

# COMPLEXITY THEORY

## Lecture 7: NP-Completeness

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 5th Nov 2024

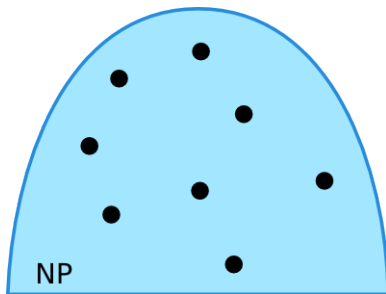
More recent versions of this slide deck might be available.  
For the most current version of this course, see  
[https://iccl.inf.tu-dresden.de/web/Complexity\\_Theory/en](https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en)

# Review

# Are NP Problems Hard?

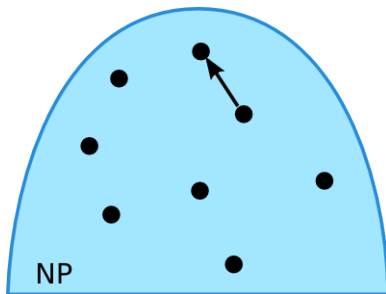
# The Structure of NP

Idea: polynomial many-one reductions define an order on problems



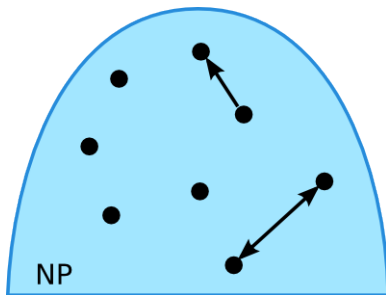
# The Structure of NP

Idea: polynomial many-one reductions define an order on problems



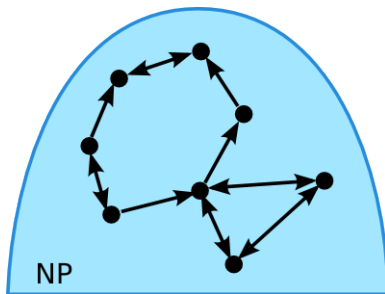
# The Structure of NP

Idea: polynomial many-one reductions define an order on problems



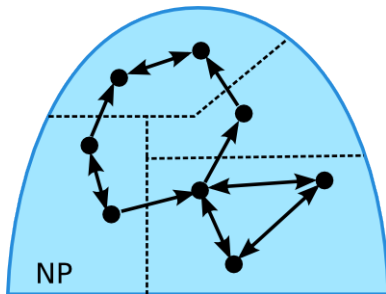
# The Structure of NP

Idea: polynomial many-one reductions define an order on problems



# The Structure of NP

Idea: polynomial many-one reductions define an order on problems





# NP-Hardness and NP-Completeness

## Definition 7.1:

- (1) A language **H** is **NP-hard**, if  $L \leq_p H$  for every language  $L \in NP$ .
- (2) A language **C** is **NP-complete**, if **C** is NP-hard and  $C \in NP$ .

## NP-Completeness

- NP-complete problems are the **hardest** problems in NP.
- They constitute the maximal class (wrt.  $\leq_p$ ) of problems within NP.
- They are all **equally** difficult – an efficient solution to one would solve them all.

**Theorem 7.2:** If **L** is NP-hard and  $L \leq_p L'$ , then **L'** is NP-hard as well.

# Proving NP-Completeness

# Proving NP-Completeness

## How to show NP-completeness

To show that **L** is NP-complete, we must show that every language in NP can be reduced to **L** in polynomial time.

## Alternative approach

Given an NP-complete language **C**, we can show that another language **L** is NP-complete just by showing that

- $\mathbf{C} \leq_p \mathbf{L}$
- $\mathbf{L} \in \text{NP}$

# Proving NP-Completeness

## How to show NP-completeness

To show that **L** is NP-complete, we must show that every language in NP can be reduced to **L** in polynomial time.

## Alternative approach

Given an NP-complete language **C**, we can show that another language **L** is NP-complete just by showing that

- $\mathbf{C} \leq_p \mathbf{L}$
- $\mathbf{L} \in \text{NP}$

However: Is there any NP-complete problem at all?

