Turingmaschine:

- Q: endliche Zustandsmenge $Q = \{q_0, ..., q_k\}$
- $\sum$: endliche Eingabealphabet
- $\Gamma \supseteq \sum$: endliche Bandalphabet
- $B \in \Gamma / \sum$: das Leerzeichen
- $q_0 \in Q$: Anfangs- (oder Start-) zustand
- $F \subseteq Q$: Menge akzeptierender Endzustände
- δ : Zustandsüberführungsfunktion
  $(\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\})$
  (*in einem Zustand* **X** *Bandzeichen lesen* à *neuer Zustand* **X** *Bandzeichen Schreiben* **X** {*Bewegung Lesekopf*))

Definition:
Eine **Konfiguration** einer Turingmaschine ist der Aktuelle Speicherinhalt und die Aktuelle Position $\Gamma^* \times Q \times \Gamma^*$ (aqb: $a, b \in \Gamma^*; q \in Q$)
Sie bedeutet, dass auf dem Band die Inschrift *ab* eingerahmt von lauter Leerzeichen steht, die Maschine im Zustand *q* ist und der Kopf auf die Zelle zeigt, die den ersten Buchstaben von *b* enthält

---

a'q'b' ist **direkte Nachfolgekonfiguration** von *aqb*, wenn *a'q'b'* in einem Rechenschritt aus *aqb* entsteht
Notation: $aqb \vdash a'q'b'$

a''q''b'' ist Nachfolgekonfiguration von *aqb*, wenn *a''q''b''* in endlich vielen Rechenschritten aus *aqb* entsteht, Notation $aqb \overset{*}{\vdash} a''q''b''$. Es gilt stets $aqb \overset{*}{\vdash} aqb$, da „keine Rechenschritte" in der Formulierung „endlich viele Rechenschritte" enthalten sind.

Die **Sprache** L_M von M ist die Menge **aller** von M akzeptierten $x \in \sum^*$, Es kann/darf Berechnungen für Eingaben $x \in \sum^*$ geben, die nicht abbrechen, falls $x \notin L_M$.

Stärkerer Begriff: Falls M die Sprache L_M akzeptiert und für alle $x \in \sum^*$ nach endlich vielen Schritten hält, so **entscheidet** M die Sprache L_M (**rekursiv**)

Definition:
Eine Sprache $L \subseteq \sum^*$ **heißt rekursiv** (***entscheidbar***) wenn es einen Turingautomaten gibt, der auf alle Eingaben stoppt und die Eingabe *w* genau dann akzeptiert wenn $w \in L$ ist.

Eine Sprache $L \subseteq \sum^*$ heißt **rekursiv aufzählbar** (*semientscheidbar*) wenn es einen det. 1-Band Turingautomaten gibt der die Eingabe w akzeptiert, die aus L ist.

Eine Funktion $f : \sum^* \to \sum^*$ heißt **total rekursiv** (*berechenbar*) wenn es eine Turingmaschine gibt, die aus Eingabe x den Funktionswert f(x)berechnet.

Eine Funktion $f : N^k \to N$ (beachte: N kein endliches Alphabet) ist **total rekursiv** wenn eine Eingabe von Typ $\text{bin}(i_1)\#\text{bin}(i_2)\#...\#\text{bin}(i_k)\#$ mit Ergebnis bin(m) stoppt wenn $m = f(i_1, ..., i_k)$. Dabei ist bin(j) Binärstelle von j.

Satz:
Eine **k-Band-Turingmaschine** M, die mit Rechenzeit *t(n)* und Speicherplatz *s(n)* auskommt, kann von einer Turingmaschine M' mit Zeitbedarf $O(t^2(n))$ und Speicherplatz $O(s(n))$ simuliert werden.

Abzählbar: Sprache kann durchnummeriert werden

3. Unterprogramme

Beim „Programmieren“ von TM können wir sagen: Wir benutzen ein Unterprogramm (ohne andere TM), um eine bestimmte Aufgabe zu lösen

1.3 Simulation zwischen RAM und TMs

Def. 1.9: Für RAM M und Eingabe x ist:

- $T_M(x) = \#$ Schritte von M gestartet mit x
- $S_M(x) = \#$ größte benutzte Speicheradresse bei der Rechnung von M gestartet mit x

„t(n)-Zeit“ und „s(n)-platzbeschränkt“ ist analog zu den TMs definiert

Satz 1.10: Jede RAM kann durch eine det. 1-Band-TM simuliert werden. Ist die RAM t(n)-zeitbeschränkt, ist die TM $O(t(n)^3)$-zeitbeschränkt

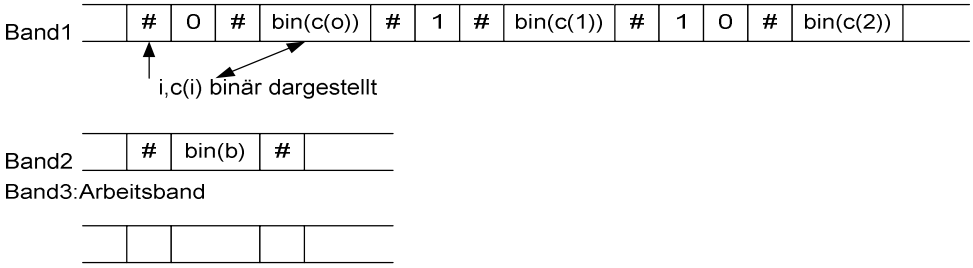
Beweis: (a) Wir müssen uns überlegen, wie eine Konfiguration der RAM auf einer Mehrband-TM gespeichert wird.
(b) Realisierung von Kofigurationsübergängen

Zu (a): $c(1),\ldots,c(k)$ enthält die Eingabe
$\in N \qquad \in N$

Konfiguration der RAM: Speicherinhalt: $c(0), c(1),\ldots$
Programmzähler b

(zu fester RAM wird eine feste, „äquivalente“ TM angegeben)

TM:



Wir müssen nun für jeden möglichen RAM-Befehl ein TM-Unterprogramm schreiben, das diesen Befehl realisiert.

z.B.: ADD i: c(0):= c(0) + c(i) b:= b+1
- suche die Speicheradresse Bin(i) auf Band1
- Kopiere bin(c(i)) auf das Arbeitsband
- Falls c(i) noch nicht auf dem Band 1: Einfügen und mit 0 füllen
- „Addiere binär bin(c(o)) und bin (c(i))“
- b um 1 erhöhen

Satz 1.11: Jede t(n)-zeitbeschränkte det. 1-Band-TM kann durch eine RAM simuliert werden
Beweis: „Programmieraufgabe“ (einfach programmieren)

Bekannt bis jetzt:
RAM, det 1Band-TM, det. k-Band-TM, Halbband-TM (gleiche wie det. 1Band-TM), 2D-TM (gleiche wie det. 1 Band-TM), „Vektor-Additionssysteme“

1.4 Die Churche-Turing-These

Alles was man machen kann ist vom Typ 0 (ist das was ich mit der TM machen kann)

Die im intuitiven Sinn berechenbaren Funktionen sind genau die, die durch Turingmaschinen (det, 1-Band-) berechenbar sind.
Vermutung: es gibt nichts über den TM

1.5 Universelle Turingmaschinen

Eine Maschine Bauen die alle anderen Maschinen beherrscht

Bislang: Unsere TM können genau eine Aufgabe lösen

$\Gamma = \{0,1,B\}$

$Q = \{q_1,...,qn\}$

Startzustand q_1

$F = \{q_n\}$

Darstellung von det 1-Band-TM m „in maschinenlesbarer“ Form OBdA

M soll über {0,1,#} codiert(gödelisiert = nummeriert) werden

$\delta(q_i, x) = (q_i, y, D)$ $x, y \in \Gamma$ wird codiert durch

0	00
1	01
B	11

$D \in \{R, L, N\}$ wird codiert durch

R	00
N	01
L	11

$q_i \rightarrow 1\underbrace{11...1}_{i}$

Trennsymbol :#

Bsp.: $\delta(q_3,1) = (q_2, B, L)$ = $1\underbrace{1}_{q_3}\#\underbrace{0}_{1}\#\underbrace{1}_{q_2}\#\underbrace{1}_{B}\#\underbrace{1}_{L}$

Die gesamte $\delta - Tabelle$ kann wie folgt codiert werden:

$B\#\#\ \underbrace{1...1}_{n}\ \#\#\text{code(erster Eintrag der } \delta\text{– Tabelle)}\#\#\text{code(2.Eintrag)}\#\#...\#\#\text{code(letzerEintrag)}\#\#\#B$

Endzustand

à „**Gödelnummer**“ von M: <M>

à Man kann jede TM eindeutig auf eine natürliche Zahl abbilden (TM wird eine Zahl zugeordnet)

Eine TM $\tilde{M}$ heißt **universell**, wenn sie bei Eingabe $< M >, x \in \{0,1\}^*$, so verhält wie M gestartet mit x.

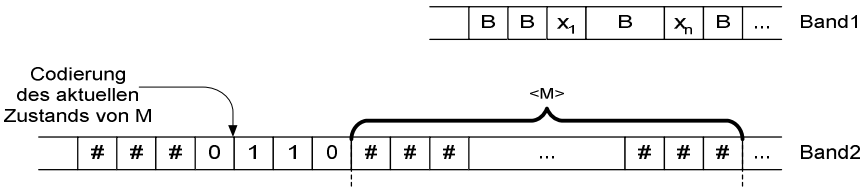
Satzt 2.12: Es gibt eine universelle det. 2-Band-TM $\tilde{M}$ die jede t(n)-zeit- und s(n)-platzbeschränkte 1-Band-TM M auf Platz O(sn) in Zeit O(t(n)) simuliert, falls M hält.

