Inhaltsverzeichnis

8. Endliche Automaten, reguläre Sprachen

8.1 Endliche Automaten

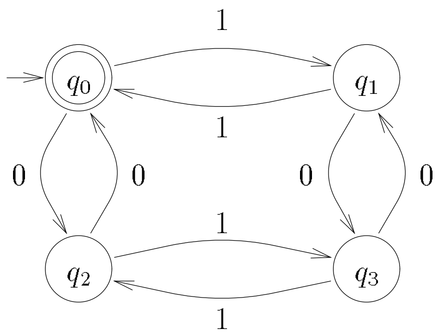
Ein endlicher Automat kann aufgefaßt werden als Turnig Maschine ohne Arbeits- und Ausgabebänder, deren Lesekopf sich nur von links nach rechts bewegt, in jedem Schritt um ein Zeichen, und die keine Randbegrenzer besitzt.

Definition 8.1.1 *Ein deterministischer endlicher Automat A ist ein 5-Tupel* $(Q, \Sigma', \delta, q_0, F)$
mit:

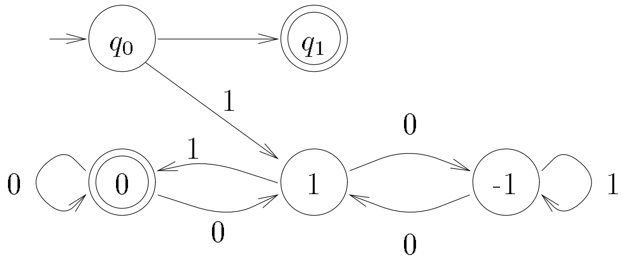
Q	:	*endliche Menge von Zuständen*
Σ'	:	*endliches Eingabealphabet*
$q_0 \in Q$	:	*Anfangszustand*
δ	:	$Q \times \Sigma \to Q$ *Zustansübergangsfunktion*
$F \subseteq Q$	:	*Menge der (akzeptierenden) Endzustände*

Beispiel

Ein endlicher Automat, welcher alle Binärzeichen mit jeweils gerader Anzahl von 0'en und 1'en akzeptiert.



Ein endlicher Automat für alle durch 3 teilbaren Binärzahlen.[1]



Wir erweitern δ kanonisch[2] zu einer Funktion[3]

$$\delta : Q \times \Sigma^* \to Q \quad \text{mittels}$$

$$\delta(q, \epsilon) = q \quad \forall q \in Q$$
$$\delta(q, wa) = \delta(\delta(q, w), a) \quad \forall w \in \Sigma^*, \quad \forall a \in \Sigma', \quad \forall q \in Q$$

Ebenso ist eine Erweiterung auch in 1. Element (Q) möglich.
Erweitere δ zu einer Funktion:

$$\delta : 2^Q \times \Sigma^* \to 2^Q \quad \text{mittels}$$

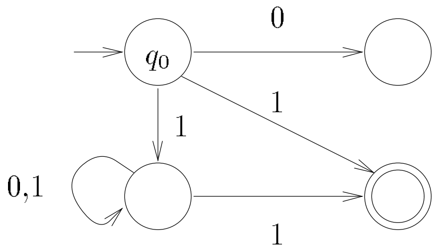
$$\delta(Q', w) = \{\delta(q', w); q' \in Q'\} \quad \forall Q' \in 2^Q, \quad w \in \Sigma^*$$

Also: $w \in L(A) \Leftrightarrow \delta(q_0, w) \in F$ (Akzeptanzbedingung).

Definition 8.1.2 *Ein nichtdeterminisctischer endlicher Automat A ist ein 5-Tupel nach Definition 8.1.1 mit $\delta : Q \times \Sigma \to 2^Q$.*

Beispiel

Nichtdeterministischer endlicher Automat für alle ungeraden Binärzahlen.



Formal: $w \in L(A) \Leftrightarrow \delta(q_0, w) \cap F \neq \emptyset$

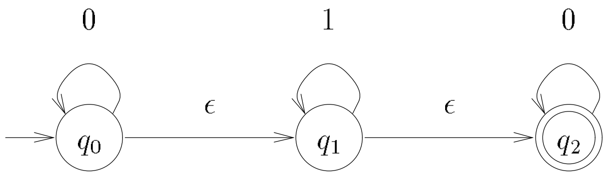
[1] δ ist hier nur partiell; δ kann, indem wir einen Fangzustand einführen, totalisiert werden. (Vgl.: Einführung in die Informatik II)
[2] natürlich
[3] Um nicht eine neue Notation einzuführen, behalten wir die Bezeichnung δ bei.

Definition 8.1.3 *Ein (nicht-deterministischer) endlicher Automat A mit ϵ-Übergängen ist ein 5-Tupel nach Definition 8.1.1 mit:*

$$\delta : Q \times (\Sigma \uplus \{\epsilon\}) \to 2^Q$$

Ein ϵ-Übergang wird ausgeführt, ohne daß ein Eingabezeichen gelesen wird.

Beispiel



Definiere $\bar{\delta}(q, a)$ als die kleinste Teilmenge $Q' = Q'(q, a)$ von Q, die $\delta(q, a)$ enthält und für die gilt:

$$q' \in Q' \Rightarrow \delta(q', \epsilon) \subseteq Q'$$

Erweitere $\bar{\delta}$ wie gehabt.

Satz 1 $w \in L(A) \Leftrightarrow \underbrace{\bar{\delta}(q_0, w)}_{\text{definiert NEA } \epsilon\text{-Übergänge}} \cap F \neq \emptyset$

Beweis als Übung.

Satz 2 *Zu jedem nicht-deterministischen endlichen Automaten* mit *ϵ-Übergängen gibt es einen nicht-deterministischen endlichen Automaten A' ohne ϵ-Übergänge, so daß gilt:*

$$L(A) = L(A')$$

Beweis: Ersetze δ durch $\bar{\delta}$ und $F' = \begin{cases} F & \epsilon \notin L(A) \\ F' \cup \{q_0\} & \epsilon \in L(A) \end{cases}$ ∎

Satz 3 *Zu jedem nicht-deterministischen endlichen Automaten (NEA) gibt es einen deterministischen endlichen Automaten (DEA) A' mit*

$$L(A) = L(A')$$

Beweis:

NEA A:	Q	Σ	δ	q_0	F
DEA A':	2^Q	Σ	δ	$\{q_0\}$	$\mathcal{F}$

wobei $\mathcal{F} = \{Q' \subseteq Q,\, Q' \cap F \neq \emptyset\}$
$\delta_{A'}$ ist die Erweiterung von δ_A auf $2^Q \times \Sigma$

Es gilt:

$$\begin{aligned} w \in L(A) &\Leftrightarrow \delta_A(q_0, w) \cap F \neq \emptyset \\ &\Leftrightarrow \delta_{A'}(q_0, w) \in \mathcal{F} \\ &\Leftrightarrow w \in L(A'). \end{aligned}$$

▮

Der zugehörige Algorithmus zur Überführung eines NEA in einen DEA heißt *Teilmengenkonstruktion, Potenzmengenkonstruktion oder Myhill-Konstruktion.*

8.2 Reguläre Ausdrücke

Definition 8.2.1 *Sei Σ ein endliches Alphabet und $\epsilon \notin \Sigma$. Die Menge der* regulären Ausdrcke *über Σ ist die kleinste Menge $RA(\Sigma)$ von Zeichenreihen über $\Sigma \uplus \{\emptyset, \epsilon, (,), *\}$, für die gilt:*

i) $\forall a \in \Sigma:\ a \in RA(\Sigma)$

ii) $\emptyset, \epsilon \in RA(\Sigma')$

iii) $r, s \in RA(\Sigma) \Rightarrow (r)(s) \in RA(\Sigma)$

iv) $r, s \in RA(\Sigma) \Rightarrow (r + s) \in RA(\Sigma)$

v) $r \in RA(\Sigma) \Rightarrow (r) \in RA(\Sigma)$

vi) $r \in RA(\Sigma) \Rightarrow (r)^* \in RA(\Sigma)$

Bermerkungen

1. Manchmal werden die Klammern weggelassen, solange die Eindeutigkeit nicht darunter leidet.

2. Alternative Schreibweise: $\begin{array}{lcl} (r)|(s) & \equiv & (r+s) \\ (r \cdot s) & \equiv & (r)(s) \end{array}$

Beispiel: $0 + 1(0 + 1)^*$ (hier: Klammern weggelassen)

Reguläre Ausdrücke bezeichnen Sprachen, und zwar wie folgt:
ist r regulärer Ausdruck, dann sei $L(r)$ die Sprache die r bezeichnet

i) $r = a \in \Sigma' \quad L(t) = L(a) = \{a\} \subseteq \Sigma^*$

ii) $\emptyset$ bezeichnet die leere Sprache; ϵ bezeichnet die Sprache $\{\epsilon\}$, die genau das leere Wort enthält.

iii) $L((r)(s)) = L(r)\,L(s)$ wobei $AB =_{\text{def}} \{uv,\; u \in A,\; v \in B\}$

iv) $L((r+s)) = L(r) \cup L(s)$

v) $L((r)) = L(r)$

vi) $L((r)^*) = (L(r))^* = \bigcup_{i \geq 0} (L(r))^i$ (Kleensche Hülle)

dabei:
$$\begin{array}{lcl} A^0 & = & \{\epsilon\} \\ A' & = & A \\ A^{(i+1)} & = & A^i A \quad \forall i \geq 1 \end{array}$$

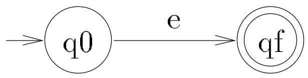
8.3 Reguläre Sprachen

Definition 8.3.1 *Wir bezeichnen die Klasse der durch reguläre Ausdrücke beschriebenen Sprachen als reguläre Sprachen (rationale Sprachen).*

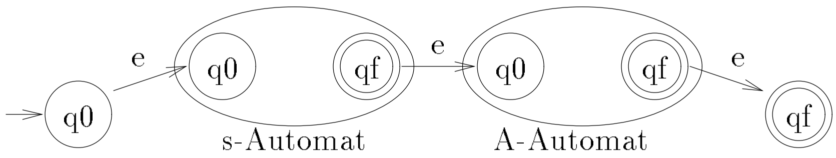
Satz 4 *Für jede reguläre Sprache L gibt es einen deterministischen endlichen Automaten A mit $L = L(A)$.*

Beweis: Induktion über die Struktur des regulären Ausdrucks:

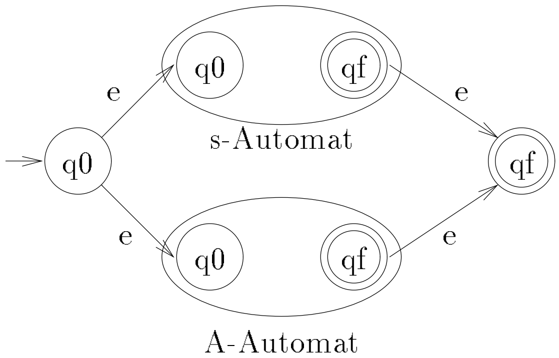
$r = \emptyset$



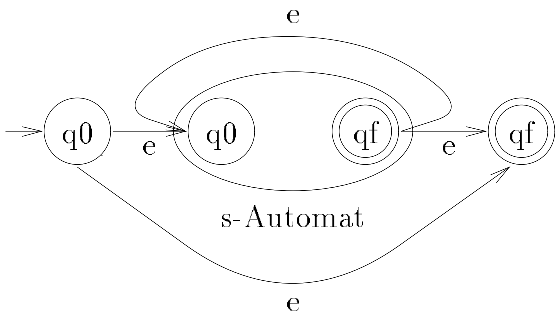
$r = \epsilon$



$r = (st) = (s)(t)$



$r = s + t$



$r = (s)^*$

Betrachte die dazugehörigen deterministischen endlichen Automaten. ∎

Satz 5 *Sei $L = L(A) \subseteq \Sigma^*$, A deterministischer endlicher Automat. Dann ist L regulär.*

Beweis: Sei $A = (Q,\, \Sigma,\, q_1,\, \delta,\, F)$, $\quad Q = \{q_1,\, \ldots,\, q_n\}$. Für $1 \leq i, j \leq n$, $0 \leq k \leq n$ sei

$$R_{i,j}^k = \{x \in \Sigma^* | \delta(q_i, x) = q_j \text{ und } x = yz, y \neq \epsilon \neq z, \delta(q_i, y) = q_l \Rightarrow l \leq k\}$$

Dann gilt:

$$R_{i,j}^0 = \begin{cases} \{a \in \Sigma^* | \delta(q_i, a) = q_j\} & \text{falls } i \neq j \\ \{a \in \Sigma^* | \delta(q_i, a) = q_j\} \cup \{\epsilon\} & \text{falls } i = j \end{cases}$$

$$R_{i,j}^k = R_{i,j}^{k-1} \cup R_{i,k}^{k-1} (R_{k,k}^{k-1})^* R_{k,j}^{k-1}$$

Also:

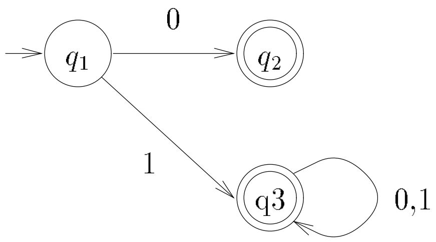
$$\begin{aligned} R_{i,j}^o &= L(a_{i_1} + a_{i_2} + \ldots + a_{i_r}) \text{ bzw.} \\ & \quad L(a_{i_1} + a_{i_2} + \ldots + a_{i_r} + \epsilon) \end{aligned}$$

$$R_{i,j}^k = L(r_{i,k}^{k-1} (r_{k,k}^{k-1})^* r_{k,j}^{k-1} + r_{i,j}^{k-1})$$

$$L(A) = \bigcup_{q_i \in F} R_{1,i}^n = L(\sum_{q_i \in F} r_{1,i}^n)$$

∎

Beispiel:



$$L(A) = L(r_{1,2}^3 + r_{1,3}^3)$$

$$r_{1,2}^3 = r_{1,2}^2 = r_{1,2}^1 = r_{1,2}^0 = 0$$

$$r_{1,3}^3 = r_{1,3}^2 (r_{3,3}^2)^* r_{3,3}^2 + r_{1,3}^2 = \ldots = 1(0+1)^*$$

8.4 Abschlußeigenschaften regulärer Sprachen

Satz 6 *Seien $R_1, R_2 \subseteq \Sigma^*$ reguläre Sprachen. Dann sind auch R_1R_2, $R_1 \cup R_2$, R_1^*, $\Sigma^* \backslash R_1 (=: \bar{R}_1)$, $R_1 \cap R_2$ reguläre Sprachen.*

Beweis: R_1R_2, $R_1 \cup R_2$, R_1^* klar.

$\Sigma^* \backslash R_1$	:	Sei $R_1 = L(A)$, A DEA, $A = (Q, \Sigma, p_0, \delta, F)$ δ vollstänig
		Betrachte $A' = (Q, \Sigma, q_0, \delta, Q \backslash F)$
		Dann $L(A') = \Sigma^* L(A)$
$R_1 \cap R_2$	:	De Morgan

$\blacksquare$

Definition 8.4.1 *Substitution (mit regulären Mengen) ist eine Abbildung, die jedem $a \in \Sigma$ eine reguläre Menge $h(a)$ zuordnet. Diese Abbildung wird kanonisch auf Σ^* erweitert.*

Ein Homomorphismus ist eine Substitution mit $\forall a \in \Sigma : |h(a)| = 1$

Satz 7 *Reguläre Sprachen sind unter (regulärer) Substitution, Homomorphismus und inversem Homomorphismus abgeschlossen.*

Beweis: Inverser Homomorphiosmus: Sei $h : \Delta \to \Sigma^*$ ein Homomorphismus, und sei $R \subseteq \Sigma^*$ regulär.

Zu zeigen: $h^{-1}(R) \subseteq \Delta^*$ ist regulär.

Sei $A = (Q, \Sigma, q_0, \delta, F)$, $L(A) = R$

Betrachte $A' = (Q, \Delta, q_0, \delta', F)$, mit $\delta'(q, a) = \delta(q, h(a)) \quad \forall q \in Q, a \in \Delta$

Also $\delta'(q_0, w) = \delta(q_0, h(w)) \in F \Leftrightarrow h(w) \in R \Leftrightarrow w \in h^{-1}(R)$ $\blacksquare$

Definition 8.4.2 *Seien L_1, $L_2 \subseteq \Sigma^*$. Dann ist der Rechtsquotient $L_1/L_2 := \{x \in \Sigma^* | \exists y \in L_2 : xy \in L_1\}$.*

Satz 8 *Seien R, $L \subseteq \Sigma^*$, R regulär. Dann ist R/L regulär.*

Beweis: A deterministischer endlicher Automat, $A = (Q, \Sigma, q_0, \delta, F)$

$$F' := \{q \in Q | \exists y \in L,\ \delta(q, y) \in F\}$$

$$A' := (Q, \Sigma, q_0, \delta, F')$$

$\blacksquare$

Lemma 8.4.1 *Es gibt einen Algorithmus, der für zwei (nichtdeterministische, mit ϵ-Übergängen) endlichen Automaten A_1 und A_2 entscheidet, ob sie äquivalent sind, d.h. ob $L(A_1) = L(A_2)$.*

Beweis: Konstruiere einen endlichen Automaten für $(L(A_1)\backslash L(A_2)) \cup (L(A_2)\backslash L(A_1))$ (symmetrische Differenz). Prüfe, ob dieser Automat ein Wort akzeptiert. ∎

Satz 9 *(Pumping Lemma für reguläre Sprachen)*

Sei $R \subseteq \Sigma^*$ regulär. Dann gibt es ein $n > 0$, so daß für jedes $z \in R$ mit $|z| \geq n$ gilt:

$$\exists u, v, w \in \Sigma^* : \ z = uvw \text{ mit } |uv| \leq n, \ |v| \geq 1, \ \forall i \geq 0 : \ uv^i w \in R$$

Beweis: $R = L(A)$, $A = (Q, \Sigma, q_0, \delta, F)$

Sei $n = |Q|$. Sei nun $z \in R$ mit $|z| \geq n$

Sei $q_0 = q^{(0)}, q^{(1)}, q^{(2)}, \ldots, q^{|z|}$ die beim Lesen von z durchlaufene Folge von Zuständen. Dann muß es $0 \leq i < j \leq n \leq |z|$ geben mit $q^{(i)} = q^{(j)}$.

Seien nun u die ersten i Zeichen von z, v die nächsten $j - i$ Zeichen und w der Rest.

$$\Rightarrow z = uvw, \ |v| \geq 1, \ |uv| \leq n, \ uv^l w \in R \quad \forall l \geq 0$$

∎

Beispiel für die Anwendung

Satz 10 $\{0^{m^2} | m \geq 0\}$ *ist nicht regulär.*

Beweis: Angenommen, sie sei doch regulär.

$$\text{Wähle } m \geq n, \ 0^{m^2}, \exists r : 1 \leq r \leq n \quad \text{so daß gilt } 0^{m^2 + ir}$$

$$\text{Aber } m^2 < m^2 + r \leq m^2 + m < m^2 + 2m + 1 = (m+1)^2$$

! ∎

Denkaufgabe

$\{a^i b^i | i \geq 0\}$ ist nicht regulär.

Definition 8.4.3 *Sei $L \subseteq \Sigma^*$ eine Sprache. Definiere Relation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ druch $x \equiv_L y \Leftrightarrow \forall z \in \Sigma^* : \ (xz \in L Leftrightarroryz \in L)$*

Lemma 8.4.2 $\equiv_L$ *ist eine rechtsinvariante Äquivalenzrelation.*

rechtsinvariant bedeutet: $x \equiv_L y \Rightarrow xu \equiv_L yu$.

Satz 11 (Myhill-Nerode) *Sei $L \subseteq \Sigma^*$. Dann sind äuivalent:*

i) L ist regulär

ii) $\equiv_L$ hat endlichen Index[4]

iii) L ist die Vereinigung einiger der endlich vielen Äquivalenzklassen von $\equiv_L$.

Beweis: (i)$\Rightarrow$(ii) $L = L(A)$, $A = (Q, \Sigma, q_0, \delta, F)$ DEA.

Dann gilt $\delta(q_0, x) = \delta(q_0, y) \Rightarrow x \equiv_L y$

Also gibt es höchstens so viele Äquivalenzklassen, wie der Automat A Zustände hat.

(ii)$\Rightarrow$(iii) Sei $[x]$ die Äquivalenzklasse von x, $y \in [x] \wedge x \in L \Rightarrow$ nach der Defintion von $\equiv_L$: $y \in L$

(iii)$Rightarrow$(i) Definiere $A' = (Q', \Sigma, q_0', \delta', F')$ mit

$$
\begin{array}{l}
Q' := \{[x] | x \in \Sigma^*\} \quad \text{(n.b. } Q' \text{ endlich)} \\
q_0' := [\epsilon] \\
\delta'([x], a) := [xa] \quad \forall x \in \Sigma^*, a \in \Sigma \quad \text{(konsisten)} \\
F' := \{[x] | x \in L\}
\end{array}
$$

Dann gilt: $L(A') = L$ ∎

8.5 Konstruktion minimaler endlicher Automaten

Satz 12 *Der nach den Satz von Myhill-Nerode konstruierte deterministische endliche Automat hat unter allen DEA's für L eine minimale Anzahl von Zuständen.*

Beweis: Sei $A = (Q, \Sigma, q_0, \delta, F)$, $L(A) = l$. Dann liefert $x \equiv_A z :\Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$ eine Äquivalenzrelation, die $\equiv_L$ verfeinert. $\Rightarrow |Q| = \text{index}(\equiv_L) \geq \text{index}(\equiv_L) =$ Anzahl der Zustände der Myhill-Nerode-Automaten. ∎

Algorithmus zur Konstruktion eines minimalen endlichen Automaten

Eingabe: $A(Q, \Sigma, q_0, \delta, F)$ DEA $\quad (L = L(A))$

Ausgabe: Äquivalenzrelation auf Q.

