

# NP-Completeness, part II

Christian Wulff-Nilsen

Department of Computer Science  
University of Copenhagen  
Denmark

# Overview for today

- NP-completeness and reductions

# Overview for today

- NP-completeness and reductions
- NP-completeness of:

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT
  - ◆ CLIQUE

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT
  - ◆ CLIQUE
  - ◆ VERTEX-COVER

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT
  - ◆ CLIQUE
  - ◆ VERTEX-COVER
  - ◆ (HAM-CYCLE)



# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT
  - ◆ CLIQUE
  - ◆ VERTEX-COVER
  - ◆ (HAM-CYCLE)
  - ◆ TSP

# Overview for today

- NP-completeness and reductions
- NP-completeness of:
  - ◆ SAT
  - ◆ 3-CNF-SAT
  - ◆ CLIQUE
  - ◆ VERTEX-COVER
  - ◆ (HAM-CYCLE)
  - ◆ TSP
  - ◆ SUBSET-SUM

# Languages

- *Alphabet*: finite set  $\Sigma$  of symbols.

# Languages

- *Alphabet*: finite set  $\Sigma$  of symbols.
- *Language*  $L$  over alphabet  $\Sigma$ : a set of strings of symbols from  $\Sigma$ .

# Languages

- *Alphabet*: finite set  $\Sigma$  of symbols.
- *Language*  $L$  over alphabet  $\Sigma$ : a set of strings of symbols from  $\Sigma$ .
- We let  $\Sigma = \{0, 1\}$  so  $L$  is a set of binary strings.

# Languages

- *Alphabet*: finite set  $\Sigma$  of symbols.
- *Language*  $L$  over alphabet  $\Sigma$ : a set of strings of symbols from  $\Sigma$ .
- We let  $\Sigma = \{0, 1\}$  so  $L$  is a set of binary strings.
- Example:  $L = \{0, 10, 11001, 11101, \dots\}$ .

# Languages

- *Alphabet*: finite set  $\Sigma$  of symbols.
- *Language*  $L$  over alphabet  $\Sigma$ : a set of strings of symbols from  $\Sigma$ .
- We let  $\Sigma = \{0, 1\}$  so  $L$  is a set of binary strings.
- Example:  $L = \{0, 10, 11001, 11101, \dots\}$ .
- $\{0, 1\}^*$ : set of all binary strings.

# Decision problems and languages

- A *decision problem*  $Q$  consists of yes-instances and no-instances.



# Decision problems and languages

- A *decision problem*  $Q$  consists of yes-instances and no-instances.
- Example,  $Q = \text{HAM-CYCLE}$ :  $\langle G \rangle$  is a yes-instance if  $G$  contains a simple cycle containing all vertices of  $G$ ; otherwise  $\langle G \rangle$  is a no-instance.

# Decision problems and languages

- A *decision problem*  $Q$  consists of yes-instances and no-instances.
- Example,  $Q = \text{HAM-CYCLE}$ :  $\langle G \rangle$  is a yes-instance if  $G$  contains a simple cycle containing all vertices of  $G$ ; otherwise  $\langle G \rangle$  is a no-instance.
- We can view a problem  $Q$  as a mapping of yes-instances to 1 and no-instances to 0.

# Decision problems and languages

- A *decision problem*  $Q$  consists of yes-instances and no-instances.
- Example,  $Q = \text{HAM-CYCLE}$ :  $\langle G \rangle$  is a yes-instance if  $G$  contains a simple cycle containing all vertices of  $G$ ; otherwise  $\langle G \rangle$  is a no-instance.
- We can view a problem  $Q$  as a mapping of yes-instances to 1 and no-instances to 0.
- We can also view  $Q$  as a language  $L$ :

$$L = \{x \in \{0, 1\}^* \mid Q(x) = 1\}.$$

# Verifying a language

- A *verification algorithm* is an algorithm  $A$  taking two arguments,  $x, y \in \{0, 1\}^*$ , where  $y$  is the *certificate*.

# Verifying a language

- A *verification algorithm* is an algorithm  $A$  taking two arguments,  $x, y \in \{0, 1\}^*$ , where  $y$  is the *certificate*.
- $A$  *verifies* a string  $x$  if there is a certificate  $y$  such that  $A(x, y) = 1$ .

# Verifying a language

- A *verification algorithm* is an algorithm  $A$  taking two arguments,  $x, y \in \{0, 1\}^*$ , where  $y$  is the *certificate*.
- $A$  *verifies* a string  $x$  if there is a certificate  $y$  such that  $A(x, y) = 1$ .
- The language verified by  $A$  is

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ such that } A(x, y) = 1\}.$$

# The complexity class NP

- NP is the class of languages that can be verified in polynomial time.

# The complexity class NP

- NP is the class of languages that can be verified in polynomial time.
- In other words,  $L \in \text{NP}$  if and only if there is a polynomial-time verification algorithm  $A$  and a constant  $c$  such that

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}.$$



# The complexity class NP

- NP is the class of languages that can be verified in polynomial time.
- In other words,  $L \in \text{NP}$  if and only if there is a polynomial-time verification algorithm  $A$  and a constant  $c$  such that

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}.$$

- We saw that  $P \subseteq \text{NP}$ .

# The complexity class NP

- NP is the class of languages that can be verified in polynomial time.
- In other words,  $L \in \text{NP}$  if and only if there is a polynomial-time verification algorithm  $A$  and a constant  $c$  such that

$$L = \{x \in \{0, 1\}^* \mid \text{there is a } y \in \{0, 1\}^* \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}.$$

- We saw that  $P \subseteq \text{NP}$ .
- Big open problem: is  $P = \text{NP}$ ?

