

Compilerbau I

WS 93/94

Prof. Dr. W.-M. Lippe

17. November 1993

Inhaltsverzeichnis

1	Sprachübersetzer	3
1.1	Compiler	3
1.2	Interpreter	8
1.3	Arbeitsweise eines Compilers.....	9
1.4	Die Arbeitungsgänge zur Compilierung eines Programms	10
1.4.1	Lexikalische Analyse.....	10
1.4.2	Syntaxanalyse	11
1.4.3	Semantische Analyse.....	13
1.4.4	Erzeugung eines Zwischencodes	14
1.4.5	Erzeugung des Zielcodes	14
2	Grammatiken	16
2.1	Kontextfreie Grammatiken.....	16
2.2	Eigenschaften von (kontextfreien) Grammatiken	21
2.3	Eindeutigkeit von kontextfreien Grammatiken.....	23
3	Lexikalische Analyse	29
3.1	Reguläre Ausdrücke	30
3.2	Automaten.....	31
3.2.1	(Nichtdeterministische) Endliche Automaten.....	31
3.2.2	Deterministische endliche Automaten.....	34
3.2.3	Minimierung der Zustandsmenge eines endlichen Automaten	36
4	Syntaxanalyse	43
4.1	Nichtdeterministische Verfahren zur Syntaxanalyse.....	43
4.1.1	Bottom-up-Analyse mit Backtracking	44
4.1.2	Top-down-Analyse mit Backtracking	46
4.2	Kellerautomaten	50
4.2.1	(Nichtdeterministische) 1-Weg-Kellerautomaten	51
4.2.2	Deterministische 1-Weg-Kellerautomaten	57
4.2.3	Leistungsfähigkeit von Kellerautomaten.....	58

4.3	Analysestrategien	59
4.3.1	Tabellenorientierte Analysestrategien	60
4.3.1.1	Analysealgorithmus von Cocke-Kasami-Younger	60
4.3.2	Ableitungsorientierte Analysestrategien.....	63
4.3.2.1	Deterministische Bottom-up-Analyse (kanonische Reduktion)	65
4.3.2.2	LR(k)-Analyse	72
4.3.2.3	LL(k)-Analyse.....	90
4.3.2.4	Tabellengesteuerte LR(k)-Analyse.....	91
4.3.3	Zusammenfassung der verschiedenen Analysestrategien	103
5	Semantische Analyse	113
5.1	Aufgaben der semantischen Analyse.....	113
5.2	Hilfsmittel zur semantischen Analyse	114
5.3	Typbestimmung für "dicke" Ausdrücke mit Hilfe eines Kellers	118
5.4	Algorithmen zur semantischen Analyse (eines Programms in MMS).....	119
5.4.1	Überprüfe Anweisungsteil.....	119
5.4.2	Überprüfe Anweisung	119
5.4.3	Überprüfe bedingte Anweisung	119
5.4.4	Überprüfe Bedingung.....	120
5.4.5	Überprüfe then-Teil	120
5.4.6	Überprüfe else-Teil	120
5.4.7	Überprüfe Wiederholungsanweisung	120
5.4.8	Überprüfe do-Teil.....	121
5.4.9	Überprüfe Wertzuweisung	121
5.4.10	Modifikationen für "in" und "out"	121
5.4.11	Überprüfe Eingabe.....	122
5.4.12	Überprüfe Ausgabe.....	122
A	Münsteraner-Minisprache (MMS)	123
	Literaturverzeichnis	126
	Stichwortverzeichnis	127

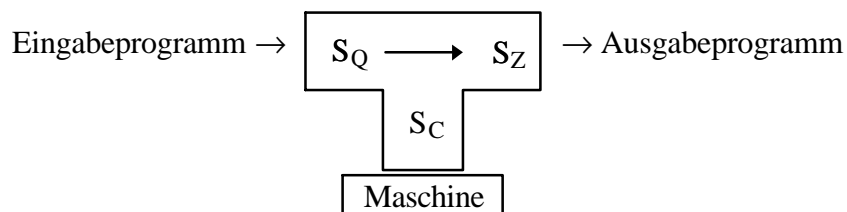
1 Sprachübersetzer

Die Aufgabe von Sprachübersetzern ist es, Programme einer Quellsprache in funktional äquivalente Programme einer Zielsprache zu transformieren. Dabei ist in der Regel die Quellsprache eine höhere Programmiersprache und die Zielsprache eine Maschinsprache.

Die zur Sprachübersetzung verwendeten Programme lassen sich in zwei Klassen einteilen:

1.1 Compiler

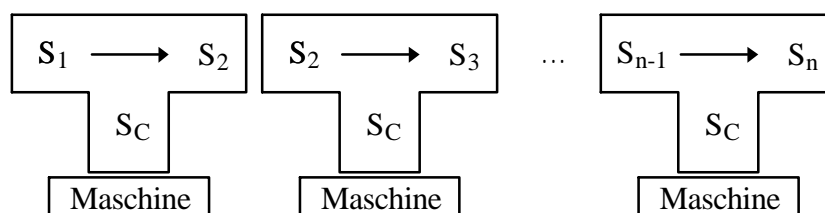
Compiler übersetzen Sätze aus einer Sprache S_Q (Quellsprache) in eine Sprache S_Z (Zielsprache), wobei der Compiler selbst in einer Sprache S_C formuliert ist; erst nach dieser Übersetzung kann ein in S_Q geschriebenes Programm ausgeführt werden. Ein solcher Compiler wird durch die folgende Figur dargestellt:

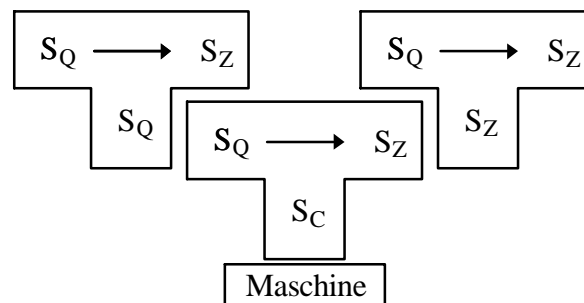


Diese Form eines T hat die Bezeichnung *T-Diagramme* geprägt. Zu beachten ist, daß das (unterste) T-Diagramm stets mit einer Maschine oder einem Interpreter verbunden sein muß.

Compiler können auf verschiedene Art und Weise zusammenwirken:

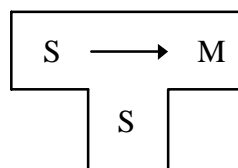
1. Compilierung über mehrere Zwischensprachen (Mehrphasen-Compilation)



2. **Compilierung eines Compilers**3. **Bootstrapping**

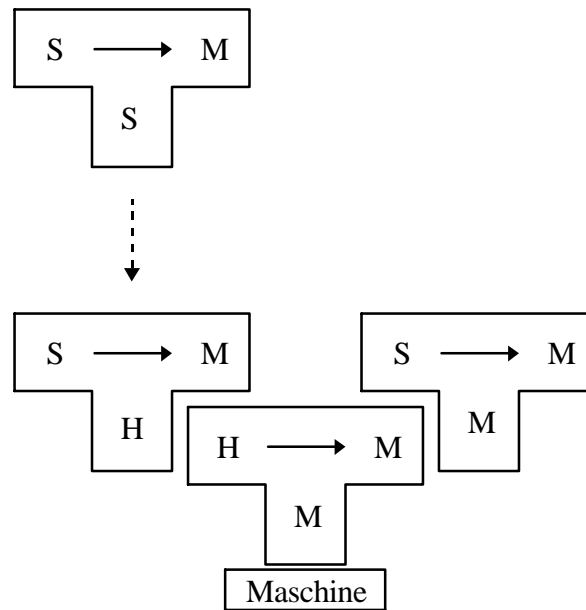
Da ein Compiler selten von Anfang an in endgültiger Fassung in Angriff genommen wird, wird vielmehr ein Kern der Sprache implementiert, worauf das Endziel schrittweise angenähert wird.

Gegeben sei ein Compiler für eine Quellsprache S , wobei der Compiler in S formuliert ist:



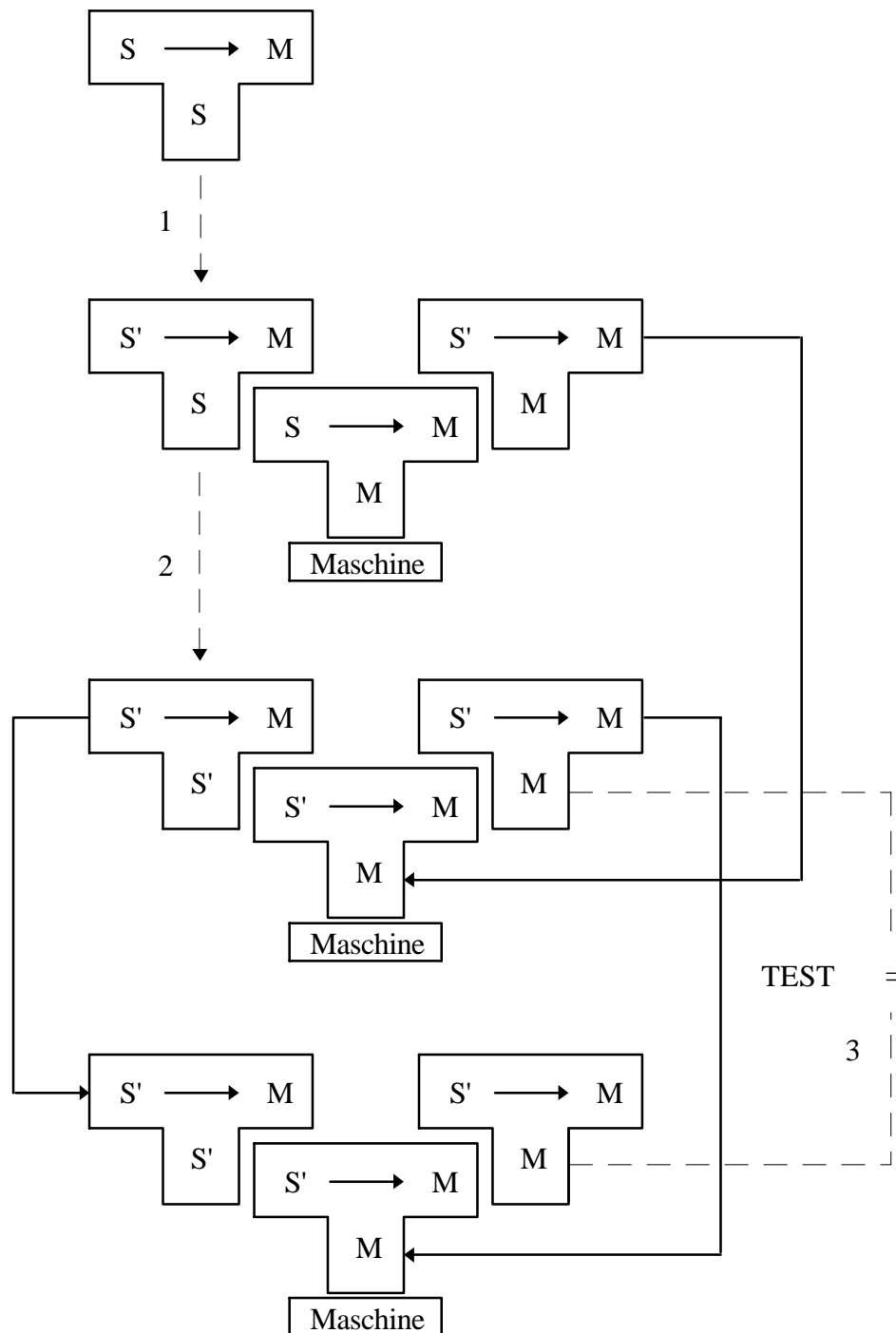
Nun soll S in eine Version S' weiterentwickelt werden. Dieses geschieht in drei Schritten, nämlich (1) durch Programmierung der Erweiterungen in S und (2) durch Neuformulierung des Compilers, in der die neuen Sprachelemente nutzbringend verwendet werden. Der große Vorteil der Formulierung eines Compilers in seiner eigenen Quellsprache besteht darin, daß der Compiler selbst von seiner Erweiterung sofort profitieren kann. Deshalb kann die Entwicklung des Compilers mit einer relativ einfachen Sprache beginnen und durch Wiederholung dieser zwei Schritte zum Endprodukt emporgehoben werden. Der Fachausdruck für dieses sich selbst Hochziehen lautet *bootstrapping*. In einem dritten Schritt wird die Richtigkeit der Umprogrammierung des Compilers von S nach S' überprüft. Diese Überprüfung erfolgt rein mechanisch, indem festgestellt wird, ob die Resultate der Schritte 2 und 3 genau identisch sind. Die Existenz eines solchen rigorosen Tests ist für die Praxis von unschätzbarem Wert.