0. Entferne aus Q alle überflüssigen, ds heist all nicht von q_0 aus erreichbaren, Zustände. Wir nehmen nun an, daß Q keine überflüssigen Zustände mehr enthält.

[4] Anzahl der Äquivalenzklassen

1. Markiere all unmarkierten Paare $\{q_i, q_j\} \in Q^2$ mit $q_i \in F$ und $q_j \notin F$ bzw. $q_i \notin F$ und $q_j \in F$

2. **for** alle unmarkierten Paare $\{q_i, q_j\} \in Q^2$, $q_i \neq q_j$ **do**
       if $(\forall a \in \Sigma)[\{\delta(q_i, a), \delta(q_j, a)\}$ ist markiert$]$ **then**
           markiere $\{q_i, q_j\}$;
           markiere alle $\{q, q'\}$ in $\{q_i, q_j\}$'s Liste und
           rekursiv alle Paare in der Liste von $\{q, q'\}$ usw.
       else
           for alle $a \in \Sigma$ **do**
               if $\delta(q_i, a) \neq \delta(q_j, a)$ **then**
                   trage $\{q_i, q_j\}$ in die Liste von $\{\delta(q_i, a), \delta(q_j, a)\}$ ein
               fi
           od
       fi
   od

3. Ausgabe: q äquivalent zu $q' \Leftrightarrow \{q, q'\}$ nicht markiert.

Satz 13 *Obiger Algorithmus liefert einen minimalen deterministischen endlichen Automaten für $L(A)$.*

Beweis: Sei $A' = (Q', \Sigma', q_0', \delta', F')$ der konstruierte Äquivalenzklassenautomat.

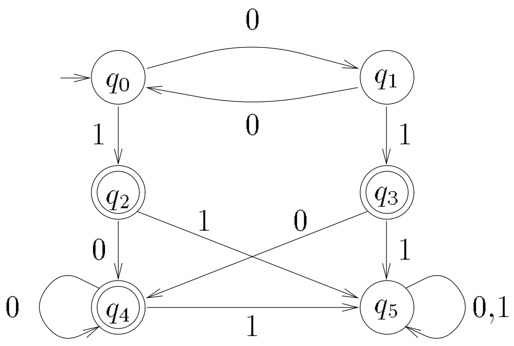
Offensichtlich ist $L(A) = L(A')$.

Es gilt: $\{q, q'\}$ wird markiert gdw. $(\exists\, z \in \Sigma^*)[\delta(q, z) \in F \wedge \delta(q', z) \notin F$ oder umgekehrt$]$, wie man durch einfach Induktion über $|z|$ sieht.

Also: Die Anzahl der Zustände von $A'\,(= |Q'|)$ ist gleich dem Index von $\equiv_L$. ∎

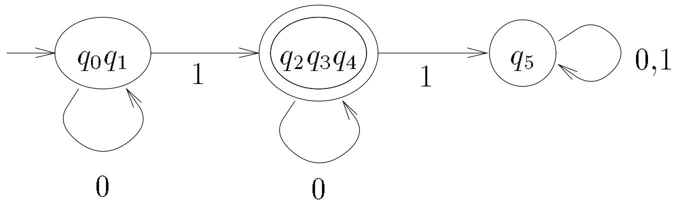
Beispiel:

A:



	q_0	q_1	q_2	q_3	q_4	q_5
q_0						
q_1						
q_2	x_1	x_1				
q_3	x_1	x_1	$(1,0)$			
q_4	x_1	x_1				
q_5	x_2	x_2	x_1	x_1	x_1	

Ergebnis:



$$L(A) = 0^*10^*$$

Satz 14 *Sei $A = (Q, \Sigma, q_0, \delta, F)$ DEA. Der Zeitaufwand des obigen Minimalisierungsalgorithmus ist $O(|Q|^2|\Sigma|)$*

9. Grammatiken, Chomsky-Hierarchie

9.1 Phrasenstrukturgrammatiken

[1] Grammatiken haben:

(i) Terminalalphabet T (oder Σ), $|T| < \infty$

(ii) einen endlichenVorrat von Nichtterminalzeichen (Variablen) V mit $V \cap T = \emptyset$

(iii) Startsymbol $S \in V$

(iv) eine endliche Menge P von Produktionen (Ableitungsregeln) der Form $l \to r$ mit $l \in (V \cup T)^+$, $r \in (V \cup T)^*$

Sei Γ Grammatik, $z, x, y \in (V \cup T)^*$.

Definition 9.1.1 *(i)* $z \underset{\Gamma}{\to} z' \Leftrightarrow z = xly,\ z' = xry,\ l \to r \in P$

(ii) $z \overset{*}{\underset{\Gamma}{\to}} z' \Leftrightarrow z = z'$ *oder* $z \underset{\Gamma}{\to} z^{(1)} \underset{\Gamma}{\to} z^{(2)} \underset{\Gamma}{\to} \dots \underset{\Gamma}{\to} z'$

(iii) Die von Γ erzeugte Sprache ist $L(\Gamma) = \{z \in T^* | S \overset{*}{\underset{\Gamma}{\to}} z\}$

Beispiel

$S \to (S) + (S) | (S) * (S) | a | a + a | a * a$

Definition 9.1.2 *(i) Grammatiken wie oben (ohne weitere Einschränkungen) heißen vom Typ Chomsky-0*

(ii) Falls alle Produktionen die Form

$$S \to \epsilon \text{ oder } l \to r \text{ mit } l \in V^+,\ r \in (V \cup T \backslash \{S\})^*,\ |l| \le |r|$$

haben, heißt die Grammatik kontextsensitiv *(auch monoton) oder vom Typ Chomsky-1*

[1]Noam Chomsky (MIT, 1956)

(iii) Haben alle Produktionen die Form

$$l \to r \quad mit\ l \in V,\ r \in (V \cup T)^*,$$

dann heißt die Grammatik kontextfrei[2] *oder vom Typ Chomsky-2.*

(iv) Haben all Produktionen die Form

$$l \to r \quad mit\ l \in V,\ r = \epsilon\ oder\ r = aA \in TV,$$

dann heißt die Grammatik rechtslinear *oder vom Typ Chomsky-3*[3]*.*

Bemerkungen

1. Kontextsensitive Grammatiken heißen manchmal auch „monoton", da abgesehen von der ϵ-Produktion $S \to \epsilon$ alle anderen Regeln wortlängenmonoton sind, d.h. die Anwendung einer Regel verkleinert nicht die Länge eines Wortes.

2. $l \to r \quad |l| \leq |r|$. In der Definition kontextsensitiver Grammatiken wurde die linke Seite einer Produktion auf Variablen beschränkt. Das stellt aber keine echte Einschränkung dar, da Variablen auch direkt auf ein Terminalzeichen abgebildet werden können (durch entsprechende Produktionen $A \to a$, $B \to b$).

3. Jede Chomsky-2 Grammatik kann in eine äquivalente Chomsky-2 Grammatik umgeformt werden, welche Chomsky-1 ist. Das heißt Chomsky-2 ist ein Spezialfall von Chomsky-1.

   Hierarchie:

   CH0 $>$ CH1 $>$ CH2 $>$ CH3

   d.h. CH$i+1$ ist echt in CHi enthalten.

9.2 Chomsky-0-Sprachen

Satz 15 *Sei Γ eine Chomsky-0-Grammatik. Dann gibt es eine NTM N, die die gleiche Sprache akzeptiert, die Γ erzeugt.*

Beweis: Sei Σ das Alphabet von Γ, sei $w \in \Sigma^*$.

Konstruiere eine NTM N, die eine Ableitung für w rät, falls $w \in L(\Gamma)$ und die ansonsten nicht terminiert. Offensichtlich gilt $L(N) = L(\Gamma)$. ∎

Korollar 15.1 *Sei Γ eine Chomsky-0-Grammatik. Dann gibt es eine deterministische Turing-Maschine M mit $L(\Gamma) = L(M)$, das heißt $L(\Gamma)$ ist die von M* akzeptierte *Sprache*[4]*.*

[2] Kontextfreie Grammatiken sind für Programmiersprachen bedeutsam

[3] Definition für linkslineare Grammatiken analog

[4] im allgemeinen finden wir zu Γ nicht unbedingt eine DTM, welche $L(\Gamma)$ *erkennt*; vgl. Gödel'scher Unvollständigkeitsatz

Beweis: Simulation von NTMs durch DTMs. ∎

Satz 16 *Sei N eine NTM. Dann gibt es eine Chomsky-0-Grammatik Γ mit $L(\Gamma) = L(N)$.*

Beweis: Ohne Beschränkung der Allgemeinheit können wir annehmen, daß

1. N nur ein Band hat (Eingabe+Arbeitsband).[5]
2. N hat (nur) einen akzeptierenden Endzustand q_f.
3. N akzeptiert mit einer Konfiguration der Form $\$_L q_f \not{b}^+ \$_R$ mit $\$_L, \$_R \notin \Sigma$ (d.h. sie räumt ihr Arbeitsband auf, bevor sie akzeptierend anhält)
4. $\$_L$ wird nie verändert, $\$_R$ begrenzt immer auf der rechten Seite den benutzten Bandteil.

Eine Rechnung von N beginnt mit $\$_L q_0 w_0 \ldots w_{n-1} \$_R$. Nach einigen Schritten könnte die Konfiguration $\$_L w'_0 \ldots w'_{n-1} q \$_R$ erreicht werden. Um $\$_R$ nach rechts zu schieben sind zwei Schritte notwendig:

$\$_L w'_0 \ldots w'_{n-1} \not{b} \tilde{q}$
$\$_L w'_0 \ldots w'_{n-1} q' \not{b} \$_R$
also: $\delta(q, \$_R) = (\tilde{q}, \not{b}, R)$, $\delta(\tilde{q}, \not{b}) = (q', \$_R, L)$

Bedingung 4. darf also kurzzeitig verletzt werden. Akzeptiert die Turing Maschine so erreicht sie die Konfiguration $\$_L q_f \not{b} \ldots \not{b} \$_R$.

Problem: die Turing Maschine startet mit dem Wort auf dem Band und hält (evtl.) in einer Endkonfiguration an. Eine Grammatik dagegen beginnt mit einem Startsymbol und produziert ein Wort.

$\Rightarrow$ Wir konstruieren eine Chomsky-0-Grammatik deren Ableitungen (insofern sie eine terminale Zeichenreihe liefern) die Berechnungen von N auf w simuliert, allerdings von hinten nach vorne.

[5] Wir wissen: zu jeder Mehrband-TM gibt es eine äquivalente 1-Band-TM.

$$
\begin{array}{lcll}
S & \to & \$_L q_f'' \$_R & \\
q_f''\$_R & \to & \not b q_f''\$_R | q_f'\$_R & \\
\not b q_f' & \to & q_f' \not b & \\
\$_L q_f' & \to & \$_L \binom{q_f}{\not b} \; ^6 & \\
\binom{q}{a} & \to & \binom{q'}{a'} & \text{falls } \delta(q',a') \ni (q,a,n) \text{ d.h. } N \text{ im Zustand } q' \text{ mit } a' \text{ unter} \\
& & & \text{dem Kopf einen Übergang besitzt in dem sie } a \text{ schreibt,} \\
& & & \text{in den Zustand } q \text{ übergeht und den Kopf nicht bewegt.} \\
a\binom{q}{b} & \to & \binom{q'}{a'}b & \text{falls } (q,a,R) \in \delta(q',a') \\
& & & a, a' \in \Sigma \quad \forall b \in \Sigma \cup \{\$_R\} \\
\binom{q}{a}b & \to & a\binom{q'}{b'} & \text{falls } (q,b,L) \in \delta(q',b') \\
& & & b, b' \in \Sigma \quad \forall a \in \Sigma \cup \{\$_L\}
\end{array}
$$

.

Speziell am rechten Ende:

$$
\begin{array}{lcl}
\binom{q'}{\not b}\$_R & \to & \not b\tilde{q} \\
\not b\tilde{q} & \to & \binom{q}{\$_R}
\end{array}
\quad \text{siehe oben}
$$

Entsprechend (aber einfacher :-) am linken Ende:

$$\$_L\binom{q_0}{a} \to \$_L a \quad \forall a \in \Sigma$$

Dann gilt: N akzeptiert $\$_L w \$_R \Leftrightarrow \exists$ akzeptierende Berechnung

$$C_0 = \$_L q_0 w \$_R \to C_1 \to C_2 \to \ldots \to C_T = \$_L q_f \underbrace{\not b \ldots \not b}_{B} \$_R$$

$\Leftrightarrow \exists$ Ableitung in der von uns konstruierten Grammatik mit

$$S \to \$_L q_f'' \$_R \to \ldots \to \$_L\binom{q_f}{\not b} \overbrace{\not b \ldots \not b}^{B-1} \$_r \to \ldots \to$$
$$\to \$_L\binom{q_0}{w_0} w_1 \ldots w_{n-1} \$_R \to$$
$$\to \$_L w \$_R$$

Detailliertes Argument: Induktion. ∎

Korollar 16.1 *Folgende Klassen von Sprachen sind gleich:*

(i) die durch Grammatiken vom Typ Chomsky-o erzeugten Sprachen.

(ii) die von nicht-deterministischen TM akzeptierten Sprachen.

(iii) die von deterministischen TM akzeptierten Sprachen

Diese Sprachen heißen auch „rekursiv-aufzählbare"-Sprachen

[6] $\binom{q_f}{\not b}$ ist ein neues Symbol (nicht-terminal) der Grammatik. Es steht für: TM ist im Zustand q_f auf dem Zeichen $\not b$

Wir können für diese Sprachen jeweils eine TM konstruieren, welche alle Wörter der Sprache in unbestimmter Reihenfolge erzeugt.

9.3 Chomsky-1-Sprachen

Satz 17 *Sei Γ eine kontextsensitive Grammatik. Dann gibt es eine NTM N, die $L(\Gamma)$ akzeptiert und die in ihrer Berechnung nie die beiden Endmarkierungen $\$_L$, $\$_r$ überschreitet oder verschiebt.*

(N hat nur ein Band, daß lese- und schreibbar ist. Sie hat also nicht mehr Arbeitsplatz, als die Länge der Eingabe.)

Beweis: rate Ableitung, geeigntes Bandalphabet ist notwendig (z.B. $\Sigma^3 \cup \{\text{einige Sonderzeichen}\}$).

Die einzelnen Schritte der Ableitung sind längenmonoton[7]. Die NTM benötigt also nie mehr Platz als das zu ratende Wort lang ist. Das spezielle Bandalphabet ermöglicht es der NTM drei Worte zugleich auf dem Band zu speichern:

1. das zu überspringende Eingabewort
2. das vorhergehende Wort der Ableitung
3. das nächste Wort der Ableitung.

∎

Definition 9.3.1 *Eine NTM mit Ein-/Ausgabeband, die die Endmarkierungen nicht überschreitet oder verändert heißt* „Linear beschränkter Automat“ *(LBA)*[8]

Satz 18 *Sei N ein LBA. Dann gibt es eine Chomsky-1-Grammatik Γ mit $L(\Gamma) = L(N)$.*

Beweis: Konstruktion wie bei Chomsky-0, ohne die Produktionen, die der Verschiebung des rechten Bandendes ($\$_R$) entsprechen.

Alle anderen Produktionen sind längenmonoton. ∎

LBAs sind grundsätzlich nicht-deterministisch. Die Frage, ob deterministische LBA's genauso mächtig sind wie (nicht-deterministische) LBA's ist *offen*.

Beispiel

die Sprache $\{0^i 1^i 2^i;\ i > 0\}$ ist kontext-sensitiv aber nicht mehr kontextfrei.

[7] Siehe Definition kontextsensitive Grammatik

[8] linear beschränkt deshalb, da der Bandplatz durch geeignete Wahl des Bandalphabets (z.B. Σ^n für festes n) eben doch linear in der Länge der Eingabe zur Verfügung steht.

9.4 Chomsky-3-Sprachen

Satz 19 *Sei Γ eine rechtslineare Grammatik. Dann gibt es einen nicht-deterministischen endlichen Automaten (NEA) A mit $L(A) = L(\Gamma)$.*

Beweis: Form der Produktionen in Γ:

$A \leftarrow \epsilon$
$(A \leftarrow a)$ wird ersetzt durch $A \leftarrow aZ$, $Z \leftarrow \epsilon$
$A \leftarrow aB$

∎

Satz 20 *Sei A ein NEA. Dann gibt es eine rechtslineare Grammatik Γ mit $L(A) = L(\Gamma)$.*

Korollar 20.1 *Folgende Klassen von Sprachen sind gleich:*

1. *die durch reguläre Ausdrücke beschriebenen Sprachen*[9]*,*
2. *die durch deterministische endliche Automaten erkannten Sprachen,*
3. *die durch nichtdeterministische endliche Automaten akzeptierten Sprachen,*
4. *die durch rechtslineare Grammatiken*[10] *erzeugten Sprachen,*
5. *die durch linkslineare Grammatiken erzeugten Sprachen.*

Aber:

Entweder ist eine Grammatik komplett links- oder komplett rechtslinear um eine reguläre Sprache zu erzeugen. Eine Grammatik die beide Sorten von linearen Regeln enthält, heißt *linear* und kann mehr als nur reguläre Sprachen erzeugen.

[9] also die regulären Sprachen
[10] also Chomsky-3

10. Kontextfeie Grammatiken und Sprachen

10.1 Grundlagen und ein Beispiel

$L_= := \{w \in \{0,1\}^*;\, w$ enthält gleich viele 0en und 1en$\}$. Sei $\#_a(w)$ die Anzahl der Zeichen a in der Zeichenreihe w. das heißt $L_=\{w \in \{0,1\}^* : \#_0(w) = \#_1(w)\}$ $L_=$ ist sicherlich nicht regulär[1].

Satz 21 *Die (kontextfreie) Grammatik* Γ.

$$\begin{array}{l} S \to 0E|1N|\epsilon \\ E \to 1S|0EE \\ N \to 0S|1NN \end{array}$$

erzeugt $L_=$.[2]

Beweis:

(i) Hilfsbehauptung: Sei $w \in \{0,1\}^*$

$$\begin{array}{lcl} S \to W & \Rightarrow & \#_0(w) = \#_1(w) \\ E \to W & \Rightarrow & \#_0(w) = \#_1(w) - 1 \\ N \to W & \Rightarrow & \#_0(w) = \#_1(w) + 1 \end{array}$$

Beweis durch Induktion über $|w|$. Also:

$$L(\Gamma) \subseteq L_=$$

(ii)

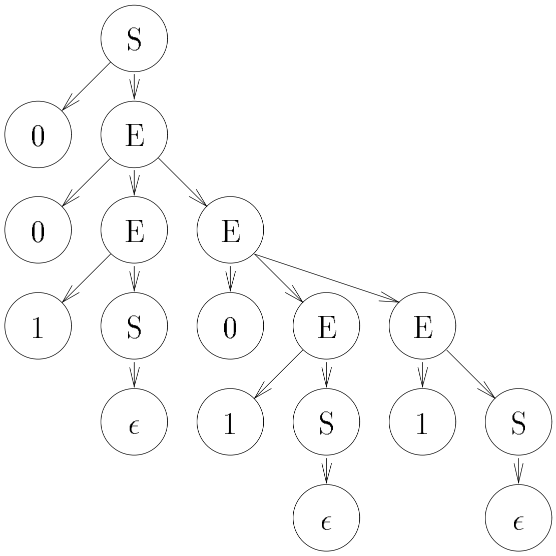
∎

[1] vgl. Pumping-Lemma

[2] E leitet alle Zeichenreihen ab, die eine 1 mehr als 0en enthalten.
N leitet alle Zeichenreihen ab, die eine 0 mehr als 1en enthalten.
$\Rightarrow S$ leitet alle Zeichenreihen ab, die gleichviele 0en und 1en enthalten

Es gilt: $S \to 0E \to 00EE \to 001S0EE \to 00101S1S \to 001011$

Dies war eine eindimensionale Darstellung einer Ableitung. Man kann solche Ableitungen auch als Bäume darstellen.



Die Beschriftung der Blätter ergibt das abgeleitete Wort. Diese Darstellung nennt man Syntaxbaum.

Beobachtungen

1. Zu jeder Ableitung gibt es einen (eindeutigen) Syntaxbaum.
2. Zu einem Syntaxbaum können mehr als eine Ableitung existieren.

Definition 10.1.1 *Eine Ableitung heißt eine* Links-Ableitung(Rechts-Ableitung)*, falls in jedem Schritt das am weitesten links (rechts) stehende Nichtterminal ersetzt wird.*

Beobachtung

Zu jedem Syntaxbaum gehört eine eindeutige Links- bzw. Rechts-Ableitung.

$w \in L(\Gamma)$ hat Syntaxbaum!

