Poly-Time Reductions II

Lecture 23
Thursday, April 18, 2019

LaTeXed: December 27, 2018 08:26

Part I

Review: Polynomial reductions

Polynomial-time Reduction

Definition

$X \leq_P Y$: ***polynomial time reduction*** from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ such that:

1. Given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y.
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$. $(|I_Y| =$ size of $I_Y)$.
3. Answer to I_X YES $\iff$ answer to I_Y is YES.

Polynomial-time Reduction

Definition

$X \leq_P Y$: ***polynomial time reduction*** from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ such that:

1. Given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y.
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$. $(|I_Y| = \text{size of } I_Y)$.
3. Answer to I_X YES $\iff$ answer to I_Y is YES.

Proposition

If $X \leq_P Y$ then a polynomial time algorithm for Y implies a polynomial time algorithm for X.

Polynomial-time Reduction

Definition

$X \leq_P Y$: ***polynomial time reduction*** from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ such that:

1. Given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y.
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$. ($|I_Y| =$ size of I_Y).
3. Answer to I_X YES $\iff$ answer to I_Y is YES.

Proposition

If $X \leq_P Y$ then a polynomial time algorithm for Y implies a polynomial time algorithm for X.

This is a ***Karp reduction***.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\left(a * n^b\right) \leq c * \left(a * n^b\right)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\left(n^{bd}\right)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\left(a * n^b\right) \leq c * \left(a * n^b\right)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\left(n^{bd}\right)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
    1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
    2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

    a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\big(a * n^b\big) \leq c * \big(a * n^b\big)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\Big(n^{bd}\Big)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\big(a * n^b\big) \leq c * \big(a * n^b\big)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\Big(n^{bd}\Big)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \le a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \le c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \le g\left(a * n^b\right) \le c * \left(a * n^b\right)^d \le c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\left(n^{bd}\right)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\big(a * n^b\big) \leq c * \big(a * n^b\big)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\Big(n^{bd}\Big)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Composing polynomials...

A quick reminder

1. f and g monotone increasing. Assume that:
   1. $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   2. $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)

   a, b, c, d: constants.
2. $g\Big(f(n)\Big) \leq g\big(a * n^b\big) \leq c * \big(a * n^b\big)^d \leq c \cdot a^d * n^{bd}$
3. $\implies g(f(n)) = O\Big(n^{bd}\Big)$ is a polynomial.
4. **Conclusion:** Composition of two polynomials, is a polynomial.

Transitivity of Reductions

Proposition

$X \leq_P Y$ and $Y \leq_P Z$ implies that $X \leq_P Z$.

1. 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.
2. To prove $X \leq_P Y$ you need to show a reduction FROM X TO Y
3. ...show that an algorithm for Y implies an algorithm for X.

Transitivity of Reductions

Proposition

$X \le_P Y$ and $Y \le_P Z$ implies that $X \le_P Z$.

1. Note: $X \le_P Y$ does not imply that $Y \le_P X$ and hence it is very important to know the FROM and TO in a reduction.
2. To prove $X \le_P Y$ you need to show a reduction FROM X TO Y
3. ...show that an algorithm for Y implies an algorithm for X.

Transitivity of Reductions

Proposition

$X \leq_P Y$ and $Y \leq_P Z$ implies that $X \leq_P Z$.

1. 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.
2. To prove $X \leq_P Y$ you need to show a reduction FROM X TO Y
3. ...show that an algorithm for Y implies an algorithm for X.

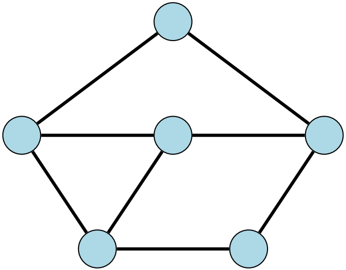
Part II

Independent Set and Vertex Cover

Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

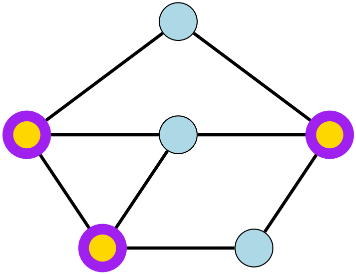
1. ***vertex cover*** if every $e \in E$ has at least one endpoint in S.



Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

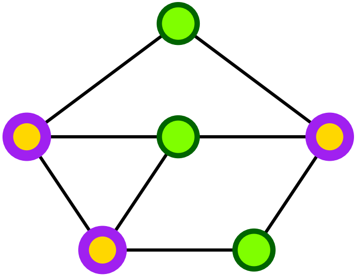
1. ***vertex cover*** if every $e \in E$ has at least one endpoint in S.



Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:

1. ***vertex cover*** if every $e \in E$ has at least one endpoint in S.



The **Vertex Cover** Problem

Problem (**Vertex Cover**)

Input: *A graph G and integer $\boldsymbol{k}$.*
Goal: *Is there a vertex cover of size $\leq \boldsymbol{k}$ in G?*

Can we relate **Independent Set** and **Vertex Cover**?

The **Vertex Cover** Problem

Problem (**Vertex Cover**)

Input: *A graph G and integer $\boldsymbol{k}$.*
Goal: *Is there a vertex cover of size $\leq \boldsymbol{k}$ in G?*

Can we relate **Independent Set** and **Vertex Cover**?

Relationship between...

Vertex Cover and Independent Set

Proposition

Let $G = (V, E)$ be a graph.
$S \subseteq V$ is independent set $\iff$ $V \setminus S$ is vertex cover.

Proof.

$(\Rightarrow)$ Let S be an independent set

1. Consider any edge $uv \in E$.
2. Since S is an independent set, either $u \notin S$ or $v \notin S$.
3. Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
4. $V \setminus S$ is a vertex cover.

$(\Leftarrow)$ Let $V \setminus S$ be some vertex cover:

1. Consider $u, v \in S$
2. uv is not an edge of G, as otherwise $V \setminus S$ does not cover uv.
3. $\implies$ S is thus an independent set. □

Independent Set $\leq_P$ Vertex Cover

1. G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
2. G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
3. (G, k): instance of **Independent Set**
   $(G, n - k)$: instance of **Vertex Cover** with the same answer.
4. $\implies$ **Independent Set** $\leq_P$ **Vertex Cover**.
5. Same argument in reverse...
6. $\implies$ **Vertex Cover** $\leq_P$ **Independent Set**.

Independent Set $\leq_P$ Vertex Cover

1. G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
2. G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
3. (G, k): instance of **Independent Set**
   $(G, n - k)$: instance of **Vertex Cover** with the same answer.
4. $\implies$ **Independent Set** $\leq_P$ **Vertex Cover**.
5. Same argument in reverse...
6. $\implies$ **Vertex Cover** $\leq_P$ **Independent Set**.

Independent Set $\leq_P$ Vertex Cover

1. G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
2. G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
3. (G, k): instance of **Independent Set**
   $(G, n - k)$: instance of **Vertex Cover** with the same answer.
4. $\implies$ **Independent Set** $\leq_P$ **Vertex Cover**.
5. Same argument in reverse...
6. $\implies$ **Vertex Cover** $\leq_P$ **Independent Set**.

Independent Set $\leq_P$ Vertex Cover

1. G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
2. G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
3. (G, k): instance of **Independent Set**
   $(G, n - k)$: instance of **Vertex Cover** with the same answer.
4. $\implies$ **Independent Set** $\leq_P$ **Vertex Cover**.
5. Same argument in reverse...
6. $\implies$ **Vertex Cover** $\leq_P$ **Independent Set**.

Independent Set $\leq_P$ Vertex Cover

1. G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
2. G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
3. (G, k): instance of **Independent Set**
   $(G, n - k)$: instance of **Vertex Cover** with the same answer.
4. $\implies$ **Independent Set** $\leq_P$ **Vertex Cover**.
5. Same argument in reverse...
6. $\implies$ **Vertex Cover** $\leq_P$ **Independent Set**.