In einem Anfangsschritt muß allerdings der gegebene Compiler so modifiziert werden, daß er nicht mehr in S, sondern in M formuliert ist. Die übliche Lösung liegt in der Übersetzung "von Hand" in eine andere, auf der Maschine verfügbare Programmiersprache H. Es hat sich als vorteilhaft erwiesen, dafür ebenfalls eine höhere Sprache anstelle von Assembler-Code zu verwenden. Den Anfangsschritt zeigt die folgende Abbildung:



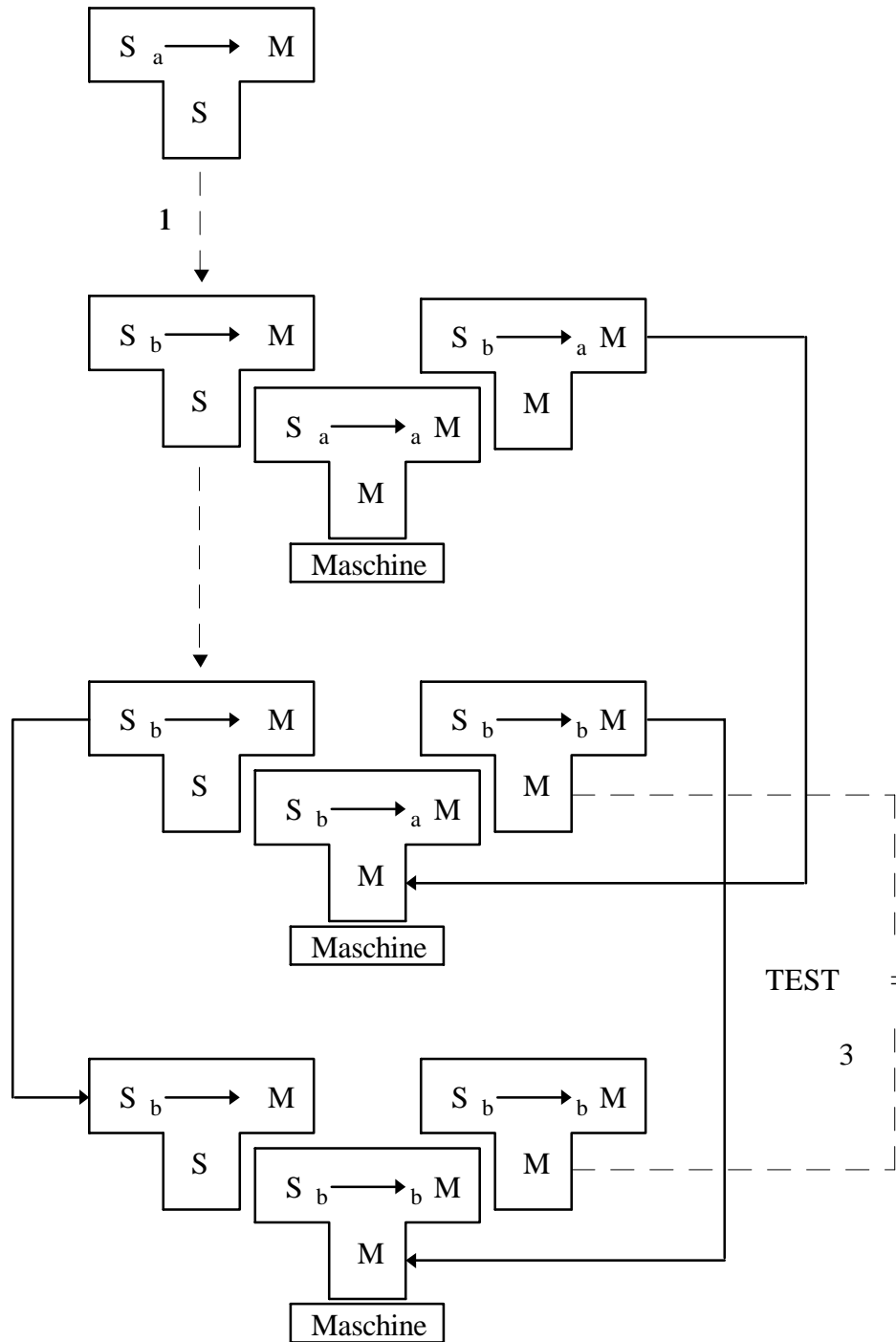
Der Anfangsschritt wird durch den glücklichen Umstand wesentlich erleichtert, daß die Programmierung in der "fremden" Hilfssprache H durchaus ohne Rücksicht auf Effizienz erfolgen darf. Die Ineffizienz des Initialproduktes pflanzt sich nämlich nicht fort und macht sich nur im ersten Bootstrap bemerkbar, wonach das Hilfsprodukt weggeworfen werden kann.

Die folgende Abbildung zeigt einen Bootstrap eines Compilers von S nach S'.



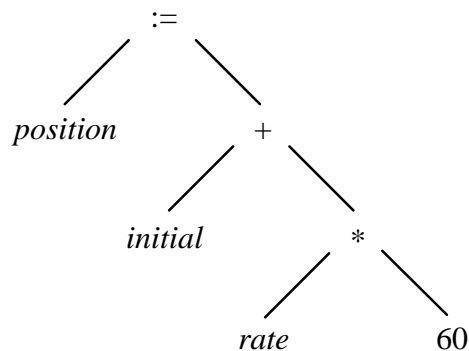
Tatsächlich braucht ein Bootstrap nicht unbedingt eine Spracherweiterung zu beinhalten. Er kann sich auch auf eine Verbesserung der Compilationsmethode und des erzeugten Codes beziehen. Einen solchen Bootstrap zeigt die nachfolgende Abbildung. In der Notation $S \overset{x}{\rightarrow} \overset{y}{M}$ bezeichne $x=a$ die Verwendung der alten, $x=b$ der verbesserten Compilationsmethode oder Codegenerierung, $y=a$, daß der Compiler selbst durch die alte, $y=b$ durch die verbesserte Version des Compilers erzeugt wurde. Offensichtlich erlaubt also die Methode

des Bootstrapping die anfängliche Verwendung einfacher, aber unter Umständen nicht optimaler Codeerzeugung und offeriert die Möglichkeit der nachträglichen Verbesserung. Ist der Compiler in der eigenen Quellsprache formuliert, so kommt er als erstes Programm in den Genuß dieser Verbesserungen. Die angefügte Testphase erlaubt wiederum die Prüfung, ob die verbesserte Version des Compilers korrekt ist und sich selbst zu reproduzieren vermag.



1.2 Interpreter

Anstatt ein Zielprogramm als Übersetzung zu erzeugen, führt ein Interpreter die im Quellprogramm enthaltenen Operationen direkt aus. Ein Interpreter könnte zum Beispiel für die Anweisung $position := initial + rate * 60$ einen Baum aufbauen



und anschließend beim Durchwandern des Baums die Operationen in den Knoten ausführen. An der Wurzel würde er feststellen, daß er eine Zuweisung ausführen muß, und deswegen eine Routine zur Auswertung des Ausdrucks auf der rechten Seite aufrufen. Der Ergebniswert würde anschließend an diejenige Stelle gespeichert werden, die dem Bezeichner *position* zugeordnet ist. Beim rechten Sohn der Wurzel würde die Routine feststellen, daß sie die Summe zweier Ausdrücke berechnen muß. Sie würde sich selbst rekursiv aufrufen, um den Wert des Ausdrucks $rate * 60$ zu berechnen. Anschließend würde sie diesen Wert zum Wert der Variablen *initial* addieren.

Interpreter werden häufig dazu benutzt, Kommandosprachen auszuführen, weil jeder in einer Kommandosprache ausgeführte Operator typischerweise die Aktivierung einer komplexen Routine darstellt, wie zum Beispiel einen Editor oder Compiler. Ähnlich werden auch "sehr hohe" Sprachen wie APL interpretiert, weil viele datenspezifische Eigenschaften wie Größe und Form von Arrays zur Übersetzungszeit noch nicht bestimmt werden können.

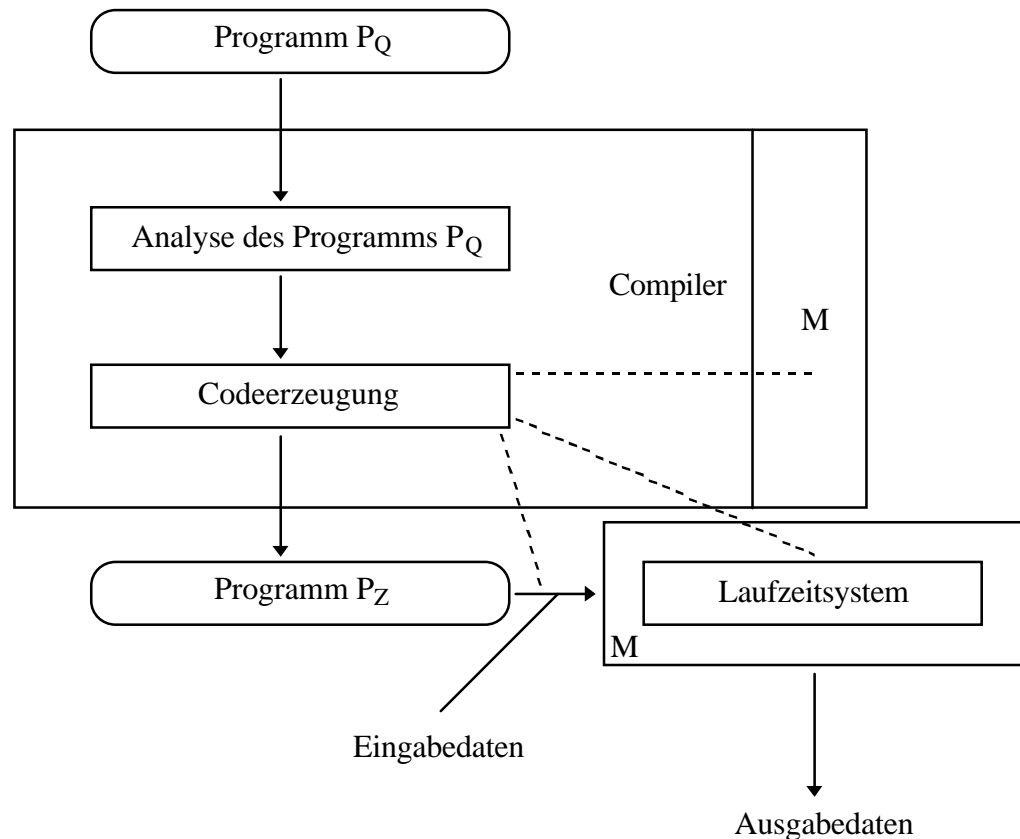
Formal läßt sich ein Interpreter durch eine Abbildung I beschreiben mit:

$$I: \{\text{Programme in } S_Q\} \times \{\text{Eingabedaten}\} \xrightarrow{M} \{\text{Ausgabedaten}\},$$

wobei M eine Maschine bezeichnet.

1.3 Arbeitsweise eines Compilers

Die nachstehende Abbildung gibt einen groben Überblick über die Arbeitsgänge, die zur Compilierung eines Programms P_Q in ein Programm P_Z nötig sind:



Dabei bedeutet: P_Q : in der Quellsprache geschriebenes Programm
 P_Z : in die Zielsprache übersetztes Programm
 M : Maschine

Die gestrichelten Linien sollen verdeutlichen, daß die Codeerzeugung in Abhängigkeit von der Maschine M , dem Laufzeitsystem und den Eingabedaten erfolgt.

Compilierte Programme (P_Z) sind im allgemeinen nicht auf einer Maschine ablauffähig! Man benötigt die Unterstützung eines Laufzeitsystems, um Fehlerquellen wie zum Beispiel falsche bzw. fehlende Eingabedaten abzufangen, dynamische Arrays zur Laufzeit anzulegen und bei formalen Prozeduraufrufen die Sprungadresse zur Laufzeit zu berechnen.

