

Informatik – 3. Semester
Studienrichtung Informationstechnik

Dozent: Herr Prof. Dr. Müller

Automatentheorie und Einführung in den Compilerbau

Erstellt von Christian Ehrlicher und Mandy Bernott

letzte Änderung: 13.12.2001

Inhaltsverzeichnis

- I. Einleitung
 - I.1. Motivation und Plan
 - I.2. Grundlegende Definitionen
- II. Endliche Automaten und Reguläre Sprachen
 - II.1. Motivation
 - II.2. Deterministische Endliche Automaten (DEAs)
 - II.3. Nichtdeterministische Endliche Automaten (NEAs)
 - II.4. Das Pumping - Lemma
- III. Formale Sprachen
 - III.1. Rechtslineare Sprachen
 - III.2. Kontextfreie Sprachen
 - III.3. Kellerautomaten
 - III.4. Chomsky-Hierarchie
- IV. Anwendung im Compilerbau
 - IV.1. Grundlagen Compiler
 - IV.2. Backus-Naur-Form (BNF)
 - IV.3. Scanner- Lexikalische Analyse
 - IV.4. Parser – Syntaxanalyse
 - IV.5. Effiziente Parser

I. Einführung

I.1. Motivation und Plan

Hauptaufgaben der Informatik

- 1) Realitätsnahe, computergerechte Modellierung
- 2) Konstruktion von Maschinen, die die Umsetzung der Modelle nachprüfbar unterstützen

Konkrete Fragen der Vorlesung

Wie muss eine Sprache aussehen, die der Computer „verstehen“ kann
Wie muss eine Maschine aussehen, die eine Sprache „verstehen“ kann
Welche Schritte sind nötig, um aus einer „höheren“ Programmiersprache Maschinencode zu konstruieren

I.2. Grundlegende Definitionen

I.2.1. Definition Worte

- a) Ein **Alphabet** ist eine endliche Menge von beliebigen Zeichen.
Bsp.: $\{0,1,2,3,4,5,6,7,8,9\}$, $\{0,1\}$, ASCII-Code
Notation: typischerweise ist Sigma (Σ) ein Alphabet
Typischerweise sind a, b, c, d, ... Elemente aus Σ , die sogenannten Buchstaben oder Symbole
- b) Ein **Wort** über Σ ist eine endliche Folge von Buchstaben aus Σ .
Bsp.: Sei $\Sigma = \{a, b\}$, dann ist abba ein Wort über Σ
Notation: typischerweise sind x, y, z Worte über einen Alphabet
- c) Die **Länge** eines Wortes x ist die Anzahl der Buchstaben in x.
Bsp.: abba hat die Länge 4
Notation: $|x|$ bezeichnet die Länge von x
- d) Es gibt genau ein Wort der Länge 0 über Σ , das **leere Wort**.
Notation: Epsilon (ϵ) ist das leere Wort $\rightarrow |\epsilon| = 0$
- e) Worte der Länge u, die aus dem gleichen Buchstaben bestehen, werden mit a^n bezeichnet.
Sie sind induktiv definiert durch:
$$a^0 = \epsilon$$
$$a^{n+1} = a^n a$$

Bsp.: $a^5 = aaaaa$
- f) Die Menge aller Worte über einen Alphabet Σ wird mit Σ^* bezeichnet
Bsp.: $\{a,b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \dots\}$
 $\{a\}^* = \{a^n \mid n \geq 0\}$
Ist $\Sigma = \phi$, so ist $\phi^* = \{\epsilon\}$
- g) Die Operation **Konkatenation** fügt zwei Worte zusammen, indem sie zwei Worte hintereinander schreibt
Bsp.:
aba \rightarrow ababba
bba
Notation: xy ist die Konkatenation der Worte x und y

Satz I.2.1. Eigenschaften der Konkatenation

Seien x, y, z Worte über einen beliebigen Alphabet Σ

Es gilt: a) die Konkatenation ist assoziativ $\rightarrow (xy)z = x(yz)$

b) das Leere Wort ist die Identität der Konkatenation $\rightarrow \varepsilon x = x\varepsilon = x$

c) die Konkatenation ist Längenadditiv $\rightarrow |xy| = |x| + |y|$

insbesondere gilt: $a^m a^n = a^{m+n} \forall n, m \geq 0$

I.2.2. Definitionen

a) Wir schreiben x^n für das Wort, das entsteht, wenn n Kopien konkateniert werden.

Formal ist x^n definiert durch: $x^0 = \varepsilon, x^{n+1} = x^n x$

Bsp.: $(aab)^5 = aabaabaabaabaab$

$(aab)^0 = \varepsilon$

b) Sei $a \in \Sigma$ und $x \in \Sigma^*$

Wir schreiben $\#a(x)$ für die Anzahl der a 's in x

Bsp.: $\#0(00110101000) = 7$

c) Ein **Präfix** von x ist ein initiales Teilwort von x .

Formal: y ist Präfix von x , falls es ein Wort z gibt, so dass $x=yz$

ε ist Präfix von jedem Wort

x ist Präfix von x

Bsp.: $abaab$ ist Präfix von $abaababab$

Ein Präfix ist ein **echtes Präfix** von x , falls y Präfix von x ist und $y \neq \varepsilon$ und $y \neq x$

I.2.3 Definition Operation auf Mengen

a) Mengen – Vereinigung

seien A, B Teilmengen von Worten über Σ^*

$\rightarrow A \vee B = \{ x \mid x \in A \text{ oder } x \in B \}$

b) Mengen – Durchschnitt

$\rightarrow A \wedge B = \{ x \mid x \in A \text{ und } x \in B \}$

c) Komplement in Σ^*

$\bar{A} = \{ x \in \Sigma^* \mid x \notin A \}$

d) Mengen – Konkatenation

$AB = \{ xy \mid x \in A \text{ und } y \in B \}$

Bsp.: $\{a, ab\} \{b, ba\} = \{ab, aba, abb, abba\}$

e) Mengen – Potenz

$A^0 = \{\varepsilon\}, A^{m+1} = AA^m$

Bsp.: $\{ab, aab\}^0 = \varepsilon$

$\{ab, aab\}^1 = \{ab, aab\}$

$\{ab, aab\}^2 = \{abab, abaab, aabab, aabaab\}$

f) Stern – Operator

$$A^* = \bigvee_{n \geq 0} A^n = A^0 \vee A^1 \vee A^2 \vee \dots \vee A^n$$

Alternative Definitionen

$$A^* = \{x_1, x_2, \dots, x_n \mid \forall n \geq 0 \text{ und } x_i \in A, 1 \leq i \leq n\}$$

Beachte: da n auch 0 sein kann, gilt insbesondere, dass $\varepsilon \in A^*$ für jedes A

g) Plus – Operator

$$A^+ = AA^* = \bigvee_{n \geq 0} A^n$$

A^+ bezeichnet die pos. Potenzen von A

Satz 1.2.2 Gesetze auf Mengenoperationen

a) $(A \vee B) \vee C = A \vee (B \vee C)$
 $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
 $(AB)C = A(BC)$

b) $A \vee B = B \vee A$
 $A \wedge B = B \wedge A$
 $AB \neq BA$

c) $A \vee \phi = \phi \vee A = A$
 $\{\varepsilon\}A = A\{\varepsilon\} = A$
 $\phi A = A\phi = A$

d) $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
 $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
 $A(B \vee C) = (AB) \vee (AC)$

e) $\overline{(A \vee B)} = \overline{A} \wedge \overline{B}$
 $\overline{(A \wedge B)} = \overline{A} \vee \overline{B}$

f) $A^*A^* = A^*$
 $(A^*)^* = A^*$
 $A^* = \{\varepsilon\} \vee AA^* = \{\varepsilon\} \vee A^*A^*$
 $\phi^* = \{\varepsilon\}$

II. Endliche Automaten und Reguläre Sprachen

II.1. Motivation

Der Zustand eines Systems beschreibt einen Schnappschuss des Systems zu einem definierten Zeitpunkt

Er enthält Informationen, die notwendig sind, um zu entscheiden, wie sich das System weiterentwickelt.

