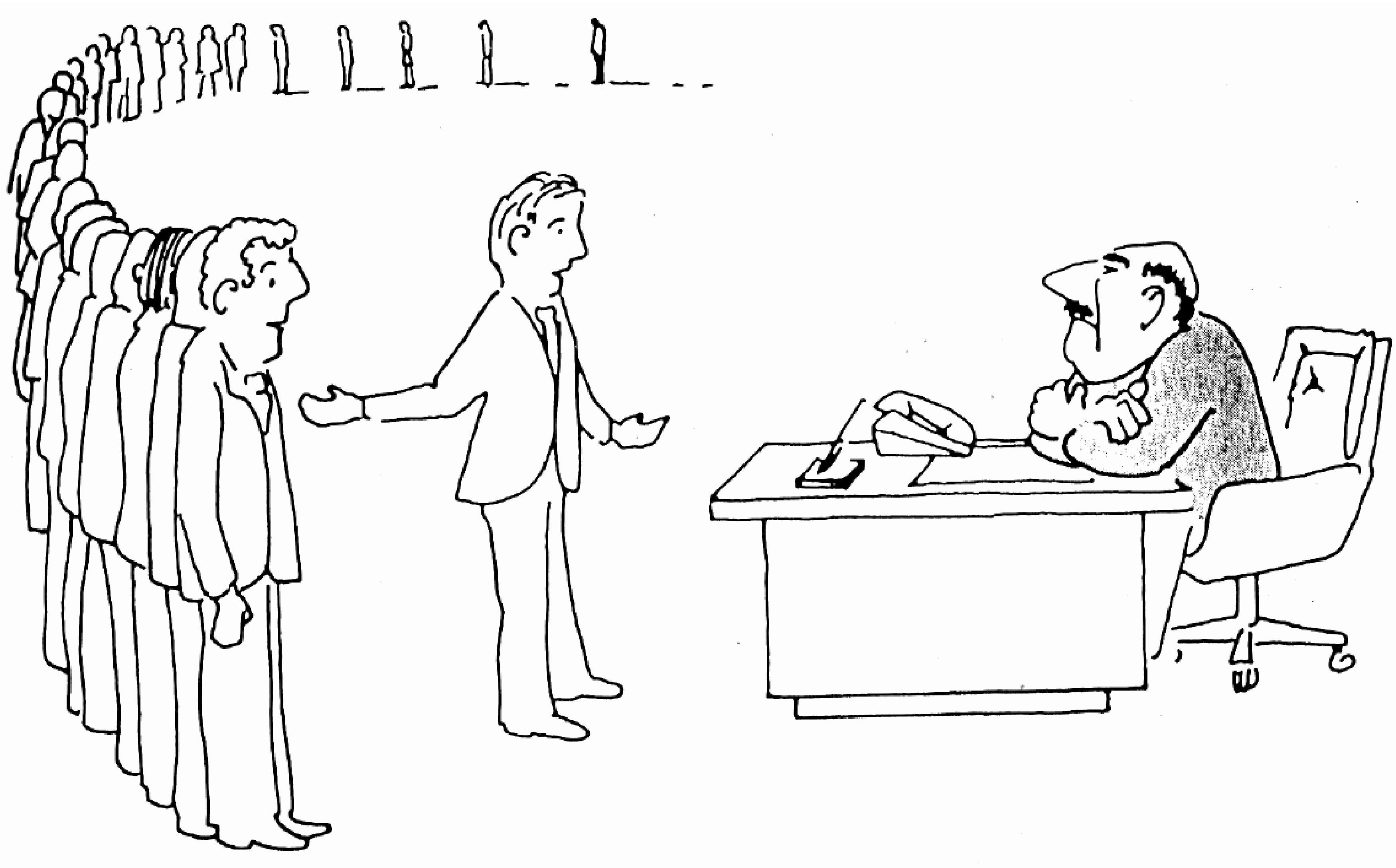
Theoretische Informatik II

Einheit 6.2

Das $\mathcal{P}$–$\mathcal{NP}$ Problem

1. Nichtdeterministische Lösbarkeit
2. Sind $\mathcal{NP}$-Probleme handhabbar?
3. $\mathcal{NP}$-Vollständigkeit
4. Der Satz von Cook

Wenn ein Problem nicht effektiv lösbar zu sein scheint



“I can’t find an efficient algorithm, but neither can all these famous people.”

