Konjunktive Normalform

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

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

- Die *Klauseln* $k_1, \ldots, k_r$ 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:

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

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

Der Satz von Cook

$KNF - SAT$ ist NP-vollständig.

- Jetzt treten wir die Lawine los:

  Wenn $L_1 \leqslant_p L_2$ 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 später.

3 — *SAT* ist NP-vollständig

$3-SAT$ ist NP-vollständig (1/3)

- 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 verifiere, dass jede Klausel erfüllt ist.

Zeige die Reduktion $KNF-SAT \leqslant_p 3-SAT$.

- Die KNF-Formel

$$w = k_1 \wedge \cdots \wedge k_r$$

sei Eingabe für $KNF - SAT$.

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

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

- 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 $l_1, l_2, l_3, l_4, l_5 \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$.

- Wir erfinden neue Variablen $x_{i,1}, x_{i,2}$ 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:**

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

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

2 – *SAT* ist effizient lösbar

$2 - SAT$

- Wie schwierig ist $2 - SAT$?
- Sei $w = k_1 \wedge \cdots \wedge k_r$ eine 2-KNF Formel.
  - Setze $x_1 =$ 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 $x_1 =$ falsch, ansonsten fahre mit x_2 fort.
- $2 - SAT \in \mathtt{P}$, aber $3 - SAT$ ist $\mathtt{NP}$-vollständig.

CLIQUE ist NP-vollständig

CLIQUE ist NP-vollständig

Satz von Karp

$3 - SAT \leqslant_p CLIQUE$ 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 $w = k_1 \wedge \cdots \wedge k_r$ mit den Klauseln $k_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ sei Eingabe für $3 - SAT$.
  - Der ungerichteten Graph $G_w = (V_w, E_w)$ ist die (transformierte) Eingabe für *CLIQUE*.
  - Für jede Klausel k_i erfinden wir eine Menge $V_i = \{(i, l_{i,1}), (i, l_{i,2}), (i, l_{i,2})\}$ von drei Knoten mit $V_w = \bigcup_{i=1}^{r} V_i$.
  - Knoten in V_i 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 Cliquen von G_w (1/2)

- 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 $w \in 3 - SAT$, 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

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

  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.

Die Cliquen von G_w (2/2)

- Wenn G_w eine Clique C der Größe r besitzt, dann ist $w \in 3 - SAT$. 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:

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

und wir definieren die effizient berechenbare Transformation

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

Zusammenfassung

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

> Erfüllende Belegungen und Cliquen der Größe r entsprechen sich wechselseitig.

Independent Set ist NP-vollständig

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.
- $IS \in \mathtt{NP}$:
  - 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

$$CLIQUE \leqslant_p IS.$$

Independent Set ist NP-vollständig

CLIQUE $\leqslant_p$ *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.
- $M(w)$ ist effizient berechenbar.
- Funktioniert die Transformation?

$(G, k) \in$ *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 $\overline{G}$ verbunden

$\Leftrightarrow$ $(\overline{G}, k) \in$ *IS*.

Set Cover ist NP-vollständig

Set Cover

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

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

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

Vertex Cover ist NP-vollständig

Vertex Cover

- 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.

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

$VC \in \mathtt{NP}$:

- 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

$VC \leqslant_p SC$.

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

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

- $(G, k) \in VC$

$$\begin{aligned} &\Leftrightarrow \quad G \text{ hat eine Überdeckung } \ddot{U} \text{ der Größe } k \\ &\Leftrightarrow \quad \text{Die Mengen } A_v \text{ für } v \in \ddot{U} \text{ überdecken alle Kanten} \\ &\Leftrightarrow \quad (A_1, \ldots, A_n; k) \in SC \end{aligned}$$

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

VC ist schwierig

$IS \leqslant_p VC$.

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

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

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

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

Zusammenfassung

- 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: $KNF - SAT \leqslant_p 3 - SAT$, $3 - Sat \leqslant_p CLIQUE$, $CLIQUE \leqslant_p IS$, $IS \leqslant VC$, $VC \leqslant SC$.

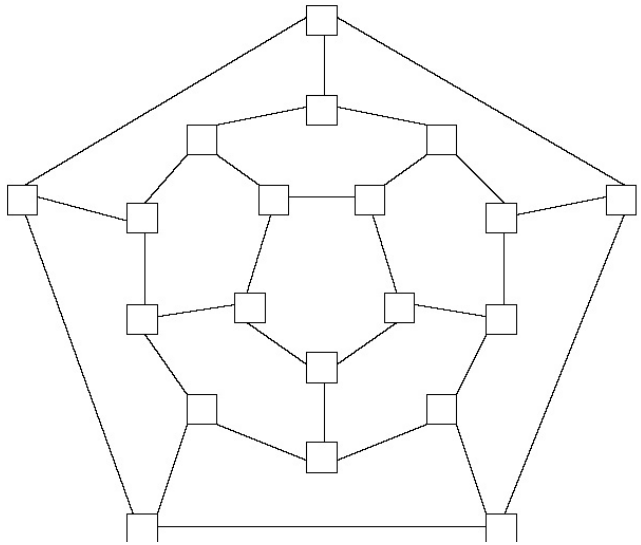
Die Lawine gewinnt an Fahrt

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

Schwierige Wegeprobleme in Graphen

Wegeprobleme (1/2)

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.



