SAT and NP

Lecture 21
April 22, 2021
Part I

The Satisfiability Problem (SAT)
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.
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.
Problem: **SAT**

**Instance:** A CNF formula $\varphi$.

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

Problem: **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. $(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5$ is satisfiable; take $x_1, x_2, \ldots, x_5$ to be all true

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

**3SAT**

Given a **3CNF** formula $\varphi$, is there a truth assignment to variables such that $\varphi$ evaluates to true?
Importance of **SAT** and **3SAT**

1. **SAT** and **3SAT** are basic constraint satisfaction problems.
2. Many different problems can be 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**.
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.
SAT \leq_P 3SAT

How SAT is different from 3SAT?
In 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 3SAT every clause must have \textbf{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.

Formal proof later.
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.
Algorithm for 2SAT

A challenging exercise: Given a 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 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...)
Part II

NP
P and NP and Turing Machines

1. **P**: set of decision problems that have polynomial time algorithms.

2. **NP**: set of decision problems that have polynomial time non-deterministic algorithms.

   - Many natural problems we would like to solve are in NP.
   - Every problem in NP has an exponential time algorithm.
   - **P ⊆ NP**
   - Some problems in NP are in P (example, shortest path problem)

**Big Question:** Does every problem in NP have an efficient algorithm? Same as asking whether **P = NP**.
Problems with no known polynomial time algorithms

<table>
<thead>
<tr>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Independent Set</td>
</tr>
<tr>
<td>2 Vertex Cover</td>
</tr>
<tr>
<td>3 Set Cover</td>
</tr>
<tr>
<td>4 SAT</td>
</tr>
<tr>
<td>5 3SAT</td>
</tr>
</tbody>
</table>

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?
Efficient Checkability

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.
Efficient Checkability

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.
3. **Homework**
Given $n \times n$ sudoku puzzle, does it have a solution?
Certifiers

**Definition**
An algorithm $C(\cdot, \cdot)$ is a certifier for problem $X$ if the following two conditions hold:

- For every $s \in X$ there is some string $t$ such that $C(s, t) = \text{"yes"}$
- If $s \not\in X$, $C(s, t) = \text{"no"}$ for every $t$.

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 the following conditions hold:

- For every $s \in X$ there is some string $t$ such that $C(s, t) = "yes"$ and $|t| \leq p(|s|)$.
- If $s \not\in X$, $C(s, t) = "no"$ for every $t$.
- $C(\cdot, \cdot)$ runs in polynomial time.
Example: Independent Set

1. Problem: Does \( G = (V, E) \) have an independent set of size \( \geq k \)?
   1. Certificate: Set \( S \subseteq V \).
   2. 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 $\leq k$?

   1. **Certificate:** $S \subseteq V$.
   2. **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.

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

Problem: 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 \( s \).
Example: Primes

Problem: Prime

Instance: A number $s$.

Question: Is the number $s$ a prime?

1. Problem: Prime.
   1. Certificate: ?
   2. Certifier: ?
Example: Primes

Problem: Prime

Instance: A number \( s \).

Question: Is the number \( s \) a prime?

1. Problem: Prime.
   1. Certificate: ?
   2. Certifier: ?

Not obvious!
Example: Primes

**Problem:** Prime

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

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

1. **Problem:** Prime.
   1. Certificate: ?
   2. Certifier: ?

Not obvious! First shown by Vaughan Pratt in 1975.
Example: Primes

Problem: Prime

Instance: A number $s$.

Question: Is the number $s$ a prime?

1 Problem: Prime.

1 Certificate: ?

2 Certifier: ?

Not obvious! First shown by Vaughan Pratt in 1975. Primes is in P which gives a different proof and an algorithm! Agarwal-Kayal-Saxena 2002.
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 if time permits (which it won’t).
Problem: **PCP**

**Instance:** Two sets of binary strings $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$

**Question:** Are there indices $i_1, i_2, \ldots, i_k$ such that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$

**Certificate:** A sequence of indices $i_1, i_2, \ldots, i_k$

**Certifier:** Check that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$
Post Correspondence Problem

Given: Dominoes, each with a top-word and a bottom-word.

Can one arrange them, using any number of copies of each type, so that the top and bottom strings are equal?

PCP = Posts Correspondence Problem and it is undecidable!

Implies no finite bound on length of certificate!
Post Correspondence Problem

Given: Dominoes, each with a top-word and a bottom-word.

Can one arrange them, using any number of copies of each type, so that the top and bottom strings are equal?

PCP = Posts Correspondence Problem and it is undecidable! Implies no finite bound on length of certificate!
Nondeterministic Polynomial Time

Definition

Nondeterministic Polynomial Time (denoted by **NP** ) is the class of all problems that have efficient certifiers.
Definition

Nondeterministic Polynomial Time (denoted by 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 NP.
Why is it called...

Nondeterministic Polynomial Time

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

NP can be equivalently described using Turing machines.
Proposition

\( P \subseteq \text{NP}. \)
**P versus NP**

**Proposition**

\[ P \subseteq NP. \]

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

**Proof.**

Consider problem \( X \in P \) 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 $\text{EXP}$) is the collection of all problems that have an algorithm which on input $s$ runs in exponential time, i.e., $O(2^{\text{poly}(|s|)})$. 
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: \( O(2^n) \), \( O(2^{n \log n}) \), \( O(2^{n^3}) \), ...
NP versus EXP

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.
Do NP problems have efficient algorithms?