Beweis: M habe Bandalphabet $\Gamma = \{0,1,B\}$

$\tilde{M}$: Auf Band 1 steht <M>x

Kopiere <M> auf Band2, lösche M von Band 1

Jetzt:



- Finde zum aktuellen Zustand q_i und Zeichen x auf Band 1 die Stelle $\#1^i\#code(x)\#1^i\#code(y)\#code(D)$ $[\delta(q_i, x) = (q_i, Y, D)]$

- Aktualisiere den aktuellen Zustand zu 1^i, ersetze auf Band 1 das x durch y und gehe auf Band 1 in Richtung D

Da $|< M >| = O(1)$, kostet die Simulation eines Schrittes nur O(1) Zeit. Der Zeit- und Platzbedarf der Gesamten Simulation ist also O(t(n)) bzw. O(s(n)).

- Platz: Band 1 Benötigt genau so viele Zellen wie M gestartet mit x, d.h. $s(|x|)$
  Band 2 benötigt O(1) viele Zellen à Gesamtplatzt O(s(n)).
- Zeit: (# Schritte von M gest. mit x) * (Zeitaufwand zur Verwaltung auf Band 2)
  t(n) * O(1) = O(t(n))

Beobachtung: Sei $x \in \{0,1,\#\}^*$. Es ist entscheidbar, ob X = <M> für eine det 1-Band-TM M ist. („Syntaxanalyse“)

1.6 Unentscheidbare Probleme

Es gibt viel mehr Sprachen als TMs (d.h. Programme)

$L \subseteq \sum^*$. L heißt ***Sprach oder man sagt Problem*** und meint das Wortproblem, d.h. die Frage: $x \in L$?

PROBLEME SIND ALSO MENGEN

Wie viele $L \subseteq \{0,1\}^*$ gibt es? $\quad |\{0,1\}^*| = |N|$

123

abzählbar unendlich

der Sprachen ist $|P(N)| = |R|$ überabzählbar unendlich

Es gibt so viele Sprachen wie es Reelle Zahlen gibt

der TM ist <m(wegen der Gödelnummern) abzählbar unendlich

Gibt es wichtige Sprachen, für die es keine Programme gibt?

Def.1.13 Die Sprache

$H := \{< M > w \mid M \text{ ist eine det.1-Band-TM, die, gestartet mit w, hält}\}$ ist das ***(allgemeine) Halteproblem***

Satz 1.14 **H ist unentscheidbar**.

Beweis: Indirekt, selbstanwendung, Diagonalisierung

Annahme: es gibt eine TM, M_H, die H entscheidet. M_H hält auf jede Eingabe x, und man kann dann am Zustand (von M_H) erkennen, ob $x \in H$ oder $x \notin H$ ist

Wir schreiben das Programm (die TM) M_{Schlau}

TM M_{Schlau}

1 – die Eingabe sei y

2 – falls y = <M>w [Syntaxanalyse], dann

3 – entscheide mittels M_H ob $< M >< M > \in H$

4 – falls ja, schreibe nach rechts endlos viele 1er aufs Band

5 – falls nein, Stopp

$< M_{schlau} >< M_{schlau} > \in H$?? („Was passiert, wenn man M_{Schlau} mit < M_{Schlau}> startet?“)

i - $< M_{schlau} >< M_{schlau} > \in H$ à die Antwort in (3) ist „Ja“ à Endlosschleife in (4)

à M_{Schlau} gestartet mit <M_{Schlau}> hält nicht à $< M_{schlau} >< M_{schlau} > \notin H$

ii - $< M_{schlau} >< M_{schlau} > \notin H$ à die Antwort in (3) ist „nein“ à Stopp in (5) wird erreicht

à M_{Schlau} gestartet mit <M_{Schlau}> hält à $< M_{schlau} >< M_{schlau} > \in H$

à M_H kann es nicht geben!!

	<M1>	<M2>	<M3>	….
M1	+	-	+	+
M2	-	+	+	+
M3	-	+	-	+
M4	-	-	-	+
….	+	-	-	+

\+ M_j gestartet mit <M_j> hält

\- M_j gestartet mit <M_j> hält nicht

$< M_j >< M_j > \in H$

Auf der Diagonale wird + zu – und umgekehrt

à Unentscheidbarkeit von TM

Satz 1.15:

a – H ist rekursiv aufzählbar

b – $\overline{H}$ ist nicht rekursiv aufzählbar $\overline{H} \in \sum^* / H$

Beweis:

a – Universelle TM zählt H auf!

b – Wäre $\overline{H}$ rekursiv aufzählbar mittels $TM\ \tilde{M}$, dann könnten wir folgende TM programmieren

Entscheider für H

\- die Eingabe sei <M>w

\- führe gleichzeitig aus und stoppe, wenn einer stoppt:

* starte mit <M>w auf Band 1 die universelle TM, die die Wörter aus H akzeptiert

* starte mit <M>w auf Band 2 die $TM\ \tilde{M}$, die die Wörter aus $\overline{H}$ akzeptiert

Entscheider für H hält für jede Eingabe <M>w und man kann erkennen ob <M>w$\in H$ oder <M>w$\notin H$ gilt

à H ist entscheidbar (WIEDERSPRUCH)

Def. 1.15:

$H_\varepsilon = \{< M > | \text{ M, gestartet mit } \varepsilon \text{ (leere Wort, "leeres and") hält}\}$

Heißt **Initiale Halteproblem** (oder auch „Spezielles Halteproblem“) (andere Abk. H_0)

Satz 1.17: H_ε ist nicht entscheidbar.

Beweis: Wir tun so, als ob H_ε entscheidbar wäre, und schreiben damit einen Entscheider für H

Annahme: Wir können entscheiden, ob ob <M>w$\in H_\varepsilon$ ist.

Die Frage: „ist <M>w$\in H$ “ soll mit Hilfe des Entscheiders für H_ε bestimmt werden

Wir programmieren zu <M>w das Programm:

feste_Maschine$_{<M>w}$
1 – Eingabe sei x
2 – starte M mit w
3 – falls x = ε , dann Stopp!

$\tilde{M}$ entscheide H_ε

a – wenn $\tilde{M}$ bei Eingabe <feste_Maschine$_{<M>w}$> akzeptierend hält („ja sagt“), feste_Maschine$_{<M>w}$ bis Zeile 3 kommen, was nur möglich ist, wenn <M> gestartet mit w hält, also <M>w$\in H$

b – wenn $\tilde{M}$ „nein sagt“, kommen wir nie zur Zeile 3, d.h. M gestartet mit w hält nicht <M>w$\notin H$

à wir können H entscheiden! (WIEDERSPRUCH)

1.7 Reduktion und der **Satz von Rice**

Def. 1.18 Eine Funktion $f: \{0,1\}^* \to \{0,1\}^*$ heißt **berechenbar**, wenn es eine det. 1-Band-TM M_f gibt, für die mit $x \in \{0,1\}^*$ gilt: - ist f(x) definiert so hält M_f gestartet mit x und f(x) steht auf dem Band
- ist f(x) nicht definiert, so hält M_f gestartet mit x nicht

Ist f(x) für alle x definiert, dann ist f **total berechenbar** (anderer Begriff: **total rekursiv**) $L \in \{0,1\}^*$

Eine Sprache L, ist **entscheidbar** genau dann wenn die charakteristische Funktion

$\aleph_L : \{0,1\}^* \to \{0,1\}$ mit

$$\aleph_L(x): \begin{cases} 1 \text{ falls } x \in L \\ 0 \text{ sonst} \end{cases}$$ total berechenbar ist.

Eine Sprach L ist **rekursiv aufzählbar** genau dann wenn

$$\aleph'_L(x): \begin{cases} 1 \text{ falls } x \in L \\ \text{undef sonst} \end{cases}$$ berechenbar ist.

Def. 1.19: $L_1, L_2 \subseteq \{0,1\}^*$. Eine **Reduktion** ist eine **total berechenbare** Funktion $f: \{0,1\}^* \to \{0,1\}^*$,

für die gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$

Funktion muss alle Möglichkeiten behandeln d.h. auf jeden Fall eine Ausgabe erzeugen.

Wir schreiben: $L_1 \le L_2$ und sagen „L_1 wird auf L_2 reduziert“ (mittels f)

Eselsbrücke: A ≤ B “A ist leichter als B“
Bei nichtentscheidbarkeit: Hier das, was man kennt(A), und hier das was man ist (B).

Nochmals: $H \le H_\varepsilon$

$$f(x): \begin{cases} < \text{feste\_Maschine}_{<M>w} > & \text{falss } x =< M > w \\ x & \text{sonst} \end{cases}$$

$$x \in H \to x =< M > w \in H \Rightarrow < \text{feste\_Maschine}_{<M>w} > \in H_\varepsilon$$

$$x \notin H = \begin{cases} x =< M > w \Rightarrow f(x) =< \text{fest\_Maschine}_{<M>w} > \notin H_\varepsilon \\ x \notin < M > w \Rightarrow f(x) = x \notin H_\varepsilon \end{cases}$$

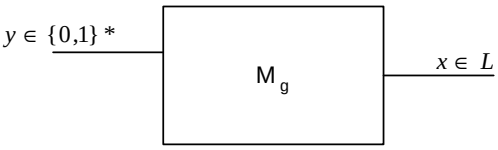
1.8 **Rekursive Aufzählbarkeit**

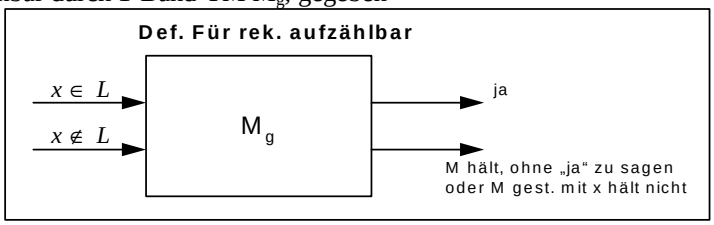
„Aufzählbar“ ist etwas anderes als „abzählbar“

Satz 1.23: Sei L eine unendliche Sprache. Dann gilt:

L ist rekursiv aufzählbar $\Leftrightarrow$ es gibt eine total berechenbare, surjektive Funktion $g \in \{0,1\}^* \to L$

Beweis: „$\Leftarrow$“ $g \in \{0,1\}^*$ à L surjektiv, total berechenbar durch 1-Band-TM M_g, gegeben





Ziel: Wir suchen $y \in \{0,1\}^*$ mit g(y) = x

Wir erzeugen systematisch die Wörter aus {0,1}*, z.B.