Vielleicht der einzig mögliche Weg

Welche Art von Problemen betrifft dies?

- **Travelling Salesman (TSP)** (Message Routing)
  Gibt es eine Rundreise zwischen n Städten mit minimalen Kosten B?
- **Cliquen-Problem (CLIQUE)**
  Hat G einen vollständig verbundenen Teilgraphen der Größe k?
- **Erfüllbarkeitsproblem (SAT)**
  Ist eine aussagenlogische Formel in KNF der Größe n erfüllbar?
- **Multiprozessor-Scheduling (MPS)**
  Können n Prozesse derart auf eine Menge von Prozessoren verteilt werden, daß alle in Zeit t abgearbeitet sind?
- **Partitionsproblem (PART)** Können n Zahlen in zwei Partitionen verteilt werden, daß die jeweiligen Summen gleich sind
- **Binpacking (BPP)** Können n verschieden große Gegenstände in maximal k Verpackungsbehältern untergebracht werden?

Keine polynomielle Lösung bekannt
Beste Lösung ist Durchsuchen aller Möglichkeiten

… aber Erfolg der Suche ist leicht zu testen

- **Travelling Salesman:** Für eine gegebene Rundreise $i_1..i_n$ können die Kosten $c_{i_1 i_2} + .. + c_{i_n i_1}$ in linearer Zeit berechnet und mit der Kostenbeschränkung B verglichen werden
- **Cliquen-Problem:** Ein gegebener Teilgraph der Größe k kann in polynomieller Zeit auf Vollständigkeit überprüft werden
- **Erfüllbarkeitsproblem:** Man kann in polynomieller Zeit testen, ob eine gegebene Belegung der Variablen eine Formel erfüllt
- **Multiprozessor-Scheduling**
  Man kann in polynomieller Zeit testen, ob eine gegebene Verteilung von Prozessen ein Ressourcenlimit einhält.
- **Binpacking:** Man kann in polynomieller Zeit testen, ob eine gegebene Verteilung der Gegenstände in k Verpackungsbehälter paßt
- **Zusammengesetztheitstest:** Man kann in quadratischer Zeit testen, ob eine gegebene Zahl Teiler von x (also x keine Primzahl) ist

Beispiel: Das Erfüllbarkeitsproblem

Ist eine aussagenlogische Formel in KNF erfüllbar?

Gegeben m Klauseln $k_1, .., k_m$ über n Variablen $x_1, .., x_n$. Gibt es eine Belegung $a_1, .., a_n \in \{0, 1\}$ der Variablen x_i, welche alle Klauseln erfüllt?

- **Klausel über den Variablen $x_1, ..x_n$**
  – Disjunktion einiger Literale der Form x_i bzw. $\overline{x_i}$

- **Belegung $a_1, ..., a_n \in \{0, 1\}$ erfüllt Klausel k_j**
  – Auswertung von k_j unter $a_1, ..., a_n$ ergibt den Booleschen Wert 1

- $$SAT = \{k_1, ..k_m \mid k_i \text{ Klausel über } x_1, ..x_n \wedge (\exists a_1, ..a_n \in \{0{,}1\}. \forall j \leq m.\ a_1, ..a_n \text{ erfüllt } k_j)\}$$

Codierbar als Teilmenge der Sprache der Aussagenlogik

Erfüllbarkeit von Formeln in KNF

$(\overline{x_1} \vee x_2) \;\wedge\; (x_1 \vee \overline{x_2} \vee \overline{x_3}) \;\wedge\; \overline{x_3}$ erfüllbar

– Setze x_3=0, x_2=1, x_1 beliebig, z.B. x_1=0

– Auswertung: $(\overline{0}+1) * (0+\overline{1}+\overline{0}) * \overline{0} = (1+1) * (0+0+1) * 1 = 1 * 1 * 1 = 1$

