Universität Trier

Fachbereich IV - Informatik

Christoph Meinel:

Komplexitätstheorie

Vorlesungsskript Sommersemester 2000

LaTeX-Satz: Wolfgang Licht

Inhaltsverzeichnis

Kapitel 1

(Konzepte der) Komplexitätstheorie

Einführung

1.1 Probleme und Algorithmen

In der Komplexitätstheorie denkt man über untere und obere Schranken für den Ressourcenbedarf zur algorithmischen Lösung von **Problemen** nach.

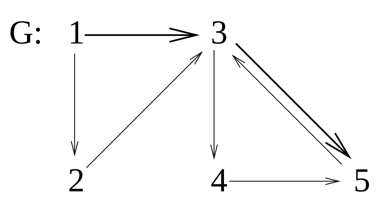
1.1.1 Graph-Erreichbarkeitsproblem

Sei $G = (V, E)$ mit
V .... endliche Menge von Knoten
$E \subseteq V \times V$ ... Menge von Kanten (gerichtet oder ungerichtet)
$v_0, v_1 \in V$

Grundlegende Frage: Existiert in G ein Weg von v_0 nach v_1?

Diese Frage nennt man **Graph-Erreichbarkeitsproblem** bzw. $REACHABILITY$.

Beispiel 1.1:



In G existiert ein Weg von 1 nach 5 ($1 \to 3 \to 5$), aber kein Weg von 5 nach 1.

Definition 1.1: Fragestellungen, die mit "Ja" oder "Nein" beantwortet werden können, heißen **Entscheidungsprobleme**.

Ein Problem hat unendlich viele **Instanzen**, d.h. Fragestellungen, z. B. *REACHABILITY* (Graph, Knotenpaar).

Wir untersuchen Algorithmen zur Lösung von Problemen.

REACHABILITY für $G = (\{v_1, \ldots, v_n\}, E), v_1, v_n$ kann durch folgenden Suchalgorithmus gelöst werden, wobei wir mit der Menge $S \subseteq V$ zur Markierung bestimmter Knoten in V arbeiten:

- Wir starten mit $S = \{v_1\}$ und markieren den Knoten v_1
- In jeder Iteration wird ein Knoten $v \in S$ gewählt und aus S entfernt
- Wir untersuchen alle in v startenden Kanten $(v, v') \in E$:
  Ist v' unmarkiert, dann wird v' markiert und in die Menge S aufgenommen.
- Wir stoppen, wenn $S = \emptyset$
- Die Antwort ist "JA", falls v_n markiert ist, ansonsten "NEIN"

Korrektheitsbeweis:
g.z.z.: v ist markiert, falls ein Weg in G von v_1 nach v existiert.

□

Achtung:

1. Wir haben nicht genau gesagt, wie die Operationen im einzelnen ausgeführt werden; wir brauchen dazu ein **Berechnungsmodell**.
   Wir wissen bereits: Verschiedene Berechnungsmodelle sind "ziemlich gleichstark".
   $\rightarrow$ Church'sche These, polynomiale Simulierbarkeit verschiedener Berechnungsmodelle, ...

2. Wir haben die Frage der Datenstrukturen für Knoten, Kanten und S vollkommen außer Acht gelassen.
   Bevor der Ressourcenbedarf eines Algorithmus analysiert werden kann, müssen Details geklärt sein:

   Bei *REACHABILITY* repräsentieren wir den Graph durch eine Adjazenzmatrix $A_G = M_{n,n}\{0, 1\}$, d. h.

   $$A_G = (a_{ij}) \qquad \text{mit} \qquad a_{ij} = \begin{cases} 1 & falls\ (v_i, v_j) \in E \\ 0 & sonst \end{cases}$$

   Wir müssen bei dem Algorithmus etwa n^2 Operationen ausführen.
   $\Rightarrow$ Der Aufwand zur Lösung von *REACHABILITY* ist proportional zu n^2.
   $\Rightarrow$ Zeitbedarf: $\mathcal{O}(n^2)$
   $\Rightarrow$ $REACHABILITY \in TIME(n^2)$

Bemerkung: Die $\mathcal{O}$**-Notation** ist ein wichtiges Konzept in der Komplexitätstheorie.

Definition 1.2: Seien $f, g : \mathbb{N} \to \mathbb{N}$.

- $f(n) = \mathcal{O}(g(n))$
  $\underset{def.}{=} \exists\, c, n_0 \in \mathbb{N}$ mit $f(n) \leq c \cdot g(n)$, falls $n \geq n_0$
  Bedeutung: f wächst nicht schneller als g.

- $f(n) = \Omega(g(n))$
  $\underset{def.}{=} g(n) = \mathcal{O}(f(n))$
  Bedeutung: f wächst wenigstens so schnell wie g.

- $f(n) = \Theta(g(n))$
  $\underset{def.}{=} f(n) = \mathcal{O}(g(n))$ und $f(n) = \Omega(g(n))$
  Bedeutung: f und g wachsen gleich schnell.

Beispiel 1.2:

1. Sei $p(n)$ Polynom vom Grad d.
   $\Rightarrow p(n) = \mathcal{O}(n^d)$

2. Sei $c \in \mathbb{N}, c > 1$. Sei $p(n)$ Polynom.
   Dann gilt: $p(n) = \mathcal{O}(c^n)$
   Aber es gilt **nicht**: $p(n) = \Omega(c^n)$
   Polynome wachsen deutlich langsamer als exponentielle Funktionen.

3. Es gilt: $\log^k n = \mathcal{O}(n)$.
   Es gilt **nicht**: $\log^k n = \Omega(n)$.

4. $REACHABILITY \in \mathcal{O}(n^2)$

Generelle These: Polynomialzeitalgorithmen können praktisch ausgeführt werden. Algorithmen, die asymptotisch mehr als Polynomialzeit benötigen (z. B. Zeit von 2^n), werden als praktisch unausführbar angesehen. $\to$ Grundvorlesung.

Achtung: Polynomialzeitalgorithmen mit Laufzeit $\mathcal{O}(n^{80})$ können praktisch unzugänglicher sein als Algorithmen mit Laufzeit $\mathcal{O}(n^{\log n})$.

Bemerkung: Neben dem Zeitbedarf ist der Speicherplatz wichtig.

$REACHABILITY$: Man benötigt Speicher für $S : \#S \leq \#V = n$
$\Rightarrow REACHABILITY$ kommt mit Speicher $\mathcal{O}(n)$ aus.
$\Rightarrow REACHABILITY \in SPACE(n)$

1.1.2 $TRAVELING\ SALESMAN$ - Problem (TSP)

geg.: n Städte $1, \ldots, n$;
(nicht negative) Abstände d_{ij} ($1 \leq i, j \leq n$) zwischen den Städten
(hier $d_{ij} = d_{ji}$ für alle i, j)

ges.: kürzeste Tour durch alle Städte, d.h. Permutation π auf $(1, \ldots, n)$ mit

$$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \textbf{ minimal}, \ \textit{wobei } \pi(n+1) := \pi(1)$$

→ **Grundlegendes Optimierungsproblem**

Bei der Untersuchung von Komplexitätsfragen sind Entscheidungsprobleme besser handhabbar.

Daher: Sei B eine Konstante ("*Budget*").

$$TSP(D)\text{: Existiert eine Tour } \pi \text{ mit } \sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \leq B \ ?$$

→ Entscheidungsproblemversion von TSP

Wir können das Problem durch Untersuchung aller möglichen Permutationen π lösen.

Zeitbedarf: $\mathcal{O}(n!)$ (genauer: $\mathcal{O}(\frac{1}{2}(n-1)!)$) $\Rightarrow$ nicht polynomial

Speicherplatzbedarf: Wir müssen lediglich die gerade untersuchte Permutation und die bisher beste Tour speichern ($\Rightarrow$ Platzbedarf: $\mathcal{O}(n)$).

Die Laufzeit kann zwar mit Hilfe der Technik des dynamischen Programmierens etwas verbessert werden, aber es ist kein Polynomialzeitalgorithmus bekannt!

Generelle Annahme: Es gibt keinen polynomialen Algorithmus, der das TSP löst.

Diese Annahme ist das Ergebnis intensiver weltweiter Forschung; man ist bisher nicht in der Lage zu beweisen, daß

Es gilt:

$$\left.\begin{array}{r} TSP(\mathrm{D}) \notin \mathcal{P} \\ TSP(\mathrm{D}) \in \mathcal{NP} \end{array}\right\} \mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

1.2 Turingmaschinen (TM)

Wir wissen aus der Berechnungstheorie:

- Turingmaschinen (TM) stellen ein grundlegendes Berechnungsmodell dar.
  $\rightarrow$ Church'sche These

- Turingmaschinen haben viele Variationen, z. B. 1-Band-TM, Mehrband-TM;

  Es gilt: Jede $f(n)$-rechenzeitbeschränkte k-Band-TM kann durch eine $\mathcal{O}(f(n)^2)$-zeitbeschränkte 1-Band-TM simuliert werden.

- Turingmaschinen mit mehreren Bändern liefern ein realistisches Berechnungsmodell

Definition 1.3: Eine k-**Band-TM**, $k \geq 1$, ist ein Quadrupel $M = (K, \Sigma, \delta, s)$ mit

K - endliche Menge von Zuständen

Σ - endliches Alphabet; $\sqcup, \triangleright \in \Sigma$ Spezialsymbole

$\delta : K \times \Sigma^k \rightarrow (K \cup \underbrace{\{h, \text{"}yes\text{"}, \text{"}no\text{"}\}}_{\substack{terminierende \\ Zustände}}) \times (\Sigma \times \underbrace{\{\leftarrow, -, \rightarrow\}}_{\substack{Lesekopf- \\ bewegung}})^k$ - Zustandsüberführungsfunktion

(*mit* $h \mathrel{\hat{=}}$ *Haltezustand*, "*yes*" $\mathrel{\hat{=}}$ *akzeptierender Zustand*, "*no*" $\mathrel{\hat{=}}$ *verwerfender Zustand*),

wobei $\delta_i(q, \triangleright) = (\triangleright, \rightarrow)$, und $\delta(q, w_1, w_2, \ldots, w_k) = (p, \delta_1(q, w_1), \delta_2(q, w_2), \ldots, \delta_k(q, w_k))$,

s - initialer Zustand

Startsituation: Bandbeschriftung: $\ldots \triangleright \underset{\substack{\uparrow \\ Lesekopf}}{\triangleright} w_1 w_2 \ldots w_k \sqcup\sqcup\sqcup \ldots$ auf erstem Band

$w_i \in \Sigma$ $(w_i \neq \triangleright, \sqcup)$

$\ldots \triangleright \sqcup\sqcup\sqcup \ldots$ auf allen anderen Bändern

M **akzeptiert** $w_1, \ldots, w_k$, falls die Berechnung in den Zustand "*yes*" führt

M **verwirft** $w_1, \ldots, w_k$, falls die Berechnung in den Zustand "*no*" führt

Erreicht M bei Eingabe $w_1, \ldots, w_k$ den Haltezustand, so ist die **Ausgabe** $M(w)$ von M
- "*yes*", falls "*yes*" Zustand
- "*no*", falls "*no*" Zustand
- oder aktueller String auf k-tem Band
- oder $\nearrow$, falls M nicht hält

Beispiel 1.3: 2-Band-TM für Palindrome (d.h. $w = w^R$), $\Sigma = \{0, 1, \triangleright, \sqcup\}$

$\rho \in K$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\delta(\rho, \sigma_1, \sigma_2)$
s	0	$\sqcup$	$(s, 0, \rightarrow, 0, \rightarrow)$
s	1	$\sqcup$	$(s, 1, \rightarrow, 1, \rightarrow)$
s	$\triangleright$	$\triangleright$	$(s, \triangleright, \rightarrow, \triangleright, \rightarrow)$
s	$\sqcup$	$\sqcup$	$(q, \sqcup, \leftarrow, \sqcup, -)$
q	0	$\sqcup$	$(q, 0, \leftarrow, \sqcup, -)$
q	1	$\sqcup$	$(q, 1, \leftarrow, \sqcup, -)$
q	$\triangleright$	$\sqcup$	$(p, \triangleright, \rightarrow, \sqcup, \leftarrow)$
p	0	0	$(p, 0, \rightarrow, \sqcup, \leftarrow)$
p	1	1	$(p, 1, \rightarrow, \sqcup, \leftarrow)$
p	0	1	$(\text{"}no\text{"}, 0, -, 1, -)$
p	1	0	$(\text{"}no\text{"}, 1, -, 0, -)$
p	$\sqcup$	$\triangleright$	$(\text{"}yes\text{"}, \sqcup, -, \triangleright, \rightarrow)$

Definition 1.4: Eine **Konfiguration** einer k-Band-TM ist ein $(2k+1)$-Tupel $(q, w_1, u_1, \ldots, w_k, u_k)$ mit

q	-	aktueller Zustand
$w_i u_i$	-	nichttriviale Inschrift des i-ten Bandes auf dem rechtesten Symbol von w_i steht der i-te Schreib-/Lesekopf

$$(q, w_1, u_1, \ldots, w_k, u_k) \overset{M}{\longmapsto} (q', w_1', u_1', \ldots, w_k', u_k'),$$

falls $\delta(q, \sigma_1, \ldots, \sigma_k) = (q', \rho_1, D_1, \ldots, \rho_k, D_k)$, wobei σ_i rechtestes Symbol von w_i ist und

falls $D_i = \rightarrow$, dann ergibt sich w_i' aus w_i durch Ersetzung von σ_i durch ρ_i und Anhängen des ersten Symbols von u_i;
u_i' ergibt sich aus u_i durch Streichen des ersten Symbols

$D_i = \leftarrow$, dann ergibt sich w_i' aus w_i durch Streichen des letzten Symbols σ_i;
u_i' ergibt sich durch Voranstellung von ρ_i

$D_i = -$, dann ergibt sich w_i' aus w_i durch Ersetzung des letzten Symbols σ_i durch ρ_i und $u_i' = u_i$

Wir definieren wie immer: $\overset{t}{\longmapsto}, \overset{*}{\longmapsto}, \ldots$

Startkonfiguration: $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup)$ mit $w = w_1, \ldots, w_k$
Gilt $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M*}{\longmapsto} (\text{"}yes\text{"}, w_1, u_1, \ldots, w_k, u_k)$, dann gilt $M(w) = \text{"}yes\text{"}$.
Gilt $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M*}{\longmapsto} (\text{"}no\text{"}, w_1, u_1, \ldots, w_k, u_k)$, dann gilt $M(w) = \text{"}no\text{"}$.
Gilt $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M*}{\longmapsto} (h, w_1, u_1, \ldots, w_k, u_k)$, dann gilt $M(w) = w_k u_k$ (ohne $\triangleright, \sqcup$).

Bemerkung:
$k = 1$: Dann erhält man eine "normale" (1-Band)-TM.

Definition 1.5:

(1) Sei M eine k-Band-TM mit Eingabe $w(\in \Sigma^*)$.
Gilt $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M^t}{\longmapsto} (H, w_1, u_1, \ldots, w_k, u_k)$ für ein $H \in \{h, "yes", "no"\}$, dann heißt t die für w benötigte **Zeit** (Rechenzeit).
Gilt $M(w) = \nearrow$, dann benötigt M für w die Zeit ∞.

(2) Sei $f : I\!N \to I\!N$
M arbeitet in der Zeit $f(n)$
$\underset{def.}{=}$ für alle Eingaben $w \in (\Sigma - \{\triangleright, \sqcup\})^*$ benötigt M höchstens die Zeit $f(|w|)$

(3) Sei $L \subseteq (\Sigma - \{\triangleright, \sqcup\})^*$ und sei L von der k-Band-TM M entscheidbar.
Dann gilt $L \in TIME(f(n))$
$\underset{def.}{=}$ M ist $f(n)$-zeitbeschränkt

Bemerkungen:

- $TIME(f(n))$ heißt (Zeit-)Komplexitätsklasse.
- Komplexitätsklassen sind Mengen von Sprachen, die sämtlich in der Zeit $f(n)$ entscheidbar sind.
- Es gilt $TIME(f(n)) \subseteq TIME(g(n))$, falls $f(n) \leq g(n)$ für alle $n > n_0 \in I\!N$.

Beispiel 1.4:
Die 2-Band-TM aus dem letzten Beispiel entscheidet $PALINDROME$ in der Zeit

$$f(n) = 3n + 3 = \mathcal{O}(n)$$

$\Rightarrow PALINDROME \in TIME(3n + 3)$

Nun stellt sich die Frage, wieviel beim Übergang von einer 1-Band-TM zu einer k-Band-TM an Zeit gewonnen werden kann.

Satz 1.1: Sei M eine $f(n)$-zeitbeschränkte k-Band-TM.
Dann existiert eine $\mathcal{O}(f(n)^2)$-zeitbeschränkte 1-Band-TM $\widetilde{M}$, die M simuliert, d.h. für alle Eingaben w gilt:

$$M(w) = \widetilde{M}(w).$$

Beweis:
Idee: Wir konkatenieren alle Bandinschriften von M; wir müssen uns dabei die Lesekopfkonfiguration der einzelnen Bänder merken.
$\widetilde{M} = (\widetilde{K}, \widetilde{\Sigma}, \widetilde{\delta}, \widetilde{s})$ mit $\widetilde{\Sigma} = \Sigma \cup \underline{\Sigma} \cup \{\triangleright', \triangleleft, \triangleleft'\}$

Sei $\underline{\Sigma} = \{\underline{\sigma} : \sigma \in \Sigma\}$ ($\underline{\sigma}$ "Lesekopfversion" von σ)
$\triangleright'$ neue Version von $\triangleright$, die auch nach links überschritten werden kann,
$\triangleleft, \triangleleft'$ Markierung des rechten Endes einer Bandinschrift

Sei $(q, w_1, u_1, \ldots, w_k, u_k)$ eine Konfiguration von M.
Wir simulieren diese Konfiguration durch die Konfiguration
$(q, \triangleright w_1' u_1 \triangleleft w_2' u_2 \triangleleft \ldots \triangleleft w_k' u_k \triangleleft \triangleleft)$, wobei w_i' aus w_i durch Ersetzung des letzten Symbols σ_i durch $\underline{\sigma_i}$ hervorgeht und Voranstellung von $\triangleright'$ anstelle von $\triangleright$.

Simulation der Startkonfiguration:
$\widetilde{M}$ bewegt die Eingabe w um eine Position nach rechts, stellt $\triangleright'$ voran und $\triangleleft(\triangleright'\triangleleft)^{k-1}\triangleleft$ dahinter ($\to 2k+2$ neue Zustände für $\widetilde{M}$).

Um einen Schritt von M zu simulieren, scannt $\widetilde{M}$ die Inschrift zweimal von links nach rechts und zurück:

1. Scan: $\widetilde{M}$ liest die k momentan gelesenen Symbole (unterstrichene Symbole); um sich diese merken zu können, benötigt $\widetilde{M}$ zusätzliche Zustände, nämlich einen für jede k-Konkatenation von Symbolen aus $\underline{\Sigma}$.
Dann ist $\widetilde{M}$ in der Lage, die Übergangsfunktion von M auszuführen (alle dazu benötigten Informationen sind bekannt).

2. Scan: $\widetilde{M}$ führt die auszuführenden Änderungen gemäß Übergangsfunktion aus ($\widetilde{M}$ verändert evtl. die unterstrichenen Symbole und Unterstreichungen selbst).

Einziges Problem: Wenn der Lesekopf am rechten Ende einer Bandinschrift ein neues Symbol einfügen will, dann muß freier Platz $\sqcup$ geschaffen werden.
Wir markieren dazu das entsprechende Endesymbol $\triangleleft$ durch Ersetzung mit $\triangleleft'$ und merken uns die vorzunehmenden Änderungen.
Beim Weitergehen verschiebt $\widetilde{M}$ alle Symbole (bei $\triangleleft'$) um eine Position nach rechts; beim Rückweg wird $\triangleleft'$ durch $\triangleleft$ ersetzt und die gemerkte Änderung vorgenommen.

Analyse:
M hält bei Eingabe w nach höchstens $f(|w|)$ Schritten
$\Rightarrow$ jede einzelne Bandinschrift wird nie länger als $f(|w|)$
$\Rightarrow$ der von $\widetilde{M}$ zu bearbeitende String hat eine Länge $\leq k \cdot (f(|w|)+1)+1$

Die Simulation eines Schrittes erfordert höchstens 2 Traversierungen (Durchläufe), d. h. $4(k(f(|w|)+1)+1)$ und höchstens $3k(f(|w|)+1)+3$ Schritte für jedes Band.
$\Rightarrow$ Zeitbedarf: $\mathcal{O}(k^2 \cdot \mathrm{f}(|w|)^2)$
k ist unabhängig von w konstant
$\Rightarrow \mathcal{O}(f(|w|)^2)$

$\square$

1.3 Lineare Speed-Ups

Bei der Analyse der Zeitkomplexität sind lineare Faktoren unwichtig.

Satz 1.2 (Linear Speed-up Theorem): Sei $L \in TIME(f(n))$.
Für jedes $\varepsilon > 0$ gilt: $L \in TIME(\widetilde{f}(n))$ mit $\widetilde{f}(n) = \varepsilon \cdot f(n) + n + 2$

Beweis: Sei $M = (K, \Sigma, \delta, s)$ eine $f(n)$-zeitbeschränkte k-Band-TM für L.
Wir konstruieren eine $\widetilde{f}(n)$-zeitbeschränkte $\widetilde{k}$-Band-TM $\widetilde{M} = (\widetilde{K}, \widetilde{\Sigma}, \widetilde{\delta}, \widetilde{s})$ für L, wobei

$$\widetilde{k} := \begin{cases} k & : \quad k > 1 \\ 2 & : \quad sonst \end{cases}$$

Idee: Wir arbeiten bei $\widetilde{M}$ mit Symbolen, die m Symbole von M zusammenfassen.
$\Rightarrow$ Wir können in einem Schritt von $\widetilde{M}$ viele Schritte von M ausführen.
Seien dazu

$$\widetilde{\Sigma} := \Sigma \cup \Sigma^m$$

$$\widetilde{K} := \bigcup_{i=1}^{m-1} (K \times \Sigma^i) \times (K \times \{1, \ldots, m\}) \times \Sigma^{3mk}$$

Zu Beginn geht $\widetilde{M}$ an das Ende des ersten Bandes und liest die Eingabe w. Sind m Symbole $\sigma_1, \ldots, \sigma_m$ gelesen, dann wird das Symbol $(\sigma_1, \ldots, \sigma_m) \in \widetilde{\Sigma}$ auf das zweite Band geschrieben (das zu diesem Zeitpunkt leer ist); dies ist ausführbar mit Hilfe der neuen Zustände aus $\widetilde{K}$.

Dann werden die nächsten m Symbole gelesen, usw. (falls $|w|$ nicht durch m teilbar ist, so werden zusätzliche $\sqcup$ betrachtet).

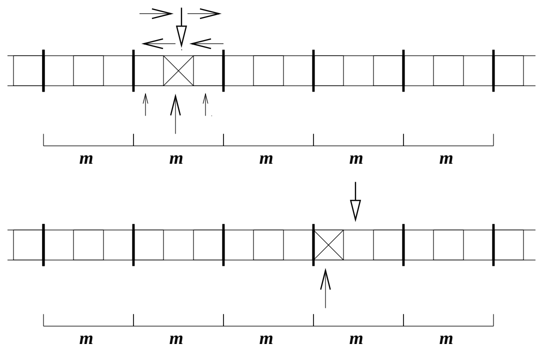
Nach $m \cdot (\frac{|w|}{m}) + 2$ Schritten enthält das zweite Band eine "kondensierte" Version der Eingabe; alle anderen Bänder bleiben bei $\triangleright$.

Nun wird das zweite Band als Eingabeband behandelt. Im Laufe der weiteren Rechnung schreiben wir lediglich m-Tupel.

$\widetilde{M}$ simuliert m Schritte von M durch höchstens 6 eigene Schritte (sogenannte m-Schritte): Zu Beginn eines m-Schritts enthalte der Zustand von $\widetilde{M}$ das Tupel $(q, j_2, \ldots, j_k)$, $j_i < m$ gibt dabei die Position des i-ten Schreib-/Lesekopfes innerhalb des aktuellen m-Tupels an.

Dann bewegt $\widetilde{M}$ alle $\widetilde{k}$ Leseköpfe simultan einen Schritt nach links, zwei Schritte nach rechts und wiederum einen Schritt nach links.

Beispiel: $m = 3$



$\widetilde{M}$ kennt alle Symbole, die von M gelesen werden und speichert diese intern mit Hilfe der Zustände aus $K \times \{1, \ldots, m\} \times \Sigma^{3mk}$.
$\widetilde{M}$ kann so genau rekonstruieren, was M tut.

Bei dieser Arbeitsweise geht $\widetilde{M}$ niemals weiter als evtl. zum linken bzw. rechten m-Tupel.
$\Rightarrow$ $\widetilde{M}$ kann mit Hilfe von $\widetilde{\sigma}$ in zwei Schritten einen Wechsel in der Bandinschrift und im Zustand realisieren.
$\Rightarrow$ Die Simulation von m Schritten von M kann in 6 Schritten von $\widetilde{M}$ erfolgen.

Das Akzeptierungs-/ Halte- und Verwerfungsverhalten von $\widetilde{M}$ entspricht dem von M.

Analyse: $\widetilde{M}$ hat bei der Eingabe w einen Zeitbedarf von höchstens $|w| + 2 + 6 \cdot \left\lceil \frac{f(|w|)}{m} \right\rceil$.
Für $m := \left\lceil \frac{6}{\varepsilon} \right\rceil$ erhält man die Behauptung.

□

Bemerkung: TM mit einem Zeitbedarf $< n$ sind uninteressant:
Sie können nicht einmal die Eingabe (der Länge n) vollständig lesen.
$\Rightarrow$ O. B. d. A. sei $f(n) \geq n$.

Gilt: $f(n) = c \cdot n \Rightarrow$ Wir können f beliebig nahe an n bringen.
Gilt: $f(n) = 13n^2 + 4n \Rightarrow$ Wir können den Parameter des führenden Terms (Polynomglied höchsten Grades) beliebig nahe an 1 bringen;
Terme niedriger Ordnung können komplett außer Acht gelassen werden.
$\rightarrow$ Rechtfertigung für die Arbeit mit der $\mathcal{O}$-Notation.

Für jede polynomialzeit-berechenbare Sprache L gilt:

$$L \in TIME(n^k) \; für \; ein \; k$$

Definition 1.6:

$$\boxed{\mathcal{P} := \bigcup_{k=0}^{\infty} TIME(n^k)}$$

1.4 Raumschranken und Platzsparen

Definition 1.7: Sei $M = (K, \Sigma, \delta, s)$ eine k-Band-TM.
Gilt für die Eingabe $w \in (\Sigma - \{\sqcup\})^*$ $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M^t}{\longmapsto} (H, w_t^{(1)}, u_t^{(1)}, \ldots, w_t^{(k)}, u_t^{(k)})$
mit $H \in \{h, "yes", "no"\}$,
und gilt ferner $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{M^r}{\longmapsto} (p, w_r^{(1)}, u_r^{(1)} \ldots, w_r^{(k)}, u_r^{(k)})$ mit $p \neq H$ für alle $r < t$, dann heißt

$$\max_{1 \leq r \leq t} \left\{ \sum_{i=1}^{k} |w_r^{(i)} u_r^{(i)}| \right\}$$

der für w von M benötigte **Raum** / **Speicher** / **Platz**.

Bemerkungen:

a) Nach obiger Definition wird bei der Eingabe w stets Speicherplatz $\geq |w|$ benötigt.
Wir wollen genauer arbeiten:
Modellmodifikation:
Auf dem Eingabeband darf nur gelesen werden, jedoch nie geschrieben
($\rightarrow$ "read-only"-Band).
Raumbedarf:

$$\max_{1 \leq r \leq t} \left\{ \sum_{i=2}^{k} |w_{r_i} u_{r_i}| \right\}$$

Beispiel 1.5:
Palindrome können mit einem Speicherplatz von $\log |w|$ erkannt werden:
Sei M eine 3-Band-TM.
Band 1 enthält die Eingabe w und darf nicht beschrieben werden.

Stufe i**:** Band 2 enthält die Binärkodierung von "i".
Wir vergleichen das i-te Symbol von w mit dem $(n - i)$-ten Symbol von w.
Dazu schreiben wir auf Band 3 binär $j = 1$ und zählen j hoch bis zu i; dabei wird der Lesekopf auf der Eingabe jeweils um eine Position nach rechts verschoben, M merkt sich das bei $j = i$ erreichte Symbol von w.
Dann wird j wieder auf 1 gesetzt und erneut bis $j = i$ hochgezählt, wobei jetzt der Lesekopf auf der Eingabe vom rechten Ende jeweils um eine Position nach links verschoben wird.
Stimmt das bei $j = i$ erreichte Symbol mit dem gespeicherten überein, so

wird $i := i + 1$ gesetzt und die Stufe $i + 1$ abgearbeitet; ansonsten wird "*no*" ausgegeben.

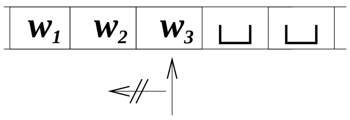
Speicherplatzanalyse: Auf Band 2 und 3 werden höchstens Worte der Länge $\log |w|$ geschrieben (Binärdarstellung der Wortlänge)

$\Rightarrow$ Speicherplatzbedarf: $\mathcal{O}(\log n)$

b) Kommt es bei der TM M auch auf die geschriebene Ausgabe an (Inschrift auf Band k beim Erreichen der Haltezustände) und ist die Ausgabe lang, so benötigt M nach der bisherigen Definition viel Speicherplatz.
Modellmodifikation:
Wir erlauben auf dem Ausgabeband nur Rechtsbewegungen und Schreiben
($\rightarrow$ "write-only"-Band).
Raumbedarf:

$$\max_{1 \leq r \leq t} \left\{ \sum_{i=2}^{k-1} |w_{r_i} u_{r_i}| \right\}$$

Beispiel:



Wir betrachten bei der Speicherplatzuntersuchung stets eine k-Band-TM folgenden Typs:

Definition 1.8: Ist $\delta(p, \sigma_1, \ldots, \sigma_k) = (q, \rho_1, D_1, \ldots, \rho_k, D_k)$, dann gilt:

a) $\rho_1 = \sigma_1$ (d. h. Band 1 ist **read-only**)

b) $D_k \neq \leftarrow$ (d. h. Band k ist **write-only**)

c) Ist $\sigma_1 = \sqcup$, dann gilt: $D_1 = \leftarrow$ (verhindert, das der Lesekopf auf dem Eingabeband über den rechten Rand hinausläuft)

Eine TM mit a) - c) heißt TM mit **Ein- und Ausgabe**.

Satz 1.3: Für jede $f(n)$-zeitbeschränkte k-Band-TM M gibt es eine $(k+2)$-Band-TM M' mit Ein- und Ausgabe, die M in der Zeit $\mathcal{O}(f(n))$ simuliert.

Beweis: M' kopiert die Eingabe von M auf das $(k+1)$-te Band und führt alle Schritte von M auf diesem anstelle des Eingabebandes aus.
Hält M, dann kopiert M' die Ausgabe von Band k auf das $(k+2)$-te Band.
Offensichtlich gilt die Behauptung.

$\square$

Definition 1.9: Sei $M = (K, \Sigma, \delta, s)$ eine k-Band-TM.