$\varepsilon,0,1,00,01,10,11,000,001,...$ also y und starten M_g mit y

Das wird solang gemacht, bis M_g x ausgibt.

M: Eingabe x

$y := \varepsilon$

While , M_g gestartet mit y, nicht x ausgibt do $y := next(y)$

$x \in L \Leftrightarrow$ M estartet mit x hält

à sage „ja“

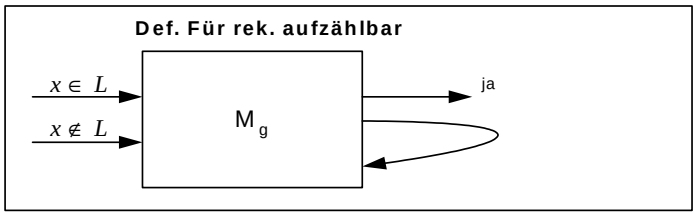
„$\Rightarrow$“ Wir müssen einen TM M_g programmieren, die

(i) für alle Eingaben y, hält

(ii) nur Wörter, aus L ausgibt

(iii) jedes Wort aus L ausgeben kann

Wenn wir M_g haben, definiert diese die Funktion g



$x \in L$: es gibt ein $t \in N$, so dass M, gestartet mit x, nach t Schritten hält.

Eingabe $y \in \{0,1\}^*$ für M_g wird interpretiert als (x,t)

$$\text{aus\_eins\_mach\_zwei}(y) = \begin{cases} (a_1....a_n, t) & \text{falls } y = a_1...a_n \underbrace{11...1}_{\text{Ende}} \\ (0,1) & \text{sonst} \end{cases}$$

Bsp: $(\underline{01101}\underline{0}\underline{111}) = (01101,3)$

$(\quad 011) = (\varepsilon,2)$

$(\quad 1111) = (0,1)$

$(\quad 0110) = (011,0)$

Sei $x_{fix} \in L$ ein fester Wert aus L

M_g: Eingabe sei y

(x,t) := aus_eins_mach_zwei(y):

Simuliere auf einem Extraband t Schritte von M gestartet mit x

Falls M eine Endkonfiguratiion erreicht hat, gib x aus

Sonst: gib x_{fix} aus

à die Punkte (i)(ii)(iii) gelten

- Eine Menge von Sprachen heißt **Sprachklasse** oder auch Sprachfamilie

$E := \{L \mid L \text{ ist entscheidbar}\}$

$L_0 := \{L \mid L \text{ ist rekursiv aufzählbar}\}$

- Sei $op(\cdot,\ldots,\cdot)$ eine k-stellige Operation auf Sprachen. (z.B. Vereinigung $\cup(L_1, L_2)$)

Eine Sprachklasse *L* ist genau dann unter op abgeschlossen (gegen op abgeschlossen), wenn gilt:

$$L_1,\ldots,L_k \in L \rightarrow op(L_1, L_2,\ldots,L_k) \in L$$

Techniken:
- "Parallele Ausführung" (2 Bänder für jeweils eine TM)
- "Zeitscheibenbetrieb"

Satz 1.24 Seien L_1 und L_2 **rekursiv aufzählbar**
- $L_1 \cup L_2$ rek. aufzählbar
- $L_1 \cap L_2$ rek. aufzählbar
- $L_1 \circ L_2$ rek. aufzählbar ($L_1 \circ L_2 = \{ w \mid w = uv,\ u \in L_1, v \in L_2 \}$)

Satz 1.25

L ist **entscheidbar** $\Leftrightarrow$ L und $\overline{L}$ sind rek. aufzählbar.

E ist unter Komplementbildung abgeschlossen.

L_0 ist unter Komplementbildung nicht abgeschlossen $\quad H \in L_0 \mid \overline{H} \notin L_0$ (H = Halteproblem)

Zusammenfassung: Kap1 (Berechenbarkeit)
- TM = RAM = JAVA-Programme
- TM = $O(t(n)^3)$ ß à t(n) = RAM
- Die TM als Prototyp für den Begriff "berechenbar".
  - Entscheidbarkeit (Maschine hält für jede Eingabe uns sagt ja oder nein)
  - Akzeptanz (Maschine hält nur für die "ja"-Eingaben) = *rek. aufzählbar*
- Halteproblem - ist nicht entscheidbar à Reduktionskonzept $[H \leq H_\varepsilon]$ "Jede nichttrivale Eigenschaft von Programmen ist nicht entscheidbar
  + ist rek. aufzählbar
- Universelle TM
- Church-Turing-These *(TM ist Grenze zu dem was berechenbar ist)*

Kapitel 2 Nichtdeterministische Turingmaschinen und das PNP Problem

Bei der PNP Problematik geht es um Sprachen, die entscheidbar sind, undzwar in Polynomzeit(?)

Def 2.1: Eine **nichtdeterministische 1-Band-TM** wird beschrieben durch 6 Komponenten = $M = (Q, \sum, \Gamma, \delta, q_0, F)$

deren Bendeutungen bis auf δ gleich denen der det. TM sind (analog k-Band-TM)

Nun ist $\delta : Q \times \Gamma = P(Q \times \Gamma \times \{L, N, R\})$

$\delta(q_3, q) = \{(q_2, b, L), (q_5, a, R)\}$

Startkonf.: q_0x , $x \in \sum^*$

ist z.B.: $(q', b, R) \in \delta(q, a)$, dann ist $\alpha q a \beta \rightarrow \alpha b q' \beta$ ein <u>möglicher</u> Übergang

eine NTM M <u>akzeptiert</u> x genau dann, wenn es eine Rechnung $q_0 x \overset{*}{\rightarrow} \alpha q \beta, q \in F$, aus möglichen Übergängen gibt.

M <u>akzeptiert</u> die Sprache L = { x | M akzeptiert x}

Bsp.: 2-Band-TM $\Gamma = \{0,1,B\}, \sum = \{0,1\}, F = \{q_2\}$

$a \in \{0,1\}$: $\delta(q_0, \begin{pmatrix} a \\ B \end{pmatrix}) = \left\{(q_o, \begin{pmatrix} a \\ a \end{pmatrix}, \begin{pmatrix} R \\ R \end{pmatrix}), (q_1, \begin{pmatrix} a \\ a \end{pmatrix}, \begin{pmatrix} R \\ N \end{pmatrix})\right\}$

$\delta(q_1, \begin{pmatrix} a \\ a \end{pmatrix}) = \left\{(q_1, \begin{pmatrix} a \\ a \end{pmatrix}, \begin{pmatrix} R \\ L \end{pmatrix})\right\}$

$\delta(q_1, \begin{pmatrix} B \\ B \end{pmatrix}) = \left\{(q_2, \begin{pmatrix} B \\ B \end{pmatrix}, \begin{pmatrix} N \\ N \end{pmatrix})\right\}$

M akzeptiert $L = \{ww^R \mid w \in \{0,1\}^*\}$ w^R ist das gespiegelte Wort zu w $(001)^R = 100$

$\Gamma = \{0,1,B\}, \sum = \{0,1\}, F = \{q_1\}$

$\delta(q_0, B) = \{(q_o, 0, R), (q_0, 1, R), (q_1, B, N)\}$

Diese NTM M schreibt eine beliebige 0-1-Folge aufs Band. Wir sagen: M rät eine 0-1-Folge

Rucksackproblem

K = { (Codierungen von) $(a_1,\ldots,a_n,b)$ | es gibt $\alpha_1,\ldots,\alpha_n \in \{0,1\} : \sum_{i=1}^{n} \alpha_i \alpha_i = b$ }

$(3,7,5,9,16) \in K$ *(fülle Rucksack bestmöglich mit Waren des gegebenes Volumens)*

$\underset{\alpha_1}{0}\ \underset{\alpha_2}{1}\ \underset{\alpha_3}{0}\ \underset{\alpha_4}{1}\ \ 0 \cdot 3 + 1 \cdot 7 + 0 \cdot 5 + 1 \cdot 9 = 16$

Vorgehensweise: - Rate eine 0-1-Folge $\alpha_1,\ldots,\alpha_n$

- Verifiziere $\sum_{i=1}^{n} \alpha_i \alpha_i = b$

NTM M Platz und Zeitbedarf:

Laufzeit: $L_M(x) = \begin{cases} \text{Länge einer } \underline{\text{kürzesten}} \text{ akzeptierenden Rechnung} & \text{falls M die Eingabe x akzeptiert} \\ 1 & \text{sonst} \end{cases}$

Platz: $S_M(x) = \begin{cases} \underline{\text{geringster}} \text{ Platzbedarf einer akz. Rechnung} & \text{falls M die Eingabe x akzeptiert} \\ 1 & \text{sonst} \end{cases}$

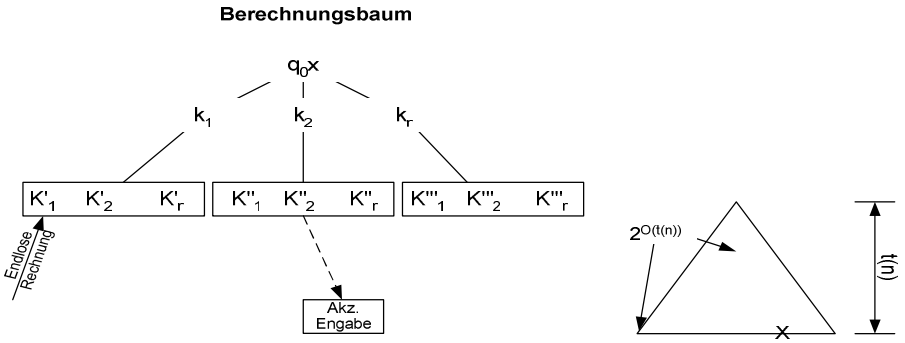
"$T_n(n)$-zeit und $S_m(n)$-platzbeschränkt ist analog den det.TM def.