Definition 10.1.2 *(i) Eine kontextfreie Grammatik Γ heißt* eindeutig *wenn es für jedes $w \in L(\Gamma)$ genau einen Syntaxbaum gibt.*

(ii) Eine kontextfreie Sprache L heißt eindeutig*, falls es eine eindeutige kontexfrei Grammatik Γ mit $L(\Gamma) = L$ gibt. Ansonsten heißt L* inhärent mehrdeutig.

Die oben angegebene Grammatik ist nicht indeutig:

$$S \to 0E \to 00EE \to 001SE \to 0010EE \to 001011$$

der zu dieser Ableitung gehörende Syntaxbaum ist vom obigen Sytaxbaum verschieden.

10.2 Die Chomsky-Normalform

Sei Γ eine kontexfreie Grammatik, $\Gamma = (V, T, S, P)$. Der folgende Algorithmus bestimmt alle $A \in V$ mit $A \to \epsilon$:

$$\Delta := \{A \in V | A \to \epsilon \in P\},\ V_\epsilon := \emptyset$$

while $\Delta \neq \emptyset$ **do**
 $V_\epsilon := V_\epsilon \cup \Delta$
 $P := P \setminus \{A \to w | A \in \Delta\}$
 ersetze alle Vorkommen von $A \in \Delta$ in rechten Seiten
 in P durch ϵ
 $\Delta := \{A \in V | A \to \epsilon \in P\}$
od

Beweis: Einfache Induktion über die Länge der Ableitung. ∎

Korollar 21.1 $\epsilon \in L(\Gamma) \Leftrightarrow S \in V_\epsilon$

Definition 10.2.1

(i) Eine kontextfreie Grammatik Γ mit $\epsilon \notin L(\Gamma)$ ist in Chomsky-Normalform, falls alle Produktionen die Form $A \to a\,(a \in T)$ oder $A \to BC\,(BC \in V^2)$ haben.

(ii) Ist $\epsilon \in L(\Gamma)$, so fordern wir, daß $S \to \epsilon \in P$, S kommt auf keiner rechte Seite vor, alle Produktionen außer $S \to \epsilon$ erfüllen (i).

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform.

Eingabe: Eine kontextfreie Gramatik Γ

/* Wir konstruiernen zuerst eine Grammatrik für $L(\Gamma) \setminus \{\epsilon\}$ */

1. Wir konstruieren zuerst ein neues Nichtterminal Y_a hinzu, ersetzten in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als Produktion hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchtens */

2. Ersetze jede Produktion $A \to B_1 B_2 \dots B_r$ $(r \geq 3)$ durch $A \to B_1 C_2$, $C_2 \to B_2 C_3$, …, $C_{r-1} \to B_{r-1} B_r$ wobei $C_2, \dots, C_{r-1}$ neue Nichtterminale sind.

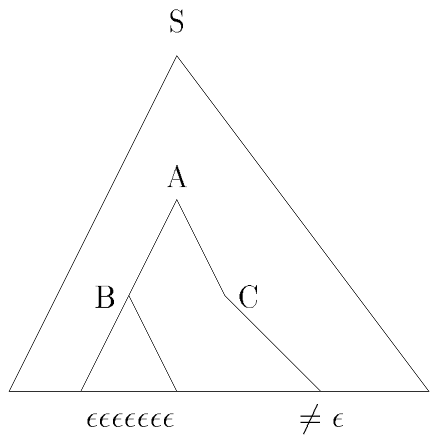
   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

3. Sei nun V_ϵ die Menge aller Nichtterminale A mit $A \xrightarrow{*} \epsilon$

   a) Streiche alle Produktionen der Form $A \to \epsilon$

   b) Für alle Produktionen $A \to BC$ mit $B \in V_\epsilon$ $(C \in V_\epsilon)$ füge die Produktion $A \to C$ $(A \to B)$ hinzu.

   /* Der Zeitaufwand ist linear in der Größe der Grammatik. Das Bestimmen von V_ϵ kostet aber $O(|V| \cdot s(G))$ */



4. Für alle Nichtterminale A, finde mittels Tiefensuche DFS, ob es eine Ableitung $A \to A_1 \to A_2 \to \dots \to A$ mit $A_1 \neq A$ gibt. Wenn ja, ersetze alle darin vorkommenden A_i's in allen Produktionen durch A und entferne alle Produktionen der Form $A \to A$

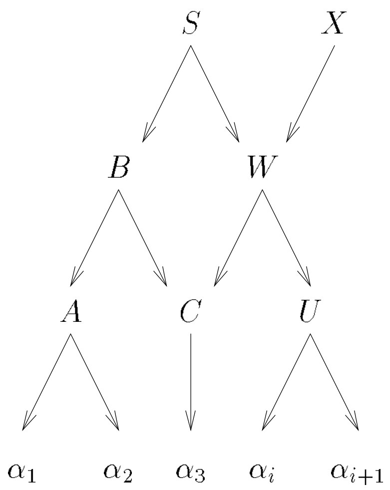
5. Benenne die Variablen so iin A_1, $dots, A_s$ um, daß gilt:

   Falls $A_i \to A_j \in P$, dann $i < j$ (topologisches Sortieren)

   /* linearer Aufwand */

6. Ersetze, in umgekehrter Reihenfolge (für j), $A_i \to A_j$ durch $A_i \to \alpha$, für alle α, die nicht Nichtterminale sind und für die $A_j \to \alpha$.

7. Falls $\epsilon \in L(\Gamma)$, füge ein neues Startsymbol S' und die Produktionen $S' \to \epsilon$, $S' \to S$ hinzu.



Betrachtung zur Komplexität des Algorithmus

1. Der Zeitaufwand des ersten Schrittes ist linear in der Größe der Grammatik. Die Größe der Grammatik ändert sich nur linear.
2. Dasselbe im zweiten Schritt.
3. Die Größe der Grammatik wächst im dritten Schritt nut linear. Das Bestimmen von V_ϵ erfordert Rechenzeit $O(|V| \cdot s(G))$ [3]
4. Schritt 4 benötigt Rechenzeit $O(|V| \cdot s(G))$ [4] und vergrößrt die Grammatik nicht.
5. Topologisches Sortieren ändert die Grammatik bis auf Isomorphie nicht und braucht lineare Zeit.
6. Schritt 6 braucht Zeit $O(|V| \cdot s(G))$ und vergrößrt die Grammatik höchstes um den Faktor $|V|$. D.h. die Größe der neuen Grammatik ist $O(|V| \cdot s(G))$, wenn $s(G)$ die Größe der alten Grammatik ist.
7. Der siebte Schritt an sich ist nicht der Rede wert, jdeoch der Test ob $\epsilon \in L(G)$, erfordert Zeitaufwand $O(|V| \cdot s(G))$

Zusammenfassend könne wir festhalten:

Satz 22 *Aus einer kontextfreien Grammatik $G = (V, T, S, P)$ der GröSSe $s(G)$ kann in Zeit $O(|V| \cdot s(G))$ eine äquivalente kontextfreie Grammatik der Größe $O(|V| \cdot s(G))$ erzeugt wreden.*

[3] $s(G)$ = Größe der Grammatik
[4] höchstens $|V|$ mal Depth First Search

10.3 Der Cocke-Younger-Kasami-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem Für kontextfreie Sprachen, falls die Sprache in Form einer Grammtik in Chomsky-Normalform gegeben ist.

Die Eingabe ist eine Grammatik $G = (V, T, S, P)$ in ϵ-freier Normalform und ein Wort $w = w_1 \dots w_n$ der Länge n.

Definition 10.3.1 $V_{ij} := \{A \in V | A \to w_i \dots w_j\}$

Es ist klar, daß $w \in L(G) \Leftrightarrow S \in V_{1n}$.

Der CYK-Algorithmus berechnet alle V_{ij} rekursiv nach wachsendem $j - i$. Den Anfang mache die $V_{ii} := \{A \in V | A \to w_i \in P\}$, der rekursive Aufbau erfolgt nach der Regel

$$V_{ij} = \bigcup_{i \leq k < j} \{A \in V | (A \to BC) \in P \land B \in V_{ik} \land C \in V_{k+1,j}\} \quad \text{für } i < j.$$

Die Korrektheit diesers Aufbaus ist klar, wenn die Grammtik in Chomsky-Normalform ist.

Zur Komplexität

Es werden $\frac{n^2+n}{2}$ Mengen V_{ij} berechnet. Für jede dieser Mengen werden $|P|$ Produktionen und weniger als N Werte für k betrachtet. Der test der Bedingung $(A \to BC) \in P \land B \in V_{ik} \land C \in V_{k+1,j}$ erfordert bei geigeter Repräsentation der Mengen V_{ij} konstanten Aufwand. Der Gesamtaufwand ist also $O(|P|n^3)$.

Mit der gleichen Methode und dem gleichen Rechenaufwand kann man zu den getsteten Wort, falls es in der Sprache ist, auch gleich einenen Ableitungsbaum konstruieren, indem man sich bei der Konstruktion der V_{ij} nicht nur merkt, welche Nichtterminale sie enthalten, sondern auch gleich, warum sie sie enthalten. D.h. aufgrund welcher Produktionen sie in die Menge aufgenommen wurden.

10.4 Das Pumping-Lemma und Ogdens Lemma für kontextfreie Sprachen

Zur Erinnerung:

Das Pumping-Lemma für reguläre Sprachen: Jedes Wort $z \in L$ mit $|z| \geq n$ läßt sich zerlegen in $z = uvw$ mit $|uv| \leq n$, $|v| \geq 1$ und $uv^*w \subseteq L$.

Zum Beweis wählte man $n = |Q|$, wobei Q die Zustandsmenge eines L erkennenden DEA war. Das Argument war dann, daß beim Erkennen von z (mindestens) ein Zustand zweimal besucht werden muß und der dazwischenliegende Weg geliebig oft wiederholt werden könnte.

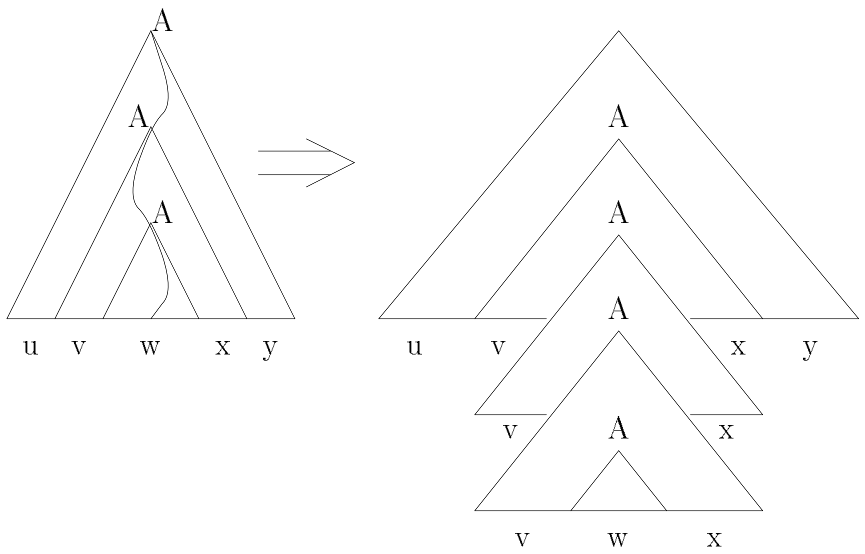
Völlig gleichwertig kann man argumentieren, daß bei der ABleitung von z mittels einer rechtslinearen Grammatik ein Nichtterminalsymbol zweimal auftreten muß und die dazwischenliegende Teilbaleitung beliebug oft wiederholt werden kann.

Genau dieses Argument kann in ähnlicher Form auch auf kontextfreie Grammatiken in Chomsky-Normalform angewendet werden und liefert die folgende Aussage:

Lemma 10.4.1 (Pumping-Lemma) *Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so daß sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen läßt in $z = uvwx$, mit $|vx| \geq 1$, $|vwx| \leq n$ und $\forall i \in \mathbb{N} : uv^iwx^iy \in L$.*

Beweisidee:

Sei $G = (V, T, S, P)$ eine Grammatik in CHomsky-Normalform mit $L(G) = L$. Wähle $n = 2^{|V|+1}$. Sei $w \in L(G)$ mit $|w| \geq n$. Dann hat der ABleitungsbaum für w mindestens die Tiefe $|V| + 1$ [5]. Auf einem Pfad der Länge $\geq |V| + 1$ kommt nun mindestens ein Nichtterminal zweimal vor[6] Die zwischen diesen beiden Vorkommen liegenden Teilableitung kann nun beliebig oft wiederholt werden.



[5]wegen Chomsky-Normalform hat er den Verzweigungsgrad 2

[6]Um $|vwx| \leq n$ zu erreichen, muß man das am weitesten unten liegende Doppelvorkommen eines solchen Nichtterminals wählen.

Anwendungsbeispiel:

Wir wollen sehen, daß die Sprache $\{a^i b^i c^i | i \in \mathbb{N}\}$ nicht kontextfrei ist. Wäre sie kontextfrei, so könnten wir das Wort $a^n b^n c^n$ (n die Konstante aus dem Pumoing-Lemma) aufpumpen, ohne aus der Sprache herauszukommen. Wir sehen aber leicht, dß bei jeder möglichen Verteilung der Teile v und x auf das Wort, Pumpen die Anzahl der a's, b's und c's unterschiedlich ändert.

Zur Vereinfachung von Beweisen wie in dem gerade gesehenen Beispiel führen wir folgenden Verschärfung des Pumping-Lemmas ein:

Lemma 10.4.2 (Ogdens Lemma) [7] *Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so daß für jedes Wort $z \in L$ mit $|z| \geq n$ die folgende Aussage gilt: Werden in z mindestens n (beliebige) Buchstaben makriert, so läßt sich z zerlegen in $z = uvwxy$, so daß in vx mindestens ein Buchstabe und in vwx höchstens n Buchstaben markiert sind und*

$$\forall i \in \mathbb{N} : \; uv^i wx^i y \in L$$

Beweis: Sei $G = (V, T, S, P)$ eine Grammatik in CHomsky-Normalform mit $L(G) = L$. Wähle $n = 2^{|V|+1}$. Sei $w \in L(G)$ und seien in w mindestens n Buchstaben markiert. In einem Ableitungsbaum für w markieren wir alle Konten deren linker und rechter Teilbaum jeweils mindestens ein markiertes Blatt enthalten. Es nun offensichtlich, daß es einen Pfad von der Wurzel zu einem Blatt gibt, auf dem mindestens $|V| + 1$ markierte Knoten liegen. Wir betrachten die letzten $|V| + 1$ markierten Knoten eines Pfades mit maximaler Anzahl markierter Knoten; nach Schubfachprinzip sind zwei mit demselben Nichtterminal A markiert, wir nennen diese Knoten v_1 und v_2. Seien die Blätter des in v_2 startenden Teilbaumes mit w und die Blätter des in v_1 startenden Teilbaumes mit vwx beschriftet. Es ist dann klar, daß die folgende Ableitung möglich ist:

$$S \xrightarrow{*} uAy \xrightarrow{*} uvAxy \xrightarrow{*} uvwxy.$$

Es ist auch klar, daß der Mittelteil dieser Ableitung weggelassen oder beliebig oft wiederholt werden kann.

Es bleibt noch zu sehen, daß vx mindestens einen und vwx höchstens n markierte Buchstaben enthält. Ersteres ist klar, da auch der andere, nicht Teilbaum des in v_1 startenden Teilbaumes ein markiertes Bltt haben muß. Das zweite is klar, da der gewählte Pfad maximale ANzahl von markierten Knoten hatte und unterhalb von v_1 nur noch höchstens $|V|$ markierte Knoten auf diesem Pfad sein konnen. Der con v_1 ausgehende Teilbaum kann also maximal $2^{|V|+1} = n$ markierte Blätter haben. ∎

Beispiel:

$L = \{a^i b^j c^k d^l | i = 0 \text{ oder } j = k = l\}$

[7] Das Pumping-Lemma ist eine triviale Folgerung aus dem Ogdens Lemma (markiere alle Buchtstaben in z).

Sei die Konstante aus Oydens Lemma n. Betrachte das Wort $ab^nc^nd^n$ und markiere darin bc^nd. Nun gibt es eine Zerlegung $ab^nc^nd^n = uvwxy$; so daß vx mindestens ein markiertes Symbol enthält und $uv^2wx^2y \in L$. Es ist jedoch leicht zu sehen, daß dies ein Widerspruch ist, da vx höschstens zwei verschiedene der Symbole b, c, d enthalten kann, damit beim Pumpen nicht die Reihenfolge durcheinander kommt.

10.5 Algorithmen für kontextfreie Sprachen/Grammatiken

Satz 23 *Sie $\Gamma = (V, T, S, P)$ kontextfrei. Dann kann die Menge der Variablen $A \in V$, für die gilt: $\exists w \in T^* : A \xrightarrow{*} w$ in Zeit $O(|V| \cdot s(\Gamma)$ berechnet werden.*

Beweis: $\Delta := \{A \in V | \exists (A \rightarrow) \in P$ mit $w \in T^*\}$; $V' := \emptyset$.

while $\Delta \neq \emptyset$ **do**
    $V' := V' \cup \Delta$
    $\Delta := \{A \in V \setminus V' | \exists (A \rightarrow \alpha) \in P$ mit $\alpha \in (V' \cup T)^*\}$,
od

Induktion über die Länge der Ableitung. ∎

Definition 10.5.1 *$A \in V$ heißt* nutzlos, *falls es keine Ableitung $S \xrightarrow{*} w \in T^*$ gibt, in der A vorkommt.*

Satz 24 *Die Menge der nutzlosen Variablen kann in Zeit $O(|V| \cdot s(\Gamma))$ berechnet werden.*

Beweis: Sei V'' die Menge der nicht nutzlosen Variablen. Offensichtlich gilt : $V \subseteq V'$ (aus dem vorigen Satz). Falls $S \notin V'$, dann sind alle Variablen nutzlos.

Ansonsten:

while $\Delta \neq \emptyset$
    $V'' := V'' \cup \Delta$
    $\Delta := \{B \in V' \setminus V'' | \exists (A \rightarrow \alpha B \beta) \in P$ mit $A \in V''$, $\alpha, \beta \in (V' \cup T)^*\}$;
od

Induktion über Länge der Ableitung zeigt: Am Ende des Algorithmus ist V'' gleich der Menge der nicht nutzlosen Variablen. ∎

Bemerkung:

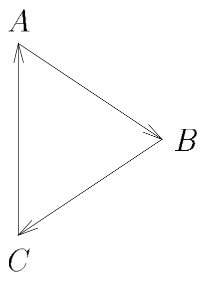
Alle nutlosen Variablen und alle Produktionen, die nutlose Variablen enthalten, können aus der Grammatik entfernt werden, ohne die erzeugte Sprache zu ändern.

Korollar 24.1 *Für eine kontextfreie Grammatik Γ kann in Zeit $O(|V| \cdot s(\Gamma))$ entschieden werden, ob $L(\Gamma) = \emptyset$.*

Beweis: $L(\Gamma) = \emptyset \Leftrightarrow S \notin V''$ (bzw. $S \notin V'$) ∎

Satz 25 *Für eine kontextfreie Grammatik Γ ohne nutlose Variablen und in CHomsky-Normalform kann in linearer Zeit entschieden werden, ob $|L(\Gamma)| < \infty$*

Beweis: Definiere Graphen mit Knotenmenge V [8] und Kante $A \to B \Leftrightarrow (A \to BC)$ oder $(A \to CB) \in P$ $L(\Gamma)$ ist endlich $\Leftrightarrow$ dieser Graph enthält keinen Zyklus.



$$\begin{array}{c} A \\ \downarrow \\ BD_1 \\ \downarrow \\ D_2CD_1 \\ \downarrow \\ D_2AD_3D_1 \end{array}$$

Verwende DFS, um in lineraer Zeit festzustellenb, ob der Graph Zyklus enthält. ∎

Satz 26 *Seien kontextfreie Grammatiken $\Gamma_1 = (V_1, T_1, S_1, P_1)$ und $\Gamma_1 = (V_1, T_1, S_1, P_1)$ gegeben. Dann können in linearer Zeit kontextfreie Grammatiken für*

(i) $L(\Gamma_1) \cup L(\Gamma_2)$

(ii) $L(\Gamma_1)L(\Gamma_2)$

(iii) $(L(\Gamma_1))^*$

konstruiert werden. Die Klasse der kontextfreien Sprachen ist also unter Vereinigung, Konkatination *und* Kleene'scher Hülle *abgeschlossen.*

Beweis: Ohne Beschränkung der Allgemeinheit[9] $V_1 \cap V_2 = \emptyset$

(i) $V = V_1 \cup V_2 \cup \{S\}$; S neu
$P = P_1 \cup P_2 \cup \{S \to S_1 | S_2\}$

(ii) $V = V_1 \cup V_2 \cup \{S\}$; S neu
$P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$

[8] wobei $\Gamma = (V, T, S, P)$

[9] notfalls werden die Variablen in einer Grammatik umbenannt

(iii) $V = V_1\{S\}$; S neu
$P = P_1 \cup \{S \to SS|S_1|\epsilon\}$

$\blacksquare$

Satz 27 *Die Klasse der kontextfreien Sprachen ist* nicht *abgeschlossen unter Durchschnitt oder Komplement.*

Beweis: Es genügt zu zeigen (wegen *de Morgan*): nicht abgeschlossen unter Durchschnitt:

$L_1 := \{a^i b^i c^j;\ i, j \geq 0\}$ ist kontextfrei
$L_2 := \{a^i b^j c^j;\ i, j \geq 0\}$ ist kontextfrei $\blacksquare$
$L_1 \cap L_2 = \{a^i b^i c^i;\ i \geq 0\}$ ist *nicht* kontextfrei

Satz 28 *Die Klasse der kontextfreien Sprachen ist abgeschlossen gegebnüber Substitution (mit kontextfreien Mengen).*

Beweis: $\Gamma = (v, T, S, P)$ $\quad \forall a \in T \quad \Gamma_a$ kontextfrei, Startsymbol S_a. $\blacksquare$

10.6 Greibach-Normalform

Definition 10.6.1 *Sei Γ eine kontextfreie Grammatik, $\epsilon \notin L(\Gamma)$. Γ heißt in Greibach-Normalform*[10]*, falls jede Produktion von der Form*

$$A \to a\alpha \textit{ ist, mit } a \in T,\ \alpha \in V^*$$

(wobei $\Gamma = (V, T, S, P)$).

