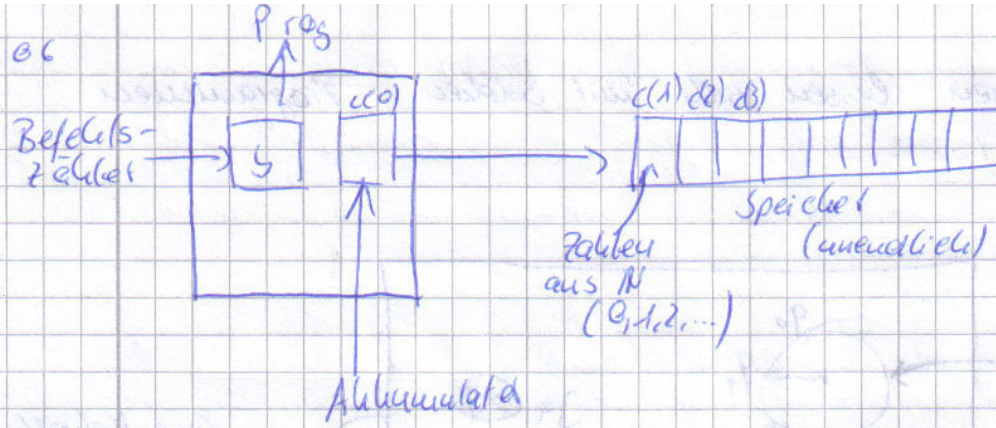
3.05.06



Befehlssatz

LOAD i (i ≙ Adresse der Speicherzelle) $c(0) = c(i)$; $b = b+1$;

STORE i $c_i = c(0)$ $b = b+1$;

ADD i $c(0) = c(0) + i$; $b = b+1$;

SUB i $c(0) = c(0) \dot{-} i$; $b = b+1$;
da $\mathbb{N}$ (3-7???)

$a \dot{-} b$ „a monus b"

$\max\{a-b, 0\}$

MULT i

DIV i $c0 = \left\lfloor \frac{c(0)}{c(i)} \right\rfloor$ unter Gauss-Klammer (nur Nachkomma

GOTO j $b := j$;

IF $c(0)$? l GOTO j ? $\in \{=, \leq, <, \geq, >\}$

END

CLOAD l $c(0) = l$

konstant

CADD l $c(0) := c(0) + l$

indirekte Speicheradressierung:

INDLOAD i $c(0) = c(c(i))$

IND STORE $c(i) = c(c(0))$

IND ADD $c(0) = c(0) + c(c(i))$

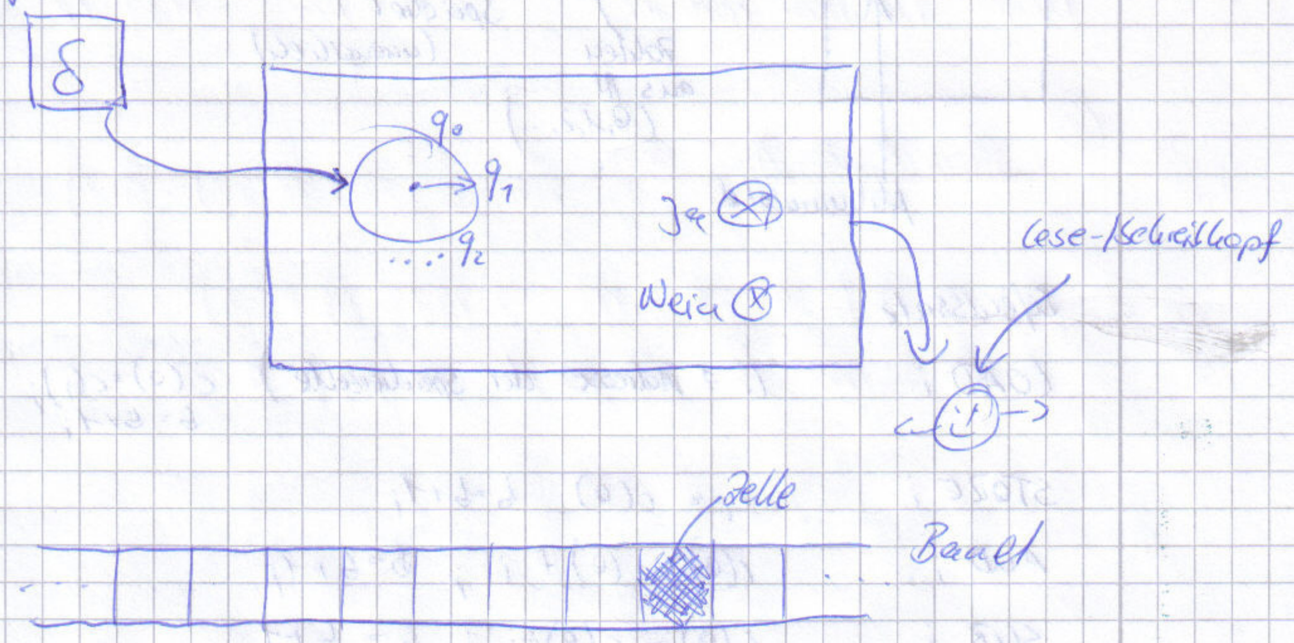
INDSUB → Programmzeilen sind durchnummeriert 1, 2, 3...

INDMULT

INDDIV → Eingabe steht in den Registern 1, 2, ..., k. Rest ist 0

→ Ausgabe steht in den Registern 1, ..., k' (d.h. Ausgabe steht im zusammenhängenden Block)

Alle Algorithmen lassen sich mit solchen Programmen programmieren!



Def.: Eine deterministische 1-Band-Turingmaschine besteht aus

Q: endl. Zustandsmenge, $Q = \{q_0, q_1, q \dots q_n\}$

Σ: endl. Eingabealphabet

Γ: endl. Bandalphabet mit $\Sigma \subseteq \Gamma$

B: $B \in \Gamma \setminus \Sigma$

$q_0 \in Q$ Anfangszustand

$F \subseteq Q$ akzeptierenden Endzustände

Kopfbewegungen

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$

1.) Teste zuerst, ob die Eingabe von der Form $0 \dots 0 1 \dots 1$ ist

2.) Lies die letzte 1 und lösche sie, „merke die gelesene 1 im Zustand" und lösche die erste 0

3.) lies die erste 0, merke sie im Zustand und lösche die letzte 1

Konfiguration K ist ein Element aus $\Gamma^* \times Q \times \Gamma^*$

Bsp.: $B00q_1\, 0111BB$

„TM ist in Konfiguration $\alpha q \beta$": Auf dem Band steht $\alpha\beta$ und der Kopf steht unter dem ersten Zeichen von β und der Zustand ist q.

$$\beta = \overset{\text{Zeichen}}{b}\overset{\text{Rest von }\beta}{\overline{\beta}}, \quad \alpha = \overline{\alpha} a$$

„$\alpha' q' \beta'$ ist direkte Nachfolgekonfiguration von $\alpha q \beta$"
falls gilt: $\delta(q, b) = (q', b', s)$, $s \in \{N, L, R\}$
und
- falls $s = N$: $\alpha' = \alpha$, $\beta' = b'\overline{\beta}$
- falls $s = R$: $\alpha' = \alpha b'$, $\beta' = \overline{\beta}$
- falls $s = L$: $\alpha' = \overline{\alpha}$, $\beta' = a b' \overline{\beta}$

Kurzschreibweise: $\alpha q \beta \vdash \alpha' q' \beta'$

$\alpha' q' \beta'$ ist die i-te Nachfolgekonfiguration, falls gilt:
- $i = 0$: $\alpha q \beta = \alpha' q' \beta'$
- $i > 0$: es gibt Konf., sodass $\alpha q \beta \vdash K_1 \vdash \ldots \vdash K_{i-1} \vdash \alpha' q' \beta'$

Kurzschreibweise: $\alpha q \beta \vdash^i \alpha' q' \beta'$

$\alpha' q' \beta'$ ist eine Nachfolgekonf., falls es ein i gibt, sodass obiges gilt. ($\alpha q \beta \vdash^* \alpha' q' \beta'$) → es gibt ein i

M ist TM, $x \in \Sigma^*$
M akzeptiert $x \in \Sigma^*$, falls es $\alpha, \beta \in \Gamma^*$ und $q \in F$ gibt mit $q_0 x \vdash^* \alpha q \beta$

Bsp.: $q_0\, 00111 \vdash^* BBBBq_1\, BBBB$

Die Sprache L von M ist die Menge aller von M akzeptierten ~~[illegible]~~ $x \in \Sigma^*$.

Sei M eine det. 1-Band-TM, $x \in \Sigma^*$
M akzeptiert x, falls es $\alpha, \beta \in \Gamma^*$ und $q \in F$ gibt mit $q_0 x \vdash^* \alpha q \beta$.

Die Menge aller von M akzeptierten $x \in \Sigma^*$ ist die von M akzeptierte Sprache.

„Stärkeren Begriff": Falls M die Sprache L akzeptiert und für alle $x \in \Sigma^*$ nach endlich vielen Schritten hält, so entscheidet M die Sprache L.

Def. 1.3: • $L \subseteq \Sigma^*$ heißt rekursiv aufzählbar (r.a., r.e.) genau dann, wenn (gdw) es eine det. 1-Band-TM M gibt, die L akzeptiert.
• $L \subseteq \Sigma^*$ heißt entscheidbar (oder: rekursiv) gdw es eine det. 1-Band-TM M gibt, die L entscheidet.

Def. 1.4: Eine det. 1-Band-TM berechnet die partielle Fkt. $f: \Sigma^* \to (\Gamma \setminus \{B\})^*$ mit
• $f(x) = y$, falls $q_0 x \vdash^* q y$ und M hält im Lauf qy für ein $q \in Q$
• $f(x)$ ist nicht definiert, falls M gestartet mit x nicht hält

M berechnet die totale Fkt. $f: \mathbb{N}^r \to \mathbb{N}$, falls $\Sigma = \{0, 1, \#\}$ und für jedes $a_1, \dots a_r \in \mathbb{N}$ gilt
$q_0\, bin(a_1) \# bin(a_2) \# \dots \# bin(a_r) \vdash^* q\, bin(f(a_1, \dots, a_r))$ für ein $q \in Q$.

1.2. Programmiertechniken bzw. Simulation

1.2.1 Endliche Speicher („Im Zustand merken")

$x_1 \dots x_n \in \Sigma^+$, Frage: Ist x_n in $x_2 \dots x_n$ enthalten

$\Gamma = \Sigma \cup \{B\}$, $Q = \{\{q_0\} \times \Sigma\} \cup \{q_0, q_1\}$

Startzustand: q_0 $\quad F = \{q_1\}$

(1) $\delta(q_0, a) = ([q_0, a], a, R)$ für alle $a \in \Sigma$

(2) $\delta([q_0, a], a) = (q_1, a, N)$ für alle $a \in \Sigma$

(3) $\delta([q_0, a], b) = ([q_0, a], b, R)$ für alle $a, b \in \Sigma$, $a \neq b$

1.2.2 Spurtechnik

Bsp. 3 Spuren (k Spuren)

	U	N	I	V	E	
B	C	R	L	A	P	B
	N	U	R	N	B	

$(U, E, N) \in \Gamma$

Def 1.5: Det. k-Band-TM hat k Bänder und k Köpfe

$\delta: Q \times \Gamma^k \to Q \times \Gamma^k \times \{R, N, L\}^k$ (rechts, links)

Eingabe x steht auf Band 1. Arbeitsweise ist analog zur 1-Band-Turingmaschine.

12.05.2006 **Def. 1.6:** M sei eine (k-Band) TM, die für jede Eingabe hält

Zeitkomplexität: $T_M(x) =$ # Schritte (bzgl. Übergänge), die M gest. mit x ausführt

Platzkomplexität: $S_M(x) =$ # versch. Zellen, die M gest. mit x besucht

$T_M(n) = \max\{T_M(x) \mid x \in \Sigma^*, |x| \leq n\}$

$S_M(n) = \max\{S_M(x) \mid x \in \Sigma^*, |x| \leq n\}$

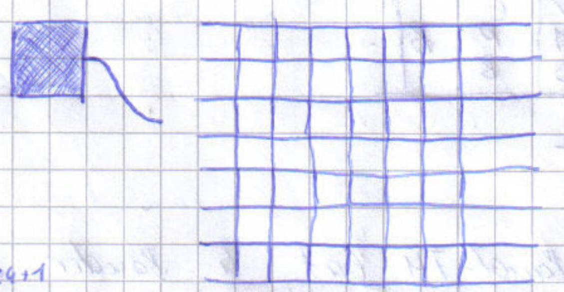
$t, s : \mathbb{N} \to \mathbb{N}$

M ist $t(n)$-zeitbeschränkt und $s(n)$-platzbeschränkt, falls für alle $n \in \mathbb{N}$: $T_M(n) \leq t(n)$, $S_M(n) \leq s(n)$

Satz 1.7: Jede $t(n)$-zeit, $s(n)$-platzbeschränkt k-Band-TM M' kann durch eine $O(t(n) \cdot s(n))$-zeit und $O(s(n))$-platzbeschr. 1-Band-TM M simuliert werden.

Beweis: $M' = (Q', \Sigma', \Gamma', \delta', q_0', F')$ k-Band-TM

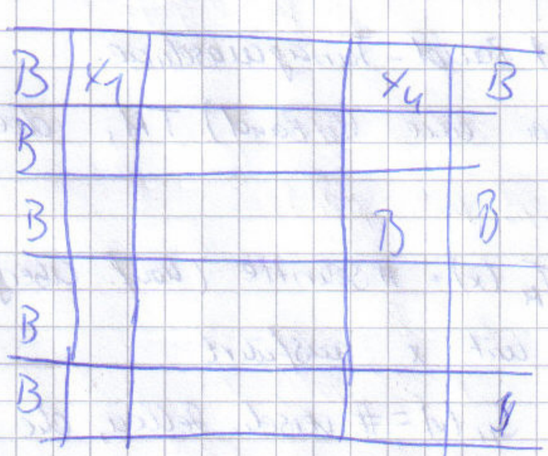
Wir benutzen die Spurtechnik



$\Gamma = \Gamma' \cup \Gamma^{2k+1}$

Platz: $O(s(n))$
Zeit: $O(t(n) \cdot s(n))$

1-Band-TM: Herstellung der Anfangssituation mit den $2k+1$ Spuren



Korollar 18: 1-Band-TM und k-Band-TM liefern die gleichen Mengen von entscheidbaren Sprachen, rekursiv aufzählbaren Sprachen und berechenbaren Fkten.

1.2.3. UNTERPROGRAMME

Wenn wir TM „programmieren", können wir sagen: Wir benutzen hier ein Unterprogramm, dass eine bestimmte Aufgabe löst.

1.3. Simulation zwischen RAMs und TMs

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

$T_M(x)$ = # Schritte von M mit x

$S_M(x)$ = größte genutzte Speicheradresse g. von M - x

„t(n)-zeit" - und „s(n)-platzbeschränkt" ist analog zu TM

17.05.2006 Satz 1.10: Jede RAM kann durch eine det. TM simuliert werden. Ist die RAM t(n)-zeitbeschränkt, ist die TM $O(t(n)^3)$-zeitbeschränkt.