# Proving NP-Completeness

## How to show NP-completeness

To show that **L** is NP-complete, we must show that *every* language in NP can be reduced to **L** in polynomial time.

## Alternative approach

Given an NP-complete language **C**, we can show that another language **L** is NP-complete just by showing that

- $\mathbf{C} \leq_p \mathbf{L}$
- $\mathbf{L} \in \text{NP}$

However: Is there any NP-complete problem at all?

Yes, thousands of them!

# The Cook-Levin Theorem

# The Cook-Levin Theorem

**Theorem 7.3 (Cook 1970, Levin 1973):** SAT is NP-complete.

# The Cook-Levin Theorem

**Theorem 7.3 (Cook 1970, Levin 1973):** SAT is NP-complete.

## Proof:

(1) SAT  $\in$  NP

Take satisfying assignments as polynomial certificates for the satisfiability of a formula.



# The Cook-Levin Theorem

**Theorem 7.3 (Cook 1970, Levin 1973):** SAT is NP-complete.

## Proof:

(1) SAT  $\in$  NP

Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

(2) SAT is hard for NP

Proof by reduction from any word problem of some polynomially time-bounded NTM.

□

# Proving the Cook-Levin Theorem: Main Objective

## Given:

- a polynomial  $p$
- a  $p$ -time bounded 1-tape NTM  $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word  $w$

**Intended reduction:** Define a propositional logic formula  $\varphi_{p, \mathcal{M}, w}$  such that

- (1)  $\varphi_{p, \mathcal{M}, w}$  is satisfiable if and only if  $\mathcal{M}$  accepts  $w$  in time  $p(|w|)$
- (2)  $\varphi_{p, \mathcal{M}, w}$  is polynomial with respect to  $|w|$

# Proving the Cook-Levin Theorem: Rationale

**Given:** polynomial  $p$ , NTM  $\mathcal{M}$ , word  $w$

**Intended reduction:** Define a propositional logic formula  $\varphi_{p,\mathcal{M},w}$  such that

- (1)  $\varphi_{p,\mathcal{M},w}$  is satisfiable if and only if  $\mathcal{M}$  accepts  $w$  in time  $p(|w|)$
- (2)  $\varphi_{p,\mathcal{M},w}$  is polynomial with respect to  $|w|$

Why does this prove NP-hardness of **SAT**?

# Proving the Cook-Levin Theorem: Rationale

**Given:** polynomial  $p$ , NTM  $\mathcal{M}$ , word  $w$

**Intended reduction:** Define a propositional logic formula  $\varphi_{p,\mathcal{M},w}$  such that

- (1)  $\varphi_{p,\mathcal{M},w}$  is satisfiable if and only if  $\mathcal{M}$  accepts  $w$  in time  $p(|w|)$
- (2)  $\varphi_{p,\mathcal{M},w}$  is polynomial with respect to  $|w|$

**Why does this prove NP-hardness of SAT?**

Because it leads to a reduction  $\mathbf{L} \leq_p \mathbf{SAT}$  for every language  $\mathbf{L} \in \mathbf{NP}$ :

- If  $\mathbf{L} \in \mathbf{NP}$ , then there is an NTM  $\mathcal{M}$  that is time-bounded by some polynomial  $p$ , such that  $\mathbf{L}(\mathcal{M}) = \mathbf{L}$ .
- The function  $f_{\mathcal{M},p} : w \mapsto \varphi_{p,\mathcal{M},w}$  shows  $\mathbf{L} \leq_p \mathbf{SAT}$ :
  - $f$  is a many-one reduction due to item (1) above
  - $f$  is polynomial due to item (2) above

**Note:** We do not claim the transformation  $\langle p, \mathcal{M}, w \rangle \mapsto \varphi_{p,\mathcal{M},w}$  to be polynomial in the size of  $p$ ,  $\mathcal{M}$ , and  $w$ . Indeed, this would not hold true under reasonable encodings of  $p$ . But being (multi-)exponential in  $p$  is not a concern since the many-one reductions  $f_{\mathcal{M},p}$  each use a fixed  $p$  and only care about the asymptotic complexity as  $w$  grows.

