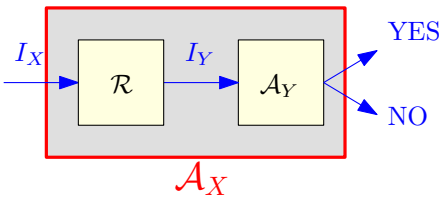
Chapter 2

Reductions and NP

NEW CS 473: Theory II, Fall 2015
August 27, 2015

2.1 Total recall...

2.1.0.1 Polynomial-time reductions



(A) Algorithm is ***efficient*** if it runs in polynomial-time.
(B) Interested only in **polynomial-time reductions**.
(C) $X \leq_P Y$: Have polynomial-time reduction from problem X to problem Y.
(D) $\mathcal{A}_Y$: poly-time algorithm for Y.
(E) $\implies$ Polynomial-time/efficient algorithm for X.

2.2 Polynomial time reductions

2.2.0.1 Polynomial-time reductions and instance sizes

Proposition $\mathcal{R}$: a polynomial-time reduction from X to Y.

Then, for any instance I_X of X, the size of the instance I_Y of Y produced from I_X by $\mathcal{R}$ is polynomial in the size of I_X. ■

Proof: $\mathcal{R}$ is a polynomial-time algorithm and hence on input I_X of size $|I_X|$ it runs in time $p(|I_X|)$ for some polynomial $p()$.

I_Y is the output of $\mathcal{R}$ on input I_X.

$\mathcal{R}$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$. ■

Note: Converse is not true. A reduction need not be polynomial-time even if output of reduction is of size polynomial in its input.

2.2.0.2 Polynomial-time Reduction

Definition 2.2.2. $X \leq_P Y$: ***polynomial time reduction*** from a *decision* problem X to a *decision* problem Y is an *algorithm* $\mathcal{A}$ such that:
(A) Given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y.
(B) $\mathcal{A}$ runs in time polynomial in $|I_X|$. ($|I_Y|$ = size of I_Y).
(C) 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***.

2.2.1 Composing polynomials...

2.2.1.1 A quick reminder

(A) f and g monotone increasing. Assume that:
  (A) $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
  (B) $g(n) \leq c * n^d$ (i.e., $g(n) = O(n^d)$)
  a, b, c, d: constants.
(B) $g\Big(f(n)\Big) \leq g\big(a * n^b\big) \leq c * \big(a * n^b\big)^d \leq c \cdot a^d * n^{bd}$
(C) $\implies g(f(n)) = O\Big(n^{bd}\Big)$ is a polynomial.
(D) **Conclusion:** Composition of two polynomials, is a polynomial.

2.2.1.2 Transitivity of Reductions

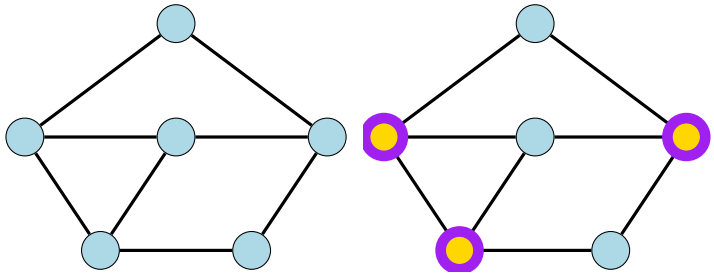
Proposition $X \leq_P Y$ and $Y \leq_P Z$ implies that $X \leq_P Z$. ■

(A) **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.
(B) To prove $X \leq_P Y$ you need to show a reduction FROM X TO Y
(C) ...show that an algorithm for Y implies an algorithm for X.

2.3 Independent Set and Vertex Cover

2.3.0.1 Vertex Cover

Given a graph $G = (V, E)$, a set of vertices S is:
(A) A ***vertex cover*** if every $e \in E$ has at least one endpoint in S.



2.3.0.2 The Vertex Cover Problem

Problem 2.3.1 (**Vertex Cover**).
Input: A graph G and integer k.
Goal: Is there a vertex cover of size $\leq k$ in G?