Lemma 10.6.1 *Sei $\Gamma = (V, T, S, P)$ kontextfrei,*
$(A \to \alpha_1 B \alpha_2) \in P$, und sei $B \to \beta_1|\beta_2|\ldots|\beta_r$ die Menge der B-Produktionen[11]*. Ersetzt man $A \to \alpha_1 B \alpha_2$ durch $A \to \alpha_1\beta_1\alpha_2|\alpha_1\beta_2\alpha_2|\ldots|\alpha_1\beta_r\alpha_2$; so ändert sich die erzeugte Sprache nicht.*

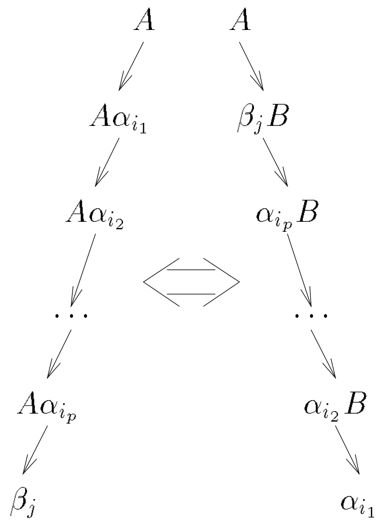
Lemma 10.6.2 *Sei $\Gamma = (V, T, S, P)$ kontextfrei, sei $A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_r$ die Menge der linksrekursiven A-Produktionen und seien $A \to \beta_1|\beta_2|\ldots|\beta_s$ die restlichen A-Produktionen. Ersetzen wir alle A-Produktionen durch*

$A \to \beta_1|\ldots|\beta_s|\beta_1 B|\ldots|\beta_s B \qquad B$ *neu*
$B \to \alpha_1|\ldots|\alpha_r|\alpha_1 B|\ldots|\alpha_r B$

so ändert sich die Sprache nicht.

[10] benannt nach Sheila Greibach
[11] $\hat{=}$ Menge der Produktionen die B ableiten

Beweis:



∎

Satz 29 *Zu jeder kontextfreien Grammatik* Γ, $\epsilon \notin L(\Gamma)$, *gibt es eine äquivalente Grammatik in Greibach-Normalform.*

Beweis: Sei $\Gamma = (V, T, S, P)$; $V = \{A_1, \dots, A_m\}$ in Chomsky-Normalform

zuerst: $A_i \to A_j\alpha \Rightarrow j > i$

for $k = 1, \dots, m$ **do**
    for $j = 1,\ dots, k-1$ **do**
        foreach $(A_k \to A_j\alpha) \in P$ **do**
                ersetze $A_k \to A_j\alpha$ durch $A_k \to \beta_1\alpha|\beta_2\alpha|\dots,$
                wobei die β_i die rechten Seiten der A_j-Produktionen sind
        od
    od
    ersetze alle linksrekursiven Produktionen der Form $A_k \to A_t\alpha_i$ durch
      $A_k \to \beta_1|\dots|\beta_s|\beta_1 B_k|\dots|\beta_s B_k$ B_k neu
      $B_k \to \alpha_1|\dots|\alpha_r|\alpha_1 B_k|\dots|\alpha_r B_k$
    wobei die β_i die rechten Seiten der restlichen A_k-Produktionen sind.
od
co jede A_m-Produktion beginnt mit Terminal,
   $A_i \to A_j\alpha \Rightarrow j > i$
   keine B_i-Produktion beginnt mit einem B_j (rechte Seite) **oc**

$k = 1$: j-Schleife leer

$A_1 \to A_1\alpha \qquad A_1 \to A_{\underset{\geq 2}{2}}$

$k = 2$: $j = 1$

$A_2 \to A_1\alpha \qquad A_2 \to A_{\underset{i \geq 2}{i}}\alpha$

$A_2 \to A_2\alpha \qquad A_2 \to A_{\underset{i \geq 3}{i}}\alpha$

$k = 3$: $j = 1$

$A_3 \to A_1\alpha \qquad A_3 \to A_{\underset{i \geq 2}{i}}\alpha$

$j = 2$

$A_3 \to A_2\alpha \qquad A_3 \to A_{\underset{i \geq 3}{i}}\alpha$

$A_3 \to A_3\alpha \qquad A_3 \to A_{\underset{i \geq 4}{i}}\alpha$

co falls nun $A_i \to A_j\alpha$, dann $j > i$, also jede rechte Seite einer A_m-Produktion beginnt mit einem Terminal; keine rechte Seite einer B_i-Produktion beginnt mit einer B-Variable **oc**

for $k = m - 1, \ldots, 1$ **do**
  foreach $(A_k \to A_j\alpha)$ **do**   **co** $j > k$ **oc**
    ersetze diese Produktion gemäß 1.Lemma
  od
od
for $k = m, \ldots, 1$ **do**
  foreach $(B_k \to A_j\alpha)$ **do**
    ersetze diese Produktion gemäß 1. Lemma
  od
od

∎

Satz 30 *Sei Γ kontextfreie Grammatik, $\epsilon \notin L(\Gamma)$. Es gibt einen Algorithmus, der eine zu Γ äquivalente Grammatik in Greibach-Normalform konstruiert, deren rechte Seite jeweils höchstens zwei Variablen enthalten.*

Beweis: siehe Hauptstudium: „*Formale Sprachen*“ ∎

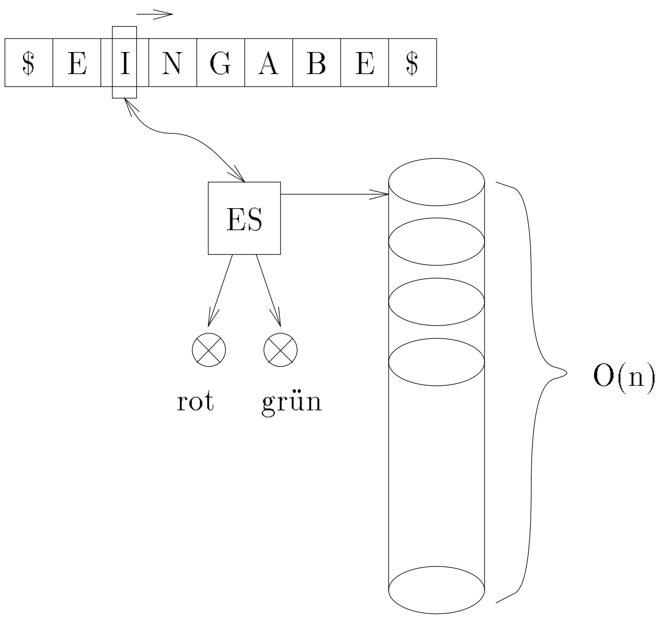
Satz 31 *Sei Γ kontextfrei Grammatik, ohne ϵ-Regeln, ohne Kettenregeln und separiert*[12]. *Sei p die Anzahl der Produktionen. Dann gibt es eine zu Γ äquivalente Grammatik in Greibach-Normalform, die $\leq \max\{p^3, 64\}$ Produktionen enthält.*

Beweis: siehe Hauptstudium: „*Formale Sprachen*“ ∎

[12] d.h. rechte Seite $\in T$ oder V^+

10.7 Kellerautomaten

In der Literatur findet man häufig auch die Bezeichnungen Stackautomaten oder Pushdownautomaten.



NDPA(=PDA)

Q	endliche Zustandsmenge
Σ	endliches Eingabealphabet
Δ	endliches Stackalphabet
$q_0 \in Q$	Anfangszustand
$Z_0 \in \Delta$	Initialisierung des Stack
δ	Übergangsrelation
	Fkt. $Q \times (\Sigma \cup \{\epsilon\}) \times \Delta \rightarrow 2^{Q \times \Delta^*}$
	wobei $\lvert\delta(q,a,Z)\rvert + \lvert\delta(q,\epsilon,Z)\rvert < \infty \quad \forall q,a,Z$
w	$F \subseteq Q$ akzeptierende Zustände

Konfiguration

(q, w, α) mit

$$\begin{array}{ll} q & \in Q \\ w & \in \Sigma^* \\ \alpha & \in \Delta^* \end{array}$$

Schritt

$(q, w_0w', Z\alpha') \to (q', w', Z_1 \dots Z_r\alpha')$
g.d.w. $(q', Z_1 \dots Z_r) \in \delta(q, w_0, Z)$
bzw.:
$(q, w, Z\alpha') \to (q', w, Z_1 \dots Z_r\alpha')$
g.d.w. $(q', Z_1 \dots Z_r) \in \delta(q, \epsilon, Z)$

Definition 10.7.1 *(i) Ein NPDA A* akzeptiert $w \in \Sigma^*$ durch leeren Stack*, falls* $(q_0, w, Z_0) \xrightarrow{*} (q, \epsilon, \epsilon)$ *für ein* $q \in Q$

(ii) Ein NDPA A akzeptiert $w \in \Sigma^*$ durch akzepierenden Zustand*, falls* $(q_0, w, Z_0) \xrightarrow{*} (q, \epsilon, \alpha)$ *für* $q \in F$, $\alpha \in \Delta^*$

(iii) Ein NPDA heißt deterministisch *(DPDA), falls* $|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1 \; \forall (q, a, Z) \in Q \times \Sigma \times \Delta$

Beispiel:

$L = \{w\#w^R, \; w \in \{0,1\}^*\}$

	$\delta(q_0, {}^0_1, *)$	$=$	$\{(q_0, {}^0_1 *)\}$	
	$\delta(q_0, \#, *)$	$=$	$\{(q_1, *)\}$	
	$\delta(q_1, 0, 0)$	$=$	$\{(q_1, \epsilon)\}$	
	$\delta(q_1, 1, 1)$	$=$	$\{(q_1, \epsilon)\}$	
(i)	$\delta(q_1, \epsilon, Z_0)$	$=$	$\{(q_1, \epsilon)\}$	akzeptiert mit leerem Stack
(ii)	$\delta(q_1, \epsilon, Z_0)$	$=$	$\{(q_a, \epsilon)\}$	akzeptiert mit akzeptierendem Zustand ($F = \{q_a\}$) (und leerem Stack)

Satz 32 *Sei $A_1 = (Q, \Sigma, \Delta', q_0, Z_0, \delta, F)$ ein NPDA, der mit Endzustand akzeptiert. Dann kann in linearer Zeit ein NPDA $A_2 = (Q', \Sigma, \Delta', q_0', Z_0', \delta', \emptyset)$ konstruiert werden, der $L(A_1)$ mit leerem Stack akzeptiert.*

Beweis:

Intuitiv:

A_2 simuliert A_1. Sobald A_1 in einem akzeptierenden Endzustand grät, rät A_2 ob die Eingabe fertiggelesen ist. Falls A_2 meint die Eingabe sei völlig gelesen, so wird der Keller geleert.

Um zu verhindern, daß bei Simulation von A_1 der Keller leer wird ohne daß A_1 akzeptiert, führen wir das neue Kellersymbol Z_0' ein.

$Q' := Q \cup \{\bar{q}, q_0'\}$
$Z' = \Delta \cup \{Z_0'\}$

$$
\begin{array}{ll}
\delta'(q_0', \epsilon, Z_0') = \{(q_0, Z_0 Z_0')\} & \\
\delta'(q, a, z) \supseteq \delta(q, a, z) & \text{für } q \in Q,\ a \in \Sigma \cup \{\epsilon\},\ z \in \Delta \\
\delta'(q, \epsilon, z) \ni (\bar{q}, \epsilon) & \text{für } q \in F,\ Z \in \Delta' \\
\delta'(q, \epsilon, z) = (\bar{q}, \epsilon) & \text{für } Z \in Z'
\end{array}
$$

Bemerkung

Akezeptieren mit leerem Keller bedeutet, der NPDA akzeptiert, falls der Keller leer ist *und* die Eingabe gelesen ist.

∎

Satz 33 *Sei $A_1 = (Q, \Sigma, \Delta, q_0, Z_0, \delta, \emptyset)$ ein NPDA der mit leerem Keller akzeptiert. Dann kann in linearer Zeit ein NPDA $A_2 = (Q', \Sigma, \Delta', q_0', Z_0', \delta', F)$ konstruiert werden, welcher $L(A_1)$ mit akzeptierendem Endzustand akzeptiert.*

Beweis:

Intuitiv:

A_2 simuliert A_1. Am Anfang steht ein neues Kellersymbol auf dem Stack, sobald bei der Simulation von A_1 dieses auf dem Stack vorgefunden wird, weiß man, daß A_1 seinen Stack leergeräumt hat und folglich akzeptieren wird. Folglich geht A_2 in einen akzeptierenden Endzustand und verharrt[13][14].

$$
\begin{array}{lcl}
Q' & = & Q \cup \{q_0', q_f\} \\
\Delta' & = & \Delta \cup \{z_0'\} \\
F & = & \{q_f\}
\end{array}
$$

$$
\begin{array}{lcl}
\delta'(q_0', \epsilon, z_0') & = & \{(q_0, z_0 z_0')\} \\
\delta'(q, a, z_a) & = & \delta(q, a, z_a) \\
\multicolumn{3}{l}{\quad \text{für } q \in Q,\ a \in \Sigma \cup \{\epsilon\},\ z \in \Delta} \\
\delta'(q, \epsilon, z_0') & = & \{(q_f, z_0')\} \quad q \in Q
\end{array}
$$

∎

10.8 Kellerautomaten und kontexfreie Sprache

Satz 34 *Sei Γ eine CFG in Greibach-Normalform. Dann kann in linearer Zeit ein NPDA A konstruiert werden (welcher mit leerem Stack akzeptiert), so daß $L(\Gamma) = L(A)$.*

[13] d.h. liest nicht weiter

[14] Akzeptieren mit Endzustand heißt, daß der NPDA in einem akzeptierenden Endzustand ist und die Eingabe vollstänig gelesen hat.

Beweis: Zur Erinnerung: Greibach - Normalform : $P \supseteq V \times TV^*$.

Intuitiv:

Der Automat startet mit S auf dem Stack, er sieht sich das oberste Stacksymbol A an. Er überprüft dann, ob es eine Regel in Γ gibt deren linke Seite A hat und auf deren rechter Seite mit dem Terminal beginnt welches in der Eingabe steht.

Sei nun $\Gamma = (V, T, S, P)$, $A = (Q, \Sigma, \Delta, q_0, z_0, \emptyset)$ mit

$$\begin{array}{lcl} Q & := & \{q_0\} \\ \Delta & := & V \\ \Sigma & := & T \\ Z_0 & := & S \end{array}$$

$$\delta(q_0, a, A) \ni (q_0, \alpha) \quad \text{für } (A \to a\alpha) \in P$$

$$\text{zu zeigen } L(A) = L(\Gamma)$$

Hilfsbehauptung:

$$s \xrightarrow{*} w_1 \dots w_i A_1 \dots A_m \quad (w_j \in T^* \; A_j \in N^*)$$

$$\Leftrightarrow (q_0, w_1 \dots w_i, S) \overset{*}{\models} (q_0, \epsilon, A_1 \dots A_m)$$

Beweis durch Induktion über i:

I.A.: $i = 0$:

$$S \xrightarrow{0} \Leftrightarrow (q_0, \epsilon, S) \overset{0}{\models} (q_0, \epsilon, S)$$

I.S.: $(i-1) \to i$:

$$\begin{array}{lll} & S & \xrightarrow{*} w_1 \dots w_i A_1 \dots A_m \\ \Leftrightarrow & S & \xrightarrow{*} w_1 \dots w_{i-1} A' A_v \dots A_m \quad v \in \{1, ldots, m+1\} \\ & & \to w_1 \dots w_i A_1 \dots A_m \\ & & (\text{also } (A' \to w;\; A_1 \dots A_{v-1}) \in P) \end{array}$$

$$\begin{array}{llll} \text{I.V.:} & \Leftrightarrow & (q_0, w_1 \dots w_{i-1}, z_0) & \overset{*}{\models} (q_0, \epsilon, A' A_v \dots A_m) \\ & \Leftrightarrow & (q_0, w_1 \dots w_{i-1} w_i, z_0) & \overset{*}{\models} (q_0, w_i, A' A_v \dots A_m) \\ & & & \overset{*}{\models} (q_0, \epsilon, A_1 \dots A_m) \\ & & & \text{da } (A' \to w;\; A_1 \dots A_{v-1}) \in P) \\ & & & \\ & \Leftrightarrow & (q_0, w_1 \dots w_i, z_0) & \overset{*}{\models} (q_0, \epsilon, A_1 \dots A_m) \end{array}$$

$\square$

Aus Hilfsbehauptung $\Rightarrow L(A) = L(\Gamma)$. ▮

Satz 35 *Sei $A = (Q, \Sigma, \Delta, q_0, z_0, \delta, \emptyset)$ ein NPDA der mit leerem Keller akzeptiert. Dann ist $L(A)$ kontextfrei.*

Beweis: Wir definieren:

$\Gamma = (V, T, S, P)$
$T := \Sigma$
$V := Q \times \Delta \times Q \cup \{S\}$
$P:\ S \rightarrow [q_0, z_0, q]$ für $q \in Q$

$[q, z, q_m] \rightarrow a[p, z, q_1][q_1, z, q_2] \dots [q_{m-1}, z_v, q_m]$
für $\delta(q, a, z) \ni (p, z_1 \dots z_m) \forall q_1 \dots q_m \in Q$

Intuitiv:

Aus $[p, x, q]$ soll sich das Wort ableiten lassen, was der *NPDA A* verarbeiten kann, wenn er mit X auf auf dem Stack im Zustand q startet und im Zustand q mit leerem Keller endet.

Hilfsbehauptung:

$[p, X, q] \xrightarrow{*} wLeftrightarrow(p, w, x) \overset{*}{\models} (q, \epsilon, \epsilon)$.

„$\Rightarrow$“: Induktion über die Länge l der Ableitung.

I.A.: $l = 1$

$$\begin{aligned} & [p, x, q] \xrightarrow{1} w \\ \Rightarrow\ & \delta(p, w, x) \ni (q, \epsilon) \\ \Rightarrow\ & (p, w, x) \overset{1}{\models} (q, \epsilon, \epsilon) \end{aligned}$$

I.S.: $(l-1) \rightarrow l$

$$\begin{aligned} [p, x, q_{m+1}] \ & \xrightarrow{1} a[q_1, x_1, q_2][q_2, x_2, q_3] \dots [q_m, x_m, q_{m+1}] \\ & \xrightarrow{*} aw_1 \dots w_m = w \end{aligned}$$

mit $([q_i, x_i, q_{i+1}] \overset{\leq(l-1)}{\rightarrow} w_i) \wedge$
$(\delta(p, a, x) \ni (q_1, x_1 \dots x_m)$

$$\begin{array}{llcl} \text{I.V.:}\ \Rightarrow & (q_i, w_i, x_i) & \models & (q_{i+1}, \epsilon, \epsilon) \quad \forall i \in \{1 \dots m\} \\ \Rightarrow & (q, \underbrace{aw_1 \dots w_m}_{=w}, x) & \overset{1}{\models} & (q_1, w_1 \dots w_m, x_1 \dots x_m) \\ & & \overset{\leq(l-1)}{\models} & (q_1, w_1 \dots w_m, x_1 \dots x_m) \\ & & \vdots & \\ & & \models & (q_{m+1}, \epsilon, \epsilon) \end{array}$$

$\square$

„$\Leftarrow$“: Induktion über die Länge l einer Rechnung des NPDA's

I.A.: $l = 1$

$$\begin{aligned}
&(p, w, x) \overset{1}{\models} (q, \epsilon, \epsilon) \\
&\Rightarrow (q, \epsilon) \in \delta(p, w, x) \quad \text{mit } |w| \leq 1 \\
&\Rightarrow [p, x, q] \rightarrow w \quad \text{(nach Definition der Größe)}
\end{aligned}$$

I.S.: $(l-1) \rightarrow l$

$$\begin{aligned}
(p, w, x) \;&\overset{1}{\models}\; (q_1, w^0, x_1, x_m) \\
&\overset{*}{\models}\; (q, \epsilon, \epsilon)
\end{aligned}$$

sei Rechnung der Länge l

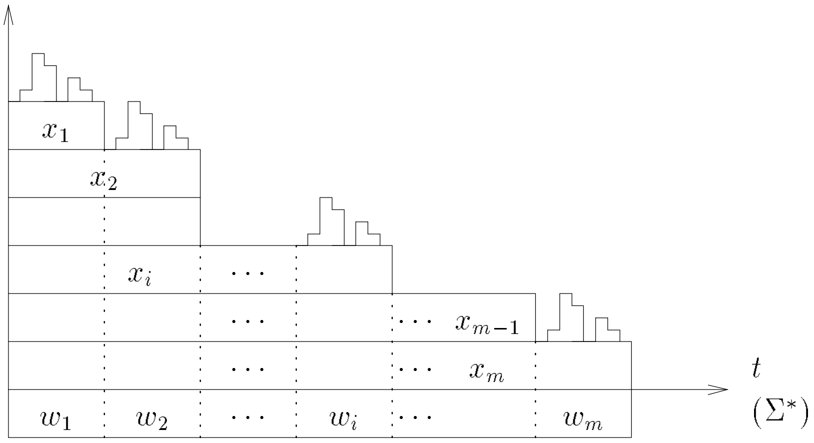
Dann ist $w = ew'$ mit $e \in \Sigma \cup \{\epsilon\}$.