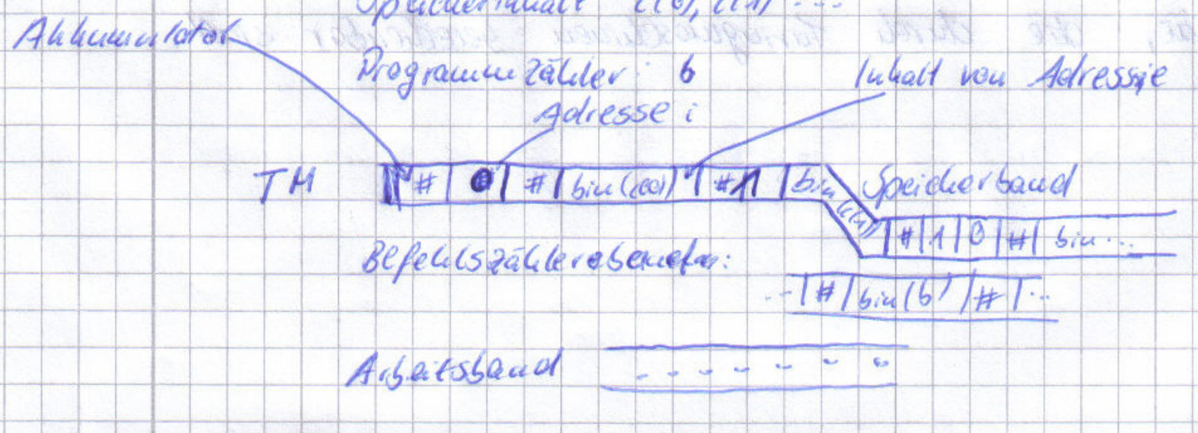
Beweis: (a) Wir müssen überlegen, wie eine Konfiguration der RAM auf einer Mehrband-TM gespeichert wird.

(b) Wir müssen Konfigurationsübergänge der RAM realisieren.

Zu a) c(1) ... c(n) enthält die Eingabe

Konf. einer RAM

Speicherinhalt c(0), c(1) ...



17.05.2006

Nun müssen wir 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 Speicher-Adresse bin(i) auf dem Speicherband (#)

⇒ kopiere bin (c(i)) auf das Arbeitsband

(falls noch nicht auf dem Speicherband: #bin(i)#0# anfügen)

→ „Addiere bin(c(0)) auf bin(c(i))"

$$\begin{array}{rl} 1101 & \text{Speicherband} \\ +\ 101110 & \text{Arbeitsband} \\ \hline 111011 & \text{Arbeitsband} \end{array}$$

→ kopiere den Inhalt des Arbeitsbandes auf Speicherband

→ addiere 1 auf den Befehlszähler □.

Satz 1.11: Jede $t(n)$-zeitbeschränkte 1-Band-TM kann durch eine RAM (Registermaschine) simuliert werden. □

RAM, 1-Band-TM, k-Band-TM, Halbband-TM, 2D-Band-TM beschreiben dieselbe Klasse von „Berechenbarkeit"

1.4. Die Church-Turing-These

Die im intuitiven Sinn berechenbaren Funktionen, sind genau die, die durch Turingmaschinen berechenbar sind.

19.05.06

1.5 Universelle Turingmaschinen

Bislang: Unsere TM können genau Aufgaben lösen.

„special purpose computer"

Darstellung von Turingmaschinen: (1-Band-TM) M

OBdA: $\Gamma = \{0, 1, B\}$, $Q = \{q_1, \dots, q_m\}$

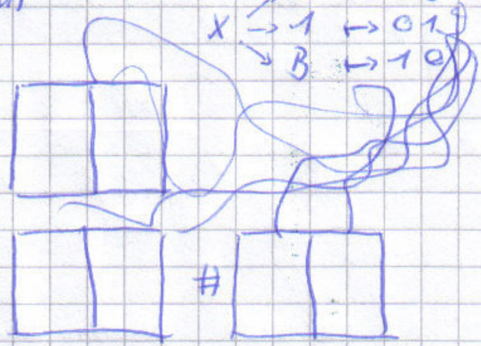
Startzustand: q_1, $F = \{q_m\}$

Darstellen über $\{0, 1, \#\}$

$X \to 0 \mapsto 00$, $1 \mapsto 01$, $B \mapsto 10$

$\delta(q_i, X) = (q_j, Y, D) \mapsto 1^i \#$

$1^i \#$ … $\#$



$\delta(q_3, 1) = (q_2, B, L)$

Endzustand

$\underbrace{111}_{q_3} \# \underbrace{01}_{1} \# \underbrace{11}_{q_2} \# \underbrace{11}_{B} \# \underbrace{11}_{L} = \text{Code}(\delta(q_3, 1) = (q_2, B, L))$

$\#\#\#1^m\#\#$ Code (ersten Eintrag δ-Tabelle) $\#\#$ Code(zweiter Eintrag) $\#\# \dots \#\#$ Code(letzten …) $\#\#\#$

„Gödelnummer" der TM M $(110\dots1)_2 \in \mathbb{N}$

$\langle M \rangle$

Eine TM $\tilde{M}$ heißt universell, wenn sie bei der Eingabe $\langle M \rangle x$, $x \in \{0,1\}^*$, sich verhält wie M gestartet mit x.

Satz 1.12: Es gibt eine universelle 2-Band-TM $\tilde{M}$, die ~~de~~ jede $t(n)$-zeit- und $s(n)$-platzbeschränkte TM M auf Platz $O(s(n))$ in Zeit $O(t(n))$ simuliert, falls M hält.

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

$\tilde{M}$: Auf Band 1 steht $\langle M \rangle x$

Kopiere $\langle M \rangle$ auf Band 2, lösche $\langle M \rangle$ von Band 1.

1.6 Unentscheidbare Probleme

Gibt es Grenzen für das, was Turingmaschinen berechnen können? Wir werden sehen, daß es solche Grenzen gibt, die sehr wichtige Probleme betreffen. Genauer gesagt: Wir werden zeigen, daß Probleme, von denen wir uns wünschten, daß sie lösbar wären, algorithmisch nicht lösbar sind!

Die Gödelnummer einer Turingmaschine M besteht nur aus Buchstaben aus $\{0,1,\#\}$. Durch die Abbildung

$$\begin{array}{ccc} 0 & \mapsto & 00 \\ 1 & \mapsto & 01 \\ \# & \mapsto & 11 \end{array}$$

dieser Zeichen kommen wir sogar nur mit dem Alphabet $\{0,1\}$ aus. Wir können also jede 1-Band-Turingmaschine M eindeutig durch eine Zeichenkette $\langle M\rangle \in \{0,1\}^*$ kodieren. Da der gleiche Kodierungstrick für beliebige Bandalphabete Σ funktioniert, gehen wir im folgenden davon aus, daß für alle Turingmaschinen, die uns begegnen, $\Sigma = \{0,1\}$ ist.

1.13 Definition:
Die Sprache

$$H := \{\langle M\rangle w \mid M \text{ ist eine 1-Band-Turingmaschine, die, gestartet mit } w, \text{ hält}\}$$

ist das (allgemeine) *Halteproblem*.

> Beachten Sie, daß wir hier eine Menge von Zeichenfolgen (ein anderes Wort für „Menge von Zeichenfolgen" ist ja *Sprache*) als *Problem* bezeichnen. Das ist eine Konvention, hinter der steckt, daß wir letztlich das zugehörige *Wortproblem* oder auch, in äquivalenter Bedeutung, *Entscheidungsproblem* meinen. Wir haben also die Menge H definiert und meinen im Hinterkopf für alle $w \in \{0,1\}^*$ die (maschinelle) Beantwortung der Frage „$w \in H$?". Trotzdem: Nur die Menge H ist das Halteproblem. Im übrigen gilt auch für H: $H \subseteq \{0,1\}^*$.

1.14 Satz: (Turing, 1936)
H ist unentscheidbar (eine andere, aber äquivalente Formulierung lautet: *H ist nicht rekursiv*).

Beweis:
Wir nehmen an, daß es eine Turingmaschine M_H gibt, die H entscheidet. D. h. M_H hält auf *jeder* Eingabe x, und man kann am erreichten Zustand erkennen, ob $x \in H$ ist oder nicht!

Diese Annahme werden wir nun auf einen Widerspruch führen.

Wenn es M_H gibt, dann gibt es auch die folgende Turingmaschine M_{schlau}:

Turingmaschine M_{schlau}

(1) die Eingabe sei y

(2) falls $y = \langle M\rangle$, dann

(3) entscheide mittels M_H, ob $\langle M\rangle\langle M\rangle \in H$
(4) falls ja: schreibe nach rechts endlos viele 1en auf das Band
(5) falls nein: bleibe stehen

Die Turingmaschine M_{schlau} können wir explizit hinschreiben („programmieren"), falls uns jemand die δ-Tabelle von M_H gibt (die als „Unterprogramm" aufgerufen würde).
Was passiert, wenn wir M_{schlau} mit der Eingabe $y = \langle M_{\text{schlau}}\rangle$ starten? Nun, es gibt nur zwei Möglichkeiten, nämlich die, daß die Maschine hält („Fall (a)"), und die, daß sie nicht hält („Fall (b)").

(a) Wenn M_{schlau}, gestartet mit $\langle M_{\text{schlau}}\rangle$, hält, heißt das, daß die Abfrage in (3) die Antwort „nein" ergeben hatte, also $\langle M_{\text{schlau}}\rangle\langle M_{\text{schlau}}\rangle \notin H$. D. h. wiederum, daß M_{schlau}, gestartet mit $\langle M_{\text{schlau}}\rangle$, nicht hält, im Widerspruch zum Anfang der Argumentation!

Also kann Fall (a) nicht gelten. Nun betrachten wir Fall (b).

(b) Wenn M_{schlau}, gestartet mit $\langle M_{\text{schlau}}\rangle$, endlos läuft, muß sich die Maschine in Zeile (4) befinden. Dorthin kommt sie nur, wenn die Abfrage in (3) die Antwort „ja" ergeben hatte, also $\langle M_{\text{schlau}}\rangle\langle M_{\text{schlau}}\rangle \in H$. D. h. wiederum, daß M_{schlau} gestartet mit $\langle M_{\text{schlau}}\rangle$ hält, im Widerspruch zum Anfang der Argumentation!

Also kann keiner der beiden Fälle eintreten, wir müssen somit irgendwo von etwas ausgegangen sein, das falsch ist. Unsere gesamte Argumentation hat aber nur eine Schwachstelle, nämlich die Annahme, daß es M_H gibt. M_H gibt es also gar nicht, und als Konsequenz ist H nicht entscheidbar! □

1.15 Satz:
(a) H ist rekursiv aufzählbar.
(b) Das Komplement von H, d. h. $\bar{H} = \{0,1\}^* \setminus H$, ist nicht rekursiv aufzählbar.

Beweis:
(a) Die universelle Turingmaschine M_0 aus Satz 1.14 ist der rekursive Aufzähler von H, d. h.

$$\langle M\rangle w \in H \iff M_0,\ \text{gestartet mit } \langle M\rangle w, \text{ hält}$$

(b) Wäre $\bar{H}$ rekursiv aufzählbar (mittels einer Turingmaschine $\tilde{M}$), könnte man folgende Turingmaschine programmieren:

Entscheider für H

die Eingabe sei $\langle M\rangle w$
führe *gleichzeitig* aus und stoppe, wenn eine stoppt:
– starte mit $\langle M\rangle w$ auf Band 1 die Turingmaschine M_0 , die die Wörter aus H akzeptiert (aus Teil (a))
– starte mit $\langle M\rangle w$ auf Band 2 die Turingmaschine $\tilde{M}$, die die Wörter aus $\bar{H}$ akzeptiert
hält M_0, halte akzeptierend, hält $\tilde{M}$, halte verwerfend.

Diese Turingmaschine hält für jede Eingabe, denn eine der beiden Untermaschinen, M_0 oder $\tilde{M}$, muß ja halten! Also entscheidet diese Turingmaschine das Halteproblem. Folglich (wegen Satz 1.14) gibt es $\tilde{M}$ nicht. $\square$

Mit der schlechten Nachricht der Unentscheidbarkeit des Halteproblems beginnt nun eine ganze Folge von derartigen schlechten Nachrichten.

1.16 Definition:
Die Sprache

$$H_\varepsilon := \{\langle M \rangle \mid M, \text{ gestartet mit leerem Band, hält}\}$$

heißt *spezielles Halteproblem.*

> Hier ist die Namensgebung in den einschlägigen Lehrbüchern nicht ganz eindeutig. Das macht jeder Autor beinahe nach Lust und Laune, aber immer begründet. Manchmal wird H_ε auch als *initiales Halteproblem* bezeichnet oder – etwas phantasielos – als *Halteproblem mit leerem Band.*

1.17 Satz:
H_ε ist nicht entscheidbar.

Beweis:
Wir zeigen, daß man einen Entscheidungsalgorithmus für H programmieren könnte, wenn H_ε entscheidbar wäre. Dazu programmieren wir um die Eingabe zum Halteproblem H so herum, daß wir eine Eingabe des speziellen Halteproblems H_ε bekommen.

Gegeben sei also die Eingabe $\langle M \rangle w$ und damit die Frage „$\langle M \rangle w \in H$?“. Wie kann man diese Frage beantworten, wenn man die Frage „$\langle M' \rangle \in H_\varepsilon$?“ für alle $\langle M' \rangle$ beantworten könnte?

Betrachten Sie zu $\langle M \rangle w$ folgende Turingmaschine:

feste_Maschine$_{\langle M \rangle w}$
(1) die Eingabe sei x
(2) starte M mit w $\quad$ ($*$ das ist kein Tippfehler $*$)
(3) falls $x = \varepsilon$, dann halte.

Nun nehmen wir an, daß wir H_ε mittels der Turingmaschine $\tilde{M}$ entscheiden können. Dann haben wir:

(a) Wenn $\tilde{M}$ bei Eingabe $\langle$feste_Maschine$_{\langle M \rangle w}\rangle$ akzeptierend hält, muß feste_Maschine$_{\langle M \rangle w}$ bis Zeile (3) kommen, was nur möglich ist, wenn M, gestartet mit w, hält. Also ist $\langle M \rangle w \in H$.

(b) Wenn $\tilde{M}$ bei Eingabe $\langle$feste_Maschine$_{\langle M \rangle w}\rangle$ verwerfend hält, kann feste_Maschine$_{\langle M \rangle w}$ nicht bis Zeile (3) kommen, was nur möglich ist, wenn M, gestartet mit w, nicht hält. Also ist $\langle M \rangle w \notin H$.

D. h.: Die Antwort von $\tilde{M}$ auf die Eingabe $\langle$feste_Maschine$_{\langle M \rangle w}\rangle$ ist die Antwort auf die Frage „$\langle M \rangle w \in H$?“!

Und damit kann es $\tilde{M}$ nicht geben, mithin ist H_ε nicht entscheidbar. □

Den Trick, das Halteproblem (oder dann weiter andere unentscheidbare Probleme) durch Drumherumprogrammieren zu maskieren, kann man immer wieder anwenden. Er heißt *Reduktion* und wird im weiteren ausführlich behandelt.