Transitionen definieren den Zustandwechsel

Bsp.: Schaltkreise
Digitaluhren
Aufzüge
Rubik's Cube ($54!/9!^6$ Zustände, 12 Transitionen)
Eight Puzzle

Ein System mit endlich vielen Zuständen und Transitionen wird in der mathematischen Abstraktion als endlicher Automat bezeichnet.

II.2. Deterministische Endliche Automaten (DEAs)

Ein DEA ist ein 5-Tupel $(Q, \Sigma, \delta, s, F)$ mit:

Q als eine endliche Menge von Zuständen

Σ als ein endliches Alphabet

$\delta: Q \times \Sigma \rightarrow Q$ ist die Transitionsfunktion mit $\forall q \in Q \forall a \in \Sigma \exists q' \in Q$ mit $\delta(q,a) = q'$

$s \in Q$ – der (eindeutige) Startzustand

$F \subseteq Q$ als die Menge der akzeptierenden Zustände (Endzustände)

II.2.1 Beispiel

Sei $M = \{Q, \Sigma, \delta, s, F\}$ ein DEA mit:

$Q = \{0,1,2,3\}$, $\Sigma = \{a,b\}$, $s = 0$, $F = \{3\}$ und

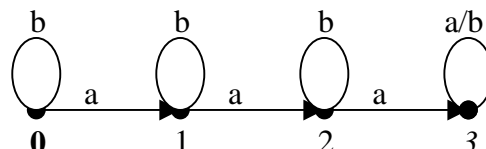
$\delta(0,a) = 1$, $\delta(1,a) = 2$, $\delta(2,a) = 3$, $\delta(3,a) = 3$, $\delta(q,b) = q$, $\forall q \in \{0,1,2,3\}$

Alternative Darstellungen für δ :

Übergangstabelle:

δ	a	b
0	1	0
1	2	1
2	3	2
3	3	3

Übergangendiagramm:



Arbeitsweise eines DEAs:

$x = abaaba \in \Sigma^*$
Zustand: 011233 $\in F$?? \rightarrow ja, also wird Wort akzeptiert

$x = abba \in \Sigma^*$
Zustand: 01112 $\in F$?? \rightarrow nein, also wird Wort nicht akzeptiert

\rightarrow Wort braucht mindestens 3 a's um vom DEA akzeptiert zu werden

II.2.2 Definitionen

a) Sei $\delta_1: Q \times \Sigma^* \rightarrow Q$

mit $\delta_1(q, \varepsilon) = q$

$\delta_1(q, xa) = \delta(\delta_1(q, x), a)$

Bsp.: $w1 = baa$

$$\begin{aligned} \delta_1(0, baa) &= \delta(\delta_1(0, ba), a) \\ &= \delta(\delta(\delta_1(0, b), a), a) \\ &= \delta(\delta(\delta(\delta_1(0, \varepsilon), b), a), a) \\ &= \delta(\delta(\delta(0, b), a), a) \\ &= \delta(\delta(0, a), a) \\ &= \delta(\delta(1, a) \\ &= 2 \end{aligned}$$

b) Ein Wort $x \in \Sigma^*$ wird von einem DEA M **akzeptiert**, falls $\delta_1(s, x) \in F$

c) Ein Wort $x \in \Sigma^*$ wird von einem DEA M **abgelehnt**, falls $\delta_1(s, x) \notin F$

d) Die Menge $L(M) = \{ x \in \Sigma^* \mid \delta_1(s, x) \in F \}$ enthält alle Worte, die von einem DEA M akzeptiert werden

$L(M)$ heißt die **Sprache**, die von M akzeptiert wird.

e) Eine Teilmenge $A \subseteq \Sigma^*$ heißt regulär, falls $A = L(M)$ für einen DEA M.

Bsp.: Die Sprache, die von M aus Beispiel II.2.1 akzeptiert wird, ist $\{ x \in \{a, b\}^* \mid x \text{ enthält mind. 3 a's} \}$.

Diese Menge ist regulär.

II.2.2 Beispiel:

Sei $A = \{ xaaaay \mid x, y \in \{a, b\}^* \}$

Frage: Ist A regulär?

Definiere $M = \{ Q, \Sigma, \delta, s, F \}$ mit

$Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}$, $S=0$, $F=\{3\}$ und

δ	a	b
0	1	0
1	2	0
2	3	0
3	3	3

Formal müsste bewiesen werden, dass $A=L(M)$ bzw. dass $A \subseteq L(M)$ und $A \supseteq L(M)$

Satz II.2.2 Regularität

- a) Ist Σ ein endliches Alphabet, so ist Σ^* regulär
- b) Ist A eine endliche Menge von Worten über einem endlichen Alphabet Σ , so ist A regulär
- c) Sind A,B reguläre Sprachen über einem endlichen Alphabet Σ , so ist $A \wedge B$ ebenfalls regulär

Beweisidee für c)

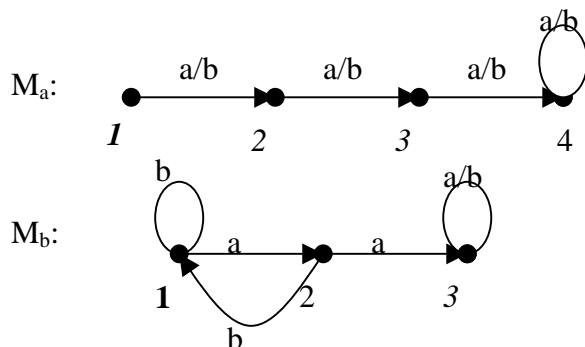
A, B sind regulär

$\exists M_1 = \{Q_1, \Sigma, \delta_1, s_1, F_1\}$ und $\exists M_2 = \{Q_2, \Sigma, \delta_2, s_2, F_2\}$ so dass $A = L(M_1)$ und $B = L(M_2)$

Konstruiere M_3 , so dass $L(M_3) = A \wedge B$

$M_3 = \{Q_3, \Sigma, \delta_3, s_3, F_3\}$ mit
 $Q_3 = Q_1 \times Q_2 = \{(p,q) \mid p \in Q_1 \text{ und } q \in Q_2\}$
 $F_3 = F_1 \times F_2 = \{(p,q) \mid p \in F_1 \text{ und } q \in F_2\}$
 $s_3 = (s_1, s_2)$
 $\delta_3 = Q_3 \times \Sigma \rightarrow Q_3$ mit $\delta_3((p,q), a) = (\delta_1(p,a), \delta_2(q,a))$

Bsp.: Sei $A = \{x \in \{a,b\}^* \mid |x| \leq 2\}$ und
 $B = \{x \in \{a,b\}^* \mid x \text{ enthält zwei aufeinanderfolgende a's}\}$

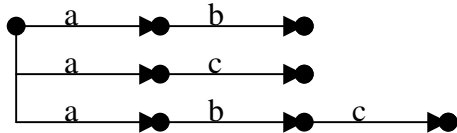


δ_3	a	b
(1,1)	(2,2)	(2,1)
(1,2)	(2,3)	(2,1)
(1,3)	(2,3)	(2,3)
(2,1)
(2,2)		
(2,3)		
(3,1)		
(3,2)		
(3,3)		
(4,1)		
(4,2)		
(4,3)		

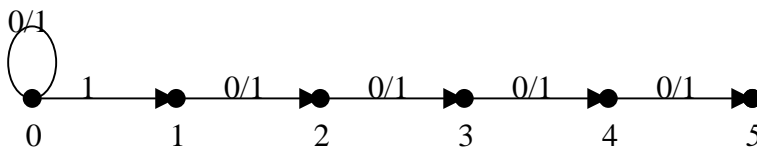
II.3 Nichtdeterministische Endliche Automaten (NEAs)

Häufig lässt sich ein endlicher Automat für eine Sprache einfacher konstruieren, wenn man Grundregeln der Definitionen verletzt.