(1) Gilt für die Eingabe $w \in (\Sigma - \{\sqcup\})^*$ $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{Mt}{\longmapsto} (H, w_t^{(1)}, u_t^{(1)}, \ldots, w_t^{(k)}, u_t^{(k)})$ mit $H \in \{h, "yes", "no"\}$,
und gilt ferner $(s, \triangleright, w, \triangleright, \sqcup, \ldots, \triangleright, \sqcup) \overset{Mr}{\longmapsto} (p, w_r^{(1)}, u_r^{(1)} \ldots, w_r^{(k)}, u_r^{(k)})$ mit $p \neq H$ für alle $r < t$, dann heißt

$$\max_{1 \leq r \leq t} \left\{ \sum_{i=1}^{k} |w_r^{(i)} u_r^{(i)}| \right\}$$

Speicherplatzbedarf / Speicherplatzkomplexität / ... von M für die Eingabe w.
Ist M eine **TM mit Ein- und Ausgabe**, so heißt

$$\max_{1 \leq r \leq t} \left\{ \sum_{i=2}^{k-1} |w_r^{(i)} u_r^{(i)}| \right\}$$

Speicherplatzkomplexität von M für die Eingabe w.

(2) Sei $f : \mathbb{N} \to \mathbb{N}$.
M benötigt Speicherplatz $f(n)$
$\underset{def.}{=}$ für alle Eingaben $w \in (\Sigma - \{\sqcup\})^*$ gilt: M benötigt für w höchstens den Raum $f(|w|)$

(3) Sei $L \subseteq (\Sigma - \{\sqcup\}^*$ und sei L von einer k-Band-TM M mit Ein- und Ausgabe entscheidbar. Dann gilt:

$$L \in SPACE(f(n)) \quad \underset{def.}{=} \quad M \text{ ist } f(n) - \text{raumbeschränkt.}$$

Bemerkungen:

- $SPACE(f(n))$ heißt **Raumkomplexitätsklasse** oder auch **Speicherkomplexitätsklasse**.
- $SPACE(f(n))$ ist eine Menge von Sprachen, die sämtlich mit dem Platz $f(n)$ entscheidbar sind.

Analog wie bei der Zeit gilt:

Satz 1.4: Sei $L \in SPACE(f(n))$. Dann gilt für alle $\varepsilon > 0$:

$$L \in SPACE(2 + \varepsilon \cdot f(n))$$

Beispiel 1.6: $PALINDROME \in SPACE(\log n)$

Definition 1.10: Wichtige Klassen:

$$\mathcal{PSPACE} := \bigcup_{k=0}^{\infty} SPACE(n^k)$$

$$\boxed{\mathcal{L} := \mathcal{LOGSPACE} := SPACE(\log n)}$$

Bemerkungen:

- $\mathcal{PSPACE}$ verhält sich bezüglich des Speichers analog zur Klasse $\mathcal{P}$ bei der Zeit. Es muß jedoch nicht sein, daß eine Sprache aus $\mathcal{PSPACE}$ in Polynomialzeit berechenbar ist.
- Bei einer Sprache aus $\mathcal{L}$ arbeiten wir jedoch nie länger als mit polynomialer Zeit.

1.5 Nichtdeterministische Turingmaschinen

Definition 1.11:

(1) Eine nichtdeterministische TM (NTM) ist ein Quadrupel $N = (K, \Sigma, \Delta, s)$ mit

K - endliche Menge von Zuständen

Σ - endliches Alphabet

$\Delta \subseteq (K \times \Sigma) \times [(K \cup \{h, "yes", "no"\}) \times \Sigma \times \{\leftarrow, -, \rightarrow\}]$

- Übergangsrelation (anstelle einer Übergangsfunktion)

s - Initialzustand

(2) Die Modifikationen wie k-Band-NTM, NTM mit Ein- und Ausgabe sind definiert wie im deterministischen Fall.

(3) Konfigurationen sind definiert wie bei deterministischen TM.
$\longmapsto$ ist keine Funktion, sondern eine Relation.
$(q, w, u) \overset{N}{\longmapsto} (q', w', u')$
$\underset{def.}{=}$ es existiert ein Übergangsschritt $((q, \sigma), (q', \rho, D)) \in \Delta$ mit

$D = \rightarrow$:	w' geht aus w durch Ersetzung des letzten Symbols σ durch ρ und Anhängen des ersten Symbols von u hervor; u' ergibt sich aus u durch Streichen des ersten Symbols
$D = \leftarrow$:	w' geht aus w durch Streichen des letzten Symbols σ hervor; u' ergibt sich aus u durch Voranstellung von ρ
$D = -$:	w' ergibt sich aus w durch Ersetzung des letzten Symbols σ durch ρ und $u' = u$

(4) Wir können wie üblich $\overset{N^t}{\vdash\!\!\!-\!\!\!-}$, $\overset{N^*}{\vdash\!\!\!-\!\!\!-}$ definieren (beides sind Relationen!).

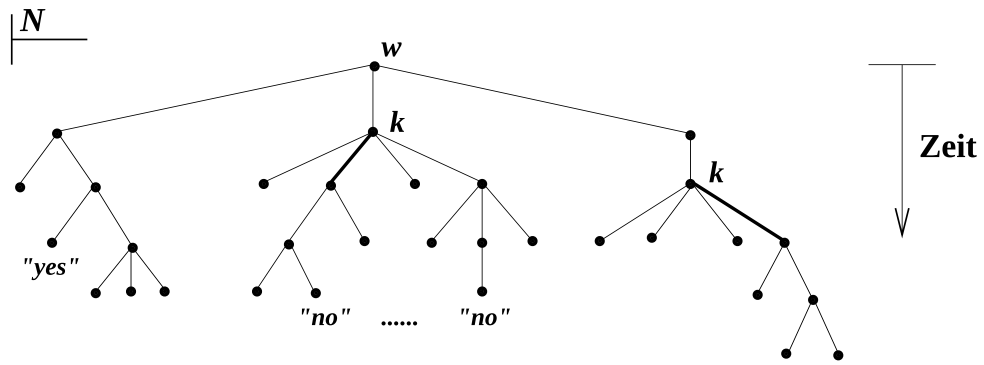
Definition 1.12:
N entscheidet (berechnet) $L \subseteq (\Sigma - \{\sqcup\})^*$
$\underset{def.}{=}$ für alle $w \in (\Sigma - \{\sqcup\})^*$ gilt:

$$w \in L \qquad \Leftrightarrow \qquad (s, \triangleright, w) \overset{N^*}{\vdash\!\!\!-\!\!\!-} (\text{"}yes\text{"}, w', u')$$

Bemerkungen:

- Wir akzeptieren den Input, falls eine Berechnung - also eine mögliche Folge von nichtdeterministischen Übergängen - das Resultat "yes" liefert.
- Es ist durchaus möglich, daß gleichzeitig andere Berechnungen verwerfend sind.
- Der Input wird verworfen, falls keine einzige Berechnung akzeptierend ist.

Beispiel 1.7: Nichtdeterministische Berechnung



Zu beachten ist, daß bei gleicher Konfiguration (k) verschiedene Wege gewählt werden können.

Definition 1.13:

(1) Eine NTM N entscheidet L in der Zeit $f(n)$, $f : I\!N \to I\!N$
$\underset{def.}{=}$ N entscheidet L und für alle $w \in (\Sigma - \{\sqcup\})^*$ gilt:
wenn $(s, \triangleright, w) \overset{N^t}{\vdash\!\!\!-\!\!\!-} (q, w', u')$, dann $t \leq f(|w|)$

(2) $NTIME(f(n))$ ist die Menge aller Sprachen, die von einer k-Band-NTM in der Zeit $\mathcal{O}(f(n))$ entschieden werden können.

Beispiel 1.8:

$$\mathcal{NP} = \bigcup_{r=1}^{\infty} NTIME(n^r)$$

Offenbar gilt:

$$\mathcal{P} \subseteq \mathcal{NP}$$

Große Frage: Gilt auch

$$\boxed{?\ \mathcal{P} \supseteq \mathcal{NP}\ ?}$$

Beispiel 1.9 ($TSP(D)$)**:**
Es gilt: $TSP(D) \in \mathcal{NP}$

- Die NTM N bekommt als Eingabe eine Repräsentation von $TSP(D)$ (d. h. (i, d_{ij}, j) für alle $1 \leq i, j \leq m$) der Länge n.

- N schreibt auf das zweite Band eine beliebige Folge von $\leq n$ Symbolen

- N stellt mit Hilfe des dritten Bandes fest, ob
  a) dieser String eine Darstellung einer Permutation der Zahlen $\{1, \ldots, m\}$ ist,
  b) die Länge des Weges $\leq$ eine vorgegebene Konstante ist.

- Gelten a) und b), so akzeptiert N

Zeitanalyse: $\mathcal{O}(n^2)$

Im Fall der Akzeptierung ist folgendes geschehen:
Die Maschine hat einen der nichtdeterministisch hingeschriebenen Strings als mögliche Tour erkannt; man sagt: "Die Maschine hat die Tour **geraten**."

Satz 1.5: Sei L von einer $f(n)$-zeitbeschränkten NTM N erkennbar.
Dann kann L von einer 3-Band-TM M in der Zeit $\mathcal{O}(c^{f(n)})$, $c = c(N) > 1$, erkannt werden.

Beweis: Sei $N = (K, \Sigma, \Delta, s)$.
Sei ferner $(q, \sigma) \in K \times \Sigma$ mit $C_{q,\sigma} = \{(q', \sigma', D) \mid ((q, \sigma), (q', \sigma', D)) \in \Delta\}$, $C_{q,\sigma}$ endlich.
Wir können $C_{q,\sigma}$ linear anordnen.

$$d := \max_{(q,\sigma)} |C_{q,\sigma}| \quad \text{"Grad des Nichtdeterminismus"}$$
$$(d = 1 \Rightarrow N \text{ ist deterministisch})$$

Simulationsidee: Jede Berechnung von N ist eine Folge nichtdeterministischer Wahlen, also eine Folge von $i \in \{1, \ldots, d\}$.
Die deterministische Maschine M betrachtet alle Folgen $(i_1, \ldots, i_t)$ in der Ordnung aufsteigender Länge und simuliert N auf jeder Folge:
Sei $(i_1, \ldots, i_t)$ eine Folge mit $i_r \in \{1, \ldots, d\}$, die wir auf einem zusätzlichem Band speichern.
M simuliert N, indem sie im Schritt r die Wahl i_r von N weiterverfolgt. Erreicht N dabei einen "*yes*"-Zustand, dann gibt M "*yes*" aus; ansonsten untersucht M die nächste Folge, die sie durch Addition von 1 auf die d-näre "Folgezahl" erzeugen kann.

Zeitanalyse: Für die Eingabe w, $|w| = n$, gilt:

$$\begin{aligned}\mathcal{O}(\sum_{r=1}^{f(n)} d^r) &= \mathcal{O}(d^{f(n)+1}) \\ &= \mathcal{O}(c^{f(n)})\end{aligned}$$

□

Korollar:

$$NTIME(f(n)) \subseteq \bigcup_{c \geq 1} TIME(c^{f(n)})$$

1.6 Komplexitätsklassen

Komplexitätsklassen hängen ab

- vom zugrundeliegenden Berechnungsmodell (Turingmaschine, Random Access Maschine, Schaltkreis, ...)
- vom Berechnungsmodus (deterministisch, nichtdeterministisch, ...)
- von der betrachteten Ressource (Zeit, Raum, Tiefe, ...)
- vom Kostenmaß (logarithmisch, uniform, ...)
- von der Schrankenfunktion (n, n^p, ...)

Wir arbeiten hierbei mit folgendem:

- Mehrband-TM
- deterministischer und nichtdeterministischer Berechnungsmodus
- Zeit und Speicherplatz
- Schrankenfunktion

Definition 1.14 (Schrankenfunktion): Sei $f : \mathbb{N} \to \mathbb{N}$
f ist eine **echte Komplexitätsfunktion**

$\underset{def.}{=}$ (1) f ist nicht-fallend, d. h. $f(n+1) \geq f(n)$ für alle n

(2) Es gilt:
Es gibt eine k-Band-TM $M_f = (K, \Sigma, \delta, s)$ mit Ein- und Ausgabe, so daß für alle n und für alle Eingaben w, $|w| = n$, gilt:
$(s, \triangleright, w, \ldots, \triangleright, \varepsilon) \overset{M_f^t}{\longmapsto} (h, w, \triangleright\sqcup^{j_2}, \ldots, \triangleright\sqcup^{j_{k-1}}, \triangleright\sqcap^{f(|w|)})$,
wobei $t = t(n) = \mathcal{O}(n + f(n))$, $j_i = \mathcal{O}(f(|w|))$, $i = 2, \ldots, k-1$
d. h. bei der Eingabe w berechnet M_f den String $\sqcap^{f(|w|)}$ für ein Quasi-Leerzeichen.
Für alle Eingaben w, $|w| = n$, hält M_f nach $\mathcal{O}(|w| + f(|w|))$ Schritten und benutzt $\mathcal{O}(f(|w|))$ Speicher (außer der Eingabe).

Bemerkung:
Mit dieser Definition werden pathologische Schrankenfunktionen ausgeschlossen.
Echte Komplexitätsfunktionen gibt es sehr viele:
z. B. $\log n$, $\log^2 n$, $n \cdot \log n$, n^2, $n^3 + 3n$, 2^n, $\sqrt{n}$, $n!$

Hilfssatz: Sind f, g echte Komplexitätsfunktionen, dann sind dies auch $f + g$, $f \cdot g$, 2^g.

Wir betrachten im folgenden stets echte Komplexitätsfunktionen.

Definition 1.15:
Eine (N)TM M heißt **präzise**
$\underset{def.}{=}$ es existieren Funktionen f und g, so daß für alle $n > 0$ und für alle Eingaben w, $|w| = n$, gilt:
M hält in genau $f(n)$ Schritten und zu jedem Zeitpunkt der Berechnung werden nicht mehr als $g(n)$ Speicherplätze benutzt.

Satz 1.6: Sei M eine (N)TM, die L in der Zeit / im Raum $f(n)$ akzeptiert.
Dann existiert eine präzise (N)TM M', die L in der Zeit / im Raum $\mathcal{O}(f(n))$ akzeptiert.

Beweis: Wir können die deterministische Zeit, den deterministischen Raum, die nichtdeterministische Zeit sowie den nichtdeterministischen Raum simultan betrachten.

- M' startet zunächst M_f auf der Eingabe w.
- Wenn M_f stoppt, dann benutzt M' die Ausgabe von M_f als "**Meßlatte**" der Länge $f(|w|)$.
- Dann wird M simuliert durch M'.
- Sobald die Meßlatte erreicht wird (Zeit / Raum), stoppt M'.

□

Bemerkung: Wir betrachten die Komplexitätsklassen

$$TIME(f),\ SPACE(f),\ NTIME(f),\ NSPACE(f)$$

stets unter der Voraussetzung, daß f eine echte Komplexitätsfunktion darstellt.

Wichtige Klassen:

$$\begin{aligned}
\mathcal{P} &= \bigcup_{j\geq 0} TIME(n^j)\\
\mathcal{NP} &= \bigcup_{j\geq 0} NTIME(n^j)\\
\mathcal{PSPACE} &= \bigcup_{j\geq 0} SPACE(n^j)\\
\mathcal{NPSPACE} &= \bigcup_{j\geq 0} NSPACE(n^j)\\
\mathcal{EXP} &= \bigcup_{j\geq 0} TIME(2^{n^j})\\
\mathcal{L} &= SPACE(\log n)\\
\mathcal{NL} &= NSPACE(\log n)
\end{aligned}$$

1.6.1 Komplementäre nichtdeterministische Komplexitätsklassen

Definition 1.16: Sei $L \subseteq \Sigma^*$
Dann ist $\overline{L} := \Sigma^* - L$ "Komplement von L"

Definition 1.17: Sei D ein Entscheidungsproblem.
"D-Komplement" ist das Entscheidungsproblem, bei dem stets "yes" geantwortet wird, wenn D "no" antwortet.

Beispiel 1.10:

- SAT**-Komplement**
  geg.: ϕ als KNF
  Frage: Ist ϕ unerfüllbar?

- $HAMILTON'scher\ Weg$ **- Komplement**
  geg.: Graph $G = (V, E)$
  Frage: Gibt es in G **keinen** $HAMILTON'schen\ Weg$?

Definition 1.18: Sei K eine Komplexitätsklasse.
Dann ist

$$co\ K := \{\overline{L} \mid L \in K\}$$

Hilfssatz: Ist K eine deterministische Komplexitätsklasse, dann gilt:

$$K = co\ K$$

("K ist abgeschlossen bzgl. Komplementbildung")

Beweis: trivial: Die akzeptierende TM M muß lediglich die "*yes*"- und "*no*"-Antworten vertauschen. □

Große Frage: Sind nichtdeterministische Komplexitätsklassen abgeschlossen gegenüber der Komplementbildung?

1.7 Hierarchietheoreme

Wir wollen zeigen, daß eine TM mit mehr Zeit / Raum tatsächlich mehr berechnen kann als eine TM mit weniger Zeit / Raum.

Sei dazu $f(n) \geq n$ eine echte Komplexitätsfunktion und

$$H_f := \{M; w \mid M \text{ akzeptiert die Eingabe } w \text{ nach höchstens } f(|w|) \text{ Schritten}\}$$

Bemerkung: H_f ist eine zeitbeschränkte Version des Halteproblems.

Lemma 1.1: $H_f \in TIME(f(n)^3)$

Beweis: Wir beschreiben eine TM U_f mit vier Bändern, die H_f in der Zeit $f(n)^3$ akzeptiert. U_f benutzt dabei

- eine universelle TM U ($\rightarrow$ Berechnungstheorie)
- einen Einband-Simulator für eine k-Band-TM
- eine lineare Speed-up Maschine
- die präzise TM M_f

U_f startet M_f auf dem zweiten Teil der Eingabe und richtet auf dem vierten Band eine Meßlatte $\sqcap^{f(|w|)}$ ein. M_f arbeitet dabei in der Zeit $\mathcal{O}(f(|w|))$.

Jetzt kopiert U_f den ersten Teil der Eingabe M auf das dritte Band und schreibt den zweiten Teil der Eingabe w als $\triangleright w$ auf das erste Band. Auf das zweite Band wird die Kodierung des Initialzustandes s von M geschrieben.

Es wird geprüft, ob M eine korrekte Beschreibung einer TM ist; falls nicht, so wird gestoppt.

Zeitbedarf: $\mathcal{O}(f(|w|) + n) = \mathcal{O}(f(|w|))$

U_f simuliert Schritt für Schritt M auf der Eingabe w analog zur Arbeit der universellen TM U unter Beachtung der Meßlatte.

Zeitbedarf für einen Schritt: $\mathcal{O}(l_M \cdot k_M^2 \cdot f(|w|))$
mit l_M - Länge der Beschreibung der Zustände bzw. Symbole von M
k_M - Zahl der Bänder von M
$\rightarrow \mathcal{O}(f(|w|)^2)$

U_f akzeptiert $M; w$, falls M die Eingabe w in der Zeit $\mathcal{O}(f(|w|))$ akzeptiert.
$\Rightarrow$ Zeitbedarf $\mathcal{O}(f(|w|)^3)$

Dieser kann mit dem linearen Speed-up Theorem auf $f(|w|)^3$ gedrückt werden.

$\square$

Lemma 1.2: $H_f \notin TIME(f(\lfloor \frac{n}{2} \rfloor))$

Beweis (durch Diagonalisierung):
Annahme: Es existiert eine TM R_{H_f}, die H_f in der Zeit $f(\lfloor \frac{n}{2} \rfloor)$ akzeptiert.

Wir können dann eine TM D_f konstruieren mit

$$D_f(M) := \begin{cases} \text{"}no\text{"} & : \quad R_{H_f}(M; M) = \text{"}yes\text{"} \\ \text{"}yes\text{"} & : \quad sonst \end{cases}$$

D_f braucht auf M die gleiche Zeit wie R_{H_f} auf $M; M$, d. h. die Zeit $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$

Was ist $D_f(D_f)$?

Annahme: $D_f(D_f) = \text{"}yes\text{"}$
$\Rightarrow R_{H_f}(D_f; D_f) = \text{"}no\text{"}$
$\Rightarrow D_f; D_f \notin H_f$
$\Rightarrow D_f$ akzeptiert D_f nicht in der Zeit $f(n)$
$\Rightarrow D_f$ verwirft D_f in der Zeit $f(n)$ (präzise TM)
$\Rightarrow D_f(D_f) = \text{"}no\text{"}$ ↯

$\Rightarrow D_f(D_f) = \text{"}no\text{"}$
Analog ergibt sich dann $D_f(D_f) = \text{"}yes\text{"}$ ↯

$\Rightarrow H_f \notin TIME(f(\lfloor \frac{n}{2} \rfloor))$

$\square$

Unter Verwendung der beiden vorherigen Lemmata erhält man folgendes

Theorem ("Zeit-Hierarchietheorem"):
Ist $f(n) \geq n$ eine echte Komplexitätsfunktion, dann gilt:

$$TIME(f(n)) \subsetneq TIME(f(2n+1)^3)$$

Bemerkung: Es gilt sogar

$$TIME(f(n)) \subsetneq TIME(f(n) \cdot \log^2 f(n))$$

Wichtig: Ist $f(n)$ polynomial, so auch $f(2n+1)^3$.

Korollar: $\mathcal{P} \subsetneq \mathcal{EXP}$

Beweis:

$\subseteq$: $P \subseteq TIME(2^n)$, da 2^n schneller wächst als jedes Polynom

$$\begin{aligned} \subsetneq : P &\subseteq TIME(2^n) \\ &\subsetneq TIME((2^{2n+1})^3) \text{ Zeit-Hierarchietheorem für } f(n) = 2^n \\ &\subseteq \mathcal{EXP} \end{aligned}$$

□

Ein analoges Ergebnis kann für den Speicherplatz bewiesen werden:

Theorem ("Raum-Hierarchietheorem"):
Ist $f(n)$ eine echte Komplexitätsfunktion, dann gilt:

$$SPACE(f(n)) \subsetneq SPACE(f(n) \cdot \log f(n))$$

Achtung: Bei der Arbeit mit Schrankenfunktionen, die keine echten Komplexitätsfunktionen sind, kann folgendes passieren:

Satz 1.7 ("Gap Theorem"):
Es gibt eine **rekursive** Funktion $f : I\!N \to I\!N$, so daß gilt:

$$TIME(f(n)) = TIME(2^{f(n)})$$

Beweis: Wir definieren f so, daß keine TM, die auf einer Eingabe der Länge n arbeitet, nach $f(n) \leq x \leq 2^{f(n)}$ Schritten stoppt; d. h. eine TM hält entweder in $x < f(n)$ Schritten, in $x > 2^{f(n)}$ Schritten oder aber überhaupt nicht.

Wir betrachten alle TM in einer z. B. lexikographischen Ordnung bzgl. ihrer Kodierung

$M_1, M_2, \ldots$. Dabei kann eine Maschine aus dieser Liste auf einigen bzw. sogar allen Eingaben nicht halten.

Nun definieren wir das Prädikat $P(i,k)$ für $i,k \geq 0$:
"Jede TM $M_1, M_2, \ldots, M_i$ hält auf jeder Eingabe der Länge i entweder nach $x < k$ Schritten, $x > 2^k$ Schritten oder überhaupt nicht."

$P(i,k)$ kann entschieden werden, auch wenn eine der Maschinen auf bestimmten Eingaben nicht hält:
Dazu können wir alle Maschinen $M_1, M_2, \ldots, M_i$ auf allen Eingaben w der Länge $|w| \leq i$ für maximal $2^k + 1$ Schritte simulieren.

Wir definieren Werte für $f(i)$ für einige $i \geq 0$. Dazu betrachten wir die folgende Reihe von Werten k: $k_0 = 2i$ und für alle $j = 1, 2, \ldots$: $k_j = 2^{k_{j-1}} + 1$.

Sei $N(i) = \sum_{j=0}^{i} |\Sigma_j|^i$ die Anzahl der möglichen Eingaben der Länge i für die ersten $i+1$ Maschinen.

Da jede solche Eingabe $P(i, k_j)$ für höchstens genau einen Wert von k_j nicht erfüllt, muß es eine Zahl $l \leq N(i)$, $l \in \mathbb{N}$, geben, für die $P(i, k_j)$ wahr ist.

Jetzt können wir $f(i)$ durch $f(i) = k_l$ definieren ($\Rightarrow f(i)$ wächst ziemlich schnell).

Sei $L \in TIME(2^{f(n)})$ eine Sprache.

L kann von einer TM (z. B. M_j) in der Zeit $2^{f(n)}$ entschieden werden.
Dann ist es für eine Eingabe w der Länge $|w| \geq j$ nicht möglich, daß M_j nach $f(|w|) \leq x \leq 2^{f(|w|)}$ Schritten hält, da wir den Funktionswert $f(n)$ für $n \geq j$ bei der Betrachtung von M_j gerade so definiert haben.
Da M_j nach spätestens $2^{f(|w|)}$ Schritten angehalten haben muß, können wir folgern, daß M_j nach höchstens $f(|w|)$ Schritten angehalten haben muß.

Bemerkung: Es gibt endlich viele Eingaben w, $|w| < j$, für die wir nicht sagen können, wann M_j hält. Aber wir können M_j so modifizieren (indem wir die Zustandsmenge entsprechend erhöhen), daß die resultierende Maschine M_j' alle diese endlich vielen Eingaben in $t < k_1 = 2|w|$ entscheidet.

Es folgt: $L \in TIME(f(n))$ und somit $TIME(f(n)) = TIME(2^{f(n)})$. □

1.8 Die Erreichbarkeitsmethode

Wir wollen den Zusammenhang zwischen deterministischen und nichtdeterministischen Komplexitätsklassen aufklären.

Satz 1.8: Sei $f(n)$ eine echte Komplexitätsfunktion. Dann gilt:

$$SPACE(f(n)) \subseteq NSPACE(f(n))$$

$$TIME(f(n)) \subseteq NTIME(f(n))$$

Beweis:
trivial: Jede deterministische Berechnung ist auch nichtdeterministisch. □

Satz 1.9: Sei $f(n)$ eine echte Komplexitätsfunktion. Dann gilt:

$$NTIME(f(n)) \subseteq SPACE(f(n))$$

Beweis: Sei $L \in NTIME(f(n))$.
Dann existiert eine präzise NTM M, die L in der Zeit $f(n)$ akzeptiert.
Wir konstruieren eine TM M', die M im Raum $f(n)$ simuliert (basierend auf der Idee aus Satz 1.5)

M' erzeugt eine Folge nichtdeterministischer Entscheidungen von M, also eine $f(n)$-lange Folge von $i \in \{0, \ldots, d-1\}$, wobei d den Grad des Nichtdeterminismus angibt.

M' simuliert M auf dieser Folge spezieller nichtdeterministischer Entscheidungen. M' benötigt dazu lediglich den Platz $f(n)$, da in der Zeit $f(n)$ höchstens $f(n)$ Felder beschrieben werden können und $f(n)$-lange Folgen ebenfalls lediglich $\mathcal{O}(f(n))$ Platz benötigen.

Bevor die nächste Folge (mittels Addition von 1) von nichtdeterministischen Entscheidungen simuliert wird, wird die aktuelle Bandinschrift gelöscht.

Da M präzise ist, kann die Anfangsfolge $0^{f(n)}$ leicht konstruiert werden.

□

Satz 1.10: Sei $f(n)$ eine echte Komplexitätsfunktion. Dann gilt:

es existiert eine Konstante c , so dass $NSPACE(f(n)) \subseteq TIME(c^{\log n + f(n)})$

Beweis:
Sei M eine k-Band-NTM mit Ein- und Ausgabe, die L im Raum $f(n)$ entscheidet.
Von den Konfigurationen $(q, w_1, u_1, \ldots, w_k, u_k)$ von M sind lediglich die folgenden Anteile interessant:
$(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$, wobei i die Position auf dem Eingabeband (read-only) angibt.

Es gilt: $i \leq n = |w|$ und $|w_j u_j| \leq f(n)$ $(j = 2, \ldots, k-1)$
$\Rightarrow$ Es gibt höchstens $|K| \cdot (n+2) \cdot |\Sigma|^{(2k-2) \cdot f(n)} = c_1^{\log n + f(n)}$ verschiedene solche Konfigurationen mit $c_1 = c_1(M)$.

Wir betrachten den **Konfigurationengraph** $G(M; w)$, wobei folgende Zuordnungen bestehen:
Knoten $\widehat{=}$ Konfigurationen c_i
Kanten $\widehat{=}$ (c_i, c_j), falls $c_i \vdash c_j$

Es gilt: $w \in L \quad \Leftrightarrow$ In $G(M;w)$ existiert eine Weg von der initialen Konfiguration $c_0 = (s, 0, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$ zu einer Konfiguration c mit $c = (\text{"}yes\text{"}, \ldots)$.

Wir haben so unser Simulationsproblem reduziert auf die Frage nach der *Erreichbarkeit* bestimmter Knoten in einem Graph mit $c_1^{\log n + f(n)}$ Knoten.

Uns ist ein Polynomialzeitalgorithmus für die $Graph - Erreichbarkeit$ bekannt.

Das *Erreichbarkeitproblem* für einen Graph mit m Knoten kann in der Zeit $c_2 \cdot m^2$ gelöst werden.
$\Rightarrow$ Die Simulation kann in der Zeit $c_2 \cdot c_1^{2\cdot(\log n + f(n))}$ erfolgen.
Mit $c := c_2 \cdot c_1^2$ folgt die Behauptung.

□

Bemerkungen:

- Die angewandte Erreichbarkeitsmethode ist auch in anderen Zusammenhängen erfolgreich ($\rightarrow$ später)

- (a) Wir können die *Erreichbarkeit* auf einer **expliziten** Beschreibung des Konfigurationengraphen (Adjazenzmatrix $A_{G(M;x)}$) lösen.
  Dazu benötigen wir Zeit zur expliziten Konstruktion von $G(M;x)$ und Speicherplatz zur Speicherung von $G(M;x)$

  (b) Wir können die *Erreichbarkeit* aber auch auf einer **impliziten** Beschreibung von $G(M;w)$ lösen:
  Steht $(c, c') \in G(M;w)$ zur Frage, dann wird eine Routine gestartet, die feststellt, ob c' aus c erreichbar ist.
  Die Routine ist sehr einfach - lediglich bei der "Start"-Konfiguration muß etwas gearbeitet werden (die "Start"-Konfiguration enthält nur die Lesekopfkonfiguration des Eingabebandes und nicht das gelesene Symbol; ist aber die Lesekopfposition klar, dann kann das entsprechende Eingabesymbol leicht gefunden werden)

Korollar:

$$\boxed{\underbrace{\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}}_{\subsetneq\ (Hierarchietheorem)}}$$

Allgemeine Annahme: Jede Inklusion ist echt (d. h. $\subseteq \rightarrow \subsetneq$)

1.8.1 Nichtdeterministischer Raum: Satz von Savitch (1970)

Wir wollen eine weitere Anwendung der Erreichbarkeitsmethode vorstellen.

Aus dem letzten Satz wissen wir, daß

$$NSPACE(f(n)) \subseteq SPACE(c^{\log n + f(n)})$$

Gibt es eine bessere Simulation von nichtdeterministischem Raum als diese exponentielle Konstruktion?

Satz 1.11 (Theorem von Savitch):

$$REACHABILITY \in SPACE(\log^2 n)$$

Beweis: Sei $G = (V, E)$ Graph, $\#V = n$.
Seien ferner $x, y \in V$ und $i \geq 0$.

"$PATH(x, y, i)$": In G existiert ein Weg von x nach y der Länge $\leq 2^i$

Offensichtlich gilt:

$$PATH(x, y, i) \text{ wahr für ein } i \quad \Leftrightarrow \quad PATH(x, y, \lceil \log n \rceil) \text{ wahr}$$

Daraus folgt:

$$REACHABILITY(G, x, y) \text{ wahr} \quad \Leftrightarrow \quad PATH(x, y, \lceil \log n \rceil) \text{ wahr}$$

Wir bauen eine TM M mit Ein- und Ausgabe und zwei Arbeitsbändern, die $PATH(x, y, i)$ entscheidet:
Die Eingabe von M ist die Adjazenzmatrix von $G = (V, E)$
Das erste Band enthalte das Tripel (x, y, i) binär kodiert.