# Proving Cook-Levin: Encoding Configurations

**Idea:** Use logic to describe a run of  $\mathcal{M}$  on input  $w$  by a formula.

**Note:** On input  $w$  of length  $n := |w|$ , every computation path of  $\mathcal{M}$  is of length  $\leq p(n)$  and uses  $\leq p(n)$  tape cells.

## Use propositional variables for describing configurations:

$Q_q$  for each  $q \in Q$  means “ $\mathcal{M}$  is in state  $q \in Q$ ”

$P_i$  for each  $0 \leq i < p(n)$  means “the head is at Position  $i$ ”

$S_{a,i}$  for each  $a \in \Gamma$  and  $0 \leq i < p(n)$  means “tape cell  $i$  contains Symbol  $a$ ”

# Proving Cook-Levin: Encoding Configurations

**Idea:** Use logic to describe a run of  $\mathcal{M}$  on input  $w$  by a formula.

**Note:** On input  $w$  of length  $n := |w|$ , every computation path of  $\mathcal{M}$  is of length  $\leq p(n)$  and uses  $\leq p(n)$  tape cells.

## Use propositional variables for describing configurations:

$Q_q$  for each  $q \in Q$  means “ $\mathcal{M}$  is in state  $q \in Q$ ”

$P_i$  for each  $0 \leq i < p(n)$  means “the head is at Position  $i$ ”

$S_{a,i}$  for each  $a \in \Gamma$  and  $0 \leq i < p(n)$  means “tape cell  $i$  contains Symbol  $a$ ”

**Represent configuration**  $(q, hp, a_0 \dots a_{p(n)})$  by truth assignments to variables from the set

$$\bar{C} := \{Q_q, P_i, S_{a,i} \mid q \in Q, a \in \Gamma, 0 \leq i < p(n)\}$$

using the truth assignment  $\beta$  defined as

$$\beta(Q_s) := \begin{cases} 1 & s = q \\ 0 & s \neq q \end{cases}$$

$$\beta(P_i) := \begin{cases} 1 & i = hp \\ 0 & i \neq hp \end{cases}$$

$$\beta(S_{a,i}) := \begin{cases} 1 & a = a_i \\ 0 & a \neq a_i \end{cases}$$

# Proving Cook-Levin: Validating Configurations

We define a formula  $\text{Conf}(\bar{C})$  for a set of configuration variables

$$\bar{C} = \{Q_q, P_i, S_{a,i} \mid q \in Q, \quad a \in \Gamma, \quad 0 \leq i < p(n)\}$$

as follows:

$\text{Conf}(\bar{C}) :=$

“the assignment is a valid configuration”:

$$\bigvee_{q \in Q} (Q_q \wedge \bigwedge_{q' \neq q} \neg Q_{q'})$$

“TM in exactly one state  $q \in Q$ ”

$$\wedge \bigvee_{hp < p(n)} (P_{hp} \wedge \bigwedge_{hp' \neq hp} \neg P_{hp'})$$

“head in exactly one position  $hp \leq p(n)$ ”

$$\wedge \bigwedge_{0 \leq i < p(n)} \bigvee_{a \in \Gamma} (S_{a,i} \wedge \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i})$$

“exactly one  $a \in \Gamma$  in each cell”

# Proving Cook-Levin: Validating Configurations

For an assignment  $\beta$  defined on variables in  $\overline{C}$  define

$$\text{conf}(\overline{C}, \beta) := \left\{ (q, hp, w_0 \dots w_{p(n)}) \mid \begin{array}{l} \beta(Q_q) = 1, \\ \beta(P_{hp}) = 1, \\ \beta(S_{w_i, i}) = 1 \text{ for all } 0 \leq i < p(n) \end{array} \right\}$$