1.4 Die Arbeitsgänge zur Compilierung eines Programms

Die Analyse des Programms P_Q wird unterteilt in die

- lexikalische Analyse
- Syntaxanalyse
- semantische Analyse.

1.4.1 Lexikalische Analyse

Eingabe : Quellprogramm in Form einer Zeichenreihe

Ausgabe: Programm als Folge von Token (= Folge von Zeichen, die bedeutungsmäßig zusammengehören; also Identifikatoren, Zahlen, Sondersymbole); Identifikatoren-
tabellen und andere Tabellen werden bereits angelegt.

Aufgaben der lexikalischen Analyse:

- a) Überprüfung, ob alle Zeichen aus dem Zeichenvorrat der Quellsprache sind.
- b) Erkennung von Grundsymbolen, Token.
- c) Entfernung von irrelevanten Zeichenfolgen (Kommentare, überflüssige Leerzeichen, Zeilenenden, Tabulatoren, ...).

Tokenklassen

T_1 : Identifikatoren

T_2 : reservierte Wörter

T_3 : logische Operatoren

T_4 : arithmetische Operatoren

T_5 : Vergleichsoperatoren

T_6 : Integer-Zahlen

T_7 : Real-Zahlen

T_8 : Sonderzeichen

T_9 : Kommentar

Die Elemente jeder Tokenklasse sind durchnummeriert.

Beispiel: $T_2 = \{ \overset{1}{\text{program}} \overset{2}{\text{begin}} \overset{3}{\text{end}} \dots \}$.

Beispiel 1

Quellprogramm P_Q : begin ... $U:=2*r*Pi$; (*Umfang*) ... end.

Programm als Folge von Grundsymbolen:

(Symbol,'begin')

...

(Bezeichner,U)

(Symbol,':=')

(Zahl,2)

(Symbol,'*')

(Bezeichner,r)

(Symbol,'*')

(Bezeichner,Pi)

(Symbol,';')

...

(Symbol,'end')

(Symbol,')

Beispiel 2

Quellprogramm P_Q : program Name; begin end.

Programm als Folge von Token:

(T_2 ,1)

(T_1 ,1)

(T_8 ,4)

(T_2 ,2)

(T_2 ,3)

(T_8 ,9)

1.4.2 Syntaxanalyse

Es wird überprüft, ob ein syntaktisch korrektes Programm vorliegt (= Überprüfung der strukturellen Korrektheit eines Programmes).

Möglichkeiten der Ausgabe sind:

- a) Zeichenreihe der Eingabe
- b) Ableitungsbaum des Programms
- c) ein zum Programm gehörender abstrakter Baum, der die für die Verarbeitung relevante Struktur des Programms beschreibt: die Knoten entsprechen Operatoren und Operanden. Sofern ein Knoten ein Element aus einer Klasse von Token repräsentiert, ist ihm eine Spezifikation zugefügt.

Beispiel

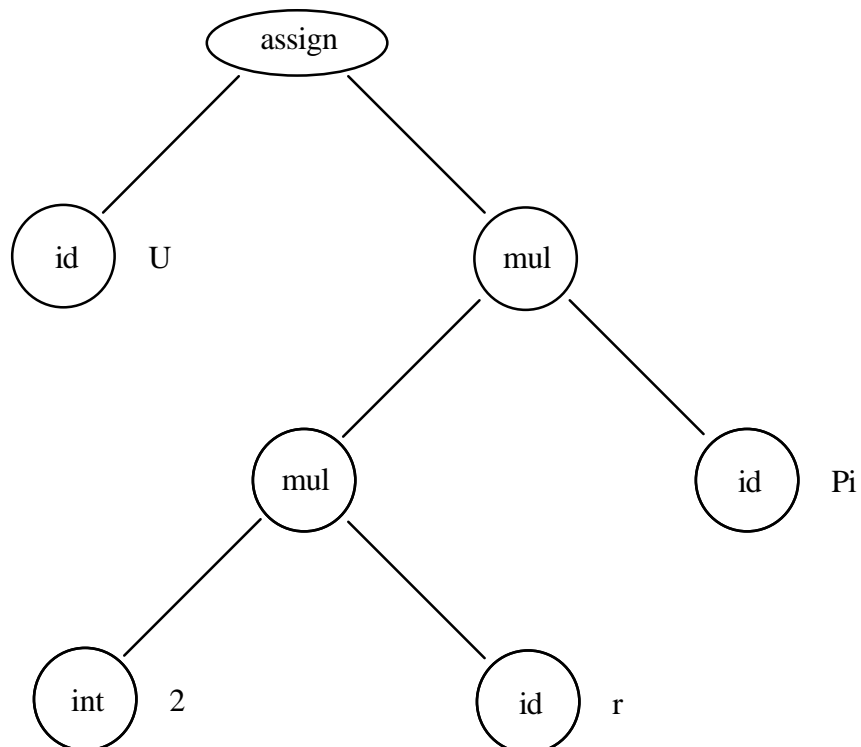
Abstrakter Baum zur Anweisung: $U := 2 * r * Pi$. Dabei bedeutet:

id : Identifikator

int : Integer-Zahl

mul : Multiplikation

assign : Zuweisung



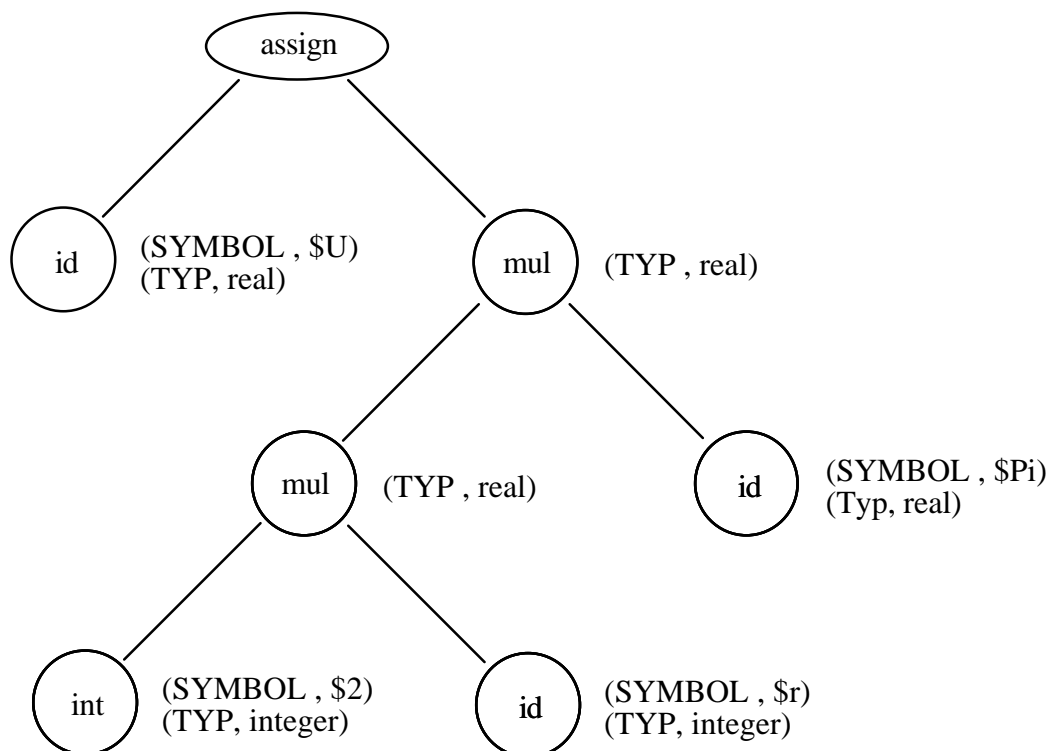
1.4.3 Semantische Analyse

Es wird überprüft, ob ein übersetzbares Programm vorliegt (= Überprüfung der statischen Semantik); es werden also die Kontextbedingungen überprüft. Hilfsmittel sind die vorher angelegten Tabellen.

Ausgabe: attributierter abstrakter Baum, sowie evtl. Informationen (Tabellen, ...) für Codegenerierung und Optimierung.

Beispiel

Attributierter abstrakter Baum zur Anweisung: $U:=2*r*Pi$:



1.4.4 Erzeugung eines Zwischencodes

Vorteil: größere Kompatibilität (d.h. leichtere Übertragung auf neue Maschinen).

Beispiel

Anweisung $U:=2*r*Pi$ in Zwischensprache:

```
[mult_int,$2,$r,$temp_1]
```

```
[conv_ir,$temp_1,$temp_2]
```

```
[mult_real,$temp_2,$Pi,$temp_3]
```

```
[assign,$temp_3,$U]
```

1.4.5 Erzeugung des Zielcodes

Ausgehend vom Zwischencode wird ein Code für die konkrete Maschine erzeugt.

Beispiel

Anweisung $U:=2*r*Pi$ im Zielcode:

```
[Load,R1 ,=2]
```

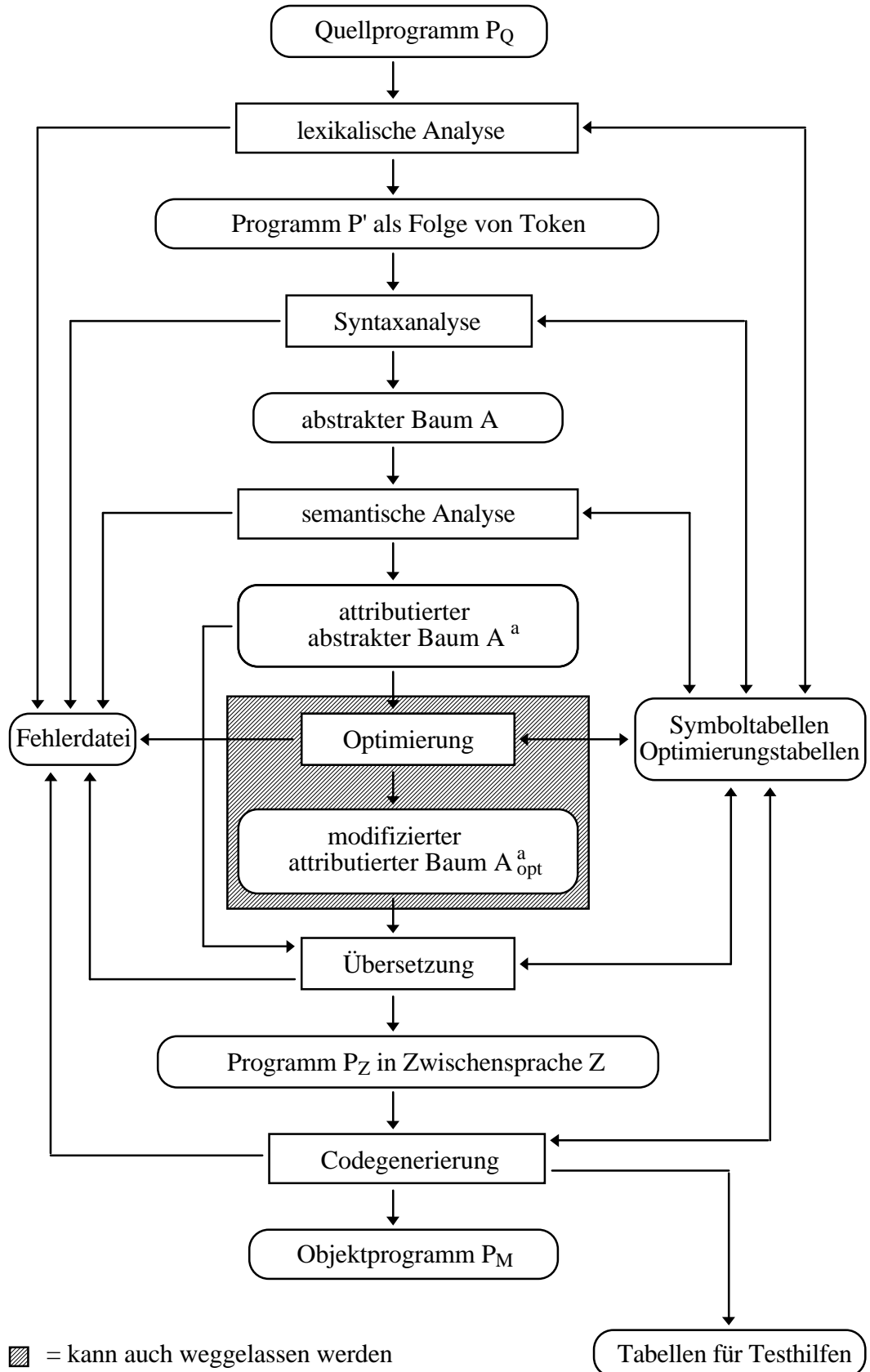
```
[Mult_Int,R1 , $\alpha_r$  ]
```

```
[Conv_IR,R1 ]
```

```
[Mult_Real,R1 , $\alpha_{Pi}$  ]
```

```
[Store,R1 , $\alpha_U$  ]
```

Schematische Darstellung der in 1.4.1 - 1.4.5 beschriebenen Phasen:



2 Grammatiken

In diesem Kapitel führen wir kontextfreie Grammatiken und die Sprachen, die sie beschreiben - die kontextfreien Sprachen - ein. Die kontextfreien Sprachen sind - wie die regulären Mengen - von großer praktischer Bedeutung, vor allem bei der Definition von Programmiersprachen, bei der Formalisierung der Syntaxanalyse, beim Vereinfachen der Übersetzung von Programmiersprachen und in anderen Prozessen, bei denen Zeichenreihen verarbeitet werden. Z.B. sind kontextfreie Grammatiken nützlich zur Beschreibung korrekt geklammerter arithmetischer Ausdrücke und der Block-Struktur in Programmiersprachen. Keiner dieser Aspekte von Programmiersprachen kann durch reguläre Ausdrücke dargestellt werden.

