

NP-Vollständigkeit

Eine Formel α in **konjunktiver Normalform** hat die Form

$$\alpha \equiv k_1 \wedge k_2 \wedge \dots \wedge k_r.$$

- Die *Klauseln* k_1, \dots, k_r sind Disjunktionen von *Literalen*, also Disjunktionen von Variablen oder negierten Variablen.
- α heißt *erfüllbar*, wenn es eine Belegung der aussagenlogischen Variablen gibt, die α wahr macht.

Ein Beispiel:

$$\alpha \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

Ein Beispiel:

$$\alpha \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

α wird durch die Belegung $x = \text{wahr}$ und $y = \text{wahr}$ erfüllt.

Ein Beispiel:

$$\alpha \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

α wird durch die Belegung $x = \text{wahr}$ und $y = \text{wahr}$ erfüllt. Die Formel

$$\beta \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

ist hingegen nicht erfüllbar.

Ein Beispiel:

$$\alpha \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y).$$

α wird durch die Belegung $x = \text{wahr}$ und $y = \text{wahr}$ erfüllt. Die Formel

$$\beta \equiv (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

ist hingegen nicht erfüllbar.

KNF-SAT ist die Menge aller erfüllbaren Formeln in konjunktiver Normalform.

KNF-SAT ist NP -vollständig

KNF-SAT ist NP-vollständig.

KNF-SAT ist NP-vollständig.

- Jetzt treten wir die Lawine los:

KNF-SAT ist NP-vollständig.

- Jetzt treten wir die Lawine los:

Wenn $L_1 \leq_p L_2$ und wenn L_1 ein NP-hartes Problem ist, dann ist auch L_2 ein NP-hartes Problem.

KNF-SAT ist NP-vollständig.

- Jetzt treten wir die Lawine los:

Wenn $L_1 \leq_p L_2$ und wenn L_1 ein NP-hartes Problem ist, dann ist auch L_2 ein NP-hartes Problem.

Zuerst reduzieren wir KNF-SAT auf 3-SAT und CLIQUE.

KNF-SAT ist NP-vollständig.

- Jetzt treten wir die Lawine los:

Wenn $L_1 \leq_p L_2$ und wenn L_1 ein NP-hartes Problem ist, dann ist auch L_2 ein NP-hartes Problem.

Zuerst reduzieren wir KNF-SAT auf 3-SAT und CLIQUE.

- Wir zeigen den Satz von Cook und Levin später.

3-SAT ist NP-vollständig

- Eine **3-KNF** Formel ist eine Formel in konjunktiver Normalform mit höchstens **drei** Variablen pro Klausel.

- Eine **3-KNF** Formel ist eine Formel in konjunktiver Normalform mit höchstens **drei** Variablen pro Klausel.
- **3-SAT** besteht aus allen **erfüllbaren** aussagenlogischen 3-KNF Formeln.

- Eine **3-KNF** Formel ist eine Formel in konjunktiver Normalform mit höchstens **drei** Variablen pro Klausel.
- **3-SAT** besteht aus allen **erfüllbaren** aussagenlogischen 3-KNF Formeln.
- **3-SAT** gehört zu NP : Rate eine Belegung und verifiziere, dass jede Klausel erfüllt ist.

- Eine **3-KNF** Formel ist eine Formel in konjunktiver Normalform mit höchstens **drei** Variablen pro Klausel.
- **3-SAT** besteht aus allen **erfüllbaren** aussagenlogischen 3-KNF Formeln.
- **3-SAT** gehört zu NP : Rate eine Belegung und verifiziere, dass jede Klausel erfüllt ist.

Zeige die Reduktion **KNF-SAT** \leq_p **3-SAT**.

- Die KNF-Formel

$$w = k_1 \wedge \dots \wedge k_r$$

sei Eingabe für KNF-SAT.

- Die KNF-Formel

$$w = k_1 \wedge \dots \wedge k_r$$

sei Eingabe für KNF-SAT.

- Wir transformieren jede Klausel k_j in eine 3-KNF Formel k_j^* . Die transformierte Eingabe ist dann

$$M(w) \equiv k_1^* \wedge \dots \wedge k_r^*.$$

- Die KNF-Formel

$$w = k_1 \wedge \dots \wedge k_r$$

sei Eingabe für KNF-SAT.

- Wir transformieren jede Klausel k_j in eine 3-KNF Formel k_j^* . Die transformierte Eingabe ist dann

$$M(w) \equiv k_1^* \wedge \dots \wedge k_r^*.$$

- Wir zeigen dann, dass
 - ▶ w genau dann erfüllbar ist, wenn $M(w)$ erfüllbar ist

- Die KNF-Formel

$$w = k_1 \wedge \dots \wedge k_r$$

sei Eingabe für KNF-SAT.

- Wir transformieren jede Klausel k_j in eine 3-KNF Formel k_j^* . Die transformierte Eingabe ist dann

$$M(w) \equiv k_1^* \wedge \dots \wedge k_r^*.$$

- Wir zeigen dann, dass
 - ▶ w genau dann erfüllbar ist, wenn $M(w)$ erfüllbar ist
 - ▶ und, dass $M(w)$ effizient bestimmt werden kann.

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale
 $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv$$

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge$$

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge (\neg x_{i,1} \vee l_3 \vee x_{i,2})$$

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge (\neg x_{i,1} \vee l_3 \vee x_{i,2}) \wedge (\neg x_{i,2} \vee l_4 \vee l_5).$$

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge (\neg x_{i,1} \vee l_3 \vee x_{i,2}) \wedge (\neg x_{i,2} \vee l_4 \vee l_5).$$

- Behauptung:**

$k_i(x_1, \dots, x_n)$ wahr \Leftrightarrow es gibt $x_{i,1}, x_{i,2}, \dots$, so dass $k_i^*(x_{i,1}, x_{i,2}, \dots; x_1, \dots, x_n)$ wahr ist.

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge (\neg x_{i,1} \vee l_3 \vee x_{i,2}) \wedge (\neg x_{i,2} \vee l_4 \vee l_5).$$

- Behauptung:**

$k_i(x_1, \dots, x_n)$ wahr \Leftrightarrow es gibt $x_{i,1}, x_{i,2}, \dots$, so dass $k_i^*(x_{i,1}, x_{i,2}, \dots; x_1, \dots, x_n)$ wahr ist.

- $w = k_1 \wedge \dots \wedge k_r$ ist genau dann erfüllbar, wenn $k_1^* \wedge \dots \wedge k_r^*$ erfüllbar ist.

Die Klausel $k_i \equiv l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5$ habe die Literale $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}, \dots$ und setzen

$$k_i^* \equiv (l_1 \vee l_2 \vee x_{i,1}) \wedge (\neg x_{i,1} \vee l_3 \vee x_{i,2}) \wedge (\neg x_{i,2} \vee l_4 \vee l_5).$$

- Behauptung:**

$k_i(x_1, \dots, x_n)$ wahr \Leftrightarrow es gibt $x_{i,1}, x_{i,2}, \dots$, so dass $k_i^*(x_{i,1}, x_{i,2}, \dots; x_1, \dots, x_n)$ wahr ist.

- $w = k_1 \wedge \dots \wedge k_r$ ist genau dann erfüllbar, wenn $k_1^* \wedge \dots \wedge k_r^*$ erfüllbar ist.
- Die Transformation $M(w) = k_1^* \wedge \dots \wedge k_r^*$ ist effizient berechenbar.

2-SAT ist effizient lösbar

- Wie schwierig ist 2-SAT?

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.
 - ▶ Berechne die Konsequenzen:

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.
 - ▶ Berechne die Konsequenzen:
 - ★ In den Klauseln, in denen $\neg x_1$ vorkommt, ist der Wahrheitswert der verbleibenden Variable **erzwungen**.

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.
 - ▶ Berechne die Konsequenzen:
 - ★ In den Klauseln, in denen $\neg x_1$ vorkommt, ist der Wahrheitswert der verbleibenden Variable **erzwungen**.
 - ★ Ermittle alle unmittelbaren und mittelbaren Konsequenzen.

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.
 - ▶ Berechne die Konsequenzen:
 - ★ In den Klauseln, in denen $\neg x_1$ vorkommt, ist der Wahrheitswert der verbleibenden Variable **erzwungen**.
 - ★ Ermittle alle unmittelbaren und mittelbaren Konsequenzen.
 - ▶ Wenn die Nicht-Erfüllbarkeit festgestellt wird, dann setze $x_1 = \text{falsch}$, ansonsten fahre mit x_2 fort.

- Wie schwierig ist 2-SAT?
- Sei $w = k_1 \wedge \dots \wedge k_r$ eine 2-KNF Formel.
 - ▶ Setze $x_1 = \text{wahr}$.
 - ▶ Berechne die Konsequenzen:
 - ★ In den Klauseln, in denen $\neg x_1$ vorkommt, ist der Wahrheitswert der verbleibenden Variable **erzwungen**.
 - ★ Ermittle alle unmittelbaren und mittelbaren Konsequenzen.
 - ▶ Wenn die Nicht-Erfüllbarkeit festgestellt wird, dann setze $x_1 = \text{falsch}$, ansonsten fahre mit x_2 fort.
- **2-SAT** $\in \text{P}$, aber **3-SAT** ist NP-vollständig.