Nach Definition der Größe gilt:

$([p, x, q] \rightarrow e[q_1 x_1 q_2] \ldots [q_m x_m q_{m+1}]) \in P \quad \text{mit } q_{m+1} = q$

und einer Relegung $w' = w_1 \ldots w_m$, so daß $w_1 \ldots w_i$ der verarbeitete Teilstring von w' ist, wenn x_i zum erstenmal gepoppt wird.

Stack



I.V.: $(q_i, w_i, x_i) \overset{\leq (l-1)}{\models} (q_{i+1}, \epsilon, \epsilon)$

$[q_i, x_i, q_{i+1}] \overset{*}{\rightarrow} w_i$

Also gilt:

$$\begin{aligned}
[p, x, q] &\overset{1}{\rightarrow} e[q_q, x_1, q_2] \ldots [q_m, x_m, q_{m+1}] \quad \text{mit } q_{m+1} = q \\
&\overset{1}{\rightarrow} e w_1 \ldots w_m = w
\end{aligned}$$

$\square$

Aus der Hilfbehauptung folgt der Satz. ∎

Satz 36 *Folgende Aussagen sind äquivalent:*

(i) L wird von einer CFL erzeugt.

(ii) L wird von einem NPDA akzeptiert.

Beweis: Folgt aus den vorherigen Sätzen ∎

Bemerkung

Die Familie der kontextfreien Sprachen heißt auch Familie der Chomsky-2-Sprachen.

Für die Praxis[15] sind polynomiale Algorithmen noch zu langsam. Für Teilklassen con *CFL*s sind schnellere Algorithmen bekannt.

Beschränkt man sich auf deterministische Kellerautomaten, so erhält man eine Teilklasse con *CFL*, welche ???

10.9 Determinitische Kellerautomaten

Definition 10.9.1 *Ein Kellerautomat heißt deterministisch, falls*

$$|\delta(q,a,X)| + |\delta(q,\epsilon,X)| \leq 1$$

für $q \in Q$, $a \in \Sigma X$, $\in \Delta$. *und der Automat mit akzeptierenden Zuständen akzeptiert (und nicht mit leerem Keller). Für einen deterministischen Kellerautomaten schreiben wir kurz DPDA.*

Bemerkung

Würde wir mit der Akzeptanz mit leerem Keller arbeiten, so wäre wir nur in der Lage Sprachen, die der Fano-Bedingung genügen, zu erkennen.

Definition 10.9.2 *Ein DPDA ist in Normalform, falls gilt:*

(i) $q', a) = \delta(q, e, X)$ *für* $e \in \Sigma \cup \{\epsilon\}$ $q, q' \in Q$, $X \in \Delta \Rightarrow \alpha \in \{\epsilon, X, YX\}$ *für* $Y \in \Delta$.

(ii) Der Automat liest jede Eingabe vollständig.

[15] Syntaxanalyse von Programmen

Satz 37 *ZU jedem DPDA* $A = (Q, \Sigma, \Delta, q_0, z_0, \delta, F)$ *kann ein äquivalenter DPDA* $A' = (Q', \Sigma', \Delta', q_0', z_0', \delta', F')$ *konstruiert werden.* $(L(A) = L(A'))$.

Beweis:

ad(i) : Werden mehree Kellersymbole gepusht, wird dies durch eine Folge von Push's mit je 2 Kellersymbolen ersetzt.

Werden 2 oder 1 Kellersymbol gepusht und dabei daß oberste Kellersymbol geändert, entfernen wir zunächst das oberste Kellersymbol und pushen dann die gewünschten Symbole. (Dabei wird jedesmal zum Merken die endliche Zustandsmenge benutzt. Da nur endlich viele Änderungen nötig sind, bleibt die Zustandmenge endlich.)

ad(ii) : δ' vollständig machen mittels eines Fangzustandes. Es könnte hier noch sein, daß der DPDA beliebig viele ϵ-Übergänge macht, ohne jedoch etwas von der Eingabe zu lesen.

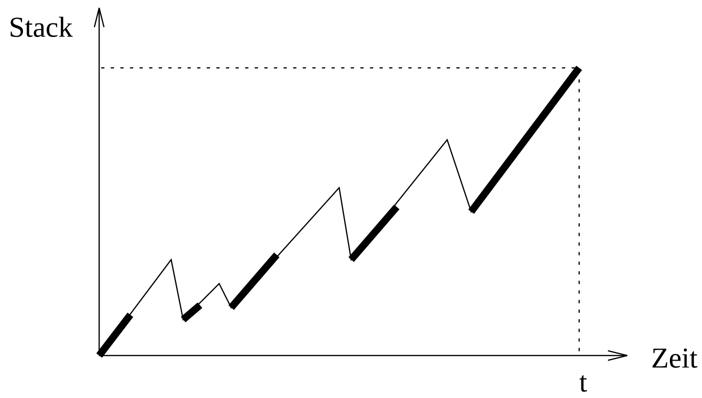
Hilfsbehauptung:

DPDA A führt ab einer bestimmten Konfiguration (q, ϵ, β) ∞ viele ϵ-Bewegungen aus, genau dann, wenn

$$\begin{array}{lcll} (q, \epsilon, \beta) & \overset{*}{\models} & (q', \epsilon', x\beta') & \text{und} \\ (q', \epsilon, x) & \overset{*}{\models} & (q', \epsilon, x\alpha) & \text{für } q, q' \in Q \\ & & & x \in \Delta,\ \alpha, \beta, \beta' \in \Delta^* \end{array}$$

„$\Leftarrow$": klar

„$\Rightarrow$": Betrachte eine unendlich lange Berechnung von ϵ-Übergängen



Sei $n := |Q| \cdot |\Delta|$

Wird die Stackhöhe n nie erreicht, so muß sich sogar eine Konfiguration des DPDA's wiederholen. Daraus folgt die Behauptung.

Ansonsten wird jede Stackhöhe $0 \ldots n$ mindestens einmal erreicht (wegen Normalform; i.e. Stackdiff $\in \{-1, 0, 1\}$).

Betrachte den Zeitpunkt t, in dem die Stackhöhe zum erstenmal n ist. Markiere für jedes $i \in (a, n)$ den Zeitpunkt t_i, wo zum letzenmal (vor Zeitpunkt t) die Stackhöhe i ist. Zu diesen Zeitpunkten t_i betrachte die Paare $(q, x) \in Q \times \Delta$, wobei q der Zustand des DPDA's und x das oberste Kellersymbol des DPDA's zu diesem Zeitpunkt ist.

Da es mehr als $|\Delta| \cdot |Q|$ markierte Paare gibt taucht ein markiertes Paar (q', x) doppelt auf. Für dieses gilt dann $(q', \epsilon, x) \overset{+}{\models} (q', \epsilon, x\alpha)$.

Ferner gibt es einen Weg dorthin,

$$(q, \epsilon, \beta) \overset{*}{\models} (q', \epsilon, x\beta')$$

□

Das gleiche Argument gilt, falls sich die Stackhöhe um $> |Q| \cdot |\Delta|$ erhöht.

Damit lassen sich alle Paare (q', x) finden, für die gilt: $(q', \epsilon, x) \overset{+}{\models} (q', \epsilon, x\alpha)$ $\alpha \in \Delta^*$. Da der DPDA nicht endlos weiterlaufen soll ersetzen wir $\delta(q', \epsilon, x)$ durch einen ϵ-Übergang in einen neuen Zustand q'' (der genau dann akzeptierend ist, wenn in der Schleife $(q', \epsilon, x) \overset{+}{\models} (q', \epsilon, x\alpha)$ ein akzeptierender Zustand auftritt) und einen ϵ-Übergang von q'' in den nichtakzeptierenden Fangzustand. ∎

Satz 38 *Die Klasse der determinitischen kontextfrein Sprachen (also der von DPDA's* erkannten *(nicht nur akzeptierten) Sprachen) [DCFL] ist unter Komplement abgeschlossen.*

11. Berechenbakeit, Entscheidbarkeit

11.0 Einleitung

- intuitiver Begriff der Berechenbarkeit
- formale Begriffe z.B.:
  - Turing-Berechenbarkeit
  - Markov-Algorithmen
  - λ-Kalkül
  - μ-rekursive Funktionen
  - Registermaschinen
  - [AWK, B, C, Euler, Fortran, Id, Lisp, Modula, Oberon, Pascal, Simula, ...]-Programme
  - while-Programme
  - goto-Programme u.v.a.m.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Church'sche These

Dieser formale Begriff der Berechenbarkeit stimmt mit dem intuitiven überein.

while-Programme:

- $x_i := x_j \pm 1$
- HALT
- **while** $x_i \neq 0$ **do**
  $\lceil$
  body
  $\lfloor$

goto-Programme

- $m_i : x_k := x_j \pm 1$
- m_i : HALT
- m_i : **if** $x_i \neq 0$ **then goto** m_j

Primitiv rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}^k \to \mathbb{N}$, für die gilt:

(i) Sie enthält die konstanten Funktionen

(ii) Sie enthält die Nachfolgerfunktionen: $n \mapsto n+1$

(iii) Sie enthält die Projektionsfunktionen:

$$\mathrm{proj}_{k,i} : \mathbb{N}^k \to \mathbb{N}; \qquad (x_1, \ldots, x_k) \mapsto x_i$$

(iv) Sie ist abgeschlossen unter Komposition

(v) Sie ist abgeschlossen unter primitiver Rekursion:

Mit $\mathrm{g} : \mathbb{N}^n \to \mathbb{N}$
$\mathrm{h} : \mathbb{N}^{n+1} \to \mathbb{N}$

ist auch

$f(0, y_1, \ldots, y_n) := \mathrm{g}(y_1, \ldots, y_n)$
$f(m+1, y_1, \ldots, y_n) := \mathrm{h}(f(m, y_1, \ldots, y_n), y_1, \ldots, y_n)$ (primitive Rekursion)

in der Klasse (und sonst nichts).

Satz 39 *Jede primitiv-rekursive Funktion ist total.*

Beweis: Induktion über den Aufbau einer primitiv-rekursive Funktion. ∎

Satz 40 *Jede primitiv-rekursive Funktion ist berechenbar.*

Beweis: Induktion über den Aufbau einer primitiv-rekursive Funktion. ∎

Korollar 40.1 *Die primitiv-rekursiven Funktionen sind eine* echte *Teilkasse der berechenbaren Funktionen*[1].

[1] Begründung: Es gibt nicht-totale berechenbare Funtkionen

Beispiele:

(i) read(n); **while true do** ;

berechnet die total undefinierte Funktion (ist berechenbar).

(ii) $f(n) := \begin{cases} 1 & \text{falls } n \text{ Anfang der Dezimalbruchentwicklung von } \pi \\ 0 & \text{sonst} \end{cases}$

$f(0) = 0,\ f(3) = 1,\ f(30) = 0,\ f(31) = 1,\ f(314159265) = 1$

f ist berechenbar.[2]

(iii) $g(n) := \begin{cases} 1 & \text{falls } n \text{ in der Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$

Es ist unbekannt, ob g berechenbar ist. [3]

(iv) $h(n) := \begin{cases} 1 & \text{falls} \geq n \text{ aufeinanderfolgende 0'en in der} \\ & \text{Dezimalbruchentwicklung von } \pi \\ \text{undefiniert} & \text{sonst} \end{cases}$

ist berechenbar.

(v) $k(n) := \begin{cases} 1 & \text{falls es unendlich viele Primzahlzwillinge gibt} \\ & (p \text{ und } p+2 \text{ sind Primzahlen}) \\ 0 & \text{sonst} \end{cases}$

ist berechenbar.[4] Entweder ist $k(n) := 0$ oder $k(n) = 1$; beide Funktionen sind berechenbar.

(vi) $l(n) := n$-te Ziffer in der Dezimalbruchzerlegung von π

ist berechenbar.

(vii) *Nicht* für jede reelle Zahl α ist $l_\alpha(n) = n$-te Ziffer in der Dezimalbruchentwicklung von α berechenbar. Dies folgt schon aus Mächtigkeitsgründen: es gibt abzählbar ($\aleph_0$) viele Programme, aber es gibt überabzählbar ($\aleph_1$) viele reelle Zahlen.[5]

Definition 11.0.1 *Sei $A \subseteq \Sigma^*$. Die* charakteristische *Funtktion χ_A von A ist*

$$\chi_A(w) = \begin{cases} 1 & \textit{falls } w \in A \\ 0 & \textit{falls } w \notin A \end{cases}$$

Satz 41 *A* rekursiv *$\Leftrightarrow \chi_A$ berechenbar.*

Beweis: Folgt aus der Definition von *rekursiv*: es gibt eine Turing Maschine, die `ja` oder `nein` liefert. Wandle das Ergebnis in 1 oder 0. ∎

[2] Es kann nie passieren, daß irgendwann nur noch 9er kommen ($3,14\ldots i\overline{9} = 3,14\ldots\langle i+1\rangle 0$), da π irrational ist.

[3] Wäre π vollkommen zufällig, was es aber nicht ist, dann würde jedes n irgendwann vorkommen.

[4] jedoch ein offenes Problem in der Zahlentheorie

[5] vgl. Cantorsches Diagonalisierungsprinzip

Definition 11.0.2 *Sei $A \subseteq \Sigma^*$. χ'_A ist definiert durch*

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{falls } w \notin A \end{cases}$$

Satz 42 *A rekursiv aufzählbar $\Leftrightarrow$ χ'_A berechenbar.*

Satz 43 *A rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ berechenbar ($\Leftrightarrow$ A und $\bar{A}$ r.a.)*

Zur bijektiven Abbildung von 2-Tupeln[6] natürlicher Zahlen in die Menge der natürlichen Zahlen verwendet man eine Paarfunktion, z.B.:

	0	1	2	3	4	⋯
0	0	2	5	9		
1	1	4	8			
2	3	7				
3	6					
⋮						

Betrachte: $p : \mathbb{N}^2 \to \mathbb{N}$ mit
$p(n_1, n_2) := \frac{(n_1+n_2)(n_1+n_2+1)}{2} + n_2$

$c_1 : \mathbb{N} \to \mathbb{N}$
$c_1(n) := s - (n - \frac{s(s+1)}{2})$; wobei
$s := \max\{i;\ \frac{i(i+1)}{2} \leq n\}$

$c_2 : \mathbb{N} \to \mathbb{N}$
$c_2(n) := n - \frac{s(s+1)}{2}$ s wie oben

Satz 44 *p stellt eine berechenbare, bijektive Paarfunktion von $\mathbb{N}^2$ nach $\mathbb{N}$ mit den Umkehrfunktionen c_1 und c_2 dar. (c_1, c_2 sind ebenfalls umkehrbar.)*

$$\begin{array}{ccccc} \aleph_0 & < & \aleph_1 & \leq & 2^{\aleph_0} \\ \| & & & & \| \\ |\mathbb{N}| & & & & |\mathbb{R}| \end{array}$$

μ-rekursive Funktionen

Die μ-rekursiven Funktionen sind definiert als Erweiterung der primitiv rekursiven Funktionen um die Abschlußeigenschaft unter dem μ-Operator, welcher wie folgt definiert ist:

$$\mu_g(y_1, \ldots, y_m) := \min\{n;\ g(n, y_1, \ldots, y_m) = 0\ \wedge\ \forall n' < n : g(n', y_1, \ldots, y_m) \text{ definiert}\}$$

Der μ-Operator läßt sich durch eine einzige `while`-Schleife implementieren.

[6] bzw. n-Tupel bei iterierter Anwendung der Paarbildung

11.1 Aufzählbarkeit

Satz 45 *Eine Sprache $A \subseteq \Sigma^*$ ist rekursiv aufzählbar (r.a.), falls $A = \emptyset$ oder es eine totale berechenbare Funktion*

$$f : \mathbb{N} \to \Sigma^* \text{ gibt, so daß gilt}: \; A = f(\mathbb{N}) \quad (A = \{f(0), f(1), \ldots\})$$

Beweis: Per Definition ist A rekursiv aufzählbar, falls $A = L(M)$ für eine Turing Maschine M also falls χ'_A berechenbar ist, ohne Beschränkung der Allgemeingeit sei $A \subseteq \mathbb{N}$.

„$\Leftarrow$“ zu zeigen $\exists\, f$ wie oben $\Rightarrow \chi'_A$ berechenbar

```
input(n);
for (m = 0; true ;m + +) do
    if n = f(m) then print “1“; exit
    fi
od
```

Dieses Programm berechnet offensichtlich χ'_A

„$\Rightarrow$“ zu zeigen χ'_A berechenbar $\Rightarrow \exists\, f$ mit obigen Eigenschaften. Sei $A \neq \emptyset$, $A = L(M)$, $a \in A$

```
input(n);
x := c_1(n); m := c_2(n);
if M akzeptiert x in genau m Schritten
  then print x
  else  print a
fi
```

Dieses Prinzip der verzahnten Rechnung nennt man *Dove-Tailing.* Die Rechnung von M auf x terminiert im allgemeinen nicht unbedingt, deshalb werden immer nur eine feste Anzahl von Rechenschritten erlaubt.

Ein ähnliches Prinzip wird beim Absuchen eines Baumes mit der Breadth-First , anstelle der Depth-First, Strategie verwendet. ▮

Satz 46 *Die folgenden Aussagen sind äquivalent:*

(i) A ist rekursiv aufzählbar

(ii) A ist Chomsky-0

(iii) $A = L(M)$ für Turing Maschine M

(iv) χ'_A *ist berechenbar*

(v) $A = \mathcal{D}(f)$, f *berechenbar, A ist also Definitionsbereich einer berchenbaren Funktion.*

(vi) $A = \mathcal{W}(f)$, f *ist berechenbar und total*[7]

[*(vii)* A *ist semi-entscheidbar*]

Bemerkung:

Eine Menge L heißt abzählbar, falls $L = \emptyset$ oder es eine totale Funktion $f: \mathbb{N} \to L$ gibt mit $f(\mathbb{N}) = L$

Unterschied: aufzählbar $\leftrightarrow$ abzählbar

Aufzählbare Mengen sind mechanisch aufzählbar; abzählbare Menegen nicht unbedingt.

Beispiel:

Es gibt Teilmengen der natürlichen Zahlen (die sicherlich abzählbar sind), die aber nicht aufzählbar sind.

Der Grund: es gibt mehr Teilmengen von $\mathbb{N}$ als Programme. Anzahl Teilmengen von $\mathbb{N}$: $2^{|\mathbb{N}|} = |\mathbb{R}|$, und die reellen Zahlen sind nicht aufzählbar[8].

11.2 Gödelisierung von Turingmaschinen

Kurt Gödel erarbeitet in den 30iger Jahren grundlegende Ergebnisse in der Berechenbarkeit. Unter anderem den Gödel'schen Unvollständigkeitssatz: *Die Theorie der natürlichen Zahlen ist unentscheidbar.*

Turing Maschinen können durch Binärzeichenreihen und damit Zahlen aus $\mathbb{N}$ dargestellt werden. Wir unterlegen eine einfache Syntax und verlangen, daß es einfach ist zu entscheiden, ob ein $w \in \mathbb{N}$ eine Turing Maschine bezeichnet oder nicht.

M_0 sei irgendeine (fest gewählte) Turing Maschine. $\mathrm{BIN} = L(0 + 1(0+1)^*) \hat{=} \mathbb{N}$

Definiere für $w \in \mathrm{BIN}$

$$M_w := \begin{cases} \text{TM } M & \text{falls } w \text{ die TM } M \text{ bezeichnet} \\ M_0 & \text{sonst} \quad \text{(Default TM)} \end{cases}$$

[7] $\mathcal{W}$ ist der Wertebereich

[8] siehe Diagonalisierungsprinzip von Kantor

Definition 11.2.1 *Die Abbildung*

$$\begin{matrix}\mathbb{N}\\ BIN\end{matrix} \ni w \mapsto M_w$$

heißt Gödelisierung. w heißt Gödelnummer der Turing Maschine M_w

11.3 Unentscheidbarkeit

Definition 11.3.1 *Das* spezielle Halteproblem *(Selbstanwendbarkeitsproblem) ist*

$$K = \{w \in BIN,\ M_w \text{ angesetzt auf } w \text{ hält}\}$$

Satz 47 *K ist* nicht *rekursiv (entscheidbar)*

Beweis: Annahme χ_K ist berechenbar, mittels der Turning Maschine M. Konstruiere aus M eine TM M', die sich zuerst wie M verhält und dann, wenn M mit Ausgabe 0 oder 1 hält sich so verhält:

M hält mit $0 \Rightarrow M'$ hält
M hält mit $1 \Rightarrow M'$ geht in Endlosschleife

Sei w' die Gödelnummer für M'.

Dann gilt: $w' \in K \Leftrightarrow M'$ auf w' angesetzt hält
$\Leftrightarrow M$ auf w' angesetzt gibt 0 aus
$\Leftrightarrow \chi_K(w') = 0 \Leftrightarrow w' \notin K$

sl Widerspruch!