$x_1 \;\wedge\; \overline{x_1}$ nicht erfüllbar

– Jede Belegung ergibt den Wert 0

$(x_1 \vee x_2) \;\wedge\; (\overline{x_1} \vee \overline{x_2})$ erfüllbar, Belegung: (1,0)

$(x_1 \vee x_2) \;\wedge\; (x_1 \vee \overline{x_2}) \;\wedge\; (\overline{x_1} \vee x_2) \;\wedge\; (\overline{x_1} \vee \overline{x_2})$ nicht erfüllbar

$(x_1 \vee \overline{x_2} \vee x_3) \;\wedge\; (\overline{x_1} \vee x_2 \vee \overline{x_4}) \;\wedge\; (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

erfüllbar, Belegung: (1,1,0,0)

$(x_1 \vee \overline{x_2}) \;\wedge\; (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \;\wedge\; (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \;\wedge\; (x_1 \vee \overline{x_2} \vee x_3 \vee \overline{x_4})$

erfüllbar, Belegung: (1,1,0,0)

Lösungen für das Erfüllbarkeitsproblem

$SAT = \{k_1..k_m |\ k_i$ Klausel über $x_1..x_n \wedge \exists a_1..a_n \in \{0{,}1\}.a_1..a_n$ erfüllt $k_1..k_m\}$

- **Deterministische Lösung ist exponentiell**
  - Werte Klauseln für alle möglichen Belegungen der Variablen aus bis erfüllende Belegung gefunden ist
  - Es gibt 2^n möglichen Belegungen von $x_1, ..x_n$
  - Auswertung linear in Größe der Formel $\mathcal{O}(m*n)$
  - **Laufzeit ist in $\mathcal{O}(2^n)$**
- **Der Aufwand liegt nur in der Suche**
  - Wenn eine erfüllende Belegung vorgegeben wird kann diese durch Auswertung der Formel in polynomieller Zeit verifiziert werden

Welches Modell kann diesen Effekt beschreiben?

ORAKEL-TURINGMASCHINEN

- **Raten und Verifizieren von Lösungsvorschlägen**

  1. Bei Eingabe von $w \in \Sigma$ generiert Orakel einen Lösungsvorschlag x

     Andere Bezeichnungen: x ist "Zertifikat oder Zeuge für $w \in L$"

  2. Verifizierer(komponente) V überprüft w, x deterministisch

  OTM akzeptiert w, wenn es ein x mit $(w, x) \in L(V)$ gibt

  $$\boldsymbol{L(M) = \{w \mid \exists x.(w, x) \in L(V)\}}$$

- **Berechnungsaufwand einer OTM bei Eingabe w**

  – Maximale Rechenzeit des Verifizierers für Prüfung eines Vorschlags x

  – Für SAT gibt es einen Polynomialzeit-Verifizierer also $\boldsymbol{SAT \in \mathcal{NP}}$

- **OTM Modell ist äquivalent zu NTMs** §4.1

  – NTM M akzeptiert, wenn ein Lösungsweg zum Erfolg führt

  – $t_M(w)$ ist maximale Zahl der Konfigurationsübergänge bis Terminierung

  Es folgt **$L \in \mathcal{NP}$ gdw. L ist polynomiell verifizierbar**

Sind $\mathcal{NP}$ Probleme effizient lösbar?

- **Gilt $\mathcal{P}$=$\mathcal{NP}$ oder $\mathcal{P}\neq\mathcal{NP}$?**
  - Eines der wichtigsten offenen Probleme der TI
  - Seit mehr als 40 Jahren ungeklärt, möglicherweise unlösbar
- **Mehr als 1000 algorithmische Probleme betroffen**
  - Suchprobleme (Travelling Salesman, . . .)
  - Reihenfolgenprobleme (Scheduling, Binpacking, . . .)
  - Graphenprobleme (Clique, Vertex cover, . . .) $\mapsto$ Operations Research
  - Logische Probleme (Erfüllbarkeit,. . .) $\mapsto$ Model Checking, Hardwareverifikation
  - Zahlenprobleme (Primfaktorisierung, . . .) $\mapsto$ Kryptographie, IT Sicherheit
- **Indizien sprechen gegen $\mathcal{P}$=$\mathcal{NP}$**
  - Zu viele $\mathcal{NP}$-Probleme ohne bekannte polynomielle Lösung
  - Über 1000 äquivalente Probleme in ‘schwerster Teilklasse’ von $\mathcal{NP}$

WIE ANALYSIERT MAN "$\mathcal{P}=\mathcal{NP}$ ODER $\mathcal{P}\neq\mathcal{NP}$"?