> Eine permanente Quelle für Mißverständnisse bei Newcomern auf diesem Gebiet ist, wer auf wen reduziert wird. Hier im Beweis wurde das Halteproblem H auf das spezielle Halteproblem H_ε reduziert. Im Fall der Unentscheidbarkeit wird also das Problem, von dem man bereits die Unentscheidbarkeit weiß, auf das Problem, von dem man es noch nicht weiß, reduziert. Machen Sie sich bitte unbedingt mit dieser (in der Tat verstehbaren) Sprechweise vertraut. Und: Üben Sie Reduktionen!

1.7 Reduktionen und der Satz von Rice

Bislang haben wir uns bei Turingmaschinen noch gar nicht dafür interessiert, was nach dem Halten auf dem Band steht. Im Vordergrund stand nur, in welchem Zustand sie stoppen, bzw., ob sie überhaupt stoppen. Wenn wir das, was auf dem Band steht, als *Ausgabe* in Abhängigkeit vom Eingabewort bezeichnen, haben wir den Schritt zur Berechnung von Funktionen vollzogen.

Formal können wir damit den Ausdruck „M berechnet eine Funktion f“ wie folgt definieren, wobei die *Berechenbarkeit* eine Eigenschaft von f sein soll, wie z. B. in der Analysis die Monotonie, Stetigkeit oder Differenzierbarkeit sehr wichtige Eigenschaften von Funktionen sind.

1.18 Definition:
Eine Funktion $f : \{0,1\}^* \to \{0,1\}^*$ heißt *berechenbar*, wenn es eine (1-Band-)Turingmaschine 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 dann auf dem Band.
- Ist $f(x)$ nicht definiert, so hält M_f, gestartet mit x, nicht.

Ist f total, d. h. für alle $x \in \{0,1\}^*$ definiert, und berechenbar, so heißt f *total berechenbar* oder auch *total rekursiv*. Diese beiden Begriffe sind synonym.

Insbesondere können wir mit dieser Definition auch noch einmal die Begriffe *entscheidbar* und *rekursiv aufzählbar* beschreiben:

- Eine Sprache L ist genau dann entscheidbar, wenn die sog. *charakteristische* Funktion $\chi_L : \{0,1\}^* \to \{0,1\}$ mit
$$\chi_L(x) = \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$
total berechenbar ist.

- Eine Sprache L ist genau dann rekursiv aufzählbar, wenn die (partielle) Funktion

$$\chi'_L(x) = \begin{cases} 1 & \text{falls } x \in L \\ \text{undef} & \text{falls } x \notin L \end{cases}$$

berechenbar ist.

Sehen wir uns nun den Beweis von Satz 1.17 noch einmal an. Wir können ihn in mehrere Teile gliedern. Was wir dort gemacht haben, ist, daß wir mit Hilfe eines (nicht existierenden) Entscheiders für H_ε einen Entscheider für H programmiert haben. Dabei bekommt der H_ε-Entscheider eine Turingmaschine in Form ihrer Gödelnummer (ihres Programms) als Eingabe, so daß die Antwort des H_ε-Entscheiders direkt (ohne Modifikationen) die Antwort auf die Frage „$\langle M\rangle w \in H$?" ist.
Das wollen wir im folgenden abstrahieren.

1.19 Definition:
Seien L_1 und L_2 Sprachen über dem Alphabet $\{0,1\}$. Eine Reduktion ist eine total berechenbare Funktion $f : \{0,1\}^* \to \{0,1\}^*$ für die gilt:

$$x \in L_1 \iff f(x) \in L_2$$

Wir schreiben dann „$L_1 \leq L_2$" und sagen, „L_1 wird (mittels f) auf L_2 reduziert".

Beachten Sie, daß f total sein muß. Das wird bei Reduktionen von Newcomern gerne übersehen. Eine weitere, sprachliche Fehlerquelle ist die, wer auf wen reduziert wird. Eine Eselsbrücke dafür, sich das zu merken, ist folgendes: Man spricht gegen die Pfeilrichtung „$\leq$". „$\leq$" können Sie (ungenau, aber hilfreich) als „ist leichter als" interpretieren.

Eine kleine Diskussion von Schwierigkeiten beim Verstehen und Anwenden des Reduktionskonzepts finden Sie am Ende dieses Abschnitts auf S. 9.

Die folgende total berechenbare Funktion f ist eine solche *Reduktionsfunktion*, die zeigt, daß $H \leq H_\varepsilon$ ist. $\langle \text{feste\_Maschine}_{\langle M\rangle w}\rangle$ ist dabei die Gödelnummer (das Programm) der im Beweis von Satz 1.17 programmierten Turingmaschine.

$$f(x) = \begin{cases} \langle \text{feste\_Maschine}_{\langle M\rangle w}\rangle & \text{falls } x \text{ von der Form } \langle M\rangle w \\ x & \text{sonst} \end{cases}$$

Ganz wichtig ist hierbei, daß die Eigenschaft „x von der Form $\langle M\rangle w$" entscheidbar ist! Dadurch kann nämlich f durch die folgende Turingmaschine berechnet werden:

H_auf_H_ε_Reduzierer (∗ berechnet die Reduktionsfunktion f ∗)

die Eingabe sei x
falls x von der Form $\langle M\rangle w$ ist: gib $\langle \text{feste\_Maschine}_{\langle M\rangle w}\rangle$ aus
anderenfalls: gib x aus

Die durch die Turingmaschine $H_auf_H_\varepsilon_$Reduzierer berechnete Funktion f ist total und berechenbar. Außerdem gilt

$$x \in H \quad \Rightarrow \quad x = \langle M \rangle w \text{ und } M\text{, gestartet mit } w\text{, hält} \Rightarrow f(x) = \langle \text{feste\_Maschine}_{\langle M \rangle w} \rangle \in H_\varepsilon$$

$$x \notin H \quad \Rightarrow \quad \begin{cases} x = \langle M \rangle w \text{ und } M\text{, gestartet mit } w\text{, hält nicht} \\ \qquad\qquad \Rightarrow f(x) = \langle \text{feste\_Maschine}_{\langle M \rangle w} \rangle \notin H_\varepsilon \\ x \text{ nicht von der Form } \langle M \rangle w \Rightarrow f(x) = x \notin H_\varepsilon \end{cases}$$

also zusammen $\big(x \in H \iff f(x) \in H_\varepsilon\big)$ und damit insgesamt $H \leq H_\varepsilon$.

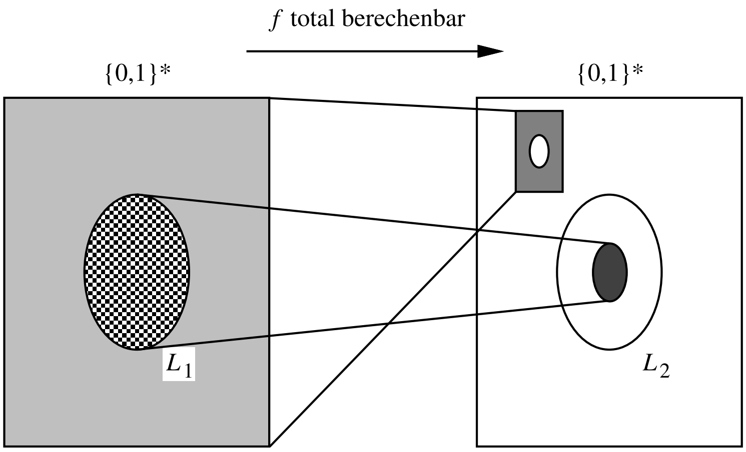


Abbildung 1.1: Wirkungsweise einer Reduktion (das Spiegelei-Bild)

Abbildung 1.1 (unser *Spiegelei-Bild*☺) zeigt, was die Reduktionsfunktion f macht. Für jedes Wort links muß es ein Wort rechts geben (aber nicht unbedingt umgekehrt). Jedes Wort aus L_1 wird dabei in die Menge L_2 abgebildet, ebenso wird jedes Wort aus $\bar{L}_1 = \{0,1\}^* \setminus L_1$ in die Menge $\bar{L}_2$ abgebildet. Die Antworten auf die Fragen „$x \in L_1$?“ und „$f(x) \in L_2$?“ sind deswegen identisch.

Reduktionen an sich kann man auch als Programmieraufgabe interpretieren: Wie würden Sie ein Programm schreiben, das die Sprache L_1 entscheidet, wenn Sie in einer externen Library eine Prozedur finden würden, die L_2 entscheiden kann? Konkret in unserem Beispiel Satz 1.17 kann man „$x \in H$?“ entscheiden, wenn man den Entscheider für H_ε, den man in der Library gefunden hat, auf die Ausgabe von $H_auf_H_\varepsilon_$Reduzierer, gestartet mit x, anwendet.

Im folgenden wird die beschriebene Intuition formaler untersucht, nämlich wie die Spracheigenschaften „entscheidbar/nicht entscheidbar“ und „rekursiv aufzählbar/nicht rekursiv aufzählbar“ durch die Eigenschaft $L_1 \leq L_2$ weitergegeben werden.

1.20 Satz:
Seien L_1 und L_2 Sprachen, und sei $L_1 \leq L_2$ mittels f.

(a) Ist L_2 entscheidbar, dann ist auch L_1 entscheidbar.

(b) *Ist L_2 rekursiv aufzählbar, dann ist es auch L_1.*

Beweis:
Der Beweis ist jetzt noch eine einfache Programmieraufgabe.
(a) Gegeben sei die Frage „$x \in L_1$?“ und eine Entscheidungsturingmaschine M_{L_2} für L_2. Der Entscheidungsalgorithmus für L_1 besteht darin, $f(x)$ zu berechnen und als Eingabe für M_{L_2} zu benutzen. Die Antwort von M_{L_2} auf die Frage „$f(x) \in L_2$?“ ist die Antwort auf die ursprüngliche Frage.
(b) geht analog, nur daß wir jetzt die Antwort „nein“ gar nicht mehr benötigen, sondern uns nur noch für Halten und Nichthalten interessieren. □

Die folgende Konsequenz bekommen Sie direkt aus Satz 1.20, indem Sie die Kontrapositionen der Aussagen bilden[1].

1.21 Korollar:
Seien L_1 und L_2 Sprachen, und sei $L_1 \leq L_2$.

(a) *Ist L_1 unentscheidbar, dann ist auch L_2 unentscheidbar.*

(b) *Ist L_1 nicht rekursiv aufzählbar, dann ist auch L_2 nicht rekursiv aufzählbar.*

Was wir als „Drumherumprogrammieren“ bezeichnet hatten, läßt sich derart verallgemeinern, daß man einen sehr mächtigen Satz beweisen kann, der, etwas ungenau formuliert, besagt, daß alle nichttrivialen semantischen[2] Eigenschaften von Turingmaschinen- und damit Computer-Programmen unentscheidbar sind.
Dazu bezeichnen wir im folgenden mit

$$\mathcal{R} = \{f \mid f : \{0,1\}^* \to \{0,1\}^* \text{ ist berechenbar}\}$$

die Menge der berechenbaren Funktionen.
Zu einer Teilmenge S, $S \subseteq \mathcal{R}$, bezeichne

$$\mathrm{Prog}(S) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion } f \in S\}$$

die Menge *aller* (! ganz wichtig) Programme, d. h. Gödelnummern von Turingmaschinen, die die Funktionen aus S berechnen.
Die Mengen $\mathrm{Prog}(\emptyset)$ und $\mathrm{Prog}(\mathcal{R})$ sind entscheidbar durch einfache Syntaxanalyse, wie wir sie schon kennengelernt hatten. Die Eigenschaften der Wörter dieser beiden Mengen werden als *trivial* bezeichnet. Wir zeigen nun, daß alle anderen Mengen Prog(.) unentscheidbar sind!

1.22 Satz: (Rice, 1953)
Sei S, $S \subseteq \mathcal{R}$, mit $\emptyset \neq S \neq \mathcal{R}$. Dann ist $\mathrm{Prog}(S)$ nicht entscheidbar.

[1] Sie wissen schon, die Aussage $(A \Rightarrow B)$ ist äquivalent zu $(\neg B \Rightarrow \neg A)$

[2] Die Semantik eines Programms ist seine Bedeutung; für jemanden ohne Wissen über Turingmaschinen ist die Gödelnummer einer Turingmaschine lediglich eine nichtssagende 0-1-Folge.

Beweis:
Sei u die für keine Eingabe definierte Funktion. u ist berechenbar im Sinne der Definition 1.18, z. B. durch die (ganz konkrete) Turingmaschine M_u, die sofort in eine Endlosschleife geht. Mit anderen Worten: $u \in \mathcal{R}$ und $\langle M_u \rangle \in \mathrm{Prog}(\mathcal{R})$. Beachten Sie, daß schon $\mathrm{Prog}(\{u\})$ unendlich viele Programme enthält.

Entweder ist $u \notin S$ („Fall (a)“) oder $u \in S$ („Fall (b)“).

Fall (a): $u \notin S$ und damit auch $\langle M_u \rangle \notin \mathrm{Prog}(S)$. Wir zeigen: $H_\varepsilon \leq \mathrm{Prog}(S)$.

Da $S \neq \emptyset$, gibt[3] es eine Funktion $s \in S$, die durch eine (wieder ganz konkrete) (1-Band-)Turingmaschine M_s mit $\langle M_s \rangle \in \mathrm{Prog}(S)$ berechnet werden kann. Nun schreiben wir ein neues Programm:

Drumherum$_{\langle M \rangle}$

die Eingabe sei y
starte M mit leerem Band auf einem Hilfsband
starte M_s mit y

Als Reduktionsfunktion nehmen wir

$$f(x) = \begin{cases} \langle \text{Drumherum}_{\langle M \rangle} \rangle & \text{falls } x = \langle M \rangle \\ \langle M_u \rangle & \text{falls } x \neq \langle M \rangle \quad \text{(der „sonst“-Fall)} \end{cases}$$

f ist total berechenbar, eine Eigenschaft, die eine Reduktionsfunktion zu erfüllen hat, und über die man zumindest nachzudenken hat. Nun gilt

$$\begin{aligned} x \in H_\varepsilon \quad &\Rightarrow \quad x = \langle M \rangle \text{ und } M\text{, gestartet mit } \varepsilon\text{, hält} \\ &\Rightarrow \quad f(x) = \langle \text{Drumherum}_{\langle M \rangle} \rangle \text{ und Drumherum}_{\langle M \rangle} \text{ berechnet } s \in S \\ &\Rightarrow \quad f(x) \in \mathrm{Prog}(S) \end{aligned}$$

$$x \notin H_\varepsilon \quad \Rightarrow \quad \begin{cases} x = \langle M \rangle \text{ und } M\text{, gestartet mit } \varepsilon\text{, hält nicht} \\ \qquad \Rightarrow f(x) = \langle \text{Drumherum}_{\langle M \rangle} \rangle \text{ und Drumherum}_{\langle M \rangle} \text{ berechnet } u \notin S \\ \qquad \Rightarrow f(x) \notin \mathrm{Prog}(S) \\ x \text{ nicht von der Form } \langle M \rangle \Rightarrow f(x) = \langle M_u \rangle \notin \mathrm{Prog}(S) \end{cases}$$

Fall (b): $u \in S$. Wir zeigen diesmal: $\bar{H}_\varepsilon \leq \mathrm{Prog}(S)$.

