

### NP-Completeness (cont)

An algorithm *accepts*  $e(i)$  if it answers YES

An algorithm *rejects*  $e(i)$  if it answers NO

An algorithm *accepts*  $L$  in polynomial time if

$\exists c$  such that the algorithm accepts  $\forall e(i) \in L$  with  $|e(i)| = n$  within time  $O(n^c)$  and does not accept any  $e(i) \notin L$

An algorithm *decides*  $L$  in polynomial time if

$\exists c$  such that the algorithm accepts all  $e(i) \in L$  within time  $O(n^c)$  and rejects all  $e(i) \notin L$  within time  $O(n^c)$

1

### NP-Completeness (cont)

$\mathbf{P} = \{ \text{polynomially solvable concrete decision problems} \}$

$\mathbf{P} = \{ L \mid \exists \text{an algorithm that decides } L \text{ in polynomial time} \}$

**Theorem**

$\mathbf{P} = \{ L \mid \exists \text{an algorithm that accepts } L \text{ in polynomial time} \}$

**Proof:** If  $L \in \mathbf{P}$  then  $\exists$ an algorithm that decides  $L$  in polynomial time.

This algorithm also accepts  $L$  in polynomial time.

2

### NP-Completeness (cont)

**Proof: (cont)** Now suppose  $\exists$ an algorithm that accepts  $L$  in polynomial time,  $\mathbf{A}$ .

Then  $\exists c$  such that  $\mathbf{A}$  accepts  $L$  in time  $T_{\mathbf{A}}(n) = O(n^c)$ .

Modify  $\mathbf{A}$  by adding a timer which expires after time  $T_{\mathbf{A}}(n)$  and make  $\mathbf{A}$  reject when the timer expires if it hasn't already accepted by then.

This new algorithm decides  $L$  in time  $T_{\mathbf{A}}(n) + 1 = O(n^c)$ .

So  $\exists$ an algorithm that decides  $L$  in polynomial time.

**QED**

3

### NP-Completeness (cont)

**PCP** - a language which cannot be decided (undecidable)

Instance:  $n$  pairs of binary strings:  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

Question: Is there a sequence of indices  $i_1, i_2, \dots, i_m$

such that  $x_{i_1} x_{i_2} \dots x_{i_m} = y_{i_1} y_{i_2} \dots y_{i_m}$  ?

Example:  $(111,1), (110,11), (0,000), (101,011111), (010,101)$

$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)$

1	1	0	0	1	0	1	0	1	1	1	1	1	1
1	1	0	0	0	1	0	1	1	1	1	1	1	1
2	3	5	4	1	1	1	1	1	1	1	1	1	1

4

### NP-Completeness (cont)

The class **NP** (non-deterministic polynomial time)

Traditional model for non-deterministic computation :  
Non-deterministic Turing Machine (choice of moves)

Text (CLR): Certificates and verification algorithms

An algorithm  $A(x,y)$ , *verifies*  $x$  if there exists a  $y$  <sup>certificate</sup> such that  $A(x,y)=1$ .

$L = \{ x \mid \exists y \ A(x,y)=1 \}$  is the language verified by  $A(x,y)$

5

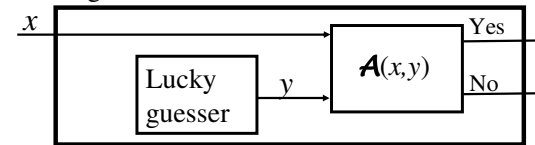
### NP-Completeness (cont)

An algorithm  $A(x,y)$ , *verifies L in polynomial time* if  $A(x,y)$  is a polynomial time algorithm and  $\exists c$ , a constant such that  $\forall x \in L \exists y$  a certificate for  $x$  ( $A(x,y)=1$ ) with

$$|y| = O(|x|^c)$$

$NP = \{ L \mid \exists \text{an algorithm that verifies } L \text{ in polynomial time} \}$

Algorithm for  $L$



6

### NP-Completeness (cont)

**3SAT**  $\in$  NP

Input:  $\langle \phi, \vec{X} \rangle$  a formula and a truth assignment.