CLIQUE ist NP-vollständig

Gegeben sei ein Graph mit maximalem Knotengrad d . Sei k die maximale Clique-Größe. Was gilt dann?

- (1) $k = \Theta(d)$
- (2) $k = O(d)$
- (3) $k = \Omega(d)$

Gegeben sei ein Graph mit maximalem Knotengrad d . Sei k die maximale Clique-Größe. Was gilt dann?

- (1) $k = \Theta(d)$
- (2) $k = O(d)$
- (3) $k = \Omega(d)$

Auflösung:

Gegeben sei ein Graph mit maximalem Knotengrad d . Sei k die maximale Clique-Größe. Was gilt dann?

- (1) $k = \Theta(d)$
- (2) $k = O(d)$
- (3) $k = \Omega(d)$

Auflösung: (2) $k = O(d)$

CLIQUE ist NP-vollständig

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln
 $k_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ sei Eingabe für 3-SAT.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.
 - ▶ Für jede Klausel k_j erfinden wir eine Menge $V_j = \{(i, l_{j,1}), (i, l_{j,2}), (i, l_{j,3})\}$ von drei Knoten

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.
 - ▶ Für jede Klausel k_j erfinden wir eine Menge $V_j = \{(i, l_{j,1}), (i, l_{j,2}), (i, l_{j,3})\}$ von drei Knoten mit $V_w = \bigcup_{i=1}^r V_i$.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.
 - ▶ Für jede Klausel k_j erfinden wir eine Menge $V_j = \{(i, l_{j,1}), (i, l_{j,2}), (i, l_{j,3})\}$ von drei Knoten mit $V_w = \bigcup_{i=1}^r V_i$.
 - ▶ Knoten in V_j werden **nicht** miteinander verbunden.

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.
 - ▶ Für jede Klausel k_j erfinden wir eine Menge $V_j = \{(i, l_{j,1}), (i, l_{j,2}), (i, l_{j,3})\}$ von drei Knoten mit $V_w = \bigcup_{i=1}^r V_i$.
 - ▶ Knoten in V_j werden **nicht** miteinander verbunden.
 - ▶ Ansonsten verbinden wir Knoten (i, l) und (j, l^*) für $i \neq j$ genau dann, wenn

Satz von Karp

3-SAT \leq_p CLIQUE und CLIQUE ist NP-vollständig.

- **CLIQUE** \in NP:
 - ▶ Um $(G, k) \in$ CLIQUE zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass je zwei Knoten durch eine Kante verbunden sind.
- **Die Transformation:**
 - ▶ Die 3-KNF Formel $w = k_1 \wedge \dots \wedge k_r$ mit den Klauseln $k_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ sei Eingabe für 3-SAT.
 - ▶ Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für CLIQUE.
 - ▶ Für jede Klausel k_j erfinden wir eine Menge $V_j = \{(i, l_{j,1}), (i, l_{j,2}), (i, l_{j,3})\}$ von drei Knoten mit $V_w = \bigcup_{i=1}^r V_i$.
 - ▶ Knoten in V_j werden **nicht** miteinander verbunden.
 - ▶ Ansonsten verbinden wir Knoten (i, l) und (j, l^*) für $i \neq j$ genau dann, wenn l und l^* sich **nicht widersprechen**, d.h. wenn l und l^* simultan erfüllt werden können.

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

Eine Clique besitzt höchstens einen Knoten aus einer Menge V_i , denn keine zwei Knoten aus V_i sind durch eine Kante verbunden.

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

Eine Clique besitzt höchstens einen Knoten aus einer Menge V_i , denn keine zwei Knoten aus V_i sind durch eine Kante verbunden.

- Wenn $w \in 3\text{-SAT}$, dann besitzt G_w eine Clique der Größe r . Warum?

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

Eine Clique besitzt höchstens einen Knoten aus einer Menge V_i , denn keine zwei Knoten aus V_i sind durch eine Kante verbunden.

- Wenn $w \in 3\text{-SAT}$, dann besitzt G_w eine Clique der Größe r . Warum?
 - ▶ w besitzt eine erfüllende Belegung b und b erfüllt mindestens ein Literal l_i in der i ten Klausel.

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

Eine Clique besitzt höchstens einen Knoten aus einer Menge V_i , denn keine zwei Knoten aus V_i sind durch eine Kante verbunden.

- Wenn $w \in 3\text{-SAT}$, dann besitzt G_w eine Clique der Größe r . Warum?
 - ▶ w besitzt eine erfüllende Belegung b und b erfüllt mindestens ein Literal l_i in der i ten Klausel.
 - ▶ Die Knotenmenge

$$\{(i, l_i) \mid 1 \leq i \leq r\}$$

ist eine Clique,

- Die maximale Größe einer Clique ist durch r , die Anzahl der Klauseln, beschränkt. Warum?

Eine Clique besitzt höchstens einen Knoten aus einer Menge V_i , denn keine zwei Knoten aus V_i sind durch eine Kante verbunden.

- Wenn $w \in 3\text{-SAT}$, dann besitzt G_w eine Clique der Größe r . Warum?
 - ▶ w besitzt eine erfüllende Belegung b und b erfüllt mindestens ein Literal l_i in der i ten Klausel.
 - ▶ Die Knotenmenge

$$\{(i, l_i) \mid 1 \leq i \leq r\}$$

ist eine Clique, denn für je zwei Knoten (i, l_i) und (j, l_j) werden l_i und l_j durch b erfüllt: l_i, l_j widersprechen sich nicht und die Knoten sind deshalb durch eine Kante verbunden.

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$.
Warum?

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_j) in V_i .

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_i) in V_i .
 - ▶ Wenn $(i, l), (j, l^*)$ in C liegen, dann widersprechen sich die Literale l und l^* nicht.

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_i) in V_i .
 - ▶ Wenn $(i, l), (j, l^*)$ in C liegen, dann widersprechen sich die Literale l und l^* nicht.
Alle Literale, die in C vorkommen, können simultan erfüllt werden!

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_i) in V_i .
 - ▶ Wenn $(i, l), (j, l^*)$ in C liegen, dann widersprechen sich die Literale l und l^* nicht.
Alle Literale, die in C vorkommen, können simultan erfüllt werden!
 - ▶ w ist erfüllbar, da die Literale von C simultan erfüllbar sind und da C ein Literal aus jeder Klausel k_i besitzt.

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_i) in V_i .
 - ▶ Wenn $(i, l), (j, l^*)$ in C liegen, dann widersprechen sich die Literale l und l^* nicht.
Alle Literale, die in C vorkommen, können simultan erfüllt werden!
 - ▶ w ist erfüllbar, da die Literale von C simultan erfüllbar sind und da C ein Literal aus jeder Klausel k_i besitzt.
- Also gilt insgesamt:

$$w \in 3\text{-SAT} \Leftrightarrow G_w \text{ hat eine Clique der Größe } r$$

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3\text{-SAT}$. Warum?
 - ▶ Die Clique C enthält für jedes i höchstens einen Knoten aus einer Menge V_i .
 - ▶ Da C aber r Knoten enthält, besitzt C genau einen Knoten (i, l_i) in V_i .
 - ▶ Wenn $(i, l), (j, l^*)$ in C liegen, dann widersprechen sich die Literale l und l^* nicht.
Alle Literale, die in C vorkommen, können simultan erfüllt werden!
 - ▶ w ist erfüllbar, da die Literale von C simultan erfüllbar sind und da C ein Literal aus jeder Klausel k_i besitzt.
- Also gilt insgesamt:

$w \in 3\text{-SAT} \Leftrightarrow G_w$ hat eine Clique der Größe r

und wir definieren die effizient berechenbare Transformation

$$M(w) = (G_w, r).$$

Wir haben mit dem graph-theoretischen Problem CLIQUE über die Aussagenlogik geredet:

Erfüllende Belegungen und Cliques der Größe r entsprechen sich wechselseitig.

Independent Set ist NP-vollständig

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.

Falls G der vollständige Graph mit n Knoten ist, dann gilt:

- (1) $|I| = O(1)$
- (2) $|I| = \Theta(\log n)$
- (3) $|I| = \Theta(\sqrt{n})$
- (4) $|I| = \Theta(n/\log n)$
- (5) $|I| = \Theta(n)$

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.

Falls G der vollständige Graph mit n Knoten ist, dann gilt:

- (1) $|I| = O(1)$
- (2) $|I| = \Theta(\log n)$
- (3) $|I| = \Theta(\sqrt{n})$
- (4) $|I| = \Theta(n/\log n)$
- (5) $|I| = \Theta(n)$

Auflösung:

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.

Falls G der vollständige Graph mit n Knoten ist, dann gilt:

- (1) $|I| = O(1)$
- (2) $|I| = \Theta(\log n)$
- (3) $|I| = \Theta(\sqrt{n})$
- (4) $|I| = \Theta(n/\log n)$
- (5) $|I| = \Theta(n)$

Auflösung: (1) $|I| = O(1)$

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
- Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
 - Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.
- In **IS** sucht man also große, „kollisionsfreie“ Knotenmengen.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
 - Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.
- In **IS** sucht man also große, „kollisionsfreie“ Knotenmengen.
 - **IS** \in NP:

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
- Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.