Bsp: $A = \{ab, ac, abc\}$



$B = \{x \in (0,1)^* \mid \text{der fünfte Buchstabe von rechts in } x \text{ ist eine } 1\}$



$11010010 \in B$

$11000010 \notin B$

Beide Automaten verletzen den Determinismus. Im ersten Schritt wird geraten, welcher Weg eingeschlagen werden soll.

Solche Automaten heißen nichtdeterministische endliche Automaten. Sie akzeptieren ein Wort, wenn es einen Weg von einem Startzustand zu einem Endzustand gibt (!) auf dem das Wort abgearbeitet werden kann.

II.3.1 Definitionen NEA

- a) ein NEA ist ein 5-Tupel $N = \{Q, \Sigma, \Delta, S, F\}$
 Q : eine Menge von Zuständen
 Σ : ein endliches Alphabet
 $S \subseteq Q$ ist die Menge der Startzustände
 $F \subseteq Q$ ist die Menge der Endzustände
 $\Delta: Q \times \Sigma \rightarrow 2^Q$, wobei $2^Q = \{P \mid P \subseteq Q\}$

Intuitiv liefert $\Delta(p, a)$ die Menge von Zuständen, die von p aus nach Abarbeitung von a erreichbar sind.

- b) $\Delta_1: 2^Q \times \Sigma^* \rightarrow 2^Q$

$$\Delta_1(p, \epsilon) = P$$

$$\Delta_1(P, xa) = \bigcup_{q \in \Delta_1(P, x)} \Delta(q, a)$$

- c) Ein NEA N akzeptiert $x \in \Sigma^*$, falls

$$\Delta_1(S, x) \cap F \neq \emptyset$$

d.h. $\exists q \in F$, so dass $q \in \Delta_1(S, x)$

d.h. x wird akzeptiert, wenn es von (irgend) einem Startzustand in einen Endzustand geführt werden kann

Satz II.3.1

Zu jedem DEA M gibt es einen NEA N mit $L(M) = L(N)$

Beweis: Sei $M = \{Q, \Sigma, \delta, s, F\}$ so ist $N = \{Q, \Sigma, \Delta, \{s\}, F\}$ mit $\Delta(p, a) = \{\delta(p, a)\}$ der gesuchte NEA

Satz II.3.2

Zu jedem NEA N gibt es (mindestens) einen DEA M mit $L(N) = L(M)$

Beweis: Sei $N = \{Q_n, \Sigma, \Delta_n, S_n, F_n\}$ so konstruiere $M = \{Q_m, \Sigma, \delta_m, s_m, F_m\}$ mit:

$$Q_m = 2^{Q_n}$$

$$F_m = \{P \subseteq Q_n \mid P \wedge F_n \neq \Phi\}$$

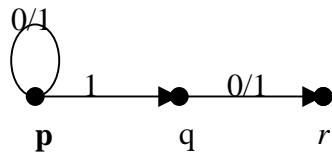
$$\delta_m(P, a) = \Delta_n(P, a)$$

$$s_m = S_n$$

[Beweis durch Beispiel]

II.3.1 Beispiel

Sei N: N akzeptiert die Sprache $A = \{x \in \{0,1\} \mid \text{Das 2. Symbol von rechts in } x \text{ ist eine „1“}\}$



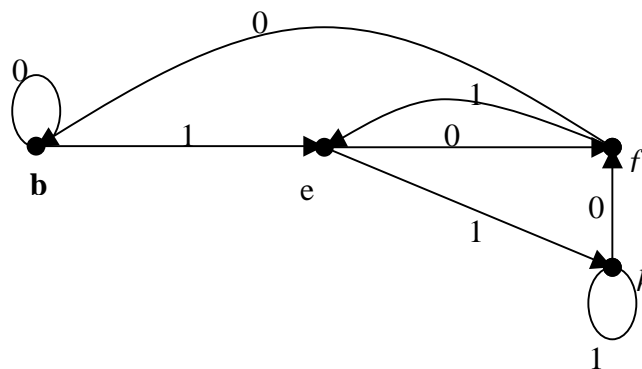
Konstruiere $2^Q: \{\Phi, \{p\}, \{q\}, \{r\}, \{pq\}, \{pr\}, \{qr\}, \{pqr}\}$ und daraus den DEA

	2^Q	0	1
a	Φ	Φ	Φ
b	$\{p\}$	$\{p\}$	$\{pq\}$
e	$\{q\}$	$\{r\}$	$\{r\}$
d	$\{r\}$	Φ	Φ
e	$\{pq\}$	$\{pr\}$	$\{pqr\}$
f	$\{pr\}$	$\{p\}$	$\{pq\}$
g	$\{qr\}$	$\{r\}$	$\{r\}$
h	$\{pqr\}$	$\{pr\}$	$\{pqr\}$

wird nie erreicht
 wird nie erreicht
 wird nie erreicht
 wird nie erreicht

→ Optimierte Zustandstabelle:

δ	0	1
b	b	e
e	f	h
f	b	e
h	f	h



→ vorletztes Zeichen muss ,1' sein: $h \rightarrow f$: h kann nur über ,1' erreicht werden
 $e \rightarrow f$ und $e \rightarrow h$: e kann nur über ,1' erreicht werden

II.4 Das Pumping – LEMMA

Auftrag: Zeigen Sie, dass $A = \{a^n b^n \mid n \geq 0\}$ **nicht** regulär ist

Idee: Als Widerspruchsbeweis \rightarrow Annahme: A ist regulär $\rightarrow \exists M$ mit $L(M) = A$
 Sei $k = |Q|$ die Anzahl der Zustände von M und $n \gg k$

Betrachte $a^n b^n$:
 $aaaaaaaaaaaaaaaaabbbbbbbbbbbbbbb$
 $\uparrow \quad \uparrow \quad \uparrow \quad \quad \quad \uparrow$
 $s \quad u \quad v \quad \quad \quad w \quad r$

Da $n \gg k \exists q \in Q$, so dass p mindestens 2 mal ‚besucht‘ wird

für $|v| = 0$
 $\delta_1(s,u) = p \quad \delta_1(s,u) = p$
 $\delta_1(p,v) = p \quad \delta_1(p,w) = r \in F$
 $\delta_1(p,w) = r \in F$
 uvw wird akzeptiert ABER: $uw = a^{n-j}b^n$, falls $|v| = j$ und $uw \notin A$