- **Untersuche die "schwierigsten" $\mathcal{NP}$-Probleme**
  - Kann man eines davon effizient lösen?
  - Wenn ja, dann gilt $\mathcal{P}=\mathcal{NP}$
  - Wenn nein, dann gibt es ein Beispiel für $\mathcal{P}\neq\mathcal{NP}$
- **Was heißt "L ist schwierigstes $\mathcal{NP}$-Problem"?**
  - Jedes andere $\mathcal{NP}$-Problem L' ist nicht schwerer als L
  - Lösungen für L könnten in Lösungen für L' umgewandelt werden
  - Transformation der Lösungen muß effizient sein
  - Entspricht funktionaler Reduzierbarkeit mit Laufzeitbedingungen
- **Formales Konzept: Polynomielle Reduzierbarkeit**
  - $\boldsymbol{L'\leq_p L}$ ("L' polynomiell reduzierbar auf L"), falls $L'=f^{-1}(L)$ für eine totale, in polynomieller Zeit berechenbare Funktion f

    f transformiert Eingaben $x\in L'$ in $f(x)\in L$, aber das Lösungsverfahren für L rückwärts(!) auf L'

Polynomielle Reduzierbarkeit: $SAT \leq_p 3SAT$

Reduziere SAT auf normierte Form des Erfüllbarkeitsproblems

$3SAT = \{k_1,..k_m \mid k_i{=}z_{i1} \vee z_{i2} \vee z_{i3}$ *mit* $z_{ij} \in \{x_1, \overline{x_1}, ..x_n, \overline{x_n}\}$
$\wedge\ \exists a_1,..a_n \in \{0,1\}.\ \forall j{\leq}m.\ a_1,..a_n$ *erfüllt* $k_j\ \}$

- **Definiere Reduktionsfunktion f auf Formeln**
  - f normalisiert die Klauseln $k_1,..k_m$, indem jede Klausel k_i (einzeln) durch eine äquivalente Menge von Dreierklauseln ersetzt wird
  - Ersetze einelementige Klauseln $k_i = z$ durch $z \vee z \vee z$
  - Ersetze zweielementige Klauseln $k_i = z \vee z'$ durch $z \vee z \vee z'$
  - Übernehme dreielementige Klauseln unverändert
  - Ersetze Klauseln $k_i = z_1 \vee z_2 \vee .. \vee z_j$ durch j-2 neue Klauseln mit neuen Variablen $y_{i,l}$: $(z_1 \vee z_2 \vee y_{i,1}) \wedge (\overline{y_{i,1}} \vee z_3 \vee y_{i,2}) \wedge .. \wedge (\overline{y_{i,j-3}} \vee z_{j-1} \vee z_j)$
- **f ist in polynomieller Zeit berechenbar**
  - Der Aufwand ist linear, da im schlimmsten Fall eine Klausel mit j Literalen durch j-2 Dreierklauseln ersetzt wird
- **Es gilt $\forall F.\ F \in SAT \Leftrightarrow f(F) \in 3SAT$**
  - Jede Klausel k_i ist erfüllbar gdw. die erzeugte Klauselmenge erfüllbar ist

$\mathcal{NP}$-Vollständigkeit

- **Reduzierbarkeit bedeutet geringere Komplexität**
  - $L \leq_p L' \land L' \in \mathcal{P} \quad \Rightarrow \quad L \in \mathcal{P}$
  - $L \leq_p L' \land L' \in \mathcal{NP} \quad \Rightarrow \quad L \in \mathcal{NP}$

  Beweis analog zu allgemeiner Reduzierbarkeit:
  - $\chi_L(x)=1 \Leftrightarrow x \in L \Leftrightarrow f(x) \in L' \Leftrightarrow \chi_{L'}(f(x))=1 \Leftrightarrow (\chi_{L'} \circ f)(x)=1$
  - $\chi_{L'} \circ f$ ist in polynomieller Zeit berechenbar, wenn dies für $\chi_{L'}$ gilt

