NEW CS 473: Theory II, Fall 2015

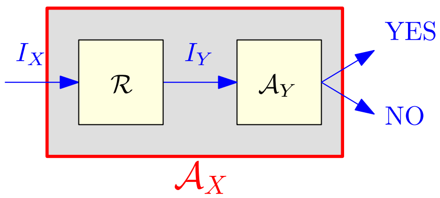
Reductions and NP

Lecture 2
August 27, 2015

Part I

Total recall...

Polynomial-time reductions



1. Algorithm is **efficient** if it runs in polynomial-time.
2. Interested only in polynomial-time reductions.
3. $X \leq_P Y$: Have polynomial-time reduction from problem X to problem Y.
4. $\mathcal{A}_Y$: poly-time algorithm for Y.
5. $\implies$ Polynomial-time/efficient algorithm for X.

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
$|I_Y| \leq p(|I_X|)$. $\square$

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

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 $|I_X|$. $\quad (|I_Y| =$ 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:
   1.1 $f(n) \leq a * n^b$ (i.e., $f(n) = O(n^b)$)
   1.2 $g(n) \leq c * n^d$ (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

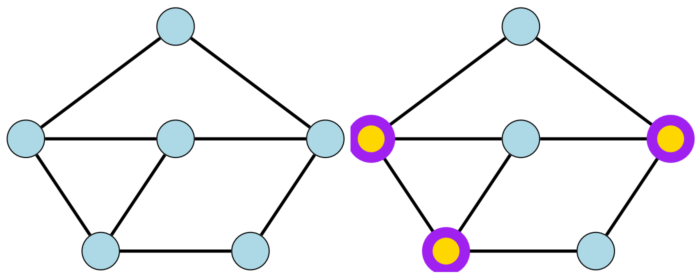
$X \leq_P Y$ and $Y \leq_P Z$ implies that $X \leq_P Z$.

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.

Vertex Cover

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

1. A **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* k.
Goal: *Is there a vertex cover of size* $\leq k$ *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 *is an independent set* $\iff$ $V \setminus S$ *is a vertex cover.*

Proof.

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

- 0.1 Consider any edge $uv \in E$.
- 0.2 Since S is an independent set, either $u \notin S$ or $v \notin S$.
- 0.3 Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
- 0.4 $V \setminus S$ is a vertex cover.

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

- 0.1 Consider $u, v \in S$
- 0.2 uv is not an edge of **G**, as otherwise $V \setminus S$ does not cover uv.
- 0.3 $\implies$ S is thus an independent set. $\square$

Independent Set $\leq_P$ **Vertex Cover**

1. $(\mathbf{G}, k)$: instance of the **Independent Set** problem.
   G: graph with n vertices. k: integer.
2. **G** has an independent set of size $\geq k$ $\iff$ **G** has a vertex cover of size $\leq n - k$
3. (G, k) is an instance of **Independent Set** , and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.
4. We conclude:
   - 4.1 **Independent Set** $\leq_P$ **Vertex Cover**.
   - 4.2 **Vertex Cover** $\leq_P$ **Independent Set**.
     (Because same reduction works in other direction.)

The **Set Cover** Problem

Problem (**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

Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

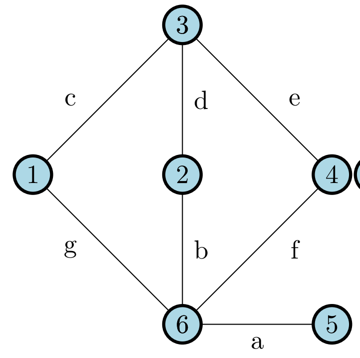
$$S_1 = \{3, 7\} \quad S_2 = \{3, 4, 5\}$$
$$S_3 = \{1\} \quad S_4 = \{2, 4\}$$
$$S_5 = \{5\} \quad S_6 = \{1, 2, 6, 7\}$$

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

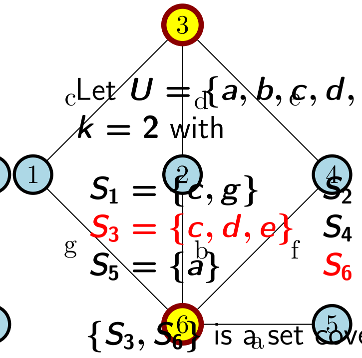
Vertex Cover $\leq_P$ Set Cover

1. Instance of **Vertex Cover**: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and integer k.
2. Construct an instance of **Set Cover** as follows:
   2.1 Number k for the **Set Cover** instance is the same as the number k given for the **Vertex Cover** instance.
   2.2 $U = \mathbf{E}$.
   2.3 We will have one set corresponding to each vertex; $S_v = \{e \mid e \text{ is incident on } v\}$.
3. Observe that $\mathbf{G}$ 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.)

