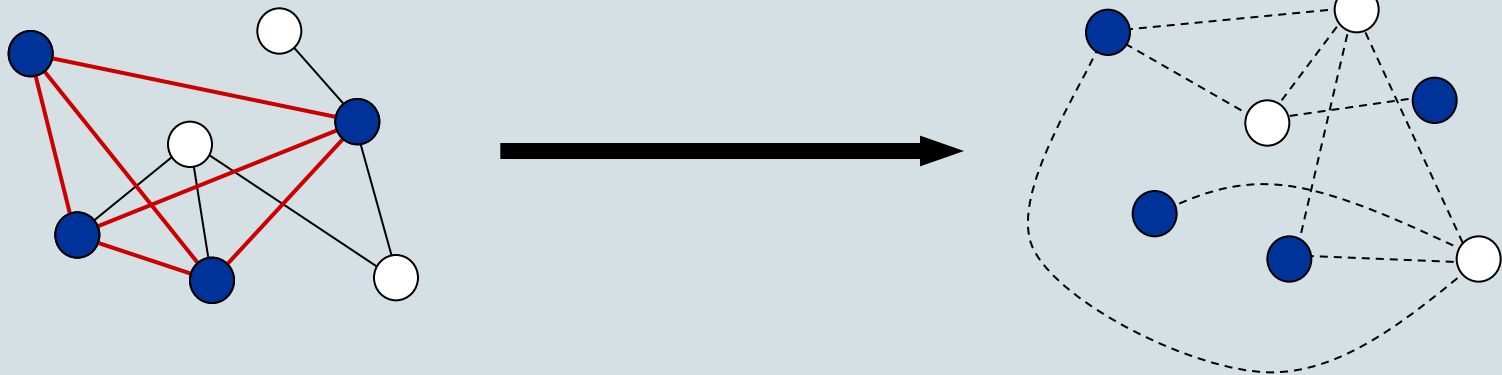


# Algorithms (2IL15) – Lecture 10

## NP-Completeness, II



## Summary of previous lecture

**P**: class of decision problems that can be solved in polynomial time

**NP**: decision problems for which there exists a polynomial-time verifier

algorithm  $A$  with two inputs

- input to the problem:  $x$
- certificate:  $y$

$A$  is **polynomial-time verifier**: for any  $x$  there **exists** certificate  $y$  such that  $A(x,y)$  outputs “yes” iff  $x$  is “yes”-instance, and  $A$  runs in polynomial time for such instances.

## Summary of previous lecture (cont'd)

### Reductions

problem A is **polynomial-time reducible** to problem B if there is a reduction algorithm mapping instances of A to instances of problem B such that

- “yes”-instances of A are mapped to “yes”-instances of B
- “no”-instances of A are mapped to “no”-instances of B
- the reduction algorithm runs in polynomial time

**Notation:** problem A  $\leq_p$  problem B

## Summary of previous lecture (cont'd)

**NP-complete problems:** problems  $A$  in NP such that  $B \leq_P A$  for any  $B$  in NP (if you can solve any NP-complete problem in polynomial time, then you can solve every NP-complete problem in polynomial time)

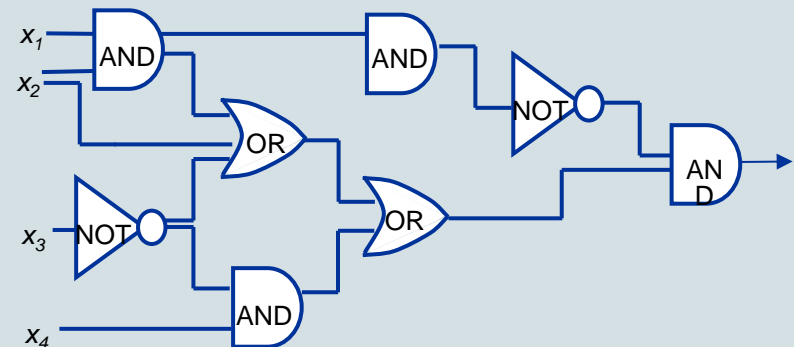
NP-complete problems cannot be solved in polynomial time, unless  $P = NP$

### Circuit-SAT

**Input:** combinatorial Boolean circuit

**Question:** Can variables be set such that formula evaluates to true ?

**Theorem:** Circuit-SAT is NP-complete



## Proving NP-completeness of other problems

**Theorem:** If problem A is NP-hard and problem  $A \leq_p$  problem B, then problem B is also NP-hard.

General strategy to prove that a problem B is NP-complete

1. Select problem A that is known to be NP-complete.
2. Prove that  $A \leq_p B$ :
  - i. Describe reduction algorithm, which maps instances  $x$  of A to instances  $f(x)$  of B.
  - ii. Prove that  $x$  is “yes”-instance for A iff  $f(x)$  is “yes”-instance for B
  - iii. Prove that reduction algorithm runs in polynomial time

(Now you have shown that B is NP-hard.)

3. Prove that B is in NP by giving polynomial-time verification algorithm.

## Summary of previous lecture (cont'd)

$$( (x_1 \rightarrow \neg x_3) \leftrightarrow (x_1 \vee \neg x_2 \vee x_3) ) \wedge ( \neg (x_2 \vee x_3 \vee x_5) \rightarrow (x_1 \vee x_3 \vee x_4) )$$

## SATISFIABILITY

**Input:** Boolean formula

**Question:** Can variables be set such that formula evaluates to true ?

**Theorem:** SATISFIABILITY is NP-complete

Proof by reduction from Circuit-SAT

**TODAY:** More reductions, more NP-complete problems

Boolean formula in 3-CNF form:

- “AND” of a number of clauses
- each clause the “OR” of exactly three literals

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4 \vee \neg x_5) \wedge (\neg x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_3 \vee x_4)$$

### 3-SAT

**Input:** Boolean formula in 3-CNF form

**Question:** Can variables be set such that formula evaluates to true ?

In the book problem is called 3-CNF-SAT, but most people just call it 3-SAT.

