CS 573: Algorithms, Fall 2014

Reductions and NP

Lecture 2

August 28, 2014

Part I

Reductions Continued

Propositional Formulas

Definition

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

1. A ***literal*** is either a boolean variable x_i or its negation $\neg x_i$.
2. A ***clause*** is a disjunction of literals.
   For example, $x_1 \vee x_2 \vee \neg x_4$ 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.

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 $3\mathrm{CNF}$:
   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

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 $3\mathrm{CNF}$ 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**, **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.

SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

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

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

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

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

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.

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 $\mathrm{3CNF}$.

3SAT $\leq_{\mathrm{P}}$ 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

- φ is satisfiable iff φ' is satisfiable.
- 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

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 $\neq \mathbf{3}$.
$\forall$ clauses with $\neq \mathbf{3}$ literals: construct set logically equivalent clauses.

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

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

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

SAT $\leq_P$ 3SAT

A clause with a single literal

Reduction Ideas

Challenge: Some clauses in φ # liters $\neq \mathbf{3}$.
$\forall$ clauses with $\neq \mathbf{3}$ literals: construct set logically equivalent clauses.

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

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

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

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

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

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 $c = \ell_1 \vee \ell_2$. Let u be a new variable. Consider

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

c is satisfiable $\iff$ c' 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

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

Claim

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

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

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

$\mathbf{SAT} \leq_P \mathbf{3SAT}$ (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

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

Claim

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

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

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

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

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

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

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

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

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

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

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

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 satisfiable

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

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 $\mathrm{3CNF}$ 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?

$(x \lor y \lor z)$: clause.
convert to collection of $2\mathrm{CNF}$ clauses. Introduce a fake variable α, and rewrite this as

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

or

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

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

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

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

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

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

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

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

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?

$(x \lor y \lor z)$: clause.
convert to collection of $2\mathrm{CNF}$ clauses. Introduce a fake variable α, and rewrite this as

$$
\begin{array}{lll}
 & (x \lor y \lor \alpha) \land (\neg\alpha \lor z) & \text{(bad! clause with 3 vars)} \\
\text{or} & (x \lor \alpha) \land (\neg\alpha \lor y \lor z) & \text{(bad! clause with 3 vars).}
\end{array}
$$

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

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

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

(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 $2\mathrm{CNF}$ 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...)

Independent Set

Independent Set

Instance: A graph $\mathbf{G}$, integer k.
Question: Is there an independent set in $\mathbf{G}$ of size k?

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

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

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

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

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).

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**

- Assign 0/1 (false/true) to vars $\implies$ formula evaluates to true.
  Each clause evaluates to true.
- Pick literal from each clause & find assignment s.t. all true.

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

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.

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

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.

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

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***,

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

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.

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

- G_φ will have one vertex for each literal in a clause
- 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
- Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- Take k to be the number of clauses