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

2.3.1 Relationship between...

2.3.1.1 Vertex Cover and Independent Set

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

Proof: ($\Rightarrow$) Let S be an independent set
(A) Consider any edge $uv \in E$.
(B) Since S is an independent set, either $u \notin S$ or $v \notin S$.
(C) Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
(D) $V \setminus S$ is a vertex cover.
($\Leftarrow$) Let $V \setminus S$ be some vertex cover:
(A) Consider $u, v \in S$
(B) uv is not an edge of G, as otherwise $V \setminus S$ does not cover uv.
(C) $\implies$ S is thus an independent set.

2.3.1.2 Independent Set $\leq_P$ Vertex Cover

(A) (G, k): instance of the **Independent Set** problem.
G: graph with n vertices. k: integer.
(B) G has an independent set of size $\geq k$ $\iff$ G has a vertex cover of size $\leq n - k$
(C) (G, k) is an instance of **Independent Set** , and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.
(D) We conclude:
(A) **Independent Set** $\leq_P$ **Vertex Cover**.
(B) **Vertex Cover** $\leq_P$ **Independent Set**.
(Because same reduction works in other direction.)

2.4 Vertex Cover and Set Cover

2.4.0.1 The Set Cover Problem

Problem 2.4.1 (**Set Cover**).
Input: Given a set U of n elements, a collection $S_1, S_2, \ldots S_m$ of subsets of U, and an integer k.
Goal: Is there a collection of at most k of these sets S_i whose union is equal to U?

Example 2.4.2. ¡2-¿Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

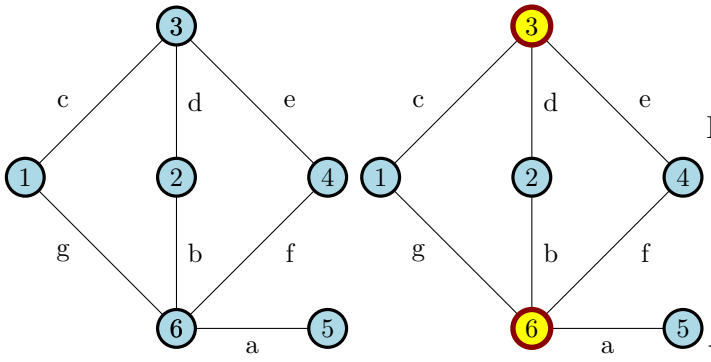
$$S_1 = \{3, 7\} \quad \text{¡3-} > S_2 = \{3, 4, 5\}$$
$$S_3 = \{1\} \quad S_4 = \{2, 4\}$$
$$S_5 = \{5\} \quad \text{¡3-} > S_6 = \{1, 2, 6, 7\}$$

$\{S_2, S_6\}$ is a set cover

2.4.0.2 Vertex Cover $\leq_P$ Set Cover

(A) Instance of **Vertex Cover**: $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ and integer k.
(B) Construct an instance of **Set Cover** as follows:
  (A) Number k for the **Set Cover** instance is the same as the number k given for the **Vertex Cover** instance.
  (B) $U = \mathsf{E}$.
  (C) We will have one set corresponding to each vertex; $S_v = \{e \mid e \text{ is incident on } v\}$.
(C) Observe that G has vertex cover of size k if and only if $U, \{S_v\}_{v \in V}$ has a set cover of size k. (Exercise: Prove this.)

2.4.0.3 Vertex Cover $\leq_P$ Set Cover: Example



Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with

$S_1 = \{c, g\}$ $S_2 = \{b, d\}$
¡3− > $S_3 = \{c, d, e\}$ $S_4 = \{e, f\}$
$S_5 = \{a\}$ ¡3− > $S_6 = \{a, b, f, g\}$

$\{S_3, S_6\}$ is a set cover

$\{3, 6\}$ is a vertex cover

2.4.0.4 Proving Reductions

To prove that $X \leq_P Y$ you need to give an algorithm $\mathcal{A}$ that:
(A) Transforms an instance I_X of X into an instance I_Y of Y.
(B) Satisfies the property that answer to I_X is YES $\iff$ I_Y is YES.
  (A) typical easy direction to prove: answer to I_Y is YES if answer to I_X is YES
  (B) **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).
(C) Runs in *polynomial* time.