Vertex Cover $\leq_P$ Set Cover: Example



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



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

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

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

Proving 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.
   2.1 typical easy direction to prove: answer to I_Y is YES if answer to I_X is YES
   2.2 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.

Summary

1. polynomial-time reductions.
   1.1 If $X \leq_P Y$ + have efficient algorithm for $Y \implies$ efficient algorithm for X.
   1.2 If $X \leq_P Y$ + no efficient algorithm for $X \implies$ **no** efficient algorithm for Y.
2. Examples of reductions between **Independent Set**, **Clique**, **Vertex Cover**, and **Set Cover**.

Propositional Formulas

Definition

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

1. **literal**: boolean variable x_i or its negation $\neg x_i$ (also written as $\overline{x_i}$).
2. **clause**: a disjunction of literals. Example: $x_1 \vee x_2 \vee \neg x_4$.
3. **conjunctive normal form** (CNF) = propositional formula which is a conjunction of clauses
   3.1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a CNF formula.
4. A formula φ is a 3CNF:
   A CNF formula such that every clause has **exactly** 3 literals.
   4.1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ 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.

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?

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 $x_1, x_2, \dots x_5$ 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**, **3SAT**: basic constraint satisfaction problems.
2. Many different problems can reduced to them: simple+powerful expressivity of constraints.
3. Arise in many hardware/software verification/correctness applications.
4. ... fundamental problem of **NP-Complete**ness.

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

z	x	y	$z = x \land y$	$z \lor \overline{x} \lor \overline{y}$	$\overline{z} \lor x \lor y$	$\overline{z} \lor x \lor \overline{y}$	$\overline{z} \lor \overline{x} \lor y$
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	00	1	1	1
1	0	0	0	1	00	1	1
1	0	1	0	1	1	00	1
1	1	0	0	1	1	1	00
1	1	1	1	1	1	1	1

$$\Big(z = x \land y\Big)$$
$$\equiv$$
$$(z \lor \overline{x} \lor \overline{y}) \land (\overline{z} \lor x \lor y) \land (\overline{z} \lor x \lor \overline{y}) \land (\overline{z} \lor \overline{x} \lor y)$$

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	00	$z \lor \overline{x} \lor \overline{y}$
1	0	0	00	$\overline{z} \lor x \lor y$
1	0	1	00	$\overline{z} \lor x \lor y$
1	1	0	00	$\overline{z} \lor x \lor y$
1	1	1	1	

$$\Big(z = x \land y\Big)$$
$$\equiv$$
$$(z \lor \overline{x} \lor \overline{y}) \land (\overline{z} \lor x \lor y) \land (\overline{z} \lor x \lor \overline{y}) \land (\overline{z} \lor \overline{x} \lor y)$$

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

Simplify further if you want to

1. Using that $(x \lor y) \land (x \lor \overline{y}) = x$, we have that:
   1.1 $\big(\overline{z} \lor x \lor u\big) \land \big(\overline{z} \lor x \lor \overline{y}\big) = \big(\overline{z} \lor x\big)$
   1.2 $\big(\overline{z} \lor x \lor y\big) \land \big(\overline{z} \lor \overline{x} \lor y\big) = \big(\overline{z} \lor y\big)$
2. Using the above two observation, we have that our formula $\psi \equiv \Big(z \lor \overline{x} \lor \overline{y}\Big) \land \Big(\overline{z} \lor x \lor y\Big) \land \Big(\overline{z} \lor x \lor \overline{y}\Big) \land \Big(\overline{z} \lor \overline{x} \lor y\Big)$
   is equivalent to
   $\psi \equiv \Big(z \lor \overline{x} \lor \overline{y}\Big) \land \Big(\overline{z} \lor x\Big) \land \Big(\overline{z} \lor y\Big)$