**Note:**  $\beta$  may be defined on other variables besides those in  $\overline{C}$ .



# Proving Cook-Levin: Validating Configurations

For an assignment  $\beta$  defined on variables in  $\bar{C}$  define

$$\text{conf}(\bar{C}, \beta) := \left\{ (q, hp, w_0 \dots w_{p(n)}) \mid \begin{array}{l} \beta(Q_q) = 1, \\ \beta(P_{hp}) = 1, \\ \beta(S_{w_i, i}) = 1 \text{ for all } 0 \leq i < p(n) \end{array} \right\}$$

**Note:**  $\beta$  may be defined on other variables besides those in  $\bar{C}$ .

**Lemma 7.4:** If  $\beta$  satisfies  $\text{Conf}(\bar{C})$  then  $|\text{conf}(\bar{C}, \beta)| = 1$ .

We can therefore write  $\text{conf}(\bar{C}, \beta) = (q, hp, w)$  to simplify notation.

# Proving Cook-Levin: Validating Configurations

For an assignment  $\beta$  defined on variables in  $\bar{C}$  define

$$\text{conf}(\bar{C}, \beta) := \left\{ \begin{array}{l} \beta(Q_q) = 1, \\ (q, hp, w_0 \dots w_{p(n)}) \mid \beta(P_{hp}) = 1, \\ \beta(S_{w_i, i}) = 1 \text{ for all } 0 \leq i < p(n) \end{array} \right\}$$

**Note:**  $\beta$  may be defined on other variables besides those in  $\bar{C}$ .

**Lemma 7.4:** If  $\beta$  satisfies  $\text{Conf}(\bar{C})$  then  $|\text{conf}(\bar{C}, \beta)| = 1$ .

We can therefore write  $\text{conf}(\bar{C}, \beta) = (q, hp, w)$  to simplify notation.

## Observations:

- $\text{conf}(\bar{C}, \beta)$  is a potential configuration of  $\mathcal{M}$ , but it may not be reachable from the start configuration of  $\mathcal{M}$  on input  $w$ .
- Conversely, every configuration  $(q, hp, w_1 \dots w_{p(n)})$  induces a satisfying assignment  $\beta$  or which  $\text{conf}(\bar{C}, \beta) = (q, hp, w_1 \dots w_{p(n)})$ .

# Proving Cook-Levin: Transitions Between Configurations

Consider the following formula  $\text{Next}(\bar{C}, \bar{C}')$  defined as

$$\text{Conf}(\bar{C}) \wedge \text{Conf}(\bar{C}') \wedge \text{NoChange}(\bar{C}, \bar{C}') \wedge \text{Change}(\bar{C}, \bar{C}').$$

$$\text{NoChange} := \bigvee_{0 \leq hp < p(n)} \left( P_{hp} \wedge \bigwedge_{i \neq hp, a \in \Gamma} (S_{a,i} \rightarrow S'_{a,i}) \right)$$

$$\text{Change} := \bigvee_{0 \leq hp < p(n)} \left( P_{hp} \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} (Q_q \wedge S_{a, hp} \wedge \bigvee_{(q', b, D) \in \delta(q, a)} (Q_{q'} \wedge S'_{b, hp} \wedge P'_{D(hp)})) \right)$$

where  $D(hp)$  is the position reached by moving in direction  $D$  from  $hp$ .

# Proving Cook-Levin: Transitions Between Configurations

Consider the following formula  $\text{Next}(\bar{C}, \bar{C}')$  defined as

$$\text{Conf}(\bar{C}) \wedge \text{Conf}(\bar{C}') \wedge \text{NoChange}(\bar{C}, \bar{C}') \wedge \text{Change}(\bar{C}, \bar{C}').$$

$$\text{NoChange} := \bigvee_{0 \leq hp < p(n)} \left( P_{hp} \wedge \bigwedge_{i \neq hp, a \in \Gamma} (S_{a,i} \rightarrow S'_{a,i}) \right)$$

$$\text{Change} := \bigvee_{0 \leq hp < p(n)} \left( P_{hp} \wedge \bigvee_{\substack{q \in Q \\ a \in \Gamma}} (Q_q \wedge S_{a, hp} \wedge \bigvee_{(q', b, D) \in \delta(q, a)} (Q'_{q'} \wedge S'_{b, hp} \wedge P'_{D(hp)})) \right)$$

where  $D(hp)$  is the position reached by moving in direction  $D$  from  $hp$ .