2.1 Kontextfreie Grammatiken

Definition 2.1 Eine *nicht eingeschränkte* Grammatik G ist ein Quadrupel $G=(N,T,P,S)$ mit:

- N : Menge der nichtterminalen Symbole, $0 < |N| < \infty$
- T : Menge der terminalen Symbole, $0 < |T| < \infty$, $N \cap T = \emptyset$
- $A := N \cup T$: Gesamtzeichenvorrat
- $P \subseteq (A^+ \times A^*)$: Produktionensystem; für $(Z, \alpha) \in P$, $Z \in A^+$, $\alpha \in A^*$, schreibt man auch $Z ::= \alpha$ (Backus-Naur-Form, BNF).
- $S \in N$: Startsymbol (Axiom).

Eine nicht eingeschränkte Grammatik wird auch als *Typ-0-Grammatik* bezeichnet.

Definition 2.2 Sei $G=(N,T,P,S)$ eine (nicht eingeschränkte) Grammatik.

- a) G heißt *kontextsensitiv* $:\Leftrightarrow$
 $0 < |P| < \infty$ und $p \in P$ ist von der Form: $uZv ::= uav$, $a, u, v \in (N \cup T)^*$, $Z \in N$.
Eine kontextsensitive Grammatik wird auch als *Typ-1-Grammatik* bezeichnet.
- b) G heißt *kontextfrei* $:\Leftrightarrow$
 $0 < |P| < \infty$ und für die Produktionen $(Z, \alpha) \in P$ gilt: $Z \in N$, $\alpha \in (N \cup T)^*$.
Eine kontextfreie Grammatik wird auch als *Typ-2-Grammatik* bezeichnet.
- c) G heißt *linkslineare Grammatik* $:\Leftrightarrow$
 $0 < |P| < \infty$ und $p \in P$ ist von der Form: $Z ::= a \vee Z ::= Ya$, $a \in T^*$, $Z, Y \in N$.
 G heißt *rechtslineare Grammatik* $:\Leftrightarrow$
 $0 < |P| < \infty$ und $p \in P$ ist von der Form: $Z ::= a \vee Z ::= aY$, $a \in T^*$, $Z, Y \in N$.
Eine links- oder rechtslineare Grammatik wird auch als *Typ-3-Grammatik* oder *reguläre Grammatik* bezeichnet.

Bemerkung:

Rechts- und linkslineare Grammatiken erzeugen dieselbe Menge von Sprachen.

Definition 2.3 Eine (nicht eingeschränkte) Grammatik $G=(N,T,P,S)$ heißt *kontextfreiartig* $:\Leftrightarrow$ für die Produktionen $(Z,\alpha)\in P$ gilt: $Z\in N$, $\alpha\in (N\cup T)^*$, wobei P nicht endlich ist.

Satz 2.1 Wenn $L=L(G)$ für eine nicht eingeschränkte Grammatik $G=(N,T,P,S)$ ist, dann ist L eine rekursiv aufzählbare Sprache.

Beweis: Hopcroft / Ullman.

Bemerkung:

Ist G eine Typ-0-Grammatik, so können wir also eine Turing-Maschine M konstruieren, mit $L(M)=L(G)$.

Satz 2.2 Ist L eine rekursiv aufzählbare Sprache, so gilt $L=L(G)$ für eine nicht eingeschränkte Grammatik G .

Beweis: Hopcroft / Ullman.

Definition 2.4 Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik.

$u\in (N\cup T)^*$ kann *unmittelbar* auf v *reduziert* werden ($u\leftarrow v$) $:\Leftrightarrow$

$u=waz$, $v=wXz$, mit $w,z,a\in (N\cup T)^*$, $(X,a)\in P$.

\leftarrow^+ , \leftarrow^* sei die transitive bzw. transitiv-reflexive Hülle von \leftarrow .

u wird auf v *reduziert* $:\Leftrightarrow u\leftarrow^* v$.

u wird in *mindestens einem Schritt* auf v *reduziert* $:\Leftrightarrow u\leftarrow^+ v$

Definition 2.5 Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik.

$u\in (N\cup T)^*$ kann *unmittelbar* aus v *abgeleitet* werden ($v\rightarrow u$) $:\Leftrightarrow$

$v=wXz$, $u=waz$, mit $w,z,a\in (N\cup T)^*$, $(X,a)\in P$.

\rightarrow^+ , \rightarrow^* sei die transitive bzw. transitiv-reflexive Hülle von \rightarrow .

u wird aus v *abgeleitet* $:\Leftrightarrow v\rightarrow^* u$.

u wird in *mindestens einem Schritt* aus v *abgeleitet* $:\Leftrightarrow v\rightarrow^+ u$.

Bemerkung:

$$v \rightarrow u \Leftrightarrow u \leftarrow v$$

Definition 2.6 Die von $G=(N,T,P,S)$ erzeugte Sprache $L(G)$ ist

$$L(G) := \{u \in T^* \mid u \leftarrow^+ S\} = \{u \in T^* \mid S \xrightarrow^+ u\}.$$

Die Elemente aus $L(G)$ heißen *Sätze* (von G), die Wörter $u \in (N \cup T)^*$ mit: $u \leftarrow^* S$ heißen *Satzformen*.

Sätze einer Programmiersprache heißen *Programme* oder *syntaktisch korrekte Programme* (*syntaktische Programme*).

Bemerkung:

Mit Hilfe eines *Kontrollwortes* lassen sich die für eine Ableitungsfolge (Reduktionsfolge) benötigten Produktionen festhalten.

Beispiel

Sei $G=(\{S\},\{0,1\},P,S)$ mit:

$$P = \left\{ \begin{array}{ll} S ::= 0 & (1) \\ S ::= 1 & (2) \\ S ::= S0 & (3) \\ S ::= S1 & (4) \end{array} \right\}$$

$$S \xrightarrow{3} S0 \xrightarrow{4} S10 \xrightarrow{4} S110 \xrightarrow{1} 0110.$$

Die Ziffernfolge 3441 heißt *Kontrollwort* von 0110: $S \xrightarrow{3441} 0110$.

Für die von G erzeugte Sprache gilt: $L(G) = \{0,1\}^+$

Definition 2.7 Eine formale Sprache heißt vom Typ 0,1,2 oder 3, wenn sie von einer Grammatik des entsprechenden Typs erzeugt werden kann. In den letzten drei Fällen sprechen wir von *kontextsensitiven, kontextfreien oder regulären Sprachen*.

Bemerkung:

Für jedes Alphabet T (mit mindestens 2 Symbolen) ist die Menge der Typ- i -Sprachen über T für $i=0,1,2$ jeweils (echte) Obermenge der Typ- $i+1$ -Sprachen (dies folgt aus der Gestalt der jeweils zugelassenen Regeln unter zusätzlicher Berücksichtigung der Tatsache, daß für jede kontextfreie Sprache eine kontextfreie Grammatik existiert, die nur Produktionen der Form $A ::= \alpha$, $\alpha \in (N \cup T)^+$ und evtl. die Produktion $S ::= \varepsilon$ enthält, wobei letztere nur auftreten kann, wenn das Startsymbol S in keiner Produktion auf der rechten Seite auftritt).

Die damit gegebene Hierarchie von Sprachen wird auch die *Chomsky-Hierarchie* genannt.

Lemma 2.1 (Pumping-Lemma für kontextfreie Sprachen) *Sei L eine kontextfreie Sprache. Dann gibt es eine von L abhängige Konstante n , so daß z sich für z aus L und $|z| \leq n$ als $z = uvwxy$ schreiben läßt mit*

- 1) $|vx| \geq 1$,
- 2) $|vwx| \leq n$ und
- 3) für alle $i \geq 0$ liegt uv^iwx^iy in L .

Beweis: Hopcroft / Ullman.

Das Pumping-Lemma für kontextfreie Sprachen kann zum Beweis benutzt werden, daß eine Vielzahl von Sprachen nicht kontextfrei sind.

Beispiel

Betrachte die Sprache $L = \{a^i b^i c^i \mid i \geq 1\}$.

Behauptung: L ist nicht kontextfrei.

Beweis (durch Kontraposition):

Annahme, L ist kontextfrei, und n ist die Konstante aus dem Pumping-Lemma. Sei $z = uvwxy$ eine Zerlegung von z , die die Bedingungen 1) - 3) des Pumping-Lemmas erfüllt. Wir müssen uns fragen, wo v und x , also die Zeichenketten, die gepumpt werden sollen, in $a^n b^n c^n$ liegen könnten: Da $|vwx| \leq n$ gilt, kann vx keine Stücke aus a und c enthalten, da das am weitesten rechts stehende a noch $n+1$ Positionen von dem am weitesten links stehende c entfernt ist. Bestehen v und x nur aus a , dann ist in $uw^i y$ (der Zeichenkette $uv^i wx^i y$ mit $i = 0$) n -mal b und n -mal c enthalten, aber weniger als n -mal a , da $|vx| \geq 1$ gilt. Also ist $uw^i y$ nicht von der Form $a^i b^i c^i$. Nach dem Pumping-Lemma liegt $uw^i y$ jedoch in L , also ein Widerspruch.

Die Fälle, in denen v und x nur aus b oder aus c bestehen, lauten ähnlich. Enthält vx sowohl a als auch b , dann enthält uw mehr c als a oder b , und es liegt wieder nicht in L . Enthält vx ebenso b wie c , so resultiert daraus ein ähnlicher Widerspruch. Insgesamt folgt, daß L keine kontextfreie Sprache ist.

Erweiterte BNF (EBNF)

Sei $G=(N,T,P,S)$, $u,v \in (N \cup T)^*$, $b \in (N \cup T)^+$, $|, \{, \} \notin (N \cup T)$. Dann entspricht:

$$\begin{aligned} z ::= a_1 | a_2 | \dots | a_n & : & z ::= a_1, z ::= a_2, \dots, z ::= a_n \\ z ::= u \{ b \} v & : & z ::= uv, z ::= ubv, z ::= ubbv, \dots, \end{aligned}$$

Bemerkung:

Im letzten Fall ist die zugehörige Grammatik kontextfreiartig.

Beispiel

Die folgende Grammatik G_0 beschreibt die Identifikatoren von Pascal:

$G_0 = (N_0, T_0, P_0, S_0)$ mit:

$T_0 = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9\}$

$N_0 = \{ \langle \text{letter} \rangle, \langle \text{digit} \rangle, \langle \text{letter or digit} \rangle, \langle \text{identifier} \rangle \}$

$P_0 = \{ \langle \text{letter} \rangle ::= A | \dots | Z | a | \dots | z$

$\langle \text{digit} \rangle ::= 0 | 1 | \dots | 9$

$\langle \text{letter or digit} \rangle ::= \langle \text{letter} \rangle | \langle \text{digit} \rangle$

$\langle \text{identifier} \rangle ::= \begin{cases} \langle \text{letter} \rangle \{ \langle \text{letter or digit} \rangle \} \\ \text{(kontextfreiartig)} \\ \langle \text{letter} \rangle | \langle \text{identifier} \rangle \langle \text{letter or digit} \rangle \\ \text{(kontextfrei)} \end{cases}$

}

$S_0 = \langle \text{identifier} \rangle$

Bemerkung:

Das Beispiel zeigt, daß eine kontextfreie Sprache auch von einer kontextfreiartigen Grammatik erzeugt werden kann.