Polynomial time reduction...

Proving Correctness of Reductions

To prove that $X \leq_P Y$ you need to give an algorithm $\mathcal{A}$ that:

1. Transforms an instance I_X of X into an instance I_Y of Y.
2. Satisfies the property that answer to I_X is YES iff I_Y is YES.
    1. typical easy direction to prove: answer to I_Y is YES if answer to I_X is YES
    2. typical difficult direction to prove: answer to I_X is YES if answer to I_Y is YES (equivalently answer to I_X is NO if answer to I_Y is NO).
3. Runs in ***polynomial*** time.

Polynomial time reduction...

Proving Correctness of Reductions

To prove that $X \leq_P Y$ you need to give an algorithm $\mathcal{A}$ that:

1. Transforms an instance I_X of X into an instance I_Y of Y.
2. Satisfies the property that answer to I_X is YES iff I_Y is YES.
   1. typical easy direction to prove: answer to I_Y is YES if answer to I_X is YES
   2. typical difficult direction to prove: answer to I_X is YES if answer to I_Y is YES (equivalently answer to I_X is NO if answer to I_Y is NO).
3. Runs in **polynomial** time.

Part III

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, $x_1 \vee x_2 \vee \neg x_4$ is a clause.
3. A ***formula in conjunctive normal form*** (CNF) is propositional formula which is a conjunction of clauses
   1. $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a CNF formula.
4. A formula φ is a $\mathrm{3CNF}$:
   A CNF formula such that every clause has **exactly** 3 literals.
   1. $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ is a $\mathrm{3CNF}$ formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is not.

Propositional Formulas

Definition

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

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, $x_1 \vee x_2 \vee \neg x_4$ is a clause.
3. A ***formula in conjunctive normal form*** (CNF) is propositional formula which is a conjunction of clauses
   1. $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a CNF formula.
4. A formula φ is a $3\mathrm{CNF}$:
   A CNF formula such that every clause has **exactly** 3 literals.
   1. $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ is a $3\mathrm{CNF}$ formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is not.

Satisfiability

Problem: SAT

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

Problem: 3SAT

Instance: A $3\mathrm{CNF}$ formula φ.
Question: Is there a truth assignment to the variable of φ such that φ evaluates to true?

Satisfiability

SAT

Given a CNF formula φ, is there a truth assignment to variables such that φ evaluates to true?

Example

1. $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is satisfiable; take $x_1, x_2, \ldots x_5$ to be all true
2. $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ is not satisfiable.

3SAT

Given a 3CNF formula φ, is there a truth assignment to variables such that φ 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-Complete**ness.

$z = \overline{x}$

Given two bits x, z which of the following **SAT** formulas is equivalent to the formula $z = \overline{x}$:

- $(\overline{z} \vee x) \wedge (z \vee \overline{x})$.
- $(z \vee x) \wedge (\overline{z} \vee \overline{x})$.
- $(\overline{z} \vee x) \wedge (\overline{z} \vee \overline{x}) \wedge (\overline{z} \vee \overline{x})$.
- $z \oplus x$.
- $(z \vee x) \wedge (\overline{z} \vee \overline{x}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x)$.

$z = x \wedge y$

Given three bits x, y, z which of the following **SAT** formulas is equivalent to the formula $z = x \wedge y$:

- $(\overline{z} \vee x \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(z \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(z \vee x \vee y) \wedge (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (\overline{z} \vee \overline{x} \vee \overline{y})$.