**Lemma 7.5:** For any assignment  $\beta$  defined on  $\bar{C} \cup \bar{C}'$ :

$\beta$  satisfies  $\text{Next}(\bar{C}, \bar{C}')$  if and only if  $\text{conf}(\bar{C}, \beta) \vdash_M \text{conf}(\bar{C}', \beta)$

# Proving Cook-Levin: Start and End

## Defined so far:

- $\text{Conf}(\bar{C})$ :  $\bar{C}$  describes a potential configuration
- $\text{Next}(\bar{C}, \bar{C}')$ :  $\text{conf}(\bar{C}, \beta) \vdash_{\mathcal{M}} \text{conf}(\bar{C}', \beta)$

# Proving Cook-Levin: Start and End

## Defined so far:

- $\text{Conf}(\bar{C})$ :  $\bar{C}$  describes a potential configuration
- $\text{Next}(\bar{C}, \bar{C}')$ :  $\text{conf}(\bar{C}, \beta) \vdash_{\mathcal{M}} \text{conf}(\bar{C}', \beta)$

**Start configuration:** For an input word  $w = w_0 \cdots w_{n-1} \in \Sigma^*$ , we define:

$$\text{Start}_{\mathcal{M}, w}(\bar{C}) := \text{Conf}(\bar{C}) \wedge Q_{q_0} \wedge P_0 \wedge \bigwedge_{i=0}^{n-1} S_{w_i, i} \wedge \bigwedge_{i=n}^{p(n)-1} S_{\perp, i}$$

Then an assignment  $\beta$  satisfies  $\text{Start}_{\mathcal{M}, w}(\bar{C})$  if and only if  $\bar{C}$  represents the start configuration of  $\mathcal{M}$  on input  $w$ .

# Proving Cook-Levin: Start and End

## Defined so far:

- $\text{Conf}(\bar{C})$ :  $\bar{C}$  describes a potential configuration
- $\text{Next}(\bar{C}, \bar{C}')$ :  $\text{conf}(\bar{C}, \beta) \vdash_{\mathcal{M}} \text{conf}(\bar{C}', \beta)$

**Start configuration:** For an input word  $w = w_0 \cdots w_{n-1} \in \Sigma^*$ , we define:

$$\text{Start}_{\mathcal{M}, w}(\bar{C}) := \text{Conf}(\bar{C}) \wedge Q_{q_0} \wedge P_0 \wedge \bigwedge_{i=0}^{n-1} S_{w_i, i} \wedge \bigwedge_{i=n}^{p(n)-1} S_{\perp, i}$$

Then an assignment  $\beta$  satisfies  $\text{Start}_{\mathcal{M}, w}(\bar{C})$  if and only if  $\bar{C}$  represents the start configuration of  $\mathcal{M}$  on input  $w$ .

## Accepting stop configuration:

$$\text{Acc-Conf}(\bar{C}) := \text{Conf}(\bar{C}) \wedge Q_{q_{\text{accept}}}$$

Then an assignment  $\beta$  satisfies  $\text{Acc-Conf}(\bar{C})$  if and only if  $\bar{C}$  represents an accepting configuration of  $\mathcal{M}$ .

# Proving Cook-Levin: Adding Time

Since  $\mathcal{M}$  is  $p$ -time bounded, each run may contain up to  $p(n)$  steps

→ we need one set of configuration variables for each

## Propositional variables:

$Q_{q,t}$  for all  $q \in Q$ ,  $0 \leq t \leq p(n)$  means “at time  $t$ ,  $\mathcal{M}$  is in state  $q \in Q$ ”

$P_{i,t}$  for all  $0 \leq i, t \leq p(n)$  means “at time  $t$ , the head is at position  $i$ ”

$S_{a,i,t}$  for all  $a \in \Gamma$  and  $0 \leq i, t \leq p(n)$  means “at time  $t$ , tape cell  $i$  contains symbol  $a$ ”