Definition 2.8 Seien G_1, G_2 (kontextfreie) Grammatiken.
 G_1 und G_2 heißen *äquivalent* $:\Leftrightarrow L(G_1)=L(G_2)$.

Beispiel

Seien $G=(\{S\},\{0,1\},P,S)$ mit:

$$P=\{ \begin{array}{l} S::=0 \\ S::=1 \\ S::=S0 \\ S::=S1 \end{array} \}$$

(G ist linkslinear)

und $G'=(\{S\},\{0,1\},P',S)$ mit:

$$P'=\{ \begin{array}{l} S::=0 \\ S::=1 \\ S::=0S \\ S::=1S \end{array} \}$$

(G' ist rechtslinear).

G und G' sind äquivalent.

2.2 Eigenschaften von (kontextfreien) Grammatiken

Sei $G=(N,T,P,S)$ eine Grammatik. Dann heißt G

- 1) **ϵ -frei**, falls entweder keine ϵ -Produktion in P vorkommt (d.h. $\neg(\exists A::=\epsilon: (A,\epsilon)\in P)$) oder $S::=\epsilon$ die einzige ϵ -Produktion in P ist, und S in keiner Produktion auf der rechten Seite auftritt.
- 2) **zyklenfrei**, falls N kein A enthält mit: $A \xrightarrow{+} A$.
- 3) **reduziert**, falls G keine nutzlosen Nichtterminalsymbole enthält. Ein Nichtterminal $A\in N$ heißt *nützlich*, falls es eine Ableitung $S \xrightarrow{*} \alpha A \beta \xrightarrow{*} w$ für $\alpha,\beta\in (N\cup T)^*$, $w\in T^*$. Eine Produktion $A::=B$, $A,B\in N$, heißt Kettenproduktion.

Bemerkung:

- 1) Bis auf Anwendung von $A ::= \varepsilon$ wird eine Satzform durch Anwendung einer Produktion nie kürzer (beim Reduzieren nie länger).
- 2) Sei $G'=(N',T',P',S)$ eine beliebige kontextfreie Grammatik mit $\varepsilon \notin L(G')$. Dann gibt es eine zu G' äquivalente kontextfreie Grammatik G mit:
 - a) G ist ε -frei
 - b) G enthält keine Kettenproduktion
 - c) G ist zyklonfrei
 - d) G ist reduziert ($L(G) \neq \emptyset$).

Beweis: Hopcroft / Ullman.

Definition 2.9 Eine Grammatik $G=(N,T,P,S)$ heißt in *Chomsky-Normalform*, wenn in P nur Produktionen folgender Form auftreten:

- 1) $A ::= BC, \quad A,B,C \in N$
- 2) $A ::= a, \quad a \in T, A \in N.$

Sofern das Startsymbol S in keiner Produktion auf der rechten Seite auftritt, ist auch die Produktion $S ::= \varepsilon$ erlaubt.

Satz 2.3 Jede kontextfreie Sprache $L-\{\varepsilon\}$ wird von einer Grammatik G in Chomsky-Normalform erzeugt.

Beweis: Hopcroft / Ullman.

Definition 2.10 Eine Grammatik $G=(N,T,P,S)$ heißt in *Greibach-Normalform*, wenn in P nur Produktionen folgender Form auftreten:

$$A ::= a\alpha, \quad a \in T, \quad \alpha \in N^*$$

Sofern das Startsymbol S in keiner Produktion auf der rechten Seite auftritt, ist auch die Produktion $S ::= \varepsilon$ erlaubt.

Satz 2.4 Jede kontextfreie Sprache $L-\{\epsilon\}$ kann durch eine Grammatik G in Greibach-Normalform generiert werden.

Beweis: Hopcroft / Ullman.

2.3 Eindeutigkeit von kontextfreien Grammatiken

Im folgenden sei $G=(N,T,P,S)$ eine kontextfreie Grammatik.

Definition 2.11 Ein *Reduktionsschritt* von u nach v ist ein Tripel $r=(w,Nt::=\alpha,z)$, $w,z \in (N \cup T)^*$, $Nt::=\alpha \in P$, und es gilt:

$$\begin{array}{c} u \equiv w \alpha z \\ \quad \uparrow \\ v \equiv w Nt z \end{array}$$

Beispiel

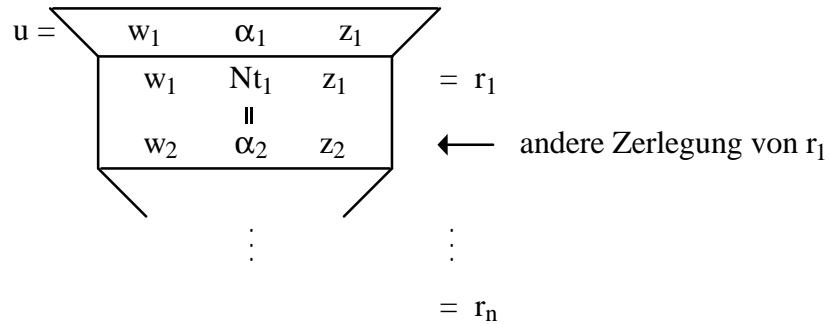
Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik. Betrachte die Produktion: $A::=AB \mid BA \in P$. Die Satzform $u:=ABBA$ lässt sich dann wie folgt reduzieren:

$$\begin{array}{c} u = A B B A \\ \quad \vee \quad \vee \\ v = A B A \\ \quad \vee \quad \vee \\ w = A A \end{array}$$

Die beiden Reduktionsschritte von u nach v lauten also: $(\epsilon,A::=AB,BA)$ und $(AB,A::=BA,\epsilon)$.

Definition 2.12 Eine *Reduktionsfolge* von u nach v ist eine endliche Folge $R=(r_1,\dots,r_n)$ mit $r_i=(w_i,Nt_i::=\alpha_i,z_i)$, $w_i,z_i \in (N \cup T)^*$, $Nt_i::=\alpha_i \in P$, und $w_i Nt_i z_i \equiv w_{i+1} \alpha_{i+1} z_{i+1}$.

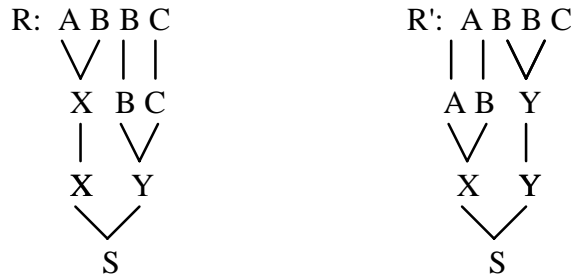
Beispiel



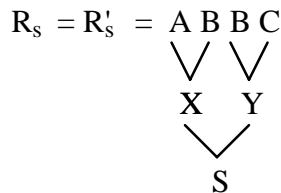
Definition 2.13 Der *Strukturbaum* einer Reduktionsfolge R ist die graphische Darstellung von R, wobei identische Übergänge weggelassen werden.

Beispiel

Betrachte die Produktionen: $S ::= XY \mid AY$, $Y ::= BY \mid BC$, $X ::= AB$. Sei $u ::= ABBC$. Dann läßt sich u beispielsweise folgendermaßen reduzieren:



R und R' stellen verschiedene Reduktionsfolgen dar, besitzen aber denselben Strukturbaum:



Reduktionsklassen:

$$r_{\alpha::=\tau} = \{\tau+, \tau \text{---}\}$$

$$r_{\alpha:=\alpha+\tau} = \{\alpha+\tau+, \alpha+\tau \text{---}\}$$

$$r_{\tau::=V} = \{V+, V^*, V \text{---}, \alpha+V+, \alpha+V^*, \alpha+V \text{---}\}$$

$$r_{\tau::=\tau^*V} = \{\tau^*V^*, \tau^*V+, \tau^*V \text{---}, \alpha+\tau^*V+, \alpha+\tau^*V^*, \alpha+\tau^*V \text{---}\}$$

$$r_{\text{Weiterlesen}} = \{\alpha+, \alpha+V, \tau^*, \tau^*V, \alpha+\tau^*, \alpha+\tau^*V, V, \varepsilon\}$$

$$r_{\text{Stop}} = \{\alpha \text{---}\},$$

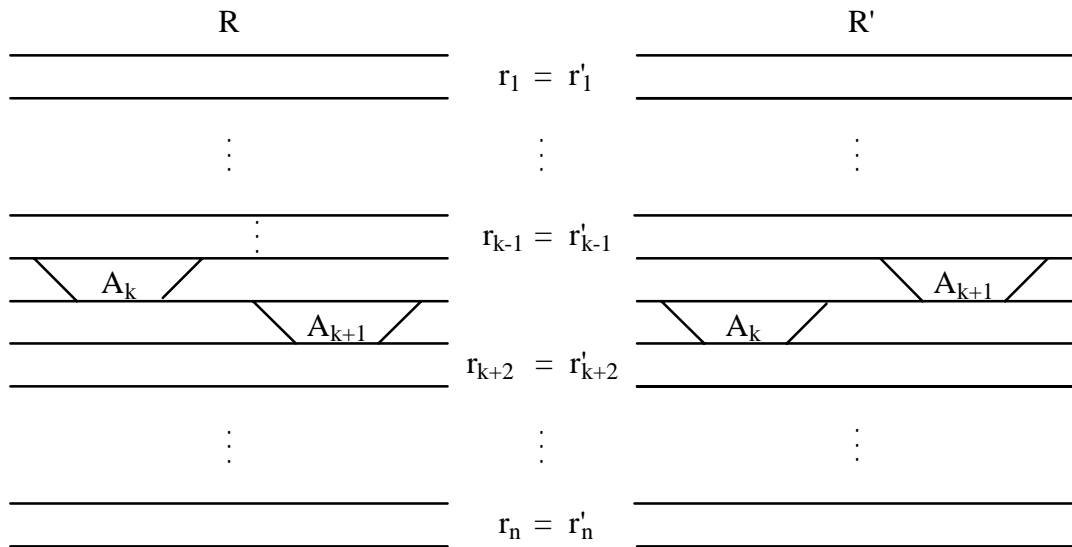
wobei "---" ein Abschlußsymbol ist.

Die Reduktionsklassen sind disjunkt !

Wir wollen nun noch einmal $w=V+V^*V$ anhand dieser Reduktionsklassen auf das Startsymbol α reduzieren:

$$\begin{array}{l} V + V * V \text{---} \\ \boxed{V} + V * V \text{---} \\ \boxed{V +} V * V \text{---} \\ \boxed{\tau +} V * V \text{---} \\ \boxed{\alpha +} V * V \text{---} \\ \boxed{\alpha + V} * V \text{---} \\ \boxed{\alpha + V * } V \text{---} \\ \boxed{\alpha + \tau * } V \text{---} \\ \boxed{\alpha + \tau * V} \text{---} \\ \boxed{\alpha + \tau * V \text{---}} \\ \boxed{\alpha + \tau} \text{---} \\ \boxed{\alpha} \text{---} \end{array}$$

Definition 2.15 Gegeben seien zwei Reduktionsfolgen $R=(r_1,\dots,r_n)$, $R'=(r'_1,\dots,r'_n)$ von u nach v . R heißt *unmittelbar kanonischer* als R' ($R < R'$), wenn R und R' bis auf den k -ten und $(k+1)$ -ten Reduktionsschritt identisch sind. Dabei muß folgende Bedingung erfüllt sein:



R' geht also aus R dadurch hervor, daß man die k -te Produktion ($A_k ::= b_k$) in R nach links unten und die $(k+1)$ -te Produktion ($A_{k+1} ::= b_{k+1}$) nach rechts oben schiebt, sofern dieses überhaupt möglich ist.

Bemerkung:

1. Die linke Reduktionsfolge wird kanonisch ausgewählt. *kanonisch* heißt: Analyse von links nach rechts (LR-Analyse).
2. Bei einer kanonischen Reduktionsfolge stehen rechts vom zu analysierenden Symbol nur terminale Zeichen.