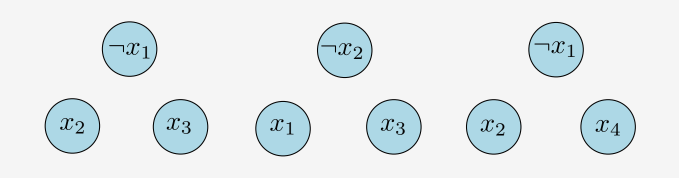

Figure: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

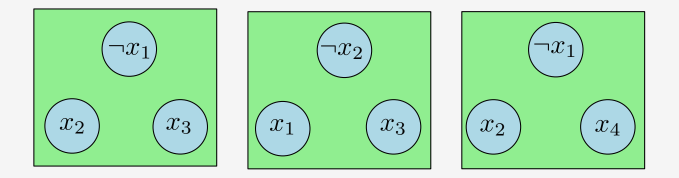

Figure: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

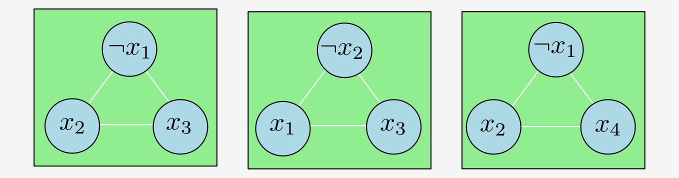

Figure: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

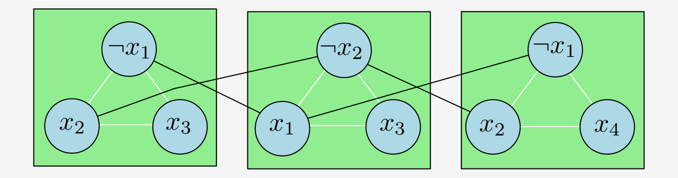

Figure: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

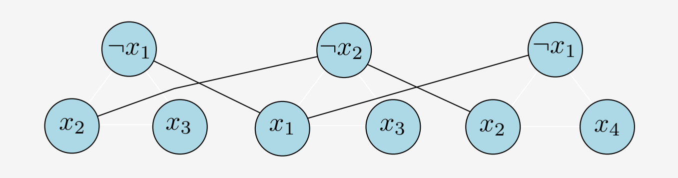

Figure: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

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

Correctness

Proposition

φ is satisfiable $\iff$ G_φ has an independent set of size k
k: number of clauses in φ.

Proof.

$\Rightarrow$ a: truth assignment satisfying φ

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

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

Transitivity of Reductions

Lemma

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

- Note: $X \leq_P Y$ does not imply that $Y \leq_P X$ and hence it is very important to know the FROM and TO in a reduction.
- To prove $X \leq_P Y$: show a reduction FROM X TO Y ... show $\exists$ algorithm for Y implies an algorithm for X.

Transitivity of Reductions

Lemma

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

- Note: $X \leq_P Y$ does not imply that $Y \leq_P X$ and hence it is very important to know the FROM and TO in a reduction.
- To prove $X \leq_P Y$: show a reduction FROM X TO Y ... show $\exists$ algorithm for Y implies an algorithm for X.

Transitivity of Reductions

Lemma

$X \leq_P Y$ 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 $X \leq_P Y$: show a reduction FROM X TO Y
   ... show $\exists$ algorithm for Y implies an algorithm for X.

Transitivity of Reductions

Lemma

$X \leq_P Y$ 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 $X \leq_P Y$: show a reduction FROM X TO Y ... show $\exists$ algorithm for Y implies an algorithm for X.

Part II

Definition of NP

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Recap ...

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\leq_P$ Clique

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\leq_P$ Clique

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\leq_P$ Clique$\leq_P$Independent Set

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\approx_P$ Clique

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\approx_P$ Clique
Independent Set $\leq_P$ Vertex Cover

Recap ...

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\approx_P$ Clique
Independent Set $\leq_P$ Vertex Cover $\leq_P$ Independent Set

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Independent Set $\approx_P$ Clique
Independent Set $\approx_P$ Vertex Cover

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover $\approx_P$ Independent Set $\approx_P$Clique

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover $\approx_P$ Independent Set $\approx_P$ Clique
3SAT $\leq_P$ SAT

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover $\approx_P$ Independent Set $\approx_P$Clique
3SAT $\leq_P$SAT$\leq_P$3SAT

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover $\approx_P$ Independent Set $\approx_P$Clique
3SAT $\approx_P$SAT

Recap . . .

Problems

1. Clique
2. Independent Set
3. Vertex Cover

1. Set Cover
2. SAT
3. 3SAT

Relationship

Vertex Cover $\approx_P$ Independent Set $\approx_P$Clique
3SAT $\approx_P$SAT
3SAT$\leq_P$Independent Set

Problems and Algorithms: Formal Approach

Decision Problems

1. Problem Instance: Binary string s, with size $|s|$
2. Problem: Set X of strings s.t. answer is “yes”: members of X are ***YES instances*** of X.
   Strings not in X are ***NO instances*** of X.

Definition

1. **alg**: algorithm for problem X if $\textbf{alg}(s) =$ ”yes” $\iff s \in X$.
2. **alg** have polynomial running time $\exists p(\cdot)$ polynomial s.t. $\forall s$, $\textbf{alg}(s)$ terminates in at most $O\Big(p\big(|s|\big)\Big)$ steps.

Problems and Algorithms: Formal Approach

Decision Problems

1. Problem Instance: Binary string s, with size $|s|$
2. Problem: Set X of strings s.t. answer is “yes”: members of X are ***YES instances*** of X.
   Strings not in X are ***NO instances*** of X.