Satz2.3: Jede NTM M kann durch eine det. TM M' simuliert werden. Falls M t(n)-zeit und s(n)-platzbeschränkt ist, so ist M' $2^{O(t(n))}$-zeit und O(s(n)*t(n))-platzbeschränkt.

Beweis: r sei die maximale Zahl an direkten Nachfolgekonfigurationen, die eine Konf. einer Rechnung von M haben kann.

$$r = \max\{\, |\delta(q,a)| \;|\; q \in Q, a \in \Gamma \}$$

Startkonf.: q_0x



1. Idee: Breiten suche auf dem Berechnungsbaum

Zeitbedarf: akz. Konf. ist spätestens auf Level t(n)

à Laufzeit von $M': O\left(r^{t(n)}\right) = O\left(2^{\log r \cdot t(n)}\right) = 2^{O(t(n))}$

Platz: $s(n) \cdot s^{O(t(n))}$ (steht nicht so im Satz à klappt nicht)

2. Idee: kontrollierte Fiefensuche auf dem Berechungsbaum

Platz. $Tiefe \cdot s(|x|) = O(t(|x|) \cdot s(|x|))$

T: = 2;
<u>while</u> *noch keine akz. Rechnung gefuden und der Baum noch nicht ganz abgearbeitet ist*
<u>do</u> *- führe Tiefensuche bis Tiefe T aus*
- t: = T+1
done

Laufzeit: $O\left(\sum_{T=2}^{t(|x|)} 2^{O(T)}\right) = 2^{O(t(|x|))}$

Korollar 2.4: **NTMs akzeptieren genau die rekursiv aufzählbaren Sprache**

Rechner, der 1000 Operation pro Sekunde ausführen kann.

		0,01s	**1s**	**1min**	**1h**	
P_1	n	10	1.000	60.000	3.600.000	Vorgegeben ist eine bestimmte Rechenzeit
P_2	n log n	4	140	4893	204.094	Gesucht: max. Problemgröße n, die in der vorgegebenen Zeit gelöst
P_3	n^2	3	31	244	1.897	werden kann.
P_4	n^3	2	10	10	153	
P_5	2^n	3	9	9	21	

Nun: Rechner, der 10 Mal so schnell ist, 10000 Op/s

		Max Eingabelänge Vorher(1000Op/s)	Nacher(10000 Ops/s	
P_1	n	m	10m	
P_2	n log n	m	fast 10m	Polynomielle Laufzeit à **„schnell“**
P_3	n^2	m	3,16m	
P_4	n^3	m	2,15m	
P_5	2^n	m	m+3.3	**„langsam“**

Def. 2.5: - DTIME(t(n)) = { L | es gibt eine O(t(n))-zeitbeschränkte det. TM, die L entscheidet }
- NTIME(t(n)) = { L | es gibt eine O(t(n))-zeitbeschränkte nicht det. TM, die L akzeptiert }

- $P = \bigcup_{k \in N} \mathrm{DTIME}(n^k)$

Menge aller in Polynominalzeit entscheid barer Programme ist hier mit enthalten

!!! T(n)-zeitbeschränkte RAM, kann von $O(t(n)^3)$-zeitbeschränkter det. TM simuliert werden (Satz 1.10)

- $NP = \bigcup_{k \in N} \mathrm{NTIME}(n^k)$

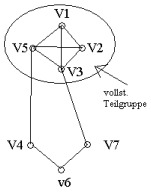
Offensichtlich: $P \subseteq NP$

Die Frage: P = NP ??? Vermutung: $P \subset NP$

Vorteile von P: - P ist robust
- Hängt nicht von der Anzahl der Bänder ab
- hängt nicht vom Maschinenmodell TM bzw. RAM ab
- Polynome sind unter Hintereinander-Ausführung abgeschlossen
$f(n) = n^3$ $g(n)=n^4$ $f(g(n)) = n^{12}$
- P enthält die Probleme, die sich in der Praxis als handhabbar erweisen haben
d.h. einigermaßen schnell gelöst werden können (*falls k eher klein*)
- P ermöglicht eine reichhaltige (und schöne) Theorie

- **Clique** = { <G,k> | G ist ein Graph der einen vollständigen Teilgraphen der Größe k enthält}

G_0:



$< G_0, 4 > \in$ CLIQUE

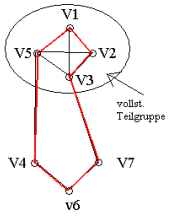
$< G_0, 5 > \notin$ CLIQUE

- **Hamilton-Kreis**-Problem (HC, Hamiltonian Cycle)

Ein Hamiltion-Kreis in einem Graphen G ist ein Kreis, der jeden Knoten genau einmal besucht.
(Alle Möglichkeiten müssen durchlaufen lassen bis Ergebnis gefunden oder Ende erreicht wurde. Das ist nur nicht det. möglich)

HC = { <G> | G enthält einen Hamilton-Kreis }

$< G_0 > \in$ HC



- **Traveling Salesperson Problem** (TSP)

TSP = { <G,c,k> | der Gewichtete Graph G enthält einen Hamilton-Kreis mit Gewicht <= k }
(Rate-Rundreise & rechne länge aus à ist mit NP möglich)

- **Knotenüberdeckungsproblem** (Vertex Cover)

Gegeben: Graph G = (V,E), Dann heißt $A \subseteq V$ eine Kantenüberdeckung, wenn jede Kante aus E mind. einen Knoten aus A berührt

VC = { <G,k> | G hat Knotenüberdeckung der Größe k }

$< G_0, 4 > \in VC$

$< G_0, 3 > \notin VC$

Def. 2.6: Sei L eine Sprache über { 0,1 }. Eine det. TM V_L heißt **t(n)-beschränkter Verifizierer** für L, wenn gilt:

(i) Die Eingaben von V_L sind von der Form x#w, $x, w \in \{0,1\}$

(ii) Die Laufzeit ist $O(t(|x|))$

(iii) Für alle $x \in \{0,1\}^*; x \in L \quad x \in L \Leftrightarrow \exists w: \underbrace{|w| \leq t(|x|)}_{\text{Zertifikat HILFE}}$ und V_L akzeptiert x#w

Rechne aus Formel x eine Nullstelle aus und teste mit w (0 einsetzten) ob das Ergebnis korrekt ist

Satz 2.7: Sei L eine Sprache. Dann ist.

$L \in \text{NTIME}(t(n)) \Leftrightarrow$ es gibt einen t(n) - Beschränkten Verifizierer V_L für L

Eingabe: x mit der Frage "$x \in L$" — Eingabe: x#w

Beweis: „$\Leftarrow$“ wir geben eine NTM M an:

(1) Eingabe sei x
(2) Rate w (in |w| Schritten)
(3) Falls V_L gestartet mit x#w akzeptierend hält, dann akzeptiere x
sonst verwerfe x

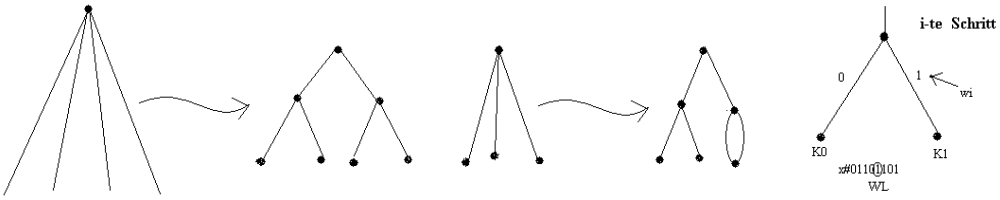
Offensichtlich akzeptiert M nur Eingaben $x \in L$.

Aus Def. 2.6 (iii) folgt, dass es ein w mit $|w| \leq t(|x|)$ gibt, V_L akzeptiert x#w in Zeit $O(t(|x|))$. Raten des „richtigen“ w braucht also nur Zeit $O(t(|x|))$. Gesamtzeit ist $O(t(|x|))$

„$\Rightarrow$“ $L \in \text{NTIME}(t(n))$; M sei t(n)-zeitbeschränkte nicht det. TM für L

OBdA *(ohne Beschränkung der Allgemeinheit)*
OE *(ohne Einschränkung)*
wlog (*without loss of generallity)*

OBdA: Jede Konfiguration einer Rechnung von M hat keine oder genau zwei Nachfolgekonfigurationen im Berechnungsbaum:



Bei Eingabe x#w arbeitet V_L wie folgt:

Falls M im 2-ten Schritt die Wahl zwischen zwei Nachfolgekonfigurationen K_0 und K_1 hat, wählt V_L die Konfiguration K_{w_i} mit w = $w_1 w_2 \ldots w_{t(|x|)}$, $w_i \in \{0,1\}$

$x \in L \Leftrightarrow$ es gibt eine akzeptierende Rechnung von M gestartet mit x der Laufzeit $t(|x|)$

$\Leftrightarrow$ es gibt ein $w \in \{0,1\}$, so dass V_L x#w akzeptiert, da jedes w eine Rechnung von M gestartet mit x beschreibt.