**Theorem:** 3-SAT is NP-complete

Proof.

Step 1: Select suitable NP-hard problem: Circuit-Sat or SATISFIABILITY ?

SATISFIABILITY

Step 2: Give polynomial-time reduction from SATISFIABILITY to 3-SAT

Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

$$( (x_1 \rightarrow \neg x_3) \leftrightarrow (x_1 \vee \neg x_2 \vee x_3) ) \wedge ( \neg (x_2 \vee x_3 \vee x_5) \rightarrow (x_1 \vee x_3 \vee x_4) )$$

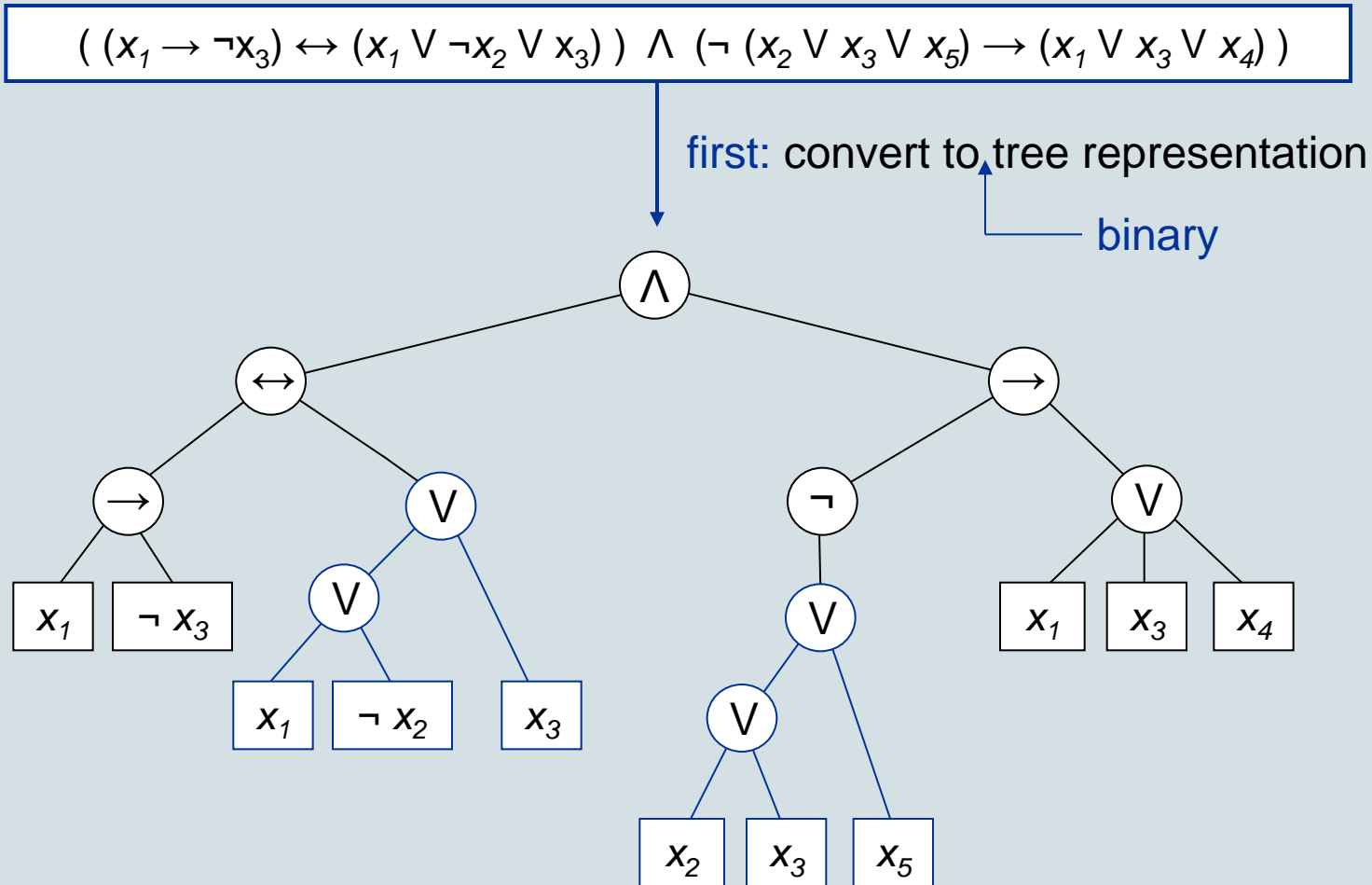
$$( .. \vee .. \vee .. ) \wedge \dots \wedge ( .. \vee .. \vee .. )$$



Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

- straightforward method does not work (gives exponential size  $F^*$ )
- we need more clever method

Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

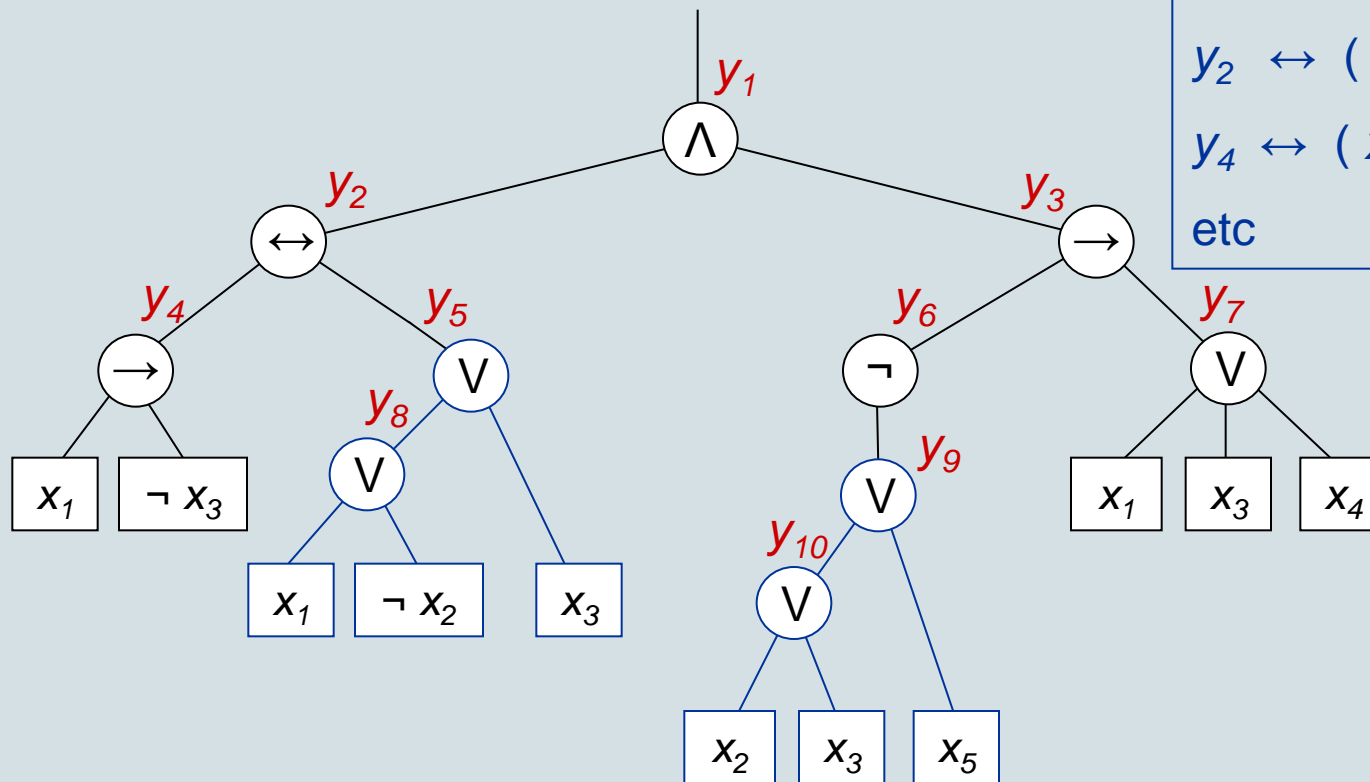


Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

second: – introduce extra variable  $y_i$  for output of every internal node

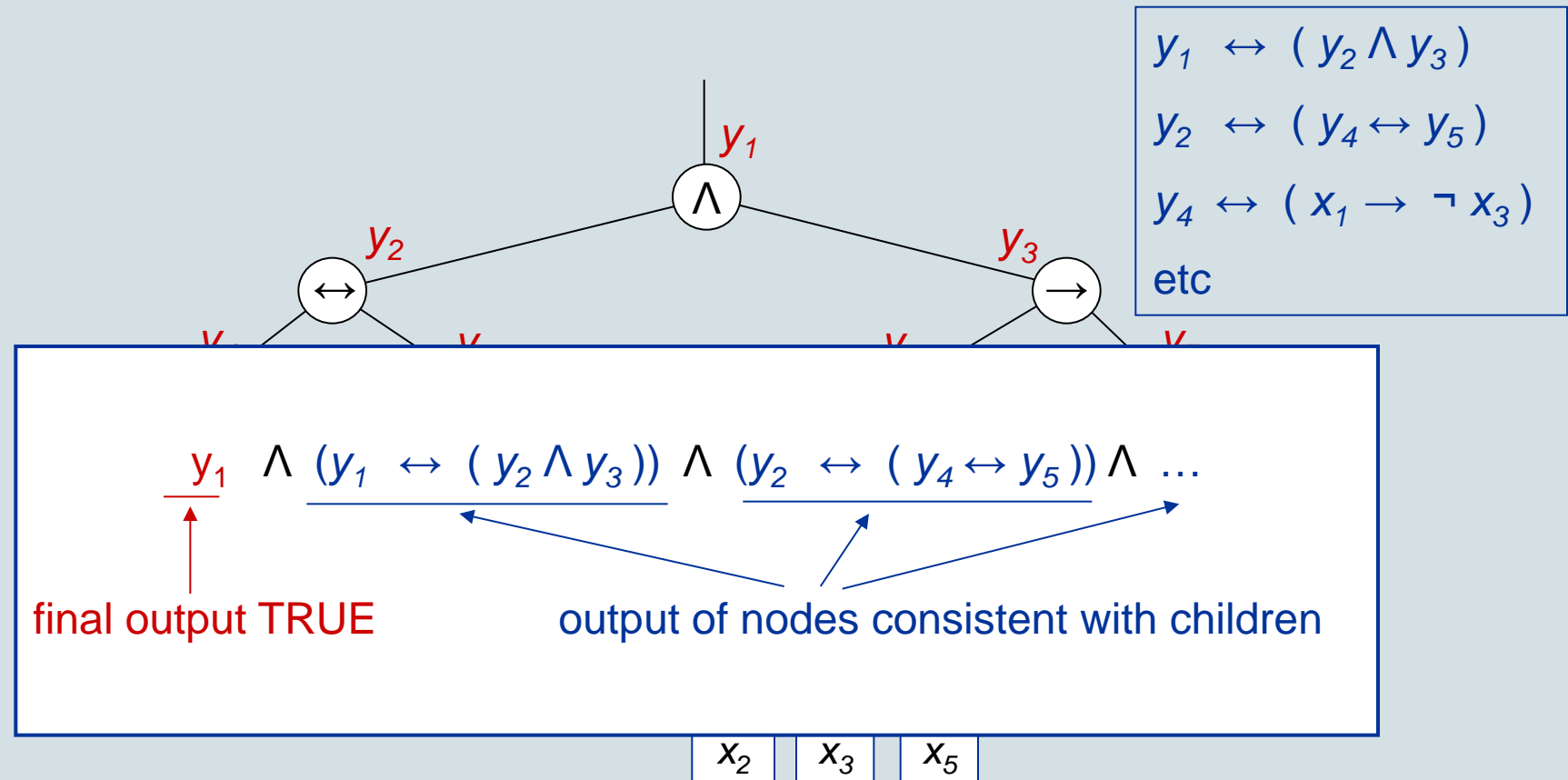
– write formulas for relations between variables:

$$\begin{aligned} y_1 &\leftrightarrow (y_2 \wedge y_3) \\ y_2 &\leftrightarrow (y_4 \leftrightarrow y_5) \\ y_4 &\leftrightarrow (x_1 \rightarrow \neg x_3) \\ &\text{etc} \end{aligned}$$



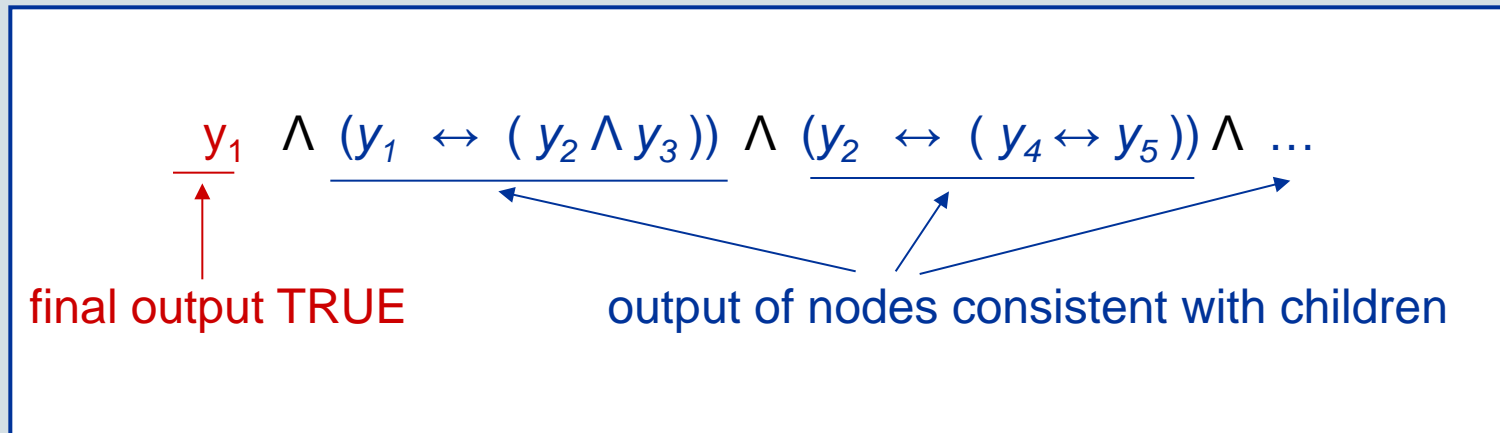
Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

third: – write formula to express satisfiability of the whole thing



Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

fourth: – rewrite each clause into CNF-form



$y_1$	$y_2$	$y_3$	$y_1 \leftrightarrow (y_2 \wedge y_3)$
1	1	1	1
1	1	0	0
1	0	1	0
...			...

clause is equivalent to:

$$\neg ( (y_1 \wedge y_2 \wedge \neg y_3) \vee (y_1 \wedge \neg y_2 \wedge y_3) \vee \dots )$$

Use De Morgan:  $\neg (a \wedge b) \equiv (\neg a \vee \neg b)$  etc

$$(\neg y_1 \vee \neg y_2 \vee y_3) \wedge (\neg y_1 \vee y_2 \vee \neg y_3) \dots$$

Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

after fourth step we have formula in CNF-form ...

$$y_1 \wedge (\neg y_1 \vee \neg y_2 \vee y_3) \wedge (\neg y_1 \vee y_2 \vee \neg y_3) \wedge \dots$$

... but some clauses have only one or two literals

fifth: – add extra variables and use them to “fill up” these clauses

For example: use extra variables  $p, q$  to replace  $y_1$  by

$$(y_1 \vee p \vee q) \wedge (y_1 \vee \neg p \vee q) \wedge (y_1 \vee p \vee \neg q) \wedge (y_1 \vee \neg p \vee \neg q)$$

Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form

after fifth step we have

- formula  $F^*$  in 3-CNF-form ...
- ... that is satisfiable if and only if original formula  $F$  is satisfiable
- ... and conversion can be done in polynomial time

**Theorem:** 3-SAT is NP-complete

Proof.

Step 1: Select suitable NP-hard problem: Circuit-Sat or SATISFIABILITY ?