Interessanterweise können wir die Reduktionsfunktion aus Fall (a) beinahe ganz übernehmen, wir ändern nur den „sonst“-Fall.
Da $S \neq \mathcal{R}$, gibt es eine Funktion $s \in \mathcal{R} \setminus S$, die durch eine (ganz konkrete) (1-Band-)Turingmaschine M_s mit $\langle M_s \rangle \notin \mathrm{Prog}(S)$ berechnet werden kann.

[3] An dieser Stelle wird der Beweis *nichtkonstruktiv*. Niemand berechnet für uns dieses s, es fällt beinahe vom Himmel.

Als Reduktionsfunktion nehmen wir diesmal

$$f(x) = \begin{cases} \langle \text{Drumherum}_{\langle M \rangle} \rangle & \text{falls } x = \langle M \rangle \\ \langle M_s \rangle & \text{falls } x \neq \langle M \rangle \end{cases}$$

Mit dieser total berechenbaren Funktion f zeigen Sie als Fingerübung nun einmal selbst: $x \in \bar{H}_\varepsilon \iff f(x) \in \text{Prog}(S)$ □

Zum besseren Verständnis des Beweis dieses Satzes sollten Sie sich noch einmal das Spiegelei-Bild (Abbildung 1.1) vornehmen und ganz konkret die beiden Fälle des Beweises untersuchen, indem Sie die vorkommenden Turingmaschinen einzeichnen.

Der Satz von Rice hat sehr weitreichende Konsequenzen. In einem konkreten Beispiel angewandt besagt er folgendes: Angenommen, es geht um die Menge *aller* Navigationsprogramme, die in Landkarten den kürzesten Weg zwischen zwei Orten berechnen. Dann gibt es kein Korrektheitsüberprüfungsprogramm, das für jedes mögliche Navigationsprogramm dessen Korrektheit überprüft. Damit wird der schwarze Peter an Sie zurückgegeben: Wenn Sie ein Navigationsprogramm schreiben, müssen Sie das selbst so systematisch machen, daß Sie Ihrem Kunden im Zweifelsfall die Korrektheit Ihres *einen, ganz konkreten* Programms beweisen können.

Machen Sie sich mit der genauen Bedeutung des Satzes von Rice vertraut. Newcomer wenden ihn häufig zu großzügig an. Insbesondere, daß es in ihm um *alle* Programme geht, die die Funktionen in S berechnen, wird leider allzu gern übersehen. Schränkt man die Programme (nicht die Funktionen!), um die es geht, ein, kann das Problem durchaus entscheidbar werden.

Einige Bemerkungen zu den Reduktionen: Wir haben in den obigen Ausführungen immer zwei Richtungen gezeigt, nämlich $(x \in L_1 \Rightarrow f(x) \in L_2)$ und $(x \notin L_1 \Rightarrow f(x) \notin L_2)$. Newcomer schreiben gerne die erste Richtung hin und machen dann einfach aus den „$\Rightarrow$"-Zeichen „$\Leftrightarrow$"-Zeichen. Wenn aber die ausgedachte „Reduktionsfunktion" nicht korrekt ist, kommt der Fehler leider meist in der „$\Leftarrow$"-Richtung vor und bleibt somit unentdeckt. Um auf der sicheren Seite zu sein, sollte man in der Anfangszeit immer beide Richtungen getrennt beweisen.

Ein Hindernis für Newcomer beim Verstehen des Reduktionskonzepts ist die verwirrende(?!?) Vielzahl der vorkommenden Turingmaschinen(-Programme): Es gibt drei Ebenen, auf denen welche auftauchen:

- Ebene 1: „Könnte man L_2 entscheiden, dann auch L_1." Hier wird eine Maschine für L_1 angegeben. Sie taucht im Beweis von Satz 1.20 auf.
- Ebene 2: „Man kann L_1 auf L_2 mittels f reduzieren." Hier wird eine Maschine für die Berechnung der Reduktionsfunktion f programmiert.
- Ebene 3: „$x \in L_1 \iff f(x) \in L_2$." Hier sind die x und $f(x)$ ganz häufig Maschinen.

Hier hilft leider nur die intensive Beschäftigung mit diesen drei Ebenen, um Klarheit in und Vertrautheit mit diesem Gebiet zu bekommen.

1.8 Rekursive Aufzählbarkeit

Bislang wurde die Mengeneigenschaft *rekursiv aufzählbar* einfach nur als abstrakte Bezeichnung benutzt. Im folgenden werden wir wenigstens den Bestandteil *aufzählbar* erklären.

1.23 Satz:
Sei L eine unendliche Sprache. Dann gilt:
L ist rekursiv aufzählbar $\iff$ es gibt eine total berechenbare surjektive Funktion $g : \{0,1\}^* \to L$

Beweis:
„$\Leftarrow$“:
Sei $g : \{0,1\}^* \to L$ eine total berechenbare surjektive Funktion, die von der 1-Band-Turingmaschine M_g berechnet werden möge, und die wir sozusagen wieder in einer Programmlibrary zur Verfügung gestellt bekommen haben. Ziel ist es, eine Turingmaschine M zu programmieren, die, gestartet mit $x \in \{0,1\}^*$, für genau die $x \in L$ hält und dabei M_g als Unterprogramm benutzt.

M arbeitet wie folgt: Sie bekommt ein beliebiges $x \in \{0,1\}^*$ als Eingabe auf Band 1 und sucht nun ein $y \in \{0,1\}^*$ mit $g(y) = x$. Das macht sie, indem sie systematisch nacheinander auf Band 2 die möglichen y-Werte ε, 0, 1, 00, 01, 10, 11, 000, … erzeugt. Dieser mögliche y-Wert wird auf Band 3 kopiert. Auf Band 3 wird M_g nun mit y gestartet. Da g total ist, hält M_g für jede Eingabe. Die Ausgabe von M_g, gestartet mit y, wird nun mit der Eingabe x auf Band 1 verglichen. Bei Gleichheit wird gestoppt, bei Ungleichheit wird der nächste y-Wert ausprobiert. Da g surjektiv ist, gibt es für jedes $x \in L$ ein y mit $g(y) = x$, d. h. ist die Eingabe in L, wird ein solches y auch gefunden. Anderenfalls versandet M in einer Endlosschleife.
Auf einem höheren Level läßt sich M algorithmisch so beschreiben:

> Die Turingmaschine M:
>
> > die Eingabe sei x;
> > $y := \varepsilon$;
> > **while** M_g, gestartet mit y, nicht x ausgegeben hat **do** $y := \text{next}(y)$.

Insgesamt haben wir: M, gestartet mit x, hält $\iff x \in L$.

„$\Rightarrow$“:
Da L rekursiv aufzählbar ist, gibt es eine Turingmaschine M, die auf genau den Eingaben $x \in L$ hält. Wir müssen nun eine Turingmaschine M_g programmieren, die

(i) für *jede* Eingabe hält,
(ii) nur Wörter aus L ausgibt und
(iii) jedes Wort aus L ausgeben kann.

Die von M_g berechnete Funktion ist dann die gesuchte Funktion g.

Was bedeutet es u. a., daß M, gestartet mit x, hält? Es gibt ein $t \in \mathbb{N}$, so daß M, gestartet mit x, nach t Schritten stoppt. D. h. als Eingabe y für M_g werden beliebige Paare (x,t) (die müssen geeignet kodiert werden!) verwendet. M_g führt dann t Schritte von M, gestartet mit x, aus. Wenn M dabei eine Endkonfiguration erreicht, wird x ausgegeben, ansonsten ein festes $x_{\text{fix}} \in L$.

Da M_g ja ein $y \in \{0,1\}^*$ als Eingabe bekommt, müssen wir uns nun überlegen, wie wir daraus ein Paar $(x,t) \in \{0,1\}^* \times \mathbb{N}$ machen. Das geht z. B. wie folgt:

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

aus_eins_mach_zwei ist total berechenbar und surjektiv. Instruktive Beispiele sind:

$$\begin{aligned} &\text{aus\_eins\_mach\_zwei}(011010111) = (01101,3), \\ &\text{aus\_eins\_mach\_zwei}(011) = (\varepsilon,2), \\ &\text{aus\_eins\_mach\_zwei}(1111) = (0,1), \\ &\text{aus\_eins\_mach\_zwei}(0110) = (011,0) \end{aligned}$$

Man sieht, wir können sagen, daß wir in y die rechteste 0 als Komma interpretieren (und den Fall abfangen, daß y gar keine 0 enthält).

Nun ist es ein leichtes, das Programm für M_g anzugeben.

Die Turingmaschine M_g:

die Eingabe sei y;
$(x,t) :=$ aus_eins_mach_zwei(y);
simuliere auf einem Extra-Band t Schritte von M, gestartet mit x;
falls M eine Endkonfiguration erreicht hat, gib x aus, sonst x_{fix}

Ebenso leicht ist es, die Punkte (i) bis (iii) zu überprüfen. □

Eine interessante Programmieraufgabe ist es, die Konstruktion aus dem gerade geführten Beweis zu benutzen, um die Funktion g sogar bijektiv zu machen, d. h. ein Verfahren anzugeben, das für jede Eingabe y ein anderes Wort $x \in L$ ausgibt.

Insgesamt können wir nun ein Programm schreiben, das nacheinander $g(\varepsilon)$, $g(0)$, $g(1)$, $g(00)$, $g(01)$, $g(10)$, $g(11)$, $g(000)$, … hinschreibt, also tatsächlich L aufzählt! Überlegen Sie sich, was das bedeutet: Wir wissen z. B., daß das spezielle Halteproblem H_ε rekursiv aufzählbar ist. D. h. jetzt also, daß es ein Verfahren gibt, das nacheinander alle Programme $\langle M \rangle$ aufschreibt, die, gestartet mit dem leeren Band, halten.

Beachten Sie, daß es im allgemeinen bei der Reihenfolge der Ausgabe keine „schöne“ Ordnung geben kann, wenn man eine rekursiv aufzählbare Sprache L mit einer wie oben konstruierten Funktion g aufzählt, da L ansonsten bereits entscheidbar wäre.

Zum Abschluß dieses Kapitels der Vorlesung wollen wir noch ein paar nicht offensichtliche Eigenschaften rekursiv aufzählbarer Sprachen untersuchen.

Dazu führen wir ein paar Sprechweisen ein:

- Eine Menge $\mathcal{L}$ von Sprachen bezeichnet man als *Sprachklasse* oder auch als *Sprachfamilie*. Z. B. ist $\mathcal{E} = \{L \mid L \text{ ist entscheidbar}\}$ die Klasse der enscheidbaren Sprachen, und $\mathcal{L}_0 = \{L \mid L \text{ ist rekursiv aufzählbar}\}$ die Klasse der rekursiv aufzählbaren Sprachen[4].

- Wenn wir eine k-stellige Operation $\mathrm{op}(.,\ldots,.)$ auf Sprachen haben, also eine Operation, die aus k Sprachen eine neue Sprache macht, dann ist eine Sprachklasse $\mathcal{L}$ genau dann *gegen* op *abgeschlossen* (eine alternative Sprechweise ist: *unter* op *abgeschlossen*), wenn gilt: $L_1,\ldots,L_k \in \mathcal{L} \Rightarrow \mathrm{op}(L_1,\ldots,L_k) \in \mathcal{L}$

  Z. B. kann man sich ganz einfach überlegen, daß die entscheidbaren Sprachen gegen die Vereinigung abgeschlossen sind: $L_1, L_2 \in \mathcal{E} \Rightarrow L_1 \cup L_2 \in \mathcal{E}$. Auch wissen wir aus Satz 1.15, daß die rekursiv aufzählbaren Sprache nicht gegen Komplementbildung abgeschlossen sind, da $H \in \mathcal{L}_0 \Rightarrow \bar{H} \notin \mathcal{L}_0$.

Wir hatten gerade erwähnt, daß es einfach ist zu zeigen, daß die entscheidbaren Sprachen gegen die Vereinigung abgeschlossen sind. Ein Beweis besteht darin, für eine Eingabe x *zuerst* zu überprüfen, ob $x \in L_1$ ist. Wenn nicht, *dann* wird $x \in L_2$ überprüft. Dieses Nacheinanderüberprüfen ist möglich, da die entscheidenden Turingmaschinen ja immer halten.

Eine Schwierigkeit beim Beweis von Abschlußeigenschaften für rekursiv aufzählbare Sprachen besteht darin, daß man dieses Nacheinander nicht machen kann. Stellen Sie sich vor, $x \notin L_1$ und $x \in L_2$. Würde zuerst überprüft, ob x in L_1 ist, würde das Verfahren in eine Endlosschleife laufen, obwohl $x \in L_1 \cup L_2$ ist.

Zum Ziel führen hier zwei mögliche Programmiertechniken:

Zum einen könnte man beide Maschinen gleichzeitig (anderes Wort, das oft benutzt wird: parallel) auf zwei Bändern mit jeweils der Eingabe x laufen lassen. Sobald eine der beiden hält, hält die Maschine für die Vereinigung.

Zum anderen könnte man wie ein Ein-Prozessor-Computer im Multitasking-Betrieb arbeiten: Jede Maschine bekommt ein Zeitfenster, während dessen sie rechnen darf, danach wird die Konfiguration gespeichert, und nun ist die nächste Maschine dran, deren abgespeicherte Konfiguration geladen wird, so daß deren Rechnung für die Dauer des Zeitfensters weitergeführt werden kann. Die Kombination der Maschinen läßt man also kontrolliert laufen. Damit nicht eine Maschine endlos läuft, läßt man jede „ein bißchen“ laufen. Implizit ist dieser Trick auch im Beweis der Richtung „$\Rightarrow$“ von Satz 1.23 zu finden, nämlich in Form der Zeitschranke t.

1.24 Satz:
Seien L_1 und L_2 rekursiv aufzählbar. Dann gilt:

(i) $L_1 \cup L_2$ *ist rekursiv aufzählbar.*

(ii) $L_1 \cap L_2$ *ist rekursiv aufzählbar.*

[4] Warum sie mit $\mathcal{L}_0$ bezeichnet wird, werden wir später noch entdecken.

Beweis:
Nach der Vorstellung des Parallellaufens und des Zeitscheibentricks ist nun klar, wie die Maschinen arbeiten müssen.

Für die Vereinigung reicht es, wenn eine der beiden Maschinen hält, beim Durchschnitt müssen beide halten. □

Im Beweis von Satz 1.15 hatten wir durch Anwendung der parallelen Ausführung von Programmen für H und $\bar{H}$ gezeigt, daß $\bar{H}$ nicht rekursiv aufzählbar ist. Der gleiche Beweis geht natürlich für alle rekursiv aufzählbaren Sprachen.

1.25 Satz:
L ist entscheidbar $\iff$ L und $\bar{L}$ sind rekursiv aufzählbar.

Um für die rekursiv aufzählbaren Sprachen zu zeigen, daß sie gegen kompliziertere Operationen wie die Konkatenation „$\circ$“ (auch Produkt genannt und definiert durch: $L_1 \circ L_2 := \{w \mid \exists u \in L_1 \exists v \in L_2\colon w = uv\}$) abgeschlossen sind, ist der Programmieraufwand zwar größer, aber die beiden genannten Tricks funktionieren noch immer.

Nachdem wir uns bislang mit dem Grenzbereich dessen, was man überhaupt mit Computern berechnen kann (und was nicht!), beschäftigt haben, wollen wir uns nun mit dem auseinandersetzen, was man mit Computern schnell berechnen kann (und was vermutlich leider nicht). Erstaunlicherweise sind auch dabei Turingmaschinen und Reduktionen extrem nützlich. Bei den Reduktionen werden wir im folgenden auch auf die Zeit achten, die man braucht, um die Reduktionsfunktion auszurechnen.