1. Fall: $i = 0$:
Da die Weglänge $\leq 2^0 = 1$ ist, können wir $PATH(x, y, i)$ berechnen durch Testen, ob $x = y$, oder durch Nachsehen in der Adjazenzmatrix, ob $(x, y) \in E$.

2. Fall: $i \geq 1$:
Wir berechnen rekursiv folgendes:

Wir testen für alle $z \in V$, ob $PATH(x, z, i-1)$ **und** $PATH(z, y, i-1)$ gilt.
(Es gibt einen Weg von x nach y der Länge $\leq 2^i$, falls es einen Zwischenpunkt z, einen Weg von x nach z der Länge $\leq 2^{i-1}$ und einen Weg von z nach y der Länge $\leq 2^{i-1}$ gibt.)
Um alle möglichen Zwischenpunkte z zu bestimmen, nutzen wir das dritte Band: Bei Erzeugen eines neuen z (jeweils um 1 hochzuzählen) wird das Tripel $(x, z, i-1)$ auf das erste Arbeitsband geschrieben und bearbeitet.

Wird eine negative Antwort erreicht, so wird das Tripel $(x, z, i-1)$ gestrichen und ein neues z bearbeitet.
Ist die Antwort positiv, dann wird $(x, z, i-1)$ gestrichen und $(z, y, i-1)$ geschrieben sowie bearbeitet (hierbei kann y aus dem linken Nachbartripel abgeschrieben werden).

Jetzt gilt:
Ist die Antwort negativ, so wird das nächste z ausprobiert.
Ist die Antwort positiv, dann wird das Tripel gestrichen und eine positive Antwort auf die Frage $PATH(x, y, i)$ gegeben.

Bemerkung: Das erste Arbeitsband ist als STACK organisiert.

Offenbar leistet der Algorithmus das Gewünschte.

Das erste Arbeitsband enthält gleichzeitig höchstens $\lceil \log n \rceil$ viele Tripel, wobei die Länge der Tripel $\leq 3 \cdot \lceil \log n \rceil$ ist.
Auf dem zweiten Arbeitsband steht jeweils ein Knoten von z.
$\Rightarrow$ Raumbedarf $\leq \mathcal{O}(\lceil \log n \rceil^2)$

□

Korollar: Falls $f(n) \geq \log n$ eine echte Komplexitätsfunktion ist, dann gilt:

$$NSPACE(f(n)) \subseteq SPACE(f(n)^2)$$

Beweis:
Wir setzen den Erreichbarkeitsalgorithmus von Savitch auf dem Konfigurationengraphen und auf die Eingabe x einer $f(n)$-raumbeschränkten NTM an.
Der Konfigurationsgraph hat die Größe $c^{f(n)}$.
$\Rightarrow$ In der Lösung ist $\mathcal{O}(f(n)^2)$ Raum ausreichend.

□

Korollar:

$$\mathcal{NPSPACE} = \mathcal{PSPACE}$$

1.8.2 Nichtdeterministischer Raum: Satz von Immerman / Szelepczényi

Wir wollen die Abgeschlossenheit nichtdeterministischer Komplexitätsklassen bzgl. der Komplementbildung nachweisen:

$$\longrightarrow \begin{array}{l} \mathcal{NL} \stackrel{?}{=} co\,\mathcal{NL} := \{L \mid \overline{L} \in \mathcal{NL}\} \\ \mathcal{NP} \stackrel{?}{=} co\,\mathcal{NP} := \{L \mid \overline{L} \in \mathcal{NP}\} \end{array}$$

Bemerkung: Diese Fragestellung ist vergleichbar mit dem LBA-Problem.

Wir **betrachten** eine NTM M, die die Funktion $F(x)$ berechnet; dabei ist M eine NTM mit Ein- und Ausgabe.

Alle erfolgreichen Berechnungen enden mit der Ausgabe $F(w)$ bei der Eingabe w; die anderen Berechnungen enden mit "no".

Wenigstens eine Berechnung muß erfolgreich sein.

M ist $f(n)$-raumbeschränkt, falls sämtliche Berechnungen bei der Eingabe w nicht mehr als $f(|w|)$ Platz benötigen.

Satz 1.12 (Immerman/Szelepczényi 1987/88):
Sei $G = (V, E)$ Graph, $\#V = n$, $v \in V$ beliebig.
Dann kann die Zahl der von v in G erreichbaren Knoten von einer **N**TM mit Ein- und Ausgabe im Raum $\log n$ berechnet werden.

Beweis: Der Algorithmus hat vier Schleifen:

1. Die äußere Schleife berechnet iterativ $\#S(1), \#S(2), \ldots, \#S(n-1)$, wobei $\#S(i)$ die Zahl der Knoten von G angibt, die von v in höchstens i Schritten erreicht werden können.

   $\#S(n-1)$ ist das gesuchte Ergebnis.

   Ist $\#S(k-1)$ berechnet, so wird die Berechnung von $\#S(k)$ gestartet:

   ```
   #S(0) := 1;
   For k = 1, 2, ..., n-1 do
      Compute #S(k) aus #S(k-1)
   ```

2. Berechne $\#S(k)$ aus $\#S(k-1)$:
   Sei l ein Zähler initialisiert mit 0.
   Wir untersuchen nacheinander alle Knoten u von G in aufsteigender Ordnung.
   Gilt: $u \in S(k)$, so wird l erhöht.

   ```
   l := 0;
   For u = 1, ... n do
      If u ∈ S(k) then l := l+1;
   ```

3. Wie wird $u \in S(k)$ entschieden?

   Wir gehen über alle Knoten w von G (in aufsteigender Ordnung).

   Ist $w \in S(k-1)$, dann wird der Zähler m der Elemente aus $S(k-1)$ um 1 erhöht; anschließend wird überprüft, ob $u = w$ bzw. $(u, w) \in E$ ist.

   Ist die Antwort positiv, so folgt: $u \in S(k)$; die entsprechende $reply$-Variable wird dann "$true$" gesetzt.
   Ist am Ende nicht $u \in S(k)$ gezeigt, so wird $u \notin S(k)$ ausgegeben.

Beispiel: $\#S(k-1)$ ist bekannt, $\#S(k)$ soll berechnet werden:

$$\begin{array}{lll} \textit{Zähler}: m & \quad \ldots w_3 w_2\ w_1 & \in S(k-1) \\ \qquad\quad l & \qquad\qquad u & \stackrel{?}{\in} S(k) \end{array}$$

Falls m am Ende nicht $\#S(k-1)$ erreicht (dies kann wegen der nichtdeterministischen Wahl von $w \in S(k-1)$ vorkommen), so geben wir auf und sagen "*no*".

```
m := 0; reply := false;
Für jeden Knoten w = 1, 2, ..., n repeat
   If w ∈ S(k-1) then m := m+1;
     If weiterhin u = w oder (u,w) ∈ G then
       reply := true;
If am Ende m < #S(k-1)
  then "no"
  else return reply;
```

4. Wie wird $w \in S(k-1)$ ermittelt?

   Unter Ausnutzung des Nichtdeterminismus wird wie bei der "$\log n$"-Berechnung von *REACHABILITY* vorgegangen:
   Sei $v \in G$.
   Wir raten $k-1$ Knoten in G und überprüfen, ob von v über diese $k-1$ Knoten ein Weg zu w existiert.

```
w0 := v;
For p = 1, ..., k-1 do
   Rate Knoten wp;
   If (wp-1,wp) ∈ G then
     If wp = w
       then w ∈ S(k-1);
       else aufgeben;
```

Raumanalyse:
Wir haben

- 10 Variablen: $k, S(k-1), l, u, m, v, w, p, w_p, w_{p-1}$
- eine NTM mit Ein- und Ausgabestring

Wir können alles im Raum $\log n$ speichern.

Korrektheit des Algorithmus:

Behauptung: Für alle k wird $\#S(k)$ richtig berechnet.

Wir nehmen an, daß die Behauptung für $k = 0$ gilt und betrachten $\#S(k)$ für $k \geq 1$ im Falle einer erfolgreichen Berechnung (d. h. es gilt **nie**: $m < \#S(k-1)$)

g.z.z.: l wird nur erhöht, wenn $u \in S(k)$

Wir erhöhen l nur, wenn $m = \#S(k-1)$.
$\Rightarrow$ Alle $w \in S(k-1)$ werden verifiziert.
"*reply*" antwortet korrekt $u \in S(k)$ und l wird nur erhöht, falls $reply = true$ ist.

Schließlich ist klar, daß wenigstens eine erfolgreiche Berechnung existiert (d. h. eine Berechnung, die $\#S(k-1)$ korrekt zählt).

□

Korollar: Ist $f(n) \geq \log n$ eine echte Komplexitätsfunktion, dann gilt:

$$NSPACE(f(n)) = co\, NSPACE(f(n))$$

Beweis: Sei $L \in NSPACE(f(n))$.
Ferner sei M eine $f(n)$-raumbeschränkte NTM, die L akzeptiert.

Wir konstruieren eine $f(n)$-raumbeschränkte NTM $\overline{M}$, die $\overline{L}$ akzeptiert:

Bei der Eingabe w startet $\overline{M}$ den Algorithmus von Savitch auf dem Konfigurationengraph $G(M;w)$ von M.
Jedesmal, wenn $\overline{M}$ Verbindungen zwischen zwei Konfigurationen testen soll, wird die Übergangsrelation von M konsultiert.
Wenn $\overline{M}$ eine akzeptierende Konfiguration in $S(k)$ findet, dann verwirft $\overline{M}$. Wird $\#S(n-1)$ berechnet und enthält $S(n-1)$ keine akzeptierende Konfiguration, dann akzeptiert $\overline{M}$ die Eingabe w.

□

Kapitel 2

Die Klassen $\mathcal{P}$ und $\mathcal{NP}$

2.1 Reduktion

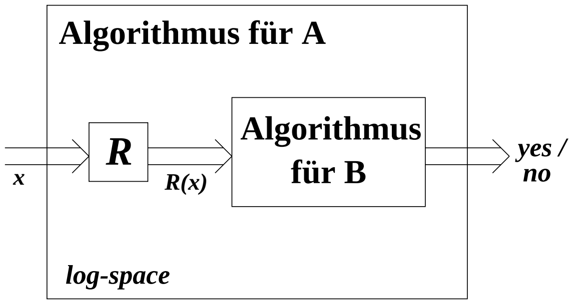
$\mathcal{NP}$ ist eine unendliche Menge von Sprachen und Problemen.

Wir wollen die Komplexität verschiedener Probleme aus $\mathcal{NP}$ miteinander vergleichen, um auszudrücken, ob ein Problem "leichter" ist als ein anderes.

Ein zentrales Konzept hierbei bilden "**Reduktionen**":

- B ist mindestens so schwer wie A, falls A **auf** B **reduzierbar**
- A ist genau dann auf B **reduzierbar**, wenn eine Transformation R existiert mit

$$x \in A \quad \Leftrightarrow \quad R(x) \in B$$



Wir wollen sagen können: "A ist höchstens so schwer wie B".
"B" ist mindestens so schwer wie "A".

Dazu ist es nötig, die Rechnungskraft von R zu beschränken.

Definition 2.1 ($\log space$ **beschränkte Reduktion):**
$L_1 \subseteq \Sigma_1^*$ ist reduzierbar auf $L_2 \subseteq \Sigma_2^*$, falls eine Funktion $R : \Sigma_1^* \to \Sigma_2^*$ existiert mit den Eigenschaften

(1) $x \in L_1 \quad \Leftrightarrow \quad R(x) \in L_2$

(2) R kann von einer $\log n$ -raumbeschränkten TM berechnet werden.

R heißt "$\log n$ Reduktion" oder "$\log space$ Reduktion" von L_1 auf L_2.

Bezeichnung: $L_1 \leq_{\log n} L_2$ bzw. $L_1 \leq L_2$

Satz 2.1: Wird eine $\log space$ Reduktion von einer TM M berechnet, dann hält M bei allen Eingaben in polynomialer Zeit.

Beweis: Berechne M die Reduktion R.
Dann hat M $\mathcal{O}(n \cdot c^{\log n})$ viele Konfigurationen bei einer Eingabe w mit $|w| = n$.
Da M deterministisch ist, kann keine Konfiguration zweimal erreicht werden (sonst würde M unendliche Zyklen durchlaufen).
$\Rightarrow$ Die Berechnung benötigt höchstens $\mathcal{O}(n^k)$ Schritte (wobei $k \in I\!N$).

$\square$

Beispiel 2.1: $HAMILTON\ PATH \leq SAT$

$HAMILTON\ PATH$
    geg.: ein Graph $G = (V, E)$.
    Frage: Existiert ein Weg durch alle Knoten von G, der jeden Knoten genau einmal durchläuft?

SAT
    geg.: ein Boolescher Ausdruck in KNF.
    Frage: Existiert eine erfüllende Belegung?

(Wir wissen bereits: $HAMILTON\ PATH$, $SAT \in \mathcal{NP}$)

Wir suchen eine Reduktion von $HAMILTON\ PATH$ auf SAT:
Dazu konstruieren wir zu einem Graphen G einen Booleschen Ausdruck $R(G)$ mit

$$R(G) \text{ erfüllbar} \quad \Leftrightarrow \quad G \text{ hat einen } HAMILTON'schen\ Weg$$

Sei $G = (\{1, \ldots, n\}, E)$.
Wir bauen $R(G)$ aus n^2 Booleschen Variablen $x_{ij},\ 1 \leq i, j \leq n$.
(Semantik von x_{ij}: j ist i-ter Knoten auf dem $HAMILTON\ Weg$)

$R(G)$ ist eine KNF mit folgenden Klauseln:

(1) $\forall\, 1 \leq j \leq n : (x_{1,j} \ \vee \ x_{2,j} \ \vee \ \ldots \ \vee \ x_{n,j})$
- j kommt auf dem Weg vor.

(2) $\forall\, 1 \leq i,j,k \leq n \ (i \neq k) : (\neg x_{i,j} \ \vee \ \neg x_{k,j})$
- j kommt nicht gleichzeitig an i-ter und k-ter Stelle vor.

(3) $\forall\, 1 \leq i \leq n : (x_{i,1} \ \vee \ x_{i,2} \ \vee \ \ldots \ \vee \ x_{i,n})$
- wenigstens ein Knoten kommt an i-ter Stelle vor.

(4) $\forall\, 1 \leq i,j,k \leq n \ (j \neq k) : (\neg x_{i,j} \ \vee \ \neg x_{i,k})$
- auf der i-ten Position können nicht zwei verschiedene Knoten stehen.

(5) $\forall\, 1 \leq i,j,k \leq n \ ((i,j) \notin G) : (\neg x_{k,i} \ \vee \ \neg x_{k+1,j})$
- ist $(i,j) \notin G$, so kann i nicht unmittelbar links vom Knoten j stehen.

$\Rightarrow R(G) = (1) \ \wedge \ (2) \ \wedge \ (3) \ \wedge \ (4) \ \wedge \ (5)$

Behauptung: R ist eine Reduktion von $HAMILTON\ PATH$ auf SAT

z.z.: (1) $R(G)$ hat eine erfüllende Belegung $\quad\Leftrightarrow\quad$ G hat einen $HAMILTON'schen\ Weg$.
(2) R kann im Raum $\log n$ berechnet werden.

zu (1):
"$\Rightarrow$": Sei T eine erfüllende Belegung von $R(G)$
$\Rightarrow$ T ist eine erfüllende Belegung für alle Klauseln
$\Rightarrow$ Für alle j existiert **genau** ein i mit $T(x_{i,j}) = true$, ansonsten könnten nicht alle Klauseln $(x_{1,j} \ \vee \ \ldots \ \vee \ x_{n,j})$ und $(\neg x_{i,j} \ \vee \ \neg x_{k,j})$ erfüllt werden.
Für alle i existiert **genau** ein j mit $T(x_{i,j}) = true$, da alle Klauseln $(x_{i,1} \vee \ldots \vee x_{i,n})$ und $(\neg x_{i,j} \ \vee \ \neg x_{i,k})$ erfüllt sind.
$\Rightarrow$ T repräsentiert eine Permutation $\pi(1), \ldots, \pi(n)$ mit

$$\pi(i) = j \quad\Leftrightarrow\quad T(x_{i,j}) = true$$

Schließlich garantieren die Klauseln $(\neg x_{k,i} \ \vee \ \neg x_{k+1,j})$, daß $(\pi(k), \pi(k+1)) \in G$.
$\Rightarrow \pi(1), \ldots, \pi(n)$ ist ein $HAMILTON'scher\ Weg$.

"$\Leftarrow$": Ist $\pi(1), \ldots, \pi(n)$ ein $HAMILTON'scher\ Weg$ in G, d. h. ist π eine Permutation mit $(\pi(k), \pi(k+1)) \in G$ für alle k, dann erfüllt T mit

$$T(x_{i,j}) \quad\Leftrightarrow\quad \pi(i) = j$$

die KNF $R(G)$.

zu (2):
Sei $G = (V, E)$ die Eingabe.

Wir bauen eine TM M, die $R(G)$ konstruiert:

Zunächst werden die Namen $1, \ldots, n$ der Knoten von G binär verschlüsselt.

Unter Ausnutzung dieser Information generiert M die nicht von G abhängenden Klauseln. Das ist möglich mit drei Zählern i, j und k, mit deren Hilfe die Indizes ermittelt werden.

Die letzte Gruppe von Klauseln wird konstruiert durch sukzessive Generierung von $(\neg x_{k,i} \ \vee \ \neg x_{k+1,j})$ und durch Nachsehen, ob $(i, j) \in G$.

$\Rightarrow$ Die Berechnung von $R(G)$ benötigt den Raum $\mathcal{O}(\log n)$.

$\square$

Beispiel 2.2: *REACHABILITY* $\leq$ *CIRCUIT VALUE*

REACHABILITY
: bereits definiert

CIRCUIT VALUE
: **geg.:** ein Boolescher Schaltkreis über $\{\wedge, \vee, \neg, 0, 1\}$ mit konstanten Inputs (variablenfreier Schaltkreis)
: **Frage:** Wie lautet der Ausgabewert des Schaltkreises?

Beide Probleme sind (leicht) in Polynomialzeit lösbar.

Wir suchen eine Reduktion von *REACHABILITY* auf *CIRCUIT VALUE*:
Eingabe: ein Graph $G = (V = \{1, \ldots, n\}, E)$
ges.: ein variablenfreier Schaltkreis $R(G)$ mit

$$\text{Ausgabe von } R(G) \text{ ist } \mathit{true}. \quad \Leftrightarrow \quad \text{in } G \text{ existiert ein Weg von 1 nach } n.$$

Wir konstruieren $R(G)$:
Dazu betrachten wir Gatter vom Typ

$$g_{i,j,k} \text{ mit } 1 \leq i, j \leq n, \ 0 \ \leq k \leq n$$

$$h_{i,j,k} \text{ mit } 1 \leq i, j, k \leq n$$

Die Intuition hierbei ist:

$g_{i,j,k}$ ist wahr (1)	$\Leftrightarrow$	in G existiert ein Weg von i nach j, der nicht über einen Knoten $> k$ läuft
$h_{i,j,k}$ ist wahr (1)	$\Leftrightarrow$	in G existiert ein Weg von i nach j, ohne Zwischenknoten $> k$, der durch k geht

Sei $k = 0$:

Alle $g_{i,j,0}$ sind Gatter ohne Inputs, wobei

$$g_{i,j,0} = 1 \quad \Leftrightarrow \quad i = j \text{ oder } (i,j) \in E$$

Es existiert kein $h_{i,j,0}$.

Für $k = 1, \ldots, n$ gilt:

$h_{i,j,k}$ sind AND-Gatter mit den Eingängen $g_{i,k,k-1}$ und $g_{k,j,k-1}$.
$g_{i,j,k}$ sind OR-Gatter mit den Eingängen $g_{i,j,k-1}$ und $h_{i,j,k}$.

$g_{1,n,n}$ ist der Ausgabeknoten.
Offenbar ist $R(G)$ variablenfrei.

Behauptung: $R(G) = true \quad \Leftrightarrow \quad$ in G existiert ein Weg von 1 nach n.

Beweis durch Induktion über k:
$k = 0$: trivial
$k - 1 \to k$:

$$h_{i,j,k} = g_{i,j,k-1} \wedge g_{k,j,k-1}$$
$$g_{i,j,k} = g_{i,j,k-1} \vee h_{i,j,k}$$

$\Rightarrow$ $h_{i,j,k}$ und $g_{i,j,k}$ haben eine intuitive Bedeutung.

$\Rightarrow$ $g_{1,n,n}$ ist $true$ $\Leftrightarrow$ es existiert ein Weg in G von 1 nach n, der nicht über einen Knoten $> n$ läuft.

R kann mit $\log n$ Speicher berechnet werden:
Die Maschine geht mit Hilfe von drei Zählern über alle Indizes i, j, k und gibt die entsprechenden Kanten und Gatter aus.

$\Rightarrow$ R ist eine Reduktion von $REACHABILITY$ auf $CIRCUIT\ VALUE$.

$\square$

Beispiel 2.3: $CIRCUIT\ SAT \leq SAT$

$CIRCUIT\ SAT$

geg.: ein Boolescher Schaltkreis (SK) über $\{\wedge, \vee, \neg, 0, 1\}$ mit variablen Eingängen
Frage: Existiert eine erfüllende Belegung für die Eingänge des SK?

Sei C gegebener Schaltkreis. Wir suchen eine Formel $R(C)$ in KNF, so dass es für $R(C)$ eine erfüllende Belegung gibt, genau dann wenn es eine erfüllende Belegung für die Eingänge des SK gibt.

$R(C)$ enthält alle Variable von C und zusätzlich eine Variable g für jedes Gatter g von C. Für jedes Gatter des SK C konstruieren wir Klauseln in $R(C)$:

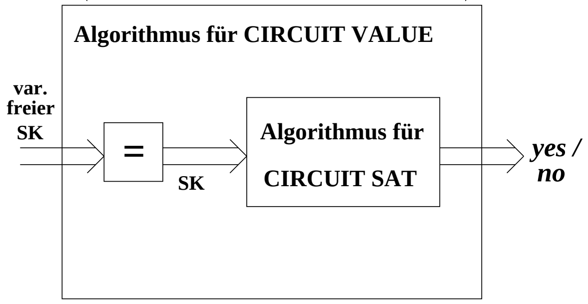
Gatter	SAT-Klausel
$g = true$	g
$g = false$	$\neg g$
$g = NOT(h)$	$(\neg g \vee \neg h)$ $(g \vee h)$
$g = x$	$(g \vee \neg x)$ $(\neg g \vee x)$
$g = OR(h, h')$	$(\neg h \vee g)$ $(\neg h' \vee g)$ $(h \vee h' \vee \neg g)$
$g = AND(h, h')$	$(\neg g \vee h)$ $(\neg g \vee h')$ $(\neg h \vee \neg h' \vee g)$

Wenn g ein Ausgangsgatter ist, dann hat $R(C)$ die Klausel (g).
Man kann die Reduktion $R(C)$ im Raum $\mathcal{O}(\log |C|)$ berechnen.

Beispiel 2.4 ("Reduktion durch Verallgemeinerung"):

$CIRCUIT\ VALUE \leq CIRCUIT\ SAT$

Offenbar ist jeder Schaltkreis mit konstanten Eingängen der Spezialfall eines Schaltkreises mit variablen Eingängen.
$\Rightarrow$ $CIRCUIT\ VALUE$ ist ein Spezialfall von $CIRCUIT\ SAT$.
$\Rightarrow$ Die Reduktion ist trivial (R ist die Identitätsfunktion).



Wir können Reduktionen auch hintereinanderschalten:

Satz 2.2: Sei R eine Reduktion von L_1 auf L_2 und sei R' eine Reduktion von L_2 auf L_3. Dann ist die Komposition $R' \circ R$ eine Reduktion von L_1 auf L_3.

Beweis: Es gilt offenbar:

$$x \in L_1 \quad \Leftrightarrow \quad R(x) \in L_2 \quad \Leftrightarrow \quad R'(R(x)) \in L_3$$

noch z.z.: $R' \circ R$ ist in $\log space$ berechenbar

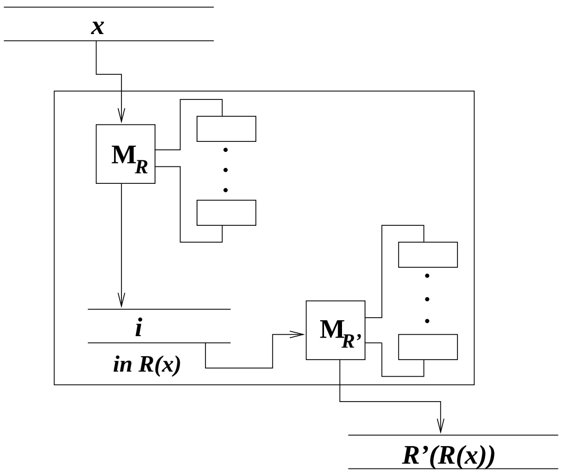
Idee: Wir schalten die Maschinen M_R und $M_{R'}$ hintereinander.

Problem: Die Ausgabe von M_R ist i.a. zu lang, um auf einem $\log n$ -Arbeitsband gespeichert werden zu können.

Ausweg: Wir speichern lediglich die Schreibkopfposition auf dem Ausgabeband M_R:
Will $M_{R'}$ das i-te Eingabesymbol (= i-tes Ausgabesymbol von M_R) lesen, so wird M_R gestartet und so lange arbeiten gelassen, bis das i-te Ausgabesymbol produziert ist.

$M_{R'} \circ M_R$ ist tatsächlich $\log n$ -raumbeschränkt, da zusätzlich zum Arbeitsspeicher von M_R und $M_{R'}$ lediglich die Schreibkopfposition auf dem Ausgabeband von M_R gespeichert werden muß.

Wir benötigen zusätzlich $\mathcal{O}(\log n)$ Speicher. □

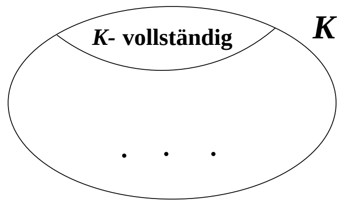


2.2 Vollständigkeit

Reduktionen sind reflexiv und transitiv.
⇒ Reduktionen definieren Halbordnungen auf der Menge aller Sprachen / Probleme.

Wir wollen die "schwersten" Sprachen / Probleme einer Klasse charakterisieren:

Definition 2.2: Sei K eine Komplexitätsklasse und $L \in K$.
L heißt **vollständig in** K bzw. K**-vollständig**
$\underset{def.}{=}$ für alle $L' \in K$ gilt: $L' \leq L$.



Eigenschaften vollständiger Probleme:

- Sie geben einen guten Aufschluß über die Natur einer Komplexitätsklasse.
- Sie dienen oft dazu, Unterschiede zwischen Komplexitätsklassen zu charakterisieren.
- Für sie ist es am unwahrscheinlichsten, daß sie zu einer kleineren Komplexitätsklasse gehören.

Definition 2.3:
Eine Komplexitätsklasse K ist abgeschlossen bzgl. Reduktion
$\underset{def.}{=}$ gilt $L \leq L'$ und $L' \in K$, dann folgt: $L \in K$

Satz 2.3: $\mathcal{P}$, $\mathcal{NP}$, $co\,\mathcal{NP}$, $\mathcal{L}$, $\mathcal{NL}$, $\mathcal{EXP}$, $\mathcal{PSPACE}$ sind alle abgeschlossen bzgl. einer $\log space$ Reduktion.

Beweis: Übungsaufgabe

$\square$

Korollar:

(1) Sei A $\mathcal{P}$-vollständig.

 (a) Gilt $A \in \mathcal{L} \;\Rightarrow\; \mathcal{L} = \mathcal{P}$
 (b) Gilt $A \in \mathcal{NL} \;\Rightarrow\; \mathcal{NL} = \mathcal{P}$

(2) Sei A $\mathcal{NP}$-vollständig.

 (a) Gilt $A \in \mathcal{L} \;\Rightarrow\; \mathcal{L} = \mathcal{NP}$
 (b) Gilt $A \in \mathcal{NL} \;\Rightarrow\; \mathcal{NL} = \mathcal{NP}$
 (c) Gilt $A \in \mathcal{P} \;\Rightarrow\; \mathcal{P} = \mathcal{NP}$

Beweis:

zu (1)(a): $\forall B \in \mathcal{P}$ gilt: $B \leq A$
Da $A \in \mathcal{L}$, $\mathcal{L}$ abgeschlossen bzgl. Reduktion
$\Rightarrow\; \forall B:\; B \in \mathcal{L}$
$\Rightarrow\; \mathcal{P} \subseteq \mathcal{L}$

□

Satz 2.4: Sind zwei Komplexitätsklassen K, K' abgeschlossen bzgl. Reduktion und existiert ein Problem L, das K-vollständig und K'-vollständig ist, dann gilt:

$$K = K'$$

2.2.1 Die Berechnungstabellen-Methode

Ein Werkzeug zum Nachweis der $\mathcal{P}$-Vollständigkeit und $\mathcal{NP}$-Vollständigkeit stellt die **Berechnungstabellenmethode** dar:

Sei dazu $M = (K, \Sigma, \delta, s)$ eine n^k-zeitbeschränkte TM für L.
Wir betrachten die Berechnung von M bei der Eingabe x als $|x|^k \times |x|^k$-Tabelle.

"Berechnungstabelle T von M für x"

Beispiel:

$$\underbrace{\begin{array}{cccccccc} \triangleright & 0_s & 1 & 1 & 0 & \sqcup & \sqcup & \sqcup \\ \triangleright & \triangleright & 1_s & 1 & 0 & \sqcup & \sqcup & \sqcup \\ \triangleright & \triangleright & 1 & 1_{q_0} & ⓪_{T_{3,5}} & \sqcup & \sqcup & \sqcup \\ \triangleright & \triangleright & 1 & 1 & 0_{q_0} & \sqcup & \sqcup & \sqcup \end{array}}_{|x|^k} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} |x|^k$$

Die Zeilen entsprechen dabei den Berechnungsschritten, die Spalten einer Bandposition.
In $T_{i,j}$ wird der Inhalt des Bandes an der j-ten Position zum Zeitpunkt i beschrieben.

Wir können die Tabelle zur Vereinfachung standardisieren.
Dazu betrachten wir lediglich:

(1) TM mit einem Band (wir können eine k-Band-TM in polynomialer Zeit durch eine 1-Band-TM simulieren).

(2) M hält bei der Eingabe x stets im Schritt $|x|^k - 2$.

(3) Wir füllen die Zeilen rechts mit "$\sqcup$" auf.

(4) Ist M im i-ten Schritt an der Position j, dann wird in $T_{i,j}$ nicht nur das Bandsymbol geschrieben, sondern auch der Zustand von M (unter Verwendung von Zeichen aus $\{\sigma_s \mid \sigma \in \Sigma,\ s \in K\}$).

(5) Die TM startet nicht auf $\triangleright$, sondern auf dem ersten Symbol der Eingabe.

(6) Der Lesekopf besucht nie die letzte Spalte am linken Tabellenrand (M würde dann nach rechts gehen; wir fassen daher beide Schritte zu einem zusammen).

(7) Hält M vor dem Erreichen des $(|x|^k - 2)$-ten Taktes, dann wird die Tabelle aufgefüllt durch Wiederholung(en) der letzten Zeile.

(8) M hält in der ersten Position des Bandes.