- In **IS** sucht man also große, „kollisionsfreie“ Knotenmengen.
- **IS** \in NP:
 - ▶ Um $(G, k) \in$ **IS** zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass keine zwei Knoten durch eine Kante verbunden sind.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
- Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.

- In *IS* sucht man also große, „kollisionsfreie“ Knotenmengen.
- $IS \in NP$:
 - ▶ Um $(G, k) \in IS$ zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass keine zwei Knoten durch eine Kante verbunden sind.
- CLIQUE und IS sind sich sehr ähnlich.

Independent Set

Sei $G = (V, E)$ ein ungerichteter Graph.

- Eine Teilmenge $I \subseteq V$ heißt **unabhängig**, falls **keine** zwei Knoten durch eine Kante verbunden sind.
- Eine Eingabe (G, k) gehört genau dann zum **Independent Set Problem IS**, wenn G eine unabhängige Menge der Größe mindestens k besitzt.

- In **IS** sucht man also große, „kollisionsfreie“ Knotenmengen.
- **IS** \in NP:
 - ▶ Um $(G, k) \in$ **IS** zu überprüfen, raten wir k Knoten und
 - ▶ verifizieren, dass keine zwei Knoten durch eine Kante verbunden sind.
- **CLIQUE** und **IS** sind sich sehr ähnlich.
- Zeige

$$\text{CLIQUE} \leq_p \text{IS}.$$

Independent Set ist NP-vollständig

CLIQUE \leq_p IS und IS ist NP-vollständig.

Independent Set ist NP-vollständig

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\overline{G} = (V, \overline{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\overline{G}, k)$ als transformierte Eingabe.

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

Independent Set ist NP-vollständig

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

$(G, k) \in \text{CLIQUE}$

$\Leftrightarrow G$ hat eine Clique C der Größe mindestens k

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

$(G, k) \in \text{CLIQUE}$

- \Leftrightarrow G hat eine Clique C der Größe mindestens k
- \Leftrightarrow Je zwei Knoten von C sind durch eine Kante in G verbunden

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

$(G, k) \in \text{CLIQUE}$

- \Leftrightarrow G hat eine Clique C der Größe mindestens k
- \Leftrightarrow Je zwei Knoten von C sind durch eine Kante in G verbunden
- \Leftrightarrow Keine zwei Knoten von C sind durch eine Kante in \bar{G} verbunden

CLIQUE \leq_p IS und IS ist NP-vollständig.

Die Transformation:

Für $G = (V, E)$ sei $\bar{G} = (V, \bar{E})$ der Komplementgraph von G .

- Für die Eingabe $w = (G, k)$ für CLIQUE definieren wir $M(w) = (\bar{G}, k)$ als transformierte Eingabe.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

$(G, k) \in \text{CLIQUE}$

- $\Leftrightarrow G$ hat eine Clique C der Größe mindestens k
- \Leftrightarrow Je zwei Knoten von C sind durch eine Kante in G verbunden
- \Leftrightarrow Keine zwei Knoten von C sind durch eine Kante in \bar{G} verbunden
- $\Leftrightarrow (\bar{G}, k) \in \text{IS}$.

Set Cover ist NP-vollständig

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

$$\text{SC} = \{(A_1, \dots, A_m; k) \mid \text{es gibt } i_1, \dots, i_k \text{ mit}$$

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

$$\text{SC} = \{(A_1, \dots, A_m; k) \mid \text{es gibt } i_1, \dots, i_k \text{ mit } \bigcup_{j=1}^k A_{i_j} = \bigcup_{j=1}^m A_j\}.$$

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

$$\text{SC} = \{(A_1, \dots, A_m; k) \mid \text{es gibt } i_1, \dots, i_k \text{ mit } \bigcup_{j=1}^k A_{i_j} = \bigcup_{j=1}^m A_j\}.$$

- SC \in NP:

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

$$\text{SC} = \{(A_1, \dots, A_m; k) \mid \text{es gibt } i_1, \dots, i_k \text{ mit } \bigcup_{j=1}^k A_{i_j} = \bigcup_{j=1}^m A_j\}.$$

- $\text{SC} \in \text{NP}$: Für Eingabe $(A_1, \dots, A_m; k)$ rate k Mengen und verifiziere, dass alle Elemente überdeckt werden.

Eine Menge U von Aufgaben und m Prozesse sind gegeben.

- Prozess i kann alle Aufgaben in $A_i \subseteq U$ erledigen.
- **Das Ziel:** Bestimme eine möglichst kleine Anzahl von Prozessen, die alle Aufgaben erledigen.
- Diese Problemstellung ist äquivalent zum Set Cover Problem SC, wobei

$$\text{SC} = \{(A_1, \dots, A_m; k) \mid \text{es gibt } i_1, \dots, i_k \text{ mit } \bigcup_{j=1}^k A_{i_j} = \bigcup_{j=1}^m A_j\}.$$

- $\text{SC} \in \text{NP}$: Für Eingabe $(A_1, \dots, A_m; k)$ rate k Mengen und verifiziere, dass alle Elemente überdeckt werden.
- Wie schwierig ist SC?

Vertex Cover ist NP-vollständig

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\ddot{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \ddot{U} besitzen.

Gilt dann für jeden Graph G mit n Knoten, dass es eine Überdeckung \ddot{U} mit der Größe $|\ddot{U}| \leq n/2$ gibt?

- (1) Ja
- (2) Nein
- (3) Keine Ahnung

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\ddot{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \ddot{U} besitzen.

Gilt dann für jeden Graph G mit n Knoten, dass es eine Überdeckung \ddot{U} mit der Größe $|\ddot{U}| \leq n/2$ gibt?

- (1) Ja
- (2) Nein
- (3) Keine Ahnung

Auflösung:

Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\ddot{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \ddot{U} besitzen.

Gilt dann für jeden Graph G mit n Knoten, dass es eine Überdeckung \ddot{U} mit der Größe $|\ddot{U}| \leq n/2$ gibt?

- (1) Ja
- (2) Nein
- (3) Keine Ahnung

Auflösung: (2) Nein, z.B. vollständiger Graph

- Um die Komplexität von **Set Cover** zu bestimmen, betrachten wir das **Vertex Cover** Problem VC.

- Um die Komplexität von **Set Cover** zu bestimmen, betrachten wir das **Vertex Cover** Problem VC.
- Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\tilde{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \tilde{U} besitzen.

- Um die Komplexität von **Set Cover** zu bestimmen, betrachten wir das **Vertex Cover** Problem VC.
- Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\tilde{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \tilde{U} besitzen.

VC = $\{(G, k) \mid G \text{ besitzt eine Menge } \tilde{U} \text{ von } k \text{ Knoten, so dass jede Kante einen Endpunkt in der Menge } \tilde{U} \text{ besitzt.}\}$

- Um die Komplexität von **Set Cover** zu bestimmen, betrachten wir das **Vertex Cover** Problem VC.
 - Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\ddot{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \ddot{U} besitzen.
- VC** = $\{(G, k) \mid G \text{ besitzt eine Menge } \ddot{U} \text{ von } k \text{ Knoten, so dass jede Kante einen Endpunkt in der Menge } \ddot{U} \text{ besitzt.}\}$

VC \in NP:

- Um die Komplexität von **Set Cover** zu bestimmen, betrachten wir das **Vertex Cover** Problem VC.
- Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge $\ddot{U} \subseteq V$ heißt eine **Überdeckung**, wenn alle Kanten einen Endpunkt in \ddot{U} besitzen.

VC = $\{(G, k) \mid G \text{ besitzt eine Menge } \ddot{U} \text{ von } k \text{ Knoten, so dass jede Kante einen Endpunkt in der Menge } \ddot{U} \text{ besitzt.}\}$

VC \in NP:

- Für Eingabe (G, k) rate k Knoten.
- Verifiziere, dass alle Kanten mindestens einen geratenen Knoten als Endpunkt haben.

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge
$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge
$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$
- $(G, k) \in VC$

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \bar{U} der Größe k

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

$\Leftrightarrow (A_1, \dots, A_n; k) \in SC$

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

$\Leftrightarrow (A_1, \dots, A_n; k) \in SC$

- **Die Transformation:**

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

$\Leftrightarrow (A_1, \dots, A_n; k) \in SC$

- **Die Transformation:** Für die Eingabe $w = (G, k)$ von VC definieren wir die transformierte Eingabe $M(w) = (A_1, \dots, A_n; k)$.

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

$\Leftrightarrow (A_1, \dots, A_n; k) \in SC$

- **Die Transformation:** Für die Eingabe $w = (G, k)$ von VC definieren wir die transformierte Eingabe $M(w) = (A_1, \dots, A_n; k)$.
 - ▶ $M(w)$ ist effizient berechenbar.

SC ist mindestens so schwer wie VC

$VC \leq_p SC$.