Output:  $\phi(\vec{X})$ , the formula's value for the given truth assignment.

$A(\phi, \vec{X})$

```

for each clause of  $\phi$ 
  flag  $\leftarrow$  0
  for each literal in the clause
    if this literal is 1 under  $\vec{X}$  then flag  $\leftarrow$  1
  endfor
  if flag = 0 then return(0)
endfor
return(1)
  
```

7

### NP-Completeness (cont)

**CLIQUE**  $\in$  NP

Input:  $\langle G, k, V' \rangle$  a graph, an integer and a subset of vertices,  
 $V' = \{v_1, v_2, \dots, v_m\}$

Output: 1 if  $V'$  is a clique of size  $\geq k$  in  $G$ , and 0 otherwise.

$A(G, k, V')$

```

if  $m < k$  then return(0)
for  $i \leftarrow 1$  to  $m-1$ 
  for  $j \leftarrow i+1$  to  $m$ 
    if  $(v_i, v_j) \notin E$  then return(0)
  endfor
endfor
return(1)
  
```

8

### NP-Completeness (cont)

$P \subseteq NP$

What can be decided in polynomial time, can also be verified in polynomial time.

$co-NP = \{\bar{L} \mid L \in NP\}$

$\overline{3SAT} = \{\varphi \mid \varphi \text{ is a formula which is not satisfiable}\}$

(Assume any binary string which is not a legal encoding of a formula represents the formula **F**.)

$\overline{3SAT} \in co-NP$

9

### NP-Completeness (cont)

**COMPOSITE** =  $\{x \mid x \text{ is an integer and not prime}\}$

**COMPOSITE**  $\in NP$

(The certificate is a non-trivial factor of  $x$ .)

**PRIMES** =  $\overline{\text{COMPOSITE}} \in co-NP$

But it can also be shown that **PRIMES**  $\in NP$

So **PRIMES**  $\in co-NP \cap NP$

10

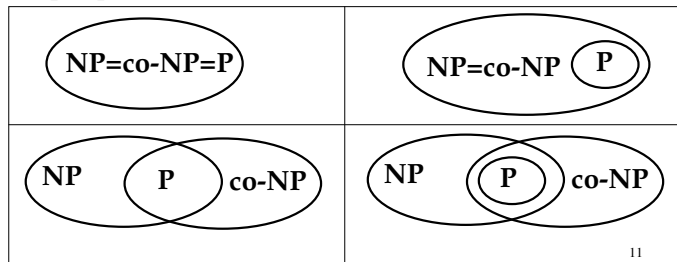
### NP-Completeness (cont)

$co-P = \{\bar{L} \mid L \in P\} = P$

If  $P = NP$ , then  $co-NP = co-P = P = NP$ .

If  $co-NP \neq NP$ , then  $P \neq NP$ .

4 open possibilities



11

### NP-Completeness (cont)

$L$  is *NP-complete* if

- 1)  $L \in NP$
- 2) for all  $L' \in NP$ ,  $L' \leq_p L$  (NP-hard)

If  $L$  is NP-complete, then  $L$  is “no easier” than anything in  $NP$ .

If  $L$  is NP-complete and  $L \in P$  then  $P = NP$ .

$NP = co-NP = P$

If  $L$  is NP-complete and  $P \neq NP$ , then  $L \notin P$ .

12

## NP-Completeness (cont)

How to show a problem is NP-complete?

**Cook's Theorem** **3SAT** is NP-complete.

Proof: Already showed **3SAT**  $\in$  **NP**.

For second requirement take a non-deterministic Turing Machine which accepts  $L'$  in polynomial time and any input  $x$ , and convert it to a 3CNF formula which is satisfiable if and only if the Turing Machine would accept  $x$  within the polynomial time bound.

13

## NP-Completeness (cont)

Text: **CIRCUIT-SAT** is NP-complete.

Proof sketch: Show that a verification algorithm and any input  $x$  can be converted to a boolean circuit which is satisfiable if and only if the verification algorithm would verify  $x$  within a polynomial time bound.

The first proof that a problem is NP-hard was hard.

Subsequent proofs can use the following lemma.