Also ist χ_K nicht berechenbar. ∎

Definition 11.3.2 *Seien $A, B \subseteq \Sigma^*$. Dann heißt A* (effektiv) reduzierbar *auf B (z.B. $A \subseteq B$) gdw. $\exists f : \Sigma^* \to \Sigma^*$, f total und berechenbar mit* $(\forall w \in \Sigma^*)[w \in A \Leftrightarrow f(w) \in B]$ $(f(A) \subseteq B,\ f(\bar{A}) \subseteq \bar{B})$

$$A \underset{f}{\hookrightarrow} B$$

Satz 48 *Sei $A \underset{f}{\hookrightarrow} B$*

(i) B rekursiv $\Rightarrow$ A rekursiv

(ii) B rekursiv aufzählbar $\Rightarrow$ A rekursiv aufzählbar

Beweis:

(i) $\chi_A = \chi_b \circ f$

(i) $\chi'_A = \chi'_b \circ f$

∎

Definition 11.3.3 *Das* Halteproblem *ist*

$$H = \{w \in \; BIN;\ M_{c_1(w)} \text{ angesetzt auf } c_2(w) \text{ hält}\}$$

Satz 49 *H ist nicht entscheidbar (nicht rekursiv).*

Beweis: Betrachte $f: \text{BIN} \ni w \mapsto p(w,w) \in \text{BIN}$. Dann $K \underset{f}{\hookrightarrow} H$. K unentscheidbar $\Rightarrow$ H unentscheidbar. ∎

Definition 11.3.4 *Das* Halteproblem auf leerem Band H_0 *ist*

$$H_0 = \{w \in \; BIN;\ M_w \text{ hält auf leerem Band}\}$$

Satz 50 *H_0 ist unentscheidbar (nicht rekursiv).*

Beweis: Betrachte die Abbildung f, die definiert ist durch:

$$\text{BIN} \ni w \mapsto f(w),$$

$f(w)$ ist die Gödelnummer einer *TM*, die auf leerem Band angesetzt, zunächst $c_2(w)$ auf das Band schreibt und sich dann wie $M_{c_1(w)}$ (angesetzt auf $c_2(w)$) verhält. Falls das Band nicht leer ist, ist es unerheblich wie sich $M_{f(w)}$ verhält.

f ist total und berechenbar.

Es gilt: $$\begin{aligned} w \in H \quad & \Leftrightarrow M_{c_1(w)} \text{ angesetzt auf } c_2(w) \text{ hält.} \\ & \Leftrightarrow M_{f(w)} \text{ hält auf leerem Band} \\ & \Leftrightarrow f(w) \in H_0 \end{aligned}$$

also $H \underset{f}{\hookrightarrow} H_0$ und damit H_0 unentscheidbar. ∎

Bemerkung

Es gibt also keine allgemeine algorithmische Methode um zu entscheiden, ob ein Programm anhält.

Satz 51 (Rice) *Sei R die Menge aller (TM)-berechenbaren Funktionen und S eine nichttriviale Teilmenge von R (also $\mathcal{S} \neq \mathcal{R}$, $\mathcal{S} \neq \emptyset$). Dann ist*

$$G(\mathcal{S}) := \{w \in \; BIN;\ \text{die von } M_w \text{ berechnete Funktion ist in } S\}$$

unentscheidbar.

Beweis: [9] Sei Ω die total undefinierte Funktion

1.Fall: $\Omega \in S$

Sei $q \in \mathcal{R} - S$, [10] Q eine TM für q.

Zu $w \in$ BIN sei $f(w) \in$ BIN Gödelnummer einer TM, die:

1. bei Eingabe x ignoriert $M_{f(w)}$ diese zunächst und verhält sich wie M_w auf leerem Band.
2. wenn obige Rechnung hält, dann verhält sich $M_{f(w)}$ wie Q auf einer Eingabe x.

f ist total und berechenbar.

$w \in H_0 \Leftrightarrow M_w$ hält auf leerem Band.
$\Leftrightarrow M_{f(w)}$ berechnet die Funktion $q(\neq \Omega)$
$\Leftrightarrow$ die von $M_{f(w)}$-berechnete Funktion ist nicht in S.
$\Leftrightarrow f(w) \notin G(S)$

Also: $\bar{H}_0 \underset{f}{\hookrightarrow} G(S)$

H_0 unentscheidbar (nicht rekursiv) $\Rightarrow \bar{H}_0$ unentscheidbar $\Rightarrow G(S)$ unentscheidbar.

2.Fall: $\Omega \notin S$

Sei $q \in S$, Q, f wie Fall 1

$w \in H_0 \Leftrightarrow M_{f(w)}$ berechnet die Funktion $q(\neq \Omega)$
$\Leftrightarrow$ die von $M_{f(w)}$ berechnete Funktion ist in S.
$\Leftrightarrow f(w) \in G(S)$

Also: $H_0 \underset{f}{\hookrightarrow} G(S)$

H_0 unentscheidbar $\Rightarrow G(S)$ unentscheidbar.

∎

11.4 Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wir wissen : reguläre Sprachen sind unter allen Bool'schen Operationen abgeschlossen.

und : kontextfreie Sprachen sind *nicht* unter Komplement und Durchschnitt abgeschlossen.

[9] Wir zeigen hier nur diesen Satz. Setzt man weitere Eigenschaften von S voraus, kann man sogar zeigen, daß $G(S)$ nicht einmal rekursiv aufzählbar ist.

[10] es gibt ein derartiges q, da S nichttrivial

Wir wollen wissen: können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

Wir werden sehen: nein!

Sie M eine (beliebige) TM (mit nur einem Band) mit Bandalphabet Σ, Zustandsmenge Q, sei $\# \notin \Sigma \cup Q$.

Die Sprachen

$Conf_M^{(0)} :=$ $\{c_0\#c_1^R\#c_2\#c_3^R \ldots c_{2m+1}^R;\quad m \geq 0,\ c_i$ ist Konfiguration von M, c_0 ist Anfangskonfiguration auf leerem Band, c_{2m+1} ist Endkonfiguration oder $c_{2m} = c_{2m+1}$ und c_{2m} ist Endkonfiguration, und c_{2j+1} ist Nachfolgekonfiguration von c_{2j} für alle j $\}$

Also: $Conf_M^{(0)}$ enthält nicht nur „echte" Rechnungen von M, da $c_{2j-1} \rightarrow c_{2j}$ nicht unbedingt ein Schritt sein muß; das fordern wir nur für $c_{2j} \rightarrow c_{2j+1}$

Deshalb symmetrisch dazu:

$Conf_M^{(1)} :=$ $\{c_0\#c_1^R\#c_2\#c_3^R \ldots c_{2m+1}^R;$ wie oben jetzt: c_{2j} ist Nachfolgekonfiguration von c_{2j-1} für alle zutreffenden $j \geq 1\}$

sind deterministisch kontextfrei. Wir konstruieren jeweils einen DPDA dafür.

Hinweis: ein Kellerautomat ist lange nicht soviel wie eine Turingmaschine. Aber zwei Kellerautomaten (oder eine endliche Kontrolle mit zwei Kellern) sind so mächtig wie eine Turingmaschine.

Dann gilt:

$$w \in H_0 \Leftrightarrow Conf_{M_w}^{(0)} \cap Conf_{M_w}^{(1)} \neq \emptyset$$ [11]

Satz 52 *Das Schnittproblem für kontextfreie Sprachen (entscheide ob der Schnitt zweier kontextfreier Sprachen leer ist) ist unentscheidbar!*

Beweis: siehe oben ∎

[11] falls M_w deterministisch ist, enthält $Conf_{M_w}^{(0)} \cap Conf_{M_w}^{(1)}$ genau ein Element, nämlich die eine Rechnung von M_w auf leerem Band.

Wir haben sogar gezeigt: Das Schnittproblem für *deterministisch* kontextfreie Sprachen ist unentscheidbar[12].

In der Literatur wird dieser Satz üblicherweise mit dem Post'schen Korrespondenzproblem (PCP) bewiesen.

Post'sches Korrespondenzproblem

Gegeben: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ mit $x_i, y_i \in \Sigma^+$

Frage : gibt es eine Folge von Indizes $i_1, i_2, \ldots i_r \in \{1, \ldots, n\}$ so daß $x_{i_1} x_{i_2} \ldots x_{i_r} = y_{i_1} y_{i_2} \ldots y_{i_r}$

Satz 53 *PCP ist unentscheidbar.*

[12]da wir gezeigt haben, daß sich das Halteproblem auf das Schnittproblem zweier deterministisch kontextfreier Sprachen reduzieren läßt

12. Primitive Datenstrukturen

12.0 Grundlegende Datentypen

- boolean
- nat
- int
- float
- char
- string
- pointer
- struct
- array

Dazu:

- Listen
- Mengen
- Relationen (Menge von Tupeln)
- Funktionen, Abbildungen

12.0.1 Listen

Eine Liste ist eine Folge beliebiger Elemente.

$q = [x_0, x_1, \ldots, x_{n-1}]$, $|q| = n$ (Länge der Liste)
$q = []$ (leere Liste)

Operationen:

- isinlist
- insert
- delete
- sublist
- concatenate

12.0.2 Typen von Listen

(i) Warteschlange, Queue:

Einfügen nur an einem, Löschen am anderen Ende. Diese Prinzip bezeichnet man als FIFO (first-in-first-out).

(ii) Keller, Stack:

Einfügen und Löschen nur an einem Ende der List. LIFO (last-in-first-out)

(iii) Deque:

Einfügen und Löschen an beiden Enden.

12.0.3 Darstellung von Listen

12.0.4 Mengen

- Aufzählung, Listen z.B.: (Sun, Mon, Tue, Wed, Thu, Fri, Sat)
- Intervale: i, $0 \leq i \leq n$
  $i \in [k, j]$
- Prädikate $i \in \mathbb{Z}$, $i \geq 1729$
- Relationeen, Funktionen, Graphen

  Dartstellung mittles:

  - charakteristischen Funktion (Bitvektor)
  - Liste der Elemente
  - Glechte der Elemente
- geordnete Mengen

12.0.5 Graphen

$G = (V, E)$ V Knotenmenge
E Kantenmenge, $E \subseteq V \times V$

Begriffe:

- gerichtet / ungerichtet
- einfacher Graph (keine Schlingen, keine Mehrfachkanten)
- Pfad
  Folge $v_0, v_1, \ldots, v_{r-1}$ von Knoten $(v_i, v_{i+1}) \in E \quad i = 0, \ldots, r - l$.
- (einfacher) Kreis (Zykel)
- azyklisch
- zusammenhängend $(\forall v, w \in V)\exists$ Pfad von v nach w
- Zusammenhangskomponenten
  $x \equiv_c y \Leftrightarrow \exists$ Pfad von x nach y
  Zeige: $\equiv_c$ ist Äquivalenzrelation
  Zusammenhangskomponenten sind Äquivalenzklassen.
- starke Zusammenhangskomponenten
  $x \equiv_{sc} y \Leftrightarrow \exists$ gerichteter Pfad von x nach y einer von y nach x
  Zeige: $\equiv_{sc}$ ist Äquivalenzrelation
  Die Äquivalenzklassen unter $\equiv_{sc}$ sind starke Zusammenhangskomponenten.

12.0.6 Darstellung von Graphen

- Adjazenzmatrix
- Inzidenzmatrix
- Adjazenzlisten

12.1 Hash-Tabellen, Streuspeicherung

Wir haben ein Universum U aus dem unsere Daten stammen.

Berabeiten wollen wir eine Teilmenge $S \subseteq U$, $|S| = n$, S beliebig aber relativ klein zu U. S bezeichne also die Menge von Schlüsseln, die die Datensätze eindeutig bezeichnet. Wir wollen ein Wörterbuch (dictionary) für diese Schlüssel konstruieren.

Wörterbuch (Dictionary) ist ein Datentyp der folgende Operationen unterstützt:

- `is_elt_of`
- `insert`
- `delete`

Realisierung von Dictionary

Hashtabelle: $T[0,\ldots,m-1]$

dabei soll $m = O(n)$ sein, damit nicht zuviel Platz verschenkt wird.

Hashfuktion: $h : U \to \{0,\ldots,m-1\}$

Da $|U| >> m$ ist h nicht injektiv, also gibt es x, y mit $h(x) = h(y)$ dies nennt man auch *Kollision.*

: Füllfkator $\alpha := \frac{n}{m}$

Ziel bei der Realisierung ist es, eine besonders schnellen Zugriff auf einzelen Datensätze zu erreichen. h soll sich also „leicht" berrechnen lassen. (z.B. $O(1)$)

12.1.1 Geschlossenes Hashingverfahren

Beim geschlossenen Hashing erfolgt die Kollisionsauflösung durch Verkettung. D.h. jedes $T[i]$ ist der Anker einer verketten Liste welche alle Elemente k_j enthält für die gilt $h(k_j) = i$.

Falls eine Ordnung auf U bekannt ist kann der Zugriff noch weiter beschleunigt werden, indem wir die Listen sortiert abspeichern.

Für die Analyse nehmen wir an, daß alle Daten aus U gleich wahrscheinlich sind. Die Hashfunktion liefert für zufällig gewählte Daten aus U die Werte aus $0,\ldots,m-1$ mit gleicher Wahrscheinlichkeit.

Die folgende Aussage über die Laufzeit für erflogreiche bzw. erfolglose Suche gilt asymptotisch für $n, m \to \infty$ und $n >> m$.

Satz 54 *Sei $\alpha = \frac{n}{m}$ der Füllgrad der Hastabelle und gelte, daß jedes EH jede Position der Hashtabelle mit gleicher Wahrscheinlichkeit trifft. Dann sind die durchschnittlichen Kosten für erfolgreiche bzw. erfolglose Suche $\Theta(1+\alpha)$ und für Einfügen und Löschen $\Theta(1)$.*

Beweis: Kosten:

	h ausrechnen	$\Theta(1)$
	erfolgreiche Suche	im Mittel muß die halbe Liste durchsucht werden
	erfolglose Suche	die ganze Liste muß durchsucht werden

$\Rightarrow T_{AC} = \frac{1}{m}\sum_{i=1}^{m} l_i(\frac{1}{2})$ mit l_i ist Länge der Liste i
$= \Theta(1) \cdot (\frac{n}{m}(\frac{1}{2})) = \Theta(1+\alpha)$ ∎

12.1.2 Offenes Hashing

Alle Schlüssel werden in der Hashtabelle gespeichert.

Vorteile:	keine Pointer
Nachteile:	Kollisionsauflösung wird schwerer

12.1.3 Hashfunktionen

Ideale Hasbedingung, falls Schlüssel $k \in U$ mit Wahrscheinlichkeit $p(k)$ vorkommet:

$$\forall j \in \{0, \ldots, m-1\} : \sum_{h(k)=j} p(k) = \frac{1}{m}$$

Problematisch ist, daß p im allgemeinen unbekannt ist.

Gewöhnlich werden die Schlüssel als Element as $\mathbb{N}$ aufgefaßt.

Beispiel für Hashfunktionen:

(a) Teilermethode $h(k) = k \bmod m$ Kriterien für m

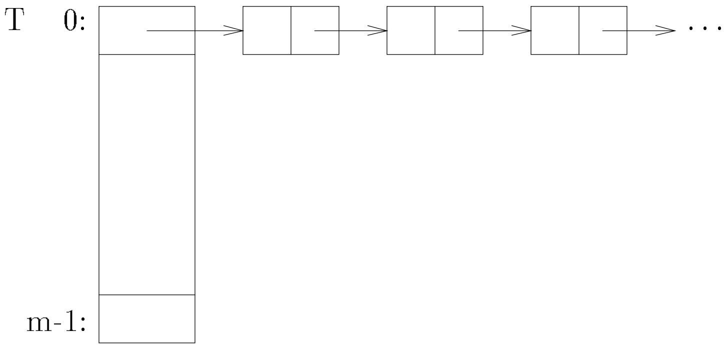
- keine Zehnerpotenz
- keine Zweierpotenz
- nicht con der Form $2^k - 1$

Eine gute Wahl für m ist: m Primzahl und nicht nahe an einer Zweierpotenz.

(b) Multiplikationsmethode[1]
Sei $0 < a < 1$ konstant.
$h(k) = \lfloor m(ka \bmod 1) \rfloor$ $a = \frac{\sqrt{5}-1}{2} \approx 0.6180339887$
$(= \frac{1}{\varphi}$ wobei $\varphi = \frac{1+\sqrt{5}}{2}$ goldener Schnitt

[1]Vgl.: Knuth „The art of programming Vol.3“

12.1.4 Universelle Hashfunktion



Definition 12.1.1 *Eine Familie* χ *con Hashfunktionen heißt* universell*, falls*

$$\forall x, y \in U,\, x \neq y \quad |\{h \in \chi | h(x) = h(y)\}| = \frac{|\chi|}{m}$$

Beobachtung:

h zufällig aus χ. Bei festem x und $n \leq m$ ist die erwartete Anzahl von Kollisionen mit x $\leq \frac{n-1}{m} < 1$.

Seien nun die Schlüssel aus $\{0, \ldots, m-1\}^{V+1}$, m Primzahl

$$\chi := \{h | h: \{0, \ldots, m-1\}^{V+1} \ni (x_0, \ldots, x_r) \mapsto \sum_{i=0}^{r} a_i x_i \bmod m \in \{0, \ldots, m-1\}$$
$$a_0, \ldots a_r \in \{0, \ldots, m-1\}\}$$

Satz 55 χ *ist universell*

Beweis: Sei $x = (x_0, \ldots, x_r)$, $y = (y_0, \ldots y_r)$ $x \neq y$ ohne Beschränkung der Allgemeinheit $x_0 = y_0$

Für beliebige, aber feste $a_1 \ldots a_r$ gilt:

Es gibt genau ein $a_0 :\ h(x) = h(y)$:

$$\sum_{i=0}^{r} a_i x_i = \sum_{i=0}^{r} a_i y_i \bmod m \Rightarrow$$

$$a_0(x_0 - y_0) = \sum_{i=1}^{r} q(y_1 - x_i) \bmod m$$

$\Rightarrow$ Für gegeben $a_1 \ldots a_v$ gibt es genau ein $a_0 \in \{0, \ldots m_0 - 1)$ daß die Kollision liefert als $\frac{1}{m}$-tel ∎

12.1.5 Kollisionsauflösung bei offenem Hashing

Erweiterte Hashfunktion:

$h(k,i) =$ Hashadresse für Schlüssel h nach $(i-1)$ Kollisionen

Probiere $h(k,1)$, $h(k,2)$, $h(k,3), \ldots$ bis freie Position gefunden wird. (Füge dann ein)

Methoden zur Kollisionsauflösung

(a) Lineares Sondieren

$$h(k,i) = (h(k) - i + 1) \bmod m$$

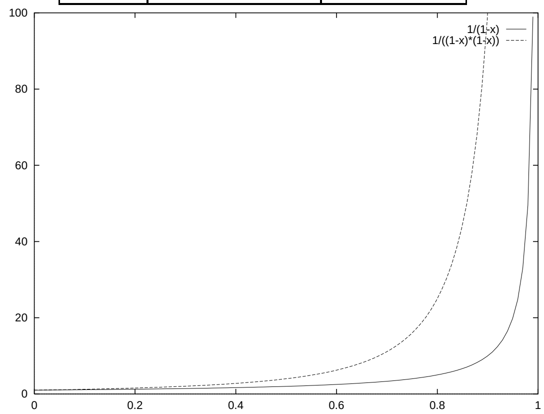
problemtisch: primäre Häufung.

Wahrscheinlichkeitsthoretische Analyse ergibt (ohne Beweis):

$$\mathbb{E}(\#\text{ Sondierungen bei erfolgloser Suche}) \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

$$\mathbb{E}(\#\text{ Sondierungen bei erfolgreicher Suche}) \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

α	erfolgreich	erfolglos
0.5	1.5	2.5
0.9	5.5	50.5
0.95	10.5	200.5



(b) Quadratisches Sondieren

$$h(k,i) = (h(k) + c_2 i^2 + c_1 i) \bmod m$$

c_1, c_2 müssen so gewählt sein, daß für $i = 0, \ldots, m-1$ alle Adressen der Hashtabelle erreicht werden. Zum Beispiel $h(k,i) = h(k) + (-1)^i \left(\left\lceil \frac{i}{2} \right\rceil^2\right)$

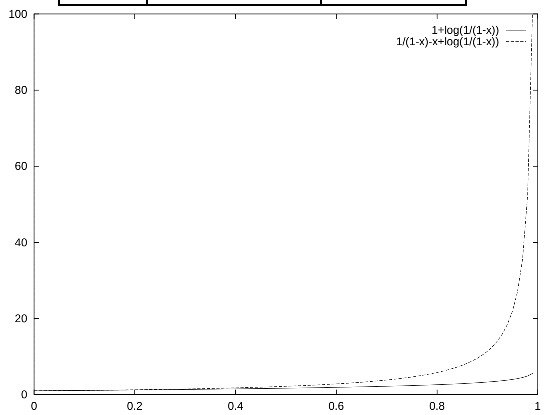
Problem bei der quadratischen Sondierung ist die sekundären Häufung. Wenn einmal zwei Schlüssel h, h' auf das selbe Element gehasht werden, dann werden auch die iterierten (wachsendes i) auf dieselbe Adresse gehasht.