\rightarrow Also ist A nicht regulär

Satz II.4.1 Pumping-Lemma

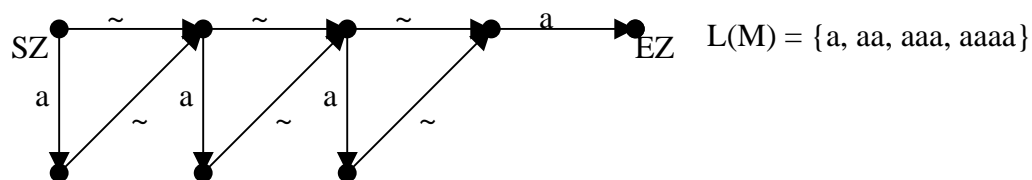
Sei A eine reguläre Menge
 Es gilt: $\exists k \geq 0 \forall x,y,z$ mit $xyz \in A$ und $|y| \geq k$
 gibt es ein uvw mit $y = uvw$, $v \neq \epsilon$
 $\forall i \geq 0 xuv^i wz \in A$

Anwendung des Satzes als Contraposition:

Satz II.4.2

Sei A eine beliebige Menge von Worten
 Wenn $\forall k \geq 0 \exists x,y,z$ mit $xyz \in A$, $|y| \geq k$
 und $\forall uvw$ mit $y = uvw$, $v \neq \epsilon$
 $\exists i \geq 0$, so dass $xuv^i wz \notin A$
 dann ist A nicht regulär

Informell: Es gibt EA's mit \sim -Übergängen (Blank – Übergänge, auch ϵ -Übergänge), bei denen in andere Zustände ‚gesprungen‘ werden kann. Sie sind äquivalent zu DEA's und NEA's
 Beispiel:



III. Formale Sprachen

III.1 Rechtslineare Sprachen

Die EA's arbeiten Worte ab. Sie akzeptieren Worte. Sie erkennen Sprachen.

Frage: Wie können Sprachen erzeugt werden?

III.1.1. Definition

a) Eine rechtslineare Grammatik (RLG) ist ein 4-Tupel $G=(N,\Sigma,P,S)$

- mit
- N: einer endlichen Menge von Zeichen
 - Σ : einer endlichen Menge von Zeichen; $\Sigma \neq N$ (oder: $N \cap \Sigma = \emptyset$)
 - P: eine Teilmenge von $(N \times \Sigma^*) \cup (N \times \Sigma^* N)$
 - $S \in N$ ist das Startsymbol

Notationen:

- Die Elemente aus N heißen „Nichtterminale“ und werden typischerweise mit A,B,C,S,... bezeichnet
- Σ ist die Menge der „Terminalen“ Symbole
typisch: a,b,c,0,1,
- P ist die Menge von Produktionsregeln. Zum besseren Verständnis schreiben wir

$A \rightarrow x$	statt	(A,x)
$B \rightarrow xC$		(B,xC)

Oft schreiben wir $A \rightarrow aa \mid bB \mid a$ für die 3 Regeln $A \rightarrow aa$; $A \rightarrow bB$; $A \rightarrow a$

b) Seien $\alpha, \beta \in (N \cup \Sigma)^*$

- β heißt ableitbar aus α in einen Schritt mit der Grammatik G falls es $\exists \alpha_1, \alpha_2 \in (N \cup \Sigma)^*$ und $\exists A \rightarrow \gamma$ aus P mit $\alpha = \alpha_1 A \alpha_2$ und $\beta = \beta_1 \gamma \beta_2$
- β entsteht aus α durch Ersetzen eines Nichtterminalen gemäß einer Produktionsregel
- Ist β aus α in einem Schritt mit der Regel aus G ableitbar, so schreiben wir $\alpha \xrightarrow[G]{1} \beta$

c) $\xrightarrow[G]{*}$ ist der reflexive und transitive Abschluss von $\xrightarrow[G]{1}$ definiert durch:

- $\alpha \xrightarrow[G]{0} \alpha$ für alle $\alpha \in (N \cup \Sigma)^*$
- $\alpha \xrightarrow[G]{n+1} \beta$, falls $\exists \gamma$ mit $\alpha \xrightarrow[G]{n} \gamma$ und $\gamma \xrightarrow[G]{1} \beta$
- $\alpha \xrightarrow[G]{*} \beta$, falls es $\exists n \geq 0$ gibt mit $\alpha \xrightarrow[G]{n} \beta$

d) Ein Wort $\alpha \in (N \cup \Sigma)^*$ heißt „gültige Form“, falls $S \xrightarrow[G]{*} \alpha$. α heißt „Satzform“, falls α eine „gültige Form“ ist und $\alpha \in \Sigma^*$ gilt.

e) Die Sprache $L(G) = \{ x \in \Sigma^* \mid S \xrightarrow[G]{*} x \}$ heißt „die durch G erzeugte Sprache“

$L(G)$ heißt rechtslinear, wenn G eine rechtslineare Grammatik ist.

Beispiel:

Sei $G = \{N, \Sigma, P, S\}$ mit $N = \{S, A, B\}$, $\Sigma = \{a, b\}$ und $P = \{S \rightarrow aA, A \rightarrow \varepsilon|aA|bB, B \rightarrow \varepsilon|bB\}$

$$\begin{array}{l}
 S \xrightarrow[G]{1} aA \quad \xrightarrow[G]{1} a \\
 \xrightarrow[G]{1} aaA \quad \xrightarrow[G]{1} aa | aaaA | aabB \\
 \xrightarrow[G]{1} abB \quad \xrightarrow[G]{1} ab | abbB
 \end{array}$$

Es gilt also z.B. $S \xrightarrow[G]{*} abbB$; $aaAbb$ ist dagegen keine gültige Satzform

$$L(G) = \{ a^n b^m \mid n > 0, m \geq 0 \}$$

Bemerkungen:

- Eine Charakteristik der rechtslinearen Sprache ist, dass Nichtterminale in den gültigen Formen immer ganz rechts stehen. Die Worte werden also von links nach rechts erzeugt.
- Wird behauptet, dass $L(G) = L$, so müsste formal gezeigt werden, dass $L(G) \subseteq L$ und $L \subseteq L(G)$ ist, also dass jedes x , das durch G erzeugt wird, in L liegt und jedes $x \in L$ durch G erzeugt werden kann.

Bsp.: Konstruiere für $L = \{x \in \{a,b\}^* \mid |x| \geq 2\}$ eine rechtslineare Grammatik mit $L = L(G)$
 $G = \{N, \Sigma, P, S\}$ mit

$$N = \{S, A\}, \Sigma = \{a, b\} \text{ und } P = \{ S \xrightarrow[G]{1} aaA \mid abA \mid baA \mid bba, A \xrightarrow[G]{1} aA \mid bA \mid \varepsilon \}$$

Behauptung: $L = L(G)$

1. Sei $x \in L(G) \rightarrow S \xrightarrow[G]{*} x$, nun ist zu zeigen, dass für $S \xrightarrow[G]{*} \alpha : \alpha \in \{ xy \mid x \in \{aa, ab, ba, bb\} \text{ und } y \in \Sigma^* \vee \Sigma^+ A \}$
2. Sei $x \in L \rightarrow |x| \geq 2$, Zeige mit Induktion nach $|x|$, dass $S \xrightarrow[G]{*} x$ gilt.

Satz III.1.1

Ist L eine endliche Sprache über einem Alphabet Σ , so gibt es eine rechtslineare Grammatik G mit $L(G) = L$

d.h. jede endliche Sprache ist rechtslinear

Beweis: Sei L endlich: $L = \{x_1, x_2, \dots, x_n\}$, dann ist $G = (\{S\}, \Sigma, \{S \rightarrow x_i \mid 0 < i \leq n\}, S)$ die gesuchte rechtslineare Grammatik.