Wir bilden nun zu $<$ wieder die transitive Hülle und erhalten $<<$. Diese erweitern wir zu $<>$, der äquivalenten Hülle. Wir führen also eine Äquivalenzrelation ein, um unsere Reduktionsfolgen in Klassen einteilen zu können.

Definition 2.16 Zwei Reduktionsfolgen R und R' heißen unwesentlich verschieden ($R <> R'$), wenn es eine Serie von Reduktionsfolgen $R=R_1, R_2, R_3, \dots, R_n=R'$ gibt, mit: $R_{i+1} < R_i$ oder $R_i < R_{i+1}$.

Satz 2.5 *Unwesentlich verschiedene Reduktionsfolgen besitzen denselben Strukturbaum.*

Satz 2.6 *Zu jeder Reduktionsfolge R gibt es eine unwesentlich verschiedene kanonische Reduktionsfolge R' .*

Satz 2.7 *$u \in L(G)$ ist genau dann eindeutig, wenn es genau eine kanonische Reduktionsfolge besitzt.*

Definition 2.17 $\alpha, \beta \in (N \cup T)^*$, $\alpha \rightarrow \beta$ mit $\alpha = \alpha_1 A \alpha_3$, $\beta = \alpha_1 \alpha_2 \alpha_3$, $A ::= \alpha_2 \in P$.

- a) $\alpha \rightarrow \beta$ heißt *Linksableitungsschritt* $(\alpha \rightarrow_L \beta) :\Leftrightarrow \alpha_1 \in T^*$.
- b) $\alpha \rightarrow \beta$ heißt *Rechtsableitungsschritt* $(\alpha \rightarrow_R \beta) :\Leftrightarrow \alpha_3 \in T^*$.
- c) entsprechend definiere *Linksableitung* und *Rechtsableitung*.

Eine *kanonische Reduktionsfolge* ist eine Rechtsableitung in umgekehrter Reihenfolge.

3 Lexikalische Analyse

Die von endlichen Automaten akzeptierten Sprachen lassen sich durch einfache Ausdrücke beschreiben, die als reguläre Ausdrücke bezeichnet werden. In diesem Abschnitt definieren wir reguläre Ausdrücke, führen die Operationen Vereinigung, Konkatenation und Hüllenbildung auf die von einem regulären Ausdruck bezeichneten Menge ein und beweisen, daß die Sprachklasse, die durch endliche Automaten akzeptiert wird, genau die Sprachklasse ist, die durch reguläre Ausdrücke beschreibbar ist.

Reguläre Ausdrücke haben sich als nützliche Werkzeuge zur Entwicklung von lexikalischen Analysen erwiesen, denn die Token einer Programmiersprache sind fast ausnahmslos als reguläre Mengen darstellbar.

Einige Generatoren lexikalischer Analyser verarbeiten eine Eingabefolge von regulären Ausdrücken zur Beschreibung der Token und produzieren einen einzigen endlichen Automaten zur Erkennung dieser Token. Normalerweise wandeln sie dazu den regulären Ausdruck in einen nichtdeterministischen endlichen Automaten mit ϵ -Übergängen um und konstruieren dann Teilmengen von Zuständen zwecks Erzeugung eines deterministischen endlichen Automaten, anstatt zuerst die ϵ -Übergängen zu entfernen. Jeder Endzustand zeigt das bestimmte gefundene Token an, so daß der Automat eigentlich ein endlicher Automat mit Ausgabe ist. Die Übergangsfunktion des endlichen Automaten wird gemäß einer aus verschiedenen möglichen Methoden kodiert, um weniger Platz zu benötigen als eine Übergangstabelle in Form eines zweidimensionalen Arrays. Der vom Generator erzeugte lexikalische Analyser ist ein festes Programm, das kodierte Tabellen zusammen mit einer den endlichen Automaten zur Token-Erkennung spezifizierenden Tabelle interpretiert; die Token wurden dabei für den Generator lediglich als reguläre Ausdrücke spezifiziert.

3.1 Reguläre Ausdrücke

Definition 3.1 Sei Σ ein beliebiges Alphabet.

1) Die Menge der *regulären Ausdrücke* über Σ , R_Σ , ist gegeben durch:

- i) $\emptyset, \varepsilon, a \in \Sigma$ sind in R_Σ
- ii) $\alpha, \beta \in R_\Sigma \Rightarrow (\alpha + \beta), (\alpha \circ \beta)$ und $(\alpha)^*$ sind Elemente von R_Σ .
- iii) Durch (i) und (ii) ist R_Σ vollständig festgelegt.

2) Sei $\zeta: R_\Sigma \xrightarrow{\text{total}} \wp(\Sigma^*)$; $\wp(\Sigma^*) =$ Potenzmenge von Σ^* .

$\zeta(\alpha)$ heißt *die durch α bezeichnete reguläre Menge über Σ* mit:

- i) $\zeta(\emptyset) = \emptyset$,
 $\zeta(\varepsilon) = \{\varepsilon\}$,
 $\forall a \in \Sigma: \zeta(a) = \{a\}$
- ii) Seien $\alpha, \beta \in R_\Sigma$ beliebig. Dann:
 $\zeta((\alpha + \beta)) = \zeta(\alpha) \cup \zeta(\beta)$
 $\zeta((\alpha \circ \beta)) = \zeta(\alpha) \circ \zeta(\beta)$
(für die Konkatenation zweier Mengen gilt: $X \circ Y := \{xy \mid x \in X, y \in Y\}$)
 $\zeta((\alpha)^*) = (\zeta(\alpha))^*$
- iii) Durch (i) und (ii) werden alle regulären Mengen über Σ erfasst.

Bemerkung

- 1) Das "+" bezeichnet die Oder-Verknüpfung und "o" die Konkatenation zweier regulärer Ausdrücke. Es ist möglich, anstelle von $\alpha \circ \beta$ auch $\alpha\beta$ zu schreiben. Bei der Auswertung gilt die Reihenfolge: "o" vor "+".
- 2) ζ ist nicht injektiv.
Beispiel: $\zeta((0+1)^*) = \zeta((1+0)^*)$.

Beispiel

Reguläre Ausdrücke über $\Sigma = \{0, 1\}$:

- 1) $0 \in R_\Sigma$: $\zeta(0) = \{0\}$
- 2) $(0+1)^* \in R_\Sigma$: $\zeta((0+1)^*) = (\zeta(0+1))^* = (\zeta(0) \cup \zeta(1))^* = \{0, 1\}^*$
- 3) $0(0+1)^*1 \in R_\Sigma$: $\zeta(0(0+1)^*1) = \{0x1 \mid x \in \{0, 1\}^*\}$
- 4) $\emptyset^* \in R_\Sigma$: $\zeta(\emptyset^*) = (\zeta(\emptyset))^* = \emptyset^* = \{\varepsilon\}$

Definition 3.2 Seien $\alpha, \beta \in R_\Sigma$. α und β heißen *äquivalent* ($\alpha \equiv \beta$) $:\Leftrightarrow \zeta(\alpha) = \zeta(\beta)$.

3.2 Automaten

3.2.1 (Nichtdeterministische) Endliche Automaten

Definition 3.3 $EA=(Q,\Sigma,\delta,q_0,F)$ heißt *endlicher Automat*, wobei:

Q : Menge der Zustände (endlich)

Σ : Eingabealphabet (endlich)

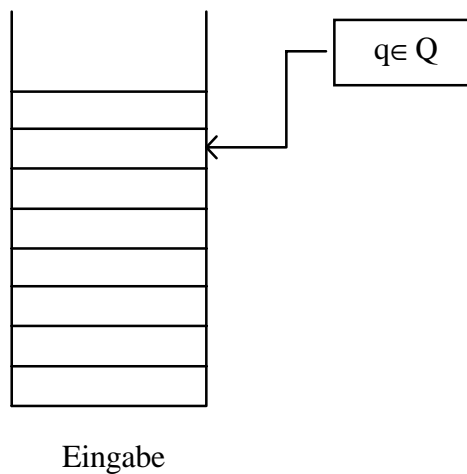
$\delta:Q\times\Sigma \rightarrow \wp(Q)$: Übergangsfunktion

Die Übergangsfunktion δ lässt sich auch beschreiben durch eine Menge der Form: $\{((q,a),p) \mid q,p \in Q, a \in \Sigma\}$.

$q_0 \in Q$: Startzustand

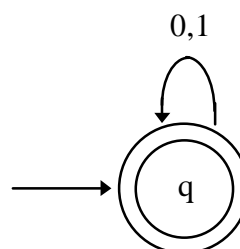
$F \subseteq Q$: Menge der Endzustände

Allgemeine Darstellung eines endlichen Automaten:



Beispiel

- 1) $A_1 = (\{q\}, \{0,1\}, \delta_1, q, \{q\})$ mit:
 $\delta_1(q,0) = \delta_1(q,1) = \{q\}$

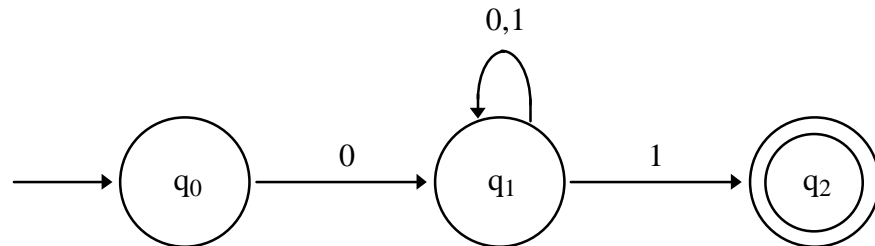


2) $A_2 = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta_2, q_0, \{q_2\})$ mit:

$$\delta_2(q_0, 0) = \delta_2(q_1, 0) = \{q_1\},$$

$$\delta_2(q_1, 1) = \{q_1, q_2\},$$

$$\delta_2(q, a) = \emptyset \text{ sonst.}$$



Die Übergangsfunktion δ kann erweitert werden zu einer Funktion $\delta: Q \times \Sigma^* \rightarrow \wp(Q)$ durch:

$$1) \quad \forall q \in Q: \delta(q, \varepsilon) = \{q\}$$

$$2) \quad \forall x \in \Sigma^*, a \in \Sigma: \delta(q, xa) = \bigcup_{q' \in \delta(q, x)} \delta(q', a).$$

Dann kann man folgende Definition formulieren:

Definition 3.4 Ein Wort $w \in \Sigma^*$ wird von einem endlichen Automaten EA *akzeptiert*

$$:\Leftrightarrow \delta(q_0, w) \cap F \neq \emptyset.$$

$L(\text{EA}) := \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$ ist die vom endlichen Automaten EA *akzeptierte Sprache*.

Satz 3.1 Sei Σ ein Alphabet und $\gamma \in R_\Sigma$ beliebig. Dann existiert ein endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$, der genau die durch γ bezeichnete reguläre Menge akzeptiert.

Beweis:

Beweis durch Induktion über die Anzahl der Operatoren im regulären Ausdruck.

Induktionsanfang (null Operatoren):

$$a) \quad \gamma = \emptyset:$$

$$\text{Sei } F = \emptyset.$$

$$\Rightarrow L(A) = \zeta(\emptyset) = \emptyset.$$

b) $\gamma = \varepsilon$:

Sei $A = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$ mit: $\forall a \in \Sigma: \delta(q_0, a) = \emptyset$.

$\Rightarrow L(A) = \zeta(\varepsilon) = \{\varepsilon\}$.

c) $\gamma = a \in \Sigma$:

Sei $A = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ mit:

$\delta(q_0, a) = \{q_1\}, \forall (q', a') \neq (q_0, a): \delta(q', a') = \emptyset$.

$\Rightarrow L(A) = \zeta(a) = \{a\}$.

Induktionsschritt (ein oder mehrere Operatoren):

a) $\gamma = (\alpha + \beta)$ (Vereinigung):

$\exists A_1, A_2: A_i = (Q_i, \Sigma, \delta_i, q_{0_i}, F_i), i=1,2, Q_1 \cap Q_2 = \emptyset$, die die durch α und β bezeichneten regulären Mengen akzeptieren.