Die Berechnungstabelle heißt **akzeptierend**, falls gilt:

$$T_{|x|^k-1,1} = \sigma_{\text{"yes"}} = \text{"}yes\text{"}.$$

Es gilt:

M akzeptiert die Eingabe $x \quad \Leftrightarrow \quad$ die Berechnungstabelle $T(M,x)$ ist akzeptierend.

Satz 2.5: *CIRCUIT VALUE* ist $\mathcal{P}$-vollständig.

Beweis:

(1) Es gilt: $CIRCUIT\ VALUE \in \mathcal{P}$

(2) Sei $L \in \mathcal{P}$ beliebig.

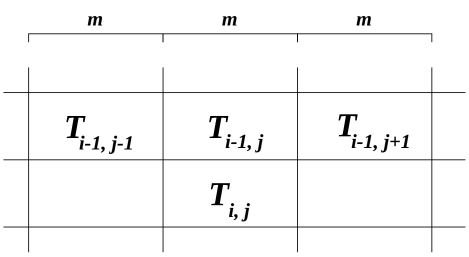
z.z.: Es gibt eine Reduktion R mit $L \leq CIRCUIT\ VALUE$.

Sei x die Eingabe.
Wir konstruieren einen variablenfreien Schaltkreis $R(x)$ mit

$$x \in L \quad \Leftrightarrow \quad R(x) \text{ ist } true.$$

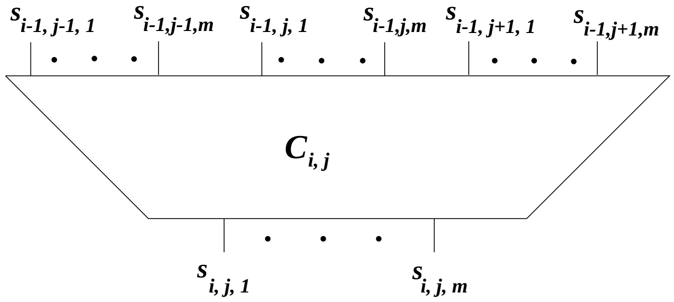
Sei M eine TM für L. M arbeite in der Zeit n^k.

Wir betrachten jetzt die Berechnungstabelle $T(M,x)$:
Für $i = 0$ oder $j = 0$ oder $j = |x|^k - 1$ sind die Werte $T_{i,j}$ bekannt. Wir betrachten daher die $T_{i,j}$ für die anderen i und j.
Dabei gibt der Wert $T_{i,j}$ den Bandinhalt der j-ten Stelle im i-ten Takt an.
$T_{i,j}$ kann lediglich abhängen von $T_{i-1,j-1}$, $T_{i-1,j}$ und $T_{i-1,j+1}$ (benachbarte Positionen).



Sei Γ die Menge aller möglichen Tabellensymbole $(\Sigma \cup \{\sigma_s \mid \sigma \in \Sigma,\ s \in K\})$.
Wir kodieren die $\gamma \in \Gamma$ als Vektoren $(s_1, \dots, s_m)$, mit $s_i \in \{0, 1\}$, $m = \lceil \log(\#\Gamma) \rceil$.
Nun haben wir eine Tabelle mit den binären Einträgen $s_{i,j,l}$, wobei $0 \leq i, j \leq n^{k-1}$, $1 \leq l \leq m$.

Jedes $s_{i,j,l}$ hängt von **höchstens** $3m$ Einträgen ab:
$s_{i-1,j-1,l'},\ s_{i-1,j,l'},\ s_{i-1,j+1,l'}$ mit $1 \leq l' \leq m$
$\Rightarrow$ Es gibt Boolesche Funktionen $F_1, \dots, F_m \in I\!B_{3m}$ mit
$s_{i,j,l} = F_l(s_{i-1,j-1,1}, \dots, s_{i-1,j-1,m}, s_{i-1,j,1}, \dots, s_{i-1,j,m}, s_{i-1,j+1,1}, \dots, s_{i-1,j+1,m})$ und $s_{i,j} = (F_1, \dots, F_m)$.



Da jede Boolesche Funktion durch einen Schaltkreis berechnet werden kann, existiert ein Schaltkreis $C_{i,j}$ für $s_{i,j}$, der aus der binären Kodierung von $T_{i-1,j-1}$, $T_{i-1,j}$ und $T_{i-1,j+1}$ die binäre Kodierung von $T_{i,j}$ berechnet.
$C_{i,j}$ hängt nur von M ab und hat eine **konstante**, von x unabhängige Größe.

Nun können wir Reduktionen konstruieren:

Sei dazu $L \in \mathcal{P}$ von M berechenbar.
Für jedes x besteht $R(x)$ aus $(|x|^k - 1) \cdot (|x|^k - 2)$ Schaltkreisen $C_{i,j}$ für jedes $T_{i,j}$ (wobei $i \geq 1,\ 1 \leq j \leq |x|^k - 2$).
Die Inputs für das Gatter $C_{i,j}$ sind die Outputs der Gatter $C_{i-1,j-1}$, $C_{i-1,j}$, und $C_{i-1,j+1}$; die Inputs für den Gesamt-Schaltkreis $R(x)$ sind die jeweiligen Inputs der Gatter der ersten Reihe sowie der ersten und letzten Spalte.
Das Ausgabegatter von $R(x)$ ist die letzte Ausgabe von $C_{|x|^k-1,1}$ (dort erhält man *"yes"* / *"no"*).

z.z.: $R(x)$ ist *true* $\Leftrightarrow$ $x \in L$

"$\Rightarrow$": Durch Induktion kann gezeigt werden, daß die Ausgaben der Schaltkreise $C_{i,j}$ die binär verschlüsselten Einträge in die Berechnungstabelle liefern.
Sei $R(x) = true$
$\Rightarrow$ In $T_{|x|^k-1,1}$ in der Berechnungstabelle steht "yes".
$\Rightarrow$ M akzeptiert x.
$\Rightarrow$ $x \in L$

"$\Leftarrow$": Sei $x \in L$
$\Rightarrow$ Die Berechnungstabelle $T(M, x)$ akzeptiert.
$\Rightarrow$ Die Ausgabe von $R(x)$ ist $true$.

noch z.z.: $R(x)$ kann mit dem Speicherplatz $\log |x|$ berechnet werden

Der Schaltkreis $R(x)$ ist nur abhängig von M.
Die Berechnung von $R(x)$ erfordert

(1) die Konstruktion der Input-Gatter:
dies ist leicht möglich durch das Einlesen von x und das Zählen bis $|x|^k$,

(2) die Erzeugung der indizierten Kopien der $C_{i,j}$,

(3) die richtige Verdrahtung dieser Teil-Schaltkreise.

(2) und (3) sind durch Manipulation der Indexmenge leicht im Raum $\log |x|$ durchführbar.

□

Bemerkung:
Das $CIRCUIT\ VALUE$-Problem bleibt $\mathcal{P}$-vollständig, wenn man variablenfreie Schaltkreise über $\{\wedge, \vee\}$ anstelle von $\{\wedge, \vee, \neg\}$ betrachtet ($\rightarrow$ "monotone Schaltkreise").
Im allgemeinen sind monotone Schaltkreise weniger ausdrucksstark als allgemeine Schaltkreise.

Korollar: $MONOTONE\ CIRCUIT\ VALUE$ ist $\mathcal{P}$-vollständig.

Beweis: Übungsaufgabe

□

Satz 2.6 (Cook's Theorem): SAT ist $\mathcal{NP}$-vollständig.

Beweis:

(1) $SAT \in \mathcal{NP}$:
Ist ein erfüllbarer Boolescher Ausdruck gegeben, so wird eine erfüllende Belegung von einer $\mathcal{NP}$-Maschine "geraten" und dann in Polynomialzeit verifiziert.

(2) Da $CIRCUIT\ SAT \ \leq\ SAT$ (Beispiel 2.3):

g.z.z.: Für alle $L \in \mathcal{NP}$ gilt: $L \ \leq\ CIRCUIT\ SAT$

Sei dazu $L \in \mathcal{NP}$.

Wir konstruieren eine Reduktion R, die jeder Eingabe x einen Schaltkreis $R(x)$ zuordnet mit

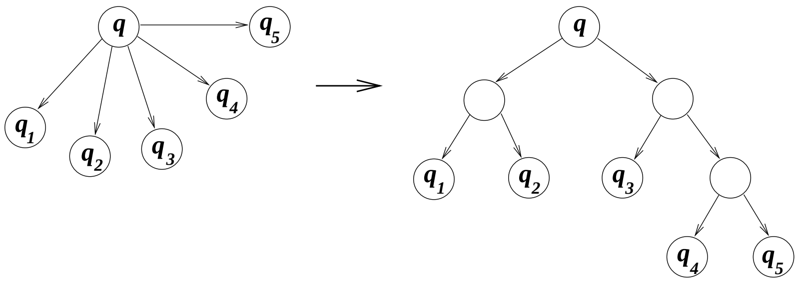
$$x \in L \quad \Leftrightarrow \quad R(x) \text{ ist erfüllbar.}$$

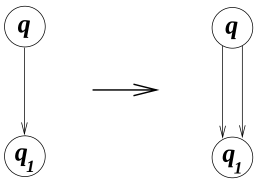
Zu $L \in \mathcal{NP}$ existiert eine NTM $M = (K, \Sigma, \Delta, s)$, die L in der Zeit n^k entscheidet, d. h.

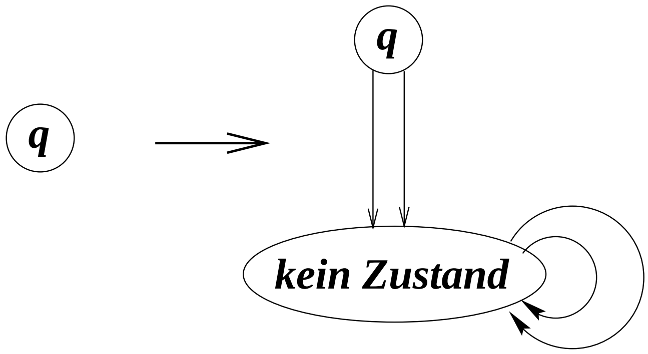
$$x \in L \;\Leftrightarrow\; \text{es gibt eine akzeptierende Berechnung von } M \text{ bei der Eingabe } x \text{ in der Zeit } |x|^k.$$

O. B. d. A. gelte: Sei M eine 1-Band-NTM und habe M in jedem Schritt genau zwei nichtdeterministische Wahlmöglichkeiten.

Beispiel:







Wir müssen dazu evtl. neue Zustände einführen und Δ modifizieren.

Eine der beiden Wahlmöglichkeiten bezeichnen wir durch 0 und die andere durch 1.

$\Rightarrow$ Wir können eine Berechnung eindeutig durch Angabe des 0,1-Strings der nichtdeterministischen Wahlmöglichkeiten $(c_1, \ldots, c_{|x|^k-1}) \in \{0,1\}^{|x|^k-1}$ beschreiben.
Also können wir eine Berechnung von M bei der Eingabe x für jeden String $c = (c_1, \ldots, c_{|x|^k-1})$ durch die Berechnungstabelle $T(M, x, c)$ beschreiben (bei vorgegebenem c ist die Berechnung von M deterministisch!).

Wieder ist $T_{i,j}$ eindeutig bestimmt durch $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$ und die Wahl c_{i-1}.
Die Ränder sind bestimmt durch die Eingabe x.

Wir können $R(x)$ im Raum $\mathcal{O}(\log |x|)$ berechnen, wobei $c_1, \ldots, c_{|x|^k-1}$ variable Eingaben sind.

Offenbar ist $R(x)$ erfüllbar (d. h. es gibt eine Belegung für $c_1, \ldots, c_{|x|^k-1}$, sodaß die entsprechende Berechnungstabelle akzeptierend ist) $\Leftrightarrow$ $x \in L$.

$\square$

2.3 Weitere Charakterisierung von $\mathcal{NP}$

Wir haben bisher $\mathcal{NP}$ als Klasse von Problemen verstanden, die von einer NTM in Polynomialzeit entschieden werden können.

Es gibt aber auch weitere Möglichkeiten, $\mathcal{NP}$ zu charakterisieren:

Definition 2.4: Sei $R \subseteq \Sigma^* \times \Sigma^*$ eine binäre Relation.
R ist **polynomial entscheidbar**

$\underset{def.}{=}$ es existiert eine deterministische TM, die die Sprachen $\{x; y \mid (x, y) \in R\}$ in Polynomialzeit entscheidet.

R heißt **polynomial balanciert**

$\underset{def.}{=}$ es existiert ein k, sodaß gilt: $(x, y) \in R \;\Rightarrow\; |y| \leq |x|^k$
(d. h. die Länge der zweiten Komponente ist polynomial in der Länge der ersten Komponente).

Satz 2.7: Sei $L \subseteq \Sigma^*$. Dann gilt:

$$L \in \mathcal{NP} \;\Leftrightarrow\; \begin{array}{l} \text{es existiert eine polynomial entscheidbare und} \\ \text{polynomial balancierte Relation } R \text{ mit} \\ L = \{x \mid \text{es existiert ein } y \text{ mit } (x, y) \in R\} \end{array}$$

Beweis:

"$\Leftarrow$": Es existiere eine polynomial entscheidbare und polynomial balancierte Relation R und $L = \{x \mid \text{es existiert ein } y \text{ mit } (x,y) \in R\}$.

L kann durch folgende NTM M entschieden werden:

Wenn für x ein y der Länge $\leq |x|^k$ (k aus Definition 2.4) existiert, dann "rät" M dieses y .
M benutzt einen polynomialen Algorithmus auf $x;y$, der $(x,y) \in R$ entscheidet.
M akzeptiert, falls $(x,y) \in R$; ansonsten verwirft M.
$\Rightarrow L \in \mathcal{NP}$

"$\Rightarrow$": Sei $L \in \mathcal{NP}$, d. h. es existiert eine NTM M, die L in der Zeit $|x|^k$ für ein fixiertes k entscheidet.

Wir **definieren** R wie folgt:

$$(x,y) \in R \quad :\Leftrightarrow \quad \begin{array}{l} y \text{ ist eine geeignete Kodierung einer Berechnung} \\ \text{von } M \text{ bei einer Eingabe } x \end{array}$$

R ist polynomial balanciert, da M polynomial-zeitbeschränkt ist.
R ist polynomial entscheidbar, da (sogar) in linearer Zeit entschieden werden kann, ob y eine akzeptierende Berechnung von M bei der Eingabe x kodiert.

Da M L akzeptiert, gilt:

$$L = \{x \mid \text{für ein } y \text{ gilt: } (x,y) \in R\}$$

□

Dieser Satz offenbart eine sehr wichtige Charakterisierung von $\mathcal{NP}$:

Für jedes $x \in L$, $L \in \mathcal{NP}$, gibt es wenigstens ein **polynomiales Zeugnis** bzw. einen polynomialen Zeugen (succinct certificate) y für die Eigenschaft "$x \in L$".

Achtung: Es wird nicht gesagt, wie zu einem $x \in L$ ein Zeugnis y gefunden werden kann. Wir wissen lediglich, daß y existiert.

SAT**:** Für SAT ist eine erfüllende Belegung eines Booleschen Ausdrucks ein Zeugnis. Die Länge der erfüllenden Belegung ist polynomial (linear) in der Länge der Formel ($|Zeugnis|$ = Zahl der Variablen).

$HAMILTON\ PATH$**:** Ein Zeugnis bildet ein $HAMILTON'scher\ Weg$ in G.

2.4 Weitere $\mathcal{NP}$-vollständige Probleme

2.4.1 Varianten des Erfüllbarkeitsproblems

Wir können durch Einschränkung bzw. Vereinfachung aus $\mathcal{NP}$-vollständigen Problemen Probleme in $\mathcal{P}$ erhalten.

Frage: Wo liegt die Grenze?

k-SAT ($k \in \mathbb{N}$)

> **geg.:** ein Boolescher Ausdruck ϕ in KNF, bei dem alle Klauseln aus höchstens k Literalen bestehen.
> **Frage:** Ist ϕ erfüllbar?

Satz 2.8: 3-SAT ist $\mathcal{NP}$-vollständig.

Beweis: Die Analyse des Beweises von Cook's Theorem (Beischpiel 2.3 , Satz 2.6) zeigt, daß lediglich Klauseln der Länge 3 konstruiert werden.

□

Satz 2.9: 3-SAT bleibt $\mathcal{NP}$-vollständig, wenn in den betrachteten Ausdrücken jede Variable höchstens **dreimal** und jedes Literal höchstens **zweimal** vorkommt.

Beweis: Wir können zu jedem Ausdruck ϕ einen logisch äquivalenten Ausdruck ϕ' der gewünschten Art konstruieren:

Komme x in ϕ k-mal vor.

Wir ersetzen das erste Vorkommen von x durch x_1.
Wir ersetzen das zweite Vorkommen von x durch x_2.
⋮
Wir ersetzen das k-te Vorkommen von x durch x_k.

Schließlich fügen wir einen Ausdruck an, der sicherstellt, daß alle x_i zueinander äquivalent sind:

$$(\overline{x_1} \vee x_2) \ \wedge \ (\overline{x_2} \vee x_3) \ \wedge \ \dots \ \wedge \ (\overline{x_k} \vee x_1)$$

□

2-3-SAT

> **geg.:** ein Boolescher Ausdruck ϕ in KNF, bei dem jedes Literal höchstens **zweimal** vorkommt; hierbei haben alle Klauseln die Länge 3.
> **Frage:** Ist ϕ erfüllbar?

Es gilt: SAT, 3-SAT und 2-3-SAT sind $\mathcal{NP}$-vollständig.

Satz 2.10: 2-$SAT \in \mathcal{P}$

Beweis: Wir betrachten Boolesche Formeln in KNF, wobei jede Klausel aus genau 2 Literalen besteht:

Jede Klausel $x \vee y$ ist berechnungsäquivalent zu $(\neg x \rightarrow y)$ und $(\neg y \rightarrow x)$.
Wir übertragen die Boolesche Formel ϕ auf einen gerichteten Graphen $G(\phi) = (V, E)$:
Die Knoten aus V werden durch die in ϕ vorkommenden Variablen und ihre Negationen gebildet. Eine Kante (a, b) existiert genau dann in $G(\phi)$, wenn es die Klausel $(\neg a \vee b)$ oder $(b \vee \neg a)$ in ϕ gibt, d. h. der berechnungsäquivalente Ausdruck $(a \rightarrow b)$ gebildet werden kann.
Da eine Klausel $(a \vee b)$ zu zwei berechnungsäquivalenten Ausdrücken $(\neg a \rightarrow b)$ und $(\neg b \rightarrow a)$ führt, existiert in $G(\phi)$ folgende Symmetrie:

$$(a, b) \in E \quad \Rightarrow \quad (\neg b, \neg a) \in E$$

Ein Pfad in $G(\phi)$ stellt ebenfalls ein logische Implikation dar (Transitivität von $\Rightarrow$).

Behauptung:

ϕ ist unerfüllbar $\Leftrightarrow$ $\exists x \in dom(\phi)$: In $G(\phi)$ existiert ein Pfad von x nach $\neg x$ sowie von $\neg x$ nach x

”$\Leftarrow$”: Seien Pfade in $G(\phi)$ mit der o.a. Eigenschaft gegeben.

Annahme: Es existiert eine erfüllende Belegung T für ϕ.

Sei o. B. d. A. $T(x) = true$.
Da ein Pfad von x nach $\neg x$ existiert und $T(x) = true$ sowie $T(\neg x) = false$ ist, muß es eine Kante (a, b) auf diesem Pfad mit $T(a) = true$ und $T(b) = false$ geben.
Da (a, b) eine Kante von $G(\phi)$ ist, folgt, daß $(\neg a \vee b)$ eine Klausel von ϕ ist.
Diese Klausel kann aber nicht durch T erfüllt werden. $\lightning$

$\Rightarrow$ ϕ ist unerfüllbar.

Bemerkung: Wenn $T(x) = false$ ist, dann verfolge den Pfad von $\neg x$ nach x.

”$\Rightarrow$”: Wir nehmen an, es gäbe keine Pfade in $G(\phi)$ mit der o.a. Eigenschaft, und konstruieren eine erfüllende Belegung für ϕ, d. h. eine Belegung, in der keine Kante in $G(\phi)$ von $true$ nach $false$ führt.

Dazu wiederholen wir folgende Schritte, bis alle Knoten eine Wertzuweisung erhalten haben:

- Wähle einen Knoten a aus $G(\phi)$, dessen Wahrheitswert noch nicht bestimmt wurde und von dem aus kein Pfad nach $\neg a$ existiert.

- Wir bestimmen alle Knoten, die von a aus in $G(\phi)$ erreichbar sind und weisen ihnen den Wert $true$ zu.

- Ebenfalls weisen wir allen Knoten, von denen aus $\neg a$ erreichbar ist, $false$ zu (dies entspricht den Negationen der oben bezeichneten Knoten).

Diese Schritte sind wohldefiniert: Es gibt keine Pfade von a nach b und $\neg b$

Wann immer ein Knoten den Wahrheitswert $true$ erhält, erhalten auch alle seine Nachfolger den Wert $true$. Auf der anderen Seite erhalten alle Vorgänger eines Knotens, der mit $false$ belegt ist, ebenfalls den Wert $false$.
Da nach Voraussetzung kein Pfad von einem Knoten x zu dessen Komplement $\neg x$ existiert, kann es keine Kante von $true$ nach $false$ in $G(\phi)$ geben.

Die so gefundene Wahrheitswertbelegung erfüllt ϕ.

Um zu entscheiden, ob eine gegebene Boolesche Formel erfüllbar ist, überprüfen wir einfach, ob es einen Pfad von einem Knoten x zu dem Knoten $\neg x$ und zurück im zugeordneten Graphen $G(\phi)$ gibt:

Zeitbedarf: Für jeden Knoten a, der einem Literal x_a zugeordnet ist, muß geprüft werden, ob ein Pfad von a nach $\neg a$ existiert ($REACHABILITY$).

$\Rightarrow$ Zeitkomplexität: $\mathcal{O}(n^3)$

$\Rightarrow$ 2-$SAT \in \mathcal{P}$.

$\square$

Korollar: 2-$SAT \in \mathcal{NL}$

Beweis: $\mathcal{NL}$ ist abgeschlossen bzgl. der Komplementbildung!

$\Rightarrow$ **g.z.z.:** Unerfüllbare Ausdrücke können in $\mathcal{NL}$ erkannt werden.

Wir können mit einer $\mathcal{NL}$-Maschine die Bedingung aus dem letzten Satz überprüfen durch Raten einer Variablen x und Raten eines Weges von x nach $\neg x$ und zurück.

$\square$

3-SAT ist eine Verallgemeinerung von 2-SAT.

2-SAT kann auch anders verallgemeinert werden:

MAX 2-SAT
geg.: eine Menge von Klauseln aus jeweils 2 Literalen bestehend; $k \in \mathbb{N}$
Frage: Gibt es eine Belegung, so daß wenigstens k Klauseln wahr werden?

Satz 2.11: MAX 2-SAT ist $\mathcal{NP}$-vollständig.

Beweis: Wir betrachten folgende 10 Klauseln:

$$(x)\ (y)\ (z)\ (w)$$

$$(\neg x \vee \neg y)\ (\neg y \vee \neg z)\ (\neg z \vee \neg x)$$
$$(x \vee \neg w)\ (y \vee \neg w)\ (z \vee \neg w)$$

Wir können **nie** alle Klauseln gleichzeitig erfüllen!

Es gilt:

- Jede erfüllende Belegung von $(x \vee y \vee z)$ kann so erweitert werden, daß genau 7 Klauseln erfüllt sind.
- Für alle übrigen Belegungen existiert keine Erweiterung, die mehr als 6 Klauseln erfüllt.

Diese Beobachtung liefert eine Reduktion von 3-SAT auf MAX 2-SAT:

Sei ϕ eine Instanz von 3-SAT.

Wir konstruieren eine Eingabe $R(\phi)$ für MAX 2-SAT:

Für jede Klausel $C_i = (\alpha \vee \beta \vee \gamma)$ von ϕ werden 10 Klauseln mit α, β, γ anstelle von x, y, z genommen.
w wird durch eine neue Variable w_i (speziell für C_i gewählt) ersetzt.
Hatϕ m Klauseln, so hat $R(\phi)$ 10 Gruppen mit insgesamt $10m$ Klauseln.
Wir setzen jetzt $k = 7m$.

Behauptung:

Das konstruierte MAX 2-SAT für $R(\phi)$ ergibt "*yes*" $\quad\Leftrightarrow\quad$ ϕ ist erfüllbar

"$\Rightarrow$": **Annahme:** $7m$ Klauseln können in $R(\phi)$ erfüllt werden

$\Rightarrow$ In m Gruppen werden jeweils 7 Klauseln erfüllt,
da in einer Gruppe nicht mehr als 7 Klauseln erfüllt werden können.
$\Rightarrow$ Wir haben eine Konstruktion, in der alle Klauseln von ϕ erfüllt werden.

"$\Leftarrow$": Jede erfüllende Belegung von ϕ liefert $7m$ erfüllende Klauseln, falls die w_i sinnvoll gewählt sind.

Offenbar gilt:

$$MAX\ 2\text{-}SAT \in \mathcal{NP}$$

Ferner ist die Reduktion in $\log space$ ausführbar.

$\square$

Interessant für andere $\mathcal{NP}$-Vollständigkeitsbeweise ist das folgende Problem:

NAE SAT ("*Not-All-Equal SAT*")

geg.: eine Menge von Klauseln mit jeweils 3 Literalen.
Frage: Existiert eine erfüllende Belegung, so daß in keiner Klausel alle 3 Literale den gleichen Wahrheitswert (*true*) haben?

Satz 2.12: *NAE SAT* ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß $CIRCUIT\ SAT \leq NAE\ SAT$:

Dazu zeigen wir, daß die schon besprochene Reduktion $CIRCUIT\ SAT \leq SAT$ ebenfalls auf unsere Reduktion angewandt werden kann.
Wir betrachten die in der Reduktion aus dem Schaltkreis erzielten Klauseln:
Dabei fügen wir allen Klauseln mit einem oder zwei Literalen das Literal z hinzu.

Behauptung:

Die erzeugte Klauselmenge als Instanz von *NAE SAT* ist erfüllbar	$\Leftrightarrow$	Der Originalschaltkreis ist erfüllbar.

"$\Rightarrow$": Die Belegung $t \in \{0,1\}^n$ erfülle alle Klauseln im Sinne von *NAE SAT*.

$\Rightarrow$ Die komplementäre Belegung $\overline{t}$ erfüllt die Klauselmenge ebenfalls im Sinne von *NAE SAT*.
In einer der beiden Belegungen hat das Literal z den Wert *false*.
Sei o. B. d. A. t diese Belegung.
Dieses t erfüllt die ursprünglichen Klauseln.
$\Rightarrow$ Gemäß der Reduktion $CIRCUIT\ SAT \leq SAT$ existiert eine erfüllende Belegung für den Schaltkreis.

"$\Leftarrow$": Die Belegung $t \in \{0,1\}^n$ erfülle den Schaltkreis.

$\Rightarrow$ Es existiert eine erfüllende Belegung t, die die Klauselmenge im Sinne von 3-*SAT* erfüllt (da $CIRCUIT\ SAT \leq 3\text{-}SAT$).
Wir setzen $t(z) = false$. Dann sind bei der Belegung t in keiner Klausel alle Literale *true*, da

- die Klauseln bei der Reduktion in Gruppen entstehen, die mit den entsprechenden Gattern des Schaltkreises korrespondieren, und
- *true* / *false* / *NOT* und die Variablen-Gatter mit den Klauseln korrespondieren, in denen z vorkommt.

Wir betrachten *AND*-Gatter:

$$(1) : (\neg g \vee h \vee z), \quad (2) : (\neg g \vee h' \vee z), \quad (3) : (\neg h \vee \neg h' \vee g)$$

t kann nicht alle Literale gleichzeitig in allen Klauseln erfüllen, da

- in (1) und (2) z nicht erfüllt ist,
- wenn in (3) alle Literale erfüllt wären, wäre (1) oder (2) nicht erfüllt.

Gatter	SAT-Klausel	$NAE\ SAT$-Klausel
$g = true$	g	$(g \vee z)$
$g = false$	$\neg g$	$(\neg g \vee z)$
$g = NOT(h)$	$(\neg g \vee \neg h)$	$(\neg g \vee \neg h \vee z)$
	$(g \vee h)$	$(g \vee h \vee z)$
$g = x$	$(g \vee \neg x)$	$(g \vee \neg x \vee z)$
	$(\neg g \vee x)$	$(\neg g \vee x \vee z)$
$g = OR(h, h')$	$(\neg h \vee g)$	$(\neg h \vee g \vee z)$
	$(\neg h' \vee g)$	$(\neg h' \vee g \vee z)$
	$(h \vee h' \vee \neg g)$	$(h \vee h' \vee \neg g)$
$g = AND(h, h')$	$(\neg g \vee h)$	$(\neg g \vee h \vee z)$
	$(\neg g \vee h')$	$(\neg g \vee h' \vee z)$
	$(\neg h \vee \neg h' \vee g)$	$(\neg h \vee \neg h' \vee g)$

Die Argumentation für AND-Gatter gilt ebenfalls für OR-Gatter.

Zeit- / Raumbedarf: Die Reduktion ist in Polynomialzeit und $\log space$ durchführbar, da dies ebenfalls für $CIRCUIT\ SAT \leq SAT$ der Fall ist.

$\square$

2.4.2 Graphtheoretische Probleme

Wir betrachten im folgenden ungerichtete Graphen $G = (V, E)$ (die Kantenrelation ist symmetrisch, und es existieren keine trivialen Schleifen).

Definition 2.5:
$I \subseteq V$ heißt **Independent Set** $\underset{def.}{=}$ für alle $i, j \in I$ gilt: $(i, j) \notin E$.

INDEPENDENT SET
geg.: ein ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$.
Frage: Existiert in G ein Independent Set I mit $\#I = k$?

Satz 2.13: $INDEPENDENT\ SET$ ist $\mathcal{NP}$-vollständig.

Beweis:
Idee: Besitzt ein Graph ein Dreieck, so kann höchstens ein Knoten des Dreiecks zum

Independent Set gehören.

Wir vereinfachen das Problem durch Betrachtung spezieller Graphen, die in disjunkte Dreiecke zerlegt werden können.

Nun können wir eine Reduktion von 3-SAT angeben:

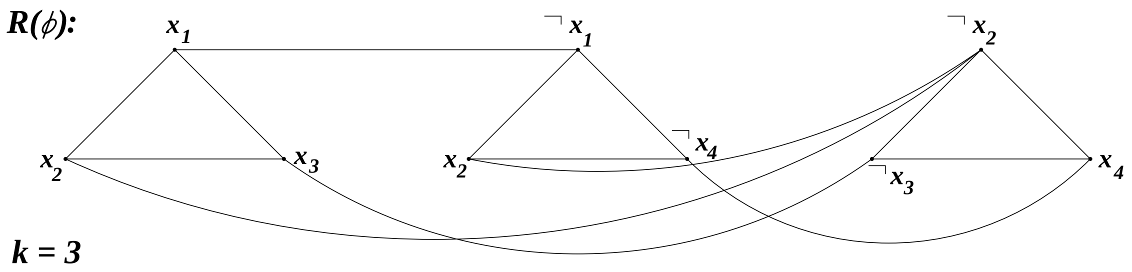
Sei ϕ eine Instanz von 3-SAT mit m Klauseln $C_1, \ldots, C_m$, wobei $C_i = (\alpha_{i_1} \vee \alpha_{i_2} \vee \alpha_{i_3})$.
Die α_{i_j} sind Literale (d. h. Boolesche Variablen oder negative Boolesche Variablen).
Wir konstruieren eine Instanz von $INDEPENDENT\ SET$ $R(\phi) = (G, k)$ mit

- $k = m$
- $G = (V, E)$ mit
    - $V = \{v_{i_j} \mid i = 1, \ldots, m;\ \text{und } j = 1, 2, 3\}$
    - $E = \{(v_{i_j}, v_{i_k}) \mid i = 1, \ldots, m;\ \text{und } j \neq k\} \ \cup\ \{(v_{i_j}, v_{l_k}) \mid i \neq l \text{ und } \alpha_{i_j} = \neg\alpha_{l_k}\}$

Die erste Kantenmenge definiert m Dreiecke, wohingegen die zweite Kantenmenge die verschiedenen Dreiecke verbindet, falls die Knoten zu gegensätzlichen Literalen gehören.