# Reducibility

- Language  $L_1$  is polynomial-time *reducible* to language  $L_2$  if there is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

# Reducibility

- Language  $L_1$  is polynomial-time *reducible* to language  $L_2$  if there is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

- We saw that

$$L_1 \leq_P L_2 \wedge L_2 \in P \Rightarrow L_1 \in P.$$

# NP-completeness

- Language  $L$  is *NP-complete* if

# NP-completeness

- Language  $L$  is *NP-complete* if
  1.  $L \in \text{NP}$  and

# NP-completeness

- Language  $L$  is *NP-complete* if
  1.  $L \in \text{NP}$  and
  2.  $L' \leq_P L$  for every  $L' \in \text{NP}$ .

# NP-completeness

- Language  $L$  is *NP-complete* if
  1.  $L \in \text{NP}$  and
  2.  $L' \leq_P L$  for every  $L' \in \text{NP}$ .
- $L$  is *NP-hard* if  $L$  satisfies property 2 (and possibly not property 1).



# NP-completeness

- Language  $L$  is *NP-complete* if
  1.  $L \in \text{NP}$  and
  2.  $L' \leq_P L$  for every  $L' \in \text{NP}$ .
- $L$  is *NP-hard* if  $L$  satisfies property 2 (and possibly not property 1).
- We saw that if any language of NPC belongs to P then  $P = \text{NP}$ .

# NP-completeness

- Language  $L$  is *NP-complete* if
  1.  $L \in \text{NP}$  and
  2.  $L' \leq_P L$  for every  $L' \in \text{NP}$ .
- $L$  is *NP-hard* if  $L$  satisfies property 2 (and possibly not property 1).
- We saw that if any language of NPC belongs to P then  $P = \text{NP}$ .
- We also saw that `CIRCUIT-SAT` is NP-complete.

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.
- If in addition  $L \in \text{NP}$  then  $L$  is NP-complete.

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.
- If in addition  $L \in \text{NP}$  then  $L$  is NP-complete.
- General technique for showing NP-completeness of a language  $L$ :

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.
- If in addition  $L \in \text{NP}$  then  $L$  is NP-complete.
- General technique for showing NP-completeness of a language  $L$ :
  - ◆ Show that  $L \in \text{NP}$ .

# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.
- If in addition  $L \in \text{NP}$  then  $L$  is NP-complete.
- General technique for showing NP-completeness of a language  $L$ :
  - ◆ Show that  $L \in \text{NP}$ .
  - ◆ Pick another language  $L'$  known to be NP-complete (for instance, `CIRCUIT-SAT`).



# NP-completeness of other problems via reduction

- Let  $L$  and  $L'$  be two languages with  $L' \in \text{NPC}$ .
- If  $L' \leq_P L$  then  $L$  is NP-hard.
- If in addition  $L \in \text{NP}$  then  $L$  is NP-complete.
- General technique for showing NP-completeness of a language  $L$ :
  - ◆ Show that  $L \in \text{NP}$ .
  - ◆ Pick another language  $L'$  known to be NP-complete (for instance, `CIRCUIT-SAT`).
  - ◆ Show that  $L' \leq_P L$ , i.e., show that there is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that

$$x \in L' \Leftrightarrow f(x) \in L.$$

# The SAT problem

- A *boolean formula*  $\phi$  consists of boolean variables  $x_1, \dots, x_n$ , boolean connectives  $\wedge, \vee$  and  $\neg, \rightarrow, \leftrightarrow$ , and parentheses ( and ).

# The SAT problem

- A *boolean formula*  $\phi$  consists of boolean variables  $x_1, \dots, x_n$ , boolean connectives  $\wedge, \vee$  and  $\neg, \rightarrow, \leftrightarrow$ , and parentheses ( and ).
- Example:  $\phi = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$ .

# The SAT problem

- A *boolean formula*  $\phi$  consists of boolean variables  $x_1, \dots, x_n$ , boolean connectives  $\wedge, \vee$  and  $\neg, \rightarrow, \leftrightarrow$ , and parentheses ( and ).
- Example:  $\phi = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$ .
- A *satisfying assignment* for a boolean formula  $\phi$  is an assignment of 0/1-values to variables that makes  $\phi$  evaluate to 1.

# The SAT problem

- A *boolean formula*  $\phi$  consists of boolean variables  $x_1, \dots, x_n$ , boolean connectives  $\wedge, \vee$  and  $\neg, \rightarrow, \leftrightarrow$ , and parentheses ( and ).
- Example:  $\phi = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$ .
- A *satisfying assignment* for a boolean formula  $\phi$  is an assignment of 0/1-values to variables that makes  $\phi$  evaluate to 1.
- $\phi$  is *satisfiable* if there exists a satisfying assignment for  $\phi$ .

# The SAT problem

- A *boolean formula*  $\phi$  consists of boolean variables  $x_1, \dots, x_n$ , boolean connectives  $\wedge, \vee$  and  $\neg, \rightarrow, \leftrightarrow$ , and parentheses ( and ).
- Example:  $\phi = (x_1 \vee x_2) \wedge (x_2 \vee x_3 \vee \neg x_4)$ .
- A *satisfying assignment* for a boolean formula  $\phi$  is an assignment of 0/1-values to variables that makes  $\phi$  evaluate to 1.
- $\phi$  is *satisfiable* if there exists a satisfying assignment for  $\phi$ .
- We can now define the problem SAT:

$$\text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable boolean formula} \}.$$

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .



# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .
- To show that  $\text{SAT} \in \text{NP}$ , we construct a verification algorithm  $A$  taking inputs  $x$  and  $y$ .

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .
- To show that  $\text{SAT} \in \text{NP}$ , we construct a verification algorithm  $A$  taking inputs  $x$  and  $y$ .
- It regards  $x$  as a boolean formula  $\phi$  and  $y$  as an assignment of values to variables of  $\phi$ .