- Für $G = (\{1, \dots, n\}, E)$ definiere für jeden Knoten $v \in V$ die Menge

$$A_v = \{e \mid e \text{ hat } v \text{ als Endpunkt}\}.$$

- $(G, k) \in VC$

$\Leftrightarrow G$ hat eine Überdeckung \tilde{U} der Größe k

\Leftrightarrow Die Mengen A_v für $v \in \tilde{U}$ überdecken alle Kanten

$\Leftrightarrow (A_1, \dots, A_n; k) \in SC$

- **Die Transformation:** Für die Eingabe $w = (G, k)$ von VC definieren wir die transformierte Eingabe $M(w) = (A_1, \dots, A_n; k)$.

- ▶ $M(w)$ ist effizient berechenbar.
- ▶ $(G, k) \in VC \Leftrightarrow M(w) \in SC$.

$IS \leq_p VC.$

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

$(G, k) \in IS \iff G$ hat eine unabhängige Menge I der Größe k

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

$(G, k) \in IS \iff G$ hat eine unabhängige Menge I der Größe k
 \iff jede Kante hat mindestens einen Endpunkt in $V \setminus I$

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

- $(G, k) \in IS \Leftrightarrow G$ hat eine unabhängige Menge I der Größe k
- \Leftrightarrow jede Kante hat mindestens einen Endpunkt in $V \setminus I$
- $\Leftrightarrow (G, |V| - k) \in VC$.

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

- $(G, k) \in IS \Leftrightarrow G$ hat eine unabhängige Menge I der Größe k
- \Leftrightarrow jede Kante hat mindestens einen Endpunkt in $V \setminus I$
- $\Leftrightarrow (G, |V| - k) \in VC$.

Die Transformation: Für Eingabe $w = (G, k)$ definieren wir die transformierte Eingabe $M(w) = (G, |V| - k)$.

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

- $(G, k) \in IS \Leftrightarrow G$ hat eine unabhängige Menge I der Größe k
- \Leftrightarrow jede Kante hat mindestens einen Endpunkt in $V \setminus I$
- $\Leftrightarrow (G, |V| - k) \in VC$.

Die Transformation: Für Eingabe $w = (G, k)$ definieren wir die transformierte Eingabe $M(w) = (G, |V| - k)$.

- $M(w)$ ist effizient berechenbar.

$IS \leq_p VC$.

(G, k) sei die Eingabe für das Independent Set Problem. Dann gilt

- $(G, k) \in IS \Leftrightarrow G$ hat eine unabhängige Menge I der Größe k
- \Leftrightarrow jede Kante hat mindestens einen Endpunkt in $V \setminus I$
- $\Leftrightarrow (G, |V| - k) \in VC$.

Die Transformation: Für Eingabe $w = (G, k)$ definieren wir die transformierte Eingabe $M(w) = (G, |V| - k)$.

- $M(w)$ ist effizient berechenbar.
- $(G, k) \in IS \Leftrightarrow M(w) \in VC$.

- Wir wissen, dass alle bisher betrachteten Probleme, also

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** in NP liegen.

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT**, **3-SAT**, **CLIQUE**, **IS**, **VC**, **SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
KNF-SAT \leq_p **3-SAT**,

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
 $\text{KNF-SAT} \leq_p \text{3-SAT}$, $\text{3-SAT} \leq_p \text{CLIQUE}$,

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
 $\text{KNF-SAT} \leq_p \text{3-SAT}$, $\text{3-SAT} \leq_p \text{CLIQUE}$, $\text{CLIQUE} \leq_p \text{IS}$,

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
 $\text{KNF-SAT} \leq_p \text{3-SAT}$, $\text{3-SAT} \leq_p \text{CLIQUE}$, $\text{CLIQUE} \leq_p \text{IS}$, $\text{IS} \leq_p \text{VC}$,

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT**, **3-SAT**, **CLIQUE**, **IS**, **VC**, **SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
 $\text{KNF-SAT} \leq_p \text{3-SAT}$, $\text{3-SAT} \leq_p \text{CLIQUE}$, $\text{CLIQUE} \leq_p \text{IS}$, $\text{IS} \leq_p \text{VC}$,
 $\text{VC} \leq_p \text{SC}$.

- Wir wissen, dass alle bisher betrachteten Probleme, also **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** in NP liegen.
- Bisher haben wir die folgenden Reduktionen hergeleitet:
 $KNF-SAT \leq_p 3-SAT$, $3-SAT \leq_p CLIQUE$, $CLIQUE \leq_p IS$, $IS \leq_p VC$, $VC \leq_p SC$.

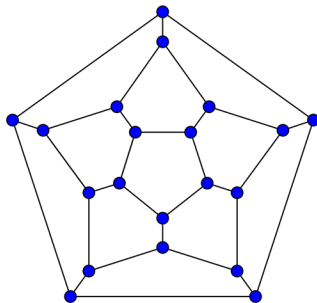
Die Lawine gewinnt an Fahrt

Die Probleme **KNF-SAT, 3-SAT, CLIQUE, IS, VC, SC** sind alle NP -vollständig, **wenn $KNF-SAT$ NP -vollständig ist.**

Schwierige Wegeprobleme in Graphen

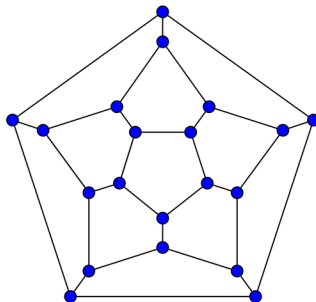
Im **Hamiltonschen Kreis-Problem HC** ist für einen ungerichteten Graphen G zu entscheiden, ob G einen **Hamiltonschen Kreis** besitzt, also einen Kreis, der jeden Knoten genau einmal durchläuft.

Im **Hamiltonschen Kreis-Problem HC** ist für einen ungerichteten Graphen G zu entscheiden, ob G einen **Hamiltonschen Kreis** besitzt, also einen Kreis, der jeden Knoten genau einmal durchläuft.



Quelle: Wikimedia, Fullerene Graphs

Im **Hamiltonschen Kreis-Problem HC** ist für einen ungerichteten Graphen G zu entscheiden, ob G einen **Hamiltonschen Kreis** besitzt, also einen Kreis, der jeden Knoten genau einmal durchläuft.



Quelle: Wikimedia, Fullerene Graphs

Hat dieser Graph einen Hamiltonschen Kreis?

Hat der n -dimensionale Würfel einen Hamiltonschen Kreis?

- **DHC** ist das Hamiltonsche Kreis-Problem für gerichtete Graphen.

- **DHC** ist das Hamiltonsche Kreis-Problem für gerichtete Graphen.
- Wir erinnern an das **Traveling Salesman Problem TSP**:
 - ▶ für den vollständigen Graphen $V_n = (\{1, \dots, n\}, \{ \{i, j\} \mid i \neq j \})$,

- **DHC** ist das Hamiltonsche Kreis-Problem für gerichtete Graphen.
- Wir erinnern an das **Traveling Salesman Problem TSP**:
 - ▶ für den vollständigen Graphen $V_n = (\{1, \dots, n\}, \{ \{i, j\} \mid i \neq j \})$,
 - ▶ die Kantenlängen länge: $\{ \{i, j\} \mid i \neq j \} \rightarrow \mathbb{R}$

- **DHC** ist das Hamiltonsche Kreis-Problem für gerichtete Graphen.
- Wir erinnern an das **Traveling Salesman Problem TSP**:
 - ▶ für den vollständigen Graphen $V_n = (\{1, \dots, n\}, \{ \{i, j\} \mid i \neq j \})$,
 - ▶ die Kantenlängenlänge: $\{ \{i, j\} \mid i \neq j \} \rightarrow \mathbb{R}$
 - ▶ sowie den Schwellenwert T zu entscheiden ist, ob es eine Rundreise der Länge höchstens T gibt.

- **DHC** ist das Hamiltonsche Kreis-Problem für gerichtete Graphen.
- Wir erinnern an das **Traveling Salesman Problem TSP**:
 - ▶ für den vollständigen Graphen $V_n = (\{1, \dots, n\}, \{ \{i, j\} \mid i \neq j \})$,
 - ▶ die Kantenlängenlänge: $\{ \{i, j\} \mid i \neq j \} \rightarrow \mathbb{R}$
 - ▶ sowie den Schwellenwert T zu entscheiden ist, ob es eine Rundreise der Länge höchstens T gibt.
- Im Problem **LW** des längsten Weges ist ein ungerichteter Graph G und ein Schwellenwert T gegeben.
 - ▶ Es ist zu entscheiden, ob G einen Weg der Länge mindestens T besitzt.

$\text{DHC} \leq_p \text{HC}.$

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.
- **Die Transformation:**

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.
- **Die Transformation:**
 - ▶ In ungerichteten Graphen gibt es „mehr“ Wege, da wir jetzt Kanten in beiden Richtungen durchlaufen können.

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.
- **Die Transformation:**
 - ▶ In ungerichteten Graphen gibt es „mehr“ Wege, da wir jetzt Kanten in beiden Richtungen durchlaufen können.
 - ▶ Deshalb erfinden wir für jeden Knoten $v \in V$ drei Knoten $(v, 0), (v, 1), (v, 2)$ und definieren $V' = V \times \{0, 1, 2\}$ als die Knotenmenge des transformierten Graphen G' .

