavor to the above.

**Question:** What is common to above problems?
Above problems share the following feature:

**Checkability**

For any YES instance $I_x$ of $X$ there is a proof/certificate/solution that is of length $\text{poly}(|I_x|)$ such that given a proof one can efficiently check that $I_x$ is indeed a YES instance.

Examples:

1. **SAT** formula $\varphi$: proof is a satisfying assignment.
2. **Independent Set** in graph $G$ and $k$: a subset $S$ of vertices.
Above problems share the following feature:

**Checkability**

For any YES instance $I_X$ of $X$ there is a proof/certificate/solution that is of length $\text{poly}(|I_X|)$ such that given a proof one can efficiently check that $I_X$ is indeed a YES instance.

Examples:

1. **SAT** formula $\varphi$: proof is a satisfying assignment.
2. **Independent Set** in graph $G$ and $k$: a subset $S$ of vertices.
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 $C(s, t) = \text{"yes"}$, and conversely, if for some $s$ and $t$, $C(s, t) = \text{"yes"}$ then $s \in X$. The string $t$ is called a **certificate** or **proof** for $s$. 
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

\[
\begin{align*}
\star & \quad s \in X \text{ if and only if} \\
\star & \quad \text{there is a string } t:
\end{align*}
\]

\[
\begin{align*}
1 & \quad |t| \leq p(|s|), \\
2 & \quad C(s, t) = "yes", \\
3 & \quad \text{and } C \text{ runs in polynomial time}.
\end{align*}
\]
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.
Example: Vertex Cover

1. **Problem:** Does $G$ have a vertex cover of size $\leq k$?
2. **Certificate:** $S \subseteq V$.
3. **Certifier:** Check $|S| \leq k$ and that for every edge at least one endpoint is in $S$. 
Example: **SAT**

1. **Problem:** Does formula $\varphi$ have a satisfying truth assignment?

2. **Certificate:** Assignment $a$ of 0/1 values to each variable.

2. **Certifier:** Check each clause under $a$ and say “yes” if all clauses are true.
Example: Composites

**Composite**

**Instance:** A number $s$.

**Question:** Is the number $s$ a composite?

1. **Problem:** Composite.
   1. **Certificate:** A factor $t \leq s$ such that $t \neq 1$ and $t \neq s$.
   2. **Certifier:** Check that $t$ divides $s$. 

Sariel (UIUC)  
CS573  
Fall 2013
Definition

Nondeterministic Polynomial Time (denoted by $\text{NP}$) is the class of all problems that have efficient certifiers.
Nondeterministic Polynomial Time

**Definition**

Nondeterministic Polynomial Time (denoted by $\text{NP}$) is the class of all problems that have efficient certifiers.

**Example**

Independent Set, Vertex Cover, Set Cover, SAT, 3SAT, and Composite are all examples of problems in $\text{NP}$.
A certifier is an algorithm $C(I, c)$ with two inputs:

1. $I$: instance.
2. $c$: proof/certificate that the instance is indeed a YES instance of the given problem.

One can think about $C$ as an algorithm for the original problem, if:

1. Given $I$, the algorithm guess (non-deterministically, and who knows how) the certificate $c$.
2. The algorithm now verifies the certificate $c$ for the instance $I$.

Usually **NP** is described using Turing machines (gag).
Asymmetry in Definition of NP

Note that only YES instances have a short proof/certificate. NO instances need not have a short certificate.

Example

SAT formula $\varphi$. No easy way to prove that $\varphi$ is NOT satisfiable!

More on this and co-NP later on.
Proposition

\( \mathbf{P} \subseteq \mathbf{NP} \).

For a problem in \( \mathbf{P} \) no need for a certificate!

Proof.

Consider problem \( X \in \mathbf{P} \) with algorithm \( A \). Need to demonstrate that \( X \) has an efficient certifier:

1. Certifier \( C \) on input \( s, t \), runs \( A(s) \) and returns the answer.
2. \( C \) runs in polynomial time.
3. If \( s \in X \), then for every \( t \), \( C(s, t) = "yes" \).
4. If \( s \not\in X \), then for every \( t \), \( C(s, t) = "no" \).
Exponential Time

**Definition**

*Exponential Time* (denoted **EXP**) is the collection of all problems that have an algorithm which on input \( s \) runs in exponential time, i.e., \( O(2^{\text{poly}(|s|)}) \).

Example: \( O(2^n) \), \( O(2^{n \log n}) \), \( O(2^{n^3}) \), ...
Exponential Time

Definition

*Exponential Time* (denoted \( \text{EXP} \)) is the collection of all problems that have an algorithm which on input \( s \) runs in exponential time, i.e., \( O(2^{\text{poly}(|s|)}) \).

Example: \( O(2^n) \), \( O(2^{n \log n}) \), \( O(2^{n^3}) \), ...
Proposition

\( \text{NP} \subseteq \text{EXP} \).

Proof.

Let \( X \in \text{NP} \) with certifier \( C \). Need to design an exponential time algorithm for \( X \).