Alternative Grammatiktypen, die die gleiche Sprachklasse erzeugen:

Typ	P-Form
rechtslinear	$A \rightarrow xB, A \rightarrow x$
streng rechtslinear	$A \rightarrow aB, A \rightarrow \varepsilon$
linkslinear	$A \rightarrow Bx, A \rightarrow x$
streng linkslinear	$A \rightarrow Ba, A \rightarrow \varepsilon$

Satz III.1.2

Sei L eine beliebige Sprache über einem Alphabet Σ . L ist genau dann regulär, wenn sie rechtslinear ist.

Beweis [Idee mit Beispiel]:

„ \Leftarrow “: Sei $L = L(G)$ für eine rechtslineare Grammatik $G=(N, \Sigma, P, S)$. Konstruiere einen NEA mit \sim Transitionen, so dass $L(G) = L(M)$

$M = (Q, \Sigma, \Delta, [S], \{[\epsilon]\})$ mit Δ wie folgt:

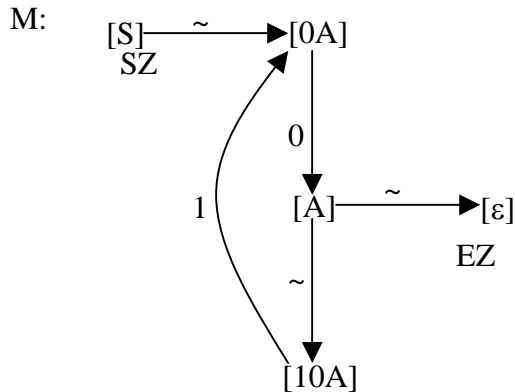
1. Ist $A \in N$ (Nichtterminal), so ist $\Delta([A], \sim) = \{[\alpha] \mid A \rightarrow \alpha \in P\}$
2. Ist $a \in \Sigma$ und $\alpha \in \Sigma^* \vee \Sigma^* N$, dann ist $\Delta([a\alpha], a) = \{[\alpha]\}$

zu zeigen wäre: $[\alpha] \in \Delta_1([S], w)$ genau dann wenn $S \xrightarrow[G]{*} xA \xrightarrow[G]{1} xy\alpha$ mit

$A \rightarrow y\alpha \in P$ und $xy = w$ oder
 $w = \epsilon$ falls $\alpha = S$

Im Beispiel: Sei $L = \{0, 0(10)^*\}$

$G = (\{S, A\}, \{0, 1\}, \{S \rightarrow 0A, A \rightarrow 10A \mid \epsilon \text{ falls } \alpha\}, S)$ erzeugt L



ist der gesuchte NEA mit \sim -Transitionen

„ \Rightarrow “: Sei $L = L(M)$ für einen DEA $M = \{Q, \Sigma, \delta, q_0, F\}$

Fall1: $q_0 \notin F$

Konstruiere $G = \{Q, \Sigma, P, q_0\}$ mit $P = \{p \rightarrow aq \mid \delta(p,a) = q\} \vee \{p \rightarrow a \mid \delta(p,a) \in F\}$

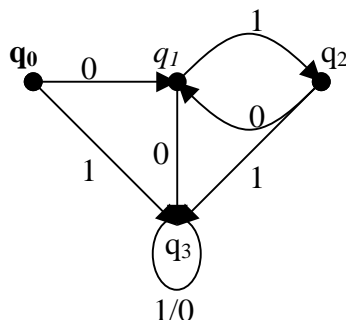
zu zeigen wäre: $\delta_1(q_0, x) \in F$ genau dann wenn $q_0 \xrightarrow[G]{*} x$

Fall2: $q_0 \in F$

wie Fall 1 + Erweitere G zu $G' = \{Q \vee \{S\}, \Sigma, P \vee \{S \rightarrow q_0 \mid \epsilon\}, S)$

G' ist rechtslinear und $L(G') = L(G) \vee \{\epsilon\}$

Im Beispiel:



M akzeptiert $L = \{0(10)^*\}$

	$p \rightarrow aq \mid \delta(p,a) = q$	$p \rightarrow a \mid \delta(p,a) \in F$
$q_0 \rightarrow$	$0q_1 \mid 1q_3$	$\mid 0$
$q_1 \rightarrow$	$0q_3 \mid 1q_2$	
$q_2 \rightarrow$	$0q_1 \mid 1q_3$	$\mid 0$
$q_3 \rightarrow$	$0q_3 \mid 1q_3$	

III.2 Kontextfreie Sprachen

Wegen des Pumping-Lemmas gibt es also keine rechtslineare Grammatik für

$$\text{z.B. } L = \{a^n b^n \mid n \geq 0\}$$

III.2.1. Definition Kontextfreie Grammatik

Eine Kontextfreie Grammatik ist ein 4-Tupel $G = (N, \Sigma, P, S)$ mit N, Σ, S wie bei den rechtslinearen Grammatiken und P eine endliche Teilmenge von $N \times (N \cup \Sigma)^*$

Beispiel: $G = (\{S\}, \{a,b\}, P, S)$ mit $P = \{S \rightarrow aSb, S \rightarrow \varepsilon\} \rightarrow$ erzeugt $L = \{a^n b^n \mid n \geq 0\}$

III.2.2. Definition Chomsky-Normalform (CNF)

Eine kontextfreie Sprache ist in CNF, falls alle Produktionsregeln die Form $A \rightarrow BC$ oder $A \rightarrow a$ haben ($A, B, C, \in N$ und $a \in \Sigma^*$)

Satz III.2.1.

Für jede kontextfreie Grammatik G gibt es eine kontextfreie Grammatik G' in CNF, so dass

$$L(G') = L(G) \setminus \{\varepsilon\}$$

Beweisidee: Transformiere G in G' , so dass G' in CNF

1. Setze $P' = P$ und wiederhole die Schritte i) und ii) bis sich P' nicht mehr verändert

i) ε - Reduktion

Ist $A \rightarrow \alpha\beta \in P$ und $B \rightarrow \varepsilon \in P$ so ist $A \rightarrow \alpha\beta$ aus P'

ii) EINER - Reduktion

Ist $A \rightarrow B \in P$ und $B \rightarrow \alpha \in P$ so ist $A \rightarrow \alpha$ aus P'

2. Entferne alle $A \rightarrow \varepsilon$ und $A \rightarrow B$

Es gilt: $L(G') = L(G) \setminus \{\varepsilon\}$

3. $P' \cup \{A_a \rightarrow a \mid \forall a \in \Sigma\}$ (für jedes Terminalsymbol wird ein Nichtterminalsymbol erzeugt)

4. Ist $A \rightarrow \alpha$ in P' , so ersetze α durch α' , indem jedes Vorkommen von $\alpha \in \Sigma$ in α durch A ersetzt wird ($S \rightarrow aBaCb$ wird zu $S \rightarrow A_a B A_a C A_b$)

5. Alle Produktionen in P' haben die Form $A \rightarrow \alpha$ oder $A \rightarrow B_1 B_2 B_3 \dots B_k$ ($k \geq 2$)

ist $k > 2$, so ersetze $A \rightarrow B_1 B_2 B_3 \dots B_k$ durch $A \rightarrow B_1 C$, $C \rightarrow B_2 B_3 \dots B_k$ für ein neues Symbol C