## Notation:

$$\bar{C}_t := \{Q_{q,t}, P_{i,t}, S_{a,i,t} \mid q \in Q, 0 \leq i \leq p(n), a \in \Gamma\}$$



# Proving Cook-Levin: The Formula

## Given:

- a polynomial  $p$
- a  $p$ -time bounded 1-tape NTM  $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word  $w$

We define the formula  $\varphi_{p, \mathcal{M}, w}$  as follows:

$$\varphi_{p, \mathcal{M}, w} := \text{Start}_{\mathcal{M}, w}(\bar{C}_0) \wedge \bigvee_{0 \leq t \leq p(n)} \left( \text{Acc-Conf}(\bar{C}_t) \wedge \bigwedge_{0 \leq i < t} \text{Next}(\bar{C}_i, \bar{C}_{i+1}) \right)$$

“ $\bar{C}_0$  encodes the start configuration” and, for some polynomial time  $t$ :

“ $\mathcal{M}$  accepts after  $t$  steps” and “ $\bar{C}_0, \dots, \bar{C}_t$  encode a computation path”

# Proving Cook-Levin: The Formula

## Given:

- a polynomial  $p$
- a  $p$ -time bounded 1-tape NTM  $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word  $w$

We define the formula  $\varphi_{p, \mathcal{M}, w}$  as follows:

$$\varphi_{p, \mathcal{M}, w} := \text{Start}_{\mathcal{M}, w}(\bar{C}_0) \wedge \bigvee_{0 \leq t \leq p(n)} \left( \text{Acc-Conf}(\bar{C}_t) \wedge \bigwedge_{0 \leq i < t} \text{Next}(\bar{C}_i, \bar{C}_{i+1}) \right)$$

“ $C_0$  encodes the start configuration” and, for some polynomial time  $t$ :

“ $\mathcal{M}$  accepts after  $t$  steps” and “ $\bar{C}_0, \dots, \bar{C}_t$  encode a computation path”

**Lemma 7.6:**  $\varphi_{p, \mathcal{M}, w}$  is satisfiable if and only if  $\mathcal{M}$  accepts  $w$  in time  $p(|w|)$ .

Note that an accepting or rejecting stop configuration has no successor.

**Lemma 7.7:** The size of  $\varphi_{p, \mathcal{M}, w}$  is polynomial in  $|w|$ .

# The Cook-Levin Theorem

**Theorem 7.3 (Cook 1970, Levin 1973):** SAT is NP-complete.

## Proof:

(1) SAT  $\in$  NP

Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

(2) SAT is hard for NP

Proof by reduction from any word problem of some polynomially time-bounded NTM.

□

# Further NP-complete Problems

# Towards More NP-Complete Problems

Starting with **SAT**, one can readily show more problems **P** to be NP-complete, each time performing two steps:

- (1) Show that  $\mathbf{P} \in \text{NP}$
- (2) Find a known NP-complete problem  $\mathbf{P}'$  and reduce  $\mathbf{P}' \leq_p \mathbf{P}$

Thousands of problems have now been shown to be NP-complete.  
(See Garey and Johnson for an early survey)

# Towards More NP-Complete Problems

Starting with **SAT**, one can readily show more problems **P** to be NP-complete, each time performing two steps:

- (1) Show that **P**  $\in$  NP
- (2) Find a known NP-complete problem **P'** and reduce **P'**  $\leq_p$  **P**

Thousands of problems have now been shown to be NP-complete.  
(See Garey and Johnson for an early survey)

In this course:

$$\begin{array}{ll} \leq_p \text{ CLIQUE} & \leq_p \text{ INDEPENDENT SET} \\ \text{SAT} \leq_p \text{ 3-SAT} & \leq_p \text{ DIR. HAMILTONIAN PATH} \\ \leq_p \text{ SUBSET SUM} & \leq_p \text{ KNAPSACK} \end{array}$$

# NP-Completeness of **CLIQUE**

**Theorem 7.8:** **CLIQUE** is NP-complete.

**CLIQUE:** Given  $G, k$ , does  $G$  contain a clique of order  $k$ ?

**Proof:**

(1) **CLIQUE**  $\in$  NP

Take the vertex set of a clique of order  $k$  as a certificate.