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .
- To show that  $\text{SAT} \in \text{NP}$ , we construct a verification algorithm  $A$  taking inputs  $x$  and  $y$ .
- It regards  $x$  as a boolean formula  $\phi$  and  $y$  as an assignment of values to variables of  $\phi$ .
- $A$  returns 1 if  $y$  defines a satisfying assignment for  $\phi$ ; otherwise,  $A$  returns 0.

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .
- To show that  $\text{SAT} \in \text{NP}$ , we construct a verification algorithm  $A$  taking inputs  $x$  and  $y$ .
- It regards  $x$  as a boolean formula  $\phi$  and  $y$  as an assignment of values to variables of  $\phi$ .
- $A$  returns 1 if  $y$  defines a satisfying assignment for  $\phi$ ; otherwise,  $A$  returns 0.
- We can easily make  $A$  run in polynomial time.

# Showing that SAT is NP-complete

- To show  $\text{SAT} \in \text{NPC}$ , we follow our recipe:
  - ◆ Show that  $\text{SAT} \in \text{NP}$ .
  - ◆ Show that  $\text{CIRCUIT-SAT} \leq_P \text{SAT}$ .
- To show that  $\text{SAT} \in \text{NP}$ , we construct a verification algorithm  $A$  taking inputs  $x$  and  $y$ .
- It regards  $x$  as a boolean formula  $\phi$  and  $y$  as an assignment of values to variables of  $\phi$ .
- $A$  returns 1 if  $y$  defines a satisfying assignment for  $\phi$ ; otherwise,  $A$  returns 0.
- We can easily make  $A$  run in polynomial time.
- Thus,  $\text{SAT} \in \text{NP}$ .

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- Given a circuit  $C$ , we transform it into a boolean function  $\phi$  as follows.

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- Given a circuit  $C$ , we transform it into a boolean function  $\phi$  as follows.
- Associate a variable  $x_i$  with each wire of  $C$ ; let  $x_m$  be the output wire variable.



# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- Given a circuit  $C$ , we transform it into a boolean function  $\phi$  as follows.
- Associate a variable  $x_i$  with each wire of  $C$ ; let  $x_m$  be the output wire variable.
- Construct a sub-formula for each gate of  $C$  so that the output wire variables are a function of the input wire variables.

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

## ■ Example:

$$\phi_1 = (x_4 \leftrightarrow \neg x_3)$$

$$\phi_2 = (x_5 \leftrightarrow (x_1 \vee x_2))$$

$$\phi_3 = (x_6 \leftrightarrow \neg x_4)$$

$$\phi_4 = (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4))$$

$$\phi_5 = (x_8 \leftrightarrow (x_5 \vee x_6))$$

$$\phi_6 = (x_9 \leftrightarrow (x_6 \vee x_7))$$

$$\phi_7 = (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)).$$

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- If  $\phi_1, \dots, \phi_k$  are the sub-formulas, we define  $\phi$  to be  $x_m \wedge \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_k$ .

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- If  $\phi_1, \dots, \phi_k$  are the sub-formulas, we define  $\phi$  to be  $x_m \wedge \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_k$ .

- Example:

$$\begin{aligned} \phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)). \end{aligned}$$

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- If  $\phi_1, \dots, \phi_k$  are the sub-formulas, we define  $\phi$  to be  $x_m \wedge \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_k$ .

- Example:

$$\begin{aligned} \phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)). \end{aligned}$$

- $\phi$  can be constructed in polynomial time.

# Showing $\text{CIRCUIT-SAT} \leq_P \text{SAT}$

- If  $\phi_1, \dots, \phi_k$  are the sub-formulas, we define  $\phi$  to be  $x_m \wedge \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_k$ .

- Example:

$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)).\end{aligned}$$

- $\phi$  can be constructed in polynomial time.
- $C$  is satisfiable if and only if  $\phi$  is satisfiable:

$$\langle C \rangle \in \text{CIRCUIT-SAT} \Leftrightarrow \langle \phi \rangle \in \text{SAT}.$$

# 3-CNF formulas

- Let  $\phi$  be a boolean formula.

# 3-CNF formulas

- Let  $\phi$  be a boolean formula.
- A *literal* in  $\phi$  is an occurrence of a variable or its negation.



# 3-CNF formulas

- Let  $\phi$  be a boolean formula.
- A *literal* in  $\phi$  is an occurrence of a variable or its negation.
- Suppose  $\phi$  is the AND of sub-formulas, called *clauses*.

# 3-CNF formulas

- Let  $\phi$  be a boolean formula.
- A *literal* in  $\phi$  is an occurrence of a variable or its negation.
- Suppose  $\phi$  is the AND of sub-formulas, called *clauses*.
- Furthermore, suppose that each clause is the OR of exactly 3 literals.

# 3-CNF formulas

- Let  $\phi$  be a boolean formula.
- A *literal* in  $\phi$  is an occurrence of a variable or its negation.
- Suppose  $\phi$  is the AND of sub-formulas, called *clauses*.
- Furthermore, suppose that each clause is the OR of exactly 3 literals.
- Then we say that  $\phi$  is in *3-conjunctive normal form*, or *3-CNF*.