Analyse:

$$\mathbb{E}(\#\text{ Sondierungen bei erfolgloser Suche}) \approx \frac{1}{1-\alpha} - \alpha + \ln\left(\frac{1}{1-\alpha}\right)$$

$$\mathbb{E}(\#\text{ Sondierungen bei erfolgreicher Suche}) \approx 1 + \ln\left(\frac{1}{1-\alpha}\right)$$

α	erfolgreich	erfolglos
0.5	1.44	2.19
0.9	2.85	11.4
0.95	3.52	22.05



(b) Double-Hashing

$$h(k,i) = (h(k) + i \cdot h'(k)) \bmod m$$

$$h'(k) \text{ relativ prim zu } m$$

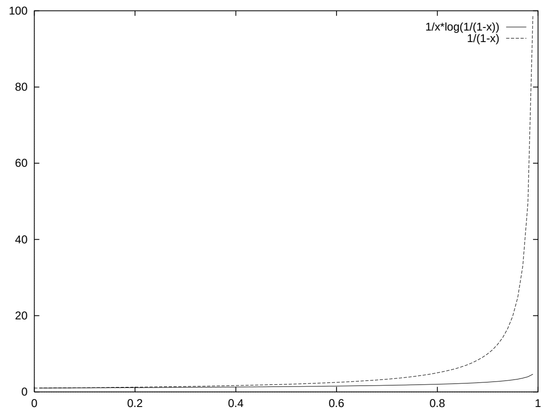
$$h'(k) = 1 + k \bmod (m-2) \quad m \text{ prim}$$

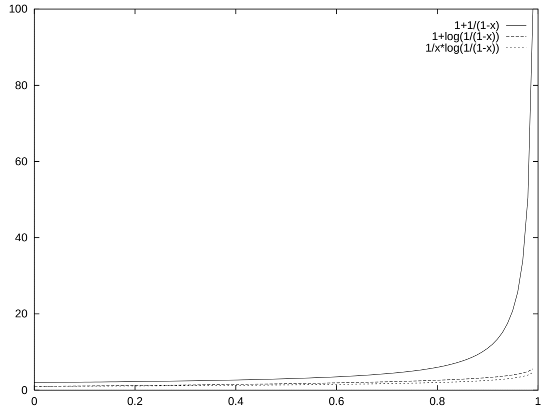
Analyse:

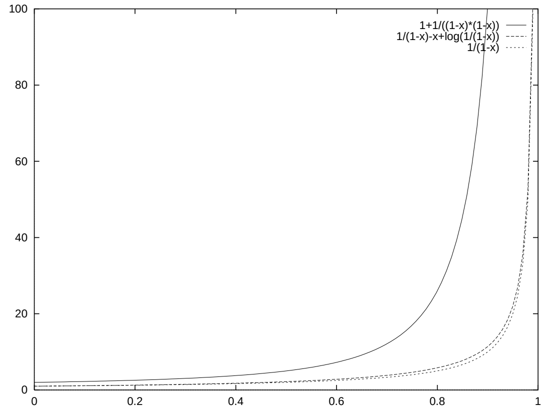
$$\mathbb{E}(\#\text{ Sondierungen bei erfolgloser Suche}) \approx \frac{1}{1-\alpha}$$

$$\mathbb{E}(\#\text{ Sondierungen bei erfolgreicher Suche}) \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

α	erfolgreich	erfolglos
0.5	1.39	2
0.9	2.56	10
0.95	3.15	20







12.2 Sortieren

Wir betrachten ein Universum U mit totaler Ordnung.

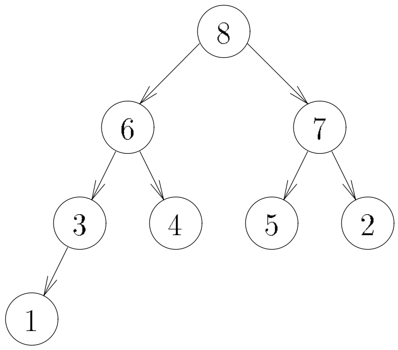
Wir betrachten (Binär)-Bäume, bei denen jedem Konten u ein Schlüssel $key(u) \in U$ zugeordnet ist.

12.2.1 Heapsort

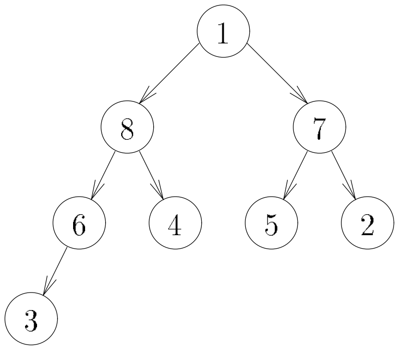
Definition 12.2.1 *Ein Baum erfüllt die* Heap-Bedingung *falls gilt: u Vorgänger von v*

$key(u) \geq key(v)$

Beispiele



Bemerkung: In der Wurzel steht das größte Element.



erfüllt *nicht* die Heap-Bedingung.

Wie kann man die Heap-Bedingung wiederherstellen?

Suche größten Schlüssel unter den Kindern und vertauschen diesen Schlüssel mit dem der Eltern. Anfangs war die Heap-Bedingung nicht erfüllt verletzt das Kind die Heap-Bedingung, das am Tausch der Eltern beteiligt war. Dies kann man forführen bis der Baum wieder die Heap-Bedingung erfüllt.

Sei H (Binär)Baum mit Wurzel v, so daß die Unterbäume, die zu den Kindern von v gehören, die Heap-Bedingung erfüllen.

Wiederherstellen der Heap-Bedingung:

procedure versickere(v);
    while true do
            a :=Kind von v mit einem maximalen Schlüssel
                (bzw. NIL, falls v kein Kind hat)
            if v =nil **then exit fi**
            if $key(v) < key(u)$ **then**
                vertausche $key(v)$ und $key(u)$
                $v := u$;

```
        else
          exit
        fi
    od
```

Lemma 12.2.1 *(i) Die Procedure versickere stellt unter den oben angegebenen Voraussetzungen die Heap-Bedingung in dem Baum mit Wurzel v wieder her.*

(ii) Der Zeitbedarf für versickere(v) ist O(Tiefe des Baumes mit Wurzel v).

Beweis: Falls Heap-Bedingung am Knoten v verletzt wird durch Austauschen das Elemnt in v größer als alle seine Kinder. Höchstens an dem Kindknoten, der an der Vertauschung beteiligt war, ist die Heap-Bedingung verletzt. Darus folgt (i) und (ii). ∎

Das Heapsort Verfahren besteht aus zwei Teilen.

(i) Konstruktion eines Baumes aus den gegebenen Schlüsseln, welcher die Heap-Bedingung erfüllt.

(ii) Entferne größtes Element aus der Wurzel und ersetze dieses durch ein Element eines tiefsten Blattes und lösche dieses Blatt. Stelle die Heap-Bedingung mit Hifle von *versickere* wieder her.

```
procedure g − heapsort(v)
    co wir nehmen an, die Daten sind Knoten
        eines (Binär)Baumes zugeordnet oc
    for alle inneren Knoten v' im Baum mit Wurzel v
          in umgekehrter topologischer Reihenfolge do
          co Knoten v wird erst behandlet, wenn die Knoten
              bereits abgearbeitet wurden oc
          versickere(v')
    od
    n := #Knoten;

    for i := n downto 1 do
          output(key(v)); co max. Elt. an der Wurzel oc
          if i > 1 then
             b := Blatt maximaler Tiefe;
             vertausche key(b) mit key(v);
             lösche Blatt b;
             versickere(v); co Heap-Bed. wiederherstellen oc
          fi
    od
```

Satz 56 *$g - heapsort$ gibt die Schlüssel in absteigend sortierter Folge aus.*

Beweis: Induktion über n (Anzahl der Elemente)

I.A.: $n = 1$: offensichtlich

I.S.: $(n-1) \to n$:

Zuerst wird das maximale Element ausgegeben und dann wird der Rest naxh I.V. in absteigender Folge ausgegeben

∎

Zeitbedarf:

Wir implementieren den Baum als vollständig linksbalazierten Binärbaum (z.B.) in Arraydarstellung. Statt Schlüssel $key(v)$ auszugeben, speichern wir ihn im rechtesten Blatt maxiamler Tiefe ab und schlißen dieses Blatt aus dem Baum aus.

Bei Arraydarstellung steht dann zum Schluß die Folge *aufsteigende* sortiert im Array!

Der Zeitbedarf für $versickere(v')$ falls der Unterbaum von v' die Höhe(Tiefe) h hat ist $O(h)$.

Bei einem vollständig linksbalanzierten Binärbaum mit n Knoten gibt es $\leq \left\lceil 2^{\frac{n}{h+1}} \right\rceil$ Knoten der Höhe h (Höhe eines Knotens ist die Höhe des zugehörigen Unterbaums). Wir werden statt $\left\lceil 2^{\frac{n}{h+1}} \right\rceil$ $O(2^{\frac{n}{h+1}})$ verwenden.

$\Rightarrow$ Zeitbedarf für eine Schleife („heapity“):

$\leq c \cdot \sum_{h=0}^{\lceil \log(n+1) \rceil} h \cdot \left\lceil 2^{\frac{n}{h+1}} \right\rceil$

$$O(n \underbrace{\sum_{h=0}^{\lceil \log(n+1) \rceil} \frac{h}{2^h}}_{\substack{\text{ist konvergent} \\ \text{als } \infty\text{-Reihe} \\ \text{also konstant}}}) = O(n)$$

Also „heapity“: linear Kosten!. Nichtsdesto trotzt lohnt es sich den genauen Wert des konstanten Faktors auszurechnen.

Kosten für i-Schleifen:

$$O(n \cdot \log n)$$

Satz 57 *Die worst-case Komplexität von Heapsort ist $O(n \log n)$.*

Der Konstante Anteil kann noch gedrückt werden $\to$ Bottom-up-heap-sort.

12.2.2 Untere Schranken

Wir betrachten *vergleichsbasierte* Sortierverfahren (Schlüssel können nur in binärem Vergleich ($k \leq k'$) verwendet werden).[2]

Seien n Schlüssel gegeben und alle ohne Beschränkung der Allgemeinheit verschieden.

Entscheidungsbaum:

Wir betrachten für einen bestimmten Algorithmus, welche Schlüssel er beim Baluaf vergleicht.

Die Folge von Vergleichen für unterschiedliche Antworten auf Ziehungen ergibt einen Biärbaum. Jedes Blatt entspricht einer permutation π aus S_n, die die Eingabe daher in sortierter Reihenfolge liefert.

Beobachtung:

Jede Permutation $\pi \in S(n)$ muß an irgendeinem Blatt dieses Binärbaums vorkommen, ansonsten gebe es eine Folge von Eingabedaten, für die der Algorithmus nicht korrekt sortiert.

$$\Rightarrow \text{ Anzahl der Blätter } ge|S_n| = n!$$

Lemma 12.2.2 *(a) Maximale Tiefe eines Blattes in einen Binärbaum mit l Blättern ist $\geq \log_2(l)$*

(b) Ebeneso ist die mittlere Tiefe eines Blattes in einem Binärbaum mit l Blättern $\geq \log_2(l)$.

Beweis: Es genügt (b) zu zeigen.

Induktion über l

I.A.: $l = 1$

[2] Buchet- oder Radix-SOrt sind *nicht* vergleichsbasiert. Heapsort, Quicksort sind vergleichsbasierte Verfahren.

mit $l_1 + l_2 = l$
die Fälle $l_1 = 0$ und $l_2 = 0$ können ausgeschlossen werden.

I.S.:

$$
\begin{aligned}
&MT(l) \geq \tfrac{l_1}{l}(MT(l_1)+1) + \tfrac{l_2}{l}(MT(l_2)+1) \\
&\overset{\text{I.V.}}{\geq} \tfrac{l_1}{l}\log(l_1) + \tfrac{l_2}{l}\log(l_2) + 1 \\
&= \tfrac{1}{l}\underbrace{(l_1 \log l_1 + l_2 \log l_2)}_{\substack{\text{nimmt Max an} \\ \text{wenn } l_1 = l_2}} + 1 \\
&\geq \tfrac{1}{l}\left(\tfrac{l}{2}\log\left(\tfrac{l}{2}\right) + \tfrac{l}{2}\log\left(\tfrac{l}{2}\right)\right) + 1 \\
&= \log l - 1 + 1 \\
&\log l
\end{aligned}
$$

∎

Satz 58 *Jeder vergleichsbasiert Sortieralgorithmus hat durchschnittliche (avarage-case) Zeitkomplexität (in der Anzahl der Vergelichsoperationen)* $\Omega(n \log n)$

Beweis:

$$
\begin{aligned}
&\text{mittlere Zeit} \geq \text{mittlere Anzahl der Vergleiche} \\
&\geq \text{mittlere Tiefe eines Binärbaums mit } n! \text{ Blätter} \\
&\overset{\text{Lemma}}{\geq} \log(n!) \geq \log\left(\tfrac{n}{e}^n\right) \\
&\geq \Omega(n \log n)
\end{aligned}
$$

∎

12.3 Suchbäume

Geg.: Universum U mit totaler Ordung $\leq$

Ges.: Datenstruktur Wörterbuch (dictionary) mit den Operationen:

(i) `IsElt`

(ii) `Insert`

(iii) `Delete`

Binärer Suchbaum

Wir unterscheiden zwei Arten:

(a) Schlüssel nur an internen Knoten[3]

[3] Wird im folgenden angenommen. Bei der Darstellung von Bäumen können die Blätter dann auch weggelassen werden.

(b) Schlüssel nur an Blättern

Sortierungsbedingung

Für alle Knoten u gilt: Alle Schlüssel im linken (rechten) Unterbaum von u sind kleiner (größer) als der Schlüssel von u.[4]

12.3.1 Natürliche binäre Suchbäume

(i) `IsElt`

vergleiche mit Knoteninhalt u:

- $x = u \Rightarrow$ fertig, ja
- $x < u \Rightarrow$ suche in linkem Unterbaum
- $x > u \Rightarrow$ suche in rechtem Unterbaum
- Baum leer $\Rightarrow$ fertig, nein

(ii) `Insert 4`

(iii) `Delete 3`

Ist der zu löschende Knoten ein Blatt[5], stellt sich kein Problem dar, anders jedoch falls es sich um einen inneren Knoten handelt. In diesem Fall geht man folgendermaßen vor:
Finde den inorder-nächsten Knoten v (welcher ein Blatt sein muß), ersetze den Inhalt von u durch den Inhalt von v und lösche v.

Satz 59 *Gegeben sei ein natürlicher binärer Suchbaum, (sukzessive) aus n Schlüsseln aufgebaut, jede Permutation ist gleich wahrscheinlich, dann gilt:*

$$\textit{Mittlere Suchpfadlänge} = O(\log n)$$

Beweis: Siehe Quicksort. ▮

Satz 60 *Gegeben sei ein natürlicher binärer Suchbaum mit n Schlüsseln, alle (Bäume) gleich wahrscheinlich, dann gilt:*

$$\textit{Mittlere Suchpfadlänge} = \Theta(\sqrt{n})$$

Ein gravierender Nachteil bei natürlichen binären Bäumen ist der Umstand, daß sie zu Listen entarten können.

[4] Ein symmetrischer Durchlauf ergibt dann eine aufsteigend sortierte Schlüsselfolge.
[5] hier: ein innerer Knoten, dessen beide Unterbäume Blätter sind

12.3.2 Balancierte Bäume

Höhenbalancierte Bäume

(AVL[6]-Bäume)

Balancierungsbedingung:

Für alle Knoten u:

|Höhe des rechten Unterbaums von u - Höhe des linken Unterbaumes von u| ≤ 1

Beispiel:

Frage:

Minimale Anzahl von Blättern (Anzahl interner Knoten +1) in einem AVL-Baum der Höhe h?

Bez.: M_h

allg.: $M_{i+1} = M_i + M_{i-1}$
$\Rightarrow M_i = F_{i+1} \quad (F_0 = F_1 = 1;\ F_{i+2} = F_{i+1} + F_i)$

Satz 61 *(a) Ein AVL-Baum der Höhe h enthält $\geq F_{h+1}$ Blätter und damit $\geq F_{h+1} - 1$ interne Knoten.*

(b) Ein AVL-Baum mit n Blättern hat Höhe $\Theta(\log n)$ $[\leq 1.44 \log_2 n]$

Beweis: siehe oben ∎

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{\sqrt{5}+1}{2}\right)^{h+1} - \left(\frac{\sqrt{5}-1}{2}\right)^{h+1}\right]$$

Operationen

bal := Höhe des rechten Unterbaums - Höhe des linken Unterbaumes

(i) `IsElt` analog zum natürlichen binären Baum.

(ii) `Insert`

[6]Adelson-Velskij/Landis, 1962

1. Der einzufügende Schlüssel ersetzt ein Blatt, das linker Unterbaum eines inneren Knotens p mit bal $= +1$ ist (d.h. der rechte Unterbaum von p ist ein innerer Knoten mit 2 Blättern). Hier ist nichts weiter zu tun, da p mit bal $= 0$ danach immer noch balanziert ist, und sich die Gesamthöhe des Unterbaums mit Wurzel p auch nicht ändert. Somit ergeben sich auch keine Änderungen in der Balanzierung von inneren Knoten, die näher zur Wurzel liegen als p.
2. Symmetrisch: bal $= -1$ und der Schlüssel wird rechts eingefügt.
3. Der einzufügende Schlüssel ersetzt ein Blatt, dessen Geschwister ebenfalls ein Blatt ist (d.h. bal $= 0$). Nach Einfügen ist p zwar noch balanziert (bal $= \pm 1$), aber die Höhe des Unterbaums an p hat sich um 1 erhöht und verletzt jetzt möglicherweise die Balanzierungsbegingung des Vaterknotens von p.

Im Fall 3 ändert sich die Höhe eines Unterbaums. Rebalancierung auf Pfad von p zur Wurzel ist eventuell nötig.

1. Problemfall

2. Problemfall

Satz 62 `IsElt`, `Insert` *und* `Delete` *erfordern bei einem AVL-Baum mit n Schlüsseln Zeit* $O(\log n)$

Beweis: siehe oben ∎

12.3.3 Vorrangwarteschlangen (Priority Queues)

Wir betrachten ein Universium U mit partieller Ordnung $\leq$.

Operationen:

- `IsElt` (eventuell zusätzliches Wörterbuch)
- `Insert`
- `Minimum`
- `DeleteMin`
- `Delete`
- `Decreasekey`
- `Merge`

Priority Queues werden oft auch als *heap* bezeichnet.

Binomialbäume

Satz 63 *B_k läßt sich zerlegen:*

(i)

(ii)

Satz 64

(i) B_n hat 2^n Knoten

(ii) Wurzel von B_n hat Grad n (n Kinder)

(iii) B_n hat Höhe n

(iv) B_n hat $\binom{n}{i}$ Konten der Tiefe i

Beweis:

(iv) Induktion : $n \in \{0, 1\}$

Induktionsschluß: Anzahl der Knoten der Tiefe i in B_n = Anzahl der Knoten der Tiefe i in B_{n-1}+ Anzahl der Knoten der Tiefe $i-1$ in B_{n-1}.
Gleiche Formel wie ür die Binomialkoeffizienten $\left(\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}\right)$
Da gleiche Anfangswerte $\Rightarrow$ Anzahl der Knoten der Tiefe i in $B_n = \binom{n}{i}$

∎

Implementierung:

Verweise auf 1.Kind, nächstes Geschwister (um für jeden Knoten eine konstante Anzahl von Zeigern zu haben).

Binomial Queues

Jedem Knoten wird ein Schlüssel $\in U$ zugeordnet, in der Weise, daß die Heapbedingung gilt (hier Min-Variante, d.h. Minimum steht oben).

Binomial Queue für beliebige Anzahl N von Elementen:

- N in Binärdarstellung: $bin(N)$

- Binomialbaum für jede 1 in $bin(N)$
- verkette Wurzeln in einer doppelt verketteten, zirkulären Liste
- Zeiger auf Elemente in Wurzelliste mit minimalem Schlüssel.

Beispiel:

Grundoperation

Mergen zweier Binomial Queues (mit disjukten Elementen)

for $k = 0, \ldots$ **do**
  if $\exists \leq B_k$ **then** übernimm diesen
  else
    verbinde zwei der B_k zu einem B_{k+1}
    (Hepabedingung beachten)
    übernimm restlichen B_k, falls vorhanden
  fi
od

Zeitbedarf: $O(\log N_1 + \log N_2) = O(\log N) \quad N = N_1 + N_2$

Aufbau einer Binomial Queue mit N Elementen, beginnend mit der leeren Queue, durch forgesetztes Einfügen eines zusätzlichen Elementes (= Merge mit B_0)

Zeitbedarf: $O\left(\sum_{k=0}^{\log N} k \cdot \frac{N}{2^k}\right) = O(N)$

Komplexität der Priority-Queue Operationen für Binomial Queues

(i) `IsElt`: durch zus Wörterbuch $O(\log n)$

(ii) `Insert`: durch Merge $O(\log n)$

(iii) `Minimum`: $O(1)$

(iv) `DeleteMin`: Lösche Wurzel, verschmelze seine Unterbäume mit den anderen Bäumen in der Wuzelliste. Verändere Minimumzeiger dabei entsprechend: $O(\log n)$

(v) `Delete`:

1.Fall: Zu löschendes Element ist Wurzel eines Binomialbaums in der Binomial Queue. (Verfahren wie unter 4.)

2.Fall: Zu löschendes Element ist keine Wurzel.