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.
- **Die Transformation:**
 - ▶ In ungerichteten Graphen gibt es „mehr“ Wege, da wir jetzt Kanten in beiden Richtungen durchlaufen können.
 - ▶ Deshalb erfinden wir für jeden Knoten $v \in V$ drei Knoten $(v, 0), (v, 1), (v, 2)$ und definieren $V' = V \times \{0, 1, 2\}$ als die Knotenmenge des transformierten Graphen G' .
 - ★ Für jede gerichtete Kante $u \rightarrow v \in E$ setzen wir die ungerichtete Kante $(u, 2) - (v, 0)$ ein und versuchen so Kantenrichtung nachzuahmen.

- Der gerichtete Graph $G = (V, E)$ sei Eingabe für DHC.
- **Die Transformation:**
 - ▶ In ungerichteten Graphen gibt es „mehr“ Wege, da wir jetzt Kanten in beiden Richtungen durchlaufen können.
 - ▶ Deshalb erfinden wir für jeden Knoten $v \in V$ drei Knoten $(v, 0), (v, 1), (v, 2)$ und definieren $V' = V \times \{0, 1, 2\}$ als die Knotenmenge des transformierten Graphen G' .
 - ★ Für jede gerichtete Kante $u \rightarrow v \in E$ setzen wir die ungerichtete Kante $(u, 2) - (v, 0)$ ein und versuchen so Kantenrichtung nachzuahmen.
 - ★ Schließlich setzen wir die Kanten $(u, 0) - (u, 1)$ und $(u, 1) - (u, 2)$ ein.

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0)$

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1)$

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1) - (v, 2)$

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1) - (v, 2) - (w, 0)$.

Der ungerichtete Graph

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1) - (v, 2) - (w, 0)$.
- Nach diesem „Strickmuster“ entspricht jedem Hamiltonschen Kreis im gerichteten Graphen G ein Hamiltonscher Kreis im ungerichteten Graphen G' .

Der ungerichtete Graph

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1) - (v, 2) - (w, 0)$.
- Nach diesem „Strickmuster“ entspricht jedem Hamiltonschen Kreis im gerichteten Graphen G ein Hamiltonscher Kreis im ungerichteten Graphen G' . Also gilt

$$G \in \text{DHC} \Rightarrow G' \in \text{HC}.$$

Der ungerichtete Graph

- Wie übersetzen sich Wege im gerichteten Graphen in Wege für den ungerichteten Graphen?
 - ▶ Der gerichtete Weg $u \rightarrow v \rightarrow w$ entspricht
 - ▶ dem ungerichteten Weg $(u, 2) - (v, 0) - (v, 1) - (v, 2) - (w, 0)$.
- Nach diesem „Strickmuster“ entspricht jedem Hamiltonschen Kreis im gerichteten Graphen G ein Hamiltonscher Kreis im ungerichteten Graphen G' . Also gilt

$$G \in \text{DHC} \Rightarrow G' \in \text{HC}.$$

- Aber hat G' Hamiltonsche Kreise, obwohl G keinen Hamiltonschen Kreis besitzt?

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten **und** verlassen!

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten **und** verlassen!
 - ▶ Ein Hamiltonscher Kreis in G' durchläuft jede Knotengruppe $\{(u, 0), (u, 1), (u, 2)\}$ also

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten **und** verlassen!
 - ▶ Ein Hamiltonscher Kreis in G' durchläuft jede Knotengruppe $\{(u, 0), (u, 1), (u, 2)\}$ also entweder **stets** in der Vorwärtsrichtung $(u, 0) - (u, 1) - (u, 2)$ oder

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten **und** verlassen!
 - ▶ Ein Hamiltonscher Kreis in G' durchläuft jede Knotengruppe $\{(u, 0), (u, 1), (u, 2)\}$ also entweder **stets** in der Vorwärtsrichtung $(u, 0) - (u, 1) - (u, 2)$ oder **stets** in der Rückwärtsrichtung $(u, 2) - (u, 1) - (u, 0)$.

Hamiltonsche Kreise im ungerichteten Graphen

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten und verlassen!
 - ▶ Ein Hamiltonscher Kreis in G' durchläuft jede Knotengruppe $\{(u, 0), (u, 1), (u, 2)\}$ also entweder **stets** in der Vorwärtsrichtung $(u, 0) - (u, 1) - (u, 2)$ oder **stets** in der Rückwärtsrichtung $(u, 2) - (u, 1) - (u, 0)$.
- Hamiltonsche Kreise für G' übersetzen sich sofort in Hamiltonsche Kreise für G .

Hamiltonsche Kreise im ungerichteten Graphen

- Wie sehen Hamiltonsche Kreise in G' aus?
 - ▶ Wenn die Knotengruppe von v zum ersten Mal im Knoten $(v, 0)$, bzw. im Knoten $(v, 2)$ betreten wird, muss Knoten $(v, 1)$ durchlaufen werden.
 - Ansonsten kann der Kreis den „mittleren“ Knoten $(v, 1)$ nicht mehr durchlaufen, also nicht mehr betreten **und** verlassen!
 - ▶ Ein Hamiltonscher Kreis in G' durchläuft jede Knotengruppe $\{(u, 0), (u, 1), (u, 2)\}$ also entweder **stets** in der Vorwärtsrichtung $(u, 0) - (u, 1) - (u, 2)$ oder **stets** in der Rückwärtsrichtung $(u, 2) - (u, 1) - (u, 0)$.
- Hamiltonsche Kreise für G' übersetzen sich sofort in Hamiltonsche Kreise für G . Also gilt

$$G' \in \text{HC} \rightarrow G \in \text{DHC}.$$

HC \leq_p TSP.

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) =$$

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

und wählen den Schwellenwert $T = n$.

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

und wählen den Schwellenwert $T = n$.

- Dann gilt

Das Traveling Salesman Problem

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

und wählen den Schwellenwert $T = n$.

- Dann gilt

$$G \in HC \iff G \text{ hat einen Hamiltonschen Kreis}$$

Das Traveling Salesman Problem

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

und wählen den Schwellenwert $T = n$.

- Dann gilt

$$\begin{aligned} G \in HC &\Leftrightarrow G \text{ hat einen Hamiltonschen Kreis} \\ &\Leftrightarrow G \text{ hat eine Rundreise der Länge } n \end{aligned}$$

Das Traveling Salesman Problem

- **Die Transformation:** Für die Eingabe $G = (\{1, \dots, n\}, E)$ von HC wählen wir den vollständigen Graphen V_n , definieren die Längenfunktion

$$\text{länge}(\{i, j\}) = \begin{cases} 1 & \text{wenn } \{i, j\} \in E, \\ 2 & \text{sonst} \end{cases}$$

und wählen den Schwellenwert $T = n$.

- Dann gilt

$$\begin{aligned} G \in HC &\Leftrightarrow G \text{ hat einen Hamiltonschen Kreis} \\ &\Leftrightarrow G \text{ hat eine Rundreise der Länge } n \\ &\Leftrightarrow (V_n, \text{länge}, n) \in \text{TSP}. \end{aligned}$$

- Wird TSP einfacher, wenn wir fordern, dass die Kantenlängen eine Metrik definieren?

Das metrische Traveling Salesman Problem

- Wird TSP einfacher, wenn wir fordern, dass die Kantenlängen eine Metrik definieren?
- Nein, denn unsere Längenfunktion definiert bereits eine Metrik:

Das metrische Traveling Salesman Problem

- Wird TSP einfacher, wenn wir fordern, dass die Kantenlängen eine Metrik definieren?
- Nein, denn unsere Längenfunktion definiert bereits eine Metrik:
Die Dreiecksungleichung

$$\text{länge}(\{i, k\}) \leq \text{länge}(\{i, j\}) + \text{länge}(\{j, k\})$$

gilt,

Das metrische Traveling Salesman Problem

- Wird TSP einfacher, wenn wir fordern, dass die Kantenlängen eine Metrik definieren?
- Nein, denn unsere Längenfunktion definiert bereits eine Metrik:
Die Dreiecksungleichung

$$\text{länge}(\{i, k\}) \leq \text{länge}(\{i, j\}) + \text{länge}(\{j, k\})$$

gilt, da nur die Kantenlängen 1 oder 2 auftreten.

Das metrische Traveling Salesman Problem

- Wird TSP einfacher, wenn wir fordern, dass die Kantenlängen eine Metrik definieren?
- Nein, denn unsere Längenfunktion definiert bereits eine Metrik:
Die Dreiecksungleichung

$$\text{länge}(\{i, k\}) \leq \text{länge}(\{i, j\}) + \text{länge}(\{j, k\})$$

gilt, da nur die Kantenlängen 1 oder 2 auftreten.

- Also haben wir sogar gezeigt, dass **M-TSP**, das metrische Traveling Salesman Problem NP -vollständig ist.

$HC \leq_p LW.$

Die Transformation: Für die Eingabe G von HC müssen wir eine Eingabe für LW basteln.

Die Transformation: Für die Eingabe G von HC müssen wir eine Eingabe für LW basteln.

- **Die wichtige Beobachtung:** G hat genau dann einen Hamiltonschen Kreis, wenn G einen Weg hat, der
 - ▶ in 1 beginnt und
 - ▶ in einem Nachbarn von 1 endet.