2.4.0.5 Summary

(A) **polynomial-time reductions**.
  (A) If $X \leq_P Y$ + have efficient algorithm for Y $\implies$ efficient algorithm for X.
  (B) If $X \leq_P Y$ + no efficient algorithm for X $\implies$ **no** efficient algorithm for Y.
(B) Examples of reductions between **Independent Set**, **Clique**, **Vertex Cover**, and **Set Cover**.

2.5 The Satisfiability Problem (SAT)

2.5.0.1 Propositional Formulas

Definition 2.5.1. Consider a set of boolean variables $x_1, x_2, \ldots x_n$.
(A) ***literal***: boolean variable x_i or its negation $\neg x_i$ (also written as $\overline{x_i}$).
(B) ***clause***: a disjunction of literals. Example: $x_1 \vee x_2 \vee \neg x_4$.
(C) ***conjunctive normal form*** (CNF) = propositional formula which is a conjunction of clauses

(A) $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a CNF formula.

(D) A formula φ is a 3CNF:
A CNF formula such that every clause has **exactly** 3 literals.
(A) $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ is a 3CNF formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is not.

2.5.0.2 Satisfiability

SAT

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

3SAT

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

2.5.0.3 Satisfiability

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

Example 2.5.2. (A) $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is satisfiable; take $x_1, x_2, \ldots x_5$ to be all true
(B) $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ 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...)

2.5.0.4 Importance of SAT and 3SAT

(A) **SAT**, **3SAT**: basic constraint satisfaction problems.
(B) Many different problems can reduced to them: simple+powerful expressivity of constraints.
(C) Arise in many hardware/software verification/correctness applications.
(D) ... fundamental problem of **NP-Complete**ness.

2.5.1 Converting a boolean formula with 3 variables to 3SAT

2.5.1.1 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

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

2.5.1.2 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	00	$z \vee \overline{x} \vee \overline{y}$
1	0	0	00	$\overline{z} \vee x \vee y$
1	0	1	00	$\overline{z} \vee x \vee y$
1	1	0	00	$\overline{z} \vee x \vee y$
1	1	1	1	

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

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

2.5.2.1 Simplify further if you want to

(A) Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
  (A) $\Big(\overline{z} \vee x \vee u\Big) \wedge \Big(\overline{z} \vee x \vee \overline{y}\Big) = \Big(\overline{z} \vee x\Big)$
  (B) $\Big(\overline{z} \vee x \vee y\Big) \wedge \Big(\overline{z} \vee \overline{x} \vee y\Big) = \Big(\overline{z} \vee y\Big)$

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

Lemma 2.5.3. $\Big(z = x \wedge y\Big) \quad \equiv \quad \Big(z \vee \overline{x} \vee \overline{y}\Big) \wedge \Big(\overline{z} \vee x\Big) \wedge \Big(\overline{z} \vee y\Big)$

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

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	00	$z \vee x \vee \overline{y}$
0	1	0	00	$z \vee \overline{x} \vee y$
0	1	1	00	$z \vee \overline{x} \vee \overline{y}$
1	0	0	00	$\overline{z} \vee x \vee y$
1	0	1	1	
1	1	0	1	
1	1	1	1	

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

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

2.5.3.1 Simplify further if you want to

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

(A) Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
  (A) $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
  (B) $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$

(B) Using the above two observation, we have the following.

Lemma 2.5.4. *The formula $z = x \vee y$ is equivalent to the CNF formula* $\Big(z = x \vee y\Big) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$

2.5.3.2 Converting $z = \overline{x}$ to CNF

Lemma 2.5.5. $z = \overline{x} \equiv (z \vee x) \wedge (\overline{z} \vee \overline{x})$.

2.5.3.3 Converting into CNF: summary

Lemma 2.5.6.

$$\begin{aligned} z = \overline{x} &\equiv (z \vee x) \wedge (\overline{z} \vee \overline{x}). \\ z = x \vee y &\equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y) \\ z = x \wedge y &\equiv (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x) \wedge (\overline{z} \vee y) \end{aligned}$$

2.5.3.4 Exercise...

(A) **Given:**
  (A) $f(x_1, \ldots, x_d)$ a boolean function
  (B) Formally: $f : \{0,1\}^d \to \{0,1\}$.
(B) Prove that there is CNF formula that computes f.
(C) Prove that there is 3CNF formula that computes f.