We know $P \subseteq NP \subseteq EXP$. 

Do NP problems have efficient algorithms?

We know $P \subseteq NP \subseteq EXP$.

**Big Question**

Is there a problem in $NP$ that does not belong to $P$? Is $P = NP$?
If $P = NP$

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

1. Many important optimization problems can be solved efficiently.
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.
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.
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 . . .
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 \( P = NP \) this implies that...

- **Vertex Cover** can be solved in polynomial time.
- \( P = \text{EXP} \).
- \( \text{EXP} \subseteq P \).
- All of the above.
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

NP-Completeness
“Hardest” Problems

Question
What is the hardest problem in \textbf{NP}? How do we define it?

Towards a definition
1. Hardest problem must be in \textbf{NP}.
2. Hardest problem must be at least as “difficult” as every other problem in \textbf{NP}.
NP-Complete Problems

Definition

A problem $X$ is said to be NP-Complete if

1. $X \in \text{NP}$, and

2. (Hardness) For any $Y \in \text{NP}$, $Y \leq_{	ext{P}} X$. 
Solving NP-Complete Problems

Proposition

Suppose $X$ is NP-Complete. Then $X$ can be solved in polynomial time if and only if $P = NP$.

Proof.

$\implies$ Suppose $X$ can be solved in polynomial time

1. Let $Y \in NP$. We know $Y \leq_P X$.
2. We showed that if $Y \leq_P X$ and $X$ can be solved in polynomial time, then $Y$ can be solved in polynomial time.
3. Thus, every problem $Y \in NP$ is such that $Y \in P$; $NP \subseteq P$.
4. Since $P \subseteq NP$, we have $P = NP$.

$\Leftarrow$ Since $P = NP$, and $X \in NP$, we have a polynomial time algorithm for $X$. 

Chandra (UIUC)
NP-Hard Problems

**Definition**
A problem $X$ is said to be **NP-Hard** if

1. (Hardness) For any $Y \in \text{NP}$, we have that $Y \leq_P X$.

An **NP-Hard** problem need not be in **NP**!

**Example:** Halting problem is **NP-Hard** (why?) but not **NP-Complete**.
If $X$ is NP-Complete

1. Since we believe $P \neq NP$,
2. and solving $X$ implies $P = NP$.

$X$ is unlikely to be efficiently solvable.
Consequences of proving NP-Completeness

If $X$ is NP-Complete

1. Since we believe $P \neq NP$,
2. and solving $X$ implies $P = NP$.

$X$ is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for $X$. 
Consequences of proving NP-Completeness

If $X$ is NP-Complete

1. Since we believe $P \neq NP$,
2. and solving $X$ implies $P = NP$.

$X$ is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for $X$. 
Consequences of proving NP-Completeness

If $X$ is NP-Complete

1. Since we believe $P \neq NP,$
2. and solving $X$ implies $P = NP.$

$X$ is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for $X.$
(This is proof by mob opinion — take with a grain of salt.)
**Question**
Are there any problems that are **NP-Complete**?

**Answer**
Yes! Many, many problems are **NP-Complete**.
Theorem (Cook-Levin)

SAT is NP-Complete.
Cook-Levin Theorem

**Theorem (Cook-Levin)**

**SAT** is NP-Complete.

Need to show

1. **SAT** is in **NP**.
2. every **NP** problem **X** reduces to **SAT**.

Steve Cook won the Turing award for his theorem.
To prove $X$ is **NP-Complete**, show

1. Show that $X$ is in **NP**.
2. Give a polynomial-time reduction *from* a known **NP-Complete** problem such as **SAT** *to* $X$
To prove $X$ is NP-Complete, show

1. Show that $X$ is in NP.
2. Give a polynomial-time reduction from a known NP-Complete problem such as SAT to $X$

$\text{SAT} \leq_P X$ implies that every NP problem $Y \leq_P X$. Why?
To prove $X$ is **NP-Complete**, show

1. Show that $X$ is in **NP**.
2. Give a polynomial-time reduction from a known **NP-Complete** problem such as **SAT** to $X$

**SAT** $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why?

Transitivity of reductions:

$Y \leq_P SAT$ and $SAT \leq_P X$ and hence $Y \leq_P X$. 
3-SAT is NP-Complete

- 3-SAT is in \( NP \)
- \( SAT \leq_p 3-SAT \) as we saw
SAT is NP-Complete due to Cook-Levin theorem

SAT $\leq_p$ 3-SAT

3-SAT $\leq_p$ Independent Set

Independent Set $\leq_p$ Vertex Cover

Independent Set $\leq_p$ Clique

3-SAT $\leq_p$ 3-Color

3-SAT $\leq_p$ Hamiltonian Cycle
SAT is NP-Complete due to Cook-Levin theorem

SAT \leq_p 3-SAT

3-SAT \leq_p Independent Set

Independent Set \leq_p Vertex Cover

Independent Set \leq_p Clique

3-SAT \leq_p 3-Color

3-SAT \leq_p Hamiltonian Cycle

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be NP-Complete.