- **$\mathcal{NP}$-hart (auch $\mathcal{NP}$-schwer): nicht leichter als $\mathcal{NP}$**
  - **L' ist $\mathcal{NP}$-hart** genau dann wenn $L \leq_p L'$ für alle $L \in \mathcal{NP}$ gilt

- **$\mathcal{NP}$-vollständig: schwierigste Teilklasse in $\mathcal{NP}$**
  - **L' ist $\mathcal{NP}$-vollständig**, wenn L' $\mathcal{NP}$-hart und $L' \in \mathcal{NP}$
  - Schreibweise: $L \in \mathcal{NPC}$

Konsequenzen von $\mathcal{NP}$-Vollständigkeit

- **Alle $\mathcal{NP}$-vollständigen Probleme sind äquivalent**
  - $L, L' \in \mathcal{NPC} \Rightarrow L' \leq_p L \wedge L \leq_p L'$
- **$\mathcal{NP}$-vollständige Probleme entscheiden '$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$'**
  - $\mathcal{P}{=}\mathcal{NP} \Leftrightarrow \exists L \in \mathcal{NPC}.\, L \in \mathcal{P} \Leftrightarrow \forall L \in \mathcal{NPC}.\, L \in \mathcal{P}$ HMU Satz 10.5

    Ist $\mathcal{P}{=}\mathcal{NP}$ dann sind alle $\mathcal{NP}$-vollständigen Probleme in $\mathcal{P}$
  - $\mathcal{P}{\neq}\mathcal{NP} \Leftrightarrow \exists L \in \mathcal{NPC}.\, L \notin \mathcal{P} \Leftrightarrow \forall L \in \mathcal{NPC}.\, L \notin \mathcal{P}$

    Ist $\mathcal{P}{\neq}\mathcal{NP}$ dann sind alle $\mathcal{NP}$-vollständigen Probleme nicht in $\mathcal{P}$
- **$\mathcal{NP}$-Vollständigkeit ist leicht nachweisbar, wenn ein $\mathcal{NP}$-vollständiges Problem bekannt ist**
  - $L \in \mathcal{NPC} \Leftrightarrow L \in \mathcal{NP} \wedge \exists L' \in \mathcal{NPC}.\, L' \leq_p L$ HMU Satz 10.4
  - $L \in \mathcal{NPC} \Leftrightarrow \exists L' \in \mathcal{NPC}.\, L' \leq_p L \wedge L \leq_p L'$

$\mathcal{NP}$-Vollständigkeit muß einmal explizit gezeigt werden

Wie zeigt man $\mathcal{NP}$-Vollständigkeit?

Beweise $\mathcal{NP}$-Vollständigkeit explizit für eine Sprache L

- **Codiere Berechnungen beliebiger NTMs in L**
  – Codierung soll zu Sprache L gehören, wenn Maschine M akzeptiert
  – Codierung soll nicht zu L gehören, wenn M nicht akzeptiert
  – Codierung 'polynomieller NTMs' muß in polynomieller Zeit geschehen

  Damit ist $L(M) \leq_p L$ für jede polynomielle NTM M, d.h. L ist $\mathcal{NP}$-hart
- **Sprache L muß selbst in $\mathcal{NP}$ liegen**
  – Ergibt zusammen mit dem obigen die $\mathcal{NP}$-Vollständigkeit von L
- **Welche Sprache ist ausdrucksstark genug?**
  – Idee: codiere mögliche Zustandsübergänge durch logische Formeln
  – Problemstellung: Können Zustandsübergänge so kombiniert werden, daß eine terminierende Berechnung codiert wird?
  – Erfüllbarkeitsproblem der (Aussagen-)logik ist Kandidat für $\mathcal{NPC}$

SAT IST $\mathcal{NP}$-VOLLSTÄNDIG (SATZ VON COOK)