Die Transformation: Für die Eingabe G von HC müssen wir eine Eingabe für LW basteln.

- **Die wichtige Beobachtung:** G hat genau dann einen Hamiltonschen Kreis, wenn G einen Weg hat, der
 - ▶ in 1 beginnt und
 - ▶ in einem Nachbarn von 1 endet.
- Also modifizieren wir den Graphen G :
 - ▶ Wir fügen einen neuen Knoten $1'$ hinzu und

Die Transformation: Für die Eingabe G von HC müssen wir eine Eingabe für LW basteln.

- **Die wichtige Beobachtung:** G hat genau dann einen Hamiltonschen Kreis, wenn G einen Weg hat, der
 - ▶ in 1 beginnt und
 - ▶ in einem Nachbarn von 1 endet.
- Also modifizieren wir den Graphen G :
 - ▶ Wir fügen einen neuen Knoten $1'$ hinzu und
 - ▶ verbinden $1'$ mit allen Nachbarn von 1.
- Jetzt gilt offensichtlich für den neuen Graph G'

$G \in HC \Rightarrow G'$ hat einen Weg der Länge n .

Die Transformation: Für die Eingabe G von HC müssen wir eine Eingabe für LW basteln.

- **Die wichtige Beobachtung:** G hat genau dann einen Hamiltonschen Kreis, wenn G einen Weg hat, der
 - ▶ in 1 beginnt und
 - ▶ in einem Nachbarn von 1 endet.
- Also modifizieren wir den Graphen G :
 - ▶ Wir fügen einen neuen Knoten $1'$ hinzu und
 - ▶ verbinden $1'$ mit allen Nachbarn von 1.
- Jetzt gilt offensichtlich für den neuen Graph G'
 $G \in HC \Rightarrow G' \text{ hat einen Weg der Länge } n.$
- Und wenn der Weg der Länge n **nicht** in 1 beginnt und in $1'$ endet?

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonischen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.
 - ▶ Wenn aber G'' einen Weg der Länge $n + 2$ hat, dann muss dieser Weg in 0 oder $0'$ beginnen und in $0'$ oder 0 enden.

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.
 - ▶ Wenn aber G'' einen Weg der Länge $n + 2$ hat, dann muss dieser Weg in 0 oder $0'$ beginnen und in $0'$ oder 0 enden.
 - ▶ Zwangsläufig ist dann aber 1 oder $1'$ der zweite Knoten und

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.
 - ▶ Wenn aber G'' einen Weg der Länge $n + 2$ hat, dann muss dieser Weg in 0 oder $0'$ beginnen und in $0'$ oder 0 enden.
 - ▶ Zwangsläufig ist dann aber 1 oder $1'$ der zweite Knoten und $1'$ oder 1 der vorletzte Knoten.

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.
 - ▶ Wenn aber G'' einen Weg der Länge $n + 2$ hat, dann muss dieser Weg in 0 oder $0'$ beginnen und in $0'$ oder 0 enden.
 - ▶ Zwangsläufig ist dann aber 1 oder $1'$ der zweite Knoten und $1'$ oder 1 der vorletzte Knoten.
 - ▶ **Die Konsequenz:** $G \in HC \Leftrightarrow G''$ hat einen Weg der Länge $n + 2$.

Und noch eine kleine Modifikation

Wie bringen wir Wege dazu,
wie gewünscht zu beginnen und zu enden?

- Wir erfinden zwei weitere Knoten 0 und $0'$ und fügen die Kanten $0 - 1$ sowie $1' - 0'$ hinzu.
- Der neue Graph sei G'' .
 - ▶ Wenn G einen Hamiltonschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \dots - 1' - 0'$ der Länge $n + 2$.
 - ▶ Wenn aber G'' einen Weg der Länge $n + 2$ hat, dann muss dieser Weg in 0 oder $0'$ beginnen und in $0'$ oder 0 enden.
 - ▶ Zwangsläufig ist dann aber 1 oder $1'$ der zweite Knoten und $1'$ oder 1 der vorletzte Knoten.
 - ▶ **Die Konsequenz:** $G \in HC \Leftrightarrow G''$ hat einen Weg der Länge $n + 2$.
- Definiere die Transformation $M(G) = (G'', n + 2)$.

Wir haben die folgenden Reduktionen nachgewiesen:

- $DHC \leq_p HC$,
- $HC \leq_p TSP$ und
- $HC \leq_p LW$.

Wir haben die folgenden Reduktionen nachgewiesen:

- $DHC \leq_p HC$,
- $HC \leq_p TSP$ und
- $HC \leq_p LW$.
- Die NP -Vollständigkeit von HC , DHC , TSP und LW folgt, wenn wir gezeigt haben, dass DHC NP -vollständig ist. (Siehe Skript.)

Der Satz von Cook und Levin: KNF-SAT ist NP -vollständig

- Wir zeigen den Satz von Cook und Levin:

KNF-SAT ist NP-vollständig.

- Wir zeigen den Satz von Cook und Levin:

KNF-SAT ist NP-vollständig.

- Dann treten wir die Lawine los:

Wenn $L_1 \leq_p L_2$ und wenn L_1 ein NP-hartes Problem ist, dann ist auch L_2 ein NP-hartes Problem.

Der Satz von Cook-Levin

KNF-SAT ist NP-vollständig. ([Web](#))

KNF-SAT ist NP -vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, B\}$,

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, B\}$, die Zustandsmenge $\{0, \dots, q\}$,

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, \text{B}\}$, die Zustandsmenge $\{0, \dots, q\}$, mit Anfangszustand 0 und 1 als einzigem akzeptierenden Zustand.

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, B\}$, die Zustandsmenge $\{0, \dots, q\}$, mit Anfangszustand 0 und 1 als einzigem akzeptierenden Zustand.
 - ▶ Wir können annehmen, dass **alle** Berechnungen auf einer Eingabe w **dieselbe** Zeit $T = T(|w|)$ in Anspruch nehmen:

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, \text{B}\}$, die Zustandsmenge $\{0, \dots, q\}$, mit Anfangszustand 0 und 1 als einzigem akzeptierenden Zustand.
 - ▶ Wir können annehmen, dass **alle** Berechnungen auf einer Eingabe w **dieselbe** Zeit $T = T(|w|)$ in Anspruch nehmen: Lasse M , wenn nötig „auf der Stelle treten“.

KNF-SAT ist NP-vollständig. (Web)

- Für ein beliebiges Problem $L \in \text{NP}$ müssen wir die Reduktion $L \leq_p \text{KNF-SAT}$ nachweisen.
- Was wissen wir über L ?
 - ▶ $L = L(M)$ gilt für eine nichtdeterministische Turingmaschine M , die in polynomieller Zeit $T(n)$ arbeitet.
 - ▶ Nach einer entsprechenden Schönheitsoperation: M hat das Bandalphabet $\Gamma = \{0, 1, B\}$, die Zustandsmenge $\{0, \dots, q\}$, mit Anfangszustand 0 und 1 als einzigem akzeptierenden Zustand.
 - ▶ Wir können annehmen, dass **alle** Berechnungen auf einer Eingabe w **dieselbe** Zeit $T = T(|w|)$ in Anspruch nehmen: Lasse M , wenn nötig „auf der Stelle treten“.
 - ▶ Beachte, der Kopf kann in Zeit T nur Zellen mit den Adressen $-T, \dots, T$ erreichen.

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

 $\alpha_w \in \text{KNF-SAT}$.

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

$\Leftrightarrow \alpha_w \in \text{KNF-SAT}$.

Wir müssen mit Hilfe **erfüllender Belegungen** für α_w über **akzeptierende Berechnung** für die Eingabe w „reden“ können.

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

$\Leftrightarrow \alpha_w \in \text{KNF-SAT}$.

Wir müssen mit Hilfe **erfüllender Belegungen** für α_w über **akzeptierende Berechnung** für die Eingabe w „reden“ können.

- Können wir mit Hilfe der Aussagenlogik programmieren?

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

$\Leftrightarrow \alpha_w \in \text{KNF-SAT}$.

Wir müssen mit Hilfe **erfüllender Belegungen** für α_w über **akzeptierende Berechnung** für die Eingabe w „reden“ können.

- ▶ Können wir mit Hilfe der Aussagenlogik programmieren?
- ▶ Was ist durch eine Konjunktion von Klauseln ausdrückbar?

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

$\Leftrightarrow \alpha_w \in \text{KNF-SAT}$.

Wir müssen mit Hilfe **erfüllender Belegungen** für α_w über **akzeptierende Berechnung** für die Eingabe w „reden“ können.

- ▶ Können wir mit Hilfe der Aussagenlogik programmieren?
- ▶ Was ist durch eine Konjunktion von Klauseln ausdrückbar?
 - ★ Die Implikation „ $a \rightarrow b$ “ entspricht der Klausel $b \vee \neg a$.
 - ★ Die konjunktive Normalform erlaubt die Forderung, dass alle Klauseln gleichzeitig erfüllt werden müssen.

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$w \in L \Leftrightarrow$ Es gibt eine Berechnung von M , die w akzeptiert

$\Leftrightarrow \alpha_w \in \text{KNF-SAT}$.