Converting $z = x \wedge y$ to 3SAT

z	x	y					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Converting $\mathbf{z = x \wedge y}$ to 3SAT

z	x	y	$z = x \wedge y$				
0	0	0	1				
0	0	1	1				
0	1	0	1				
0	1	1	0				
1	0	0	0				
1	0	1	0				
1	1	0	0				
1	1	1	1				

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$				
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	**0**	1	1	1
1	0	0	0	1	**0**	1	1
1	0	1	0	1	1	**0**	1
1	1	0	0	1	1	1	**0**
1	1	1	1	1	1	1	1

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$			
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	**0**	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$	$\overline{z} \vee x \vee y$		
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	**0**	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	**0**	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x} \vee \overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

$$\Big(z = x \wedge y\Big)$$

$$\equiv$$

$$(z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y		
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Converting $\mathbf{z = x \wedge y}$ to 3SAT

z	x	y	$z = x \wedge y$	
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Converting $\mathbf{z = x \land y}$ to 3SAT

z	x	y	$z = x \land y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$z \lor \overline{x} \lor \overline{y}$
1	0	0	0	$\overline{z} \lor x \lor y$
1	0	1	0	$\overline{z} \lor x \lor y$
1	1	0	0	$\overline{z} \lor x \lor y$
1	1	1	1	

Converting $z = x \wedge y$ to 3SAT

z	x	y	$z = x \wedge y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	0	$\overline{z} \vee x \vee y$
1	1	0	0	$\overline{z} \vee x \vee y$
1	1	1	1	

$$\left(z = x \wedge y\right)$$
$$\equiv$$
$$(z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y)$$

Converting $\mathbf{z = x \wedge y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   2. $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma

$$\left(z = x \wedge y\right) \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z = x \wedge y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   2. $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y)$ is equivalent to $\psi \equiv (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x) \wedge (\overline{z} \vee y)$

Lemma

$$(z = x \wedge y) \equiv (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x) \wedge (\overline{z} \vee y)$$

Converting $\mathbf{z = x \wedge y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   2. $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma

$$\left(z = x \wedge y\right) \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z = x \wedge y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $\left(\overline{z} \vee x \vee u\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) = \left(\overline{z} \vee x\right)$
   2. $\left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right) = \left(\overline{z} \vee y\right)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma

$$\left(z = x \wedge y\right) \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $z = x \wedge y$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $\left(\overline{z} \vee x \vee u\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) = \left(\overline{z} \vee x\right)$
   2. $\left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right) = \left(\overline{z} \vee y\right)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma

$$\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

$z = x \vee y$

Given three bits x, y, z which of the following **SAT** formulas is equivalent to the formula $z = x \vee y$:

- $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(z \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y})$.
- $(z \vee x \vee y) \wedge (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y) \wedge (\overline{z} \vee \overline{x} \vee \overline{y})$.
- $(\overline{z} \vee x \vee y) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y})$.

Converting $z = x \vee y$ to 3SAT

z	x	y		
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Converting $z = x \vee y$ to 3SAT

z	x	y	$z = x \vee y$	
0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $\mathbf{z = x \vee y}$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $z = x \vee y$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	$z \vee x \vee \overline{y}$
0	1	0	0	$z \vee \overline{x} \vee y$
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $z = x \vee y$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	$z \vee x \vee \overline{y}$
0	1	0	0	$z \vee \overline{x} \vee y$
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	1	
1	1	0	1	
1	1	1	1	

$$\left(z = x \vee y\right)$$

$$\equiv$$

$$(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   2. $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma

The formula $z = x \vee y$ is equivalent to the CNF formula

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $z = x \vee y$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   2. $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma

The formula $z = x \vee y$ is equivalent to the CNF formula

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $\mathbf{z = x \vee y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   2. $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma

The formula $z = x \vee y$ is equivalent to the CNF formula

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $\mathbf{z = x \vee y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1. $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   2. $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma

The formula $z = x \vee y$ is equivalent to the CNF *formula*

$$\left(z = x \vee y\right) \quad \equiv \quad (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

In **SAT** clauses might have arbitrary length: $1, 2, 3, \ldots$ variables:

$$\left(x \vee y \vee z \vee w \vee u\right) \wedge \left(\neg x \vee \neg y \vee \neg z \vee w \vee u\right) \wedge \left(\neg x\right)$$

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.

SAT $\leq_P$ 3SAT

How **SAT** is different from **3SAT**?