1. For every \( t \), with \( |t| \leq p(|s|) \) run \( C(s, t) \); answer “yes” if any one of these calls returns “yes”.

2. The above algorithm correctly solves \( X \) (exercise).

3. Algorithm runs in \( O(q(|s| + |p(s)|)2^{p(|s|)}) \), where \( q \) is the running time of \( C \).
Examples

1. **SAT**: try all possible truth assignment to variables.
2. **Independent Set**: try all possible subsets of vertices.
3. **Vertex Cover**: try all possible subsets of vertices.
Is $\text{NP}$ efficiently solvable?

We know $\text{P} \subseteq \text{NP} \subseteq \text{EXP}$. 
Is $\text{NP}$ efficiently solvable?

We know $\text{P} \subseteq \text{NP} \subseteq \text{EXP}$.

**Big Question**

Is there a problem in $\text{NP}$ that does not belong to $\text{P}$? Is $\text{P} = \text{NP}$?
If $P = NP$...

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
3. No security on the web.
4. No e-commerce . . .
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).
If $P = NP \ldots$

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The RSA cryptosystem can be broken.
3. No security on the web.
4. No e-commerce \ldots
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).
If $P = NP$...

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The **RSA** cryptosystem can be broken.
3. No security on the web.
4. No e-commerce . . .
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).
If \( P = NP \ldots \)

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The **RSA** cryptosystem can be broken.
3. No security on the web.
4. No e-commerce \ldots
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).
If $P = NP$ ...

Or: If pigs could fly then life would be sweet.

1. Many important optimization problems can be solved efficiently.
2. The **RSA** cryptosystem can be broken.
3. No security on the web.
4. No e-commerce . . .
5. Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).


**P versus NP**

**Status**

Relationship between $P$ and $NP$ remains one of the most important open problems in mathematics/computer science.

**Consensus:** Most people feel/believe $P \neq NP$.

Resolving $P$ versus $NP$ is a Clay Millennium Prize Problem. You can win a million dollars in addition to a Turing award and major fame!
Part III

Not for lecture: Converting any boolean formula into CNF
Consider an arbitrary boolean formula $\phi$ defined over $k$ variables. To keep the discussion concrete, consider the formula $\phi \equiv x_k = x_i \land x_j$. We would like to convert this formula into an equivalent CNF formula.
Formula conversion into CNF

Step 1

Build a truth table for the boolean formula.

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$x_i$</th>
<th>$x_j$</th>
<th>value of $x_k = x_i \land x_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Given an assignment, say, \( x_k = 0 \), \( x_i = 0 \) and \( x_j = 1 \), consider the CNF clause \( x_k \lor x_i \lor \overline{x_j} \) (you negate a variable if it is assigned one). Its truth table is

<table>
<thead>
<tr>
<th>( x_k )</th>
<th>( x_i )</th>
<th>( x_j )</th>
<th>( x_k \lor x_i \lor \overline{x_j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Observe that a single clause assigns zero to one row, and one everywhere else. An conjunction of several such clauses, as such, would result in a formula that is 0 in all the rows that corresponds to these clauses, and one everywhere else.
Write down the **CNF** clause for every row in the table that is zero.

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$x_i$</th>
<th>$x_j$</th>
<th>$x_k = x_i \land x_j$</th>
<th>CNF clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$x_k \lor \overline{x_i} \lor \overline{x_j}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\overline{x_k} \lor x_i \lor x_j$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\overline{x_k} \lor \overline{x_i} \lor x_j$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\overline{x_k} \lor \overline{x_i} \lor \overline{x_j}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The conjunction (i.e., and) of all these clauses is clearly equivalent to the original formula. In this case $\psi \equiv (x_k \lor \overline{x_i} \lor \overline{x_j}) \land (\overline{x_k} \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i \lor \overline{x_j}) \land (\overline{x_k} \lor \overline{x_i} \lor x_j)$.
Formula conversion into CNF

Step 3 - simplify if you want to

Using that \((x \lor y) \land (x \lor \overline{y}) = x\), we have that:

1. \((\overline{x_k} \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i \lor \overline{x_j})\) is equivalent to \((\overline{x_k} \lor x_i)\).
2. \((\overline{x_k} \lor x_i \lor x_j) \land (\overline{x_k} \lor \overline{x_i} \lor x_j)\) is equivalent to \((\overline{x_k} \lor x_j)\).

Using the above two observation, we have that our formula \(\psi \equiv (x_k \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i \lor x_j) \land (\overline{x_k} \lor \overline{x_i} \lor x_j)\) is equivalent to

\[\psi \equiv (x_k \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i) \land (\overline{x_k} \lor x_j).\]

We conclude:

Lemma

The formula \(x_k = x_i \land x_j\) is equivalent to the CNF formula \(\psi \equiv (x_k \lor x_i \lor x_j) \land (\overline{x_k} \lor x_i) \land (\overline{x_k} \lor x_j)\).