A surprisingly frequent phenomenon!
Part IV

Reducing 3-SAT to Independent Set
Problem: 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.
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 $\varphi$. 
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.
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.
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.
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

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

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.

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.

![Graph Diagram]

**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

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

Proposition

\( \varphi \) is satisfiable iff \( G_{\varphi} \) has an independent set of size \( k \) (\( = \) number of clauses in \( \varphi \)).

Proof.

\( \Rightarrow \) Let \( a \) be the truth assignment satisfying \( \varphi \).
Correctness

**Proposition**

\( \varphi \) is satisfiable iff \( G_\varphi \) 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. Why?
Correctness (contd)

Proposition

\( \varphi \) is satisfiable iff \( G_\varphi \) has an independent set of size \( k \) (\( k \) = number of clauses in \( \varphi \)).

Proof.

\( \iff \) Let \( S \) be an independent set of size \( k \)

1. \( S \) must contain exactly one vertex from each clause triangle
2. \( S \) cannot contain vertices labeled by conflicting literals
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
Part V

SAT reduces to 3-SAT
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 \textbf{exactly} 3 different literals.
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

SAT $\leq_P$ 3SAT.

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

1. $\phi$ is satisfiable iff $\phi'$ is satisfiable.

2. $\phi'$ can be constructed from $\phi$ in time polynomial in $|\phi|$.

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

\textbf{SAT} \leq_p \textbf{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 \( |\varphi| \).
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.
SAT $\leq_p$ 3SAT

A clause with two literals

Reduction Ideas: clause with 2 literals

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

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

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
SAT \leq_P 3SAT

A clause with a single literal

**Reduction Ideas: clause with 1 literal**

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

\[
\begin{align*}
c' &= (\ell \lor u \lor v) \land (\ell \lor u \lor \neg v) \\
& \quad \land (\ell \lor \neg u \lor v) \land (\ell \lor \neg u \lor \neg v).
\end{align*}
\]

2. Suppose \( \varphi = \psi \land c \). Then \( \varphi' = \psi \land c' \) is satisfiable iff \( \varphi \) is satisfiable.
**SAT $\leq_P$ 3SAT**

A clause with more than 3 literals

---

**Reduction Ideas: clause with more than 3 literals**

1. **Case clause with five literals:** Let $c = \ell_1 \lor \ell_2 \lor \ell_3 \lor \ell_4 \lor \ell_5$. Let $u$ be a new variable. Consider

   $$c' = (\ell_1 \lor \ell_2 \lor \ell_3 \lor u) \land (\ell_4 \lor \ell_5 \lor \neg u).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
SAT $\leq_P$ 3SAT

A clause with more than 3 literals

**Reduction Ideas: clause with more than 3 literals**

1. **Case clause with $k > 3$ literals:** Let $c = \ell_1 \lor \ell_2 \lor \ldots \lor \ell_k$. Let $u$ be a new variable. Consider

   $$c' = \left(\ell_1 \lor \ell_2 \ldots \ell_{k-2} \lor u\right) \land \left(\ell_{k-1} \lor \ell_k \lor \neg u\right).$$

2. Suppose $\varphi = \psi \land c$. Then $\varphi' = \psi \land c'$ is satisfiable iff $\varphi$ is satisfiable.
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

$$c' = \left( \ell_1 \lor \ell_2 \lor u_1 \right) \land \left( \ell_3 \lor \neg u_1 \lor u_2 \right) \land \left( \ell_4 \lor \neg u_2 \lor u_3 \right) \land \cdots \land \left( \ell_{k-2} \lor \neg u_{k-4} \lor u_{k-3} \right) \land \left( \ell_{k-1} \lor \ell_k \lor \neg u_{k-3} \right).$$

**Claim**

$\varphi = \psi \land c$ is satisfiable iff $\varphi' = \psi \land c'$ is satisfiable.

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

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

Example

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

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \]
Example

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

Equivalent form:

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

Example

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

Equivalent form:

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

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

Equivalent form:

\[ \psi = \left( \neg x_1 \lor \neg x_4 \lor z \right) \land \left( \neg x_1 \lor \neg x_4 \lor \neg z \right) \land \left( x_1 \lor \neg x_2 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_3 \lor y_1 \right) \land \left( x_4 \lor x_1 \lor \neg y_1 \right) \land \left( x_1 \lor u \lor v \right) \land \left( x_1 \lor \neg u \lor \neg v \right) \land \left( x_1 \lor \neg u \lor \neg v \right). \]
Overall Reduction Algorithm
Reduction from SAT to 3SAT

\textbf{ReduceSATTo3SAT}(\varphi):

\begin{verbatim}
  // \varphi: CNF formula.
  for each clause \(c\) of \(\varphi\) 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)
\end{verbatim}

Correctness (informal)

\(\varphi\) is satisfiable iff \(\psi\) is satisfiable because for each clause \(c\), the new 3CNF formula \(c'\) is logically equivalent to \(c\).