7.06.06

TM $t(n^3)$ ⟷ RAM $t(n)$

„L-Computer"

Entscheidbarkeit: Maschine hält und sagt „ja" oder „nein"

Akzeptanz: Maschine hält für Eingaben aus L, sonst nicht (rekursiv aufzählbar

Halteproblem ist nicht entscheidbar, aber es ist rekursiv aufzählbar

Universelle Turingmaschine

$\boxed{\text{CHURCH-TURING-THESE}}$

Reduktions- $\le_L$ konzept

$H \le H_\varepsilon \le L_2$

Kapitel 2 – Nichtdeterministische TM & das P-NP-Problem

2.1 Def.: Eine nicht-deterministische (1-Band) TM (NTM) wird beschrieben durch 6 Komponenten $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$ deren Bedeutung bis auf δ gleich denen der det. TM sind.

Nun ist $\delta: Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,N,R\})$ Analog: k-Band-TM

Startkonfiguration: $q_0 x$ $x \in \Sigma^*$

$\alpha q a \beta$ z.B. $(q', b, R) \in \delta(q,a)$

möglicher Schritt $\vdash \alpha b q' \beta$

Nichtdeterministische TM M akzeptiert x.

Es gibt eine Rechnung

$q_0 x \vdash^* \alpha q \beta$, $q \in F$

M akzeptiert die Sprache L

$L = \{x \mid M \text{ akzeptiert } x\}$

2-Band NTM $\Gamma = \{0,1,B\}$, $\Sigma = \{0,1\}$, $F = \{q_2\}$

$\delta(q_0,(a,B)) = \{(q_0,(a,a),(R,R)),\ (q_1,(a,a),(N,N))\}$

$\delta(q_1,(a,a)) = \{(q_1,(a,a),(R,L))\}$

$\delta(q_1,(B,B)) = \{(q_2,(B,B),(N,N))\}$ $a \in \{0,1\}$

M akzeptiert $L = \{ww^R \mid w \in \{0,1\}^*\}$

9.06.2006

$NTM \Rightarrow \Gamma = \{0, 1, B(\text{leer})\}$, $\Sigma = \{0,1\}$

Haltezustand

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

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

Rucksackproblem

$$K = \{(\text{Kodierung von})\, a_1, \dots, a_n, b \mid \text{es gibt } \alpha_1, \dots, \alpha_n \in \{0,1\}: \sum_{i=1}^{n} \alpha_i a_i = b\}$$

$\left[\begin{array}{l} \to \text{rate eine 0-1-Folge } \alpha_1, \dots, \alpha_n \\ \to \text{verifiziere } \sum_{i=1}^{n} \alpha_i a_i = b \end{array}\right\}$ NICHTDETERMINISMUS

NTM M

$$\text{Laufzeit } T_M(x) = \begin{cases} \text{Länge einer \underline{kürzesten} akz. Rechnung} \\ \quad\text{falls } M \text{ die Eingabe } x \text{ akzeptiert} \\ 1, \text{ sonst} \end{cases}$$

$$\text{Platz } S_M(x) = \begin{cases} \text{geringste Platzbedarf einer akz. Rechnung} & \text{falls } M, x \text{ akzeptiert} \\ 1 & \text{sonst} \end{cases}$$

Zahlen

$\left.\begin{array}{ll} T_M(n) & S_M(n) \\ \text{zeitbeschränkt} & \text{platzbeschränkt} \end{array}\right\}$ analog wie bei den det. TMs

„Eine nicht-deterministische RAM braucht $O(n)$ Schritte und $O(n)$ Platz für K"

Satz 2.3: 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 $O(s(n) \cdot t(n))$-platzbeschränkt!

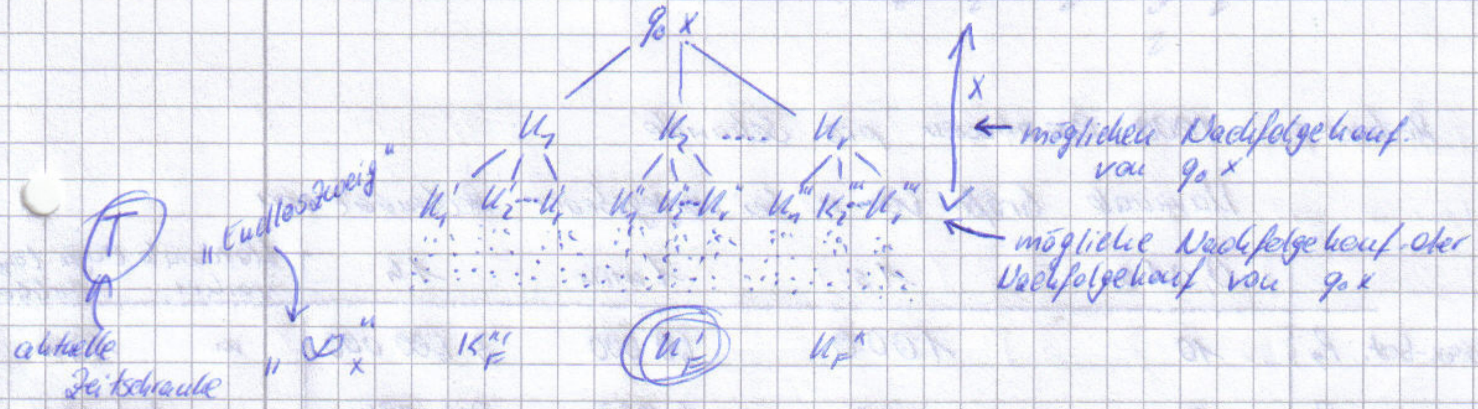
9.06.2006

Beweis: r sei die maximale Zahl an Nachfolgekonfigurationen, die eine Konfiguration für eine Rechnung von M haben kann.

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

Startkonfiguration $q_0 x$

Interpretation der Rechnung als r-ären Baum.



1. Idee Breitensuche auf dem Berechnungsbaum

Platz: $2^{O(t(|x|))} \cdot s(|x|)$ ☹

Zeit: $2^{t(|x|) \cdot \log r} = 2^{O(t(|x|))}$ $(r = 2^{\log r})$

2. Idee: kontrollierte Tiefensuche auf dem Binärbaum

Platz ist „Tiefe $\cdot\, s(n)$"

$T := 2$

while noch keine akzeptierte Rechnung gefunden und der Baum noch nicht abgearbeitet ist.

führe Tiefensuche bis Tiefe T aus.

$T := T + 1$

done

→

9.06.2006

Zeit: $O\left(\sum_{T=2}^{t(n)} 2^{O(T)}\right) = O\left(2^{O(t(n))}\right) = 2^{O(t(n))}$

Korollar 2.4: NTMs akzeptieren genau die rekursiv aufzählenden Sprachen.

Probleme P_1, P_2, P_3, P_4, P_5

n, $n \log n$, n^2, n^3, 2^n # Operationen

Rechner: 1000 Operationen pro Sekunde

Maximale Größe von n bei vorgegebener Rechenzeit von

	0,01 s	1 s	1 min	1 h	Maximale Eingabelänge vorher	nachher
max-Best. P_1	10	1000	60.000	3.600.000	m	10 m
Sortieren P_2	4	140	4.893	204.094	m	fast 10 m
Kantengraph P_3	3	31	244	1897	m	3,16 m
Matrix P_4	2	10	39	153	m	2,15 m
Rucksack P_5	3	9	15	21	m	m+3,3

Hardware 10x schneller!

14.06.06

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 nichtdet. TM, die L akzeptiert}
- $P = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$
- $NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

Offensichtlich: $P \subseteq NP$. Die Frage: „$P = NP$?" oder „$P \neq NP$?"

14.06.06

Vorteile von P:

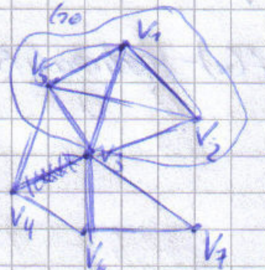
→ robust

- hängt nicht von der Anzahl der Bänder ab
- hängt nicht davon ab, ob man TM oder RAM betrachtet
- Polynome sind gegen Hintereinanderausführung abgeschlossen

$$P_2(P_1(|x|))$$

→ enthält die Probleme, die sich in der Praxis als handhabbar erwiesen haben

→ ermöglicht eine reichhaltige Theorie



- CLIQUE = $\{\langle G, k\rangle \mid G$ ist ein Graph, der einen vollständigen Teilgraphen der Größe k enthält$\}$

  z.B. $\langle G_0, 4\rangle \in$ CLIQUE

  $\langle G_5, 5\rangle \notin$ CLIQUE

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

  ist ein Kreis in einem Graph G, der jeden Knoten genau einmal enthält

Umkreis??! Mail!

  HC $= \{\langle G\rangle \mid G$ enthält den Hamilton-Kreis$\}$

  z.B.: $\langle G_0\rangle \in$ HC

Min. Spannbaum? Fragen! Ausprobieren?

- Travelling Salesperson Problem (TSP)

  TSP $= \{\langle G, c, k\rangle \mid$ Der gewichtete Graph G enthält den Hamilton-Kreis mit Gewicht $\leq k\}$

„günstigste Route"
„wie verlege ich Rohre"

- Knotenüberdeckungsproblem (Vertex Cover)

  Gegeben: Graph $G = (V, E)$. $A \subseteq V$ ist eine Knotenüberdeckung, sodass jede Kante aus E mindestens einen Knoten aus A berührt.

VC $= \{\langle G, k\rangle \mid G$ hat eine Knotenüberdeckung der Größe $k\}$

$\langle G_0, 4\rangle \in$ VC, $\langle G_0, 3\rangle \notin$ VC

<u>Def.: 2.6:</u> Sei L eine Sprache über dem Alphabet $\{0,1\}$. Eine det. TM V_L heißt ein $t(n)$-beschränkter <u>Verifizierer</u> für L, wenn gilt: 16.06.06

Student-Korrektor-Analogie:

(i) die Eingaben von V_L sind von der Form: $x\#\omega$, Abgabe!
$x, \omega \in \{0,1\}^*$

(ii) die Laufzeit ist $O(t(|x|))$;

(iii) für alle $x \in \{0,1\}^*$: $x \in L \Leftrightarrow \exists \omega: |\omega| \le t(|x|)$ und V_L akzeptiert $x\#\omega$

<u>Satz 2.7:</u> Sei L eine Sprache. Dann gilt:

$L \in NTIME(t(n)) \Leftrightarrow$ es gibt einen $t(n)$-beschränkten Verifizierer V_L für L.

<u>Beweis:</u> „$\Leftarrow$“

M: die Eingabe sei x;

rate ω; (* in $|\omega|$ Schritten *)

<u>falls</u> V_L gestartet mit $x\#\omega$ akzeptierend hält,

<u>dann</u> akz. x, <u>sonst</u> verwerfe x;

Offensichtlich akzeptiert M nur Eingaben $x \in L$.
Aus (iii) folgt, dass es ein ω mit $|\omega| \le t(|x|)$ gibt.
V_L akzeptiert $x\#\omega$ in Zeit $\cancel{\le} O(t(|x|))$. Erraten von ω dauert $t(|x|)$ Schritte. M ist $O(t(n))$-zeitbeschränkt.

„$\Rightarrow$“ $L \in NTIME(t(n))$, M sei eine $t(n)$-zeitbeschränkte NTM für L. OBdA (OE, wlog):

Jede Konfiguration einer Rechnung von M hat <u>keine</u> oder <u>genau zwei</u> Nachfolgekonfigurationen!

Bei Eingabe $x\#\omega$ arbeitet V_L wie folgt:
Falls M im i-ten Schritt die Wahl zwischen Konfig. K_0 und K_1 hat, wählt V_L die Konfiguration ω_i.

16.06.06

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

$\Leftrightarrow$ es gibt $\omega \in \{0,1\}^{t(|x|)}$, sodass V_L $x\#\omega$ akzeptiert, da jedes ω eine Rechnung von M gestartet mit ω beschreibt. $\square$

Korollar 2.8: Die Sprachklasse $NP = \{L \mid$ es gibt einen polynomiellen Verifizierer für $L\}$

2.2 NP-VOLLSTÄNDIGKEIT

Def. 2.9: $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$

L_1 ist polynomiell reduzierbar auf L_2 ($L_1 \leq_p L_2$) Laufzeit

genau dann wenn: (i) $L_1 \leq L_2$ (kann reduziert werden) (mittels f)

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

„Hintereinanderausführung von Reduktionen"

Lemma 2.10: „$\leq_p$" ist transitiv.

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

$L_2 \leq_p L_3$ " f_2, – " – $O(n^{k_2})$.

~~Behauptung~~: $L_1 \leq_p L_3$ mittels $f_3(x) = f_2(f_1(x))$. ✓

$\Rightarrow$ ist eine Reduktion der Laufzeit

$O(n^{k_1 k_2})$ $\quad O((n^{k_1})^{k_2})$

$\square$

Def. 2.11:

L heißt NP-schwer (NP-hard, NP-schwierig, NP-hart), wenn gilt:

„ist leichter als"

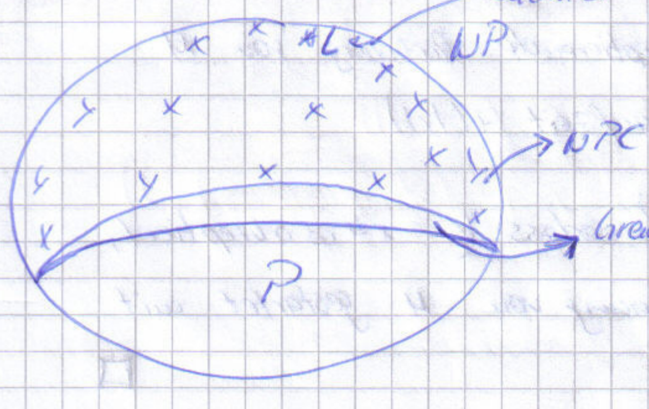
$\forall L' \in NP: L' \leq_p L$

L ist NP-vollständig, wenn gilt:

(i) $L \in NP$ (ii) L ist NP-schwer

leichter Problem, da außen

16.06.06



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!

Die Entdeckung ~~sei~~ eines Polynomzeit ~~A~~ algorithmisch für ein NP-vollständiges Problem hat zur Folge: $P = NP$.

2.3 Der Satz von Cook

21.06.06

Def. 2.13:

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

Literal ist eine Variable oder eine negierte Variable (x bzw. $\bar{x}$). Eine Klausel K der Länge s ist eine Boolscher Ausdruck aus S literalen $K = y_1 \vee y_2 \vee ... \vee y_s$ mit $y_i \in \{x_1, ..., x_n\} \cup \{\bar{x}_1, ..., \bar{x}_n\}$

- Ausdruck in konjunktiven Normalform (KNF) ist ein Boolscher Ausdruck Φ der Form

$$\Phi = K_1 \wedge K_2 \wedge ... \wedge K_r$$

für Klauseln K_i. Die Anzahl der in Φ vorkommenden $\wedge$- und $\vee$-Operationen ist die Größe $size(\Phi)$ von Φ.