6. Wiederhole 5. bis $k = 2$ für alle Regeln $A \rightarrow B_1 B_2 B_3 \dots B_k \rightarrow$ Die so entstandene Grammatik G' ist in CNF und es gilt $L(G') = L(G) \setminus \{\varepsilon\}$

Beispiel: $S \rightarrow aSb \mid \varepsilon$

1. i) $S \rightarrow aSb \mid ab \mid \varepsilon$

ii) keine EINER-Regel zu bearbeiten

2. $S \rightarrow aSb \mid ab$

3. $S \rightarrow aSb \mid ab$, $A_a \rightarrow a$, $A_b \rightarrow b$

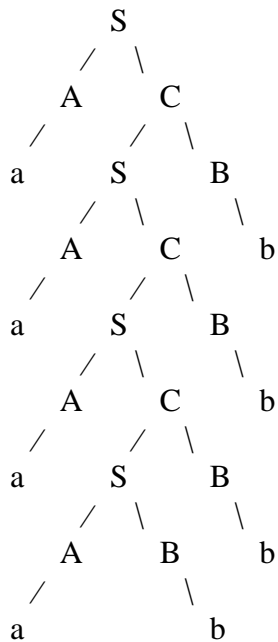
4. $S \rightarrow A_a S A_b \mid A_a A_b$, $A_a \rightarrow a$, $A_b \rightarrow b$

5. $S \rightarrow A_a C \mid A_a A_b$, $A_a \rightarrow a$, $A_b \rightarrow b$, $C \rightarrow S A_b$

Zur Darstellung der Ableitungen werden häufig Ableitungsbäume (Parse-Trees) benutzt

Beispiel: Ableitungsbaum für a^4b^4 mit Hilfe obiger Grammatik

$S \rightarrow AC \mid AB, A \rightarrow a, B \rightarrow b, C \rightarrow SB$



Ableitungsbäume werden zur Feststellung der syntaktischen Korrektheit von Programmen eingesetzt. Ferner dienen sie zum Nachweis des Pumping-Lemmas für kontextfreie Sprachen. Zu einem Ableitungsbaum kann es verschiedene Ableitungen geben

Typisch sind: Linksableitungen: $S \xrightarrow[G]{1} AC \xrightarrow[G]{1} aC \xrightarrow[G]{1} aSB \xrightarrow[G]{1} aACB \xrightarrow[G]{1} aaCB \xrightarrow[G]{1} aaSBB$
 $\xrightarrow[G]{1} aaACBBB \xrightarrow[G]{1} aaaCBBB \xrightarrow[G]{1} \dots \xrightarrow[G]{1} aaaabbbb$

Rechtsableitungen: $S \xrightarrow[G]{1} AC \xrightarrow[G]{1} ASB \xrightarrow[G]{1} ASb \xrightarrow[G]{1} AACb \xrightarrow[G]{1} AASBb \xrightarrow[G]{1} AASbb$
 $\xrightarrow[G]{1} AAACbb \xrightarrow[G]{1} \dots \xrightarrow[G]{1} aaaabbbb$

III.3 Kellerautomaten

Intuitiv ist ein Kellerautomat ein EA mit zusätzlichem Speicher, dem Keller. Der Keller funktioniert nach dem LIFO-Prinzip (LastInFirstOut \rightarrow Stack)

III.3.1 Definition Kellerautomat

a) ein nichtdeterministischer Kellerautomat (NKA) ist ein 7-Tupel:

$M = \{Q, \Sigma, \Gamma, \delta, s, \perp, F\}$ mit

Q : eine endliche Menge von Zuständen

Σ : ein endliches Eingabealphabet

Γ : ein endliches Kelleralphabet

δ : eine endliche Anzahl von Transitionen: $\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$

$\perp \in \Gamma$: das initiale Kellersymbol

$F \subseteq Q$: die akzeptierenden Zustände

$S \subseteq Q$: der Startzustand

$((p, a, A), (q, B_1 B_2 \dots B_k))$ bedeutet:

Ist M im Zustand p und a das aktuelle Eingabesymbol und ist A das oberste Kellersymbol, so gehe in Zustand q und ersetze A durch $B_1 B_2 \dots B_k$.

$((p, \varepsilon, A), (q, B_1 B_2 \dots B_k))$: wie oben, nur dass keine Eingabe verarbeitet wird

b) eine Konfiguration ist ein Tripel $Q \times \Sigma^* \times \Gamma^*$ die den aktuellen Zustand des Kellerautomaten beschreibt

z.B. $(q, a_1 \dots a_n, B_1 \dots B_k \perp)$

(s, x, \perp) ist die Startkonfiguration

c) Ein NKA M akzeptiert Eingabe x , falls $(s, x, \perp) \xrightarrow[M]{*} (q, \varepsilon, \gamma)$ für $q \in F$ und $\gamma \in \Gamma^*$

Alternativ:

Ein NKA M akzeptiert Eingabe x mit leerem Keller, falls $(s, x, \perp) \xrightarrow[M]{*} (q, \varepsilon, \varepsilon)$ für $q \in Q$

Beide Varianten sind äquivalent. Wir geben die jeweilige Variante an.

III.3.1 Beispiel

Sei $Q = \{q\}$, $\Sigma = \{<, >\}$, $\Gamma = \{\perp, <\}$, $s = q$, initiales Kellersymbol: \perp , $F = \emptyset$, NKA akzeptiert mit leerem Keller

$\delta = ((q, <, \perp), (q, <))$

$((q, >, <), (q, \varepsilon))$

$((q, <, <), (q, <<))$

$((q, \varepsilon, \perp), (q, \varepsilon))$

Satz III.3.1

Sei L eine Sprache über Σ

L ist genau dann kontextfrei (d.h. wird von einer kontextfreien Grammatik erzeugt), wenn L von einem NKA akzeptiert wird

III.3.2 Definition deterministischer KA (DKA)

Ein NKA heißt DKA, falls

1. es genau eine Transition für jede mögliche Situation (Konfiguration) gibt.
d.h. $((p,a,A),(q,\beta))$ oder $((p,\varepsilon,A),(q,\beta)) \forall p \in Q, a \in \Sigma, A \in \Gamma$
2. das Bodenelement \perp nie verschwindet
d.h. $((p,a,\perp),(q,\beta,\perp)) \forall p \in Q, a \in \Sigma$

Satz III.3.2

Es gibt Sprachen, die kontextfrei, aber nicht deterministisch kontextfrei sind.

Bew.: $L = \{a,b\}^* \setminus \{ww \mid w \in \{a,b\}^*\}$

Satz III.3.3

Es gibt Sprachen, die nicht kontextfrei sind.

Bew.: $L_1 = \{ww \mid w \in \{a,b\}^*\}$

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

III.4 Die Chomsky Hierarchie

III.4.1 Definition

Eine Grammatik heißt ‚vom Typ 0‘, falls ihre Produktionsregeln die Form $\alpha \rightarrow \beta$ haben für $\alpha, \beta \in (N \cup \Sigma)^*$

Beispiel: $\alpha B A b C \rightarrow b A$
 $a \rightarrow A b A b$

Eine Grammatik heißt ‚vom Typ 1‘ (oder kontext-sensitiv), falls ihre Produktionsregeln die Form $\alpha \rightarrow \beta$ haben für $a, \beta \in (N \cup \Sigma)^*$ und $|\alpha| \leq |\beta|$

Beispiel: $A B a \rightarrow a B B B$ (erlaubt)
 $a A b B A \rightarrow a$ (nicht erlaubt)

Eine Grammatik heißt ‚vom Typ 2‘, falls sie kontextfrei ist.