### Lemma

If  $L' \leq_p L$  and  $L'$  is NP-hard, then  $L$  is NP-hard.

14

## NP-Completeness (cont)

### Lemma

If  $L' \leq_p L$  and  $L'$  is NP-hard, then  $L$  is NP-hard.

**Proof** :  $\forall L'' \in \mathbf{NP}$ ,  $L'' \leq_p L'$  since  $L'$  is NP-hard.

Then  $L'' \leq_p L' \leq_p L$ , so  $L'' \leq_p L$ .

So  $L$  is NP-hard.

QED

### Corollary

If  $L \in \mathbf{NP}$  and  $L' \leq_p L$  and  $L'$  is NP-hard, then  $L$  is NP-complete.

15

## NP-Completeness (cont)

To show a problem  $\mathcal{Q}$  is NP-complete:

1) show the problem is in **NP**.

Come up with a polynomial verification algorithm.

2) show the problem is NP-hard.

Take a known NP-hard problem and reduce it to  $\mathcal{Q}$ .

16

### NP-Completeness (cont)

**3SAT, CLIQUE, VERTEX-COVER, and SUBSET-SUM** are NP-complete.

**Proof:** Already showed **3SAT** and **CLIQUE** are in **NP**.

**VERTEX-COVER** is in **NP**.

```
A( $G, k, V'$ )  $V' = \{v_1, v_2, \dots, v_m\}$ 
  if  $m > k$  then return(0)
  for each edge  $e = (v_i, v_j) \in E$ 
    if  $v_i \notin V'$  and  $v_j \notin V'$  then return(0)
  endfor
  return(1)
```

17

### NP-Completeness (cont)

**Proof: (cont)**

**SUBSET-SUM** is in **NP**.

```
A( $S, t, S'$ )  $S' = \{s_1, s_2, \dots, s_m\}$ 
  for  $i \leftarrow 1$  to  $m$ 
    if  $s_i \notin S$  then return(0)
     $T \leftarrow T + s_i$ 
  endfor
  if  $T \neq t$  then return(0)
  return(1)
```

18

### NP-Completeness (cont)

**Proof: (cont)** Using previous reductions,

since **CIRCUIT - SAT**  $\leq_p$  **3SAT**, **3SAT** is NP-hard.

since **3SAT**  $\leq_p$  **CLIQUE**, **CLIQUE** is NP-hard.

since **CLIQUE**  $\leq_p$  **VERTEX - COVER**,  
**VERTEX-COVER** is NP-hard.

since **VERTEX - COVER**  $\leq_p$  **SUBSET - SUM**,  
**SUBSET-SUM** is NP-hard. QED

If anyone of these problems is in **P**, then **P=NP**.

19

### NP-Completeness (cont)

#### 0-1 KNAPSACK

Instance: A  $n$  items where item  $i$  has weight  $w_i$  and cost  $c_i$  and two integers  $W$  and  $C$ .

Question: Is there a subset  $J$  of  $\{1, 2, \dots, n\}$  such that

$$\sum_{j \in J} w_j \leq W \text{ and } \sum_{j \in J} c_j \geq C ?$$

**0-1 KNAPSACK** is NP-complete.

20

### NP-Completeness (cont)

**0-1 KNAPSACK** is NP-complete.

**Proof:** First show that **0-1 KNAPSACK** is in NP.

$\mathcal{A}(W[], C[], W, C, J)$

```
A ← 0 B ← 0
for each j in J
  A ← A + W[j]
  B ← B + C[j]
endfor
if A > W or B < C then return(0)
return(1)
```

21

### NP-Completeness (cont)

**Proof:(cont)** Now show that **0-1 KNAPSACK** is NP-hard.

**SUBSET-SUM** is a known NP-hard problem.

**SUBSET - SUM**  $\leq_p$  **0-1 KNAPSACK**

Step 1: Construct  $f : \{(S, t)\} \rightarrow \{(W[], C[], W, C)\}$

Given  $(S, t)$  where  $S = \{s_1, s_2, \dots, s_k\}$ ,

let  $n = k$ ,  $w_i = c_i = s_i$ , and  $W = C = t$ .

• • •

22