Definition

1. alg: algorithm for problem X if $\text{alg}(s) = \text{"yes"} \iff s \in X$.
2. alg have polynomial running time $\exists p(\cdot)$ polynomial s.t. $\forall s$, $\text{alg}(s)$ terminates in at most $O\Big(p\big(|s|\big)\Big)$ steps.

Polynomial Time

Definition

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

Polynomial Time

Definition

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

Example

Problems in **P** include

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

Efficiency Hypothesis

Efficiency hypothesis.

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

1. Justifications:
   1. Robustness of definition to variations in machines.
   2. A sound theoretical definition.
   3. Most known polynomial time algorithms for "natural" problems have small polynomial running times.

Efficiency Hypothesis

Efficiency hypothesis.

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

1. Justifications:
   1. Robustness of definition to variations in machines.
   2. A sound theoretical definition.
   3. Most known polynomial time algorithms for "natural" problems have small polynomial running times.

Efficiency Hypothesis

Efficiency hypothesis.

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

1. Justifications:
   1. Robustness of definition to variations in machines.
   2. A sound theoretical definition.
   3. Most known polynomial time algorithms for “natural” problems have small polynomial running times.

Efficiency Hypothesis

Efficiency hypothesis.

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

1. Justifications:
   1. Robustness of definition to variations in machines.
   2. A sound theoretical definition.
   3. Most known polynomial time algorithms for "natural" problems have small polynomial running times.

Problems that are hard...

...with no known polynomial time algorithms

Problems

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

- undecidable problems are way harder (no algorithm at all!)
- ...but many problems want to solve: similar to above.
- Question: What is common to above problems?

Problems that are hard...

...with no known polynomial time algorithms

Problems

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

1. undecidable problems are way harder (no algorithm at all!)
2. ...but many problems want to solve: similar to above.
3. Question: What is common to above problems?

Problems that are hard...

...with no known polynomial time algorithms

Problems

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

1. undecidable problems are way harder (no algorithm at all!)
2. ...but many problems want to solve: similar to above.
3. Question: What is common to above problems?

Problems that are hard...

...with no known polynomial time algorithms

Problems

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

1. undecidable problems are way harder (no algorithm at all!)
2. ...but many problems want to solve: similar to above.
3. Question: What is common to above problems?

Efficient Checkability

- Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.

(B) Length of certificate $|C| \leq \mathbf{poly}(|I_X|)$.

(C) Given C, I_x: efficiently check that I_X is YES instance.

- Examples:
  - **SAT** formula φ: proof is a satisfying assignment.
  - **Independent Set** in graph G and k:
    Certificate: a subset S of vertices.

Efficient Checkability

- Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.
(B) Length of certificate $|C| \leq \mathrm{poly}(|I_X|)$.
(C) Given C, I_x: efficiently check that I_X is YES instance.

- Examples:
  - **SAT** formula φ: proof is a satisfying assignment.
  - **Independent Set** in graph G and k:
    Certificate: a subset S of vertices.

Efficient Checkability

1. Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.
(B) Length of certificate $|C| \leq \text{poly}(|I_X|)$.
(C) Given C, I_x: efficiently check that I_X is YES instance.

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

Efficient Checkability

1. Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.

(B) Length of certificate $|C| \leq \mathrm{poly}(|I_X|)$.

(C) Given C, I_x: efficiently check that I_X is YES instance.

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

Efficient Checkability

1. Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.

(B) Length of certificate $|C| \leq \mathrm{poly}(|I_X|)$.

(C) Given C, I_x: efficiently check that I_X is YES instance.

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

Efficient Checkability

1. Above problems have the property:

Checkability

For any YES instance I_X of X:

(A) there is a proof (or certificate) C.

(B) Length of certificate $|C| \leq \text{poly}(|I_X|)$.

(C) Given C, I_x: efficiently check that I_X is YES instance.

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

Certifiers

Definition

Algorithm $C(\cdot, \cdot)$ is ***certifier*** for problem X: $\forall s \in X$ there $\exists t$ such that $C(s,t) = \text{"YES"}$, and conversely, if for some s and t, $C(s,t) =$ "yes" then $s \in X$.

t is the certificate or proof for s.