Korollar 2.8:

NP = { L | es gibt eine polynominell beschränkten Verifizierer für L }

2.1 NP-Vollständigkeit

Def. 2.9: $L_1 \subseteq \sum_1^*, L_2 \subseteq \sum_2^*$

L_1 ist polynominell reduzier bar auf L_2 $(L_1 \leq_p L_2)$

$\Leftrightarrow$ (i) $(L_1 \leq L_2)$

(ii) Die Laufzeit zur Berechnung von f(x) ist $O(|x|^k)$ für ein $k \in N$

Lemma 2.10: „$\leq_p$“ ist transitiv

Beweis: $L_1 \leq_p L_2$ mittels f_1, berechenbar in Zeit $O(n^{k_1})$

$L_2 \leq_p L_3$ mittels f2, berechenbar in Zeit $O(n^{k_2})$

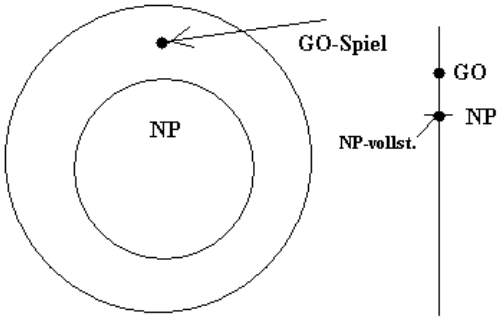
$\Rightarrow L_1 \leq_p L_3$ mittels $f_2(f_1(x)) = f_3(x)$

$f_3(x)$ ist Reduktionsfunktion der Laufzeit $O((n^{k_1})^{k_2}) = O(n^{k_1 \cdot k_2})$

Def. 2.11: L heißt **NP-schwer** *(NP-hard, NP-schwierig, NP-hart)* wenn gilt:

$\forall L' \in NP : L' \leq_p L$

Fast alle 2 Personen spiele sind nicht in NP weil exponentiell viele Züge möglich sind
Welt außerhalb von NP ist nicht leer.



L heißt **NP-vollständig** falls gilt:

(i) $L \in NP$

(ii) L ist NP-schwer

Lemma 2.12:

(i) L sei NP-schwer. dann gilt:

(a) $L \in P \Rightarrow P = NP$

(b) $P \neq NP \Rightarrow L \notin P$

(ii) L sei NP-vollständig. dann gilt:

(a) $L \in P \Leftrightarrow P = NP$

(b) $L' \in NP$ und $L \leq_p L' \Rightarrow L'$ ist NP-vollständig

2.2 Der Satz von Cook

Def. 2.13 - $V = \{x_1, ..., x_n\}$ Menge Boolescher Variablen

Literal ist eine Variable oder eine negierte Variable x oder $\bar{x}$

Eine **Klausel** K der Länge s ist ein Boolescher Ausdruck aus s Literalen.

K = y_1 v y_2 v … y_s mit $K \in \{x_1, ..., x_n\} \cup \{\bar{x}_1, ..., \bar{x}_n\}$

- Ein Ausdruck in **konjunktiver Normalform (KNF)** ist ein Boolescher Ausdruck □ der Form

□ $= K_1 \wedge K_2 \wedge ... \wedge K_t$ für Klauseln K_i

Die Anzahl der in □ vorkommenden $\wedge$ und $\vee$ Operation ist die **Größe** Size(□) von □.

- Eine totale Abbildung c:V à {TRUE,FALSE}, die jeder Variable einen Wahrheitswert zweist, heißt Belegung der Variablen. c wird kanonisch auf Literale, Klauseln K, und die KNF □, festgesetzt. c(K) bzw. c(□) ist das Ergebnis der Auswertung von K bzw. □ unter Auswertung von c.

- Eine KNF □ heißt **erfüllbar**, wenn es eine Belegung c gibt mit c(□) = TRUE

Bsp.: □ $= (x_1 \vee \bar{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \bar{x}_4)$ Size(□ = 5)

$c(x_1)$:=TRUE $c(x_3)$,$c(x_4)$ BELIEBIG

$c(x_2)$:=TRUE à c(□)=TRUE

- □ wird kodiert durch <□>:

$< x_i >$ = 1#bin(i)

$< \bar{x}_i >$ = 0#bin(i)

<K> = < y_1 v y_2 v … y_s > = <y_1>##<y_2>##...##<y_s>

< □> = $< K_1 \wedge K_2 \wedge ... \wedge K_t >$ = <K_1>###...###<K_t>

|<□>| = O(Size(□)*log|V|)

Dar **Erfüllbarkeitsproblem** SAT (Satisfability Problem) ist SAT = {□□> | Φ ist erfüllbare KNF}.

Satz 2.14 (Satz von Cook, Cook's Theorem, 1971): **SAT ist NP-vollständig**.

Beweis: (i) SAT ∈ NP: □Öe c und verifiziere, das c(Φ) = TRUE ist.
Das geht in Zeit polynominell in|<□>|

(ii) SAT ist NP-schwer, also zu zeigen $\forall L \in NP : L \leq_p SAT$

Sei L beliebig aus NP, aber fest. Sei M_L eine nicht det. polynominalzeit-beschränkte 1-Band-TM, die L akzeptiert in Zeit $c \cdot n^k$;

$M_L = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Ziel: Für $w \in L$ □□ss eine KNF Φw konstruiert werden („mit vielen Variablen"), so dass gilt:

$w \in L \Leftrightarrow$ □$_w \in SAT$ □$_w$ *soll die möglichen Polynome von M_L gestartet mit nachbilden*

OBdA: $F = \{q_F\}, \delta(q_F, a) = \{(q_F, a, N)\}$ für alle $a \in \Gamma$

$w \in T \Leftrightarrow$ es gibt eine Rechnung von M_L gestartet mit w der Länge $T = c \cdot n^k$ mit n=|w|

$\Leftrightarrow q_0 w$ a K_1 a K_2 a ... a K_T und hier ist der Zustand q_F

die Variablenmange $V_{M_L} = \{\text{zelle}_{t,i,a} \mid 0 \leq t \leq T, -T \leq i \leq T, a \in \Gamma\}$

$\cup \{\text{kopf}_{t,i} \mid 0 \leq t \leq T, -T \leq i \leq T\}$

$\cup \{\text{zustand}_{t,q} \mid 0 \leq t \leq T, q \in Q\}$

$|V_{M_L}| = (T+1)\cdot(2T+1)\cdot|\Gamma| + (T+1)\cdot(2T+1) + (T+1)\cdot|Q| = O(T^2)$

$\text{zelle}_{t,i,a}$ = TRUE $\Leftrightarrow$ in der Rechnung von M_L gest. mit w steht in der Konf. K_t in Zelle i das Zeichen a

$\text{kopf}_{t,i}$ = TRUE $\Leftrightarrow$ In der Berechnung von M_L gest. mit w seht in der Konf. K_t der Kopf an der Zelle i

$\text{zustand}_{t,q}$ = TURE $\Leftrightarrow$ In der Berechnung von M_L gest. mit w seht in der Konf. K_t der Zustand q

□ Öiesen Variablen konstruieren wir jetzt Φ_W, so dass die Rechnung $q_0 w$ a ... a K_T beschrieben wird

GenauEineVariableIstTrue(x_1,…,x_2) = $(x_1 \vee \ldots \vee x_r) \wedge \bigwedge_{i+j \text{ Größe } O(r^2)} (\bar{x}_i \vee \bar{x}_j)$

istGleich(x,y) = $(\bar{x} \vee y) \wedge (x \vee \bar{y})$

V_t = { $\text{Zelle}_{t,i,a}$,$\text{Kopf}_{t,i}$,$\text{zustand}_{t,q}$ | $-T \leq i \leq T, a \in \Gamma, q \in Q$ }

Konf(V_t) = TRUE $\Leftrightarrow$ die Belegung der Var, in V_t beschreibt genau eine Konfiguration

Konf(V_t) = GenauEineVariableIstTrue$(\text{zustand}_{t_i,q_0}, \text{zustand}_{t_i,q_1}, \ldots, \text{zustand}_{t_i,q_{|a|-1}})$

$\wedge$ GenauEineVariableIstTrue$(\text{kopf}_{t,-T}, \ldots, \text{kopf}_{t,0}, \text{kopf}_{t,1}, \ldots, \text{kopf}_{t,T})$

$\wedge \bigwedge_{-T \leq i \leq T}$ GenauEineVariableIstTrue$(\text{zelle}_{t,i,a_1}, \ldots, \text{zelle}_{t,i,a_{|r|}})$

Größe: O(T^2)

□$_W$ = Rechnung (V0,V1,…,VT)

= $\underbrace{\text{Startkof}(V_0)}_{\text{hier geht die konkrete Eingabe w für } M_L \text{ in dei KNF ein}} \wedge \bigwedge_{t=0}^{T-1} \underbrace{\text{EinSchritt}(V_t, V_{t+1})}_{\text{hier geht das } \delta \text{ in von } M_L \text{ in die Konstruktion ein}} \wedge \text{zustand}_{T,q_F}$

Startkonf(V0) = $\bigwedge_{i=0}^{n-1} (\text{zelle}_{0,i,W_{i+1}}) \wedge \bigwedge_{\substack{-T \leq i < 0 \\ n \leq i \leq T}} (\text{zelle}_{0,i,B}) \wedge \text{kopf}_{0,0} \wedge \text{zustand}_{0,q_0}$

/* Dass Startkonf wirklich eine Konfiguration bezeichnet, wird bei EinSchirtt( ) mit überprüft */

EinSchritt(V_t,V_{t+1}) = $\bigcup_{\gamma \in \delta} \text{Schritt\_durch\_□}(V_t, V_{t+1})$

$\gamma = [(q', a', D) \in \delta(q, a)]$

□öhritt_durch_γ(V_t,V_{t+1}) = Konf(V_t) $\wedge$ Konf(V_{t+1}) $\wedge$ $\text{zustand}_{t,q}$ $\wedge$ $\text{zustand}_{t+1,q'}$

$\wedge$ "unter Kopf in K_t steht a"

$\wedge$ " an der selben Position in K_{t+1} steht a' "

$\wedge$ " kopf wird in Richtung D bewegt"

$\wedge$ " Rest vom Bandinhalt ist unverändert beim Übergang von K_t nach K_{t+1}"

„unter Kopf in K_t steht a und an derselben Stelle in K_{t+1} steht a' „

$$\bigwedge_{i=-T}^{T-1} \left(\overline{\text{kopf}}_{t,i} \vee \left(\text{zelle}_{t,i,a} \wedge \text{zelle}_{t+1,i,a'}\right)\right)$$