(2) **CLIQUE** is NP-hard

We show **SAT**  $\leq_p$  **CLIQUE**

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

$$\varphi \text{ satisfiable} \iff G_\varphi \text{ contains clique of order } k_\varphi$$

# SAT $\leq_p$ CLIQUE

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

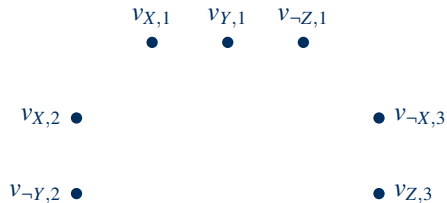
$\varphi$  satisfiable if and only if  $G_\varphi$  contains clique of order  $k_\varphi$

Given  $\varphi = C_1 \wedge \dots \wedge C_k$ :

- Set  $k_\varphi := k$
- For each clause  $C_j$  and literal  $L \in C_j$  add a vertex  $v_{L,j}$
- Add edge  $\{v_{L,j}, v_{K,i}\}$  if  $i \neq j$  and  $L \wedge K$  is satisfiable (that is: if  $L \neq \neg K$  and  $\neg L \neq K$ )

## Example 7.9:

$$\underbrace{(X \vee Y \vee \neg Z)}_{C_1} \wedge \underbrace{(X \vee \neg Y)}_{C_2} \wedge \underbrace{(\neg X \vee Z)}_{C_3}$$





# SAT $\leq_p$ CLIQUE

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

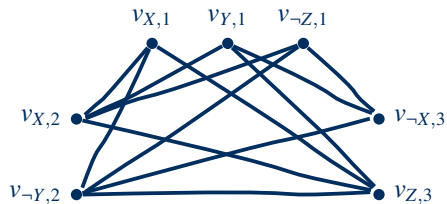
$\varphi$  satisfiable if and only if  $G_\varphi$  contains clique of order  $k_\varphi$

Given  $\varphi = C_1 \wedge \dots \wedge C_k$ :

- Set  $k_\varphi := k$
- For each clause  $C_j$  and literal  $L \in C_j$  add a vertex  $v_{L,j}$
- Add edge  $\{v_{L,j}, v_{K,i}\}$  if  $i \neq j$  and  $L \wedge K$  is satisfiable (that is: if  $L \neq \neg K$  and  $\neg L \neq K$ )

## Example 7.9:

$$\underbrace{(X \vee Y \vee \neg Z)}_{C_1} \wedge \underbrace{(X \vee \neg Y)}_{C_2} \wedge \underbrace{(\neg X \vee Z)}_{C_3}$$



# SAT $\leq_p$ CLIQUE

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

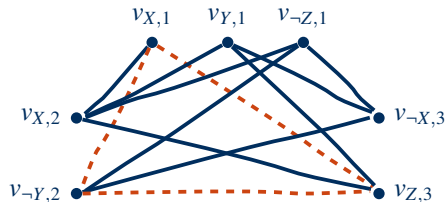
$\varphi$  satisfiable if and only if  $G_\varphi$  contains clique of order  $k_\varphi$

Given  $\varphi = C_1 \wedge \dots \wedge C_k$ :

- Set  $k_\varphi := k$
- For each clause  $C_j$  and literal  $L \in C_j$  add a vertex  $v_{L,j}$
- Add edge  $\{v_{L,j}, v_{K,i}\}$  if  $i \neq j$  and  $L \wedge K$  is satisfiable (that is: if  $L \neq \neg K$  and  $\neg L \neq K$ )

## Example 7.9:

$$\underbrace{(X \vee Y \vee \neg Z)}_{C_1} \wedge \underbrace{(X \vee \neg Y)}_{C_2} \wedge \underbrace{(\neg X \vee Z)}_{C_3}$$