21.06.06 $\Phi = (x_1 \vee \bar{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \bar{x}_4)$

$size(\Phi) = 5$

- Eine totale Abbildung $c: V \mapsto \{TRUE, FALSE\}$, die jeder Var. einen Wahrheitswert zuweist, heißt Belegung der Variablen. c wird kanonisch auf die Literale, Klauseln k und die KNF Φ festgesetzt. $c(k)$ bzw. $c(\Phi)$ ist das Ergebnis der <u>Auswertung</u> von k bzw. Φ unter Anwendung von c.

$c(x_1) = c(x_2) = TRUE$, Rest beliebig

$c(\Phi) = TRUE$

- Eine KNF Φ heißt erfüllbar, wenn es eine Belegung c gibt, sodass $c(\Phi) = TRUE$ ist.

„phi" $\leftarrow \Phi$ wird kodiert durch $\langle\Phi\rangle$:

$\langle x_i \rangle = 1 \# bin(i)$ ← Binärdarstellung von i

$\langle \bar{x}_i \rangle = 0 \# bin(i)$

$\langle k \rangle = \langle y_1, \dots, y_s \rangle = \langle y_1 \rangle \#\# \langle y_2 \rangle \#\# \dots \#\# \langle y_s \rangle$

$\langle \Phi \rangle = \langle k_1, \dots, k_n \rangle = \langle k_1 \rangle \#\#\# \langle k_2 \rangle \#\#\# \dots \#\#\# \langle k_n \rangle$

$|\langle\Phi\rangle| = O(size(\Phi) \cdot \log(|V|))$ Kodierung

Das Erfüllbarkeitsproblem (engl. Satisfiability Problem) SAT ist

$$SAT = \{\langle\Phi\rangle \mid \Phi \text{ ist erfüllbare KNF}\}$$

Satz 2.14 (Satz von Cook, 1971)

SAT ist NP-vollständig.

21.06.06

Beweis: (i) SAT $\in NP$

Rate eine Belegung c und überprüfe, ob $c(\Phi) = TRUE$ ist.

„Offensichtlich in Polynomzeit in $|\langle \Phi \rangle|$."

(ii) z.z.: SAT ist NP-schwer, d.h. also $\underline{\forall} L \in NP: L \leq_p SAT$

Sei $L \in NP$ beliebig, aber fest. Sei (M_L eine nichtdeterministische) $M_L = (Q, \Sigma, \Gamma, \delta, q_0, F)$ eine nichtdeterministische 1-Band Turingmaschine der Laufzeit $c \cdot n^k$, die L akzeptiert.

Ziel: Für $w \in L$ muss eine KNF Φ_w konstruiert werden (mit „vielen" Variablen), sodass:

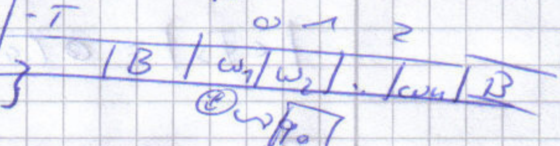
$w \in L \Leftrightarrow$ g.d.w. Φ_w erfüllbar

Soll mögliche Rechnungen von M_L gestartet mit w beschreiben.

23.06.06

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

$\Leftrightarrow q_0 w = K_0 \vdash K_1 \vdash \dots \vdash K_T$ und der Zustand ist q_F



Zelle, Zeichen

$V_{M_L} = \{ zelle_{t,i,a} \mid 0 \leq t \leq T, -T \leq i \leq T, a \in \Gamma \}$
$\cup \{ kopf_{t,i} \mid 0 \leq t \leq T, -T \leq i \leq T \}$
$\cup \{ zustand_{t,q} \mid 0 \leq t \leq T, q \in Q \}$

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

$Zelle_{t,i,a} = TRUE \xLeftrightarrow{g.d.w.}$ in der Berechnung von M_L gestartet mit w in der Konfig K_t steht in der Zelle i das Zeichen a

$kopf_{t,i} = TRUE \Leftrightarrow$ in der Berechnung von M_L gestartet mit w steht in der Konfig K_t der Kopf unter Zelle i.

23.06.06 $Zustand_{t,q} = TRUE \Leftrightarrow$ in der Berechnung von M_L gestartet mit ω ist in der Konfiguration K_t der Zustand q.

genau_eine_Var_ist_true $(x_1, \dots, x_r)$
$= (x_1 \vee \dots \vee x_r) \wedge \bigwedge_{i \neq j} (\bar{x}_i \vee \bar{x}_j)$

000	→ 0
001	→ 1
010	→ 1
011	→ 0
100	→ 1
101	→ 0
110	→ 0
111	→ 0

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

$V_t = \{ zelle_{t,i,a}, kopf_{t,i}, zustand \mid -T \leq i \leq T, a \in \Gamma, q \in Q \}$

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

$=$ genau_eine_Var_ist_true $(zustand_{t,q_0}, \dots, zustand_{t,q_{|Q|}})$

Größe: $O(T^2)$

$\wedge$ genau_ein_Var_ist_true $(kopf_{t,-T}, \dots, kopf_{t,T})$

$\wedge \bigwedge$ genau_eine_Var_ist_true $(zelle_{t,i,a}, \dots, zelle_{t,i,a_{|\Gamma|}})$

$-T \leq i \leq T$

Φ_ω = Rechnung$_\omega (V_0, V_1, \dots, V_T)$ $\quad K_0 + K_1 + \dots + K_T$ am Schluss

$=$ Startkonf$_\omega (V_0) \wedge \bigwedge_{t=0}^{T-1}$ Ein_Schritt $(V_t, V_{t+1}) \wedge zustand_{T,q_F}$

Startkonfiguration$_\omega$ $(V_0) = \bigwedge_{i=0}^{n-1} zelle_{0,i,w_{i+1}} \wedge$ ~~zelle~~ $\bigwedge_{\substack{-T \leq i < 0 \\ n \leq i \leq T}} zelle_{0,i,B}$

$\wedge\, kopf_{0,0} \wedge zustand_{0,q_0}$

Ein_Schritt $(V_t, V_{t+1}) = \bigvee_{\gamma \in \delta}$ Schritt_durch_γ (V_t, V_{t+1})

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

$\delta(q_7, a) = \{(q_3, b, L), (q_5, c, R)\}$

$\wedge$ Schritt_durch_γ $(V_t, V_{t+1}) = Konf(V_t) \wedge Konf(V_{t+1})$

$\wedge\, zustand_{t,q} \wedge zustand_{t+1}$ ~~q~~ $/q'/$

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

(*) $\wedge$ „unter Kopf in K_t steht a und unter Kopf in K_{t+1} steht a'"

$\wedge$ Kopf wird in Richtung D bewegt

23.06.06

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

$D = L:$ $\bigwedge_{i=-T+1}^{T-1} \text{sind-gleich}(\text{kopf}_{t,i-1}, \text{kopf}_{t+1,i-1})$

N ≠:

R:

Größe von Φ_w ist $O(T^3) = O(n^{3k})$

$w \in L$ → es gibt eine akzeptierende Rechnung von M_L gestartet mit w → die liefert eine erfüllbare Belegung von Φ_w

28.06.06

Satz 24: SAT ist NP-vollständig

Beweis: Masterreduktion

L (NP-vollst.) $\leq_p$ L' ?

(i) $L' \in NP$

(ii)

3SAT Φ KNF, in der jede Klausel aus genau 3 Literalen aus 3 verschiedenen Variablen besteht.

$= \{ <\Phi> \mid \Phi \text{ ist erfüllbar} \}$

Offensichtlich: $3SAT \leq SAT$

$((2SAT \leq SAT, \quad 2SAT \in P)$

28.06.06

Th-1

Satz 2.15: 3SAT ist NP-vollständig

Beweis: (i) $3SAT \in NP$ offensichtlich

(ii) $SAT \leq_p 3SAT$

„$\langle \Phi \rangle \in SAT?$" $\iff$ „$f(\langle \Phi \rangle) \in 3SAT?$"

$\Phi = K_1 \wedge K_2 \wedge \ldots \wedge K_T$ ← Klauseln

$K_i = \ell \qquad f(K_i) = (\ell \vee y_{i1} \vee y_{i2}) \wedge$

$(\ell \vee y_{i1} \vee \overline{y_{i2}}) \wedge$

$(\ell \vee \overline{y_{i1}} \vee y_{i2}) \wedge (\ell \vee \overline{y_{i1}} \vee \overline{y_{i2}})$

size $(K_i) = 0$

size $(f(K_i)) = 11$

$K_i = \ell_1 \vee \ell_2 \qquad f(K_i) = (\ell_1 \vee \ell_2 \vee y_{i1})$ Nr. der Klausel

size (1)

$\wedge (\ell_1 \vee \ell_2 \vee \overline{y_i})$

(5)

$K_i = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_k \quad , k > 3 \qquad$ size$(K_i) = k-1$

size$(f(K_i)) = \underline{3k-7}$

$f(K_i) = (\ell_1 \vee \ell_2 \vee y_{i1}) \wedge (\ell_3 \vee \overline{y_{i1}} \vee y_{i2}) \wedge$

$\wedge (\overline{y_{i2}} \vee \ell_4 \vee y_{i3}) \wedge (\overline{y_{i3}} \vee \ell_5 \vee y_{i4}) \wedge \ldots \wedge (\overline{y_{k-4}} \wedge \ell_{k-2} \wedge y_{ik-3})$

$\wedge (\overline{y_{ik-3}} \vee \ell_{k-1} \vee \ell_k)$

30.06.06 Satz 2.16: CLIQUE ist NP-vollständig

Beweis: (i) $CLIQUE \in NP$ ist offensichtlich

(ii) CLIQUE ist NP-schwer

z.z.: $SAT \leq_p CLIQUE$

$\Phi = \bigwedge_{i=1}^{T} K_i$ über den Variablen $\{x_1, \ldots, x_n\}$

$K_i = y_1^{(i)} \vee \ldots \vee y_{s_i}^{(i)}$

30.06.06

$\langle G_\Phi, T \rangle$ $\quad f(\langle \Phi \rangle) = \langle G_\Phi, T \rangle \in CLIQUE$

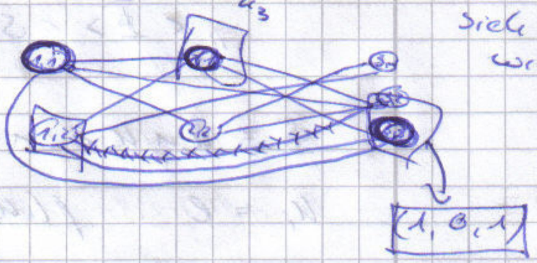
$G_\Phi = (V, E)$

$V = \{(i,j) \mid i \in \{1,\dots,T\}$
$\quad j \in \{1,\dots,S_i\}\}$

$E = \{(i,j),(i',j')\} \mid i \neq i'$ und $y_j^{(i)} \neq \overline{y}_{j'}^{(i')}\}$

$\Phi = \underbrace{(x_1 \vee \overline{x}_2)}_{u_1} \wedge \underbrace{(x \vee \overline{x}_3)}_{u_2} \wedge \underbrace{(\overline{x}_1 \vee x_2 \vee x_3)}_{u_3}$

Die die sich nicht widersprechen



Beh.: Φ erfüllbar $\Leftrightarrow$ G_Φ enthält eine T-Clique.

Beweis: $c = (c_1, \dots, c_n)$ sei eine erfüllende Belegung der Variablen $\{x_1, \dots, x_n\}$ $\quad c = (1,0,1)$

Dann wird in jedem u_i mind. ein Literal $y_j^{(i)}$ erfüllt.

OBdA seien diese $y_1^{(1)}, \dots, y_1^{(T)}$. Dann ist nie $\overline{y}_1^{(i)} = y_1^{(i')}$, $i \neq i'$. Also ist immer $\{(i,1),(i',1)\} \in E$ d.h. $(1,1) \dots (T,1)$ bilden eine T-Clique in G_Φ.

„$\Leftarrow$" Sei $C = \{(i_1, j_1), \dots, (i_T, j_T)\}$ eine T-Clique in G_Φ. Dann ist, wegen des „$i \neq i'$" in der Def. von E, aus jeder Spalte des Graphen genau ein Knoten vorhanden.

Durchnummerieren z

$C = \{(1, \ell_1), (2, \ell_2), \dots, (T, \ell_T)\}$. Seien $y_{\ell_i}^{(i)} = x_{s_i}^{\alpha_i}$

($\alpha_i \in \{$negiert, nicht negiert$\}$)

Es sind alle (s_i, α_i) verschieden, sonst gäbe es i, i' mit $y_{\ell_i}^{(i)} = \overline{y}_{\ell_{i'}}^{(i')}$

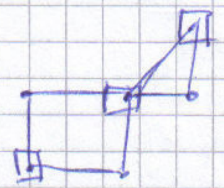
Somit gibt es eine Belegung, die alle $y_{\ell_i}^{(i)}$, $i \in \{1, \dots, T\}$, erfüllt. ~~Aber~~ Also wird dadurch Φ erfüllt, d.h. $\langle \Phi \rangle \in SAT$

$\square$

30.06.06

Th-1

$VC = \{\langle G, k\rangle \mid G$ Graph, $k \in \mathbb{N}$, es gibt k Knoten, sodass jede Kante zu mind. einem dann inzident ist$\}$



[Knotenüberdeckung]

Satz 2.17: VC ist NP-vollständig

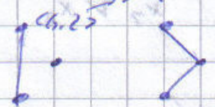
Beweis: (i) $VC \in NP$ ist offensichtlich

(ii) VC ist NP-schwer

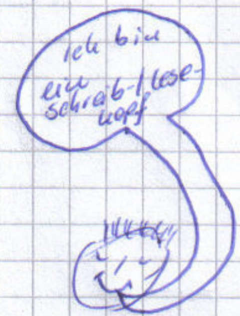
z.z.: $CLIQUE \leq_p VC$

$K(G)$ sei der Komplementgraph von G

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



Behauptung: G enthält k-Clique $\Longleftrightarrow$ $K(G)$ enthält eine Knotenüberdeckung der Größe $n-k$.



$|U| = k$ $\quad K(G),\ |V| - k$

Wähle als Knotenüberdeckung

$V \setminus U$

$\Rightarrow$ Es ist auch tatsächlich eine Knotenüberdeckung, da es Knoten innerhalb von $V \setminus U$ und Kanten nach U gibt, aber keine innerhalb von U.