SATISFIABILITY

Step 2: Give polynomial-time reduction from SATISFIABILITY to 3-SAT

Convert arbitrary Boolean formula  $F$  into formula  $F^*$  in 3-CNF form



Step 3. Prove 3-SAT is in NP by giving polynomial-time verification algorithm.

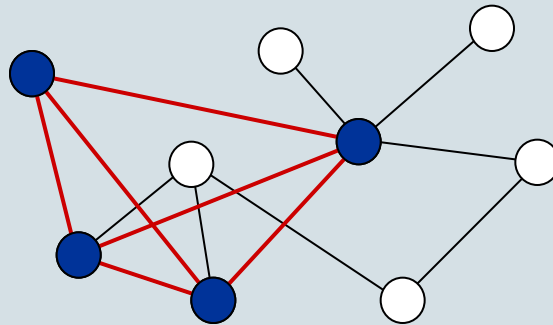
certificate = satisfying assignment





$G = (V, E)$  is undirected graph

**clique** in  $G$ : subset  $C \subseteq V$  such that  $(u, v) \in E$  for all pairs  $u, v$  in  $C$



## CLIQUE

**Input:** undirected graph  $G = (V, E)$  and a positive integer  $k$

**Question:** Does  $G$  have a clique of size  $k$ ?

**Theorem:** CLIQUE is NP-complete

Proof.

Step 1: Select suitable NP-hard problem: Circuit-Sat, SATISFIABILITY or 3-SAT ?

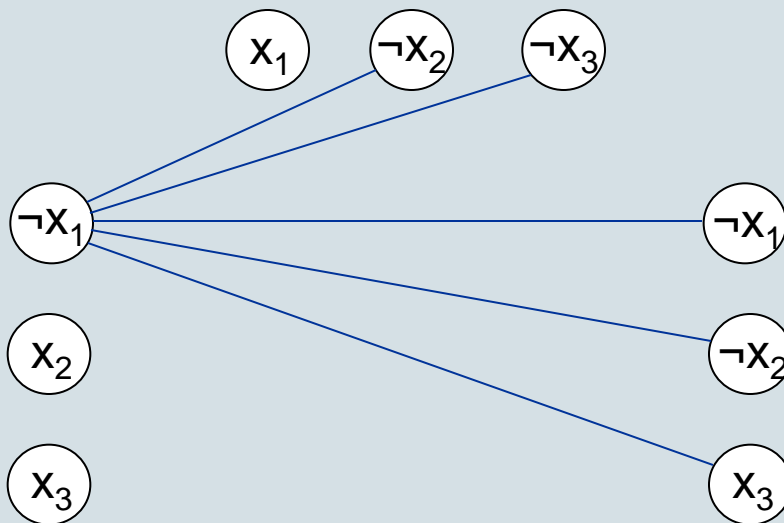
3-SAT

Step 2: Give polynomial-time reduction from 3-SAT to CLIQUE

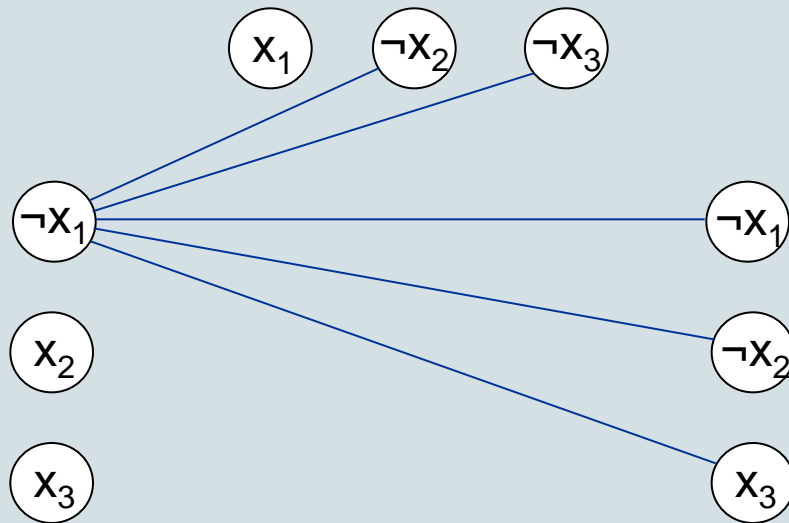
## Polynomial-time reduction from 3-SAT to CLIQUE

F: 3-SAT formula  $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

Construct graph  $G = (V, E)$  as follows



- introduce node for each literal in each clause of  $F$
- put edge between each pair of nodes such that
  - nodes are in different clauses
  - nodes are not each other's opposite



- introduce node for each literal in each clause of  $F$
- put edge between each pair of nodes such that
  - nodes are in different clauses
  - nodes are not each other's opposite

$k$  = number of clauses of  $F$

**Lemma:**  $F$  is satisfiable  $\iff G$  has clique of size at least  $k$

**Proof:**  $\implies$  : Assume  $F$  is satisfiable. For each clause, select TRUE node. Then these nodes must form a clique.

$\impliedby$  : Assume  $G$  has clique of size at least  $k$ .

Set variables such that these nodes evaluate to TRUE.

Must be a consistent setting that makes  $F$  true. ■

**Theorem:** CLIQUE is NP-complete

Proof.

Step 1: Select suitable NP-hard problem: Circuit-Sat, SATISFIABILITY or 3-SAT ?

3-SAT

Step 2: Give polynomial-time reduction from 3-SAT to CLIQUE

Reduction maps “yes”-instances to “yes”-instances  
and “no”-instances to “no”-instances.

Reduction runs in time  $O((\#clauses)^2)$ .