Hat dieser Graph einen Hamiltonschen Kreis?
Hat der n-dimensionale Würfel einen Hamiltonschen Kreis?

Wegeprobleme (2/2)

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

$$DHC \leqslant_p HC.$$

DHC und *HC*

- 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 $u \to v \in E$ 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.

Der ungerichtete Graph

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

$$G \in DHC \Rightarrow G' \in HC.$$

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

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

$$G' \in HC \rightarrow G \in DHC.$$

$$HC \leqslant_p TSP.$$

Das Traveling Salesman Problem

- **Die Transformation:** Für die Eingabe $G = (\{1, \ldots, n\}, E)$ 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

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

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

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

  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.

$$HC \leqslant_p LW.$$

Das längste Wege Problem *LW*

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?

- 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 Hamiltionschen Kreis hat, dann hat G'' einen Weg $0 - 1 - \cdots - 1' - 0'$ 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 $M(G) = (G'', n + 2)$.

Was haben wir erreicht?

Wir haben die folgenden Reduktionen nachgewiesen:

- $DHC \leqslant_p HC$,
- $HC \leqslant_p TSP$ und
- $HC \leqslant_p LW$.
- 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: *KNF* − *SAT* ist NP-vollständig

Der Fahrplan

- Wir zeigen den Satz von Cook:

  $KNF - SAT$ ist $\mathtt{NP}$-vollständig.

- Dann treten wir die Lawine los:

  Wenn $L_1 \leqslant_p L_2$ und wenn L_1 ein $\mathtt{NP}$-hartes Problem ist, dann ist auch L_2 ein $\mathtt{NP}$-hartes Problem.

Der Satz von Cook

$KNF - SAT$ ist $\mathtt{NP}$-vollständig. (Web)

- Für ein beliebiges Problem $L \in \mathtt{NP}$ müssen wir die Reduktion $L \leqslant_p KNF - SAT$ 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 $\Gamma = \{0, 1, \mathtt{B}\}$, die Zustandsmenge $\{0, \ldots, q\}$, 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 $-T, \ldots, T$ erreichen.

Unsere Aufgabe

Konstruiere in polynomieller Zeit eine Formel α_w für Eingabe w mit

$$w \in L \quad \Leftrightarrow \quad \text{Es gibt eine Berechnung von } M, \text{ die } w \text{ akzeptiert}$$
$$\overset{!}{\Leftrightarrow} \quad \alpha_w \in \mathit{KNF} - \mathit{SAT}.$$

Wir müssen mit Hilfe erfüllender Belegungen 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 „$a \to b$“ 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 von α_w

- Die Variablen

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

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

- Die Variablen

$$\text{Kopf}_t(\texttt{wo}),\ 0 \leqslant t \leqslant T,\ -T \leqslant \texttt{wo} \leqslant T$$

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

- Die Variablen

$$\text{Zelle}_t(\texttt{wo}, \texttt{was}),\ 0 \leqslant t \leqslant T,\ -T \leqslant \texttt{wo} \leqslant T,\ \texttt{was} \in \Gamma$$

sollen auszudrücken, dass die Zelle mit Adresse `wo` zum Zeitpunkt t das Symbol `was` speichert.

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 $1, \ldots, |w|$ die Eingabe w speichern und die restlichen Zellen das Blanksymbol speichern.

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

Ein kleines Problem

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 $y_1, \ldots, y_s$ wahr ist: Wir benutzen die Exklusivitätsformel

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

γ_s ist eine KNF-Formel.

Eindeutigkeit von Zuständen, Kopfpositionen und Zellinhalten

Eindeutigkeit für alle T Schritte erreichen wir durch:

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

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.

- 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.

Änderung der Zellinhalte

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

  Für jede Position $\mathtt{wo} \in \{-T, \ldots, T\}$ und jedes Symbol $\mathtt{was} \in \{0, 1, \mathtt{B}\}$ nehmen wir deshalb die Klausel

  $$\mathsf{Zelle}_t(\mathtt{wo}, \mathtt{was}) \wedge \neg \mathsf{Kopf}_t(\mathtt{wo}) \rightarrow \mathsf{Zelle}_{t+1}(\mathtt{wo}, \mathtt{was})$$

  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

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

    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

$$(q, \texttt{was}) \to (q', \texttt{was}', \texttt{Richtung}).$$

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

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

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:

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

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

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

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:

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

- 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!

Zusammenfassung

- 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 mit einem nicht zu komplizierten Argument führen zu können.
  - Aber wir wissen, dass die Aussagenlogik ebenfalls Aussagen über nichtdeterministische Berechnungen von Supercomputern treffen kann.