Lemma

$$\Big(z = x \land y\Big) \quad \equiv \quad \Big(z \lor \overline{x} \lor \overline{y}\Big) \land \Big(\overline{z} \lor x\Big) \land \Big(\overline{z} \lor y\Big)$$

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

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

$$\Big(z = x \lor y\Big)$$
$$\equiv$$
$$(z \lor x \lor \overline{y}) \land (z \lor \overline{x} \lor y) \land (z \lor \overline{x} \lor \overline{y}) \land (\overline{z} \lor x \lor y)$$

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:
   1.1 $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   1.2 $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
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 $z = \overline{x}$ to CNF

Lemma

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

Converting into CNF: summary

Lemma

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

Exercise...

1. **Given:**
   1.1 $f(x_1, \ldots, x_d)$ a boolean function
   1.2 Formally: $f : \{0,1\}^d \to \{0,1\}$.
2. Prove that there is CNF formula that computes f.
3. Prove that there is 3CNF formula that computes f.

SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

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

$$\left(x \lor y \lor z \lor w \lor u\right) \land \left(\neg x \lor \neg y \lor \neg z \lor w \lor u\right) \land \left(\neg x\right)$$

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

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

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 **3**, replace it with several clauses of length exactly **3**.

SAT $\leq_P$ 3SAT

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.

1. Clause with one literal: $c = \ell$ clause with a single literal.
   u, v be new variables. Consider

$$\begin{aligned} c' = \; & \left(\ell \lor u \lor v\right) \land \left(\ell \lor u \lor \neg v\right) \\ & \land \left(\ell \lor \neg u \lor v\right) \land \left(\ell \lor \neg u \lor \neg v\right). \end{aligned}$$

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

SAT $\leq_P$ 3SAT

A clause with two literals

Reduction Ideas: 2 and more literals

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

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

c is satisfiable $\iff$ c' is satisfiable

Breaking a clause

Lemma

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

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

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

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

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

SAT $\leq_P$ 3SAT (contd)

Clauses with more than **3** literals

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

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

Claim

c is satisfiable $\iff$ c' is satisfiable.

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

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

An Example

Example

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

Equivalent form:

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

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

What about **2SAT**?

1. **2SAT** can be solved in poly time! (specifically, linear time!)
2. No poly time reduction from **SAT** (or **3SAT**) to **2SAT**.
3. 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 $2\mathrm{CNF}$ clauses. Introduce a fake 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.)

Independent Set

Independent Set

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

3SAT $\leq_P$ Independent Set

The reduction **3SAT $\leq_P$ Independent Set**

Input: Given a $3\mathrm{CNF}$ 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

1. **Importance of reduction:** Although **3SAT** is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.
2. **Notice:** Handle only $3\mathrm{CNF}$ formulas (fails for other kinds of boolean formulas).

Interpreting 3SAT

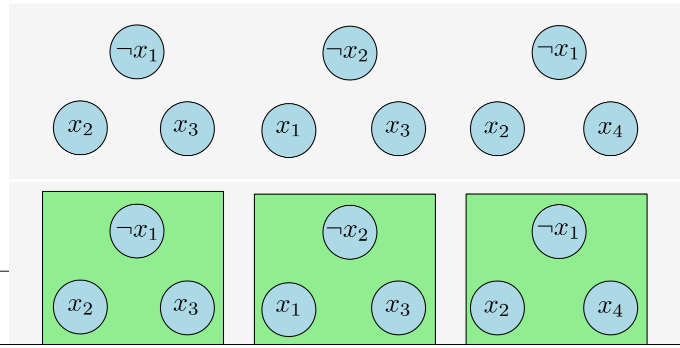
There are two ways to think about **3SAT**

1. Assign 0/1 (false/true) to vars $\implies$ formula evaluates to true.
   Each clause evaluates to true.
2. 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.

The Reduction

1. G_φ 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



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

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

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

0.1 S must contain exactly one vertex from each clause
0.2 S cannot contain vertices labeled by conflicting clauses
0.3 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 $\square$