In **SAT** clauses might have arbitrary length: $1, 2, 3, \ldots$ variables:

$$\left(x \vee y \vee z \vee w \vee u\right) \wedge \left(\neg x \vee \neg y \vee \neg z \vee w \vee u\right) \wedge \left(\neg x\right)$$

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**.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ ***3SAT***.

Given φ a **SAT** formula we create a **3SAT** formula φ' such that

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

Idea: if a clause of φ is not of length $\mathbf{3}$, replace it with several clauses of length exactly $\mathbf{3}$.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ 3SAT.

Given φ a **SAT** formula we create a **3SAT** formula φ' such that

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

Idea: if a clause of φ is not of length 3, replace it with several clauses of length exactly 3.

SAT $\leq_P$ 3SAT

Claim

SAT $\leq_P$ ***3SAT***.

Given φ a **SAT** formula we create a **3SAT** formula φ' such that

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

Idea: if a clause of φ 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 $c = \ell_1 \vee \ell_2$. Let u be a new variable. Consider

$$c' = \left(\ell_1 \vee \ell_2 \vee u\right) \wedge \left(\ell_1 \vee \ell_2 \vee \neg u\right).$$

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

$$c' = \left(\ell \vee u \vee v\right) \wedge \left(\ell \vee u \vee \neg v\right) \wedge \left(\ell \vee \neg u \vee v\right) \wedge \left(\ell \vee \neg u \vee \neg v\right).$$

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

Reduction Ideas: clause with more than 3 literals

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

$$c' = \left(\ell_1 \vee \ell_2 \vee \ell_3 \vee u\right) \wedge \left(\ell_4 \vee \ell_5 \vee \neg u\right).$$

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

SAT $\leq_P$ 3SAT

Reduction Ideas: clause with more than 3 literals

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

$$c' = \Big(\ell_1 \vee \ell_2 \ldots \ell_{k-2} \vee u\Big) \wedge \Big(\ell_{k-1} \vee \ell_k \vee \neg u\Big).$$

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

Breaking a clause

Lemma

For any boolean formulas $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *and* $\boldsymbol{z}$ *a new boolean variable. Then*

$$\boldsymbol{X} \vee \boldsymbol{Y} \text{ is satisfiable}$$

if and only if, $\boldsymbol{z}$ *can be assigned a value such that*

$$\left(\boldsymbol{X} \vee \boldsymbol{z}\right) \wedge \left(\boldsymbol{Y} \vee \neg \boldsymbol{z}\right) \text{ is satisfiable}$$

(with the same assignment to the variables appearing in $\boldsymbol{X}$ *and* $\boldsymbol{Y}$*).*

SAT $\leq_P$ 3SAT (contd)

Clauses with more than 3 literals

Let $c = \ell_1 \vee \cdots \vee \ell_k$. Let $u_1, \ldots u_{k-3}$ be new variables. Consider

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

Claim

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

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

$$c' = \Big(\ell_1 \vee \ell_2 \ldots \vee \ell_{k-2} \vee u_{k-3}\Big) \wedge \Big(\ell_{k-1} \vee \ell_k \vee \neg u_{k-3}\Big).$$

An Example

Example

$$\varphi = \left(\neg x_1 \vee \neg x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right)$$
$$\wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\right) \wedge \left(x_1\right).$$

Equivalent form:

$$\psi = (\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z)$$
$$\wedge (x_1 \vee \neg x_2 \vee \neg x_3)$$
$$\wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1)$$
$$\wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v)$$
$$\wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v).$$

An Example

Example

$$\varphi = \left(\neg x_1 \vee \neg x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\right) \wedge \left(x_1\right).$$

Equivalent form:

$$\psi = (\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1) \wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v) \wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v).$$

An Example

Example

$$\varphi = \left(\neg x_1 \vee \neg x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\right) \wedge \left(x_1\right).$$

Equivalent form:

$$\begin{aligned}\psi = &(\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z)\\ &\wedge (x_1 \vee \neg x_2 \vee \neg x_3)\\ &\wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1)\\ &\wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v)\\ &\wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v).\end{aligned}$$