# 3-CNF formulas

- Let  $\phi$  be a boolean formula.
- A *literal* in  $\phi$  is an occurrence of a variable or its negation.
- Suppose  $\phi$  is the AND of sub-formulas, called *clauses*.
- Furthermore, suppose that each clause is the OR of exactly 3 literals.
- Then we say that  $\phi$  is in *3-conjunctive normal form*, or *3-CNF*.
- Example:

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

# NP-completeness of 3-CNF-SAT

- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle \phi \rangle \mid \phi \text{ is in 3-CNF and satisfiable}\}.$$

# NP-completeness of 3-CNF-SAT

- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle \phi \rangle \mid \phi \text{ is in 3-CNF and satisfiable}\}.$$

- We will show that  $3\text{-CNF-SAT} \in \text{NPC}$  in two steps:

# NP-completeness of 3-CNF-SAT

- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle \phi \rangle \mid \phi \text{ is in 3-CNF and satisfiable}\}.$$

- We will show that  $3\text{-CNF-SAT} \in \text{NPC}$  in two steps:
  - ◆  $3\text{-CNF-SAT} \in \text{NP}$ ,

# NP-completeness of 3-CNF-SAT

- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle \phi \rangle \mid \phi \text{ is in 3-CNF and satisfiable}\}.$$

- We will show that  $3\text{-CNF-SAT} \in \text{NPC}$  in two steps:
  - ◆  $3\text{-CNF-SAT} \in \text{NP}$ ,
  - ◆  $\text{SAT} \leq_P 3\text{-CNF-SAT}$ .



# NP-completeness of 3-CNF-SAT

- We define the language 3-CNF-SAT by

$$3\text{-CNF-SAT} = \{\langle \phi \rangle \mid \phi \text{ is in 3-CNF and satisfiable}\}.$$

- We will show that  $3\text{-CNF-SAT} \in \text{NPC}$  in two steps:
  - ◆  $3\text{-CNF-SAT} \in \text{NP}$ ,
  - ◆  $\text{SAT} \leq_P 3\text{-CNF-SAT}$ .
- Showing  $3\text{-CNF-SAT} \in \text{NP}$  is done using the same argument as for SAT.

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .
- We construct a *parse tree* for  $\phi$  where leaves are literals and internal nodes are connectives.

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .
- We construct a *parse tree* for  $\phi$  where leaves are literals and internal nodes are connectives.
- Regard the tree as a circuit with internal nodes as gates.

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .
- We construct a *parse tree* for  $\phi$  where leaves are literals and internal nodes are connectives.
- Regard the tree as a circuit with internal nodes as gates.
- We can construct formulas  $\phi'_1, \dots, \phi'_k$  for each of these gates, as before.

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .
- We construct a *parse tree* for  $\phi$  where leaves are literals and internal nodes are connectives.
- Regard the tree as a circuit with internal nodes as gates.
- We can construct formulas  $\phi'_1, \dots, \phi'_k$  for each of these gates, as before.
- Let  $\phi' = x_m \wedge \phi'_1 \wedge \phi'_2 \wedge \dots \wedge \phi'_k$ .

# NP-completeness of 3-CNF-SAT

- Remains to show  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .
- Let  $\langle \phi \rangle \in \text{SAT}$ .
- We construct a *parse tree* for  $\phi$  where leaves are literals and internal nodes are connectives.
- Regard the tree as a circuit with internal nodes as gates.
- We can construct formulas  $\phi'_1, \dots, \phi'_k$  for each of these gates, as before.
- Let  $\phi' = x_m \wedge \phi'_1 \wedge \phi'_2 \wedge \dots \wedge \phi'_k$ .
- $\phi'$  is the same formula as  $\phi$  but written in a different form where each clause  $\phi'_i$  has at most 3 literals.



# NP-completeness of 3-CNF-SAT

- Consider one clause  $\phi'_i$ .

# NP-completeness of 3-CNF-SAT

- Consider one clause  $\phi'_i$ .
- Example:  $\phi'_i = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$ .

# NP-completeness of 3-CNF-SAT

- Consider one clause  $\phi'_i$ .
- Example:  $\phi'_i = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$ .

$y_1$	$y_2$	$x_2$	$\phi'_i$	$\phi''_i = \neg\phi'_i$
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

# NP-completeness of 3-CNF-SAT

- Consider one clause  $\phi'_i$ .
- Example:  $\phi'_i = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$ .

$y_1$	$y_2$	$x_2$	$\phi'_i$	$\phi''_i = \neg\phi'_i$
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$\begin{aligned}\phi''_i = & (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \\ & \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).\end{aligned}$$

# NP-completeness of 3-CNF-SAT

- $\phi_i''$  is in *disjunctive normal form*:

$$\begin{aligned}\phi_i'' = & (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \\ & \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).\end{aligned}$$

# NP-completeness of 3-CNF-SAT

- $\phi_i''$  is in *disjunctive normal form*:

$$\begin{aligned}\phi_i'' = & (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \\ & \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).\end{aligned}$$

- Negating and applying DeMorgan's laws converts  $\phi_i''$  into 3-CNF:

$$\begin{aligned}\neg \phi_i'' = & (y_1 \vee \neg y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \\ & \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2).\end{aligned}$$

# NP-completeness of 3-CNF-SAT

- $\phi_i''$  is in *disjunctive normal form*:

$$\begin{aligned}\phi_i'' = & (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \\ & \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2).\end{aligned}$$

- Negating and applying DeMorgan's laws converts  $\phi_i''$  into 3-CNF:

$$\begin{aligned}\neg \phi_i'' = & (y_1 \vee \neg y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \\ & \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2).\end{aligned}$$

- This formula is equivalent to the original  $\phi_i'$ .

# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.



# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.
- Each clause has *at most* 3 literals.

# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.
- Each clause has *at most* 3 literals.
- We can easily extend this to *exactly* 3 literals.

# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.
- Each clause has *at most* 3 literals.
- We can easily extend this to *exactly* 3 literals.
- In polynomial time, we convert  $\phi'$  into a formula  $\phi'''$  in 3-CNF.

# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.
- Each clause has *at most* 3 literals.
- We can easily extend this to *exactly* 3 literals.
- In polynomial time, we convert  $\phi'$  into a formula  $\phi'''$  in 3-CNF.
- We have

$$\langle \phi \rangle \in \text{SAT} \Leftrightarrow \langle \phi''' \rangle \in \text{3-CNF-SAT}.$$

# NP-completeness of 3-CNF-SAT

- Applying this technique to each clause converts  $\phi'$  into a formula which is almost in 3-CNF.
- Each clause has *at most* 3 literals.
- We can easily extend this to *exactly* 3 literals.
- In polynomial time, we convert  $\phi'$  into a formula  $\phi'''$  in 3-CNF.
- We have

$$\langle \phi \rangle \in \text{SAT} \Leftrightarrow \langle \phi''' \rangle \in 3\text{-CNF-SAT}.$$

- Thus,  $\text{SAT} \leq_P 3\text{-CNF-SAT}$ .

# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .

# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .
- Is there a subset  $S'$  of  $S$  summing to  $t$ ?

# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .
- Is there a subset  $S'$  of  $S$  summing to  $t$ ?
- As a language:

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s \}.$$



# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .
- Is there a subset  $S'$  of  $S$  summing to  $t$ ?
- As a language:

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s \}.$$

- We will show that SUBSET-SUM is NP-complete.

# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .
- Is there a subset  $S'$  of  $S$  summing to  $t$ ?
- As a language:

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s \}.$$

- We will show that SUBSET-SUM is NP-complete.
- Clearly, SUBSET-SUM  $\in$  NP.

# The subset-sum problem

- Given a set  $S$  of positive integers and given integer target  $t > 0$ .
- Is there a subset  $S'$  of  $S$  summing to  $t$ ?
- As a language:

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid \exists S' \subseteq S \text{ so that } t = \sum_{s \in S'} s \}.$$

- We will show that SUBSET-SUM is NP-complete.
- Clearly, SUBSET-SUM  $\in$  NP.
- To show NP-hardness, we reduce from 3-CNF-SAT:

$$3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}.$$

# Showing $3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}$

- Consider a 3-CNF-formula  $\phi$  with  $n$  variables  $x_1, \dots, x_n$  and  $k$  clauses  $C_1, \dots, C_k$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}$

- Consider a 3-CNF-formula  $\phi$  with  $n$  variables  $x_1, \dots, x_n$  and  $k$  clauses  $C_1, \dots, C_k$ .
- We will construct an instance  $\langle S, t \rangle$  such that:

$$\phi \in 3\text{-CNF-SAT} \Leftrightarrow \langle S, t \rangle \in \text{SUBSET-SUM}.$$

# Showing $3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}$

- Consider a 3-CNF-formula  $\phi$  with  $n$  variables  $x_1, \dots, x_n$  and  $k$  clauses  $C_1, \dots, C_k$ .
- We will construct an instance  $\langle S, t \rangle$  such that:

$$\phi \in 3\text{-CNF-SAT} \Leftrightarrow \langle S, t \rangle \in \text{SUBSET-SUM}.$$

- In other words, we want that  $\phi$  is satisfiable if and only if  $S$  has a subset summing to  $t$ .

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .



# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .
- Finally, create a decimal number  $t$ .

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .
- Finally, create a decimal number  $t$ .
- Each number has  $n + k$  digits.

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .
- Finally, create a decimal number  $t$ .
- Each number has  $n + k$  digits.
- The  $n$  most significant digits are associated with  $x_1, \dots, x_n$ .

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .
- Finally, create a decimal number  $t$ .
- Each number has  $n + k$  digits.
- The  $n$  most significant digits are associated with  $x_1, \dots, x_n$ .
- The  $k$  least significant digits are associated with  $C_1, \dots, C_k$ .

# Constructing $S$ and $t$

- For each variable  $x_i$ , create two decimal numbers  $v_i$  and  $v'_i$ .
- For each clause  $C_j$ , create two decimal numbers  $s_j$  and  $s'_j$ .
- Finally, create a decimal number  $t$ .
- Each number has  $n + k$  digits.
- The  $n$  most significant digits are associated with  $x_1, \dots, x_n$ .
- The  $k$  least significant digits are associated with  $C_1, \dots, C_k$ .
- $S$  consists of numbers  $v_1, v'_1, v_2, v'_2, \dots, v_n, v'_n$  and  $s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k$ .

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .
- Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .



# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .
- Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .
- Digit  $C_i$  of  $s_i$  is 1 and digit  $C_i$  of  $s'_i$  is 2.

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .
- Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .
- Digit  $C_i$  of  $s_i$  is 1 and digit  $C_i$  of  $s'_i$  is 2.
- Target  $t$  is defined to be:

$$t = \overbrace{11 \dots 1}^n \overbrace{44 \dots 4}^k.$$

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .
- Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .
- Digit  $C_i$  of  $s_i$  is 1 and digit  $C_i$  of  $s'_i$  is 2.
- Target  $t$  is defined to be:

$$t = \overbrace{11 \dots 1}^n \overbrace{44 \dots 4}^k.$$

- All digits not specified above are 0.

# Constructing $S$ and $t$

- Digit  $x_i$  of  $v_i$  and digit  $x_i$  of  $v'_i$  is 1.
- Digit  $C_j$  of  $v_i$  is 1 if  $x_i \in C_j$ .
- Digit  $C_j$  of  $v'_i$  is 1 if  $\neg x_i \in C_j$ .
- Digit  $C_i$  of  $s_i$  is 1 and digit  $C_i$  of  $s'_i$  is 2.
- Target  $t$  is defined to be:

$$t = \overbrace{11 \dots 1}^n \overbrace{44 \dots 4}^k.$$

- All digits not specified above are 0.
- Simplifying assumptions: no clause contains both a variable and its negation and each variable appears somewhere. This ensures unique numbers.

Showing  $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .

# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.

# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.
- We will find subset  $S'$  of  $S$  with  $\sum_{s \in S'} s = t$ .

# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.
- We will find subset  $S'$  of  $S$  with  $\sum_{s \in S'} s = t$ .
- Assign values to  $x_1, \dots, x_n$  that satisfy  $\phi$ .



# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.
- We will find subset  $S'$  of  $S$  with  $\sum_{s \in S'} s = t$ .
- Assign values to  $x_1, \dots, x_n$  that satisfy  $\phi$ .
- For  $i = 1, \dots, n$ , if  $x_i = 1$  then include  $v_i$  in  $S'$ ; otherwise include  $v'_i$ .

# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.
- We will find subset  $S'$  of  $S$  with  $\sum_{s \in S'} s = t$ .
- Assign values to  $x_1, \dots, x_n$  that satisfy  $\phi$ .
- For  $i = 1, \dots, n$ , if  $x_i = 1$  then include  $v_i$  in  $S'$ ; otherwise include  $v'_i$ .
- Include additional numbers from  $\{s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k\}$  to reach target  $t$ .

# Showing $\phi \in 3\text{-CNF-SAT} \Rightarrow \langle S, t \rangle \in \text{SUBSET-SUM}$

- Assume  $\phi \in 3\text{-CNF-SAT}$ .
- In other words, assume that  $\phi$  is in 3-CNF and satisfiable.
- We will find subset  $S'$  of  $S$  with  $\sum_{s \in S'} s = t$ .
- Assign values to  $x_1, \dots, x_n$  that satisfy  $\phi$ .
- For  $i = 1, \dots, n$ , if  $x_i = 1$  then include  $v_i$  in  $S'$ ; otherwise include  $v'_i$ .
- Include additional numbers from  $\{s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k\}$  to reach target  $t$ .
- Why is this possible?

Showing  $\langle S, t \rangle \in \text{SUBSET-SUM} \Rightarrow \phi \in 3\text{-CNF-SAT}$

- Let  $S'$  be a subset of  $S$  summing to  $t$ .

Showing  $\langle S, t \rangle \in \text{SUBSET-SUM} \Rightarrow \phi \in \text{3-CNF-SAT}$

- Let  $S'$  be a subset of  $S$  summing to  $t$ .
- We need to find a satisfying assignment for  $\phi$ .

# Showing $\langle S, t \rangle \in \text{SUBSET-SUM} \Rightarrow \phi \in 3\text{-CNF-SAT}$

- Let  $S'$  be a subset of  $S$  summing to  $t$ .
- We need to find a satisfying assignment for  $\phi$ .
- We set  $x_i$  to 1 if and only if  $v_i \in S'$ .

# Showing $\langle S, t \rangle \in \text{SUBSET-SUM} \Rightarrow \phi \in \text{3-CNF-SAT}$

- Let  $S'$  be a subset of  $S$  summing to  $t$ .
- We need to find a satisfying assignment for  $\phi$ .
- We set  $x_i$  to 1 if and only if  $v_i \in S'$ .
- Why is this a satisfying assignment?

# The CLIQUE-problem

- Let  $G = (V, E)$  be an undirected graph.



# The CLIQUE-problem

- Let  $G = (V, E)$  be an undirected graph.
- A *clique* in  $G$  is a subset  $V' \subseteq V$  such that  $(u, v) \in E$  for all distinct  $u, v \in V'$ .

# The CLIQUE-problem

- Let  $G = (V, E)$  be an undirected graph.
- A *clique* in  $G$  is a subset  $V' \subseteq V$  such that  $(u, v) \in E$  for all distinct  $u, v \in V'$ .
- The size of the clique is  $|V'|$ .

# The CLIQUE-problem

- Let  $G = (V, E)$  be an undirected graph.
- A *clique* in  $G$  is a subset  $V' \subseteq V$  such that  $(u, v) \in E$  for all distinct  $u, v \in V'$ .
- The size of the clique is  $|V'|$ .
- The CLIQUE problem is the problem of determining if  $G$  contains a clique of a given size  $k$ :

$$\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is a graph containing a clique of size } k \}.$$

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .
- To show  $\text{CLIQUE} \in \text{NP}$ , consider an algorithm  $A$  taking two inputs,  $\langle G, k \rangle$  and a certificate  $y$ .

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .
- To show  $\text{CLIQUE} \in \text{NP}$ , consider an algorithm  $A$  taking two inputs,  $\langle G, k \rangle$  and a certificate  $y$ .
- $y$  specifies a subset  $V'$  of vertices of  $G$ .



# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .
- To show  $\text{CLIQUE} \in \text{NP}$ , consider an algorithm  $A$  taking two inputs,  $\langle G, k \rangle$  and a certificate  $y$ .
- $y$  specifies a subset  $V'$  of vertices of  $G$ .
- $A$  checks that  $|V'| = k$  and that  $V'$  is a clique in  $G$ .

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .
- To show  $\text{CLIQUE} \in \text{NP}$ , consider an algorithm  $A$  taking two inputs,  $\langle G, k \rangle$  and a certificate  $y$ .
- $y$  specifies a subset  $V'$  of vertices of  $G$ .
- $A$  checks that  $|V'| = k$  and that  $V'$  is a clique in  $G$ .
- This can easily be done in polynomial time.