D = L: „Kopf wird in Richtung D bewegt“

$$\bigwedge_{i=-T+1}^{T-1} \left(\text{SindGleich}\left(\text{kopf}_{t,i}, \text{kopf}_{t+1,i-1}\right)\right)$$

D = R, D = N …
Größe von $\Box_W$ ist $O(T^3) = O(|w|3^k)$

□d.h. die Laufzeit zur Berechnung von $<\Phi_W>$ ist $O(|w|3^k)$, also Polynominell

$w \in L \Leftrightarrow$ es gibt eine akz. Rechnung von M_L gestartet mit w der Laufzeit $c \cdot n^k \Leftrightarrow$ □ö $_W>\in$ SAT

$\Rightarrow$ diese akzeptierende Rechnung liefert eine Belegung der Variablen, die $\Box_W$ erfüllt

$\Rightarrow$ □ö $_W>\in$ SAT

□ö $_W>\in$ SAT à $\Box_W$ ist erfüllbar à jede Klausel ist erfüllt

à entspricht eine akz. Rechnung von ML gest. mit w der Länge $c \cdot |n|^k$

à $w \in L$ à **Bewiesen □**

□□ist KNF, in der jede Klausel aus genau 3 Literalen aus 3 verschiedenen Variablen besteht.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$$

□ũAT = {<Φ>| □ŭst *(„das was drüber steht“)*, und □□ist erfüllbar}
Offensichtlich: 3SAT $\subseteq$ SAT

Satz 2.15: **3SAT ist NP-vollständig**

Beweis: (i) 3SAT ∈ NP √

(ii) SAT $\leq_p$ 3SAT

, □ ∈ SAT ? $\Leftrightarrow$ “f(<□>)∈ SAT ?

□ = $K_1 \wedge K_2 \wedge \ldots \wedge K_T$

K_i = l (literal) $f(K_i) = (l \vee y_{i1} \vee y_{i2}) \wedge (l \vee \overline{y}_{i1} \vee y_{i2}) \wedge (l \vee y_{i1} \vee \overline{y}_{i2}) \wedge (l \vee \overline{y}_{i1} \vee \overline{y}_{i2})$

size(K_i) = 0 size(f(K_i) = 11

$K_i = (l_1 \vee l_2)$ $f(K_i) = (l_1 \vee l_2 \vee y_i) \wedge (l_1 \vee l_2 \vee \overline{y}_i)$

size(K_i) = 1 size(f(K_i) = 5

$K_i = (l_1 \vee l_2 \vee \ldots \vee l_k), k > 3$

$f(K_i) = (l_1 \vee l_2 \vee y_{i1}) \wedge (\overline{y}_{i1} \vee l_3 \vee y_{i2}) \wedge (\overline{y}_{i2} \vee l_4 \vee y_{i3}) \wedge \ldots \wedge (\overline{y}_{i\cdot k-3} \vee l_{k-1} \vee l_k)$

size(K_i) = k-1 size(f(K_i) = 3k-7

f(□)= $f(K_1) \wedge \ldots \wedge f(K_T)$
size(f(□)) = O(size(□))

□ ist erfüllbar $\Leftrightarrow$ f(□) erfüllbar

f ist in Polynom zeit berechenbar. à **Bewiesen □**
Vermutung: 3SAT ∉ P
aber: 2SAT ∈ P

Satz 2.16: **CLIQUE ist NP-vollständig**: CLIQUE = { <G,k> | G enthält einen vollst. Teilgraphen der Größe k }

Beweis: (i) CLIQUE$\notin$ NP offensichtlich

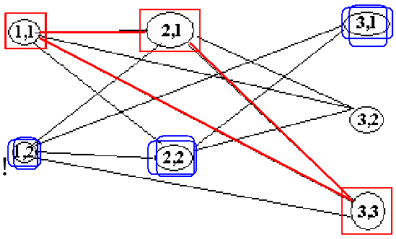
(ii) CLIQUE ist NP-schwer. Z.z: SAT $\leq_p$ CLIQUE *SAT auf CLIQUE reduzieren*

geg.: $\square = \bigwedge_{i=1}^{T} (K_i), K_i = y_1^{(i)} \vee \ldots \vee y_{s_i}^{(i)}, y_{s_i}^{(i)}$ ist literal

ziel: Konstruktion eines Graphen $G_\square$ und Angabe eines k$\in$ N, mit <$\square$>$\in$ SAT $\Leftrightarrow$ < $G_\square$,k>$\in$ CLIQUE

$G_\square$: $V = \{(i, j_i) \mid i \in \{1,T\}, j_i \in \{1,\ldots,s_i)\}$

$E = \{\{(i, j),(i', j')\} \mid i + i' \text{ und } y_j^{(i)} \neq y_{j'}^{(i')}\}$



$\square = (x_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$ In den Spalten gibt es keine Kanten!

$x_1 = x_3 = TRUE \quad x_2 = FALSE = x_3 = x_1$

Behauptung: $\square$ ist erfüllbar $\Leftrightarrow$ $G_\square$ enthält eine Clique der Größe T

Beweis: $\Rightarrow$: c = (c_1,…,c_n) sei eine erfüllende Belegung der Variablen (x_1,…,x_n), c$\in$ {TRUE,FALSE}

Dann wird in jedem K_i mindestens ein Literal $y_j^{(i)}$ erfüllt

OBdA seien diese $y_1^{(1)},\ldots, y_1^{(i)}$, Dann ist nie $\bar{y}_1^{(i)} = y_1^{(i')}, i \neq i'$;

also ist immer { (i,1), (i,1) } $\in$ E, d.h. (1,1), … ,(T,1) bilden eine Clique der Größe T

$\Leftarrow$: Sei { (i_1,j_1), … , (i_T,j_T) } eine Clique der Größe T in $G_\square$

Dann ist wegen des „ i $\neq$ i‘ „ in der Def. von E aus jeder Spalte des Graphen genau ein Knoten vorhanden. Durchnummerieren zu { (1,l_1),(2,l_2), … , (T,l_T) }

Seien $y_{l_i}^{(i)} = x_{s_i}^{\alpha_i}, \alpha \in \{\text{negiert}, \text{nicht negiert}\}$ z.B. $y_{l_i}^{(i)} = \bar{x}_1, \alpha_i = \text{negiert}$

Es sind alle (s_i, α_i) verschieden, sonst gäbe es i,i‘ mit $y_{l_i}^{(i)} = \bar{y}_{l_{i'}}^{(i')}$

Somit gibt es eine Belegung, die alle Literale $y_{l_i}^{(i)}, i \in \{1,\ldots,T\}$ erfüllt

Damit ist jede Klausel K erfüllt, dann ist $\square$ erfüllt à **Bewiesen** □

Die Konstruktion von $G_\square$ ist offensichtlich in Zeit O(poly(|<$\square$>|)) möglich à **Satz** □

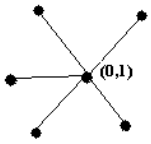
Satz 2.17: **VERTEXCOVER ist NP-vollständig**

Beweis: CLIQUE auf VERTEXCOVER reduzieren

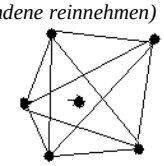
(i) VC$\in$ NP ist offensichtlich

(ii) VC ist NP-schwer

zu zeigen: CLIQUE $\leq_P$ VC



K(G) sei der Komplementgraph zu G = (V,E) *(Kanten die Vorhanden rausnehmen und nichtvorhandene reinnehmen)*

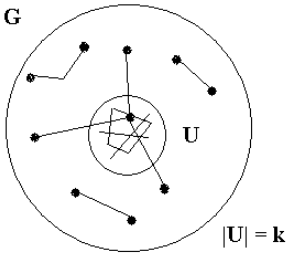


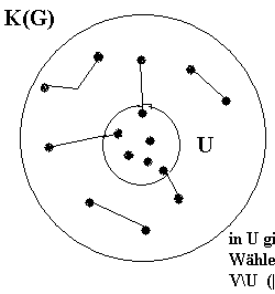
$f(< G, k >) = \langle K(G), |V| = k \rangle$

Behauptung: <G,k>$\in$ CLIQUE $\Leftrightarrow$ <K(G),|V|=k> $\in$ VC

d.h.: G enthält k-CLIQUE $\Leftrightarrow$ K(G) enthält keine Knotenüberdeckung der Größe |V| = k

<G,k>$\in$ CLIQUE





V\U ist tatsächlich eine Knotenüberdeckung, da es innerhalb von U keine Kanten gibt

à **BEHAUPTUNG** □

à **SATZ** □

Binary Programming BP $n \times m$ Matrizen mit ganzzahligen Einträgen

$$BP = \left\{ < A, b > \middle| A \in M_{n,m}(Z), b \in Z^n, \exists y \in \{0,1\}^n : A \cdot y \leq b \right\}$$

„Lineare Ungleichungssysteme, in denen die Variablen nur mit 0 oder 1 belegt werden dürfen“

Satz 2.18: **BP ist NP-vollständig**

Beweis: (i) BP∈ NP (ist offensichtlich)
(ii) BP ist NP-schwer
zu zeigen: SAT $\leq_P$ BP

□ = $K_1 \wedge K_2 \wedge ... \wedge K_T, K_i = y_1^{(i)} \vee ... \vee y_{s_i}^{(i)}$ über {x_1, … x_n }
Ungleichungssystem über den Variablen z_1, … , z_n , z_1‘, … , z_n‘

für jede Boolesche Variabel x_i: $z_i + z_i' = 1$ *$z_i = 0$ à $z_i' = 1$ und umgekehrt*

für jede Klausel K_i: $\sum \tilde{y}_j^{(i)} \geq 1, \tilde{y}_j^{(i)} = \begin{cases} z_l & y_j^{(i)} = x_l \\ z_l' & y_j^{(i)} = \bar{x}_l \end{cases}$ $x_1 \vee \bar{x}_2 \vee \bar{x}_3$
1 2 3
mind 1 muss 1 sein