Zeige $L \leq_p SAT$ für jede Sprache $L \in \mathcal{NP}$

- **Gegeben: NTM M für L, die in polynomieller Zeit terminiert**
- **Ziel: Codiere Berechnung von M bei Eingabe w durch KNF-Formel, die erfüllbar ist, g.d.w. $w \in L$**
  - Codierung muß in polynomieller Zeit (relativ zu $|w|$) berechenbar sein
  - Codierung darf von Kenntnissen über L und M abhängen
- **Vorgehen: Beschreibe mögliche Konfigurationsübergänge von M durch aussagenlogische Klauseln**
  - Codiere Zustand, Kopfposition und Bandzellen durch Literale
  - Es werden nur polynomiell viele Literale und Klauseln benötigt
  - Formel ist erfüllbar, wenn Konfigurationsübergänge zu akzeptierender Berechnung zusammengesetzt werden können

Aufwendiger Beweis mit sehr vielen Details

Satz von Cook: Grundannahmen

- **L wird von NTM M akzeptiert**
  - M = (Q, Σ, Γ, δ, q_0, B, F) mit Q=$\{q_0, .., q_k\}$, Γ=$\{X_1, .., X_m\}$
- **M zeitbeschränkt durch ein Polynom $p(n)$**
  - $t_M(w) \leq p(n)$ für jedes Wort $w \in \Sigma^*$ mit $|w|$=n
  - Es sind genau $p(n)$ Berechnungsschritte als Formel zu codieren

    o.B.d.A.: M 'verharrt' in den Endzuständen anstatt abzubrechen

    d.h. $(u,q,v) \vdash (u,q,v)$ für $q \in F$
- **M ist auch platzbeschränkt durch $p(n)$**
  - M kann während der Berechnung maximal $p(n)$ Bandzellen aufsuchen

    o.B.d.A.: M arbeitet mit halbseitig unendlichem Band

Es reicht, genau die Bandzellen 1 . . . $p(n)$ zu modellieren

–Schreibe Konfiguration (u,q,v) als String $\boldsymbol{uqvB^j}$ der Länge $p(n)$+1

Satz von Cook: zu codierende Aussagen

- **Verwende Konfigurationsvariablen $y_{t,i,A}$**
  "Nach t Schritten steht an der i-ten Stelle der Konfiguration ein A"
- **Anfangsbedingungen bei Eingabe w**
  – M startet im Zustand q_0 und der Kopf ist über Bandzelle 0
  – Anfangskonfiguration ist $q_0 w_1..w_n B^{p(n)-(n+1)}$
  $A \equiv y_{0,0,q_0} \wedge y_{0,1,w_1} \wedge .. \wedge y_{0,n,w_n} \wedge y_{0,n+1,B} \wedge .. \wedge y_{0,p(n),B}$
- **Übergangsbedingungen**
  – Zu jedem Zeitpunkt t steht der Kopf an einer Stelle j und verändert Bandinhalt und Zustand entsprechend der Tabelle von δ
- **Endbedingung**
  – Nach $p(n)$ Schritten befindet sich M in einem Endzustand $q_f \in F$
  – Endkonfiguration hat die Form $X_0..X_{j-1} q_f X_{j+1}..X_{p(n)}$ für ein j
- **Randbedingungen für eindeutiges Verhalten**
  – Zu jedem Zeitpunkt t befindet sich M in genau einer Konfiguration

Summe dieser Aussagen codiert Berechnung von M auf w

Die Codierung und ihre Korrektheit (Skizze)

- **Codiere Aussagen durch KNF-Formeln $A, \ddot{U}, E, R$**
  – Jede Teilformel ist in der Zeit $\mathcal{O}(p(n)^3)$ konstruierbar (Details im Anhang)
- **Setze $\alpha(M,w) \equiv A \wedge \ddot{U} \wedge E \wedge R$**
- **$\alpha(M,w)$ ist in KNF,** da jede der Teilformeln in KNF ist
- **$\alpha(M,w)$ ist in polynomieller Zeit konstruierbar**
- **$w \in L \;\Rightarrow\; \alpha(M,w) \in SAT$**
  Für $w \in L$ gibt es eine akzeptierende Berechnung $K_0,..,K_{p(n)}$.
  Setze: $y_{t,i,A}$:=1, falls A das i-te Symbol von K_t ist, $y_{t,i,A}$:=0, sonst.
  Per Konstruktion erfüllt dies die Formel $\alpha(M,w)$, also $\alpha(M,w) \in SAT$.
- **$\alpha(M,w) \in SAT \;\Rightarrow\; w \in L$**
  Ist $\alpha(M,w)$ erfüllbar, so kann mit Ü die Belegung der Variablen in eine Konfigurationsfolge $K_0,..,K_{p(n)}$ umgerechnet werden. Wegen R gibt es genau eine solche Konfigurationsfolge. Wegen A und E repräsentiert diese Folge eine akzeptierende Berechnung für w. Also gilt $w \in L$.