# NP-completeness of CLIQUE

- We will show that CLIQUE is NP-complete as follows:
  - ◆  $\text{CLIQUE} \in \text{NP}$ ,
  - ◆  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .
- To show  $\text{CLIQUE} \in \text{NP}$ , consider an algorithm  $A$  taking two inputs,  $\langle G, k \rangle$  and a certificate  $y$ .
- $y$  specifies a subset  $V'$  of vertices of  $G$ .
- $A$  checks that  $|V'| = k$  and that  $V'$  is a clique in  $G$ .
- This can easily be done in polynomial time.
- Thus,  $\text{CLIQUE} \in \text{NP}$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.
- We will construct a graph  $G$  such that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.
- We will construct a graph  $G$  such that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- For each  $C_r = \ell_1^r \vee \ell_2^r \vee \ell_3^r$ , we include three vertices  $v_1^r$ ,  $v_2^r$ , and  $v_3^r$  to  $G$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.
- We will construct a graph  $G$  such that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- For each  $C_r = \ell_1^r \vee \ell_2^r \vee \ell_3^r$ , we include three vertices  $v_1^r$ ,  $v_2^r$ , and  $v_3^r$  to  $G$ .
- There is an edge  $(v_i^r, v_j^s)$  in  $G$  if and only if  $r \neq s$  and  $\ell_i^r$  is not the negation of  $\ell_j^s$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.
- We will construct a graph  $G$  such that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- For each  $C_r = \ell_1^r \vee \ell_2^r \vee \ell_3^r$ , we include three vertices  $v_1^r$ ,  $v_2^r$ , and  $v_3^r$  to  $G$ .
- There is an edge  $(v_i^r, v_j^s)$  in  $G$  if and only if  $r \neq s$  and  $\ell_i^r$  is not the negation of  $\ell_j^s$ .
- $G$  can be constructed in polynomial time from  $\phi$ .



# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Given a formula  $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  in 3-CNF.
- We will construct a graph  $G$  such that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- For each  $C_r = \ell_1^r \vee \ell_2^r \vee \ell_3^r$ , we include three vertices  $v_1^r$ ,  $v_2^r$ , and  $v_3^r$  to  $G$ .
- There is an edge  $(v_i^r, v_j^s)$  in  $G$  if and only if  $r \neq s$  and  $\ell_i^r$  is not the negation of  $\ell_j^s$ .
- $G$  can be constructed in polynomial time from  $\phi$ .
- We will show:

$$\langle \phi \rangle \in 3\text{-CNF-SAT} \Leftrightarrow \langle G, k \rangle \in \text{CLIQUE}.$$

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .
- In a satisfying assignment for  $\phi$ , each clause  $C_i$  of  $\phi$  has at least one true literal.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .
- In a satisfying assignment for  $\phi$ , each clause  $C_i$  of  $\phi$  has at least one true literal.
- Pick the corresponding vertex in  $G$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .
- In a satisfying assignment for  $\phi$ , each clause  $C_i$  of  $\phi$  has at least one true literal.
- Pick the corresponding vertex in  $G$ .
- This gives a total of  $k$  vertices.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .
- In a satisfying assignment for  $\phi$ , each clause  $C_i$  of  $\phi$  has at least one true literal.
- Pick the corresponding vertex in  $G$ .
- This gives a total of  $k$  vertices.
- There must be an edge between each pair of these vertices since no picked literal can be the negation of another picked literal.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Need to show that  $\phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- Assume first that  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .
- In a satisfying assignment for  $\phi$ , each clause  $C_i$  of  $\phi$  has at least one true literal.
- Pick the corresponding vertex in  $G$ .
- This gives a total of  $k$  vertices.
- There must be an edge between each pair of these vertices since no picked literal can be the negation of another picked literal.
- Hence,  $G$  has a clique of size  $k$  so  $\langle G, k \rangle \in \text{CLIQUE}$ .



# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .
- Each vertex triple of  $G$  has exactly one vertex in  $V'$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .
- Each vertex triple of  $G$  has exactly one vertex in  $V'$ .
- We assign 1 to the literal of  $\phi$  corresponding to that vertex.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .
- Each vertex triple of  $G$  has exactly one vertex in  $V'$ .
- We assign 1 to the literal of  $\phi$  corresponding to that vertex.
- No variable of  $\phi$  is assigned both 0 and 1 in this way since there is no edge between a variable and its negation in  $G$ .

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .
- Each vertex triple of  $G$  has exactly one vertex in  $V'$ .
- We assign 1 to the literal of  $\phi$  corresponding to that vertex.
- No variable of  $\phi$  is assigned both 0 and 1 in this way since there is no edge between a variable and its negation in  $G$ .
- This gives a legal assignment of values to variables.

# Showing $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$

- Remains to show that if  $G$  has a clique of size  $k$  then  $\phi$  is satisfiable.
- Let  $V'$  be a clique of  $G$  of size  $k$ .
- Each vertex triple of  $G$  has exactly one vertex in  $V'$ .
- We assign 1 to the literal of  $\phi$  corresponding to that vertex.
- No variable of  $\phi$  is assigned both 0 and 1 in this way since there is no edge between a variable and its negation in  $G$ .
- This gives a legal assignment of values to variables.
- This assignment satisfies  $\phi$  as it makes each clause true.

# The VERTEX-COVER problem

- A *vertex cover* of  $G = (V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .



# The VERTEX-COVER problem

- A *vertex cover* of  $G = (V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .
- Vertex cover problem: find a minimum-size vertex cover of  $G$ .

# The VERTEX-COVER problem

- A *vertex cover* of  $G = (V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .
- Vertex cover problem: find a minimum-size vertex cover of  $G$ .
- Restating as a decision problem: does  $G$  have a vertex cover of a given size  $k$ ?

# The VERTEX-COVER problem

- A *vertex cover* of  $G = (V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .
- Vertex cover problem: find a minimum-size vertex cover of  $G$ .
- Restating as a decision problem: does  $G$  have a vertex cover of a given size  $k$ ?
- Stated as a language:

$$\text{VERTEX-COVER} = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } k \}.$$

# The VERTEX-COVER problem

- A *vertex cover* of  $G = (V, E)$  is a subset  $V' \subseteq V$  such that every edge of  $E$  has at least one endpoint in  $V'$ .
- Vertex cover problem: find a minimum-size vertex cover of  $G$ .
- Restating as a decision problem: does  $G$  have a vertex cover of a given size  $k$ ?
- Stated as a language:

$$\text{VERTEX-COVER} = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } k \}.$$

- We will show that VERTEX-COVER is NP-complete.