Certifiers

Definition

Algorithm $C(\cdot,\cdot)$ is ***certifier*** for problem X: $\forall s \in X$ there $\exists t$ such that $C(s,t) =$ "YES", and conversely, if for some s and t, $C(s,t) =$ "yes" then $s \in X$.

---

t is the certificate or proof for s.

Certifiers

Definition

Algorithm $C(\cdot, \cdot)$ is ***certifier*** for problem X: $\forall s \in X$ there $\exists t$ such that $C(s, t) = \text{"YES"}$, and conversely, if for some s and t, $C(s, t) =$ "yes" then $s \in X$.

t is the certificate or proof for s.

Definition (Efficient Certifier.)

Certifier C is ***efficient certifier*** for X if there is a polynomial $p(\cdot)$ s.t. for every string s:

- ⋆ $s \in X$ if and only if
- ⋆ there is a string t:
  1. $|t| \leq p(|s|)$,
  2. $C(s, t) =$ "yes",
  3. and C 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 a of $0/1$ values to each variable.
    2. Certifier: Check each clause under a and say "yes" if all clauses are true.

Example:Composites

Composite

Instance: A number s.
Question: Is the number s a composite?

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

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:
  - I: instance.
  - c: proof/certificate that the instance is indeed a YES instance of the given problem.
- Think about C as algorithm for original problem, if:
  - Given I, the algorithm guess (non-deterministically, and who knows how) the certificate c.
  - The algorithm now verifies the certificate c for the instance I.
- Usually **NP** is described using Turing machines (gag).

Why is it called...

Nondeterministic Polynomial Time

1. 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.
2. Think about C as algorithm for original problem, if:
   1. Given I, the algorithm guess (non-deterministically, and who knows how) the certificate c.
   2. The algorithm now verifies the certificate c for the instance I.
3. Usually **NP** is described using Turing machines (gag).

Why is it called...

Nondeterministic Polynomial Time

1. 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.
2. Think about C as algorithm for original problem, if:
   1. Given I, the algorithm guess (non-deterministically, and who knows how) the certificate c.
   2. The algorithm now verifies the certificate c for the instance I.
3. Usually **NP** is described using Turing machines (gag).

Why is it called...

Nondeterministic Polynomial Time

1. 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.
2. Think about C as algorithm for original problem, if:
   1. Given I, the algorithm guess (non-deterministically, and who knows how) the certificate c.
   2. The algorithm now verifies the certificate c for the instance I.
3. Usually **NP** is described using Turing machines (gag).

Asymmetry in Definition of NP

1. Only YES instances have a short proof/certificate. NO instances need not have a short certificate.
2. For example...

   Example

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

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

Asymmetry in Definition of NP

1. Only YES instances have a short proof/certificate. NO instances need not have a short certificate.
2. For example...

Example

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

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

Asymmetry in Definition of NP

1. Only YES instances have a short proof/certificate. NO instances need not have a short certificate.
2. For example...

Example

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

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

$\mathbf{P}$ versus $\mathbf{NP}$

Proposition

P $\subseteq$ **NP**.

For a problem in **P** no need for a certificate!

Proof.

Consider problem $X \in$ **P** with algorithm **alg**. Need to demonstrate that X has an efficient certifier:

- Certifier C (input s, t):
  runs **alg**(s) and returns its answer.
- C runs in polynomial time.
- If $s \in X$, then for every t, $C(s,t) =$ "YES".
- If $s \notin X$, then for every t, $C(s,t) =$ "NO". $\square$

P versus NP

Proposition

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

For a problem in **P** no need for a certificate!

Proof.

Consider problem $X \in$ **P** with algorithm **alg**. Need to demonstrate that X has an efficient certifier:

- Certifier C (input s, t):
  runs **alg**(s) and returns its answer.
- C runs in polynomial time.
- If $s \in X$, then for every t, $C(s, t) =$ "YES".
- If $s \notin X$, then for every t, $C(s, t) =$ "NO". □

P versus NP

Proposition

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

For a problem in P no need for a certificate!

Proof.

Consider problem $X \in \mathsf{P}$ with algorithm alg. Need to demonstrate that X has an efficient certifier:

1. Certifier C (input s, t): runs alg(s) and returns its 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". $\square$

P versus NP

Proposition

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

For a problem in P no need for a certificate!

Proof.