Binary Programming

$BP = \{\langle A, b\rangle \mid A \in M_{n,m}(\mathbb{Z}),\ b \in \mathbb{Z}^n,$

$\exists\, y \in \{0,1\}^m : A \cdot y \leq b\}$

Satz 2.18: BP ist NP-vollständig

Beweis: (i) $BP \in NP$ ist offensichtlich

(ii) BP ist NP-schwer

22

30.06.06

$SAT \leq_p BP$

$\Phi = u_1 \wedge \ldots \wedge u_r$

$u_i = y_1^{(i)} \vee \ldots \vee y_{s_i}^{(i)}$ $\qquad x_1, \ldots, x_n$

Ungl. über den Variablen $z_1, \ldots, z_n, z_1', \ldots, z_n'$

für jedes x_i: $z_i + z_i' = 1$

für jede u_i: $\sum \tilde{y}_j^{(i)} \geq 1$ $\qquad \tilde{y}_j^{(i)} = \begin{cases} z_\ell & y_j^{(i)} = x_\ell \\ z_\ell' & y_j^{(i)} = \bar{x}_\ell \end{cases}$

Bsp.:

$x_1 \vee \bar{x}_2 \qquad z_1 + z_2' \geq 1$

5.07.06

Aufbauend auf unseren Betrachtungen zur NP-Vollständigkeit:

- Komplexitätstheorie: $\quad$ Co-DSPACE(s(n)) = DSPACE(s(n))
  - Platz: Platzhierarchie
  - Zeit: Zeithierarchie
  - Nichtdeterminismus: P SPACE = NP SPACE
- Approximationsalgorithmen

KAPITEL 3: FORMALE SPRACHEN

Bisher: Erkennen („Automaten, Maschinen")

" Syntaxanalyse

Neu: Erzeugen

Regeln, die mächtig genug sind, um aufwendige Konstrukte zu beschreiben;

Regeln, die einfach genug sind, um eine effiziente Analyse zu erlauben;

5.07.06

Def 3.1: Eine Grammatik G vom Typ Chomsky-0 ist beschrieben durch ein 4-Tupel (V, Σ, P, S)

V ist eine endliche Menge Variablen

Σ ist ein endliches Alphabet von Terminalen

$S \in V$ ist das Startsymbol

unendlich groß

$P \subseteq ((V \cup \Sigma)^+ \setminus \Sigma^* \times (V \cup \Sigma)^*)$ ist endliche Menge

Teilmenge von Produktionen oder Ersetzungsregeln (⇒ nur endlich viele Regeln)

Statt $(u, v) \in P$ schreiben wir: $u \to v$ (u wird durch v ersetzt)

Wir sagen: $\overset{\text{wort}}{\omega'} \in (V \cup \Sigma)^*$ ist aus $\omega \in (V \cup \Sigma)^+$ direkt ableitbar $(\omega \to \omega')$, falls es $(u \to v) \in P$ und $\alpha, \beta \in (V \cup \Sigma)^*$ gibt mit $\omega = \alpha u \beta$ und $\omega' = \alpha v \beta$.

ω' ist aus ω indirekt ableitbar $(\omega \to^* \omega')$, falls ω' durch endl. viele Ableitungsschritte aus ω erzeugbar ist.

Die von G erzeugte Sprache $L(G)$ ist

$$L(G) = \{\omega \in \Sigma^* \mid S \xrightarrow{*} \omega\}$$

Beweisablauf: $L = \{a^n b^n c^n \mid n \geq 1\}$ $\quad L \subseteq L(G) \wedge L(G) \subseteq L$

7.07.06

→ G heißt kontextsensitiv oder vom Typ Chomsky-1, falls für jede Regel ~~gilt~~ $(u \to v) \in P$ gilt: $|u| \leq |v|$

→ G heißt kontextfrei oder vom Typ Chomsky-2, falls für alle Regeln $(u \to v) \in P$ gilt: $u \in V$ (und $v \in (V \cup \Sigma)^*$)

→ G heißt regulär oder vom Typ Chomsky-3, falls alle Regeln von der Art $u \to v$ sind mit $u \in V$ und $u = \varepsilon$ oder $v = a$, $v \in \Sigma$, $v = a\omega$ mit $a \in \Sigma$ und

7.07.06

$\omega \in V \quad (A \to \varepsilon,\ A \to a,\ A \to bC)$

Erweiterung zu „kontextsensitiv":

erlaubt ist: $S \to \varepsilon$, und S taucht nie auf einer rechten Seite einer Produktion auf!

$S \xrightarrow{(1)} aSBC \xrightarrow{(2)} aaBCBC \xrightarrow{(4)} aabBCBC \xrightarrow{(6)} aabC\,BC$ (fertig) $\Rightarrow$ kein Wort $\in L$

$\xrightarrow{(3)} aabBCC \xrightarrow{(5)} aabbCC \xrightarrow{(6)} aabbcC \xrightarrow{(7)} aabbcc \in L(G)$

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

1. Wende $(n-1)$ Mal die Produktion (1) an und 1 Mal Produktion (2):

$S \xrightarrow{*} a^n (BC)^n = \underbrace{a \ldots a}_{n\text{-mal}} \underbrace{BC \ldots BC}_{n\text{-Mal}}$

2. Wende $1+2+\ldots+(n-1) = \frac{n(n-1)}{2}$ Mal die Regel (3) an:

$S \xrightarrow{*} a^n B^n C^n$

3. Wende einmal Regel (4) an und dann $(n-1)$ Mal Regel (5) $\quad S \xrightarrow{*} a^n b^n C^n$

4. Wende einmal Regel (6) an und $(n-1)$ Mal Regel (7).

$S \xrightarrow{*} a^n b^n c^n$

„$\subseteq$" In jeder Regel ist die Anzahl der a's und A's zusammen gleich der Anzahl der b's und B's. Gleiches gilt für die c's und C's.

7.07.06

$\Rightarrow$ In jeder Satzform ist die Anzahl der a, A gleich der Anzahl b, B und gleich der Anzahl der c, C.

~~bbaaBacCc~~

a's können nur durch (1) und (2) erzeugt werden, und daher stehen sie ganz vorne.
Das heißt in einem $w \in L(G)$ stehen a's immer vorne.
b's werden erzeugt durch (4) und (5).
b's folgen daher immer den a's. Gleiches gilt für die c's. In einem $w \in L(G)$ sind die a's vor den kleinen b's; die kleinen b's sind vor den kleinen c's

$\square$

Satz 3.3:

Eine Sprache L ist genau dann rekursiv aufzählbar, wenn es eine Chomsky-0-Grammatik G mit $L(G) = L$ gibt. (L_0 ist die Klasse der rek. Sprachen)

Beweis:

G Gegeben. TM M angegeben, die genau die $w \in L(G)$ akzeptiert, TM-Programm. (Eingabe ist w). Rate eine Ableitung in G und vergleiche das erzeugte Wort mit w.

TM M gegeben: Ziel: Grammatik G angeben mit
$L(G) = \{w \mid M \text{ gest. mit } w \text{ hält}\}$

- Es gibt nur einen Endzustand q_{accept}, und wenn der erreicht ist, ist das Band leer.

7.07.06

- ▶ M arbeitet nur auf den Zellen $i,j>0$
- ▶ M ist nur zu Beginn im Zustand q_0

$q_0 w \vdash^* \triangleright q_{accept}$

$\triangleright q_0 w \vdash u_1 \vdash u_2 \ldots \vdash \triangleright q_{accept}$

Ziel: Grammatik angeben, die diese Rechnung „rückwärts" ausführt.

$V = Q \cup \Gamma \setminus \Sigma \cup \{S\}$: Startsymbol: (q_{accept}) S

$S \to \triangleright q_{accept}$ $\quad q_{accept} \to q_{accept} B$ ← „Blanks"

$\triangleright BBBBBBBB\ldots$

$\boxed{q_{accept}}$

für $\delta(q,a) = (q', a', R)$ 12.07.06

$u = \alpha \boxed{q\ a} \beta \vdash u' = \alpha \boxed{a'q'} \beta$

$\Rightarrow a'q' \to qa$

$\delta(q,a) = (q',a',L)$: $u = \alpha \boxed{b q a} \beta \vdash u' = \alpha \boxed{q' b a'} \beta$

$\forall b \in \Gamma$ $q' b a' \to bqa$

$q_0 a \to a q_0 \ \forall a \in \Sigma$ $\quad S \vdash^* \triangleright q_0 w_1 w_2 w_3 BBB$

$\xrightarrow{*} w_1 w_2 w_3 \in L$

$q_0 B \to q_0$

$a q_0 \to q_0 a \ \forall a \in \Sigma$

$\triangleright q_0 \to \varepsilon$

1. Die Endkonfiguration inkl. B. erzeugen
2. Rechnung vom M rückwärts
3. Aufräumen (B's, q_0 und $\triangleright$ loswerden) □

12.07.06

3.3 Kontextfreie Grammatiken

Eine Sprache heißt kontextfrei, wenn es eine kontextfreie Grammatik G gibt mit $L = L(G)$.

$L_1 = \{a^n b^n \mid n \geq 0\}$ $\qquad G_1: S \to aSb \mid \varepsilon$

G_2: <Ausdruck> → <Term>

<Ausdruck> → <Ausdruck> + <Term>

<Term> → <Faktor> | <Term> · <Faktor>

<Faktor> → a | (<Ausdruck>)

<Ausdruck> → <Ausdruck> + <Term> (Linksableitung: Nimm immer die linkste Variable)

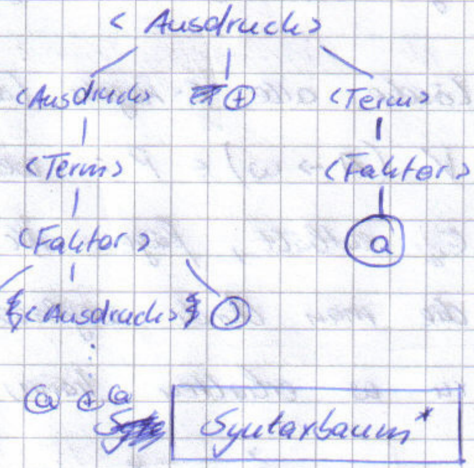
→ <Term> + <Term>

→ <Faktor> + <Term>

→ (<Ausdruck>) + <Term>

→ [...]

„viele Ableitungen" $\overset{*}{\to}$ (a + a) + a



*„Herstellung" des zugehörigen Syntaxbaums ist die Syntaxanalyse

14.07.06

Def 3.4: Eine kontextfreie Grammatiken ist in

(CH-NF) Chomsky-Normalform, wenn jede Regel der Form

$$\boxed{\begin{array}{l} A \to BC \\ \text{oder } A \to a \end{array}} \qquad \begin{array}{l} B, C \in V \setminus \{S\} \\ a \in \Sigma \end{array}$$

ist. Zusätzlich ist $S \to \varepsilon$ erlaubt.

Satz 3.5: In jeder kontextfreie Grammatik G kann eine kf Grammatik G' in CH NF konstruiert werden, sodass $L(G) = L(G')$ ist 14.07.06

Beweis:

Lemma 3.6.: Zu G gibt es eine kontextfreie Grammatik $G_{\varepsilon frei}$ mit $L(G)$ und $G_{\varepsilon frei}$ enthält keine ε-Regeln (bis auf $S \to \varepsilon$)

Beweis Lemma 3.6:

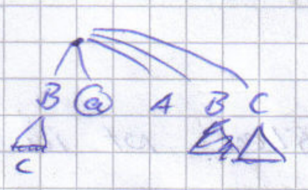
$E_0 = \{A \mid (A \to \varepsilon) \in P\}$

$E_i = \{A \mid A \to B_1 \dots B_k \,;\, B_j \in E_{i-1}\} \cup E_{i-1}$

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

- Lösche alle ε-Regeln (alles für S aber erlaubt)
- $\forall (A \to \omega) \in P$: wenn ω k Variablen aus E_{i_0} enthält, füge alle 2^k-1 möglichen Regeln, die man durch weglassen von Variablen aus E_{i_0} in ω erhalten kann, hinzu, außer $A \to \varepsilon$.

$A \to \check{B} a A B \check{C}$ $\quad E_{i_0} = \{B, C\}$



> $A \to aAbC$

> $A \to BaAb$

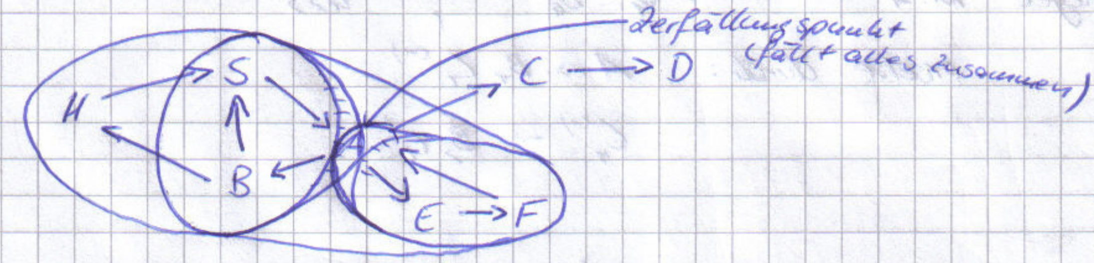
> $A \to aAb$

Kettenregeln (allg. Form): $A \to B$

Lemma 3.7. Zu jeder kontextfreien Grammatik G gibt es eine kontextfreie Grammatik G' ohne Kettenregeln mit $L(G) = L(G')$.

14.07.06

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



$A \to C \ / \ C \to aBa$

$A \to aBa$

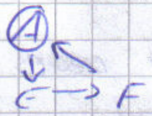
› Ersetze in den starken Zusammenhangskomponenten („diese Sachen mit den Kreisen") die Variablen durch eine davon („S muss übrig bleiben, falls S darin enthalten ist) und ersetze diese in allen Regeln an Stelle der eliminierten.

$A \to BC \ / \ \cancel{C}$

$A \cancel{C} \to XY \ / \ \cancel{X}$ ⇒ das gleiche!

$F \not\to A$

$Z \to aA$



› Ersetze nun im übrig gebliebenen ~~Graphen~~ gerichteten ~~Graphen~~ kreisfreien Graphen „von unten nach oben" (topologische Orientierung) die Kettenregeln $A \to B$ durch die Regeln $A \to$ „alle rechten Seiten von B". (L3.7)

□

Wir konstruieren G' in Ch-NF:

- eliminiere die ε-Regeln
- eliminiere die Kettenregeln
- für alle $a \in \Sigma$: $A_a \to a$ und ersetzen alle

a in den Regeln durch A_a außer wenn dadurch neue Kettenregeln entstehen würden.

Regel Nr. 1 : $A \to B_1 \dots B_k$, $k \geq 3$

ersetze durch: $A \to B_1 C_1^{(1)}$

$C_1^{(1)} \to B_2 C_2^{(1)}$

$\vdots$

$C_{k-2}^{(1)} \to B_{k-1} B_k$ $\square$

$S \to AB \mid CD$

$B \to BC \mid BF$

Eine Variable A heißt nutzlos wenn es kein $w \in \Sigma^*$ gibt mit $A \overset{*}{\Rightarrow} w$. Sonst heißt sie nützlich.

Satz 3.8: G kf. Grammatik in Ch. NF.

Man kann die nutzlosen Variablen und alle Regeln, indem sie auftauchen, ersatzlos streichen, ohne $L(G)$ zu verändern.

Falls $S \in E_{\varepsilon}$: Füge ein neues Startsymbol S' und die Regel $S' \to S$ hinzu. 19.07.06

- Eliminiere die Kettenregeln

DER CYK-ALGORITHMUS

(Cocke / Younger / Kasami)

Gegeben ist eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ in Ch-NF und ein Wort $w \in \Sigma^*$, $w = a_1 a_2 \dots a_n$.

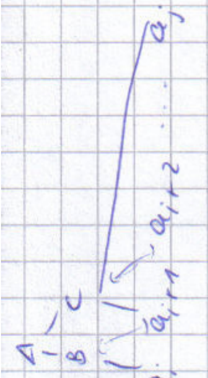
Frage: $w \in L(G)$?