$x_1 \vee \bar{x}_2 \vee \bar{x}_7 \vee x_9$

$\hat{=} z_1 + z_2' + z_7' + z_9 \geq 1$

Eine erfüllende Belegung liefert Lösung des Ungleichungssystems und umgekehrt □

Aufbauend auf die NP-Vollständigkeit:
- Approximationsalgorithmen
- Komplexitätstheorie
  - Platzhierarchie
  - Zeithierarchie
  - Konsequenzen für Nichtdeterminismus

Es gibt keine Perfekten Schachalgorithmen weil NP-Vollständig und nicht die Optimale Lösung geliefert wird sondern nur eine gute Lösung

Kapitel 3: Formale Sprachen

Bisher: Maschinen erkennen Wörter einer Sprach

Jetzt: Verfahren, Methoden, die Wörter einer Sprache erzeugen

L = { <M> | <M> ist ein syntaktisch korrektes C-Programm }

Sie kennen Syntax-Diagramme, die Programmiersprachen definieren

3.1 Grammatiken

Def. 3.1: **Eine Grammatik G vom Typ Chomsky-0**

durch ein 4-Tupel (V,∑,P,S) mit

- V: endliche Menge von Variablen (Konvention: Große Buchstaben, wie A,B,C)
- ∑: endliches Alphabet von Terminalen (Konvention: kleine Buchstaben oder Ziffern: 0,1,… a,b,…
- S: Start symbol, S∈ V
- P ⊆ $\left((V \cup \square)^* / \square^* \times (V \cup \square)^*\right)$ ist endliche Menge von Produktionen aller Ersetzungsregeln

  z.B. (aAb,Bq) ∈ P ist erlaubt

  statt (u,v) ∈ P schreiben wir auch u à v

Wir sagen: - w‘ ∈ $(V \cup \square)^*$ ist aus w ∈ $(V \cup \square)^*$ direkt ableitbar (w $\longrightarrow$ w‘) fall es (u $\longrightarrow$ v) ∈ P und

$\alpha, \beta \in (U \cup \Sigma)^*$ mit w = □őβ und w‘ = □Őβ

- w‘ ist aus w indirekt ableitbar (w $\xrightarrow{*}$ w‘),

  falls w* durch endlich viele Ableitungsschritte aus w erzeugt ist.

Die von G erzeugte Sprache ist $L(G) = \{w \in \Sigma^* \mid s \xrightarrow{*} w\}$

Um zu zeigen, dass eine Grammatik G eine Sprache L erzeugt, ist zu zeigen:

L(G) ⊆ L und L(G) ⊇ L

$w \in (V \cup \square)^+$ und $s \xrightarrow{*} w$: w heißt Satz form. Satzformen können noch Variablen enthalten.

Bsp. auf HP

*(1)*S à aSBC (2)S à aBC

(3)CB à BC (4)aB à ab

(5)bB à bb (6)bC à bc

(7)cC à cc

Eine mögliche Ableitung:

$S \rightarrow^{(1)} aSBC \rightarrow^{(2)} aaBCBC \rightarrow^{(4)} aabCBC \rightarrow^{(6)} aabBC$

$\notin$ L
keine weiteren
Möglichkeiten

$S \rightarrow aSBC \rightarrow aaBCBC \rightarrow aabCBC \rightarrow aabBCC \rightarrow aabbCC \rightarrow aabbcC \rightarrow aabbcc \in L$

L(g) = { $a^n b^n c^n$ | n s ≥ w 1 }

Def. 3.1: (Fortsetzung)

g = (V,∑,P,S)

- G heißt **kontext sensitiv** oder vom Typ Chomsky-1,

  falls für jede Regel ($u \longrightarrow v$) gilt : $|u| \leq |v|$ *linke Seite immer kleiner als rechte Seite (kann nur länger werden, nicht kürzer*
- G heißt **kontext frei** oder vom Typ Chomsky-2

  falls für alle Regeln ($u \longrightarrow v$)∈ P gilt: u∈ V und v ∈ $(V \cup \square)^*$ A à aBCa

  (kontext-freie Grammatiken sind das Grundgerüst für Programmiersprachen)
- G heißt regulär oder vom Typ Chomsky-3,

  falls alle Regeln von der Form u à v sind und u∈ V und v = ε , oder v = a, a∈ □

  oder v = aw, a∈ □, w∈ V *(A à ε ,A à a,A à bC)*

Erweiterung zu “kontext-sensitiv”: S à ε ist zusätzlich erlaubt und S taucht in keiner rechten Seite der Produktion auf.

Satz 3.2: $L(G) = \{ a^n b^n c^n \mid n \geq 1 \}$

Beweis: „$\supseteq$“ Sei $a^n b^n c^n$, $n \geq 1$, gegeben. Ziel ist es eine Herleitung anzugeben.

1. Wende (n-1) mal die Prod(1) an und einmal die Prod(2).
   $S \xrightarrow{*} a^n (BC)^n$
2. Wende 1 + 2 + 3 + … + n-1 = n(n-1)/2 mal die Regel (3)
   $S \xrightarrow{*} a^n B^n C^n$
3. Wende einmal Regel (4) und dann (n-1) mal Regel(5) an
   $S \xrightarrow{*} a^n b^n C^n$
4. Wende einmal Regel (6) und dann (n-1) Mal Regel (7) an

$G = (V, \Sigma, P, S)$ mit

$V = \{S, B, C\}$ Menge der Variablen

$\Sigma = \{a, b, c\}$ Menge der Terminale

Menge P der Produktionen:

(1) $S \rightarrow aSBC$ (2) $S \rightarrow aBC$

(3) $CB \rightarrow BC$ (4) $aB \rightarrow ab$

(5) $bB \rightarrow bb$ (6) $bC \rightarrow bc$

(7) $cC \rightarrow cc$

„$\subseteq$“ In jeder Regel ist die Anzahl der a’s gleich der Anzahl der b’s und B’s
In jeder Regel ist die Anzahl der b’s und B’s gleich der Anzahl der c’s und C’s
à in jeder Satz form ist die Anzahl der a’s gleich der Anzahl der (b’s und B’s)
und gleich der Anzahl der (c’s und C’s).

a’s können nur aus (1) und (2) erzeugt werden, und daher stehen sie ganz vorne.
D.h. in einem w∈ L(G) stehen a’s immer ganz vorne.
b’s werden durch die Regeln (4) und (5) erzeugt
c’s werden durch die Regeln (6) und (7) erzeugt
Satzformen, die „cB“ enthalten, können nicht zu einem Wort der Sprache abgeleitet werden,
das das große B nicht mehr wegzubekommen ist.
à in w∈ L(G) sind die a’s vor den b’s, und die b’s vor den c’s,
d.h. $w = a^n b^n c^n$, $n \geq 1$

Satz 3.3. Eine Sprache L ist genau dann rekursiv aufzählbar, wenn es eine Chomsky-0-Grammatik G, mit L(g) = L gibt
(L0: Klasse der rek. aufzählbaren Sprachen, hat was mit Typ 0 Grammatiken zu tun)

Beweis: „$\Leftarrow$“ G gegeben. Ziel: TM M angeben, die genau die w∈ L(G) akzeptiert.
TM M: Eingabe w
Rate irgendeine Ableitung eines Wortes w‘(∈ L(G))
und vergleiche w mit w‘. falls w = w‘: stop.

„$\Rightarrow$“ TM M gegeben Ziel: Grammatik G anzugeben mit L(G) = { w | M gest. mit w hält }
- es gibt nur einen akzeptierenden Endzustand: q_{accept} ; und wenn der erreicht ist, ist das Band lehr
- M arbeitet nur auf en Zellen i, i >= 0; und Zelle-1 enthält ►
- M ist nur zu Beginn in Zustand q_0
  ►q_0 à K1 à … à ►q_{accept}

Ziel: Grammatik angeben, die diese Rechnung „rückwärts“ ausführt

- $V = Q \cup \Gamma / \Sigma \cup \{S\}$ Startsymbol: S
  $S \xrightarrow{*}$ ►q_{accept} q_{accept} à q_{accept}B
- $\delta(q,a) = (q', a', R)$ *rechts Bewegung*
  □⊨ αqaβ à □□= αa’q’□
  a’q‘ à qa für alle Einträge in δ mit Kopfbewegung nach <u>rechts</u>
- $\delta(q,a) = (q', a', N)$
  □⊨ αqaβ à □ǔ= αq’a‘□
  q’a‘ à qa alle Einträge in δ-Tabelle mit Kopfbewegung nach <u>N</u>
- $\delta(q,a) = (q', a', L)$
  □⊨ αb□□β à ﻭ □= αq’ba’□ □*ır brauchen das letzte Zeichen von α /in qa passiert was und b ist neuer zustand*

□*β sind mehrere Zeichen rechts und Links von der Aktuellen Position.*

Für alle $b \in \Gamma$:= q’ba‘ à bqa für alle Einträge der δ-Tabelle mit Richtung L

$S \xrightarrow{*}$ ►q_{accept}B…B $\xrightarrow{*}$ ►q_0wBB…B $\xrightarrow{*}$ w

$\forall a \in \Sigma : q_0 a \rightarrow a q_0$ *// laufe nach rechts*

$q_0 B \rightarrow q_0$ *// lösche die Bs wenn q_0B kommt werden die B aufgefressen*

$\forall a \in \Sigma : a q_0 \rightarrow q_0 a$ *// laufe nach links*

►q_0 à ε *// lösche ►und q_0*

Die Grammatik funktioniert so: 1.Die End-Konfiguration inklusive B’s erzeugen
2. Rechnung von M rückwärts durchführen
3. Aufräumen

□

3.2 Kontextfreie Grammatiken und kontextfreie Sprachen

Eine Sprache L heißt **kontextfrei** (kf), wenn es eine kf Grammatik G gibt mit L = L(G)