Beispiel:

$$(x_1 \vee x_2 \vee x_3)\ (\neg x_1 \vee x_2 \vee \neg x_4)\ (\neg x_2 \vee \neg x_3 \vee x_4)$$



Behauptung:

Es existiert ein Independent Set I der Größe m in $R(\phi)$ $\quad\Leftrightarrow\quad$ ϕ ist erfüllbar.

"$\Rightarrow$": Es existiere ein Independent Set I mit $\#I = m$.

$\Rightarrow$ I enthält genau einen Knoten jedes Dreiecks.
I enthält keine gegensätzlichen Literale, da diese durch Kanten verbunden sind.
$\Rightarrow$ I liefert eine erfüllende Belegung für ϕ:
$true$-Literale sind alle Literale, die zu Knoten aus I gehören.
Die nichtvorkommenden Literale können frei gewählt werden.
Da I Knoten aus jedem Dreieck enthält, erfüllt die Belegung jede Klausel.

"$\Leftarrow$": Es existiere eine erfüllende Belegung für ϕ.

Aus jeder Klausel wird ein *true*-Literal ausgewählt.
Wir nehmen die entsprechenden Knoten aus jedem Dreieck und erhalten ein Independent Set der Größe m.

$\square$

4-*DEGREE-INDEPENDENT SET*

geg.: ein ungerichteter Graph $G = (V, E)$ mit dem Grad 4, $k \in \mathbb{N}$
Frage: Existiert in G ein Independent Set I mit $\#I = k$?

Korollar: 4-*DEGREE-INDEPENDENT SET* ist $\mathcal{NP}$-vollständig.

Beweis: Der Beweis des letzten Satzes arbeitet auch, wenn in der Klauselmenge ϕ jedes Literal höchstens zweimal auftritt (vgl. $2 - 3 - SAT$).

$\Rightarrow$ Der konstruierte Graph hat den Grad 4:

Es gibt 2 Kanten innerhalb des Dreiecks und höchstens 2 Kanten zu gegensätzlichen Literalen (auftretende Klauseln mit 2 Literalen werden durch Kanten anstelle von Dreiecken repräsentiert).

$\square$

Definition 2.6:
$C \subseteq V$ heißt **Clique** in G $\underset{def.}{=}$ für alle $i, j \in C$ gilt: $(i, j) \in E$

CLIQUE

geg.: ein ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$.
Frage: Existiert in G eine Clique der Größe k?

Definition 2.7:
$C \subseteq V$ heißt **Node Cover** (Knoten-Überdeckung) in G
$\underset{def.}{=}$ für alle $i, j \in E$ gilt: $i \in C$ oder $j \in C$.

NODE COVER

geg.: ein ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$.
Frage: Existiert in G ein Node Cover der Größe k?

Korollar: *CLIQUE* und *NODE COVER* sind $\mathcal{NP}$-vollständig.

Beweis:

(1) Wird anstelle eines Graphen sein Komplement betrachtet (d. h. die Knoten im Komplement sind durch Kanten verbunden $\Leftrightarrow$ die Knoten im Ursprungsgraphen sind nicht verbunden), dann wird *CLIQUE* zu *INDEPENDENT SET* und umgekehrt.

(2) Weiter gilt: Ist I ein Independent Set, so ist $V - I$ ein Node Cover (Übungsaufgabe).

□

Definition 2.8: Sei $G = (V, E)$ ein ungerichteter Graph.
Ein **Cut (Schnitt)** von G ist eine Partition von V (d. h. eine disjunkte Zerlegung von V in S und $V - S$).
Die Größe eines Cuts ist die Zahl der Kanten zwischen S und $V - S$.

MAX CUT

geg.: ein ungerichteter Graph $G = (V, E)$, $k \in I\!N$, wobei in G Mehrfachkanten erlaubt sind.
Frage: Existiert in G ein Cut (Schnitt) der Größe $\geq k$?

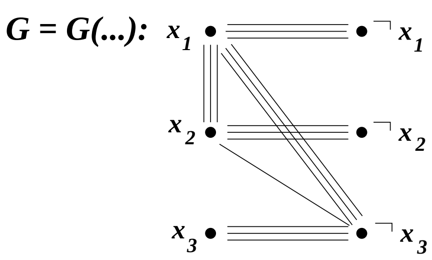
Satz 2.14: *MAX CUT* ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß *NAE SAT* $\leq$ *MAX CUT*:

Seien $C_1, \ldots, C_m$ Klauseln und $x_1, \ldots, x_n$ die darin auftretenden Variablen.
G hat $2n$ Knoten $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$.
Ferner sei $C_i = (\alpha \vee \beta \vee \gamma)$.
Wir nehmen die Kanten des Dreiecks $[\alpha, \beta, \gamma]$ (stimmen hier zwei Variablen überein, so wird anstelle eines Dreiecks eine Doppelkante hinzugefügt).
Weiter nehmen wir für jede Variable x_i n_i Kopien der Kante $(x_i, \neg x_i)$ hinzu, wobei n_i die Anzahl der Vorkommen der x_i oder $\neg x_i$ in den Klauseln angibt.
Schließlich setzen wir $k = 5m$.

Beispiel:

$$(x_1 \vee x_2 \vee x_2) \;\wedge\; (x_1 \vee \neg x_3 \vee \neg x_3) \;\wedge\; (x_1 \vee x_2 \vee \neg x_3)$$



Annahme: Der Cut $(S, V - S)$ hat die Größe $\geq 5m$.
O. B. d. A. gelte: x_i und $\neg x_i$ kommen für alle i auf den verschiedenen Seiten im Cut vor:
Annahme: x_i und $\neg x_i$ kommen auf der gleichen Seite vor.
$\Rightarrow$ Diese Knoten tragen $l \leq 2n_i$ Kanten zum Cut bei.

$\Rightarrow$ Wir können eine der beiden Variablen auf die andere Seite des Cuts legen, dann wird seine Größe auf midenstens $(2n_i - l)$ grösser.

Die Literale aus S werden als $true$ angesehen, die Literale aus $V - S$ als $false$.
Die Gesamtzahl der Kanten im Cut, die gegensätzliche Variablen verbinden, ist $3m$ ($\widehat{=}$ Gesamtzahl der Literale).
Die übrigen $2m$ Kanten stammen aus den Dreiecken, die zu den Klauseln gehören. Jedes Dreieck trägt höchstens 2 Kanten zum Cut bei.
$\Rightarrow$ Jedes Dreieck wird durch den Cut geschnitten.

$\Rightarrow$ Wenigstens ein Literal einer Klausel hat einen anderen Wahrheitswert als die übrigen Literale, d. h. wenigstens ein Literal hat den Wahrheitswert $true$ und wenigstens eins den Wert $false$.

$\Rightarrow$ Wir haben eine erfüllende Belegung im Sinne von $NAE\ SAT$.

Umgekehrt gilt: Jede erfüllende Belegung von $NAE\ SAT$ kann leicht in einen Cut der Größe $\geq 5m$ überführt werden.

□

$MIN\ CUT$

> **geg.:** ein ungerichteter Graph $G = (V, E)$, $k \in I\!N$, wobei in G Mehrfachkanten erlaubt sind.
> **Frage:** Existiert in G ein Cut der Größe $\leq k$?

Achtung: $MIN\ CUT\ \in\ \mathcal{P}$

Die Größe eines minimalen Cuts, der $s, t \in V$ separiert, entspricht der Größe des maximalen Flusses von s nach t (Maxflow):
Beweis -Übungsaufgabe.

$MAX\ BISECTION$

> **geg.:** ein ungerichteter Graph $G = (V, E)$, $k \in I\!N$.
> **Frage:** Existiert in G ein Cut $(S, V - S)$ der Größe $\geq k$ und $\#S = \#(V - S)$?

$MAX\ BISECTION$ könnte sowohl leichter als auch schwerer sein als $MAX\ CUT$ (je nachdem, ob die Gleichheit der Mengengrößen als zusätzliche Bedingung oder als Einschränkung angesehen wird)!

Satz 2.15: $MAX\ BISECTION$ ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß $MAX\ CUT \ \leq\ MAX\ BISECTION$:

Wir fügen $\#V$ vollständig isolierte Knoten zu G hinzu.
Nun können wir jeden Cut von G durch eine geeignete Verteilung der neuen isolierten Knoten in eine Bisection verwandeln.

□

$BISECTION\ WIDTH$

geg.: ein ungerichteter Graph $G = (V, E)$, $k \in I\!N$.
Frage: Existiert in G eine Bisection der Größe $\leq k$?

Satz 2.16: $BISECTION\ WIDTH$ ist $\mathcal{NP}$-vollständig.

Beweis:
$G = (V, E)$ mit $\#V = 2n$ hat eine Bisection der Größe $\geq k$
$\Leftrightarrow\ \overline{G} = (V, V^2 - E)$ hat eine Bisection der Größe $\leq n^2 - k$.

□

Im Falle der Bisection sind das Maximierungs- und das Minimierungsproblem gleich schwer.

Satz 2.17: $HAMILTON\ PATH$ ist $\mathcal{NP}$-vollständig.

Beweis: Übungsaufgabe*. **Hinweis**: Man kann $3 - SAT$ auf $HAMILTON\ PATH$ reduzieren.

□

Korollar: $TSP(D)$ ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß $HAMILTON\ PATH \ \leq\ TSP(D)$:

Dazu kodieren wir $HAMILTON\ PATH$ mit einer polynomialzeit- / log *space*-beschränkten Transformation in ein $TRAVELING\ SALESMAN$-Problem.
Eine Eingabe für $HAMILTON\ PATH$, die nicht die Kodierung eines Graphen darstellt, wird dabei auf einen String abgebildet, der auch nicht die Kodierung einer Eingabe für ein TSP ist.
Sei $G = (V = \{1, \ldots, n\}, E)$. G liege dabei als Adjazenzmatrix $A_{i,j}$ vor.
Weiterhin sei $f(G)$ die Eingabe für das TSP.
Wir bilden die Kostenmatrix $C_{i,j}$ für das TSP durch Transformation der Adjazenzmatrix

$A_{i,j}$ von G:
Dazu setzen wir

$$c(i,j) = \begin{cases} 1 & falls\ a(i,j) = 1,\ d.h.\ (i,j) \in E \\ 2 & sonst \end{cases}$$

Die Kostengrenze D sei $D = n + 1 = |V| + 1$.

Behauptung:

G beinhaltet einen Hamilton-Pfad $\Leftrightarrow$ $f(G)$ erlaubt eine Rundreise mit $D \leq n + 1$ beschränkten Kosten

"$\Rightarrow$": Ein Hamilton-Pfad verursacht die Kosten $D = n - 1$.
Wir fügen eine Kante vom Endpunkt des Hamilton-Pfades zu dessen Anfangspunkt ein. Dies erhöht die Kosten auf $D \in \{n, n + 1\}$.

"$\Leftarrow$": Rundreisen in $f(G)$ bestehen aus genau n Wegstrecken und haben nur dann die Kosten $D \leq n + 1$, wenn alle Wegstrecken (i, j) des Hamiltonkreises mit $c(i, j) = 1$ ausser einer genutzt werden.

Zeit- / Raumbedarf:

- Die Transformation der Matrix ist in $\mathcal{O}(n^2)$ Zeit durchführbar.
- Der benötigte Raum ist $\log space$, da die Umkodierung der Matrizen unmittelbar ohne notwendigen Zwischenspeicher erfolgt. Ferner ist nur ein binär kodierter Zähler erforderlich, um die Anzahl n festzustellen.

□

Eine wichtige Klasse von Problemen bilden die sogenannten Färbungsprobleme:

k-COLORING
geg.: ein ungerichteter Graph $G = (V, E)$, die Zahl k ($\in \mathbb{N}$) der Farben.
Frage: Existiert eine Färbung der Knoten von G mit k Farben, so daß keine zwei benachbarte (durch Kanten verbundene) Knoten die gleiche Farbe haben?

Satz 2.18: 3-*COLORING* ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß *NAE SAT* $\leq$ 3-*COLORING*:

Es sei die 3-Klauselmenge $\{C_1, \ldots, C_m\}$ über den Variablen $x_1, \ldots, x_n$ gegeben.

NAE SAT fragt, ob eine erfüllende Belegung existiert, bei der nicht alle drei Literale einer Klausel gleichzeitig wahr sind.

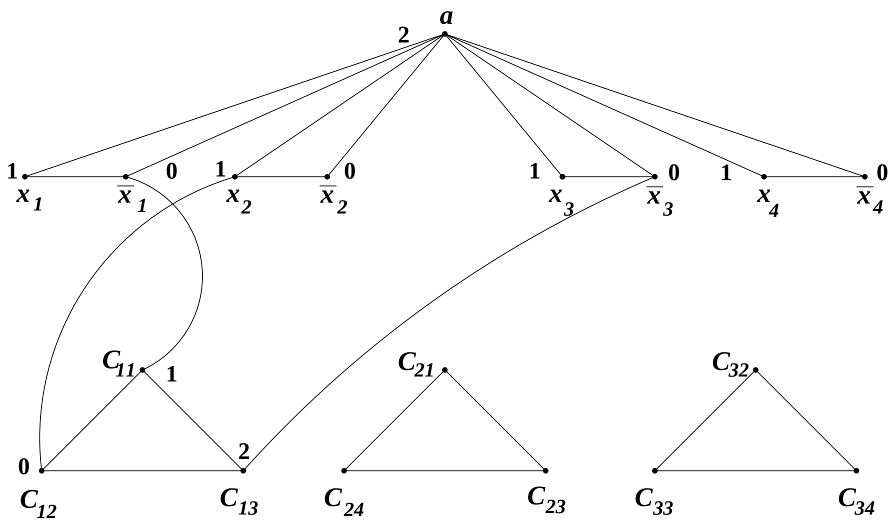
Wir konstruieren einen aus Dreiecken aufgebauten Graphen G, so daß gilt:

$$G \text{ ist 3-färbbar} \quad \Leftrightarrow \quad NAE\ SAT \text{ ist erfüllbar}$$

Für jede Variable x_i nehmen wir ein Dreieck $[a, x_i, \neg x_i]$, wobei a ein gemeinsamer Knoten für alle Variablen / Dreiecke ist.
Für jede Klausel $C_i = (x_{j_1}^{\varepsilon_1} \vee x_{j_2}^{\varepsilon_2} \vee x_{j_3}^{\varepsilon_3})$ nehmen wir das Dreieck $[C_{i,j_1}, C_{i,j_2}, C_{i,j_3}]$ hinzu. Anschließend fügen wir Kanten zwischen $C_{i,j}$ und dem Komplement des j-ten Literals im "Variablendreieck" ein.

Beispiel:

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



Behauptung:

Wir können G mit den 3 Farben $\{0, 1, 2\}$ färben $\quad \Leftrightarrow \quad NAE\ SAT$ ist erfüllbar

"$\Rightarrow$": O. B. d. A. sei a mit 2 gefärbt

$\Rightarrow$ Jeweils eine der Variablen x_i und $\overline{x_i}$ ist mit 0 bzw. 1 gefärbt.

Ist x_i mit 1 gefärbt, so wird x_i als $true$ angesehen. Umgekehrt wird bei 0-Färbung x_i als $false$ angesehen.

Besitzen alle Literale einer Klausel den gleichen Wert, so läßt sich das Klauseldreieck nicht färben.

"$\Leftarrow$": Wir nehmen eine erfüllende Belegung für $NAE\ SAT$, färben a mit der Farbe 2 und die Variablendreiecke entsprechend der Belegung.

Für die Färbung der Klauseldreiecke nehmen wir zwei gegensätzlich belegte Literale (diese existieren, da $NAE\ SAT$ erfüllt ist) und färben diese entsprechend mit 0 oder 1. Der dritte Knoten wird mit 2 gefärbt.

$\Rightarrow$ Wir erhalten eine 3-Färbung von G.

□

2.4.3 Mengen- und Zahlentheoretische Probleme

TRIPARTITE MATCHING

geg.: drei Mengen M, F, H (Männer, Frauen, Häuser) mit $\#M = \#F = \#H = n$ und $T \subseteq M \times F \times H$.

Frage: Existieren n Tripel in T, so daß keine zwei Tripel eine gemeinsame Komponente besitzen (jeder Mann hat eine andere Frau und ein eigenes Haus)?

Satz 2.19: *TRIPARTITE MATCHING* ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß 3-$SAT \leq TRIPARTITE\ MATCHING$:

Sei $\phi = \bigwedge_{j=1}^{m} C_j$ eine Boolesche Formel in KNF mit $C_j = (y_{j,1} \vee y_{j,2} \vee y_{j,3})$ Klauseln über den Variablen x_i $(1 \leq i \leq n)$: $y_{j,k} \in \{x_i\} \cup \{\overline{x_i}\}$.

Wir konstruieren die disjunkten Mengen W, X, Y mit $|W| = |X| = |Y|$ und $M \subseteq W \times X \times Y$, so daß gilt:

$$M \text{ beinhaltet ein Matching} \quad \Leftrightarrow \quad \phi \text{ ist erfüllbar.}$$

Zuerst separieren wir M gemäß der Funktion ϕ in drei Klassen:

- T_i - *Truth Setting*: Hat die Variable x_i in allen Klauseln denselben Wert?
- S_i - *Satisfaction Testing*: Sind alle Klauseln erfüllt?
- G - *Garbage Collection*: Dies sorgt dafür, daß das Matching aufgeht.

T_i korrespondiert mit der Variablen x_i , $i = 1, \ldots, n$, wobei folgendes gilt:
Die Tripel T_i können in zwei Mengen unterteilt werden:

$$T_i^0 = \{(u_{i,j}, a_{i,j+1}, b_{i,j})\} \cup \{u_{i,m}, a_{i,1}, b_{i,m})\} \text{ mit } 1 \leq j \leq m$$

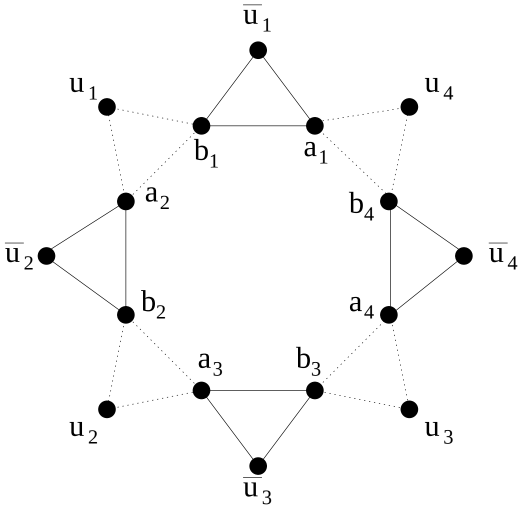
und

$$T_i^1 = \{(\overline{u_{i,j}}, a_{i,j}, b_{i,j})\} \text{ mit } 1 \leq j \leq m$$

Die $a_{i,j}$ und $b_{i,j}$ sind nur in den Tripeln aus $T_i = T_i^0 \cup T_i^1$ enthalten.

- $W = \{u_{i,j}, \overline{u_{i,j}} | i = 1, \ldots, n; j = 1, \ldots, m\}$
- $a_{i,j} \in X$ und $b_{i,j} \in Y$, $1 \leq i \leq n, 1 \leq j \leq m$.

Beispiel: Sei $m = 4$ und i fix. In dem Diagramm ist der Index i zur Vereinfachung weggelassen.



Jedes Matching M' schließt genau m Tripel aus T_i mit ein, entweder T_i^0 oder T_i^1.
Wir entscheiden

$$x_i = 1 \quad \Leftrightarrow \quad M' \cap T_i = T_i^1$$

S_j entspricht jeweils einer Klausel in C_j und formt 3 Tripel $(u_{i,j}^{\sigma}, s_{1,j}, s_{2,j})$, $\sigma \in \{0, 1\}$, wobei folgendes gilt:

- $u_{i,j}, \overline{u_{i,j}} \in W$
- $s_{1,j} \in X$ und $s_{2,j} \in Y$ für jedes j, $1 \leq j \leq m$
- $S_j = \{(u_{i,j}^{\sigma}, s_{1,j}, s_{2,j}) | \sigma \in \{0, 1\}, \exists k \in \{1, 2, 3\} : y_{j,k} \in C_j, x_i^{\sigma} = y_{j,k}\}$: wir wählen das Tripel in direkter Abhängigkeit davon, ob $x_i \in C_j$ oder $\overline{x_i} \in C_j$.

Jedes Matching M' enthält exakt ein Tripel aus S_j, und zwar nur das zu einem $u_{i,j}$ (bzw. $\overline{u_{i,j}}$) für ein x_i ($\overline{x_i}$), das nicht in $M' \cap T_i$ vorkommt. Dies ist genau dann der Fall, wenn die durch M' ermittelte Belegung gerade die Klausel C_j erfüllt.

G beinhaltet $u_{i,j}, \overline{u_{i,j}} \in W$, $1 \leq i \leq n, 1 \leq j \leq m$, sowie $g_{1,k} \in X$ und $g_{2,k} \in Y$ $(1 \leq k \leq m(n-1))$.

$$G = \{(u_{i,j}, g_{1,k}, g_{2,k})\} \cup \{(\overline{u_{i,j}}, g_{1,k}, g_{2,k})\}$$

Hierbei müssen die $g_{1,k}, g_{2,k}$ mit denjenigen $u_{i,j}$ ($\overline{u_{i,j}}$) gematched werden, die nicht in Tripeln von $M' - G$ auftauchen, von denen es genau $m(n-1)$ geben muß.

Die Menge $M - G$ sorgt dafür, daß die Bedingungen bzgl. der Wertebelegungen erfüllt werden können und G dient zur Erfüllung des Matchings.

Es ergibt sich:

- $W = \{u_{i,j}, \overline{u_{i,j}} : 1 \leq j \leq m, 1 \leq i \leq n\}$
- $X = A \cup S_1 \cup G_1$
  mit $A = \{a_{i,j}\}$, $S_1 = \{s_{1,j} : 1 \leq j \leq m\}$ und $G_1 = \{g_{1,k} : 1 \leq k \leq m(n-1)\ \}$
- $Y = B \cup S_2 \cup G_2$
  mit $B = \{b_{i,j}\}$, $S_2 = \{s_{2,j} : 1 \leq j \leq m\}$ und $G_2 = \{g_{2,k} : 1 \leq k \leq m(n-1)\ \}$

$\Rightarrow$

$$M = \bigcup_{i=1}^{n} T_i \ \cup \ \bigcup_{j=1}^{m} S_j \ \cup \ G$$

In jeder der Mengen W, X, Y sind genau $2mn$ Elemente und in der Relation M genau $2nm + 3m + 2mn(mn - m)$ Tripel.

Behauptung:

$$\phi \text{ ist erfüllbar} \quad \Leftrightarrow \quad M \text{ beinhaltet ein Matching.}$$

"$\Rightarrow$": Sei $t \in \{0,1\}^n$ eine erfüllende Belegung für ϕ.

Wir konstruieren ein Matching $M' \subseteq M$ wie folgt:

Ist C_j erfüllt, so ist für ein i x_i oder $\overline{x_i}$ aus C_j erfüllt.
Wir wählen $(x_{i,j}, s_{1,j}, s_{2,j})$ bzw. $(\overline{x_{i,j}}, s_{1,j}, s_{2,j})$ entsprechend der erfüllenden Belegung von x_i in der Klausel C_j zum Matching dazu (insgesamt: m Tripel).
Dann wählen wir T_i^0 oder T_i^1 entsprechend der komplementären erfüllenden Belegung für $x_i = 0$ oder $x_i = 1$ (insgesamt: mn Tripel).
Schließlich ergänzen wir die Tripel auf triviale Weise aus G, so daß das Matching erfüllt wird (insgesamt: $mn - n$ Tripel).

Wir erhalten eine Menge aus genau $2mn$ Tripeln, bei denen keine Komponente in einem anderen Tripel enthalten ist.

$\Rightarrow$ Diese Menge ist das gesuchte Matching M'.

"$\Leftarrow$": Gegeben sei ein Matching M'.

Wir konstruieren eine erfüllende Belegung für ϕ:

$$x_i := \begin{cases} 0 & M' \textit{ enthält die Tripel aus } T_i^0 \\ 1 & M' \textit{ enthält die Tripel aus } T_i^1 \end{cases}$$

Entsprechend der Konstruktion des Matchings für S_j erfüllt die ermittelte Belegung alle Klauseln C_j.

$\Rightarrow$ ϕ ist erfüllbar.

Zeit- / Raumbedarf:

- Die Konstruktion der Formel verläuft straightforward. Die Länge der ursprünglichen Formel ist $3m$ und die Anzahl der zu konstruierenden Tripel $2nm+3m+2mn(mn-m)$
  $\Rightarrow$ Die Konstruktion ist also in polynomialer Zeit durchführbar.
- Wir benötigen einen Zähler für jedes Tripelelement, d. h. den Raum $\leq \mathcal{O}(\log mn)$, also log *space*.

$\square$

SET COVERING ("Mengen-Überlagerung")

geg.: $\mathcal{F} = \{S_1, \ldots, S_n\}$ Familie von Mengen, mit $S_i \subseteq U$, U endliches Universum; $k \in I\!N$

Frage: Gibt es k Mengen $S_{i_1}, \ldots, S_{i_k} \in \mathcal{F}$ mit

$$\bigcup_{j=1}^{k} S_{i_j} = U?$$

SET PACKING

geg.: $\mathcal{F} = \{S_1, \ldots, S_n\}$, $S_i \subseteq U$, U endlich; $k \in I\!N$

Frage: Existieren k **paarweise disjunkte** Mengen $S_{i_1}, \ldots, S_{i_k} \in \mathcal{F}$ mit

$$\bigcup_{j=1}^{k} S_{i_j} = U?$$

EXACT COVER BY 3-SETS

geg.: $\mathcal{F} = \{S_1, \ldots, S_n\}$, $S_i \subseteq U$, $\#S_i = 3$, $\#U = 3m$

Frage: Existieren m paarweise disjunkte Mengen $S_{i_1}, \ldots, S_{i_m} \in \mathcal{F}$ mit

$$\bigcup_{j=1}^{m} S_{i_j} = U?$$

Korollar: *EXACT COVER BY* 3-*SETS*, *SET COVERING* und *SET PACKING* sind $\mathcal{NP}$-vollständig.

Beweis:
Es gilt: *TRIPARTITE MATCHING* $\leq$ *EXACT COVER BY* 3-*SETS*:

TRIPARTITE MATCHING ist ein Spezialfall von *EXACT COVER BY* 3-*SETS*, indem die Menge $\mathcal{F}$ bestimmte Eigenschaft hat: und zwar U kann in drei gleichgroße Mengen M, F und H zerlegt werden, so dass jede Menge aus $\mathcal{F}$ jeweils ein Element dieser drei Mengen enthält ($S_i = (m_i, f_i, h_i)$).

Es gilt: *EXACT COVER BY* 3-*SETS* $\leq$ *SET COVERING*:

EXACT COVER BY 3-*SETS* ist ein Spezialfall von *SET COVERING* mit $\#U = 3m$, $\#S_i = 3$ für alle $S_i \in \mathcal{F}$ und $k = m$.

Analoges gilt für *SET PACKING*.

$\square$

SET COVERING *SET PACKING*

EXACT COVER BY 3-*SETS*

INTEGER PROGRAMMING ("ganzzahlige Optimierung")
geg.: eine $n \times m$ Matrix A über $\mathbb{Z}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$; $k \in \mathbb{Z}$.
Frage: Existiert eine ganzzahlige Lösung x für

$$Ax \leq b \text{ mit } x \geq 0,$$

die $c^T x$ minimiert ($c^T x \leq k$)?

Korollar: *INTEGER PROGRAMMING* ist $\mathcal{NP}$-vollständig.

Beweis:
Wir können z. B. zeigen, daß *SET COVERING* $\leq$ *INTEGER PROGRAMMING*:

SET COVERING kann als ganzzahliges Optimierungsproblem ausgedrückt werden:
geg.: $\mathcal{F} = \{S_1, \ldots, S_n\}$ Familie von Mengen, mit $S_i \subseteq U$, U endliches Universum; $k \in \mathbb{N}$.
Sei $x = (x_1, \ldots, x_n)$, $x_i \in \{0, 1\}$ mit

$$x_i = 1 \quad \Leftrightarrow \quad S_i \text{ gehört zum Cover } C.$$

A besteht aus Bitvektoren der einzelnen S_i.

noch z.z.: $Ax \geq 1$ (bzw. $-Ax \leq -1$) ($b = -1$)
$\sum x_i \leq k$ (mit $c = (1, \ldots, 1)$)
$0 \leq x_i \leq 1$ für alle i

Beispiel:

Seien $U := \{1, 2, 3, 4\}$
$\mathcal{F} := \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{1, 2, 3\}, \{2, 3\}\}$

$$A^T = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{array}{l} \widehat{=} S_1 \\ \widehat{=} S_2 \\ \widehat{=} S_3 \\ \widehat{=} S_4 \\ \widehat{=} S_5 \end{array}$$

Für $x = (\ 1 \quad 1 \quad 0 \quad 0 \quad 0\ )^T$ gilt:

$$A \cdot x = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 0 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \not\geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$\Rightarrow Cx = S_1 \cup S_2 = \{1, 2, 3\} \neq U$

Für $x = (\ 0 \quad 1 \quad 1 \quad 0 \quad 0\ )^T$ gilt:

$$A \cdot x = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$\Rightarrow Cx = S_2 \cup S_3 = \{1, 2, 3, 4\} = U$

□

KNAPSACK ("Rucksack")
geg.: $W \in \mathbb{Z}$, n Gegenstände S_i mit einem Gewicht w_i und einem Wert v_i, wobei $w_i, v_i \in \mathbb{Z}$.

Frage: Gesucht ist das Maximum von $\sum_{j=1}^{m} v_{i_j}$, wobei für jede Auswahl von Gegenständen $\{S_{i_j}\}_{j=1,\ldots,m}$

$$\sum_{j=1}^{m} w_{i_j} \leq W$$

gilt.

Entscheidungsvariante von $KNAPSACK$

geg.: $W \in \mathbb{Z}$, $k \in \mathbb{Z}$, n Gegenstände S_i mit einem Gewicht w_i und einem Wert v_i, wobei $w_i, v_i \in \mathbb{Z}$.

Frage: Existiert eine Teilmenge $\{S_{i_j}\}_{j=1,\ldots,m}$ mit

$$\sum_{j=1}^{m} w_{i_j} \leq W$$

und

$$\sum_{j=1}^{m} v_{i_j} \geq k \ ?$$

Satz 2.20: $KNAPSACK$ ist $\mathcal{NP}$-vollständig.

Beweis: Wir betrachten den Spezialfall $KNAPSACK^*$:

$$v_i = w_i \text{ und } k = W$$

d. h. wir haben eine Menge von n ganzen Zahlen $w_1, \ldots, w_n$ und ein $k \in \mathbb{Z}$.
Frage: Existiert eine Auswahl von $j = 1, \ldots, m$ Zahlen w_{i_j} mit

$$\sum_{j=1}^{m} w_{i_j} = k \ ?$$

Behauptung: $KNAPSACK^*$ ist $\mathcal{NP}$-vollständig.