Satz von Cook: Zusammenfassung

- **Aufwendige Codierung von Berechnungen**
  - Formel $\alpha(M,w)$ codiert Berechnung der NTM M bei Eingabe w
  - $\alpha(M,w)$ ist in polynomieller Zeit berechenbar (relativ zu $|w|$)
  - Es gilt $w \in L(M) \;\Leftrightarrow\; \alpha(M,w) \in SAT$
  - Es folgt $L(M) \leq_p SAT$
- **Konstruktion ist uniform für polynomielle NTMs**
  - Es folgt **$L \leq_p SAT$ für jedes $L \in \mathcal{NP}$**
- **SAT ist selbst in $\mathcal{NP}$** vgl Folie 7
  - Belegungen können leicht als erfüllend überprüft werden

$$\Downarrow$$

SAT ist $\mathcal{NP}$-vollständig

ANHANG

Details der Codierung: Anfangsbedingungen

Anfangskonfiguration ist $q_0 w_1..w_n B^{p(n)-(n+1)}$

- **Verwende Konfigurationsvariablen** $y_{t,i,A}$
  - Zeit t und Zelle i sind Zahlen zwischen 0 und $p(n)$
  - A ist ein Symbol aus Γ oder ein Zustand $(A \in \{X_1, .., X_m, q_0, .., q_k\})$
- **Codiere Anfangsbedingungen als Formel** A **mit**

$$A \equiv y_{0,0,q_0} \wedge y_{0,1,w_1} \wedge .. \wedge y_{0,n,w_n} \wedge y_{0,n+1,B} \wedge .. \wedge y_{0,p(n),B}$$

- A **ist in KNF** — Rein konjunktive Formel
- **Größe:** $\mathcal{O}(p(n))$ — $p(n)+1$ Variablen
- **Berechnungsaufwand:** $\mathcal{O}(p(n))$ — Bestimmung von $p(n)$

Details der Codierung: Übergangsbedingungen

Konfigurationsübergänge sind verträglich mit δ

Definiere Formeln $\ddot{\mathrm{U}}(t,i)$ für Zeit t und Stelle i

– *Falls M an Stelle i steht, kann sich der Bereich $i{-}1..i{+}1$ ändern*

$$
\begin{aligned}
&(y_{t,i-1,Z} \wedge y_{t,i,q} \wedge y_{t,i+1,X}) \\
&\Rightarrow\ (y_{t+1,i-1,p_1} \wedge y_{t+1,i,Z} \wedge y_{t+1,i+1,Y_1}) \\
&\qquad \vee\ ..\ \vee (y_{t+1,i-1,p_l} \wedge y_{t+1,i,Z} \wedge y_{t+1,i+1,Y_l}) \\
&\qquad \vee (y_{t+1,i-1,Z} \wedge y_{t+1,i,Y'_1} \wedge y_{t+1,i+1,p'_1}) \\
&\qquad \vee\ ..\ \vee (y_{t+1,i-1,Z} \wedge y_{t+1,i,Y'_r} \wedge y_{t+1,i+1,p'_r})
\end{aligned}
$$

für jedes $Z \in \Gamma$, falls $\delta(q,X) = \{(p_1,Y_1,L),..,(p_l,Y_l,L),(p'_1,Y'_1,R),..,(p'_r,Y'_r,R)\}$