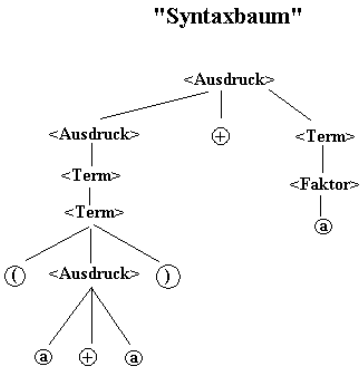
Bsp.: G_1 S à aSb | ε $L_1 = \{a^n b^n \mid n >= 0\} = L(G_1)$

Eigentlich müsste noch gezeigt werden, dass L(G1) = L1 mit beiden Richtungen „$\subseteq$“ und „$\supseteq$“

G_2: <Ausdruck> à <Term>
<Ausdruck> à <Ausdruck> + <Term>
<Term> à <Vaktor> | <Term> * <Faktor>
<Faktor> à a | (<Ausdruck>) $\sum = \{*,a,(,)\}$

(a+a)+a ε $L(G_2)$

<Ausdruck> à <Ausdruck> + <Term>
à <Term> + <Term>
à <Faktor> + <Term>
à (<Ausdruck>) + <Term>
à (<Ausdruck> + <Term>) + <Term>
à (<Term> + <Term>) + <Term>
à (<Faktor> + <Term>) + <Term>
à * (a + a) + a



Linksableitung: Ersetze immer die linkste Variable (Nichterminale Symbol)

Grammatik ist eindeutig: Es gibt nur eine Herleitung (einen Syntaxbaum)

Def. 3.4 Eine kf. Grammatik ist in **Chomsky-Normalform**

Wenn jede Regel von der Form ist: A à BC ,B,C∈ V\{S}
A à a a ∈ Σ
S à ε erlaubt *//Startsymbol*

Satz 3.5: Zu jeder kf. Grammatik G kann eine kf. Grammatik G‘ in Chomsky Normalform konstruiert werden, so dass L(G) = L(G‘)

Beweis: Lemma 3.6: zu G gibt es eine kf. Grammatik G ε -frei mit L(G) = L (G ε -frei)
und G ε -frei enthält keine ε -Regeln (bis auf S à ε)

Beweis von L 3.6: $E_0 = \{A \mid (A \to \square\square \in P\}$

$E_i = \{A \mid (A \to B_1 ... B_i) \in P, B_i \in E_{i-1}\} \cup E_{i-1}$

Es gibt ein i_0 mit $E_{i_0} = E_{i_{0+1}}$

- Lösche die ε -Regeln
- $\forall (A \to w) \in P$: Wenn w k Variablen aus E_{i_0} enthält, füge s^k-1 mögliche Regeln, die man durch Weglassen von Variablen aus E_{i_0} in w erhalten hinzu, außer A à ε **□(L. 3.6)**

Beispiel: $E_{i_0} = \{B, C\}$

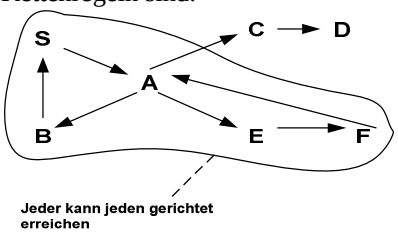
die Möglichkeiten $B \to \varepsilon, C \to \varepsilon$ sind nicht erlaubt und müssen ersetzt werden

nur das Startsymbol darf auf ε abgebildet werden

$$A \to BaAbC \mid aAbC \mid BaAb \mid BaAb \mid aAb$$

Lemma 3.7: Zu jeder kf. Grammatik G gibt es eine kf. Grammatik G‘ ohne Kettenregeln (Regeln der Form A à B) mit L(G= = L(G‘).

Beweis: Konstruiere einen gerichteten Graphen $G_{Kette} = (V, E_{Kette})$, in dem die Knoten die Variablen sind (V) und die Kanten die Kettenregeln sind.



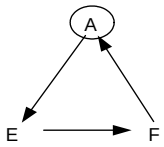
S→A E→F
A→B F→A
B→S A→C
A→E C→D

// in den „starken Zusammenhangskomponenten“ (z.B.: das Eingekreiste) gibt es die Möglichkeit Knoten zu ersetzten

Ersetze in den starken Zusammenhangskomponenten (Menge der Knoten wo jeder, die gegenseitig erreichbar sind hier: AEFBS *jeder kann jeden erreichen*) die Variablen durch eine davon (S muss übrig bleiben) und ersetze diese in allen Regeln an Stelle der aliminierten Knoten(Variablen).

Bsp.:

A à BC|~~E~~	A à BC
~~E~~ à XY|~~F~~	A à XY
~~F~~ à A	
Z à a~~F~~	Z à aA



Ersetze nun im übriggebliebenem gerichteten kreisfreiem Graphen „von unten nach oben“ („topologische Orientierung“) die Kettenregeln A à B durch die Regeln: A à “*alte rechten Seiten von B*“ **□(L3.7)**

Wir konstruieren G‘ in Chomsky-Normalform

- Falls $S \in E_{i_0}$: füge ein neues Startsymbol S und die Regel S‘ à S ein
- eliminiere die ε -Regeln
- eliminiere die Kettenregeln
- für alle a∈ Σ: A_a à a und ersetze alle Vorkommen von a durch A_a
  außer wenn dadurch neue Kettenregeln entstehen würden
  X à a wird nicht ersetzt weil sonst X à A_a wieder Verkettung ist.
- Regeln Nr.i: A à $B_1B_2 \ldots B_k$ k≥3
  Wird ersetzt durch:
  A à $B_1C_1^{(i)}$ $C_1^{(i)}$ à $B_2C_1^{(i)}$ $C_{k-2}^{(i)}$ à $B_{k-1}B_k$

□(S3.5)

Eine Variable heißt nutzlos, wenn es kein Wort w∈ Σ^* gibt mit $A \overset{*}{\to} w$. Sonst heißt sie nützlich
Variable aus der man nichts herleiten kann.

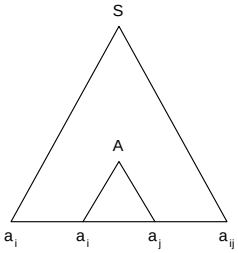
Satz 3.8: G sei kontextfreie Grammatik in Chomsky-Normalform. Man kann die nutzlosen Variablen und alle Regeln, in denen sie auftauchen, ersatzlos streichen, ohne L(G) zu verändern.

Der CYK-Algorithmus (Cocke,Younger,Kasumi)

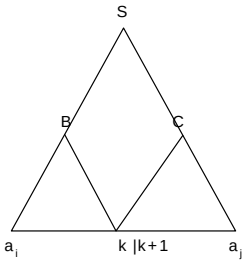
Gegeben ist eine kontextfreie Grammatik: G = (V,∑,P,S) in Chomsky-Normalform und ein Wert w∈ Σ^* w = $a_1 \ldots a_n$.
Frage: w∈ L(G)?

Berechne: $V(i,j) = \{A \mid A \in V, A \overset{*}{\longrightarrow} a_i \ldots a_j\}$
Ist S∈ V(1,n), dann ist w∈ L(G)

$V(i,i) = \{A \mid (A \to a_i) \in P\}$



$V(i,j) = \{A \mid (A \to BC) \in P, B \in V(i,k), C \in V(k+1,j), k \in \{i,\ldots,j-1\}\}$



Satz 3.9: Der CYK-Algorithmus entsteht in Zeit
$O(|P| * |w|^3)$, ob w∈ L(G)
Beweis: „Correct by Construction“
Laufzeit: Tabelle enthält $O(|w|^2)$ Einträge.
Je Eintrag bis zu |w| k-Werte → $O(|w|^3)$ mal die |P| Produktionen durchsuchen.

Korollar 3.10:
L = { <G>w | G ist Kontext-freie Grammatik in Chomsky-Normalform , w∈ L(G) } ∈ P

Das Pumping-Lemma für Kontext-freie Sprachen

Def. 3.11: Sei L eine Kontext-freie Sprache. L hat die Kontextfreie Pumpeigenschaft, falls gilt:

$$\exists n_L \in N \forall z \in L, |z| \geq n_L \;\; \exists u,v,w,x,y : z = uvwxy :$$

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n_L$

(iii) $\forall i \geq 0 : uv^i wx^i y \in L$

Pumping Lemma kann auch benutzt werden um Wort kleiner zu machen, indem Pump teil einfach entfernt wird (Rest ist muss dann auch wieder in Sprache sein)

Bsp 3.12: (a) L1 = {a, ba} hat die kf. PE: Setzte n_{L1} = 3.

(b) $L2 = \{a^n, b^n \mid n \geq 1\}$ hat die kf.PE. Setzte n_{L2} = 4. $z = a^j aabbb^j$, j≥0

4 hat was damit zu tun, dass das Leere Wort nicht in L_2 ist

Setzte **u** = a^ja & **v** = a & **w** = □& **x** = b & **y** = bb^j

(i) |vx| = |ab| = 2 ≥ 1 √

(ii) |vwx| = |ab| = 2 ≤ 4 √

(iii) $\underbrace{a^j}_{u}\underbrace{a}_{v^i}\underbrace{a^i}_{x^i}\underbrace{b^i}_{y}\underbrace{bb^j}_{} = a^{j+i+1}b^{j+i+1} \in L_2$ für alle $i \geq 0$ √

(c) $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ hat die kf.PE nicht.

Annahme: L_3 hat die kf.PE doch. Sei n_{L3} die Konstante

$$\forall n_{L_3} : \exists z, | z | \geq n_{L_3} \;\; \forall u,v,w,x,y; uvwxy = z$$

(i) √

(ii) √

(iii) $\exists i \geq 0 : uv^i wx^i y \notin L$

$z = a^{n_{L_3}} b^{n_{L_3}} c^{n_{L_3}} \in L_3$.

Da $|vwx| \leq n_{L_3}$ besteht vwx aus höchstens 2 verschiedenen Buchstaben → $uv^2 wx^2 y \notin L$