Wir zeigen, daß $EXACT\ COVER\ BY$ 3-$SETS \ \leq \ KNAPSACK^*$:

Sei $\{S_1, \ldots, S_n \mid \#S_i = 3\}$ die Eingabe für $EXACT\ COVER\ BY$ 3-$SETS$.

Die Frage gilt der Auswahl von m **disjunkten** Mengen S_{i_j} mit

$$\bigcup_{j=1}^{m} S_{i_j} = \{1, 2, \ldots, 3m\} = U$$

Wir repräsentieren S_i, $1 \leq i \leq n$ durch den Bitvektor $\widehat{S}_i = (s_1^{(i)}, \ldots, s_{3m}^{(i)})$ mit

$$s_l^{(i)} = \begin{cases} 1 & l \in S_i \\ 0 & sonst \end{cases}$$

Die Bitvektoren $\widehat{S}_i$ interpretieren wir als Binärzahlen $bin(\widehat{S}_i)$
$\Rightarrow$ *SET COVERING* wird zur Addition von ganzen Zahlen.
Wir suchen eine Teilmenge $\{bin(\widehat{S}_{i_j})\}_{j=1,\dots,m}$ mit

$$\sum_{j=1}^{m} bin(\widehat{S}_{i_j}) = 2^{3m} - 1 = bin(\widehat{U}) = k$$

Achtung: Es entsteht ein Übertragsproblem:

Beispiel:

$$3 + 5 + 7 = 15$$

bedeutet

$$0011 + 0101 + 0111 = 1111,$$

jedoch ist

$$\{3,4\} \ \cup \ \{2,4\} \ \cup \ \{2,3,4\} = \{2,3,4\}$$

Ausweg: Statt Bitvektoren betrachten wir Vektoren mit Elementen aus $\{0,\dots,3m\}$: $\widehat{S}_i = \sum_{l \in S_i}(3m+1)^{3m-l}, k = \sum_{l=0}^{3m-1}(3m+1)^l$, dann können keine Überträge entstehen.
□

Achtung: Jede Eingabe von *KNAPSACK* kann in der Zeit $\mathcal{O}(n \cdot W)$ gelöst werden. Aber $n \cdot W$ ist **nicht** polynomial in der Eingabelänge $\approx n \cdot \log W$.

BIN PACKING

geg.: n Gegenstände $a_1,\dots,a_n \in \mathbb{N}$, $b \in \mathbb{N}$ Kisten mit jeweils einer Kapazität $C \in \mathbb{N}$

Frage: Kann $\{a_1,\dots,a_n\}$ so in b Teilmengen (Kisten) $(K_1,\dots,K_b)$ zerlegt werden, daß die Summe der einzelnen $a_i \in K_j$, $1 \le j \le b$ gleich der Kapazität C ist?

Satz 2.21: *BIN PACKING* ist $\mathcal{NP}$-vollständig.

Beweis: Wir zeigen, daß *TRIPARTITE MATCHING* $\le$ *BIN PACKING*:

Gegeben sei eine Instanz von *TRIPARTITE MATCHING* mit den Mengen $B = \{b_1,\dots,b_n\}, G = \{g_1,\dots,g_n\}$ und $H = \{h_1,\dots,h_n\}$ sowie der Relation $T = \{t_1,\dots,t_m\} \subseteq B \times G \times H$.
Wir konstruieren eine Instanz vo *BIN PACKING* mit $N = 4n$ Gegenständen:

- einen Gegenstand für jedes Tripel,

- einen Gegenstand für jedes Vorkommen eines Elements aus B, G und H.

Wir bezeichnen die Gegenstände, die mit dem Vorkommen eines Elements, z. B. b_i, korrespondieren, mit $b_i(1), b_i(2), \ldots, b_i(N_{b_i})$, wobei N_{b_i} gleich der Anzahl der Vorkommen von b_i in den Tripeln der Relation T ist.
Ebenso definieren wir $g_i(j)$ und $h_i(k)$ für $1 \leq j \leq N_{g_i}$, $1 \leq k \leq N_{h_i}$ und $1 \leq i \leq n$.
Die Bezeichnung der Gegenstände, die die einzelnen Tripel repräsentieren, bezeichnen wir als t_j (für $1 \leq j \leq m$).
Jetzt bestimmen wir die Größe bzw. das Gewicht der einzelnen Gegenstände:

- $G(b_i(j)) = \begin{cases} 10M^4 + iM + 1 & f\ddot{u}r\ j = 1 \\ 11M^4 + iM + 1 & sonst \end{cases}$
- $G(g_i(j)) = \begin{cases} 10M^4 + iM^2 + 2 & f\ddot{u}r\ j = 1 \\ 11M^4 + iM^2 + 2 & sonst \end{cases}$
- $G(h_i(j)) = \begin{cases} 10M^4 + iM^3 + 4 & f\ddot{u}r\ j = 1 \\ 8M^4 + iM^3 + 4 & sonst \end{cases}$
- Tripel $t_l = (b_i, g_j, h_k) \in T: \ G(t_l) = 10M^4 + 8 - iM - jM^2 - kM^3$

wobei M eine sehr große Zahl, z. B. $M = 100n$ ist.
Die Konstruktion ist so gewählt, daß das erste Vorkommen eines Elements in den Tripeln der Relation ein vom Rest verschiedenes Gewicht besitzt.
Wir setzen die Gesamtkapazität einer Kiste auf $C = 40M^4 + 15$.
Dies erlaubt es gerade, genau ein Tripel sowie die im Tripel vorkommenden Elemente b_i, g_j und h_k in eine Kiste aufzunehmen, wenn entweder

- $(b_i(1), g_j(1), h_k(1))$, also immer nur das Tripel, das die ersten Vorkommen unserer Elemente enthält, oder
- $(b_i(l), g_j(m), h_k(n))$, mit $l, m, n > 1$, also immer nur ein Tripel, das keine ersten Vorkommen der jeweiligen Elemente enthält.

Die Anzahl der Kisten b ist gleich der Anzahl der Tripel: $b = m$.

Behauptung:

Das *TRIPARTITE MATCHING* enthält ein Matching
$\Leftrightarrow$ In *BIN PACKING* ist eine der Fragestellung entsprechende Aufteilung möglich.

"$\Leftarrow$": Wir betrachten eine Instanz von *BIN PACKING*, die gemäß der Fragestellung positiv lösbar ist.

Die Summe der Gewichte aller Kisten ist dann genau $C \cdot m$, so daß alle exakt mit ihrer Kapazität C gefüllt sind.

Wir betrachten eine einzelne Kiste, die gemäß der Konstruktion genau 4 Gegenstände enthält:

Es muß gelten: $C \ mod \ M = 15$. Dies kann nur auf einem Weg erreicht werden, und zwar, wenn 15 aus 4 von den jeweils zur Verfügung stehenden Resten $8, 4, 2, 1$ konstruiert werden soll:
Die Kiste muß ein Tripel enthalten ($\rightarrow$ Rest: 8) und je ein Element aus B, G und H ($\rightarrow$ Rest: $4, 2, 1$).
Jetzt stellt sich die Frage, welches Tripel $(b_i, g_j, h_k) \in T$ und welche Elemente $b_{i'}, g_{j'}, h_{k'}$ in der Kiste enthalten sind:
Da auch $C \ mod \ M^2 = 15$ gilt, muß sich beim Aufsummieren der Summand mit M wegkürzen; also muß folgen: $i = i'$.
Weiter gilt auch, daß $C \ mod \ M^3 = 15$, weshalb auch der Summand mit M^2 wegfallen muß; also gilt: $j = j'$.
Schließlich gilt auch noch, daß $C \ mod \ M^4 = 15$, weshalb der Summand mit M^3 ebenfalls wegfällt; also gilt auch: $k = k'$.
$\Rightarrow$ Die Kiste enthält folglich das Tripel $(b_i, g_j, h_k) \in T$ zusammen mit je einem Vorkommen der Tripelemente b_i, g_j und h_k.
Dabei muß es sich dann jeweils entweder um das erste Vorkommen $b_i(1)$, $g_j(1)$, $h_k(1)$ handeln, oder bei allen Elementen um ein späteres Vorkommen $b_i(l)$, $g_j(m)$, $h_k(n)$ mit $l, m, n > 1$, da sonst nicht $40M^4$ in der Kapazität erreicht werden kann.
Wir legen fest, daß $l = m = n = 1$ ist.

Nun haben wir also n Tripel ausgesucht, wobei die in den einzelnen Tripeln auftauchenden Elemente nur jeweils einmal vorkommen.
$\Rightarrow$ Wir haben ein Matching konstruiert.

"$\Rightarrow$": Wir betrachten eine Instanz von *TRIPARTITE MATCHING*, in der ein Matching existiert.

Wir nehmen ein Tripel aus dem Matching, zusammen mit den Gegenständen, die seinen einzelnen Bestandteilen zugeordnet sind, und deponieren diese jeweils in einer der m Kisten.

Nun legen wir fest, daß sich in den Tripeln des Matchings jeweils nur die ersten Vorkommen von dessen Bestandteilen befinden.

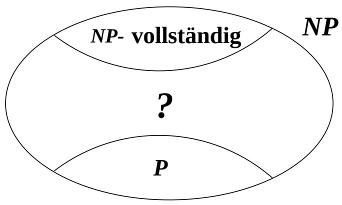
Gemäß der Konstruktion und den Gewichtszuweisungen erhalten wir eine Lösung des $BIN\ PACKING$-Problems.

Zeit- / Raumbedarf:

- Die Gewichte, die den einzelnen Gegenständen zugewiesen werden, besitzen nur polynomiale Größe bzgl. der Länge der Eingabe ($\mathcal{O}(n^4)$).
- Der benötigte Zwischenspeicher kann daher auf $\log space$ beschränkt werden.

□

2.5 $\mathcal{NP}$ und $co\,\mathcal{NP}$



$\mathcal{NP}$ - Eine Klasse von Problemen mit kurzen "Zeugnissen" (Beweisen) für eine "ja"-Antwort

$co\,\mathcal{NP}$ - Eine Klasse von Problemen mit kurzen "Zeugnissen" für eine "nein"-Antwort

Beispiel 2.5:

$VALIDITY$

geg.: ein Boolescher Ausdruck ϕ in KNF.

Frage: Ist ϕ allgemeingültig, d. h. erfüllt **jede** Belegung ϕ?

$VALIDITY$ ist in $co\,\mathcal{NP}$, da $\overline{VALIDITY(\phi)} = SAT(\neg\, \phi)$.

Satz 2.22: Ist ein Problem $H \subseteq \Sigma^*$ $\mathcal{NP}$-vollständig, dann ist $\overline{H} = \Sigma^* - H$ *co* $\mathcal{NP}$-vollständig.

Beweis:

$$H \text{ ist } \mathcal{NP}\text{-vollständig} \quad \Leftrightarrow \quad \begin{array}{ll} (1) & H \in \mathcal{NP} \\ (2) & \text{Für alle } H' \in \mathcal{NP} \text{ gilt: } H' \leq H. \end{array}$$

Aus (1) folgt: $\overline{H} \in co\,\mathcal{NP}$ nach Definition

Aus (2) folgt: Für alle $H'' \in co\,\mathcal{NP}$ gilt:

$$\overline{H''} \leq H \quad \Rightarrow \quad H'' = \overline{\overline{H''}} \leq \overline{H}$$

□

Offenbar gilt:

$$\mathcal{P} \subseteq co\,\mathcal{NP}$$

Es stellt sich die grundsätzliche **Frage**:

$$\boxed{\mathcal{NP} \stackrel{?}{=} co\,\mathcal{NP}}$$

Mögliche Hilfestellungen zu einer Beantwortung der Frage geben folgende Aspekte:

(1) Aus $\mathcal{P} = \mathcal{NP}$ würde folgen: $\mathcal{NP} = co\,\mathcal{NP}$, da $\mathcal{P}$ bzgl. der Komplementbildung abgeschlossen ist

(2) Aber auch, falls $\mathcal{P} \neq \mathcal{NP}$ ist, kann gelten: $\mathcal{NP} = co\,\mathcal{NP}$

Generelle Annahme:

$$\boxed{\mathcal{NP} \neq co\,\mathcal{NP}}$$

Satz 2.23: Sei H $co\,\mathcal{NP}$-vollständig. Dann gilt:

$$H \in \mathcal{NP} \quad \Rightarrow \quad \mathcal{NP} = co\,\mathcal{NP}$$

Beweis: Sei $H \in \mathcal{NP}$, wobei H $co\,\mathcal{NP}$-vollständig ist.

Wir zeigen, daß $co\,\mathcal{NP} \leq \mathcal{NP}$:

Sei $L \in co\,\mathcal{NP}$.

$\Rightarrow L \leq_R H \Rightarrow$ es existiert eine polynomialzeit-beschränkte NTM N, die L entscheidet: N arbeitet dabei zunächst wie die Reduktionsmaschine R und dann wie eine $\mathcal{NP}$-Maschine für H.

Die Umkehrung ($\mathcal{NP} \leq co\,\mathcal{NP}$) gilt analog (Übungsaufgabe).

□

Bemerkung (Kochrezept):
Wir erhalten aus einem $\mathcal{NP}$-Problem ein $co\,\mathcal{NP}$-Problem, wenn wir den $\exists$-Quantor durch den $\forall$-Quantor ersetzen.

Frage: Wie sieht $\mathcal{NP} \cap co\,\mathcal{NP}$ aus?

Offenbar gilt:

$$\mathcal{P} \subseteq \mathcal{NP} \cap co\,\mathcal{NP}$$

Generelle Annahme:

$$\mathcal{P} \subsetneq \mathcal{NP} \cap co\,\mathcal{NP}$$

Ein Indiz für diese Annahme ist:

$$PRIMES \in \mathcal{NP} \cap co\,\mathcal{NP},\ \text{aber } PRIMES \overset{?}{\in} \mathcal{P} \text{ ist unbekannt}$$

$PRIMES$

geg.: $N \in \mathbb{N}$ **binär verschlüsselt.**
Frage: Ist N eine Primzahl?

Ein deterministisches Verfahren zur Entscheidung von $PRIMES$ stellt das **Sieb des Erathostenes** dar.
Der Aufwand $\mathcal{O}(\sqrt{N})$ dieses Verfahrens ist jedoch nicht polynomial in der Eingabelänge $\log N$!

Satz 2.24: $PRIMES \in co\,\mathcal{NP}$

Beweis: Wir raten einen Teiler.
Dies ist möglich, falls N **keine** Primzahl ist.

$\square$

Satz 2.25 (Pratt's Theorem):

$$PRIMES \in \mathcal{NP} \cap co\,\mathcal{NP}$$

Beweis: z.z.: $PRIMES \in \mathcal{NP}$

Wir benutzen als Hilfsmittel einen **Fakt aus der der Zahlentheorie**:

$p > 1$ ist eine Primzahl $\Leftrightarrow$ Es existiert ein r $(1 < r < p)$ mit:
(1) $r^{p-1} = 1 \; mod \; p$ und
(2) **für alle Primteiler** q von $p-1$ gilt:
$$r^{\frac{p-1}{q}} \neq 1 \; mod \; p$$

Sei p eine Primzahl. Sei $q_1, \ldots, q_k$ Primteiler von $p-1$ und r eine Zahl für die Bedingung (1). Wir haben eine Folge $C(p) = (r; q_1, \ldots, q_k)$ (Zertifikat für p) und führen (1), (2) aus.

Wir wollen die Bedingungen (1) und (2) aus dem Fakt anwenden, um kurze Zertifikate für Primzahlen zu bekommen (d. h. $PRIMES$ nichtdeterministisch zu entscheiden).

Behauptung: Wir können (1) in Polynomialzeit in $l = \lceil \log p \rceil$ testen.

Wir können die Multiplikation *modulo* p durch eine gewöhnliche Multiplikation und anschließend eine gewöhnliche Division durch p ausführen.
$\Rightarrow$ Zeitbedarf: $\mathcal{O}(l^2)$ (für l-Bit-Zahlen)

Zur Berechnung von r^{p-1} wird r l-mal hintereinander quadriert:

$$r^2, r^4, \ldots, r^{2^l} \; modulo \; p$$

$\Rightarrow$ Zeitbedarf: $\mathcal{O}(l^3)$

Mit $\leq l$ zusätzlichen Multiplikationen *modulo* p kann r^{p-1} *mod* p ausgerechnet und Gleichheit mit 1 getestet werden.
$\Rightarrow$ Zeitbedarf: $\mathcal{O}(l^3)$

Behauptung: Für jedes q_i Primteiler von $p-1$ kann (2) in Polynomialzeit getestet werden.

Es bleibt zu zeigen, daß die q_i Primteiler von $p-1$ sind.
Wir raten rekursiv Zertifikate für die q_i.
Ein neues Zertifikat für p lautet: $C(p) = (r; q_1, C(q_1), \ldots, q_k, C(q_k))$.
Es gilt:

$$C(p) \text{ existiert} \quad \Leftrightarrow \quad p \text{ ist eine Primzahl}$$

noch z.z.: $|C(p)| \leq 4\, l^2$, also polynomial
$= 4 \log^2 p$

Induktion über p:

Induktionsanfang: $p = 2$, $p = 3$ klar

Induktionsschritt:

Für ein beliebiges p hat $p-1$ weniger als $\log p$ mögliche Primteiler $q_1 (= 2), \ldots, q_k$.
$C(p)$ beinhaltet

- 2 Außenklammern
- $2\,k \;\leq\; 2\,(\log p - 1)$ Kommas
- r mit höchstens $\log p$ Bits
- 10 ($q_1 = 2$) und das dazugehörige Zertifikat (1), also 6 Bits

- die q_i's , $2 \le i \le k$, mit insgesamt $(2 \log p - 4)$ Bits: wenn q_i genau a_i Bits hat, dann hat die Zahl $(p-1) \ge q_1 \cdots q_k$ (mit $\log p$ Bits) nicht weniger als $(2 + \sum_{i=2}^{k}(a_i - 1)) \ge (\sum_{i=2}^{k} a_i - \log p + 4)$ Bits
- die $C(q_i)$'s

Wegen $C(q_i) \le 4 \log^2 q_i$ nach Induktionsvoraussetzung gilt:

$$|C(p)| \le 4 \log p + 6 - 4 + 4 \cdot \sum_{i=2}^{k} \log^2 q_i$$

Wegen

$$\begin{aligned} \sum_{i=1}^{k} \log q_i &\le \log \frac{p-1}{2} \\ &\le \log p - 1 \end{aligned}$$

folgt

$$\sum_{i=1}^{k} \log^2 q_i \le (\log p - 1)^2$$

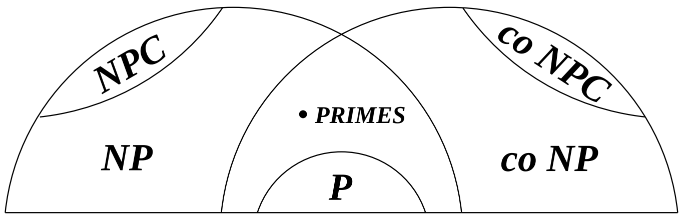
$\Rightarrow$

$$\begin{aligned} |C(p)| \le\ & 4 \log^2 p + 2 - 4 \log p \\ & 4 \log^2 p \quad \text{falls } p \ge 3 \end{aligned}$$

Tatsächlich kann $C(p)$ in Polynomialzeit ausgewertet werden.

$\square$

Zusammenfassung:
($\mathcal{NPC} \widehat{=} \mathcal{NP}$-vollständige Probleme)



2.6 Randomisierte Berechnungen

Beobachtung: Wir können Probleme effizient lösen durch Zuhilfenahme des Zufalls, d. h. "Ein Algorithums darf würfeln bzw. Münzen werfen".

2.6.1 Monte Carlo Algorithmen

Wir betrachten im folgenden die **Symbolische Determinante**.

Wir hatten bereits das Problem

BIPARTITE MATCHING

geg.: ein bipartiter Graph $G = (U, V, E)$ mit $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$ und $E \subseteq U \times V$.

Frage: Existiert ein perfektes Matching, d. h. existiert ein $M \subseteq E, |M| = n$, so daß für alle (u, v) und $(u', v') \in M$ gilt:

$$(u, v) \neq (u', v') \quad \Rightarrow \quad u \neq u' \text{ und } v \neq v' \text{ ?}$$

Anders formuliert: Existiert eine Permutation π von $\{1, \ldots, n\}$, so daß gilt:

$$(u_i, v_{\pi(i)}) \in E \text{ für alle } i \text{ ?}$$

Wir können dieses Problem in der Sprache der Matrizen und Determinanten wie folgt beschreiben:

geg.: ein bipartiter Graph $G = (U, V, E)$.

Wir betrachten eine $n \times n$ Matrix $A^G = (a_{ij})$ mit

$$a_{i,j} = \begin{cases} x_{ij} & falls\ (u_i, v_j) \in E\ (x_{ij} : Variablen) \\ 0 & sonst \end{cases}$$

Von dieser Matrix A^G betrachten wir jetzt die Determinante

$$det\ A^G = \sum_{\pi} \sigma(\pi) \prod_{i=1}^{n} a_{i,\pi(i)}$$

mit

$$\sigma(\pi) = \begin{cases} 1 & falls\ \pi\ das\ Produkt\ einer\ geraden\ Anzahl\ von\ Transpositionen\ ist \\ -1 & sonst \end{cases}$$

Offenbar gilt:

$$\prod a_{i,\pi(i)} \neq 0 \quad \Leftrightarrow \quad \pi \text{ ist ein Perfect Matching}$$

Ferner gilt: In jedem dieser Terme sind alle Variablen voneinander verschieden und die verschiedenen Terme unterscheiden sich.

$\Rightarrow$

$$\boxed{G \text{ hat ein Perfect Matching.} \quad \Leftrightarrow \quad det\ A^G \not\equiv 0}$$

$\Rightarrow$ Wir können durch die Berechnung der Determinante "symbolischer Matrizen" (symbolisch $\widehat{=}$ Variablen können vorkommen) $PERFECT\ MATCHING$ lösen.

Lösungsverfahren für Determinanten - Gauß'sche Elimination:

Idee: Wir erzeugen eine Dreiecksmatrix, z. B. durch eine Linearkombination von Zeilen, die die Determinante nicht verändert.
Die Determinante der Dreiecksmatrix ist das Produkt der Elemente auf der Hauptdiagonalen.

Zeitbedarf: A^G ist eine $n \times n$ Matrix mit Einträgen mit höchstens b Bits.

Wir können induktiv zeigen, daß alle auftretenden Ergebnisse lediglich polynomiale Länge in n und b haben.

$\Rightarrow$ Das Verfahren benötigt lediglich polynomiale Zeit.

Die Anwendung auf symbolische Matrizen ist hierbei sehr problematisch:

- Die Zwischenresultate können sich sehr aufblähen.
- Schon die Frage, ob ein spezielles Monom in der Determinante mit von 0 verschiedenen Koeffizienten auftaucht, ist $\mathcal{NP}$-vollständig.

Aber: Wir wollen eigentlich nicht die Determinante der symbolischen Matrix berechnen, sondern nur entscheiden, ob diese 0 ist oder nicht.

Idee: Wir ersetzen die Variablen x_{ij} durch beliebige Zahlen und wenden nun das Gauß'sche Eliminationsverfahren an:

Ist die Determinante $\neq 0$, so muß auch die Determinante der symbolischen Matrix $\neq 0$ sein.

Vorsicht bei der Umkehrung: Es könnte durch spezielle Wahl der Zahlen (Nullstellen des Polynoms) gelten, daß die numerische Determinante $= 0$, aber die symbolische $\neq 0$ ist.

Beispiel: $p(x) = x^2 - 1$:

$$x = 1 \quad \Rightarrow \quad p(1) = 0$$
$$x = 2 \quad \Rightarrow \quad p(2) \neq 0$$

Lemma 2.1: Sei $M \in \mathbb{Z}$ mit $M > 0$ und $p(x_1, \ldots, x_m)$ ein Polynom mit $p \not\equiv 0$. Jedes x_i habe höchstens den Grad d. Dann gilt:

Die Zahl der Tupel $(a_1, \ldots, a_m) \in \{0, 1, \ldots, M-1\}^m$ mit $p(a_1, \ldots, a_m) = 0$ ist $\leq m \cdot d \cdot M^{m-1}$.

Beweis: durch Induktion über m

$m = 1$: Die Aussage ist gut bekannt: Kein Polynom vom Grad d kann mehr als d Nullstellen haben (dies gilt auch *modulo* p).

$m - 1 \hookrightarrow m$:
Wie oft kann $p(a_1, \ldots, a_m) = 0$ für $a_1, \ldots, a_m \in \{0, \ldots, M-1\}$ gelten?

Wir fassen p als Polynom $\widetilde{p}$ in x_m auf:
$\widetilde{p}$ ist ein Polynom, dessen Koeffizienten Polynome in $x_1, \ldots, x_{m-1}$ sind.

$$\begin{aligned} p(x_1, \ldots, x_m) &= \widetilde{p}(x_m) \\ &= p_0(x_1, \ldots, x_{m-1}) + \ldots + p_d(x_1, \ldots, x_{m-1}) \cdot x_m^d \end{aligned}$$

Es gilt: $deg\ p_i \leq d$.

1. Fall: Für alle i, $1 \leq i \leq d$, gilt: $p_i(a_1, \ldots, a_{m-1}) = 0$. Dann $p(a_1, \ldots, a_m) = p_0(a_1, \ldots, a_{m-1})$.
$\Rightarrow$ Nach der Induktionsvoraussetzung gibt es höchstens $(m-1) \cdot d \cdot M^{m-2}$ viele Tupel $(a_1, \ldots, a_{m-1})$, für die $p_0(a_1, \ldots, a_{m-1}) = 0$ gilt.
$\Rightarrow$ Da x_m M mögliche Werte haben kann, gibt es höchstens $(m-1) \cdot d \cdot M^{m-2} \cdot M = (m-1) \cdot d \cdot M^{m-1}$ Tupel $(a_1, \ldots, a_m)$ mit $p(a_1, \ldots, a_m) = 0$ in diesem Fall.

2. Fall: O. B. d. A. gilt: $p_d(a_1, \ldots, a_{m-1}) \neq 0$.
Wegen $deg\ p \leq d$ hat p für jede Kombination der $x_1, \ldots, x_{m-1}$ höchstens d Nullstellen, d. h. für höchstens $d \cdot M^{m-1}$ Tupel $(a_1, \ldots, a_m)$ gilt: $p(a_1, \ldots, a_m) = 0$.

1. und 2. Fall geben insgesamt höchstens $(m-1) \cdot d \cdot M^{m-1} + d \cdot M^{m-1} = m \cdot d \cdot M^{m-1}$ Tupel mit $p(a_1, \ldots, a_m) = 0$. $\square$

Durch dieses Lemma wird ein **randomisierter Algorithmus** für *PERFECT MATCHING* induziert:

Sei $G = (V, E)$ und $A^G(x_1, \ldots, x_m)$ die zugehörige symbolische Matrix.
Ferner sei $deg(det\ A^G) \leq 1$, da jede Variable höchstens vom Grad 1 ist. $M = 2m$.

Algorithmus A:

- Wähle zufällig m Zahlen $i_1, \ldots, i_m$ zwischen 0 und $M - 1$.
- Berechne die Determinante $det\ A^G(i_1, \ldots, i_m)$ mit dem Gauß'schen Eliminationsverfahren.
- Wenn $det\ A^G(i_1, \ldots, i_m) \neq 0$ gilt, dann ist die Ausgabe: "G hat ein Perfect Matching."

- Wenn $det\ A^G(i_1, \ldots, i_m) = 0$ gilt, dann ist die Ausgabe: "G hat kein Perfect Matching."

Bemerkungen:

- Der Algorithmus ist vom Typ "Monte Carlo", denn die Antwort "G hat ein Perfect Matching" ist immer korrekt.
- Die Antwort von A "G hat **kein** Perfect Matching" gilt nur mit einer Wahrscheinlichkeit $\geq \frac{1}{2}$ (da $\frac{m \cdot d \cdot M^{m-1}}{M^m} = \frac{m \cdot d}{M}$ und $M = 2m = 2md$).
- Durch Vergrößerung von M gegenüber m kann die Fehlerwahrscheinlichkeit gedrückt werden.
- Auch die Zahl der unabhängigen Experimente verkleinert die Fehlerwahrscheinlichkeit.
- k unabhängige Experimente ergeben eine Fehlerwahrscheinlichkeit von $\leq \frac{1}{2}^k$.

2.6.2 Random Walks

Wir betrachten einen randomisierten Algorithmus für SAT:

- Starte mit einer Wahrheitswertebelegung T und wiederhole r-mal:
  - Gibt es keine unerfüllte Klausel, dann ist die Ausgabe: "Die Formel ist erfüllbar."
  - Ansonsten nimm eine unerfüllte Klausel.
    $\Rightarrow$ Alle Literale in dieser Klausel sind $false$ unter T.
  - Wähle ein beliebiges dieser Literale und wechsle den Wahrheitswert.
  - Aktualisiere T.
- Nach r Wiederholungen ist die Ausgabe: "Die Formel ist unerfüllbar."

Analyse:
Gibt der Algorithmus die Antwort "erfüllbar", so ist die Formel erfüllbar.
Wir betrachten den Fall mit der Antwort "unerfüllbar":
Ein Random Walk Algorithmus für SAT ist im allgemeinen nicht sehr zuverlässig. Aber es gilt:

Satz 2.26: Für $r = n^2$ liefert ein Random Walk Algorithmus für 2-SAT-Formeln eine erfüllende Belegung mit einer Wahrscheinlichkeit $> \frac{1}{2}$.

Beweis: Übungsaufgabe

□

2.6.3 Fermat Test

Wir betrachten das Problem $COMPOSITE = \overline{PRIMES}$.

Wir können einen randomisierten Algorithmus für das zahlentheoretische Problem "Ist die Eingabe eine zusammengesetzte Zahl?" angeben:

Satz 2.27: Der folgende Monte Carlo Algorithmus arbeitet in Polynomialzeit und entscheidet mit einer Fehlerwahrscheinlichkeit $< \frac{1}{2}$, ob eine eingegebene Zahl zusammengesetzt ist:

- Sei N gegeben mit $N = q_1 \cdot \ldots \cdot q_n$.
- Generiere zufällig eine Zahl M zwischen 2 und $N-1$.
- Berechne $ggt(M, N)$.
- Ist $ggt(M, N) > 1$, dann ist die Ausgabe: "N ist zusammengesetzt."
- Ansonsten berechne
  $$M^{\frac{N-1}{2}} \; mod \; N$$
  und
  $$(M|N) := \prod_{i=1}^{n}(M|q_i) \quad \text{mit} \quad (M|q_i) = M^{\frac{q_i-1}{2}} \; mod \; q_i \; (\in \{-1, +1\})$$

  Bemerkungen:
  Man benutzt dafür folgende Eigenschaften dieser Operation:

  (a) $(2|M) = (-1)^{\frac{M^2-1}{8}}$;
  (b) $(M + N|N) = (M|N)$;
  (c) $(M_1M_2|N) = (M_1|N))(M_2|N)$;
  (d) Wenn M und N ungerade Zahlen sind, dann gilt $(M|N)\cdot(N|M) = (-1)^{\frac{M-1}{2}\frac{N-1}{2}}$.

- Sind beide Ausdrücke nicht gleich, dann antworte: "N ist zusammengesetzt".
- Ansonsten antworte: "N ist (wahrscheinlich) eine Primzahl."