2.5.4 SAT and 3SAT

2.5.4.1 SAT$\leq_P$ 3SAT

How **SAT** is different from **3SAT**?In **SAT** clauses might have arbitrary length: $1, 2, 3, \ldots$ variables:

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

In **3SAT** every clause must have ***exactly*** 3 different literals.
  Reduce from of **SAT** to **3SAT**: make all clauses to have 3 variables...
  Basic idea
(A) Pad short clauses so they have 3 literals.
(B) Break long clauses into shorter clauses.
(C) Repeat the above till we have a 3CNF.

2.5.4.2 3SAT$\leq_P$ SAT

(A) **3SAT** $\leq_P$ **SAT**.
(B) Because...
  A **3SAT** instance is also an instance of **SAT**.

2.5.4.3 SAT$\leq_P$ 3SAT

Claim 2.5.7. **SAT** $\leq_P$ **3SAT**.

  Given φ a **SAT** formula we create a **3SAT** formula φ' such that
(A) φ is satisfiable iff φ' is satisfiable.
(B) 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.

2.5.5 SAT$\leq_P$ 3SAT

2.5.5.1 A clause with a single literal

Reduction Ideas **Challenge:** Some clauses in φ # liters $\neq 3$.
  $\forall$ clauses with $\neq 3$ literals: construct set logically equivalent clauses.
(A) **Clause with one literal:** $c = \ell$ clause with a single literal.
  u, v be new variables. Consider

$$\begin{aligned} c' = \quad & \Big(\ell \vee u \vee v\Big) \wedge \Big(\ell \vee u \vee \neg v\Big) \\ & \wedge \Big(\ell \vee \neg u \vee v\Big) \wedge \Big(\ell \vee \neg u \vee \neg v\Big). \end{aligned}$$

Observe: c' satisfiable $\iff$ c is satisfiable

2.5.6 SAT$\leq_P$ 3SAT

2.5.6.1 A clause with two literals

Reduction Ideas: 2 and more literals
(A) **Case clause with 2 literals:** Let $c = \ell_1 \vee \ell_2$. Let u be a new variable. Consider

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

c is satisfiable $\iff$ c' is satisfiable

2.5.6.2 Breaking a clause

Lemma 2.5.8. *For any boolean formulas X and Y and z a new boolean variable. Then*

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

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

$$\Big(X \vee z\Big) \wedge \Big(Y \vee \neg z\Big) \text{ is satisfiable}$$

(with the same assignment to the variables appearing in X and Y).

2.5.7 SAT$\leq_P$ 3SAT (contd)

2.5.7.1 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

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

Claim 2.5.9. *c is satisfiable $\iff$ c' is satisfiable.*

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

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

2.5.7.2 An Example

Example 2.5.10.

$$\begin{aligned} \varphi = &\Big(\neg x_1 \vee \neg x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \\ &\wedge \Big(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1\Big) \wedge \Big(x_1\Big). \end{aligned}$$

Equivalent form:

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

2.5.8 Overall Reduction Algorithm

2.5.8.1 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 satisfiable

... $\forall c \in \varphi$: new 3CNF formula c' is equivalent to c.

2.5.8.2 What about 2SAT?

(A) **2SAT** can be solved in poly time! (specifically, linear time!)
(B) No poly time reduction from **SAT** (or **3SAT**) to **2SAT**.
(C) If $\exists$ reduction $\implies$ **SAT**, **3SAT** solvable in polynomial time.
Why the reduction from **3SAT** to **2SAT** fails?

$(x \vee y \vee z)$: clause.

convert to collection of 2CNF clauses. Introduce a fake variable α, and rewrite this as

$$\begin{aligned} & (x \vee y \vee \alpha) \wedge (\neg\alpha \vee z) && \text{(bad! clause with 3 vars)} \\ \text{or} \quad & (x \vee \alpha) \wedge (\neg\alpha \vee y \vee z) && \text{(bad! clause with 3 vars).}\end{aligned}$$

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

2.5.9 Reducing 3SAT to Independent Set

2.5.9.1 Independent Set

Independent Set

Instance: A graph G, integer k.
Question: Is there an independent set in G of size k?