An Example

Example

$$\varphi = \left(\neg x_1 \vee \neg x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\right) \wedge \left(x_1\right).$$

Equivalent form:

$$\begin{aligned}\psi = &(\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z)\\ &\wedge (x_1 \vee \neg x_2 \vee \neg x_3)\\ &\wedge (\neg x_2 \vee \neg x_3 \vee y_1) \wedge (x_4 \vee x_1 \vee \neg y_1)\\ &\wedge (x_1 \vee u \vee v) \wedge (x_1 \vee u \vee \neg v)\\ &\wedge (x_1 \vee \neg u \vee v) \wedge (x_1 \vee \neg u \vee \neg v).\end{aligned}$$

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
    ψ is conjunction of all c′ constructed in loop
    return Solver3SAT(ψ)
```

Correctness (informal)

φ is satisfiable iff ψ is satisfiable because for each clause c, the new $3\mathrm{CNF}$ 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 $(x \vee y \vee z)$. We need to reduce it to a collection of $2\mathrm{CNF}$ clauses. Introduce a face variable α, and rewrite this as

$$(x \vee y \vee \alpha) \wedge (\neg\alpha \vee z) \qquad \text{(bad! clause with 3 vars)}$$

or

$$(x \vee \alpha) \wedge (\neg\alpha \vee y \vee z) \qquad \text{(bad! clause with 3 vars).}$$

(In animal farm language: **2SAT** good, **3SAT** bad.)

What about 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 ever $\mathbf{2}\mathrm{CNF}$ 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...)

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**
   SAT $\leq_P$ **3SAT**.
   $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover**
   3SAT. $\cong_P$ **SAT**.

What do we know so far

1. **Independent Set $\leq_P$ Clique**
   Clique $\leq_P$ Independent Set.
   $\implies$ **Clique $\approx_P$ Independent Set.**
2. Vertex Cover $\leq_P$ Independent Set
   Independent Set $\leq_P$ Vertex Cover.
   $\implies$ Independent Set $\approx_P$ Vertex Cover.
3. 3SAT $\leq_P$ SAT
   SAT $\leq_P$ 3SAT.
   $\implies$ 3SAT $\approx_P$ SAT.
4. Clique $\approx_P$ Independent Set $\approx_P$ Vertex Cover
   3SAT. $\approx_P$ SAT.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. 3SAT $\leq_P$ SAT
   SAT $\leq_P$ 3SAT.
   $\implies$ 3SAT $\cong_P$ SAT.
4. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover**
   3SAT. $\cong_P$ SAT.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. 3SAT $\leq_P$ SAT
   SAT $\leq_P$ 3SAT.
   $\implies$ 3SAT $\cong_P$ SAT.
4. Clique $\cong_P$ Independent Set $\cong_P$ Vertex Cover
   3SAT. $\cong_P$ SAT.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**
   SAT $\leq_P$ **3SAT**.
   $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover**
   3SAT. $\cong_P$ **SAT**.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**
   SAT $\leq_P$ **3SAT**.
   $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover**
   3SAT. $\cong_P$ **SAT**.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**
   Clique $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**
   Independent Set $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**
   SAT $\leq_P$ **3SAT**.
   $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover**
   3SAT. $\cong_P$ **SAT**.

Part IV

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 $\boldsymbol{NP}$.
- Every problem in $\boldsymbol{NP}$ has an exponential time algorithm
- $\boldsymbol{P} \subseteq \boldsymbol{NP}$
- Some problems in $\boldsymbol{NP}$ are in $\boldsymbol{P}$ (example, shortest path problem)

Big Question: Does every problem in $\boldsymbol{NP}$ have an efficient algorithm? Same as asking whether $\boldsymbol{P} = \boldsymbol{NP}$.

Problems with no known polynomial time algorithms

Problems

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

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 poly($|I_X|$) such that given a proof one can efficiently check that I_X is indeed a YES instance.

Examples:

1. **SAT** formula φ: proof is a satisfying assignment.
2. **Independent Set** in graph G and k: a subset S of vertices.
3. **Homework**

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 poly($|I_X|$) such that given a proof one can efficiently check that I_X is indeed a YES instance.

Examples:

1. **SAT** formula φ: proof is a satisfying assignment.
2. **Independent Set** in graph G and k: a subset S of vertices.
3. **Homework**

Sudoku

			2	5				
	3	6		4		8		
	4					1	6	
2								