Eine Grammatik heißt ‚vom Typ 3‘, falls sie rechtslinear ist.

Satz III.4.2

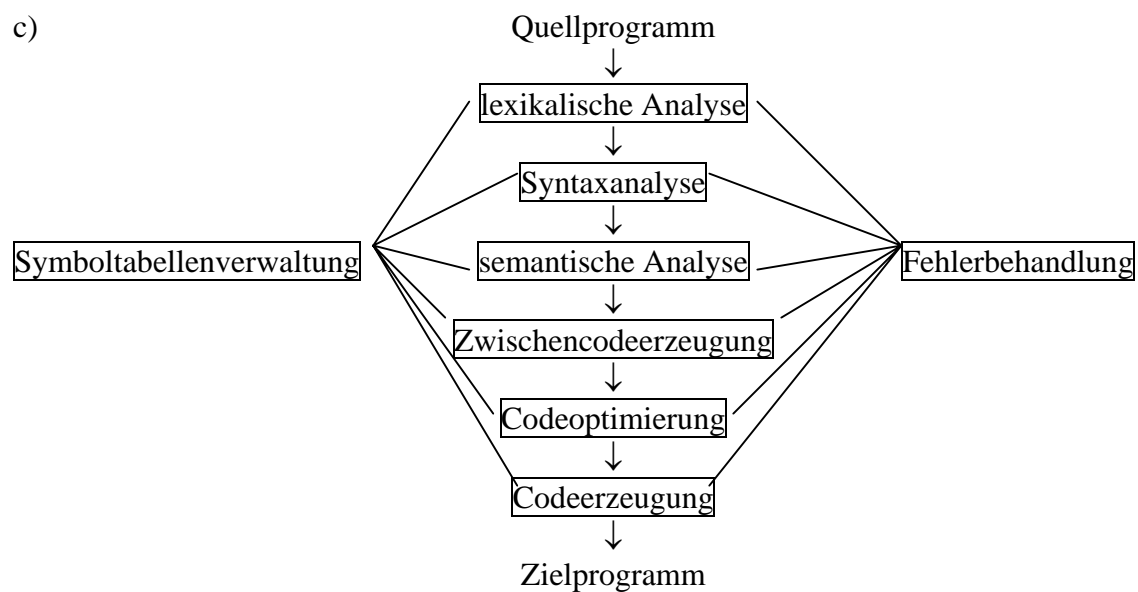
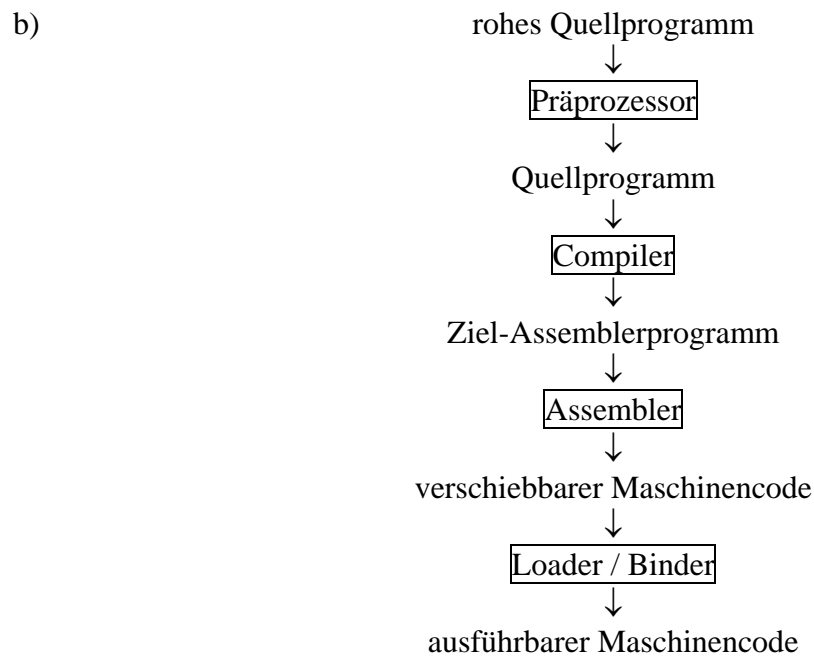
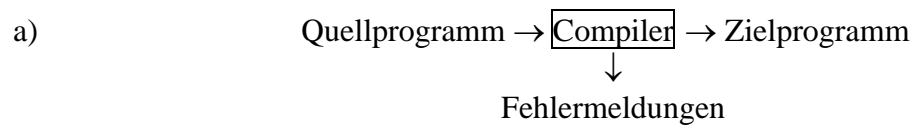
Sei G_i eine Grammatik vom Typ i , so gibt $L(G_i) \supset L(G_{i+1})$ für $0 \leq i < 3$ wobei $L(G_i)$ die Menge aller Sprachen vom Typ G_i ist.

Bemerkungen

- die Tourringmaschinen sind die erkennenden Automaten der Typ-0 Sprachen
 - Die einseitig beschränkten Tourringmaschinen (Band hat definierten Anfang) sind die erkennenden Automaten der Typ-1 Sprachen
- es gibt eine Hierarchie der Automaten

IV. Anwendung im Compilerbau

IV.1 Grundlagen Compiler



Ein Programm erkennt nicht POSITION, sondern die einzelnen Zeichen ,P', ,O', ,S', ...

IV.2 Backus-Naur-Form

Programme werden oft in BNF beschrieben

<stmt>	::= <if-stmt> <while-stmt> <begin-stmt> <assign-stmt>	(I → I W B' A)
<if-stmt>	::= if <bool-expr.> then <stmt> else <stmt>	(I → iBtSeS)
<while-stmt>	::= while <bool-expr.> do <stmt>	(W → wBdS)
<begin-stmt>	::= begin <stmt-list> end	(B' → bLe')
<stmt-list>	::= <stmt> ; <stmt-list>	(L → S S;L)
<assign-stmt>	::= <var> := <const>	(A → V := C)
<bool-expr.>	::= <arith.-expr.> <compare-op.> <arith.-expr.>	
<arith.-expr.>	::= <var> <const> <arith.-expr.> <arith.-op> <arith.-expr.>	
<compare-op>	::= < > ≤ ≥ = ≠	
<arith.op.>	::= + - * /	
<const>	::= 0 1 2 3 4 5 6 7 8 9	
<var>	::= a b c ... x y z	

Liebt man die Ausdrücke in den eckigen Klammern <...> als Nichtterminale und alle fetten Worte plus die Zeichen als Terminale, so stellt die BNF gerade die Menge der Produktion der Grammatik dar.