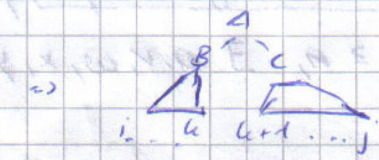
19.07.06

Berechne $V(i,j) = \{A \mid A \in V, A \xrightarrow{*} a_i \dots a_j\}$ $\quad i \le j$

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

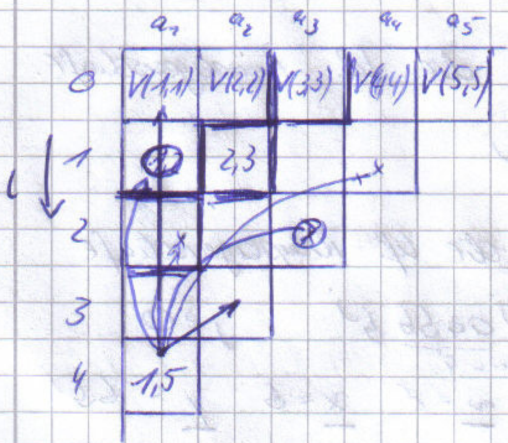
$V(i,j) = \{A \mid (A \to BC) \in P, B \in V(i,k), C \in V(k+1,j); k \in \{i \dots j-1\}\}$



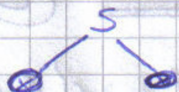


[...]

Konstruktion der $V(i,j)$ nach der Länge $l = j - i$



$w \in L(G) \iff S \in V(1,n)$



„Rückwärts durch die Tabelle laufend kann der Syntaxbaum konstruiert werden."

Satz 3.9: Der CYK-Algo entscheidet in Zeit $O(|P| \cdot |w|^3)$, ob $w \in L(G)$ ist.

Beweis: Korrektheit: "correct by construction"

Laufzeit: für die Berechnung von $V(1,n)$ läuft k von 1 bis $n-1$. Es gibt $O(n^2)$ Felder, also $O(n^3)$ „Schritte". Jeder ~~muss~~ „Schritt" muss die $|P|$ vielen Regeln durchsuchen.

$\Rightarrow$ Gesamtlaufzeit: $O(|P| \cdot n^3)$ $\quad\square$

Korollar 3.10:

$L = \{\langle G\rangle w \mid G \text{ ist kf. Grammatik in ChNF}, w \in L(G)\} \in P$

21.07.06

Das Pumping-Lemma für kf. Sprachen

Def. 3.11: Sei L eine beliebige Sprache.

L hat die kontextfreie Pumpeigenschaft, falls gilt:

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

(i) $|vx| \geq 1$

(ii) $|\cancel{q}vwx| \leq n_L$

(iii) $\forall i \geq 0: uv^iwx^iy \in L$

Bsp.: (a) $L_1 = \{a, ba\}$ hat die kf. Pumpeigenschaft.

Setze $n_{L_1} = 3$!

(b.) $L_2 = \{a^n b^n \mid n \geq 1\}$ hat die kf. Pumpeigenschaft

$n_{L_2} = 4$: $z = a^j \underbrace{aabb}_{\text{min. } 4} b^j \quad j \geq 0$

$u = a^j a \quad v = a \quad w = \varepsilon \quad x = b \quad y = bb^j$

(i) $|vx| = |ab| = 2 \geq 1$ ✓

(ii) $|vwx| = |a\varepsilon b| = 2 \leq 4$ ✓

(iii) $a^j a\, a^i b^i\, bb^j = a^{j+i+1} b^{j+i+1} \in L$

für $i \geq 0$

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

Annahme: L_3 hat die kf. Pumpeigenschaft

Sei n_{L_3} die Konstante aus der Eigenschaft

$\exists n_L \;\forall z \;|z| > n_L \;\exists u, v, w, x, y,\; uvwxy = z$

(i)
(ii)
(iii) $\forall i \geq 0 \;\; uv^iwx^iy \in L$

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

(i)
(ii)
(iii) $\exists i \geq 0: uv^iwx^iy \notin L$

21.07.06 $\Rightarrow z = a^{n_{L3}} b^{n_{L3}} c^{n_{L3}} \in L$

Da $|vwx| \leq n_{L}$, besteht vwx höchstens aus zwei verschiedenen Buchstabensorten. Da $|vx| \geq 1$, kann uv^2wx^2y nicht von allen drei Buchstabensorten gleich viel enthalten.

(d) $L_4 = \{a^{n^2} \mid n \geq 0\}$ hat die kf Pumpeigenschaft nicht!

$|uwy| + i|vx|$ Länge nach Pumpen!

i wächst nicht x^2!

Sei n_{L4} die Konstante

$z = a^{n_{L4}^2}$: $a^{n_{L4}^2 + |vx|}$ müsste Quadratzahl sein!

(d.h. $i = 2$)

$n_{L4}^2 \overset{(i)}{\lneqq} n_{L4}^2 + |vx| \overset{(ii)}{\leq} n_{L4}^2 + n_{L4} \lneqq n_{L4}^2 + 2n_{L4} + 1 = (n_{L4}+1)^2$

ist aber keine Quadratzahl!

$\Rightarrow \notin L_4$

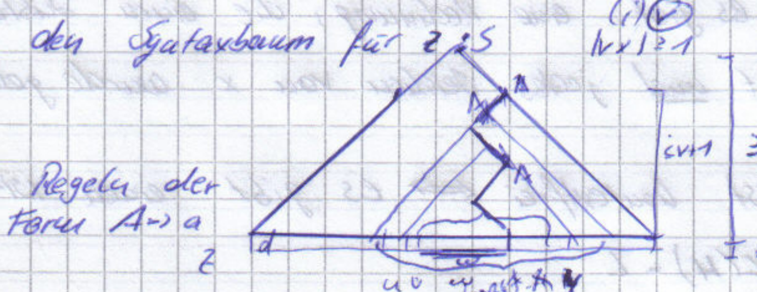
Satz 3.13 (Pumping Lemma für kf. Sprachen)

L ist kf. $\Rightarrow L$ hat die kf. P.E.

[L hat die kf. Pumpeigenschaft nicht $\Rightarrow L$ ist nicht kf.]

Beweis: L ist kontextfrei $\Rightarrow$ es gibt Gramm. $G = (V, \Sigma, P, S)$ in Ch. NF mit $L = L(G)$.

Setze $n_L = 2^{|V|+1}$, Sei $z \in L$ mit $|z| \geq n_L$. Betrachte den Syntaxbaum für z:



$|vwx| \leq n_L$ ✓

Betrachte den längsten Pfad von Wurzel zu Blatt und wähle die Variable A von unten betrachtet, die als erste doppelt vorkommt.

□

Die Chomsky-Hierarchie

Sei L_i, $i \in \{0, 1, 2, 3\}$, die Klassen der Sprachen, die von Grammatiken vom Typ Chomsky-i erzeugt werden.

Satz 3.14: $L_3 \subsetneq L_2 \subsetneq L_1 \subsetneq L_0 \subsetneq$ Rest

Beweis: a^*b^* $\{a^n b^n \mid n \geq 0\}$ $\{a^n b^n c^n \mid n \geq 0\}$ H

Kellerautomaten

Def. 3.15: Ein nichtdet. Kellerautomat ((N)PDA, Pushdown Accepted) ist beschrieben mit 7 Komponenten

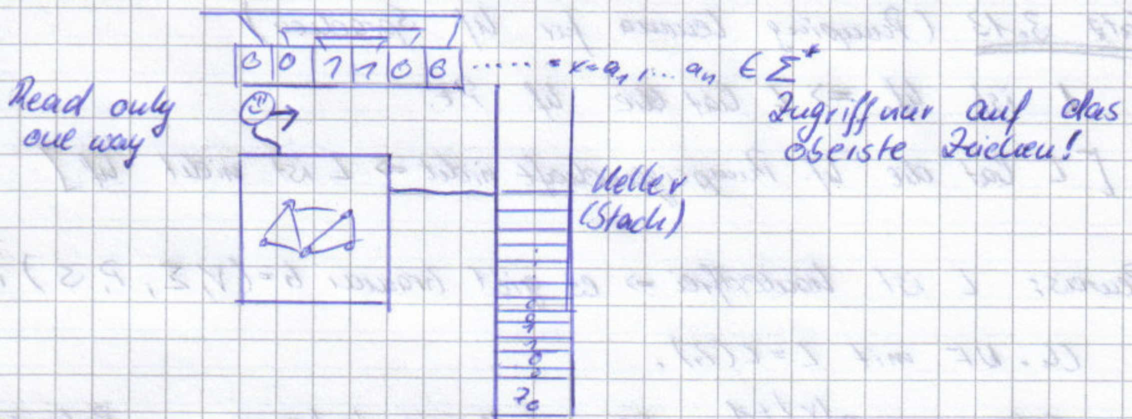
$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

Q Zustände Σ Eingabealphabet

Γ Kelleralphabet

$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times \Gamma^*)$

$z_0 \in \Gamma$ Kellergrundsymbol



$x \in L(M) \Leftrightarrow$ es gibt eine Rechnung, die einen Zustand $q \in F$ erreicht <u>und</u> jedes Zeichen von x wurde gelesen!

Satz 3.16: L ist kontextfrei $\Longleftrightarrow$ es gibt einen NPDA M mit $L(M) = L$

26.07.06

Beweis: „⇐" sehr schwerer Beweis (nachlesbar z.B. im Wegener) benutzt den Nichtdet. auf sehr [illegible] Weise!

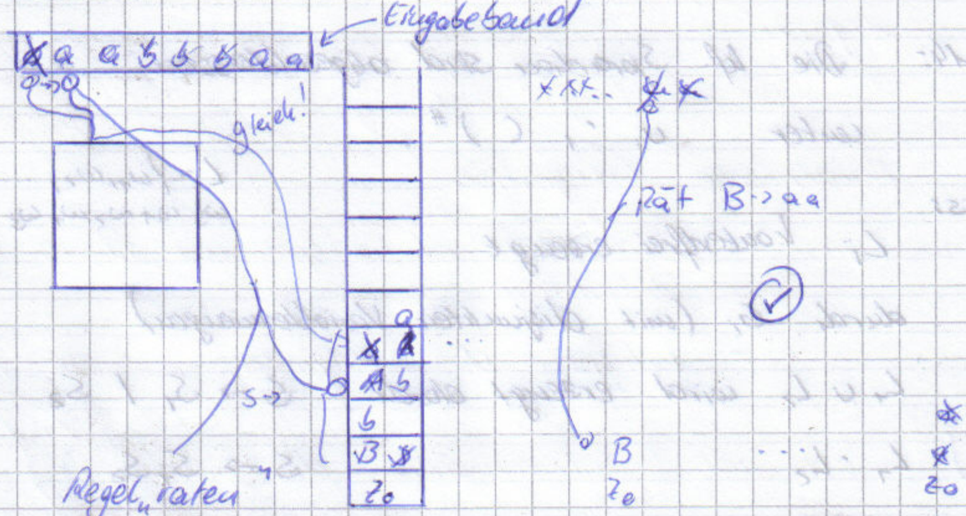
„⇒": Sei $G = (V, \Sigma, S, P)$ eine kf. Grammatik für L. Der NPDA M vollzieht eine Linksableitung für das Eingabeelement x nach.

$S \to \varepsilon \mid aAbB$

$A \to ab \mid aAb$

$B \to aBb \mid aa$

$S \to aAbB$
$\to aaAbbB$
$\to aaabbbB$
$\to aaabbbaa$



~~Satz 3.16~~: $\Gamma = \{z_0\} \cup V \cup \Sigma$

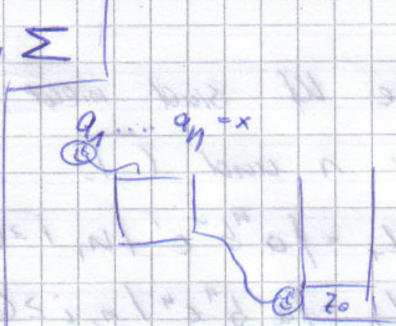
28.07.06

$Q = \{q_0, \bar{q}, q_{accept}\}$

q_0 = Startzustand

z_0

$F = \{q_{accept}\}$



$\delta(q_0, \varepsilon, z_0) = \{(\bar{q}, Sz_0)\}$

$\forall A \in V: \quad \delta(\bar{q}, \varepsilon, A) = \{(\bar{q}, w) \mid (A \to w) \in P\} \qquad w = w_1 \dots w_k$

$\forall a \in \Sigma: \quad \delta(\bar{q}, a, a) = \{(\bar{q}, \varepsilon)\}$

$\delta(\bar{q}, \varepsilon, z_0) = \{(q_{accept}, z_0)\}$

□

28.07.06

Fakt $\mathcal{L}_{DPDA} \subsetneq \mathcal{L}_{NPDA}$

alles was damit geht ... geht damit auch

$\mathcal{L}_{NPDA} \ni \underbrace{\{a^i b^i c^j \mid i,j \geq 0\} \cup \{a^j b^i c^i \mid i,j \geq 0\}}_{L} \notin \mathcal{L}_{DPDA}$

$S \to AB$

$A \to aAb \mid \varepsilon$

$B \to cB \mid \varepsilon$

ABSCHLUSSEIGENSCHAFTEN KF. SPRACHEN

Satz 3.14: Die kf. Sprachen sind abgeschlossen unter $\cup$, $\cdot$, $(\ )^*$.

$L = \{w_1, w_2, \dots\}$
$w_3 w_1 w_2 w_1 w_3 \in L^*$

Beweis:
L_i kontextfrei erzeugt durch G_i (mit disjunkten Variablenmengen)

$L_1 \cup L_2$ wird erzeugt durch $S \to S_1 \mid S_2$

$L_1 \cdot L_2$ $S \to S_1 S_2$

L_1^* $S \to SS_1 \mid \varepsilon$ $\square$

Satz 3.18: Die kf. sind nicht abgeschlossen unter $\cap$ und $\overline{(\ )}$.

Beweis: $L_1 = \{a^n b^n c^i \mid n, i \geq 0\}$ $L_1 \cap L_2 \{a^n b^n c^n \mid n \geq 0\} \notin \mathcal{L}_2$

$L_2 = \{a^i b^n c^n \mid n, i \geq 0\}$

$L_1, L_2 \in \mathcal{L}_2$

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, d.h. wären die kf. Sprachen unter Komplementbildung abgeschlossen, wären sie es auch unter $\cap$.

Bsp: $(a \cup b \cup c)^* \setminus \{a^n b^n c^n \mid n \geq 0\} = \overline{\{a^n b^n c^n \mid n \geq 0\}}$ ist kf.!

28.07.2006 **Satz 3.19.** Sei L eine kontextfreie Sprache und R eine reguläre Sprache. Dann ist $L \cap R$ kontextfrei.

Beweis: Übung $\square$

$$L = \{a, b, c\}^* \cup \{ ☺^i a^n b^n c^n \mid n, i \geq 1\}$$

hat die Pumpeigenschaft!

$$L \cap \underbrace{☺\, a^* b^* c^*}_{R} = \{ ☺\, a^n b^n c^n \mid n \geq 1\}$$

Wäre die kf., dann auch L

Wegen Pumping lemma nicht kontextfrei!

$$\{w \mid \#_a(w) \#_b(w) \#_c(w)\} \cap a^* b^* c^* = \{a^n b^n c^n \mid n \geq 0\}$$