Sei $A := (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma, \delta, q_0, F)$ mit:

$$\delta := \delta_1 \cup \delta_2 \cup \{((q_0, a), \delta_1(q_{0_1}, a) \cup \delta_2(q_{0_2}, a)) \mid a \in \Sigma\},$$

$$F := \begin{cases} F_1 \cup F_2 & , \text{ falls } q_{0_1} \notin F_1 \wedge q_{0_2} \notin F_2 \\ F_1 \cup F_2 \cup \{q_0\} & , \text{ sonst} \end{cases}.$$

Dann gilt: $L(A) = \zeta(\alpha) \cup \zeta(\beta) = \zeta((\alpha + \beta)) = \zeta(\gamma)$.

b) $\gamma = (\alpha \circ \beta)$ (Konkatenation):

$\exists A_1, A_2: A_i = (Q_i, \Sigma, \delta_i, q_{0_i}, F_i), i=1,2, Q_1 \cap Q_2 = \emptyset$, die die durch α und β bezeichneten regulären Mengen akzeptieren.

Sei $A := (Q_1 \cup Q_2, \Sigma, \delta, q_{0_1}, F)$ mit:

$$\delta(q, a) := \begin{cases} \delta_1(q, a) & , \forall q \in Q_1 - F_1 \\ \delta_1(q, a) \cup \delta_2(q_{0_2}, a) & , \forall q \in F_1 \\ \delta_2(q, a) & , \forall q \in Q_2 \end{cases},$$

$$F := \begin{cases} F_2 & , \text{ falls } q_{0_2} \notin F_2 \\ F_1 \cup F_2 & , \text{ sonst} \end{cases}.$$

Dann gilt: $L(A) = \zeta(\alpha) \circ \zeta(\beta) = \zeta((\alpha \circ \beta)) = \zeta(\gamma)$.

c) $\gamma = (\alpha)^*$:

$\exists A_1 : A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, der die durch α bezeichnete reguläre Menge akzeptiert.

Sei $A := (Q_1 \cup \{q_0\}, \Sigma, \delta, q_0, F_1 \cup \{q_0\})$ mit:

$$\delta(q, a) := \begin{cases} \delta_1(q_{01}, a) & , q = q_0 \\ \delta_1(q, a) & , q \in Q_1 - F_1 \\ \delta_1(q, a) \cup \delta_1(q_{01}, a) & , q \in F_1 \end{cases}$$

Dann gilt: $L(A) = (\zeta(\alpha))^* = \zeta((\alpha)^*) = \zeta(\gamma)$.

q.e.d

Bemerkung:

Die Umkehrung des Satzes gilt auch.

Beweis: Manna, Z., *Mathematical Theory of Computation*, McGraw Hill (1974), S. 11 ff.

Somit haben wir folgenden zusammenfassenden

Satz 3.2 Eine Sprache wird genau dann von einem endlichen Automaten EA akzeptiert, wenn sie eine reguläre Menge ist.

3.2.2 Deterministische endliche Automaten

Definition 3.5 A heißt *deterministischer endlicher Automat* (DEA)

$:\Leftrightarrow \forall q \in Q, a \in \Sigma: |\delta(q, a)| \leq 1$.

A heißt *vollständig* $:\Leftrightarrow \forall q \in Q, a \in \Sigma: |\delta(q, a)| \geq 1$.

Bemerkung:

Zu jedem DEA $A = (Q, \Sigma, \delta_A, q_0, F)$ läßt sich ein äquivalenter vollständiger DEA $A' = (Q', \Sigma, \delta_{A'}, q_0, F)$ angeben, mit:

$Q' := Q \cup \{\bar{q}\}$,

$\delta_{A'} := \delta_A \cup \{((q, a), \bar{q}) \mid (q, a) \in Q \times \Sigma \wedge \delta_A(q, a) = \emptyset\} \cup \{((\bar{q}, a), \bar{q}) \mid a \in \Sigma\}$.

\uparrow_1

\uparrow_2

\uparrow_1 : für alle Eingaben a mit $\delta_A(q, a) = \emptyset$ gilt im neuen Automaten A': $\delta_{A'}(q, a) = \bar{q}$

\uparrow_2 : Simulation einer Endlosschleife

Satz 3.3 Zu jedem endlichen Automaten A gibt es einen äquivalenten DEA A' .

Beweis:

Geben sei $A=(Q,\Sigma,\delta,q_0,F)$.

Konstruiere $A'=(Q',\Sigma,\delta',q'_0,F')$:

1. Die Zustände in A' haben die Form: $[q_{j_0},\dots,q_{j_m}]$, $m \geq -1$, $q_{j_i} \in Q$, $i=0,\dots,m$, wobei $[q_{j_0},\dots,q_{j_{-1}}] = []$
2. $Q' := \{ [q_{j_0},\dots,q_{j_m}] \mid \{q_{j_0},\dots,q_{j_m}\} \in \wp(Q) \}$.
3. $q'_0 := [q_0]$.
4. $F' := \{ [q_{j_0},\dots,q_{j_m}] \in Q' \mid F \cap \{q_{j_0},\dots,q_{j_m}\} \neq \emptyset \}$.
5. Seien $q' := [q_{j_0},\dots,q_{j_m}] \in Q'$ und $a \in \Sigma$ beliebig.

$$\text{Dann setze: } \delta'(q',a) = r' \in Q' \text{ mit } r' = [r_{t_0},\dots,r_{t_k}] \quad \Leftrightarrow \quad \bigcup_{i=j_0}^{j_m} \delta(q_i,a) = \{r_{t_0},\dots,r_{t_k}\}.$$

Behauptung: $\forall w \in \Sigma^*: \delta'(q'_0,w) = [q_{j_0},\dots,q_{j_m}] \Leftrightarrow \delta(q_0,w) = \{q_{j_0},\dots,q_{j_m}\}$.

Beweis durch vollständige Induktion über die Länge der Worte $w \in \Sigma^*$:

I.A. $|w|=0$, d.h. $w=\varepsilon$:

$$\text{Nach Definition gilt: } \delta'(q'_0,\varepsilon) = [q_0] \Leftrightarrow \bigcup_{i=0}^0 \delta(q_i,a) = \delta(q_0,\varepsilon) = \{q_0\}.$$

I.V. Die Behauptung gilt für alle $w \in \Sigma^*$ mit $|w| \leq n$.

I.S. Sei $w \in \Sigma^*$ beliebig mit $|w|=n$ und $a \in \Sigma$. Dann gilt nach der Induktionsvoraussetzung:

$$\delta'(q'_0,w) = [q_{j_0},\dots,q_{j_k}] \Leftrightarrow \delta(q_0,w) = \{q_{j_0},\dots,q_{j_k}\}.$$

Weiter gilt:

$$\delta'(q'_0,wa) = \bigcup_{q' \in \delta(q'_0,w)} \delta'(q',a) = \delta'([q_{j_0},\dots,q_{j_k}],a) = [q_{t_0},\dots,q_{t_n}].$$

Nach Definition erhalten wir daraus:

$$\delta'([q_{j_0},\dots,q_{j_k}],a) = [q_{t_0},\dots,q_{t_n}] \quad \Leftrightarrow \quad \bigcup_{i=j_0}^{j_k} \delta(q_i,a) = \{q_{t_0},\dots,q_{t_n}\}.$$

Insgesamt folgt:

$$\delta'(q'_0,wa) = [q_{t_0},\dots,q_{t_n}] \Leftrightarrow \delta(q_0,wa) = \{q_{t_0},\dots,q_{t_n}\}.$$

q.e.d.

Beispiel

Sei $A=(\{q_0, q_1, q_2\}, \{0,1\}, \delta_2, q_0, \{q_2\})$ mit:

$$\delta_2(q_0, 0) = \delta_2(q_1, 0) = \{q_1\}$$

$$\delta_2(q_1, 1) = \{q_1, q_2\}$$

$$\delta_2(q, a) = \emptyset \text{ sonst.}$$

(siehe auch Seite 32, Beispiel 2)

Der äquivalente DEA A' lautet: $A'=(Q', \{0,1\}, \delta', q'_0, F')$ mit:

$$Q' := \{[\], [q_0], [q_1], [q_2], [q_0, q_1], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$$

$$q'_0 := [q_0]$$

$$F' := \{[q_2], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$$

$$\delta'([\], a) = [\] \quad \forall a \in \{0,1\}$$

$$\delta'([q_0], 0) = [q_1]$$

$$\delta'([q_0], 1) = [\]$$

$$\delta'([q_1], 0) = [q_1]$$

$$\delta'([q_1], 1) = [q_1, q_2]$$

$$\delta'([q_2], a) = [\] \quad \forall a \in \{0,1\}$$

$$\delta'([q_0, q_1], 0) = [q_1]$$

$$\delta'([q_0, q_1], 1) = [q_1, q_2]$$

$$\delta'([q_0, q_2], 0) = [q_1]$$

$$\delta'([q_0, q_2], 1) = [\]$$

$$\delta'([q_1, q_2], 0) = [q_1]$$

$$\delta'([q_1, q_2], 1) = [q_1, q_2]$$

$$\delta'([q_0, q_1, q_2], 0) = [q_1]$$

$$\delta'([q_0, q_1, q_2], 1) = [q_1, q_2]$$

3.2.3 Minimierung der Zustandsmenge eines endlichen Automaten

Satz 3.4 Sei $A=(Q, \Sigma, \delta, q_0, F)$ ein vollständiger DEA. Dann existiert ein äquivalenter vollständiger DEA $A'=(Q', \Sigma, \delta', q'_0, F')$ derart, daß es keinen zu A äquivalenten Automaten gibt, der weniger Zustände hat als A' .

Beweis:

Sei die Relation R in Σ^* wie folgt definiert:

$$\forall x, y \in \Sigma^*: xRy \Leftrightarrow (\forall z \in \Sigma^*: xz \in L(A) \Leftrightarrow yz \in L(A)).$$

R ist eine Äquivalenzrelation.

Sei $Q' := \{[x] \mid x \in \Sigma^*\}$ die Menge der durch R in Σ^* erzeugten Äquivalenzklassen.

Seien $x, y \in \Sigma^*$ mit: $\delta(q_0, x) = \delta(q_0, y) = q$. Dann folgt nach Definition von R : xRy , und somit $|Q'| \leq |Q|$.

Seien $q'_0 := [\epsilon]$, $F' := \{[x] \mid x \in L(A)\}$ und für alle $[x] \in Q'$, $a \in \Sigma$: $\delta'([x], a) := [xa]$. Da aus xRy immer folgt: $xaRya$, ist diese Definition konsistent, d.h. unabhängig von der Wahl des Repräsentanten. Dann gilt für beliebiges $x \in \Sigma^*$:

$$\delta'(q'_0, x) = [x] \text{ und somit: } x \in L(A') \Leftrightarrow [x] \in F' \Leftrightarrow x \in L(A).$$

Also: $L(A') = L(A)$.

q.e.d.

Hilfssatz 3.1 Seien zwei Automaten A und A' wie im obigen Satz gegeben.

Sei $T: Q \xrightarrow{\text{total}} Q'$, $T(q) = [x] \Leftrightarrow \delta(q_0, x) = q$.

Dann gilt:

$$\forall q_1, q_2 \in Q: T(q_1) \neq T(q_2) \Leftrightarrow$$

$$\exists z \in \Sigma^*: (\delta(q_1, z) \in F \wedge \delta(q_2, z) \in Q-F) \vee (\delta(q_1, z) \in Q-F \wedge \delta(q_2, z) \in F).$$

Wir sagen: "**z trennt q_1 und q_2** ".

Beweis:

Seien $q_1, q_2 \in Q$ beliebig.

$$\text{Dann: } \exists x_1, x_2 \in \Sigma^*: q_1 = \delta(q_0, x_1) \wedge q_2 = \delta(q_0, x_2)$$

$$T(q_1) \neq T(q_2) \Leftrightarrow [x_1] \neq [x_2] \Leftrightarrow \neg(x_1 R x_2)$$

$$\Leftrightarrow \exists z \in \Sigma^*: (x_1 z \in L(A) \wedge x_2 z \notin L(A)) \vee (x_1 z \notin L(A) \wedge x_2 z \in L(A))$$

$$\Leftrightarrow \exists z \in \Sigma^*: (\delta(q_1, z) \in F \wedge \delta(q_2, z) \in Q-F) \vee (\delta(q_1, z) \in Q-F \wedge \delta(q_2, z) \in F)$$

q.e.d.

Algorithmus

zur Minimierung der Zustandsmenge eines endlichen Automaten.