IV.3. Scanner - Lexikalische Analyse

Gesucht: Programm, das die Tabelle erzeugt mit

Input: Ein Programm als Folge von Zeichen

Output: Tabelle mit Variablen

Beispiel:

WHILE $x \leq y$ DO BEGIN $x := (x+1)$; $y = (y+1)$; END EOF

Grundfunktion: Read(Zeichen) liest ein Zeichen und setzt den Eingabezeiger eins nach rechts

Schlüsselworte erkennen:

Idee: ein EA, der alle Schlüsselworte erkennt

Schlüsselworte sind:

WHILE $\delta(0,B) \rightarrow „B“$, $\delta(„B“,E) \rightarrow „BE“$, $\delta(„BE“,G) \rightarrow „BEG“$, $\delta(„BEG“,I) \rightarrow „BEGI“$,
 $\delta(„BEGI“,N) \rightarrow „BEGIN“$
DO $\delta(0,D) \rightarrow „D“$, $\delta(„D“,O) \rightarrow „DO“$
BEGIN ...
END
IF
THEN
ELSE
:= <
; >
+ ≤
- ≥
* =
/ ≠

Umsetzung:

```
READ(Zeichen);
While Zeichen ≠ EOF Do Begin
  Zustand :=Delta(Zustand, Zeichen);
  If Zustand ∈ F Then Reserviertes Wort erkannt;
  Read(next_Zeichen);
  Case (next_Zeichen) of
    „~“: Variable gefunden, Tabelleneintrag    (~: Leerzeichen)
    „.“: Variable gefunden, Tabelleneintrag
    „+“: Variable gefunden, Tabelleneintrag
    „-“: Variable gefunden, Tabelleneintrag
    ...
    „A“: Wort gelesen, Zeichen := next_Zeichen;
    „B“: Wort gelesen, Zeichen := next_Zeichen;
    ...
  End
```

mit F = array of String
F[0] = „While“
F[1] = „Else“
...

Zusammenfassung:

Scanner führen die lexikalische Analyse des Programms durch



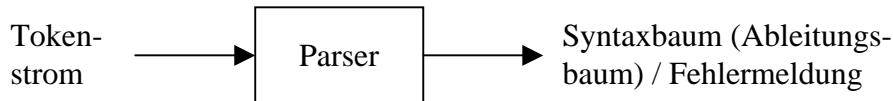
“W“,“H“,“I“,“L“,“E“	→	While
“.“,“=“	→	:=
“I“,“N“,“T“,“T“	→	id und Eintrag in Symboltabelle

Gesucht: Ein Programm zum Scannen mit
Input: Ein Programm als Textfile
Output: Tokenfolge mit Symboltabelle

- IDEE: 1. NEA zum Erkennen der reservierten Wörter
2. Erweitern mit Look-Ahead und Generierung der Symboltabellen
3. Umsetzung in (Pseudo-)Code mit
i) Realisierung der Zustandsübergänge
ii) großes CASE-Statement

IV.4 Parser - Syntaxanalyse

Ein Parser führt syntxgesteuerte Übersetzung aus. Hauptaufgabe ist es, zu erkennen, ob ein Programm syntaktisch korrekt ist (d.h. $w \in L(G)$). Ist dies nicht der Fall, müssen fehlerhafte Teile angezeigt werden. Falls das Programm syntaktisch korrekt ist, kann zur Codeerzeugung übergegangen werden.



Gesucht: Ein Programm zum Parsen mit

Input: Programm als Tokenfile

Output: Syntaxbaum, ggf. Fehler

IDEE: Kellerautomat mit impliziten Aufbau des Syntaxbaums

Konstruktion für:

$E \rightarrow (EBE \mid UE) \mid C \mid V$

$B \rightarrow \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$

$U \rightarrow \neg$

$C \rightarrow \perp \mid T$

$V \rightarrow P \mid Q \mid R \mid \dots$

$((P \vee Q) \wedge R) \vee (Q \wedge (\neg P))$

Beschreibung des KA, der boolesche Ausdrücke parst:

READ(Token)

CASE Token OF

(: PUSH(„(“)

^ : PUSH(„^“)

∨ : PUSH(„∨“)

¬ : PUSH(„¬“)

P : PUSH(↑.Symboltab[P])

Q : PUSH(↑.Symboltab[Q])

R : PUSH(↑.Symboltab[R])

) : REDUCE

END

REDUCE:

Alloziere Platz für Baumerweiterungen

POP(Token)

IF (Token ≠ ↑) THEN WRITE(ERROR)

ELSE Syntaxbaum.rechts := Token

POP(Token)

IF (Token ∉ {^, ∨, ⇒, ⇔, ¬}) then WRITE(ERROR)

ELSE Syntaxbaum.root := Token

IF (Token ≠ ¬) THEN

POP(Token)

IF (Token ≠ ↑) THEN WRITE(ERROR)

ELSE Syntaxbaum Syntaxbaum.links :=Token

ELSE POP(Token)

IF (Token ≠ „(“) THEN WRITE(ERROR)

PUSH(↑.root)

IV. Effiziente Parser

Prinzipiell wird in Bottom-Up Parser und Top-Down Parser unterschieden

Bottom-Up Parser bauen den Syntaxbaum von den Blättern zur Wurzel auf (siehe IV.4)

- typische Vertreter: LR(1)-Parser

L Left to right Abarbeitung

R Reproduktion von Rechtsableitungen

1 1 Zeichen Look-ahaed

- Problem: Erkennen der richtigen rechten Seite der Regel

Bsp.: $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

abbcde $A \rightarrow b$

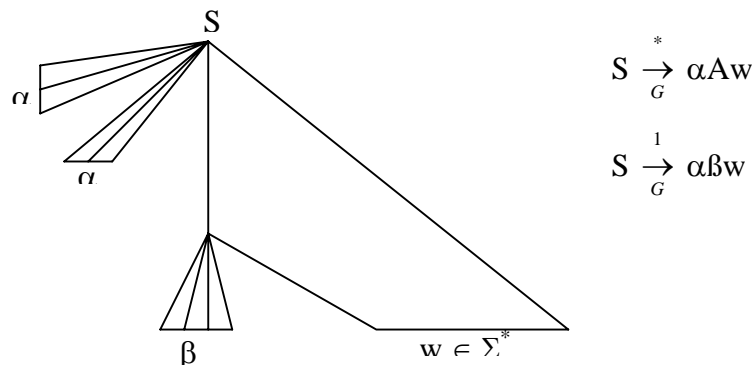
aAbcde $A \rightarrow Abc$

aAde $B \rightarrow d$

aABe $S \rightarrow aABe$

S

Allgemein:



Der Syntaxbaum stellt die aktuelle Satzform dar (gemäß einer Rechtsableitung). β heißt „Handle“. Kennt man diesen Handle, so kann man reduzieren. Aber man muß sich auch noch a merken.

Idee: Erweitere die Grammatik $A \rightarrow \alpha.\beta,a$ bedeutet: habe α gelesen und a ist der nächste Buchstabe in w

$A \rightarrow \alpha.,a$ bedeutet: habe α komplett gelesen und kann reduzieren falls a der nächste Buchstabe ist

Top-Down Parser bauen den Syntaxbaum von der Wurzel zu den Blättern auf. Steuerung durch eine Parser-Tabelle

- typische Vertreter: LL(1)-Parser

L Left to right Parser

L Reproduktion von Linksableitungen

1 1 Zeichen Look-ahaed

- Bsp.: $S \rightarrow iEtS$ if <expr> then <stmt>

$S \rightarrow iEtSeS$ if <expr> then <stmt> else <stmt>

$S \rightarrow a$

$E \rightarrow b$

if b then if b then a else a : ibtibtaea

→ Mehrdeutig!

Lösungsansatz:

Linksfaktorisierung:

$$S \rightarrow iEtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$E \rightarrow b$$

Manche Mehrdeutigkeiten können dadurch aufgelöst werden. Hier ist es allerdings nicht der Fall. Aber man muss die Entscheidung, wann eine Regel angewendet wird, erst „in letzter Minute“ treffen.

Im Algorithmus wird eine Parse-Tabelle benutzt:

N	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow eS$ $S' \rightarrow \varepsilon$			$S' \rightarrow \varepsilon$
E		$E \rightarrow b$				

→ Doppeleinträge sind ein Zeichen, dass die Grammatik nicht LL(1) ist.