7	6						1	9
								3
	1	5					7	
		9		8		2	4	
				3	7			

Given $n \times n$ sudoku puzzle, does it have a solution?

Solution to the Sudoku example...

1	8	7	**2**	**5**	6	9	3	4
9	**3**	**6**	7	**4**	1	**8**	5	2
5	**4**	2	8	9	3	**1**	**6**	7
2	9	1	3	7	4	6	8	5
7	**6**	3	5	2	8	4	**1**	**9**
8	5	4	6	1	9	7	2	**3**
4	**1**	**5**	9	6	2	3	**7**	8
3	7	**9**	1	**8**	5	**2**	**4**	6
6	2	8	4	**3**	**7**	5	9	1

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 $C(s, t) =$ "yes"
- If $s \notin X$, $C(s, t) =$ "no" for every t.

The string t is called a certificate or proof for s.

Efficient (polynomial time) Certifiers

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 $s \notin X$, $C(s, t) =$ "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 φ have a satisfying truth assignment?
    1. Certificate: Assignment $\boldsymbol{a}$ of $\mathbf{0/1}$ values to each variable.
    2. Certifier: Check each clause under $\boldsymbol{a}$ 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: NFA Universality

Problem: NFA Universality

> **Instance:** Description of a NFA M.
> **Question:** Is $L(M) = \Sigma^*$, that is, does M accept all strings?

1. Problem: **NFA Universality**.
   1. Certificate: A DFA M' equivalent to M
   2. Certifier: Check that $L(M') = \Sigma^*$

Certifier is efficient but certificate is not necessarily short! We do not know if the problem is in NP.

Example: NFA Universality

Problem: NFA Universality

> **Instance:** Description of a NFA M.
> **Question:** Is $L(M) = \Sigma^*$, that is, does M accept all strings?

1. Problem: **NFA Universality**.
   1. Certificate: A DFA M' equivalent to M
   2. Certifier: Check that $L(M') = \Sigma^*$

Certifier is efficient but certificate is not necessarily short! We do not know if the problem is in NP.

Example: A String Problem

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 $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$

1. Problem: **PCP**
   1. Certificate: A sequence of indices $i_1, i_2, \ldots, i_k$
   2. Certifier: Check that $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$

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

Example: A String Problem

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 $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$

1. Problem: **PCP**
    1. Certificate: A sequence of indices $i_1, i_2, \ldots, i_k$
    2. Certifier: Check that $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$

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.

Nondeterministic Polynomial Time

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.

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 φ. No easy way to prove that φ is NOT satisfiable!

More on this and **co-NP** later on.

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 $s \notin X$, then for every t, $C(s, t) =$ "no". □

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 $s \notin X$, 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., $O(2^{\mathrm{poly}(|s|)})$.

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

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., $O(2^{\mathrm{poly}(|s|)})$.

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

NP versus EXP

Proposition

NP $\subseteq$ **EXP**.

Proof.

Let $X \in$ **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 **NP** efficiently solvable?

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

Is NP efficiently solvable?

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

Big Question

Is there are problem in **NP** that does not belong to **P**? Is $\mathbf{P} = \mathbf{NP}$?

If $\mathbf{P} = \mathbf{NP}$ . . .

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 $\mathbf{P} = \mathbf{NP}$ . . .

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 $\mathbf{P} = \mathbf{NP}$ . . .

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 $\mathsf{P} = \mathsf{NP}$ . . .

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 $\mathbf{P} = \mathbf{NP}$ . . .

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 $\mathbf{P} = \mathbf{NP}$ this implies that...

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

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 $\boldsymbol{P \neq NP}$.

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!