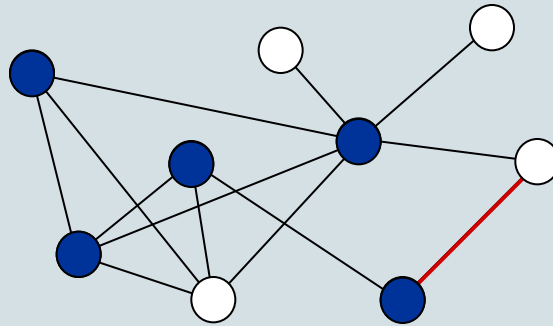
Step 3. Prove CLIQUE is in NP by giving polynomial-time verification algorithm.

certificate = subset of vertices forming clique of required size



$G = (V, E)$  is undirected graph

**vertex cover** in  $G$ : subset  $C \subset V$  such that for each edge  $(u, v)$  in  $E$  we have  $u$  in  $C$  or  $v$  in  $C$  (or both)



### Vertex Cover

**Input:** undirected graph  $G = (V, E)$  and a positive integer  $k$

**Question:** Does  $G$  have a vertex cover of size  $k$ ?

**Theorem:** Vertex Cover is NP-complete

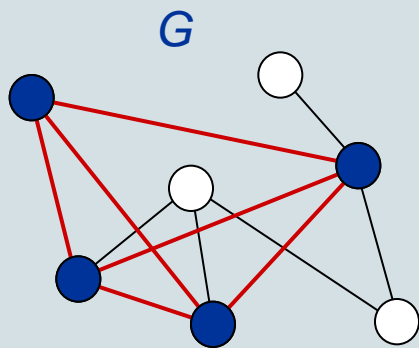
Proof.

Step 1: Select suitable NP-hard problem:

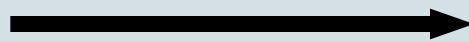
Circuit-Sat, SATISFIABILITY, 3-SAT, or CLIQUE?

CLIQUE

Step 2: Give polynomial-time reduction from CLIQUE to Vertex Cover

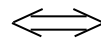


Construct complement  $\bar{G}$

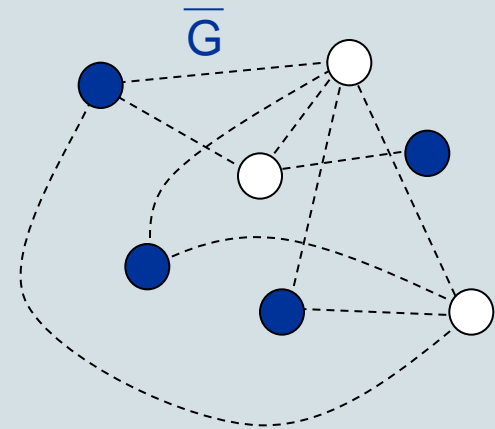


**Lemma:**

subset  $W \subseteq V$  is clique in  $G$



subset  $V - W$  is cover in  $\bar{G}$



**Theorem:** Vertex Cover is NP-complete

Proof.

Step 1: Select suitable NP-hard problem:

Circuit-Sat, SATISFIABILITY, 3-SAT, or CLIQUE?

CLIQUE

Step 2: Give polynomial-time reduction from CLIQUE to Vertex Cover

- $W$  is clique in  $G$  iff  $V - W$  is vertex cover in complement  $\overline{G}$ 
  - ➔  $G$  has clique of size  $\geq k$  iff  $\overline{G}$  has vertex cover of size  $\leq |V| - k$
- so “yes”-instances map to “yes”-instances, and “no”-instances ...
- Reduction can be done in  $O(|V|^2)$  time

Step 3. Prove Vertex Cover in NP by giving polynomial-time verification algorithm.  
certificate = subset of vertices that is cover of required size ■



## Subset Sum

**Input:** set  $X$  of non-negative integers, non-negative integer  $k$

**Question:** Does  $X$  have a subset  $S$  such that  $\sum_{x \in S} x = k$  ?

Example:  $X = \{ 1, 3, 3, 5, 7, 11, 17, 23, 41 \}$        $k = 25$

$S = \{ 3, 5, 17 \}$

**Theorem:** Subset Sum is NP-complete

Proof.

Step 1: Select suitable NP-hard problem:

Circuit-Sat, SATISFIABILITY, 3-SAT, CLIQUE, Vertex Cover ?

3-SAT

Step 2: Give polynomial-time reduction from 3-SAT to Subset Sum

Have to convert 3-SAT instance to Subset-Sum instance.

**WARNING:** When converting to a problem involving numbers, make sure that numbers do not become too large. Explicitly work with binary (or base-10) representation.

## A polynomial-time reduction from 3-SAT to Subset Sum

$F$ : 3-SAT formula with clauses  $C_1, \dots, C_m$  over variables  $x_1, \dots, x_n$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$$

Convert  $F$  to set  $S$  of  $2n+2m$  numbers, each consisting of  $n+m$  digits (base 10)

(First, remove clauses containing both  $x_i$  and  $\neg x_i$  for some  $i$ )

	$x_1$	$x_2$	$x_3$	$x_4$	$C_1$	$C_2$	$C_3$		
two numbers per variable, representing TRUE and FALSE	$v_{1,1}$	1	0	0	0	1	0	0	$v_{1,1} = 1000100$
	$v_{1,2}$	1	0	0	0	0	0	1	
	$v_{2,1}$	0	1	0	0	1	1	0	
	$v_{2,2}$	0	1	0	0	0	0	1	
	⋮								
two numbers per clause	$c_{1,1}$	0	0	0	0	1	0	0	$c_{1,2} = 0000200$
	$c_{1,2}$	0	0	0	0	2	0	0	
	⋮								

two numbers per variable, representing TRUE and FALSE

two numbers per clause

target sum (k)