Wir müssen mit Hilfe **erfüllender Belegungen** für α_w über **akzeptierende Berechnung** für die Eingabe w „reden“ können.

- ▶ Können wir mit Hilfe der Aussagenlogik programmieren?
- ▶ Was ist durch eine Konjunktion von Klauseln ausdrückbar?
 - ★ Die Implikation „ $a \rightarrow b$ “ entspricht der Klausel $b \vee \neg a$.
 - ★ Die konjunktive Normalform erlaubt die Forderung, dass alle Klauseln gleichzeitig erfüllt werden müssen.

Wir benutzen aussagenlogische Variablen, um über Zustände, Bandinhalte und Kopfpositionen reden zu können.

- Die Variablen

Zustand $_t(i)$, $0 \leq t \leq T$, $i \in \{0, \dots, q\}$

sollen auszudrücken, dass M zum Zeitpunkt t im Zustand i ist.

- Die Variablen

$$\text{Zustand}_t(i), 0 \leq t \leq T, i \in \{0, \dots, q\}$$

sollen auszudrücken, dass M zum Zeitpunkt t im Zustand i ist.

- Die Variablen

$$\text{Kopf}_t(w_0), 0 \leq t \leq T, -T \leq w_0 \leq T$$

soll genau dann wahr sein, wenn sich der Kopf von M zum Zeitpunkt t auf der Zelle mit Adresse w_0 befindet.

- Die Variablen

$$\text{Zustand}_t(i), 0 \leq t \leq T, i \in \{0, \dots, q\}$$

sollen auszudrücken, dass M zum Zeitpunkt t im Zustand i ist.

- Die Variablen

$$\text{Kopf}_t(w_0), 0 \leq t \leq T, -T \leq w_0 \leq T$$

soll genau dann wahr sein, wenn sich der Kopf von M zum Zeitpunkt t auf der Zelle mit Adresse w_0 befindet.

- Die Variablen

$$\text{Zelle}_t(w_0, w_{as}), 0 \leq t \leq T, -T \leq w_0 \leq T, w_{as} \in \Gamma$$

sollen auszudrücken, dass die Zelle mit Adresse w_0 zum Zeitpunkt t das Symbol w_{as} speichert.

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet,

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest,

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

$$\alpha_0 \equiv \text{Zustand}_0(0) \wedge$$

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

$$\alpha_0 \equiv \text{Zustand}_0(0) \wedge \text{Kopf}_0(1) \wedge$$

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

$$\alpha_0 \equiv \text{Zustand}_0(0) \wedge \text{Kopf}_0(1) \wedge \bigwedge_{w_0=1}^{|w|} \text{Zelle}_0(w_0, w_{w_0}) \wedge$$

Wie programmieren wir mit der Aussagenlogik?

Das Ziel: Erfüllende Belegungen von α_w entsprechen akzeptierenden Berechnungen.

- α_w wird die triviale Klausel $\text{Zustand}_T(1)$ besitzen:
 - ▶ α_w ist nur dann wahr, wenn $\text{Zustand}_T(1)$ wahr ist.
 - ▶ Die modellierte Berechnung endet in dem einzigen akzeptierenden Zustand!
- **Kodierung der Startkonfiguration:** Drücke aus, dass sich M im Zustand 0 befindet, ihr Kopf die Zelle 1 liest, die Zellen $1, \dots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

$$\alpha_0 \equiv \text{Zustand}_0(0) \wedge \text{Kopf}_0(1) \wedge \bigwedge_{w_0=1}^{|w|} \text{Zelle}_0(w_0, w_{w_0}) \wedge \bigwedge_{w_0 \in \{-T, \dots, T\} \setminus \{1, \dots, |w|\}} \text{Zelle}_0(w_0, B).$$

Wir müssen aber noch ausdrücken, dass M

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Allgemein müssen wir ausdrücken, dass genau eine der Variablen y_1, \dots, y_s wahr ist:

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Allgemein müssen wir ausdrücken, dass genau eine der Variablen y_1, \dots, y_s wahr ist: Wir benutzen die **Exklusivitätsformel**

$$\gamma_s(y_1, \dots, y_s) \equiv$$

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Allgemein müssen wir ausdrücken, dass genau eine der Variablen y_1, \dots, y_s wahr ist: Wir benutzen die **Exklusivitätsformel**

$$\gamma_s(y_1, \dots, y_s) \equiv (y_1 \vee \dots \vee y_s) \wedge$$

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Allgemein müssen wir ausdrücken, dass genau eine der Variablen y_1, \dots, y_s wahr ist: Wir benutzen die **Exklusivitätsformel**

$$\gamma_s(y_1, \dots, y_s) \equiv (y_1 \vee \dots \vee y_s) \wedge \bigwedge_{1 \leq i < j \leq s} (\neg y_i \vee \neg y_j)$$

Wir müssen aber noch ausdrücken, dass M

- sich in genau einem Zustand befindet,
- ihr Kopf genau eine Zelle liest und
- jede Zelle genau ein Symbol speichert.

Allgemein müssen wir ausdrücken, dass genau eine der Variablen y_1, \dots, y_s wahr ist: Wir benutzen die **Exklusivitätsformel**

$$\gamma_s(y_1, \dots, y_s) \equiv (y_1 \vee \dots \vee y_s) \wedge \bigwedge_{1 \leq i < j \leq s} (\neg y_i \vee \neg y_j)$$

γ_s ist eine KNF-Formel.

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\gamma \equiv \bigwedge_{t=0}^T$$

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\gamma \equiv \bigwedge_{t=0}^T \gamma_{q+1}(\text{Zustand}_t(0), \dots, \text{Zustand}_t(q))$$

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\gamma \equiv \bigwedge_{t=0}^T \gamma_{q+1}(\text{Zustand}_t(0), \dots, \text{Zustand}_t(q))$$
$$\wedge \bigwedge_{t=0}^T$$

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\gamma \equiv \bigwedge_{t=0}^T \gamma_{q+1}(\text{Zustand}_t(0), \dots, \text{Zustand}_t(q)) \\ \wedge \bigwedge_{t=0}^T \gamma_{2T+1}(\text{Kopf}_t(-T), \dots, \text{Kopf}_t(T))$$

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\begin{aligned} \gamma \equiv & \bigwedge_{t=0}^T \gamma_{q+1}(\text{Zustand}_t(0), \dots, \text{Zustand}_t(q)) \\ & \wedge \bigwedge_{t=0}^T \gamma_{2T+1}(\text{Kopf}_t(-T), \dots, \text{Kopf}_t(T)) \\ & \wedge \bigwedge_{t=0}^T \bigwedge_{w \circ = -T}^T \end{aligned}$$

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

$$\begin{aligned} \gamma \equiv & \bigwedge_{t=0}^T \gamma_{q+1}(\text{Zustand}_t(0), \dots, \text{Zustand}_t(q)) \\ & \wedge \bigwedge_{t=0}^T \gamma_{2T+1}(\text{Kopf}_t(-T), \dots, \text{Kopf}_t(T)) \\ & \wedge \bigwedge_{t=0}^T \bigwedge_{w_0=-T}^T \gamma_3(\text{Zelle}_t(w_0, 0), \text{Zelle}_t(w_0, 1), \text{Zelle}_t(w_0, B)). \end{aligned}$$

Beschreibung eines einzigen Schritts von M

- Angenommen, wir haben erreicht, dass eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t$$

einer Berechnung von M der Länge t entspricht.

Beschreibung eines einzigen Schritts von M

- Angenommen, wir haben erreicht, dass eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t$$

einer Berechnung von M **der Länge t** entspricht.

- Wie muß α_{t+1} konstruiert werden, damit eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t \wedge \alpha_{t+1}$$

einer Berechnungen von M **der Länge $t + 1$** entspricht?

Beschreibung eines einzigen Schritts von M

- Angenommen, wir haben erreicht, dass eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t$$

einer Berechnung von M **der Länge t** entspricht.

- Wie muß α_{t+1} konstruiert werden, damit eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t \wedge \alpha_{t+1}$$

einer Berechnungen von M **der Länge $t + 1$** entspricht?

α_{t+1} muß ausdrücken, dass

Beschreibung eines einzigen Schritts von M

- Angenommen, wir haben erreicht, dass eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t$$

einer Berechnung von M **der Länge t** entspricht.

- Wie muß α_{t+1} konstruiert werden, damit eine erfüllende Belegung von

$$\gamma \wedge \alpha_0 \wedge \alpha_1 \wedge \cdots \wedge \alpha_t \wedge \alpha_{t+1}$$

einer Berechnungen von M **der Länge $t + 1$** entspricht?

α_{t+1} muß ausdrücken, dass sich Zustand, Kopfposition und Zelleninhalt gemäß dem Programm δ und der Konfiguration zum Zeitpunkt t ändern.

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$Zelle_t(w_0, was) \wedge$$

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$\text{Zelle}_t(w_0, was) \wedge \neg \text{Kopf}_t(w_0)$$

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$\text{Zelle}_t(w_0, was) \wedge \neg \text{Kopf}_t(w_0) \rightarrow \text{Zelle}_{t+1}(w_0, was) \quad \leftarrow$$

in die zu konstruierende Formel α_{t+1} auf.