– *Falls M nicht im Bereich $i{-}1..i{+}1$ steht, bleibt Stelle i unverändert*

$$
\begin{aligned}
&(\overline{y_{t,i-1,q_0}} \wedge .. \wedge \overline{y_{t,i-1,q_k}} \wedge \overline{y_{t,i,q_0}} \wedge .. \wedge \overline{y_{t,i,q_k}} \wedge \overline{y_{t,i+1,q_0}} \wedge .. \wedge \overline{y_{t,i+1,q_k}}) \\
&\Rightarrow\ (y_{t,i,X_1} \wedge y_{t+1,i,X_1}) \vee .. \vee (y_{t,i,X_m} \wedge y_{t+1,i,X_m})
\end{aligned}
$$

Details der Codierung: Übergangsbedingungen (II)

Konfigurationsübergänge sind verträglich mit δ

- **Kombiniere Übergangsbedingungen zu Formel Ü**

$$\text{Ü} \equiv \text{Ü}(0,0) \wedge .. \wedge \text{Ü}(0,p(n))$$
$$\wedge\ \text{Ü}(p(n),0) \wedge .. \wedge \text{Ü}(p(n),p(n))$$

Formeln werden zuvor in KNF transformiert (Standardverfahren)

- **Ü ist in KNF** — Alle Ü(t,i) wurden normalisiert

- **Größe: $\mathcal{O}(p(n)^2)$** — $p(n)^2$ Komponentenformeln

  Je Komponente nach Normalisierung maximal $k*m*3^{2m*k}+3k*2^m$ Symbole

- **Berechnungsaufwand: $\mathcal{O}(p(n)^2)$**

Details der Codierung: Endbedingung

Endkonfiguration hat Form $X_0..X_{j-1}q_fX_{j+1}..X_{p(n)}$

- **Sei $F=\{q_r, .., q_e\}$**

  Codiere Endbedingungen als Formel E mit

$$
\begin{aligned}
E = \; & (y_{p(n),0,q_r} \vee .. \vee y_{p(n),0,q_e}) \\
& \vee (y_{p(n),1,q_r} \vee .. \vee y_{p(n),1,q_e}) \\
& \vee \qquad \vdots \\
& \vee (y_{p(n),p(n),q_r} \vee .. \vee y_{p(n),p(n),q_e})
\end{aligned}
$$

- **E ist in KNF** Einfache Klausel

- **Größe: $\mathcal{O}(p(n))$** $p(n) * (e-r)$ Variablen

- **Berechnungsaufwand: $\mathcal{O}(p(n))$**

DETAILS DER CODIERUNG: RANDBEDINGUNGEN

Eindeutige Konfiguration $X_0..X_{j-1}qX_{j+1}..X_{p(n)}$

- **Codiere Randbedingungen als Formel R:**
  - Zu jedem Zeitpunkt steht an jeder Stelle genau ein Symbol
  - Zu jedem Zeitpunkt steht nur an einer Stelle ein Zustand
  - Optimierungen möglich ("*maximal eine Konfiguration*" reicht)

$$R \equiv \exists_1(y_{0,0,X_1},..,y_{0,0,q_k}) \wedge \ldots \wedge \exists_1(y_{p(n),p(n),X_1},..,y_{p(n),p(n),q_k})$$
$$\wedge\ \exists_1(y_{0,0,q_1},..,y_{0,p(n),q_k}) \wedge \ldots \wedge \exists_1(y_{p(n),0,q_1},..,y_{p(n),p(n),q_k})$$

Dabei steht $\exists_1(x_1,..,x_m)$ für "genau eines der x_i gilt'

$$\exists_1(x_1,..,x_j) \equiv (x_1 \vee .. \vee x_j) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge .. \wedge (\bar{x}_1 \vee \bar{x}_j)$$
$$\wedge (\bar{x}_2 \vee \bar{x}_3) \wedge .. \wedge (\bar{x}_2 \vee \bar{x}_j) \wedge .. \wedge (\bar{x}_{j-1} \vee \bar{x}_j)$$

- **R ist in KNF** — Konjunktion von $\exists_1$-Formeln
- **Größe: $\mathcal{O}(p(n)^3)$** — $p(n)^2 * (m{+}k)^2 + p(n) * (k * p(n))^2$ Variablen
- **Berechnungsaufwand: $\mathcal{O}(p(n)^3)$** — Bestimme $p(n)$,...