# Showing that VERTEX-COVER $\in$ NP

- Verification algorithm takes an instance  $\langle G, k \rangle$  and a certificate denoting a subset  $V'$  of vertices of  $G$ .

# Showing that VERTEX-COVER $\in$ NP

- Verification algorithm takes an instance  $\langle G, k \rangle$  and a certificate denoting a subset  $V'$  of vertices of  $G$ .
- It checks that  $V'$  has size  $k$  and that every edge of  $G$  is incident to at least one vertex of  $V'$ .

# Showing that VERTEX-COVER $\in$ NP

- Verification algorithm takes an instance  $\langle G, k \rangle$  and a certificate denoting a subset  $V'$  of vertices of  $G$ .
- It checks that  $V'$  has size  $k$  and that every edge of  $G$  is incident to at least one vertex of  $V'$ .
- Can easily be done in polynomial time.

# Showing that VERTEX-COVER $\in$ NP

- Verification algorithm takes an instance  $\langle G, k \rangle$  and a certificate denoting a subset  $V'$  of vertices of  $G$ .
- It checks that  $V'$  has size  $k$  and that every edge of  $G$  is incident to at least one vertex of  $V'$ .
- Can easily be done in polynomial time.
- Hence, VERTEX-COVER  $\in$  NP.



# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .

# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .
- Given instance  $\langle G, k \rangle$  of the clique problem.

# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .
- Given instance  $\langle G, k \rangle$  of the clique problem.
- Let  $n$  denote the number of vertices of  $G$ .

# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .
- Given instance  $\langle G, k \rangle$  of the clique problem.
- Let  $n$  denote the number of vertices of  $G$ .
- We transform it in polynomial time to the instance  $\langle \overline{G}, n - k \rangle$  of the vertex cover problem.

# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .
- Given instance  $\langle G, k \rangle$  of the clique problem.
- Let  $n$  denote the number of vertices of  $G$ .
- We transform it in polynomial time to the instance  $\langle \overline{G}, n - k \rangle$  of the vertex cover problem.
- Here,  $\overline{G}$  is the *complement* of  $G$  which has an edge between two vertices  $u$  and  $v$  if and only if there is no edge between  $u$  and  $v$  in  $G$ .

# Showing that VERTEX-COVER is NP-hard

- We show  $\text{CLIQUE} \leq_P \text{VERTEX-COVER}$ .
- Given instance  $\langle G, k \rangle$  of the clique problem.
- Let  $n$  denote the number of vertices of  $G$ .
- We transform it in polynomial time to the instance  $\langle \overline{G}, n - k \rangle$  of the vertex cover problem.
- Here,  $\overline{G}$  is the *complement* of  $G$  which has an edge between two vertices  $u$  and  $v$  if and only if there is no edge between  $u$  and  $v$  in  $G$ .
- Need to show:

$$\langle G, k \rangle \in \text{CLIQUE} \Leftrightarrow \langle \overline{G}, n - k \rangle \in \text{VERTEX-COVER}.$$

# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .

# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .
- A *tour* of  $G$  is a Hamilton cycle of  $G$ .



# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .
- A *tour* of  $G$  is a Hamilton cycle of  $G$ .
- The travelling salesman problem is that of finding a minimum-cost tour of  $G$ .

# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .
- A *tour* of  $G$  is a Hamilton cycle of  $G$ .
- The travelling salesman problem is that of finding a minimum-cost tour of  $G$ .
- Stated as a language:

$$\text{TSP} = \{ \langle G, c, k \rangle \mid G \text{ has a tour of cost at most } k \}.$$

# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .
- A *tour* of  $G$  is a Hamilton cycle of  $G$ .
- The travelling salesman problem is that of finding a minimum-cost tour of  $G$ .
- Stated as a language:

$$\text{TSP} = \{ \langle G, c, k \rangle \mid G \text{ has a tour of cost at most } k \}.$$

- Clearly,  $\text{TSP} \in \text{NP}$ .

# The travelling-salesman problem

- Let  $G = (V, E)$  be a complete graph and let  $c$  be a non-negative integer cost function on the edge set of  $G$ .
- A *tour* of  $G$  is a Hamilton cycle of  $G$ .
- The travelling salesman problem is that of finding a minimum-cost tour of  $G$ .
- Stated as a language:

$$\text{TSP} = \{ \langle G, c, k \rangle \mid G \text{ has a tour of cost at most } k \}.$$

- Clearly,  $\text{TSP} \in \text{NP}$ .
- We show that  $\text{TSP}$  is NP-complete by:

$$\text{HAM-CYCLE} \leq_P \text{TSP}.$$

- Let  $G = (V, E)$  be an instance of the Hamilton-cycle problem.

- Let  $G = (V, E)$  be an instance of the Hamilton-cycle problem.
- We construct a complete graph  $G' = (V, E')$  on vertex set  $V$ .

- Let  $G = (V, E)$  be an instance of the Hamilton-cycle problem.
- We construct a complete graph  $G' = (V, E')$  on vertex set  $V$ .
- Define a cost function  $c$  on  $E'$  by

$$c(u, v) = \begin{cases} 0 & \text{if } (u, v) \in E, \\ 1 & \text{if } (u, v) \notin E. \end{cases}$$

- Let  $G = (V, E)$  be an instance of the Hamilton-cycle problem.
- We construct a complete graph  $G' = (V, E')$  on vertex set  $V$ .
- Define a cost function  $c$  on  $E'$  by

$$c(u, v) = \begin{cases} 0 & \text{if } (u, v) \in E, \\ 1 & \text{if } (u, v) \notin E. \end{cases}$$

- Show that:

$$\langle G \rangle \in \text{HAM-CYCLE} \Leftrightarrow \langle G', c, 0 \rangle \in \text{TSP}.$$