2.7 Randomisierte Komplexitätsklassen

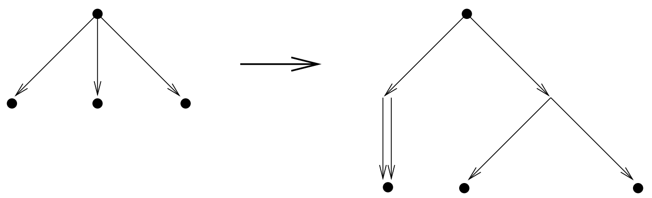
Wir hatten gezeigt, daß wir uns jede **polynomial**-zeitbeschränkte NTM o. B. d. A. als

(a) präzise (d. h. bei jeder Eingabe wird nach der gleichen Zahl von Schritten gehalten) und

(b) in jedem Schritt genau 2 nichtdeterministische Wahlmöglichkeiten besitzend

vorstellen (**standardisierte NTM**).

Beispiel: zu (b):



Wir betrachten im folgenden stets eine standardisierte NTM.

Definition 2.9:

$L \in \mathcal{RP} \underset{def.}{=}$ es existiert eine (standardisierte) $p(n)$-zeitbeschränkte NTM N, wobei p ein Polynom ist, mit

$x \in L$	$:\Leftrightarrow$	Wenigstens die Hälfte der $2^{p(\lvert x\rvert)}$ Berechnungen von N ist akzeptierend.
$x \notin L$	$:\Leftrightarrow$	Alle Berechnungen sind verwerfend.

Bemerkungen:

- Falsche Eingaben werden mit Sicherheit verworfen.
  Richtige Eingaben werden durch einen Majoritätsentscheid entschieden.
- $\mathcal{RP}$ beschreibt die Komplexitätsklasse der Probleme, für die ein polynomial-zeitbeschränkter Monte Carlo Algorithmus existiert.

Interpretation: Nichtdeterministische Wahlen werden als Münzwürfe angesehen. Dabei wird jedes Blatt des Berechnungsbaumes mit der Wahrscheinlichkeit $\frac{1}{2^{p(|x|)}}$ erreicht.

$$\begin{aligned} \mathcal{RP} = \{L : x \in L \;&:\Leftrightarrow\; Pr(x \textit{ wird akzeptiert}) \geq \tfrac{1}{2}, \\ x \notin L \;&:\Leftrightarrow\; Pr(x \textit{ wird verworfen}) = 1 \;\Leftrightarrow\; Pr(x \textit{ wird akzeptiert}) = 0\} \end{aligned}$$

Satz 2.28: Es gilt: $\mathcal{P} \subseteq \mathcal{RP} \subseteq \mathcal{NP}$.

Beweis:
Jede deterministische Berechnung kann als Monte Carlo Algorithmus angesehen werden, bei der der Ausgang des Münzwurfs ignoriert wird.
Jede Monte Carlo Berechnung ist nach Definition nichtdeterministisch.

□

Frage: Ist $\mathcal{RP}$ abgeschlossen bzgl. der Komplementbildung?

Definition 2.10:

$$\mathcal{ZPP} := \mathcal{RP} \cap co\,\mathcal{RP} \quad (\mathcal{ZPP} \mathrel{\widehat{=}} Zero\ Probabilistic\ Polynomialtime)$$

Beispiel 2.6: $PRIMES \in \mathcal{ZPP}$

Wir haben für jedes Problem in $\mathcal{ZPP}$ zwei Monte Carlo Algorithmen:

- ein Algorithmus für die Entscheidung $x \in L$
- ein Algorithmus für die Entscheidung $x \notin L$

Nach k unabhängigen Läufen beider Algorithmen ist die Wahrscheinlichkeit für eine falsche Antwort $\leq 2^{-k}$.

Dieser "Doppelpack"-Algorithmus ist vom Typ "Las Vegas".

Definition 2.11:

$L \in \mathcal{PP} \underset{def.}{=}$ es existiert eine (standardisierte) NTM N mit

$x \in L \quad :\Leftrightarrow$ **Mehr** als die Hälfte der Berechnungen von N bei der Eingabe x ist akzeptierend, d. h.
$Pr(N\ akzeptiert\ x) > \frac{1}{2}$.

$MAJ\ SAT$

geg.: eine Boolesche Formel in KNF.
Frage: Ist mehr als die Hälfte aller Belegungen erfüllend?

Beispiel 2.7:

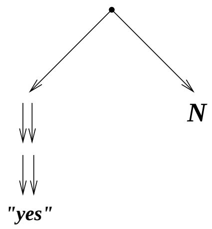
- $MAJ\ SAT \in \mathcal{PP}$
- $MAJ\ SAT$ ist $\mathcal{PP}$-vollständig (Übungsaufgabe*. **Hinweis.** Beschreiben Sie eine NTM durch eine KNF.)

Satz 2.29: $\mathcal{NP} \subseteq \mathcal{PP}$

Beweis: Sei $L \in \mathcal{NP}$ berechnet durch eine polynomial-zeitbeschränkte NTM N.

Wir konstruieren eine $\mathcal{PP}$-Maschine N' für L:

N' ist identisch mit N, mit dem einzigen Unterschied, daß N' einen neuen initialen Zustand hat, aus dem nichtdeterministisch in den initialen Zustand von N oder direkt in den akzeptierenden Endzustand übergegangen werden kann.



Analyse:

Sei x die Eingabe von N'.

N arbeitet auf x $p(|x|)$ viele Schritte und hat $2^{p(|x|)}$ viele Berechnungen; N' hat $2^{p(|x|)+1}$ viele Berechnungen.

Wenigstens die Hälfte der Berechnungen von N' ist akzeptierend.

$$\Rightarrow \quad N' \text{ akzeptiert} \Leftrightarrow N \text{ hat wenigstens eine akzeptierende Berechnung} \Leftrightarrow x \in L$$

□

Offenbar gilt:

$$\mathcal{PP} = co\,\mathcal{PP}$$

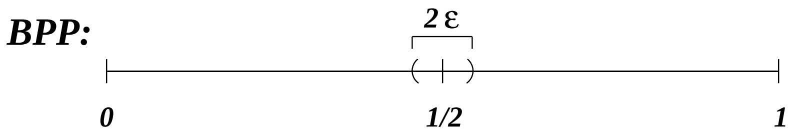
(Die Definition ist symmetrisch: Aus einer $\mathcal{PP}$-Maschine für L kann durch Vertauschung der "*yes*"- und "*no*"-Zustände sofort eine $\mathcal{PP}$-Maschine für $\overline{L}$ konstruiert werden!)

Definition 2.12:

$L \in \mathcal{BPP}$ ($\mathcal{BPP} \mathrel{\hat{=}} Bounded\ Probabilistic\ Polynomialtime$)

$\underset{def.}{=}$ es existiert ein $\varepsilon > 0$ und eine (standardisierte) polynomial-zeitbeschränkte NTM N mit

$$x \in L \quad :\Leftrightarrow \quad Pr(N\ akzeptiert\ x) \geq \tfrac{1}{2} + \varepsilon$$
$$x \notin L \quad :\Leftrightarrow \quad Pr(N\ verwirft\ x) \geq \tfrac{1}{2} + \varepsilon, \text{ d.h. } Pr(N\ akzeptiert\ x) \leq \tfrac{1}{2} - \varepsilon$$



Satz 2.30: Es gilt:

$$\mathcal{RP} \overset{(1)}{\subseteq} \mathcal{BPP} \overset{(2)}{\subseteq} \mathcal{PP}$$

Beweis:

(2) trivial

(1) Der $\mathcal{RP}$-Algorithmus wird zweimal ausgeführt.

$\Rightarrow x \in L$: Dann wird x mit einer Wahrscheinlichkeit $\geq 1 - \frac{1}{4}$ akzeptiert.

$x \notin L$: Dies wird nach wie vor richtig beantwortet.

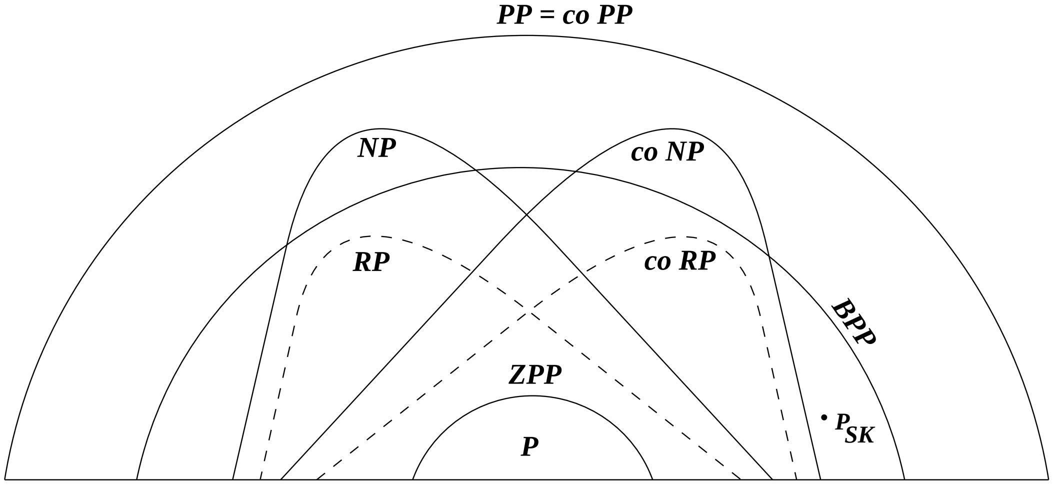
$\square$

Offenbar gilt:

$$\mathcal{BPP} = co\,\mathcal{BPP}$$

Zusammenfassung:

($\mathcal{P}_{SK}$ korrespondiert mit polynomialen Schaltkreisen)



2.8 Die Polynomialzeithierarchie

Wir wollen die "$\mathcal{P} \overset{?}{=} \mathcal{NP}$"-Frage weiter untersuchen.

Dazu betrachten wir ein TM-Modell, das mit Hilfe eines Orakels in der Lage ist, "ohne Berechnungsaufwand" Antworten auf schwere Fragen zu geben.

Frage: Welche Berechnungskraft haben solche Maschinen?

Definition 2.13: Eine Orakel-TM $M^?$ ist eine deterministische Mehrband-TM mit einem speziellen "Query"-Band und drei "Query"-Zuständen $q_?$, q_{yes} und q_{no}.

Sei $A \subseteq \Sigma^*$ ein beliebiges "Orakel".
M^A arbeitet wie eine gewöhnliche TM, solange sie nicht im Query-Zustand ist.
Erreicht M^A den Query-Zustand $q_?$, dann geht M^A in den Zustand q_{yes} bzw. q_{no} über, je nachdem, ob für den String a auf dem Query-Band gilt: $a \in A$ bzw. $a \notin A$.
Die Zeitkomplexität ist definiert wie immer; auch der Query-Schritt zählt wie ein gewöhnlicher Schritt.

Definition 2.14: Sei K eine Komplexitätsklasse und $A \subseteq \Sigma^*$.
K^A ist die Klasse aller Sprachen, die mit dem gleichen Aufwand wie Sprachen in K entschieden werden können, wobei die Benutzung des Orakels A erlaubt ist.

Offenbar gilt:

$$K^\emptyset = K$$

Beispiel 2.8:

$$\mathcal{P}^\emptyset = \mathcal{P} \text{ und } \mathcal{NP}^\emptyset = \mathcal{NP}$$

Satz 2.31: Es gibt ein Orakel A, so daß gilt:

$$\mathcal{P}^A = \mathcal{NP}^A$$

Beweis: Wir nehmen für A die $\mathcal{PSPACE}$-vollständigen Sprachen.
Erinnerung: Nach dem Satz von Savitch (Satz 1.11) gilt:

$$\mathcal{PSPACE} = \mathcal{NPSPACE}$$

$$\Rightarrow \qquad \mathcal{PSPACE} \overset{(1)}{\subseteq} \mathcal{P}^A \overset{(2)}{\subseteq} \mathcal{NP}^A \overset{(3)}{\subseteq} \mathcal{NPSPACE} = \mathcal{PSPACE}$$

(1) Sei $L \in \mathcal{PSPACE}$.
Es existiert eine deterministische polynomial-zeitbeschränkte TM M für die Reduktion $L \leq A$.
Wir können L mit M^A entscheiden:
Dazu berechnen wir $L \leq A$ und fragen das Orakel.
$\Rightarrow L \in \mathcal{P}^A$

(2) trivial

(3) Wir können eine $\mathcal{NP}^A$-Orakelmaschine durch eine $\mathcal{NPSPACE}$-Maschine simulieren:
Anstelle der Anfrage $a \overset{?}{\in} A$ an das Orakel A wird $a \in A$ ausgerechnet.
Dies ist möglich in $\mathcal{NPSPACE}$, da A $\mathcal{PSPACE}$-vollständig ist und da gilt:

$$\mathcal{NPSPACE} = \mathcal{PSPACE}$$

□

Satz 2.32: Es gibt ein Orakel B, so daß gilt:

$$\mathcal{P}^B \neq \mathcal{NP}^B$$

Beweisskizze: Wir suchen zu dem Orakel B eine Sprache L mit $L \in \mathcal{NP}^B - \mathcal{P}^B$:

Sei $L = \{0^n \mid \text{es existiert ein } x \in B \text{ mit } |x| = n\}$.

Offenbar gilt $L \in \mathcal{NP}^B$:

Zur Eingabe 0^n wird ein x geraten und $x \overset{?}{\in} B$ mit Hilfe des Orakels entschieden.

Wir definieren B so, daß $L \notin \mathcal{P}^B$.

Dies ist möglich mit dem Verfahren der Diagonalisierung, indem man alle TM mit Orakel nummeriert. (Übungsdaufgabe)

□

Bemerkungen:

- Die letzten beiden Sätze zeigen, daß die "$\mathcal{P} \overset{?}{=} \mathcal{NP}$"-Frage nicht mit Hilfe der Orakel-Technik gelöst werden kann.
- Es ist kein Beweis möglich, der sich auf Orakelmaschinen übertragen läßt.

Definition 2.15:
Sei K eine Komplexitätsklasse und A ein K'-vollständiges Problem für eine Komplexitätsklasse K'. Dann setzen wir

$$K^{K'} := K^A$$

(Anstelle von A hätte jedes K'-vollständige Problem als Orakel genommen werden können, ohne daß sich die Klasse ändert!)

Beispiel 2.9:

(1) $\mathcal{P}^{\mathcal{NP}}$ ist eine deterministische Komplexitätsklasse, wobei das Orakel eine beliebige $\mathcal{NP}$-vollständige Menge ist, z. B. SAT.

(2) $\mathcal{NP}^{\mathcal{NP}}$ ist eine nichtdeterministische Komplexitätsklasse.
Frage: Ist $\mathcal{NP}^{\mathcal{NP}}$ abgeschlossen bzgl. der Komplementbildung?

Offenbar gilt:

$$\mathcal{P}^{\mathcal{P}} = \mathcal{P} \text{ und } \mathcal{NP}^{\mathcal{P}} = \mathcal{NP}$$

Allgemein gilt:

$$K^{K'} = K \text{ für } K' \subseteq K$$

Definition 2.16:
Die **Polynomialzeithierarchie** ist die folgende Folge von Komplexitätsklassen:

- $\Delta_0\mathcal{P} \;=\; \Sigma_0\mathcal{P} \;=\; \Pi_0\mathcal{P} \;=\; \mathcal{P}$
- $\Delta_{i+1}\mathcal{P} \;=\; \mathcal{P}^{\Sigma_i\mathcal{P}}$
- $\Sigma_{i+1}\mathcal{P} \;=\; \mathcal{NP}^{\Sigma_i\mathcal{P}}$
- $\Pi_{i+1}\mathcal{P} \;=\; co\,\mathcal{NP}^{\Sigma_i\mathcal{P}}$

für alle $i > 0$.

Schließlich ist
$$\mathcal{PH} \;=\; \bigcup_{i\geq 0}\Sigma_i\mathcal{P}$$

Offenbar gilt:

Die 0. Ebene der Polynomialzeithierarchie ist
$$\Sigma_0\mathcal{P} \;=\; \mathcal{P}$$
Die 1. Ebene der Polynomialzeithierarchie ist
$$\Delta_1\mathcal{P} \;=\; \mathcal{P},\;\; \Sigma_1\mathcal{P} \;=\; \mathcal{NP},\;\; \Pi_1\mathcal{P} \;=\; co\,\mathcal{NP}$$
Die 2. Ebene der Polynomialzeithierarchie ist
$$\Delta_2\mathcal{P} \;=\; \mathcal{P}^{\mathcal{NP}},\;\; \Sigma_2\mathcal{P} \;=\; \mathcal{NP}^{\mathcal{NP}},\;\; \Pi_2\mathcal{P} \;=\; co\,\mathcal{NP}^{\mathcal{NP}}$$

Satz 2.33: Sei $L \subseteq \Sigma^*$ und $i \geq 1$.

$$\begin{array}{lcl} L \in \Sigma_i\mathcal{P} & \Leftrightarrow & \text{Es gibt eine polynomial balancierte Relation } R, \text{ so daß gilt:} \\ & & \{x; y \mid (x,y) \in R\} \;\in\; \Pi_{i-1}\mathcal{P} \text{ und} \\ & & L = \{x \mid \text{es gibt ein } y \text{ mit } (x,y) \in R\} \end{array}$$

Beweis: durch Induktion über i

$i = 1$: Der Sachverhalt ist bereits bewiesen.

$i - 1 \hookrightarrow i$:

"$\Leftarrow$": Seien $i > 1$ und R gegeben.

z.z.: $L \;\in\; \Sigma_i\mathcal{P}$

g.z.z.: Es existiert eine polynomial-zeitbeschränkte NTM N mit einem $\Sigma_{i-1}\mathcal{P}$-Orakel, die L entscheidet.

Dazu rät N bei der Eingabe x ein y und fragt das $\Sigma_{i-1}\mathcal{P}$-Orakel, ob $(x,y) \in R$ (eigentlich $(x,y) \notin R$, da R eine $\Pi_{i-1}\mathcal{P}$-Relation ist).

"$\Rightarrow$": Sei $L \in \Sigma_i\mathcal{P}$.

z.z.: Es existiert eine polynomial balancierte Relation R mit "$R \in \Pi_{i-1}\mathcal{P}$".

L kann in Polynomialzeit von einer Orakel-NTM $M^?$ mit einem Orakel $K \in \Sigma_{i-1}\mathcal{P}$ entschieden werden.

Wegen $K \in \Sigma_{i-1}\mathcal{P}$ existiert nach der Induktionsvoraussetzung eine Relation $S \in \Pi_{i-2}\mathcal{P}$ mit

$$z \in K \quad \Leftrightarrow \quad (z, w) \in S \text{ für ein } w$$

Wir konstruieren eine Relation R für L:

$$x \in L \quad \Leftrightarrow \quad \text{Es existiert eine akzeptierende Berechnung von } M^K \text{ für ein } x.$$

Sei y eine Beschreibung dieser Berechnung; y hat polynomiale Länge.
Die Berechnung von M^K enthält Anfragen an das Orakel $K \in \Sigma_{i-1}\mathcal{P}$.
Wir ersetzen jede "*yes*"-Antwort für z_i in y durch ein Zertifikat w_i für z_i, mit $(z_i, w_i) \in S$.
Jetzt vervollständigen wir y und erhalten den String $\widetilde{y}$.
Wir setzen

$$(x, \widetilde{y}) \in R \;\Leftrightarrow\; \begin{array}{l}\widetilde{y} \text{ beschreibt eine vervollständigte akzeptierende} \\ \text{Berechnung von } M^K \text{ für } x\end{array}$$

R ist polynomial balanciert.

Behauptung: $R \in \Pi_{i-1}\mathcal{P}$

Wir müssen zuerst prüfen, ob jeder Schritt von M^K legal ist (dies ist möglich in deterministischer Polynomialzeit).
Dann müssen wir für polynomial viele Paare (z_i, w_i) entscheiden, ob $(z_i, w_i) \in S$ (dies ist möglich in $\Pi_{i-2}\mathcal{P} \subseteq \Pi_{i-1}\mathcal{P}$).
Nun müssen wir für alle "*no*"-Antworten z_i' testen, ob $z_i' \notin K$ (wegen $K \in \Sigma_{i-1}\mathcal{P}$ ist dies möglich in $\Pi_{i-1}\mathcal{P}$).

$$\Rightarrow \qquad (x, y) \in R \;\Leftrightarrow\; \begin{array}{l}\text{Verschiedene } \Pi_{i-1}\mathcal{P}\text{-Orakel-Anfragen werden} \\ \text{mit "}yes\text{" beantwortet.}\end{array}$$

Es ist leicht zu sehen, daß viele Fragen durch eine einzige (komplexe) $\Pi_{i-1}\mathcal{P}$-Frage zusammengefaßt werden knnen.

□

Dual gilt:

Korollar: Sei $L \subseteq \Sigma^*$ und $i \geq 1$. Dann gilt:

$L \in \Pi_i\mathcal{P} \Leftrightarrow$ Es gibt eine polynomial balancierte Relation R, so daß gilt:
$\{x; y \mid (x, y) \in R\} \in \Sigma_{i-1}\mathcal{P}$ und
$L = \{x \mid$ für alle y mit $|y| \leq |x|^k$ gilt: $(x, y) \in R\}$

Korollar: Sei $L \subseteq \Sigma^*$ und $i \geq 1$. Dann gilt:

$L \in \Sigma_i\mathcal{P} \Leftrightarrow$ Es existiert eine polynomial balancierte polynomialzeit-entscheidbare $(i+1)$-äre Relation R, so daß gilt:
$L = \{x \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i$ mit $(x, y_1, y_2, \ldots, y_i) \in R\}$,
wobei $Q = \forall$, falls $i = 2m$ und $Q = \exists$ sonst.

Beweis: Wir ersetzen sukzessive $\Pi_j\mathcal{P}$ bzw. $\Sigma_j\mathcal{P}$ gemäß Satz 2.33 und dem daraus folgenden Korollar.

□

Bedeutung der Polynomialzeithierarchie:

Satz 2.34: Gilt für ein $i \geq 1$ $\Sigma_i\mathcal{P} = \Pi_i\mathcal{P}$, dann gilt für alle $j \geq i$:

$$\Sigma_j\mathcal{P} = \Pi_j\mathcal{P} = \Delta_j\mathcal{P} = \Sigma_i\mathcal{P}$$

Beweis:
g.z.z.: Aus $\Sigma_i\mathcal{P} = \Pi_i\mathcal{P}$ folgt $\Sigma_{i+1}\mathcal{P} = \Pi_{i+1}\mathcal{P}$

Dies ist leicht möglich mit Hilfe der vorigen Sätze.
Übungsaufgabe: Beweisen Sie andere Gleichungen. □

$\Rightarrow$ Man sagt: "Die Polynomialzeithierarchie kollabiert (auf der i-ten Ebene)."

Korollar: Gilt $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{NP} = co\,\mathcal{NP}$, dann kollabiert die Polynomialzeithierarchie bereits auf der 1. Ebene.

Bedeutung vollständiger Probleme bei der Klärung dieser Fragen:

Satz 2.35: Existiert ein $\mathcal{PH}$-vollständiges Problem, dann kollabiert die Polynomialzeithierarchie auf endlicher Ebene.

Beweis: Sei L ein $\mathcal{PH}$-vollständiges Problem.
Wegen $L \in \mathcal{PH}$ existiert ein $i > 0$ mit

$$L \in \Sigma_i\mathcal{P}.$$

Für ein beliebiges $L' \in \Sigma_{i+1}\mathcal{P}$ gilt: $L' \leq L$.
$\Rightarrow L' \in \Sigma_i\mathcal{P}$
$\Rightarrow \Sigma_{i+1}\mathcal{P} \subseteq \Sigma_i\mathcal{P}$
Übungsaugabe. Beweisen Sie, dass Voraussetzungen des Satzes 2.34 gelten. □

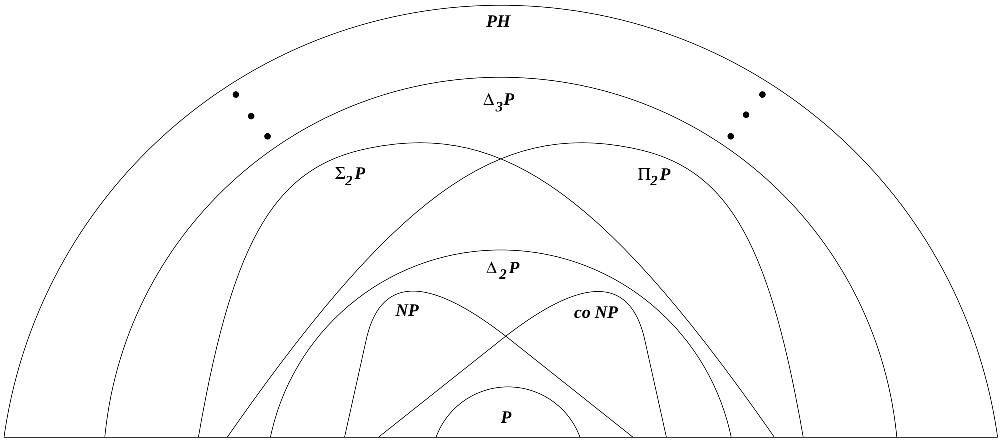
Satz 2.36: $\mathcal{PH} \subseteq \mathcal{PSPACE}$

Beweis: Auf dem Arbeitsband werden der Reihe nach alle möglichen Worte polynomialer Länge aufgeschrieben und untersucht, ob sie eine Lösung darstellen.

□

Satz 2.37: $\mathcal{BPP} \subseteq \Sigma_2\mathcal{P}$ (Übungsaufgabe**).

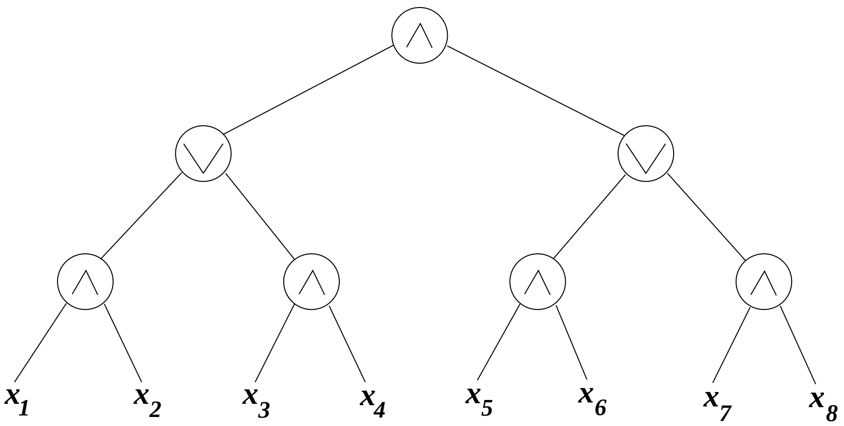
Zusammenfassung:



2.9 $\mathcal{L}$ und $\mathcal{NL}$

Definition 2.17:

$$PT/WK(f(n), g(n)) := \{L \subseteq \{0,1\}^* \mid \text{Es existiert eine SK-Familie } C\text{, die } L \text{ in } \mathcal{O}(f(n))\text{-paralleler Zeit und dem Gesamtaufwand } \mathcal{O}(g(n)) \text{ entscheidet.}\}$$



Definition 2.18:

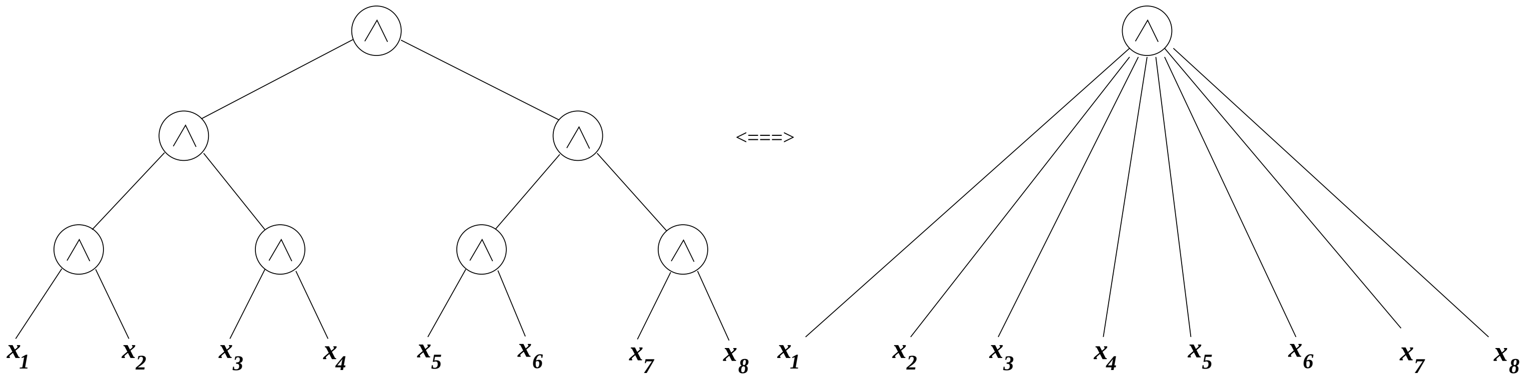
$\mathcal{NC}_j := PT/WK(\log^j, n^k)$: Dies sind die Probleme, die in polylogarithmischer paralleler Zeit und polynomialem Gesamtaufwand lösbar sind. (Nick Pippenger, 1979)

Definition 2.19:

$$\boxed{\mathcal{NC} := \bigcup_j \mathcal{NC}_j}$$

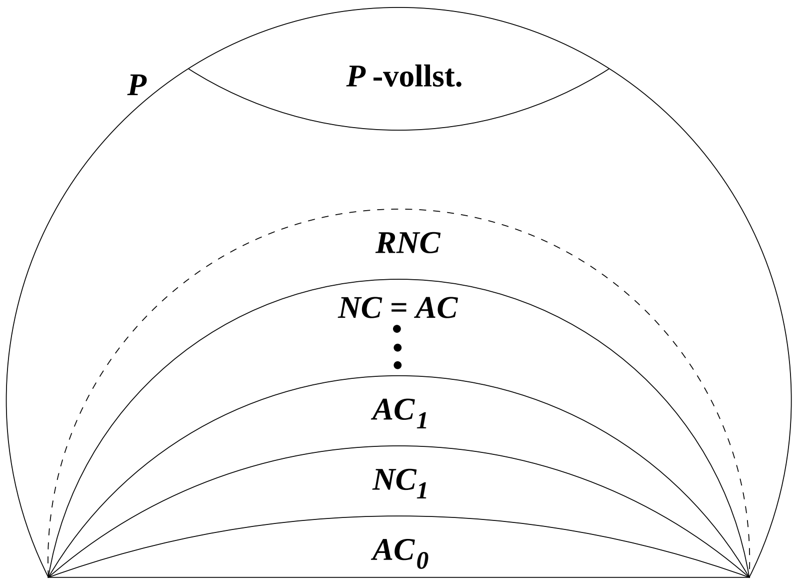
Definition 2.20:

$\mathcal{AC}$:= {Schaltkreise mit:
- unbeschränktem Fan-in,
- polylogarithmischer Berechnungszeit,
- polynomialem Gesamtaufwand}



Hierarchie innerhalb von $\mathcal{P}$:

$\mathcal{RNC}$ ist die randomisierte Variante von $\mathcal{NC}$.



2.9.1 Das Problem $\mathcal{L} \stackrel{?}{=} \mathcal{NL}$

Es ist bekannt, daß $\mathcal{L}$ und $\mathcal{NL}$ in $\mathcal{NC}$ liegen.

Satz 2.38:

$$\mathcal{NC}_1 \overset{(1)}{\subseteq} \mathcal{L} \overset{(2)}{\subseteq} \mathcal{NL} \overset{(3)}{\subseteq} \mathcal{NC}_2$$

Beweis:

(2) trivial

(3) Um zu entscheiden, ob eine Eingabe x von einer log *space*-beschränkten NTM N akzeptiert wird, konstruieren wir den Konfigurationsgraphen von N auf der Eingabe x und überprüfen, ob der akzeptierte Knoten vom Ursprungsknoten aus erreichbar ist.