# SAT $\leq_p$ CLIQUE

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

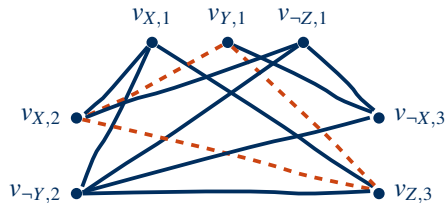
$\varphi$  satisfiable if and only if  $G_\varphi$  contains clique of order  $k_\varphi$

Given  $\varphi = C_1 \wedge \dots \wedge C_k$ :

- Set  $k_\varphi := k$
- For each clause  $C_j$  and literal  $L \in C_j$  add a vertex  $v_{L,j}$
- Add edge  $\{v_{L,j}, v_{K,i}\}$  if  $i \neq j$  and  $L \wedge K$  is satisfiable (that is: if  $L \neq \neg K$  and  $\neg L \neq K$ )

## Example 7.9:

$$\underbrace{(X \vee Y \vee \neg Z)}_{C_1} \wedge \underbrace{(X \vee \neg Y)}_{C_2} \wedge \underbrace{(\neg X \vee Z)}_{C_3}$$



# SAT $\leq_p$ CLIQUE

To every CNF-formula  $\varphi$  assign a graph  $G_\varphi$  and a number  $k_\varphi$  such that

$\varphi$  satisfiable if and only if  $G_\varphi$  contains clique of order  $k_\varphi$

Given  $\varphi = C_1 \wedge \dots \wedge C_k$ :

- Set  $k_\varphi := k$
- For each clause  $C_j$  and literal  $L \in C_j$  add a vertex  $v_{L,j}$
- Add edge  $\{v_{L,j}, v_{K,i}\}$  if  $i \neq j$  and  $L \wedge K$  is satisfiable (that is: if  $L \neq \neg K$  and  $\neg L \neq K$ )

Correctness:

$G_\varphi$  has clique of order  $k$  iff  $\varphi$  is satisfiable.

Complexity:

The reduction is clearly computable in polynomial time.

# NP-Completeness of **INDEPENDENT SET**

## **INDEPENDENT SET**

Input: An undirected graph  $G$  and a natural number  $k$

Problem: Does  $G$  contain  $k$  vertices that share no edges (independent set)?

**Theorem 7.10:** **INDEPENDENT SET** is NP-complete.

# NP-Completeness of **INDEPENDENT SET**

## **INDEPENDENT SET**

Input: An undirected graph  $G$  and a natural number  $k$

Problem: Does  $G$  contain  $k$  vertices that share no edges (independent set)?

**Theorem 7.10:** **INDEPENDENT SET** is NP-complete.

**Proof:** Hardness by reduction **CLIQUE**  $\leq_p$  **INDEPENDENT SET**:

- Given  $G := (V, E)$  construct  $\bar{G} := (V, \{\{u, v\} \mid \{u, v\} \notin E \text{ and } u \neq v\})$

# NP-Completeness of **INDEPENDENT SET**

## **INDEPENDENT SET**

Input: An undirected graph  $G$  and a natural number  $k$

Problem: Does  $G$  contain  $k$  vertices that share no edges (independent set)?

**Theorem 7.10:** **INDEPENDENT SET** is NP-complete.

**Proof:** Hardness by reduction **CLIQUE**  $\leq_p$  **INDEPENDENT SET**:

- Given  $G := (V, E)$  construct  $\bar{G} := (V, \{\{u, v\} \mid \{u, v\} \notin E \text{ and } u \neq v\})$
- A set  $X \subseteq V$  induces a clique in  $G$  iff  $X$  induces an independent set in  $\bar{G}$ .
- **Reduction:**  $G$  has a clique of order  $k$  iff  $\bar{G}$  has an independent set of order  $k$ .

□

# Summary and Outlook

NP-complete problems are the hardest in NP

Polynomial runs of NTMs can be described in propositional logic (Cook-Levin)

**CLIQUE** and **INDEPENDENT SET** are also NP-complete

## What's next?

- More examples of problems
- The limits of NP
- Space complexities