(vi) `DecreaseKey`: `Delete + Insert`, $O(\log n)$

(vii) `Merge`: $O(\log n)$

12.3.4 Kürzeste Pfade (Dijkstra)

Geg. : Gewichteter Graph $G = (V, A)$, mit der Kantenbewertung $l : A \rightarrow \mathbb{R}^+$ und Startknoten $s \in V$.

Gesucht: Kürzeste Entfernung von s zu allen $v \in V$ (single-source-shortest-path-problem)

procedure shortestpath((V, A) : *digraph*, s : *node*);
    co berechne $\forall v \in V$ die Länge $d(v)$ eines kürzesten
        Pfades von s nach v **oc**
    begin
        forall $v \in V$ **do** $d(v) := \infty$ **od**;
        $d(s) := 0$; $S := \emptyset$;
        co initialisiere $V - S$ als Priorityqueue mit $d(v)$ als Schlüssel **oc**
        while $V - S \neq \emptyset$ **do**
                $v := \min(V - S)$; **co** das Element mit minimalem Schlüssel **oc**
                delete $\min(V - S)$;
                $S := S \cup \{V\}$;
                forall $w \in N(v) - S$ **do co** $N(v) = \{w; (v, w) \in A\}$ **oc**
                        if $d(v) + l(v, w) < d(w)$ **then**
                            decrease $key(w, d(v) + l(v, w))$;
                        fi
                od
        od
    end

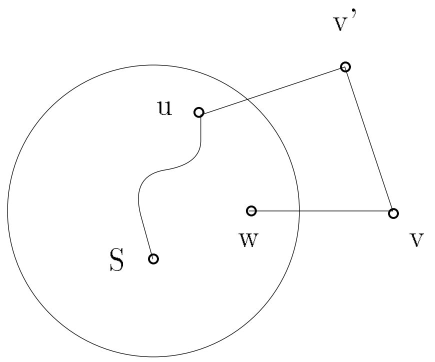
Satz 65 *Dijkstra's Algorithmus (shortestpath) löst das single-source-shortest-path Problem in Zeit*

$$O(t_{init(n)} + n \cdot (t_{\min} + t_{deletemin}) + m \cdot t_{decreasekey})$$

mit $n = |V|$, $m = |A|$.

Mit Binomial Queues als Datenstruktur ergibt sich als Zeitbedarf $O((n + m) \log n) = O(m \log n)$, *(da der Graph zusammmenhängend sein soll, gilt* $m \geq n - 1$*).*

Beweis: Beweis durch Widerspruch. Sei $G = (V, A)$, l Kantenbewertung.



Sei $v \in V - S$ mit minimalem Schlüssel $d(v)$, $d(v) = d(w) + l(v, w)$ (Betrachte nur den Fall $d(v) < \infty$) $d(w)$ korrekt, $w \in S$, $d(v)$ nicht korrekt (sei erster Fehler des Algorithmus)

$\Rightarrow$ $\exists$Pfad $\underset{\in S}{s} \xrightarrow{*} n \rightarrow \underset{\in V-S}{v'} \xrightarrow{*} V$
der kürzer ist als $d(v)$.
$\Rightarrow$ $d(v') < d(v)$
$\Rightarrow$ Widerspruch zur Wahl von v

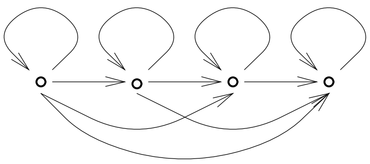
▮

12.3.5 Zusammenhang von Graphen und Transitiver Hülle

$G = (V, A)$ Digraph. $G^* = (V, A^*)$ heißt die reflexive transitive Hülle von G, gegeben durch

$$(u, v) \in A^* \Leftrightarrow \exists \text{ Pfad in } G \text{ von } u \text{ nach } v$$

Beispiel:



G^* für G

Sei ohne Beschränkung der Allgemeinheit $V = \{v_0, \dots, v_{n-1}\}$. Sei weiterhin G gegeben durch Adjazenzmatrix A, mit

$$A[i, j] = \begin{cases} 1 & \text{falls } (v_i, v_j) \in A \\ 0 & \text{sonst} \end{cases}$$

Warshall-Algorithmus

```
for i := 0 to n - 1 do A[i, j] = 1 od;
for j := 0 to n - 1 do
    for i := 0 to n - 1 do
        if A[i, j] = 1 then
            for k := 0 to n - 1 do
                if A[j, k] = 1 then a[i, k] := 1 fi
            od
        fi
    od
od
```

Satz 66 *Der Warshall-Algorithmus berechnet die reflexive, transitive Hülle in der Zeit* $O(n^3)$.

Beweis: Vor dem j-ten Durchlauf der äußersten Schleife gilt:

$\forall i, k\ A[i,k] = 1$ gdw. $\exists$ Pfad von v_i nach v_k, der nur innere Knoten aus $\{v_0, \ldots, v_{j-1}\}$ enthält.

I.A.: $j = 0$

I.S.: $j \to j+1$

$\exists$ Pfad von v_i nach v_k, der nur innere Knoten aus $\{v_0, \ldots, v_j\}$ enthält

$\Leftrightarrow$ ($\exists$ Pfad von v_i nach v_k, der nur innere Knoten aus $\{v_0, \ldots, v_{j-1}\}$ enthält, oder ($\exists$ Pfad von v_i nach v_j, der nur innere Knoten aus $\{v_0, \ldots, v_{j-1}\}$ enthält und $\exists$ Pfad von v_j nach v_k, der nur innere Knoten aus $\{v_0, \ldots, v_{j-1}\}$ enthält))

$\Leftrightarrow$ $A[i,k] = 1$ bereits vor dem vorhergehenden Schleifendurchlauf oder $A[i,j] = A[j,k] = 1$ bereits vor dem vorhergehenden Schleifendurchlauf

$\Leftrightarrow$ $A[i,k] = 1$ vor der $(j+1)$-ten Iteration.

∎

Es gibt Algorithmen zur Berechnung der transitiven Hülle, die etwas schneller sind: $O(\frac{n^3}{\log n})$ oder sogar $O(\frac{n^3}{(\log n)^2})$, aber diese Algorithmen sind aufwendiger zu programmieren und für kleine n langsamer als Warshall.

Beobachtung:

Ist $G = (V, E)$ ungerichtet, so lassen sich die Zusammenhangskomponenten von G durch DFS oder BFS in der Zeit $O(|V| + |E|)$ bestimmen. Damit läßt sich die Adjazenzmatrix A^* der transitiven Hülle von G in Zeit $O(|V|^2) = O(n^2)$ berechnen.

12.3.6 Union-Find-Strukturen

Operationen:

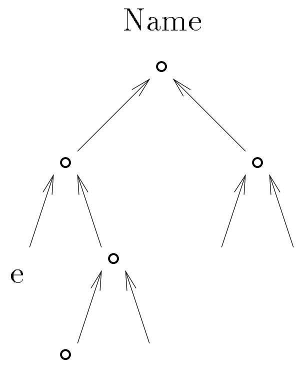
(i) `MakeSet(`e`)`

(ii) `Find(`e`)`

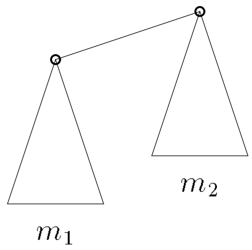
(iii) `Union(`m_1, m_2`)`

Implementierung als Intree, d.h. die Elemente der Menge entsprechen den Knoten, der Name der Menge der Wurzel.

damit: `MakeSet(`e`)`: erzeugt Baum mit einem Element.
`Find(`e`)`: folge Pfad zur Wurzel und lese Namen



`Union(`m_1, m_2`)`:



Gewichtete Union

mache den „kleineren" Baum zum neuen Unterbaum der Wurzel des „größeren" Baumes; ändere gegebenenfalls den Namen des „größeren" Baums.

Satz 67 *Bei gewichteter Union (Gewicht = Größe des Baums) gilt: die Anzahl der Elemente im Baum* $\geq 2^{\text{Höhe des Baums}}$

Beweis: Induktion über die Höhe

I.A.: Höhe des Baums = 0

I.S.: $h = \max\{h_1, h_2 + 1\}$ und $n_2 \leq n_1$

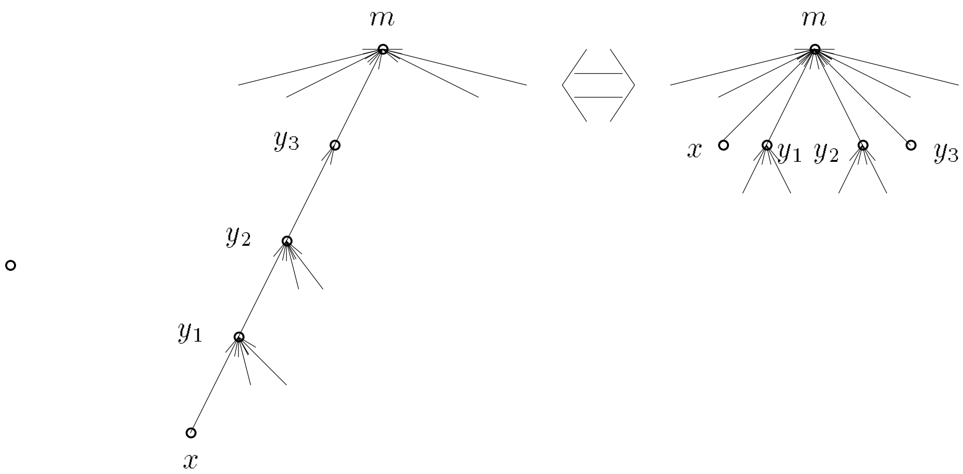
$$\begin{aligned} \Rightarrow n = n_1 + n_2 &\geq \max\{n_1, 2n_2\} \\ &\underset{\text{I.A.}}{\geq} \max\{2^{h_1}, 2^{h_2+1}\} \\ &= 2^h \end{aligned}$$

∎

Bemerkungen:

1. Gleiches gilt bei Gewichtung nach Höhe

2. *Pfadkompression* führt zu sehr kleinen mittleren Kosten für `Find`, z.B. Kosten pro `Find` im Mittel $O(\log^* n)$



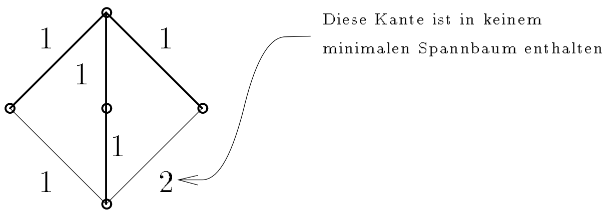
12.3.7 Minimale Spannbäume

$G = (V, E)$ ungerichtet, Kantenbewertung $w: E \to \mathbb{R}^+$
G zusammenhängend
$T = (V, E' \subseteq E)$ Spannbaum von G
ewicht des Spannbaums: $w(T) = \sum_{e \in E'} w(e)$

Definition 12.3.1 *T ist ein* minimaler *Spannbaum von G, falls gilt:*

$$w(T) \leq w(T') \quad \forall T, T' \textit{ Spannbaum von } G$$

Beispiel:

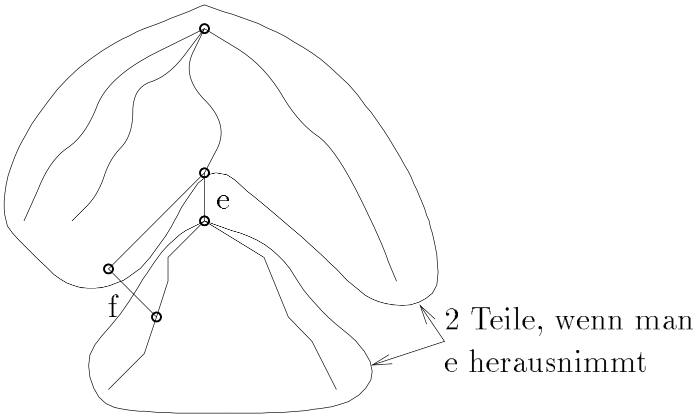


Wir nehmen im weiteren an: alle $w(e)$ verschieden

Satz 68 *Sei C ein Kreis in G, $e \in C$ mit maximalem Gewicht. Dann kommt e in keinem MST (minimaler Spannbaum) von G vor.*

Beweis: Annahm, es wäre anders:

MST:



$$w(f) < w(e) \Rightarrow T - \{e\} + \{f\} \text{ hat kleineres Gewicht}$$

▮

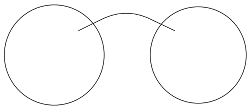
Kruskal's Algorithmus (1956)

procedure mst($G = (V, E) :$ *graph*)
    $E' := \emptyset$
    $K := \{\{v\};\ v \in V\}$
    Q :=Priority Queue aller $e \in E$, mit $w(e)$ als Schlüssel;
    while (K enthält mehr als eine Menge) **do**
        $\{v, w\} := \min(Q)$; delete min(Q);
        $v_0 :=$ Find(v); w_0 :=Find(w);
        if $v_0 \neq w_0$ **then**
          Union(v_0, w_0)
          $E' := E' \cup \{\{v, w\}\}$;
        fi
    od
return $T = (V, E')$

Satz 69 *Kruskal's Algorithmus berechnet einen*[7] *minimalen Spannbaum in Zeit* $O(m \log n)$ $(m = \Omega(n))$.

Beweis:

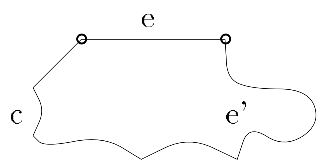
(i) Algorithmus liefert Wald



(ii) Wald ist Baum

[7]Falls alle Kantengewichte verschiednen sind - *den*

(iii) Spannbaum hat minimales Gewicht: $\forall e' \in C - \{e\} e: \; w(e') < w(e)$.



∎

13. $\mathcal{NP}$-Vollstänigkeit

13.1 Grundlagen

Definition 13.1.1 *(i) Eine aussagenlogische Formel ist aus Variablen, booleschen Operatoren* $(\vee, \wedge, \neg)$*, Konstanten (0,1) und Klammern aufgebaut.*

(ii) Eine boolesche Formel heißt erfüllbar*, wenn es Belegungen der Variablen mit Werten aus* $\{0, 1\}$ *gibt, so daß die Formel* $\equiv 1$ *wird. Z.B.* $(x \wedge y) \vee (\bar{x} \wedge \bar{y})$

(iii) SAT sei die Sprache, die die erfüllbaren aussagenlogischen Ausdrücke enthält, (in geeigneter Kodierung).

(iv) Eine aussagenlogische Formel F *ist in* konjuktiver Normalform*, falls* F *die Form hat* $F = C_1 \dots C_m$ *wobei* $C_i = z_{i1} v z_{i2} \dots v z_{ir_i}$ *für* $i \in [1 : m]$ *(die* c_i *nennt man auch Klauseln) und* $z_{ij} \in \{x_1, \bar{x_1}, x_2, \bar{x_2}, \dots x_m, \bar{x_m}\}$ *für* $i \in [1 : m]$, $j \in [1 : r_i]$ *(die* $z_i j$ *nennt man Literale).*

Definition 13.1.2 *(a) CNF-SAT ist die Sprache, die die erfüllbaren aussagenlogischen Formeln in konjunktiver Normalform enthält.*

(b) 3-CNF-SAT bzw. 3-SAT ist die Sprache, die die erfüllbaren aussagenlogischen Formeln in konjunktiver Normalform enthält, bei denen jede Klausel aus maximal drei Literalen besteht.

Bemerkung:

2-SAT $\in \mathcal{P}$. (Idee $(a \vee b) \Leftrightarrow (\bar{a} \Rightarrow b \wedge \bar{b} \Rightarrow a)$)

Satz 70 *(i) SAT ist* $\mathcal{NP}$*-vollständig*

(ii) CNF-SAT ist $\mathcal{NP}$*-vollständig*

(iii) 3-SAT ist $\mathcal{NP}$*-vollständig*

Beweis: (Skizze)

3-SAT ist die stärkste Behauptung.

(i) 3-SAT $\in \mathcal{NP}$:

Teste nichtdeterministisch alle Belegungen der Variablen. Ist eine erfüllt so akzeptiert auch die $\mathcal{NP}$-Maschine.

(ii) 3-SAT $\mathcal{NP}$-hat:

Sei N irgendeine $\mathcal{NP}$-Maschine.

zu zeigen: $L(N) \overset{P}{\underset{m}{\leq}}$ 3-SAT

Da N eine $\mathcal{NP}$-Maschine ist gibt es ein Polynom p, so daß N auf Eingabe maximal $p(|x|)$ Schritte macht.

Betrachte Berechnung von N auf x. (Ohne Beschränkung der Allgemeinheit akzeptiert mit leerem Band und Kopf ganz links – also eindeutiger Endzustand).

Boolesche Variable

$x_{t,i,a} \;\hat{=}\;$ zum Zeitpunkt t, enthält das i-te Feld das Zeichen a

also $t, i \in [1, p(n)] \quad a \in \cup Q \times$.

Formel:

	„erste Zeile richtig“	Startzustand p_0; Eingabe $x_0 \dots x_{n-1}$ sonst blank
$\wedge$	„$p(n)$-te Zeile richtig“	links q_a; Band leer
$\wedge$	„in jeder Zeile steht nur ein Zustand“	das Paar (q, x) taucht in jeder Zeile nur einmal auf
$\wedge$	„stimmt die Übergangsfunktion“	alle „T -Felder“ müssn konsisten gemäß der δ-Relation sein.

Man kann zeigen, daß die Formel polynomial groß ist (etwa $p^3(n)$) und ist in polynomialer Zeit berechenbar (etwa $p^3(n)$).

∎

Definition 13.1.3 *CLIQUE ist Sprache, $\{(G, k)|$ der (ungerichtete) Graph G enthält ine Clique mit $\geq k$ Knoten$\}$*

Satz 71 *CLIQUE ist $\mathcal{NP}$-vollständig.*

Beweis:

(i) $\in \mathcal{NP}$: klar

(ii) $\mathcal{NP}$ hat:

zu zeigen: 3SAT $\leq_p^m$ CLIQUE (da $\leq_m^p$ transitiv)

Sei $C = \{C_1, \dots C_m\}$ die Klauselmenge

$$C_i = z_{i_1} \cup z_{i_2} \cup z_{i_3} \quad i \in [1:m]$$
$$z_{ij} \in \{x_1, \bar{x_1}, x2, \bar{x_2}, \dots x_n, \bar{x_n}\}$$

Konstruiere Graphen $G = (V, E)$ mit

$$V = \{(i,j) | i \in [1:m], j \in \{1,2,3\} \Rightarrow |V| = 3m$$
$$E = \{\{(i,j),(i',j')\} | \quad \underbrace{i \neq i'}_{\substack{\text{in verschiedenen} \\ \text{Klauseln}}} \quad \wedge \quad \underbrace{z_{ij} \neq \bar{z'_i}j'}_{\substack{\text{Negationen sind} \\ \text{nicht voneinander} \\ \text{abhänig, d.h.} \\ \text{gemeinsam erfüllbar}}} \quad \}$$

Behauptung:

G enthält Clique der Größe $\geq m \Leftrightarrow$ Die Menge der Klauseln ist erfüllbar.

$\Leftarrow$ Erfüllbar $\Rightarrow$ für jede der Klausel C_i existiert $k_i \in \{1,2,3\}$ so daß die Menge $\{(i, k_i)\}$ ein vollständiger Teilgraph ist.

$\Rightarrow$ Haben wir einen vollständigen Teilgraphen der Größe m, so ist aus jeder Klausel ein (i, k_i) in diesem enthalten. Diese z_{i,k_i} können dann auf wahr gesetzt werden.

$\blacksquare$

13.2 Ein kleine Sammlung $\mathcal{NP}$-vollständiger Probleme

Definition 13.2.1 *(i) Partition ist die Sprache*

$$\{(b_1, \dots, b_n) | b_i \in \mathbb{N} \exists I \supseteq \{1 \dots n\} \textstyle\sum_{i \in I} b_i = \sum_{i \notin I} b_i\}$$

(ii) Binpacking ist die Sprache

$$\{(a_1 \dots a_n) b, h | \quad a_i,\ b,\ h \in \mathbb{N} \textit{ die Menge der } a_i \textit{ kann in } \leq h \textit{ Teile paritioniert werden, wobei die Summe Teil maximale } b \textit{ ist } \}$$

(iii) Hamiltonion circuit ist die Sprache

$$\{G = (V, E) | \quad G \textit{ ungerichtet, der einen einfachen Kreis der Länge } |V| \textit{ enthält.}\}$$

(iv) TSP(=Traveling salesman problem) ist die Sprache

$$\{(G, w, B) | \quad G = (V, E) \textit{ gerichtet, } w \textit{ Gewichtung der Kanten von } G \; w: E \to \mathbb{N}. \; G \textit{ enthält einen Hamilton'schen Kreis der Länge (bzg. Gewicht) } \leq B\}$$

Satz 72 *Partition, Hamiltonion Circuit und TSP sind $\mathcal{NP}$-vollständig.*

Literaturverzeichnis

[1] Uwe Schöning. *Theoretische Informatik kurzgefasst*. BI Wissenschaftsverlag, 1992.

[2] Ingo Wegener. *Theoretische Informatik*. B.G. Teubner, 1993.

[3] A. Aho, J.E. Hopcroft, J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1976.

[4] J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[5] W. Brauer. *Automatentheorie*. B. G. Teubner, 1984.