Consider problem $X \in \mathsf{P}$ with algorithm alg. Need to demonstrate that X has an efficient certifier:

1. Certifier C (input s, t):
   runs $\mathsf{alg}(s)$ and returns its answer.
2. C runs in polynomial time.
3. If $s \in X$, then for every t, $C(s,t) = \text{"YES"}$.
4. If $s \notin X$, then for every t, $C(s,t) = \text{"NO"}$. $\square$

$\mathbf{P}$ versus $\mathbf{NP}$

Proposition

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

For a problem in P no need for a certificate!

Proof.

Consider problem $X \in \mathsf{P}$ with algorithm alg. Need to demonstrate that X has an efficient certifier:

1. Certifier C (input s, t):
   runs $\mathsf{alg}(s)$ and returns its 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

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

For a problem in P no need for a certificate!

Proof.

Consider problem $X \in \mathsf{P}$ with algorithm **alg**. Need to demonstrate that X has an efficient certifier:

1. Certifier C (input s, t):
   runs $\mathsf{alg}(s)$ and returns its 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". $\square$

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

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: $O(2^n)$, $O(2^{n \log n})$, $O\left(2^{n^3}\right)$, $\ldots$

NP versus EXP

Proposition

$\mathbf{NP} \subseteq \mathbf{EXP}$.

Proof.

Let $X \in \mathbf{NP}$ 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. $\square$

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 $\mathbf{NP}$ efficiently solvable?

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

Is $\mathbf{NP}$ efficiently solvable?

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

Big Question

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

If $\mathrm{P} = \mathrm{NP} \ldots$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

P versus NP

Status

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

Consensus: Most people feel/believe $\mathbf{P} \neq \mathbf{NP}$.

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

Part III

Not for lecture: Converting any boolean formula into CNF

The dark art of formula conversion into CNF

Consider an arbitrary boolean formula ϕ defined over $\boldsymbol{k}$ variables. To keep the discussion concrete, consider the formula $\phi \equiv \boldsymbol{x}_k = \boldsymbol{x}_i \wedge \boldsymbol{x}_j$. We would like to convert this formula into an equivalent CNF formula.

Formula conversion into CNF

Step 1

Build a truth table for the boolean formula.

x_k	x_i	x_j	value of $x_k = x_i \land x_j$
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

Formula conversion into CNF

Step 1.5 - understand what a single CNF clause represents

Given an assignment, say, $x_k = 0$, $x_i = 0$ and $x_j = 1$, consider the CNF clause $x_k \vee x_i \vee \overline{x_j}$ (you negate a variable if it is assigned one). Its truth table is

x_k	x_i	x_j	$x_k \vee x_i \vee \overline{x_j}$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

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

Formula conversion into CNF

Step 2

Write down CNF clause for every row in the table that is zero.

x_k	x_i	x_j	$x_k = x_i \wedge x_j$	CNF clause
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$x_k \vee \overline{x_i} \vee \overline{x_j}$
1	0	0	0	$\overline{x_k} \vee x_i \vee x_j$
1	0	1	0	$\overline{x_k} \vee x_i \vee \overline{x_j}$
1	1	0	0	$\overline{x_k} \vee \overline{x_i} \vee x_j$
1	1	1	1	

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

Formula conversion into CNF

Step 3 - simplify if you want to

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

1. $(\overline{x_k} \vee x_i \vee x_j) \wedge (\overline{x_k} \vee x_i \vee \overline{x_j})$ is equivalent to $(\overline{x_k} \vee x_i)$.
2. $(\overline{x_k} \vee x_i \vee x_j) \wedge (\overline{x_k} \vee \overline{x_i} \vee x_j)$ is equivalent to $(\overline{x_k} \vee x_j)$.

Using the above two observation, we have that our formula $\psi \equiv (x_k \vee \overline{x_i} \vee \overline{x_j}) \wedge (\overline{x_k} \vee x_i \vee x_j) \wedge (\overline{x_k} \vee x_i \vee \overline{x_j}) \wedge (\overline{x_k} \vee \overline{x_i} \vee x_j)$

is equivalent to

$\psi \equiv (x_k \vee \overline{x_i} \vee \overline{x_j}) \wedge (\overline{x_k} \vee x_i) \wedge (\overline{x_k} \vee x_j)$.

We conclude:

Lemma

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

Notes

Notes

Notes

Notes