Lemma 2.2: $REACHABILITY \in \mathcal{NC}_2$ (d. h. $REACHABILITY$ kann in paralleler Zeit $\mathcal{O}(\log^2 n)$ berechnet werden).

Beweis:
Probleme:

- Suchverfahren können nicht ausreichend parallelisiert werden.
- Auch wenn viele Prozessoren benutzt werden, die Knoten optimal arrangiert werden und wir die Knoten simultan bearbeiten, ist die parallele Schrittzahl mindestens so groß wie der kürzeste Weg vom Start- zum Zielpunkt.

Idee: Matrizenmultiplikation

Sei A die Adjazenzmatrix des Graphen.

Wir fügen Schlingen $a_{ii} = 1$ für alle i hinzu.
Dann berechnen wir

$A^2 = A \cdot A$ mit

$$a_{ij}^{(2)} = \bigvee_{k=1}^{n} a_{ik} \wedge a_{kj},$$

wobei gilt:

$$a_{ij}^{(2)} = 1 \quad \Leftrightarrow \quad \text{Es existiert ein Pfad der Länge 2 von } i \text{ nach } j$$

$A^4 = A^2 \cdot A^2$ – Dies ergibt Pfade der Länge 4.

$\vdots$

$A^{2^{\lceil \log n \rceil}}$ – Wir erhalten nach $\lceil \log n \rceil$ Schritten die Adjazenzmatrix der transitiven Hülle, d. h. wir konzentrierten die Antwort auf alle möglichen Instanzen von *REACHABILITY* in A.

$\Rightarrow$ Die transitive Hülle kann in $\mathcal{O}(\log^2 n)$ Schritten mit dem Gesamtaufwand $\mathcal{O}(n^3 \cdot \log n)$ berechnet werden.
$\Rightarrow$ $REACHABILITY \in \mathcal{NC}_2$.

□

(1) Wir müssen einen Algorithmus finden, der jede uniforme SK-Familie logarithmischer Tiefe in logarithmischem Raum simuliert.

Dieser Algorithmus setzt sich aus drei Teilen zusammen:

1. Aus der gegebenen SK-Familie wird ein SK generiert:

   Der SK wird repräsentiert als Gatterliste, wobei zu jedem Gatter der Typ und die Liste der Vorgänger gespeichert werden. Dabei gilt:

   – *true*/*false* haben keine Vorgänger.
   – *NOT* hat genau einen Vorgänger.
   – *AND*/*OR*-Gatter haben jeweils 2 Vorgänger.

   Das erste Gatter in der Liste ist das *OUTPUT*-Gatter.
   In einem SK kann ein Gatter mehrere Nachfolger haben ($AUSGRAD \geq 1$).

2. Der Schaltkreis wird in einen äquivalenten SK transformiert, dessen Gatter alle den $AUSGRAD = 1$ besitzen:

Betrachte Pfade im Original-SK vom $OUTPUT$ zum $INPUT$.
Repräsentiere die Gatter entlang dem Pfad als Bit-Strings, die der Länge des Pfades entsprechen und angeben, ob der erste oder der zweite Vorgänger eines Gatters auf dem betrachteten Pfad liegt.
$\Rightarrow$ Der SK hat logarithmische Tiefe.
$\Rightarrow$ Der Pfad besitzt logarithmische Länge.

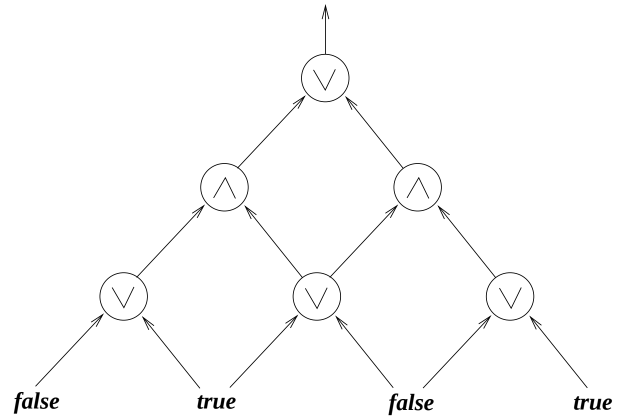
Der äquivalente transformierte SK besitzt die Gestalt eines Binärbaums mit den eben kodierten Pfaden als Gattern.
$\Rightarrow$ Für die Gatter gilt:

- Das $OUTPUT$-Gatter erhält die Bezeichnung ε ($\widehat{=}$ leerer String).
- Die Vorgänger erhalten entsprechend 0 oder 1.

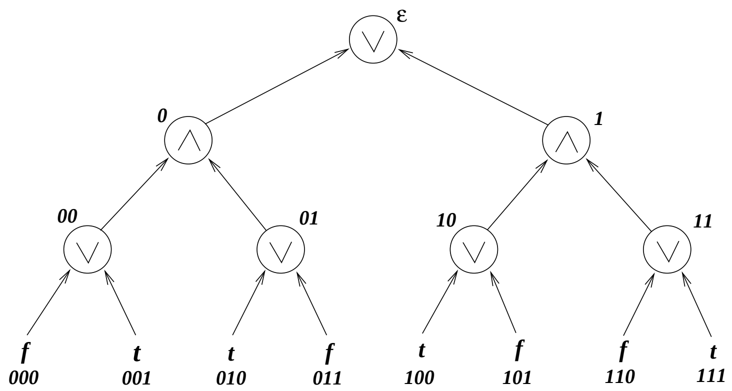
Die Gatter und die Verbindungen des neuen SK werden nacheinander generiert, wobei der Speicherplatz wiederverwendet wird.
$\Rightarrow$ Der SK liegt als Liste von Bit-Strings und Gattertypen vor.

Beispiel:

(a)



(b)



3. Das $OUTPUT$-Gatter des transformierten SK wird berechnet:

   Dabei gilt für die verschiedenen Gatter:

   - Für ein UND-**Gatter** g berechne rekursiv das Ergebnis der beiden Vorgänger g_0 und g_1.
     Wenn dabei g_0 $false$ ergibt, so braucht man g_1 nicht zu berechnen.
     $\Rightarrow$ g erhält den Wert $false$.
   - Für ein $ODER$-**Gatter** braucht man den 2. Vorgänger nur dann auszuwerten, wenn der 1. Vorgänger den Wert $false$ berechnet hat.
   - Für NOT wird der berechnete Wert des Vorgängers invertiert.
   - $true/false$ geben ihren Wert an ihre Nachfolger weiter.

   Wenn der Wert eines Gatters berechnet ist, wird mit der Berechnung des Wertes seines **einzigen** Nachfolgers fortgefahren.
   Der Name des Nachfolgers kann dabei ermittelt werden, indem einfach das letzte Zeichen des aktuellen Bit-Strings gestrichen wird ($100 \rightarrow 10$).
   Sobald der Wert des $OUTPUT$-Knotens berechnet ist, kennt man den vom SK berechneten Wert und ist fertig.

   Frage: Wieviel Information muß dabei zwischengespeichert werden?

   Die $false$-$first$-Regel für UND-knoten garantiert, daß wir uns nur den Namen und Typ des aktuellen Gatters merken müssen.

   Wenn wir die Berechnung des Wertes für den zweiten Nachfolger beendet haben, wissen wir bereits aus der Tatsache, daß wir den zweiten berechnet haben, automatisch den Wert, den der erste berechnet haben muß.

$\Rightarrow$ Der 3. Algorithmus berechnet den Wert des SK in logarithmischem Raum.

$\square$

Wir können den letzten Satz über $\log space$ hinaus verallgemeinern:

Korollar:

$$\begin{aligned} PT/WK(f(n), k^{f(n)}) &\subseteq SPACE(f(n)) \\ &\subseteq NSPACE(f(n)) \\ &\subseteq PT/WK(f(n)^2, k^{f(n)^2}) \end{aligned}$$

Nach Savitch gilt:

$$\mathcal{NL} \subseteq SPACE(\log^2 n)$$
$$\mathcal{NL} = co\,\mathcal{NL}$$

Satz 2.39: $REACHABILITY$ ist $\mathcal{NL}$-vollständig.

Beweis:

(1) $REACHABILITY \in \mathcal{NL}$

(2) Wir müssen zeigen, daß jede Sprache $L \in \mathcal{NL}$ auf $REACHABILITY$ reduziert werden kann.

Dazu benutzen wir die Erreichbarkeitsmethode:

Wir nehmen an, daß L von einer $\log space$-beschränkten NTM N entschieden werden kann.

Zu einer gegebenen Eingabe x konstruieren wir den Konfigurationsgraphen von N: $G(N, x)$.
Wir nehmen an, daß $G(N, x)$ nur einen akzeptierenden Knoten hat (jede akzeptierende Konfiguration besitzt eine Kante zu dem akzeptierenden Knoten).

$G(N, x)$ besitzt nur eine Startkonfiguration und es gilt:

$$x \in L \quad \Leftrightarrow \quad \text{Die erzeugte Instanz von } REACHABILITY \text{ liefert die Antwort "}yes\text{".}$$

□

Satz 2.40: 2-SAT ist $\mathcal{NL}$-vollständig.

Beweis:

(1) 2-$SAT \in \mathcal{NL}$: siehe Korollar aus Satz 2.10.

(2) Wir zeigen, daß $UNREACHABILITY \leq$ 2-SAT, da $\mathcal{NL} = co\ \mathcal{NL}$ und $REACHABILITY$ $\mathcal{NL}$-vollständig ist:

Wir starten in einem azyklischen Graphen $G = (V, E)$.

Für alle Kanten $(x, y) \in E$ erzeugen wir die Klauseln $(\neg x \vee y)$, d. h. jedem Knoten im Graphen wird eine Boolesche Variable zugeordnet.
Dann fügen wir die Klauseln (s) und $(\neg t)$ als Start- und Endknoten s und t hinzu.

$\Rightarrow$ Die resultierende Instanz von 2-SAT ist nur dann erfüllbar, wenn es keinen Weg im konstruierten Graphen von s nach t gibt.

□

2.9.2 Alternierende Komplexitätsklassen

Wir betrachten eine alternative Definition für den Nichtdeterminismus:

"Eine Konfiguration führt zur Akzeptierung, wenn sie eine finale akzeptierende Konfiguration ist, oder (rekursiv), wenn mindestens eine ihrer Nachfolgekonfigurationen zur Akzeptierung führt."

Dann gilt:

- So betrachtet stellt eine Konfiguration eine implizite $ODER$-Verknüpfung ihrer Nachfolgekonfigurationen dar.
- Zur Entscheidung über das Komplement der vorliegenden Sprache können wir eine Konfiguration als implizite UND-Verknüpfung ihrer Nachfolgekonfigurationen auffassen.
- Wir lassen beide Modi in unserer nichtdeterministischen TM zu, d. h. wir unterscheiden UND-Konfigurationen, die nur akzeptieren, wenn **alle** Nachfolgekonfigurationen akzeptieren, und $ODER$-Konfigurationen, die akzeptieren, wenn mindestens eine Nachfolgekonfiguration akzeptiert.
- Der Modus der Konfigurationen wird durch den Zustand q festgelegt.

Definition 2.21: Eine alternierende TM (ATM) ist eine präzise NTM $N = (K, \Sigma, \Delta, s)$, deren Zustandsmenge partitioniert wird in $K = K_{AND} \cup K_{OR}$.

Sei x eine Eingabe.
Wir betrachten den Berechnungsbaum von N auf der Eingabe x:

- Jeder Knoten ist eine Konfiguration von N und beinhaltet ebenfalls die Schrittnummer von N.
- Wir definieren rekursiv, "bottom-up", "eventuell akzeptierende Konfigurationen":
  (1) Alle Blattknoten, die im Zustand "yes" enden, sind "eventuell akzeptierend".
  (2) Eine Konfiguration mit einem Zustand aus K_{AND} ist "eventuell akzeptierend".
  $\Leftrightarrow$
  Alle Nachfolgekonfigurationen sind "eventuell akzeptierend".
  (3) Eine Konfiguration mit einem Zustand aus K_{OR} ist "eventuell akzeptierend".
  $\Leftrightarrow$
  Mindestens eine der Nachfolgekonfigurationen ist "eventuell akzeptierend".
  (4) N akzeptiert x, wenn die Startkonfiguration "eventuell akzeptierend" ist.
- Eine ATM N entscheidet die Sprache L, wenn N alle Strings $x \in L$ akzeptiert und $x \notin L$ zurückweist.

Definition 2.22:

$ATIME(f(n))$ (Alternating Time): die Klasse aller Sprachen, die von einer ATM entschieden werden können, deren Berechnungen auf der Eingabe x nach höchstens $\mathcal{O}(f(n))$ Schritten stoppen.

$ASPACE(f(n))$ (Alternating Space): die Klasse aller Sprachen, die von einer ATM mit dem Platz $f(n)$ entscheidbar sind.

Definition 2.23:

$$\mathcal{AP} := ATIME(n^k)$$
$$\mathcal{AL} := ASPACE(\log n)$$

Satz 2.41: *$MONOTONE\ CIRCUIT\ VALUE$ (MCV)* ist $\mathcal{AL}$-vollständig.

Beweis:

(1) **z.z.:** $MCV \in \mathcal{AL}$:

Die Eingabe der ATM ist ein Schaltkreis, gegeben als Liste von Kanten und Gattertypen.

Die ATM untersucht das $OUTPUT$-Gatter des SK:

Dabei geht die ATM in einen UND-Zustand, falls der $OUTPUT$ ein AND-Gatter ist, und andererseits in einen $ODER$-Zustand, falls der $OUTPUT$ ein OR-Gatter ist.
In beiden Fällen werden dann die Vorgängergatter nichtdeterministisch evaluiert, während der $OUTPUT$ gemerkt wird.
Dieser Prozeß wird solange wiederholt, bis das $INPUT$-Gatter erreicht wird.

Die ATM akzeptiert bzw. verwirft, je nach dem Typ des $INPUT$-Gatters.

Aus der Induktion über die Tiefe eines Gatters, unter Ausnutzung unserer Definition des "eventuellen Akzeptierens" für ATM's, können wir schließen, daß eine Gatterkonfiguration genau dann eine "eventuell akzeptierende" Konfiguration ist, wenn das korrespondierende Gatter den Wert *true* berechnet.

Es gilt:

Die Startkonfiguration ist akzeptierend.
$\Leftrightarrow$ Das $OUTPUT$-Gatter berechnet den Wert *true*.

$\Rightarrow$ Die Berechnung erfolgt in $\log space$.

(2) **z.z.:** $\forall L \in \mathcal{AL}$: L ist reduzierbar auf MCV.

Gegeben seien $L \in \mathcal{AL}$ und die zu L korrespondierende ATM $N = ((K_{AND} \cup K_{OR}), \Sigma, \Delta, s)$.

Wir konstruieren einen monotonen SK C, so daß gilt:

$$C \text{ berechnet } true. \quad \Leftrightarrow \quad N \text{ akzeptiert die Eingabe } x.$$

Bemerkung: Jede Transition von N umfasse 2 Wahlmöglichkeiten.

Dabei gilt:

Gatter (C, i)**:** C ist eine Konfiguration von N auf der Eingabe x, i ist ein Schrittzähler mit $0 \leq i \leq |x|^k$.

Kante $((C_1, i), (C_2, j))$**:** Aus der Konfiguration C_2 kann in genau einem Schritt nach C_1 ubergegangen werden, $i = j + 1$.

Für die **Gattertypen** gilt:

Typ	Zustand
AND	$q \in K_{AND}$
OR	$q \in K_{OR}$
$true$	$q = $ "yes"
$false$	$q = $ "no"

Das $OUTPUT$-Gatter entspricht der Startkonfiguration auf der Eingabe x.

Über die Korrespondenz SK $\leftrightarrow$ ATM aus dem vorherigem Beweis erhalten wir:

$$\text{Der SK berechnet } true. \quad \Leftrightarrow \quad x \in L.$$

$\square$

Korollar:

$$\mathcal{AL} = \mathcal{P}$$

Verallgemeinerung:

$$ASPACE(f(n)) = TIME(k^{f(n)})$$

2.9.3 $REACHABILITY$ für ungerichtete Graphen

Definition 2.24: $\mathcal{RL}$ ist die Klasse aller Sprachen L, die durch eine präzise log *space*-beschränkte TM folgendermaßen entschieden werden kann:

(1) Alle Berechnungen enden nach identischer (polynomialer) Schrittzahl.

(2) Aus jeder Konfiguration gibt es 2 nichtdeterministische Wahlmöglichkeiten.

(3) $x \in L \quad \Rightarrow$ Mindestens die Hälfte aller Berechnungen enden im Zustand "yes".
$x \notin L \quad \Rightarrow$ Alle Berechnungen enden im Zustand "no".

Bemerkung:
Diese Maschine ist eine $\mathcal{RP}$-Maschine mit logarithmischer Raumbeschränkung.

Satz 2.42: *UNDIRECTED REACHABILITY (UR)* $\in \mathcal{RL}$.

Beweis: Sei $G = (V, E)$ ein ungerichteter Graph mit $V = \{1, \ldots, n\}$.
Wir definieren einen Random-Walk-Algorithmus, der die Frage "Gibt es einen Pfad von 1 nach n?" entscheidet:

- Starte im Knoten 1.
- Wähle zufällig eine Kante $(1, i) \in E$.
- Wiederhole diesen Schritt im Knoten i.

Dabei müssen wir aus technischen Gründen die Möglichkeit berücksichtigen, am Knoten i zu verweilen $\rightarrow (i, i) \in E,\ 1 \leq i \leq n$.
Wir definieren:

- v_t ist der Knoten v, der vom Random Walk in der Zeit t besucht wird.
- $v_0 = 1$
- $P_t(i)$ ist die Wahrscheinlichkeit, daß der Knoten i zum Zeitpunkt t besucht wird.
- $P_{t+1}(i) = \frac{1}{d_j}$, wenn $v_t = j$ mit $(j, i) \in E$, wobei $d_j = \#(\textit{aus } j \textit{ ausgehende Kanten})$.

Lemma 2.3: Sei $G = (V, E)$ zusammenhängend. Dann gilt für $1 \leq i \leq n$:

$$\lim_{t \to \infty} P_t(i) = \frac{d_i}{2|E|}$$

Beweis: $P_t(i)$ wird zum Zeitpunkt t vom asymptotischen Wert $\frac{d_i}{2|E|}$ abweichen. Sei:

$$\delta_t(i) = P_t(i) - \frac{d_i}{2|E|} \quad (*)$$

$$\Delta_t = \sum_{i \in V} |\delta_t(i)|$$

Wir berechnen $P_{t+1}(i)$ aus $P_t(i)$:

- Die Wahrscheinlichkeit, daß der Random Walk irgendeinen Nachbarn von i wählt, ist gleich.
- Jeder Knoten i teilt $P_t(i)$ in d_i gleiche Teile und reicht diese an die Nachbarn weiter.
- Jeder Knoten i summiert die an ihn weitergegebenen Teile auf.
  $\Rightarrow$ Wir erhalten $P_{t+1}(i)$.

Aus $(*)$ folgt:

$$P_t(i) \;=\; \frac{d_i}{2|E|} \;+\; \delta_t(i)$$

Da der Tausch der $\delta_t(i)$ nur zwischen benachbarten Knoten abläuft, kann Δ_t nicht wachsen. Es folgt:

$$\Delta_{t+1} \leq \Delta_t$$

Δ_t kann fallen, wenn $2\delta_t(i)$ mit umgekehrten Vorzeichen an einen gemeinsamen Knoten weitergegeben werden.

Behauptung: Für t und Δ_t gilt:

$$\begin{array}{rrrl} \exists \text{ Knoten } i^+ & \text{mit} & \delta_t(i^+) \;\geq & \dfrac{\Delta}{2|V|} \\ i^- & \text{mit} & \delta_t(i^-) \;\leq & -\dfrac{\Delta}{2|V|} \end{array}$$

Es existiert ein Pfad $(i_0 = i^+, i_1, i_2, \ldots, i_m, \ldots, i_{2m} = i^-)$.
Die positive Abweichung $\delta_t(i^+)$ wird bis i_m weitergereicht und dabei durch den Grad der Zwischenknoten dividiert. Analog gilt dies für $\delta_t(i^-)$.
Mindestens eine positive Abweichung $\geq \frac{1}{|V|^m}$ kommt in i_m an. Ebenso gilt dies für mindestens eine negative Abweichung $\leq -\frac{1}{|V|^m}$.
Nach $m < n$ Schritten wird die positive Abweichung $\delta_t(i^+) \geq \frac{\Delta}{2|V|^n}$ durch die betragsgleiche negative Abweichung eliminiert.

$\Rightarrow\;\; \Delta_{t+n} \;\leq\; \Delta_t(1 - \frac{1}{|V|^n})$

$\Rightarrow\; \lim\limits_{t\to\infty} \Delta_t = 0$

$\Rightarrow P_t(i)$ konvergiert gegen $\frac{d_i}{2|E|}$.

$\square$

Das Lemma sagt andererseits, daß der Random Walk nach $\frac{2|E|}{d_i}$ Schritten voraussichtlich zu i zurückkehrt.
Nach $\frac{d_i}{2}$ Runden gelangen wir zum nächsten Knoten auf unserem Pfad ($|E|$-Schritte).
$\Rightarrow$ Wir erreichen n nach voraussichtlich $\leq n \cdot |E|$ Schritten ($\Rightarrow \mathcal{O}(n^3)$).

Algorithmus:

(1) Starte im Knoten 1 und laufe genau $2 \cdot n \cdot |E|$ Schritte.

(2) Wird n besucht, dann existiert ein Pfad $(1, \ldots, n)$.

(3) Wird n nicht besucht, dann existiert wahrscheinlich kein Pfad $(1, \ldots, n)$.

Die Wahrscheinlichkeit für eine Negativantwort ist $\leq \frac{1}{2}$.

Ressourcen: Jeder Berechnungsschritt kann in log *space* durchgeführt werden.

$\square$

Korollar:

$$\mathcal{L} \subseteq \mathcal{RL} \subseteq \mathcal{NL}$$

2.10 Zählklassen oder die Bedeutung des Zählens

2.10.1 Die Permanente

Wir hatten bisher Probleme betrachtet, die

- nach der Existenz einer Lösung fragen oder
- die Lösung produzieren.

Jetzt betrachten wir Probleme, denen die Frage nach der Zahl der existierenden Lösungen zugrundeliegt.

Beispiel 2.10:

$\#SAT$ ($COUNTING\ SAT$)

geg.: eine Menge von Klauseln.
Frage: Wie hoch ist die Zahl der verschiedenen erfüllenden Belegungen?

Offenbar bleibt die Lösung von $\#SAT$ eine Lösung von SAT:

$$SAT \text{ ist lösbar.} \quad \Leftrightarrow \quad \#SAT \text{ ermittelt eine Zahl } > 0.$$

Analoges gilt für $\#HAMILTON\ PATH$ und $\#CLIQUE$.

Achtung: Die Komplexitäten der Lösungen von Entscheidungsproblemen und den zugehörigen Zählproblemen können sehr verschieden sein!

Beispiel 2.11: Wir betrachten $MATCHING$ und $\#MATCHING$:

Sei $G = (U, V, E)$ ein bipartiter Graph mit $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$ und $E \subseteq U \times V$.

Wir betrachten die Adjazenzmatrix $A^G = (a_{ij})$ mit

$$a_{i,j} = \begin{cases} x_{ij} & falls\ (u_i, v_j) \in E \\ 0 & sonst \end{cases}$$

Dann ist

$$det\ A^G \ = \ \sum_{\pi\ Perm.} \sigma(\pi) \prod_{i=1}^{n} a_{i,\pi(i)},$$

wobei $\sigma(\pi) \in \{-1, +1\}$ ist.

Achtung: $det\ A^G$ liefert **nicht** die Anzahl der Perfect Matchings!

Zur Bestimmung der Anzahl muß nicht die Determinante, sondern die **Permanente** berechnet werden:

$$perm\ A^G \ = \ \sum_{\pi\ Perm.} \prod_{i=1}^{n} a_{i,\pi(i)}$$

Die Permanente liefert die Zahl der Perfect Matchings.

$\Rightarrow$ $\#MATCHING$, besser bekannt als $PERMANENT$, ist ein sehr schwieriges Problem.

Definition 2.25: Sei Q eine polynomial balancierte, polynomialzeit-berechenbare binäre Relation. Das zu Q gehörende **Counting Problem / Zählproblem** ist wie folgt definiert:

geg.: x.
Frage: Wie hoch ist die Anzahl der y mit $(x, y) \in Q$ (Ausgabe als Binärzahl)?

Definition 2.26: $\#\mathcal{P}$ ("Counting $\mathcal{P}$", "Sharp $\mathcal{P}$") ist die Klasse aller Zählprobleme, die zu polynomial balancierten, **polynomialzeit-berechenbaren** Relationen gehören.

Beispiel 2.12: $\#HAMILTON\ PATH$, $\#SAT$, $PERMANENT$.

Wir benötigen einen Reduktionsbegriff, um Konzepte wie Vollständigkeit auch im Bereich der Zählklassen definieren zu können:

Definition 2.27: Seien A, B Zählprobleme mit $A \subseteq \Sigma^*$ und $B \subseteq \Pi^*$.
$R : \Sigma^* \to \Pi^*$ ist eine Reduktion (wobei R $\log space$-berechenbar)

$\underset{def.}{=}$ (1) $x \in A \quad \Leftrightarrow \quad R(x) \in B$, und
(2) x hat die gleiche Zahl von Lösungen wie $R(x)$.

Beispiel 2.13: Eine Reduktion von $CIRCUIT\ SAT$ auf 3-SAT ist die Reduktion der zugehörigen Zählprobleme $\#CIRCUIT\ SAT$ auf $\#3$-SAT:

Der Schaltkreis C hat die gleiche Zahl von erfüllenden Belegungen wie die konstruierte 3-SAT-Formel.

Satz 2.43: $\#SAT$ ist $\#\mathcal{P}$-vollständig.

Beweis: Sei ein beliebiges Zählproblem aus $\#\mathcal{P}$ gegeben, beschrieben durch eine polynomial balancierte, polynomialzeit-berechenbare binäre Relation Q.

Wir reduzieren Q auf $\#SAT$:

Da Q polynomial balanciert ist, gilt:
Für jedes x habe die Lösung y eine Länge $\leq |x|^k$ ($(x, y) \in Q$).
O. B. d. A. gilt: Die Lösung hat die Länge $= |x|^k$ und $y \in \{0, 1\}^*$.
Sei M eine polynomial-zeitbeschränkte TM, die Q entscheidet.
Wir können zu M und x einen Schaltkreis $C(M; x)$ mit Eingaben der Länge $|x|^k$ konstruieren, so daß gilt:

$$y \text{ erfüllt } C(M; x) \quad \Leftrightarrow \quad M \text{ akzeptiert die Eingabe } x; y \textbf{ bzw. } (x, y) \in Q$$

Die Konstruktion von $C(M; x)$ liefert eine Reduktion von Q auf $\#CIRCUIT\ SAT$.

Wegen $\#CIRCUIT\ SAT \leq \#SAT$ folgt die Behauptung.

□

Bemerkung:
Viele unserer Reduktionen liefern Reduktionen der zugehörigen Zählprobleme.
Ausnahme: Die Reduktion von 3-SAT auf $HAMILTON\ PATH$ (jeder erfüllenden Belegung werden viele Hamilton-Wege zugeordnet).

Satz 2.44: $\#HAMILTON\ PATH$ ist $\#\mathcal{P}$-vollständig.

Von besonderem Interesse ist:

Satz 2.45: $PERMANENT$ ist $\#\mathcal{P}$-vollständig.

Achtung:
Das zugehörige Entscheidungsproblem kann in polynomialer Zeit gelöst werden.

Offenbar gilt:

Satz 2.46:

$$\#\mathcal{P} \subseteq \mathcal{PSPACE}$$

Beweis: Wir können alle Lösungen in lexikographischer Ordnung aufschreiben und abzählen.

□

Frage: Wie sieht das Verhältnis von $\#\mathcal{P}$ auf einer Polynomialzeit-Maschine (PM) aus?

Beobachtung: $\#\mathcal{P}$ hat "Ähnlichkeit" mit $\mathcal{PP}$ (mehr als die Hälfte der Berechnungen sind akzeptierend).

Satz 2.47 (Toda):

$$\bigcup_{i \geq 0} \Sigma_i \mathcal{P} = \mathcal{PH} \subseteq \mathcal{P}^{\mathcal{PP}}$$

2.11 Die Klasse $\oplus\mathcal{P}$ (Parity $\mathcal{P}$)

$\mathcal{PP}$ - Das erste Bit der Zahl der erfüllenden Belegungen ist wichtig (Mehrzahl).

$\oplus\mathcal{P}$ - Eine analoge Klasse, bei der das letzte Bit der Zahl der erfüllenden Belegungen wichtig ist.

$\oplus SAT$

geg.: eine Menge von Klauseln.
Frage: Ist die Zahl der erfüllenden Belegungen **ungerade**?

Definition 2.28:
$L \in \oplus\mathcal{P}$ ("Parity $\mathcal{P}$")
$\underset{def.}{=}$ Es existiert eine polynomial-zeitbeschränkte NTM M mit

$$x \in L \;\Leftrightarrow\; \text{Die Zahl der akzeptierenden Berechnungen von } M \text{ ist ungerade.}$$

Analog:
$L \in \oplus\mathcal{P}$
$\underset{def.}{=}$ Es existiert eine polynomial balancierte und polynomialzeit-berechenbare Relation R mit

$$x \in L \;\Leftrightarrow\; \text{Die Zahl der } y \text{ mit } (x, y) \in R \text{ ist ungerade.}$$

Satz 2.48: $\oplus SAT$ und $\oplus HAMILTON\ PATH$ sind $\oplus\mathcal{P}$-vollständig.

Beweis:

(1) $\oplus SAT$ und $\oplus HAMILTON\ PATH \in \oplus\mathcal{P}$: klar.

(2) Die $\oplus\mathcal{P}$-Vollständigkeit ergibt sich mit Hilfe der Reduktionen, die schon im Falle von $\#SAT$ und $\#HAMILTON\ PATH$ angewandt wurden.

$\square$

Satz 2.49: $\oplus\mathcal{P}$ ist abgeschlossen bzgl. der Komplementbildung.

Beweis: $\overline{\oplus SAT}$ ("Ist die Anzahl der erfüllenden Belegungen **gerade** ?") ist $co \oplus \mathcal{P}$-vollständig.

Wir zeigen, daß $\overline{\oplus SAT} \leq \oplus SAT$ ($\Rightarrow co \oplus \mathcal{P} \subseteq \oplus\mathcal{P}$):

Gegeben sei eine Klauselmenge F mit n Variablen $x_1, \ldots, x_n$.
Wir nehmen eine neue Variable z hinzu und fügen z jeder Klausel hinzu.
Weiter nehmen wir n Klauseln $(z \Rightarrow x_i)$, $i = 1, \ldots, n$, hinzu.
Dann erhalten wir die Klauselmenge F'.

Es gilt:

- Jede erfüllende Belegung von F ist für $z = 0$ eine erfüllende Belegung für F'.
- Die Belegung $x_i = z = 1$ $(i = 1, \ldots, n)$ ist ebenfalls erfüllend.
- Ansonsten gibt es keine weiteren erfüllenden Belegungen.

$\Rightarrow$ Wir haben für F' genau ($alte\ Zahl + 1$) erfüllende Belegungen, wobei mit "*alte Zahl*" die Anzahl der erfüllenden Belegungen von F gemeint ist.
$\Rightarrow co \oplus \mathcal{P} \subseteq \oplus\mathcal{P}$.

Analog folgt: $\oplus\mathcal{P} \subseteq co \oplus \mathcal{P}$.

$\square$

Satz 2.50:

$$\mathcal{NP} \subseteq \mathcal{RP}^{\oplus\mathcal{P}}$$