	$x_1$	$x_2$	$x_3$	$x_4$	$C_1$	$C_2$	$C_3$
$V_{1,1}$	1	0	0	0	1	0	0
$V_{1,2}$	1	0	0	0	0	0	1
$V_{2,1}$	0	1	0	0	1	1	0
$V_{2,2}$	0	1	0	0	0	0	1
⋮							
$C_{1,1}$	0	0	0	0	1	0	0
$C_{1,2}$	0	0	0	0	2	0	0
⋮							

$V_{1,1} = 1000100$

$C_{1,2} = 0000200$

1 1 1 1 4 4 4

**Lemma:** F has satisfying assignment  $\iff$  S has subset summing to 1...14...4

must choose exactly one from each pair  $V_{i,1}, V_{i,2}$

for each clause, must make at least one literal TRUE;

if at least one literal TRUE, can select "clause numbers" to get to 4

$n$  times

$m$  times

**Theorem:** Subset Sum is NP-complete

Proof.

Step 1: Select suitable NP-hard problem:

Circuit-Sat, SATISFIABILITY, 3-SAT, CLIQUE, Vertex Cover ?

3-SAT

Step 2: Give polynomial-time reduction from 3-SAT to Subset Sum

Can convert 3-SAT instance to Subset-Sum instance in

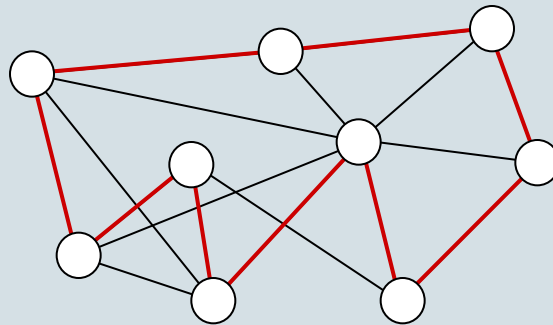
$O((n+m)^2)$  time, based on representation base-10.

Step 3. Prove Subset Sum in NP by giving polynomial-time verification algorithm.  
certificate = subset summing to  $k$ .



$G = (V, E)$  is undirected graph

**Hamiltonian cycle** in  $G$ : cycle that visits every vertex exactly once



### Hamiltonian Cycle

**Input:** undirected graph  $G = (V, E)$

**Question:** Does  $G$  have a Hamiltonian cycle?

**Theorem:** Hamiltonian Cycle is NP-complete

Proof.

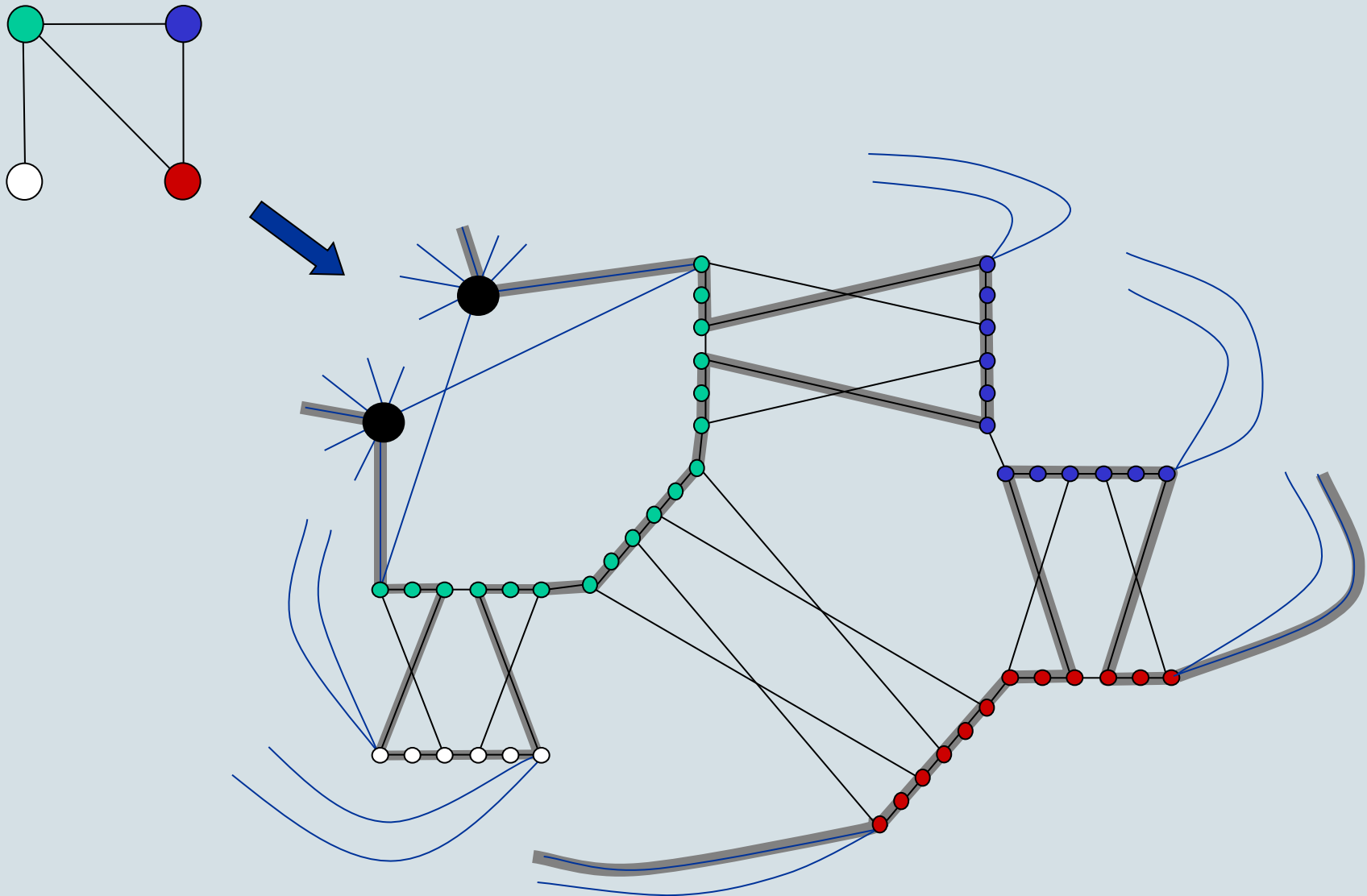
Step 1: Select suitable NP-hard problem:

Circuit-Sat, SATISFIABILITY, 3-SAT, CLIQUE, VertexCover, SubsetSum?

Vertex Cover

Step 2: Give polynomial-time reduction from Vertex Cover to Ham-Cycle

## A polynomial-time reduction from Vertex Cover to Ham-Cycle





**Theorem:** Hamiltonian Cycle is NP-complete

Proof.

Step 1: Select suitable NP-hard problem:

Circuit-Sat, SATISFIABILITY, 3-SAT, CLIQUE, VertexCover, SubsetSum?

Vertex Cover

Step 2: Give polynomial-time reduction from Vertex Cover to Ham-Cycle

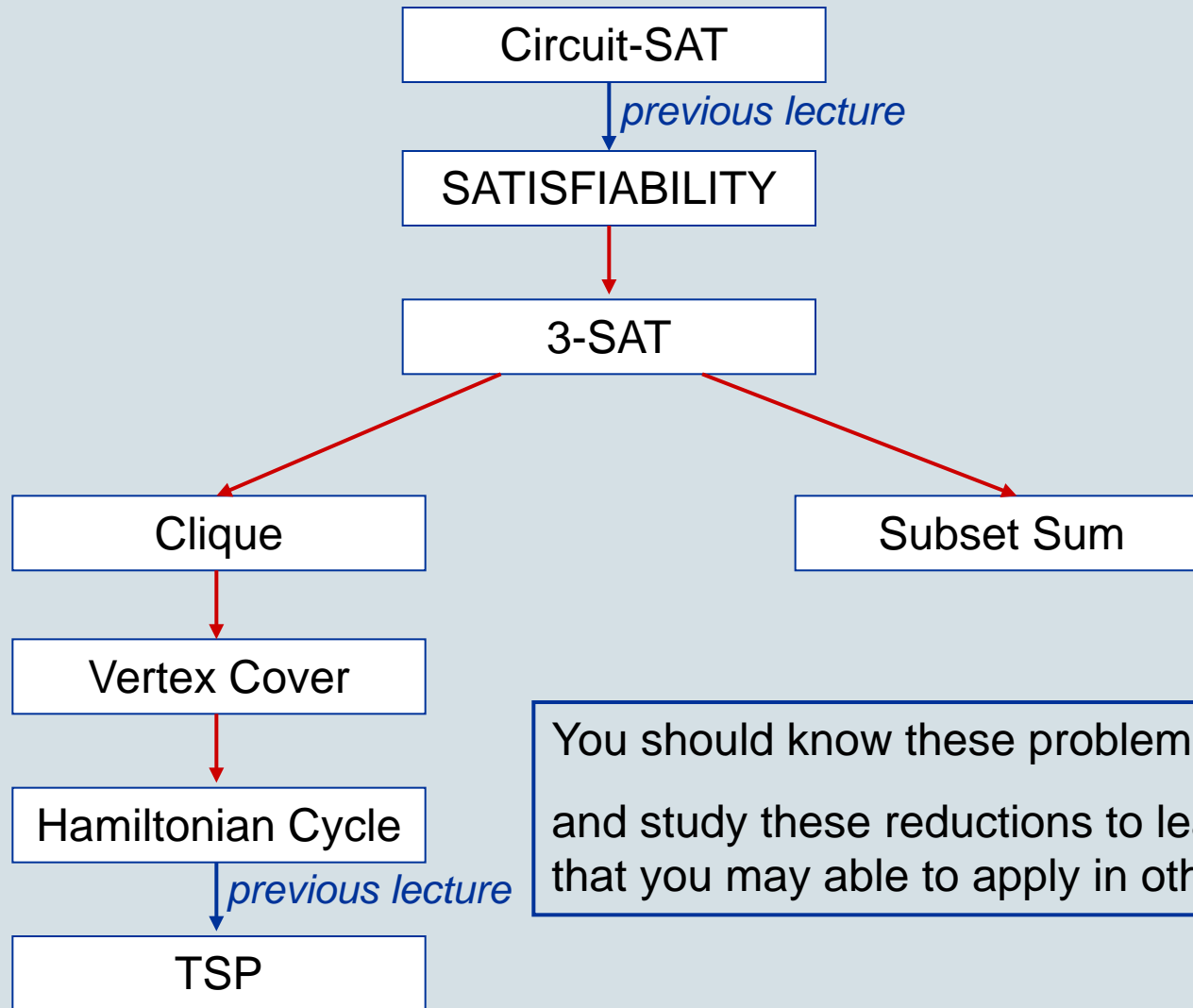
Complicated, but can be done.

Step 3. Prove Ham-Cycle in NP by giving polynomial-time verification algorithm.

certificate = permutation of vertices forming a cycle



## Summary



You should know these problems are NP-complete and study these reductions to learn some tricks that you may be able to apply in other proofs.