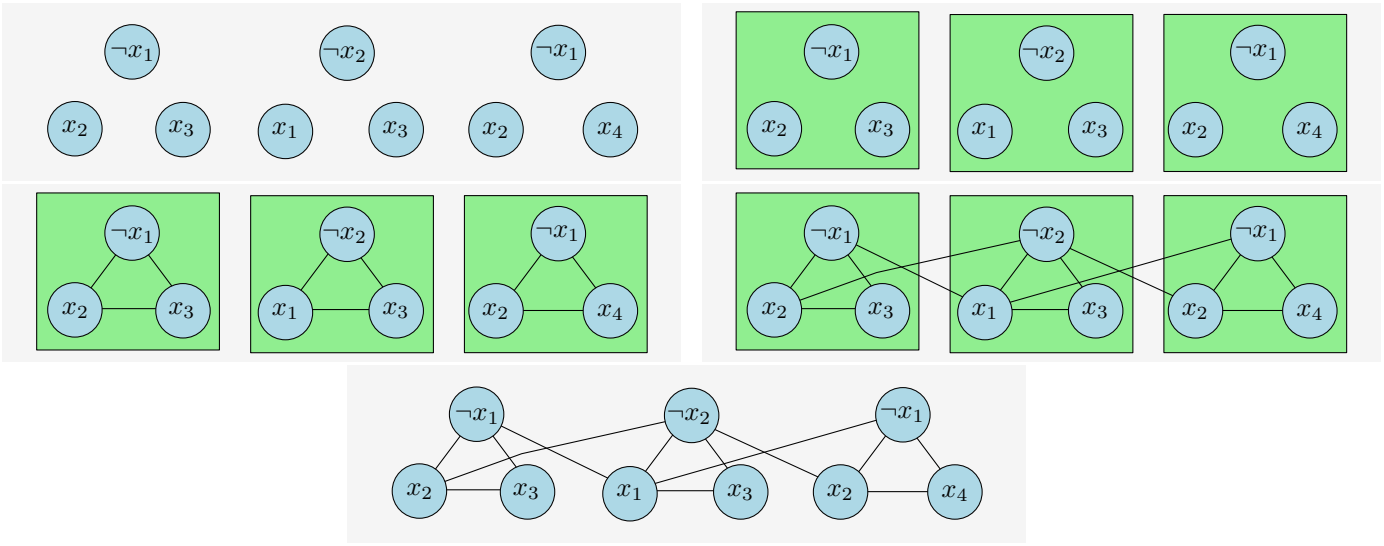


Figure 2.1: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

2.5.9.2 3SAT$\leq_P$ Independent Set

The reduction **3SAT$\leq_P$ Independent Set Input:** Given a 3CNF formula φ

Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.

G_φ should be constructable in time polynomial in size of φ

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

(B) **Notice:** Handle only 3CNF formulas (fails for other kinds of boolean formulas).

2.5.9.3 Interpreting 3SAT

There are two ways to think about **3SAT**

(A) Assign 0/1 (false/true) to vars $\implies$ formula evaluates to true.
Each clause evaluates to true.

(B) Pick literal from each clause & find assignment s.t. all true.
... Fail if two literals picked are in ***conflict***,
e.g. you pick x_i and $\neg x_i$

Use second view of **3SAT** for reduction.

2.5.9.4 The Reduction

(A) G_φ will have one vertex for each literal in a clause

(B) 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

(C) Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict

(D) Take k to be the number of clauses

2.5.9.5 Correctness

Proposition φ is satisfiable $\iff$ G_φ has an independent set of size k

k: number of clauses in φ. ■

Proof: $\Rightarrow$ a: truth assignment satisfying φ

(A) Pick one of the vertices, corresponding to true literals under a, from each triangle. This is an independent set of the appropriate size

2.5.9.6 Correctness (contd)

Proposition φ is satisfiable $\iff$ G_φ has an independent set of size k (= number of clauses in φ). ■

Proof: $\Leftarrow$ S: independent set in G_φ of size k

(A) S must contain exactly one vertex from each clause

(B) S cannot contain vertices labeled by conflicting clauses

(C) 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