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$\text{Zelle}_t(w_0, was) \wedge \neg \text{Kopf}_t(w_0) \rightarrow \text{Zelle}_{t+1}(w_0, was) \quad \leftarrow$$

in die zu konstruierende Formel α_{t+1} auf.

- Wenn eine Zelle vom Kopf besucht wird, dann konsultiere das Programm δ .

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$\text{Zelle}_t(w_0, was) \wedge \neg \text{Kopf}_t(w_0) \rightarrow \text{Zelle}_{t+1}(w_0, was) \quad \leftarrow$$

in die zu konstruierende Formel α_{t+1} auf.

- Wenn eine Zelle vom Kopf besucht wird, dann konsultiere das Programm δ .
 - ▶ Wenn es B Befehle in δ gibt, dann führe die B Befehlsvariablen

$$\text{Befehl}_t(i) \text{ für } i = 1, \dots, B$$

ein, und fordere, dass zu jedem Zeitpunkt t genau ein Befehl i ausgeführt werden darf.

Änderung der Zellinhalte

- Eine vom Kopf **nicht** besuchte Zelle bleibt unverändert:
Für jede Position $w_0 \in \{-T, \dots, T\}$ und jedes Symbol $was \in \{0, 1, B\}$ nehmen wir deshalb die Klausel

$$\text{Zelle}_t(w_0, was) \wedge \neg \text{Kopf}_t(w_0) \rightarrow \text{Zelle}_{t+1}(w_0, was) \quad \leftarrow$$

in die zu konstruierende Formel α_{t+1} auf.

- Wenn eine Zelle vom Kopf besucht wird, dann konsultiere das Programm δ .
 - ▶ Wenn es B Befehle in δ gibt, dann führe die B Befehlsvariablen

$$\text{Befehl}_t(i) \text{ für } i = 1, \dots, B$$

ein, und fordere, dass zu jedem Zeitpunkt t genau ein Befehl i ausgeführt werden darf.

- ▶ Verändere den Zellinhalt, den neuen Zustand und die Kopfposition entsprechend dem auszuführenden Befehl.

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

$$\begin{aligned} & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zustand}_{t+1} \end{aligned}$$

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

$$\begin{aligned} & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zustand}_{t+1}(q') \end{aligned}$$

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

$$\begin{aligned} & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zustand}_{t+1}(q') \\ & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zelle}_{t+1}(\text{wo}, \end{aligned}$$

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

$$\begin{aligned} & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zustand}_{t+1}(q') \\ & \text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i) \\ \rightarrow & \text{Zelle}_{t+1}(\text{wo}, \text{was}') \end{aligned}$$

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}


- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- $\text{Zustand}_{t+1}(q')$
- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- $\text{Zelle}_{t+1}(\text{wo}, \text{was}')$
- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- Kopf_{t+1}

Wenn die Zelle vom Kopf besucht wird, ...

Der i te Befehl habe die Form

$$(q, \text{was}) \rightarrow (q', \text{was}', \text{Richtung}).$$

Es sei zum Beispiel $\text{Richtung} = \text{links}$. Dann erscheinen die folgenden Klauseln in α_{t+1}

- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- $\text{Zustand}_{t+1}(q')$
- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- $\text{Zelle}_{t+1}(\text{wo}, \text{was}')$
- $\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \wedge \text{Befehl}_{t+1}(i)$
- $\text{Kopf}_{t+1}(\text{wo} - 1)$. 

- Für jedes Paar (q, a) von Zustand q und Bandsymbol a sei

$B(q, a) = \{i \mid \text{Befehl } i \text{ ist im Zustand } q \text{ ausführbar, wenn Buchstabe } a \text{ gelesen wird}\}.$

- Für jedes Paar (q, a) von Zustand q und Bandsymbol a sei

$$B(q, a) = \{i \mid \text{Befehl } i \text{ ist im Zustand } q \text{ ausführbar, wenn Buchstabe } a \text{ gelesen wird}\}.$$

- M wählt einen ausführbaren Befehl nichtdeterministisch:

$$\text{Zelle}_t(\text{wo, was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \rightarrow$$

Auswahl des Befehls

- Für jedes Paar (q, a) von Zustand q und Bandsymbol a sei

$$B(q, a) = \{i \mid \text{Befehl } i \text{ ist im Zustand } q \text{ ausführbar, wenn Buchstabe } a \text{ gelesen wird}\}.$$

- M wählt einen ausführbaren Befehl nichtdeterministisch:

$$\text{Zelle}_t(\text{wo, was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \rightarrow \bigvee_{i \in B(q, \text{was})} \text{Befehl}_{t+1}(i).$$

Auswahl des Befehls

- Für jedes Paar (q, a) von Zustand q und Bandsymbol a sei

$$B(q, a) = \{i \mid \text{Befehl } i \text{ ist im Zustand } q \text{ ausführbar, wenn Buchstabe } a \text{ gelesen wird}\}.$$


- M wählt einen ausführbaren Befehl nichtdeterministisch:



$$\text{Zelle}_t(\text{wo}, \text{was}) \wedge \text{Kopf}_t(\text{wo}) \wedge \text{Zustand}_t(q) \rightarrow \bigvee_{i \in B(q, \text{was})} \text{Befehl}_{t+1}(i).$$




- Es ist genau ein Befehl auszuführen und wir erreichen dies durch




$$\gamma_B(\text{Befehl}_{t+1}(0), \text{Befehl}_{t+1}(1), \dots, \text{Befehl}_{t+1}(B)).$$

- Zuerst zur Definition von α_{t+1} :




- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben 

- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .




- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .

- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:




Definition von α_w

- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:




$$\alpha_w = \gamma \quad \img alt="blue play button icon" data-bbox="292 522 332 562"/>$$



- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:




$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{ $$



- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:

$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{  } \wedge \alpha_1 \wedge \cdots \wedge \alpha_T \wedge$$

- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:




$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{  } \wedge \alpha_1 \wedge \cdots \wedge \alpha_T \wedge \text{Zustand}_T(1).$$



- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:

$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{  } \wedge \alpha_1 \wedge \cdots \wedge \alpha_T \wedge \text{Zustand}_T(1).$$

- Die wesentlichen Eigenschaften von α_w :
 - ▶ Eine erfüllende Belegung entspricht einer akzeptierenden Berechnung und umgekehrt.




Definition von α_w



- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:

$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{  } \wedge \alpha_1 \wedge \cdots \wedge \alpha_T \wedge \text{Zustand}_T(1).$$

- Die wesentlichen Eigenschaften von α_w :
 - ▶ Eine erfüllende Belegung entspricht einer akzeptierenden Berechnung und umgekehrt.
 - ▶ Für die Konstruktion von α_w müssen wir nur die Eingabe w und das Programm der Turingmaschine kennen. Die Konstruktionsvorschrift ist einfach.

Definition von α_w

- Zuerst zur Definition von α_{t+1} :
Wir nehmen alle Klauseln auf,
 - ▶ die Zellinhalte nicht besuchter Zellen beschreiben  sowie
 - ▶ die Klauseln, die den neuen Zustand, die neue Kopfposition und den neuen Zellinhalt der besuchten Zelle beschreiben .
 - ▶ Gleiches gilt für die Klauseln, die den neuen Befehl auswählen. .
- Die Definition von α_w ist jetzt abgeschlossen:

$$\alpha_w = \gamma \text{  } \wedge \alpha_0 \text{  } \wedge \alpha_1 \wedge \cdots \wedge \alpha_T \wedge \text{Zustand}_T(1).$$

- Die wesentlichen Eigenschaften von α_w :
 - ▶ Eine erfüllende Belegung entspricht einer akzeptierenden Berechnung und umgekehrt.
 - ▶ Für die Konstruktion von α_w müssen wir nur die Eingabe w und das Programm der Turingmaschine kennen. Die Konstruktionsvorschrift ist einfach.
 - ▶ α_w kann in polynomieller Zeit (in $|w|$) berechnet werden!

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.
- Warum ist KNF-SAT schwierig?

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.
- Warum ist KNF-SAT schwierig?
 - ▶ Gerade weil wir Aussagen über nichtdeterministische Berechnungen machen können.

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.
- Warum ist KNF-SAT schwierig?
 - ▶ Gerade weil wir Aussagen über nichtdeterministische Berechnungen machen können.
- Warum haben wir mit Turingmaschinen gearbeitet?

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.
- Warum ist KNF-SAT schwierig?
 - ▶ Gerade weil wir Aussagen über nichtdeterministische Berechnungen machen können.
- Warum haben wir mit Turingmaschinen gearbeitet?
 - ▶ Um den Satz von Cook-Levin mit einem nicht zu komplizierten Argument führen zu können.

- Die Aussagenlogik kann über nichtdeterministische Berechnungen reden.
- Warum ist KNF-SAT schwierig?
 - ▶ Gerade weil wir Aussagen über nichtdeterministische Berechnungen machen können.
- Warum haben wir mit Turingmaschinen gearbeitet?
 - ▶ Um den Satz von Cook-Levin mit einem nicht zu komplizierten Argument führen zu können.
 - ▶ Aber wir wissen, dass die Aussagenlogik ebenfalls Aussagen über nichtdeterministische Berechnungen von Supercomputern treffen kann.