Eingabe: Ein vollständiger DEA $A=(Q,\Sigma,\delta,q_0,F)$.

Alle Zustände in Q seien erreichbar.

Ausgabe: Der minimale DEA $A'=(Q',\Sigma,\delta',q'_0,F')$ mit $L(A')=L(A)$.

A' braucht nicht vollständig zu sein.

Methode:

- 1) Falls $F=\emptyset$, sei $A':=({q'_0},\Sigma,\emptyset,q'_0,\emptyset)$. In diesem Fall ist $L(A)=L(A')=\emptyset$.
- 2) Falls $Q-F=\emptyset$, sei $A':=({q'_0},\Sigma,\delta',q'_0,{q'_0})$ mit $\delta'(q'_0,a)=q'_0$ für alle $a\in\Sigma$. In diesem Fall ist $L(A)=L(A')=\Sigma^*$.

- 3) Im folgenden gelte $F \neq \emptyset$ und $Q - F \neq \emptyset$. Es wird eine Hilfsvariable PART benutzt, deren Werte Partitionen der Menge Q sind:

begin

PART := {F, Q-F};

** Initialisierung; für beliebiges $q_1 \in F$, $q_2 \in Q - F$ gilt: $\varepsilon \in \Sigma^*$ trennt q_1 und q_2 .

**

repeat

** Sei $P = \{Q_0, \dots, Q_m\}$ der aktuelle Wert von PART.

Es gilt: 1. $m \geq 1$

2. P ist eine Partition (= disjunkte Aufteilung in nicht leere Mengen) von Q, d.h.

a. $\forall i \in [0, m]: Q_i \neq \emptyset$

b. $\forall i, j \in [0, m], i \neq j: Q_i \cap Q_j = \emptyset$

c. $Q = \bigcup_{i=0}^m Q_i$.

Es wird nun eine Verfeinerung P' von P bestimmt, derart daß $P' = \{Q'_{ij} \mid i \in [0, m], j \in [0, k_i] \text{ für ein } k_i \geq 0\}$, wobei für jedes $i \in [0, m]$ die Menge $P_i := \{Q'_{i0}, \dots, Q'_{ik_i}\}$ eine Partition von Q_i ist.

Für Q_i wird genau dann eine nichttriviale Partition P_i mit $k_i \geq 1$ erzeugt, wenn es $q, q' \in Q_i$ und ein $a \in \Sigma$ gibt, so daß:

$$\delta(q, a) \in Q_r \wedge \delta(q', a) \in Q_s \text{ mit } r \neq s \text{ und } Q_r, Q_s \in P.$$

Die Schleife terminiert, sobald $P = P'$, d.h. δ bildet jeden Zustand jeder Menge Q_i bei gleicher Eingabe in dasselbe Q_j ab.

**

PART_{alt} := PART;

for i:=0 to m do

Bestimme die Partition $P_i := \{Q'_{i0}, \dots, Q'_{ik_i}\}$ von Q_i derart daß:

$(\forall v \in [0, k_i]: \forall q, q' \in Q'_{iv}: \forall a \in \Sigma): \exists r \in [0, m]:$

$(\delta(q, a) \in Q_r \wedge \delta(q', a) \in Q_r)$

und

$\forall v, v' \in [0, k_i]:$

$(v \neq v' \Rightarrow (\forall q \in Q'_{iv}: \forall q' \in Q'_{iv'}): \exists a \in \Sigma, r, s \in [0, m]:$

$(\delta(q, a) \in Q_r \wedge \delta(q', a) \in Q_s \wedge r \neq s))$

end;

PART := { $Q'_{ij} \mid i \in [0, m], j \in [0, k_i]$ }

until PART_{alt} = PART; ** Stabilisierung **

** Sei $P = \{Q_0, \dots, Q_m\}$ der Wert von PART nach der Stabilisierung.
 \Rightarrow Für alle $i \in [0, m]$ ist Q_i Repräsentant einer Äquivalenzklasse $[x_i]$ von Σ^* bezüglich P (vgl. auch Satz 3.4).

O.B.d.A. gelte:

1. Q_0 ist Repräsentant von $[\epsilon]$
2. $\forall v \in [0, k], f_v \in [0, m]: Q_{f_v}$ seien die Repräsentanten aller $[x_{f_v}]$ mit $x_{f_v} \in L(A)$.

Man setzt:

$A_1 := (P, \Sigma, \delta_1, Q_0, F_1)$ mit:

$F_1 = \{Q_{f_v}\}$ und

$\delta_1(Q_i, a) = Q_j \Leftrightarrow [x_j] = [x_i a]$

A_1 ist der minimale vollständige deterministische endliche Automat mit $L(A_1) = L(A)$.

Falls P einen Zustand Q_i enthält mit $Q_i \notin F_1$ und $\forall a \in \Sigma: \delta_1(Q_i, a) = Q_i$, dann setzt man:

- $A' := (\overline{P}, \Sigma, \overline{\delta_1}, Q_0, F_1)$, mit:

$\overline{P} = P - \{Q_i\}$ und

$\overline{\delta_1} : \overline{P} \times \Sigma \rightarrow \overline{P}$ mit: $\forall (q, a) \in \overline{P} \times \Sigma: \overline{\delta_1}(q, a) = \delta_1(q, a)$.

und

- $A' = A_1$ sonst

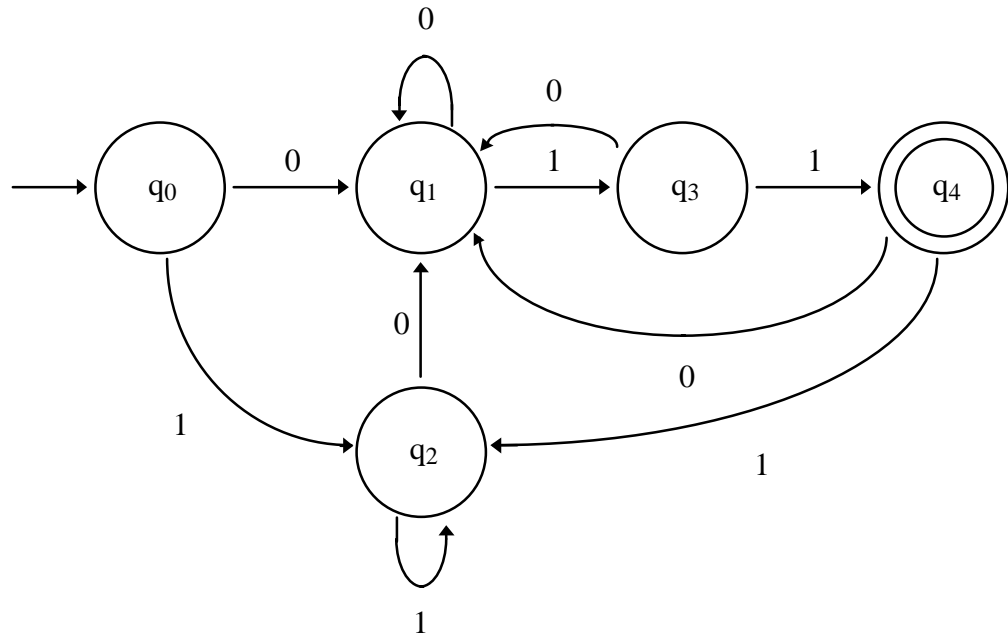
**

end.

Beispiel

(zur Minimierung der Zustandsmenge eines endlichen Automaten)

Betrachte folgenden vollständigen deterministischen endlichen Automaten A:



Erzeugte Sprache: $(0+1)^*011$.

Wir wollen nun den minimalen DEA A' nach obigen Algorithmus konstruieren.

Initialisierung: $P_0 := \{\{q_4\}, \{q_0, q_1, q_2, q_3\}\} = \{Q_1, Q_0\}$

Gesucht wird eine Partition von Q_0 . Da:

- 1) i) $\delta(q_3, 0) \in Q_0, \delta(q_3, 1) \in Q_1$
- ii) $\forall i \in \{0, 1, 2\}: \forall a \in \Sigma: \delta(q_i, a) \in Q_0$
- 2) $\delta(q_3, 1) = Q_1, \forall i \in \{0, 1, 2\}: \delta(q_i, 1) \in Q_0$

$\Rightarrow P_1 = \{\{q_4\}, \{q_3\}, \{q_0, q_1, q_2\}\} = \{Q_2, Q_1, Q_0\}$
Partition

Gesucht wird eine Partition von Q_0 . Da:

- 1) i) $\delta(q_1, 0) \in Q_0, \delta(q_1, 1) \in Q_1$
- ii) $\forall i \in \{0, 2\}: \forall a \in \Sigma: \delta(q_i, a) \in Q_0$
- 2) $\delta(q_1, 1) = Q_1, \forall i \in \{0, 2\}: \delta(q_i, 1) \in Q_0$

$\Rightarrow P_2 = \{\{q_4\}, \{q_3\}, \{q_1\}, \{q_0, q_2\}\} = \{Q_3, Q_2, Q_1, Q_0\}$
Partition

Da: $\forall x \in \{0,1\}: \delta(q_0,x)=\delta(q_2,x)$ gibt es keine weitere Partition.

Die Übergangsfunktion δ bildet nun jeden Zustand jeder Menge Q_i , $i=0,\dots,3$, bei gleicher Eingabe in dasselbe Q_j ab. Wir erhalten also die Übergangsfunktion δ' des neuen Automaten A' dadurch, daß wir die Funktion δ auf einen beliebigen Zustand aus jeder Menge Q_i , $i=0,\dots,3$, anwenden und nachschauen, in welcher der Mengen Q_j , $j=0,\dots,3$, das Ergebnis liegt:

$$\delta(q_0,0)=q_1 \in Q_1$$

$$\delta(q_0,1)=q_2 \in Q_0$$

$$\delta(q_1,0)=q_1 \in Q_1$$

$$\delta(q_1,1)=q_3 \in Q_2$$

$$\delta(q_2,0)=q_1 \in Q_1$$

$$\delta(q_2,1)=q_2 \in Q_0$$

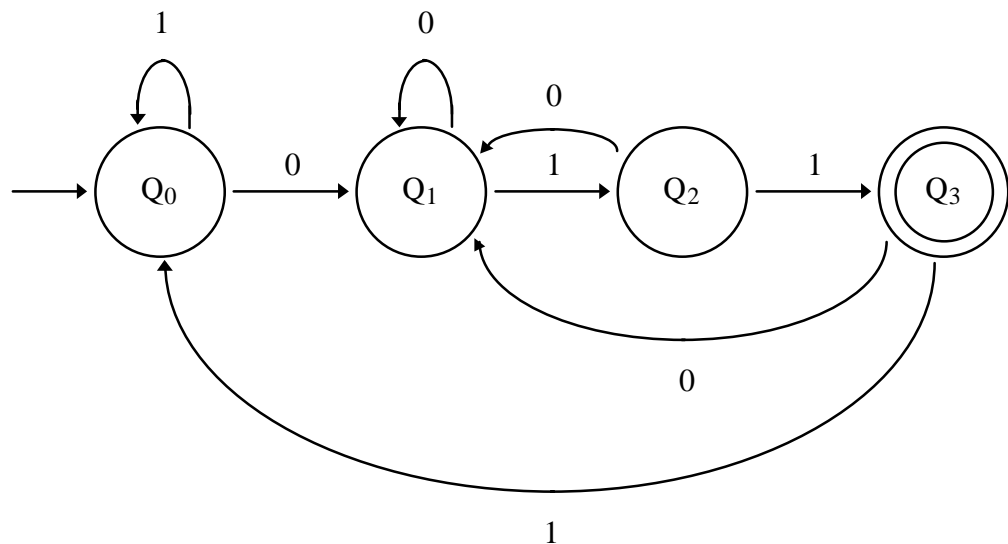
$$\delta(q_3,0)=q_1 \in Q_1$$

$$\delta(q_3,1)=q_4 \in Q_3$$

$$\delta(q_4,0)=q_1 \in Q_1$$

$$\delta(q_4,1)=q_2 \in Q_0$$

Der neue Automat ergibt sich somit zu:



4 Syntaxanalyse

In Abschnitt 4.1.1 und 4.1.2 werden wir zwei (nichtdeterministische) Verfahren zur Syntaxanalyse vorstellen. Sie haben den Vorteil, daß sie einfach zu implementieren sind, allerdings muß man dafür aber lange Laufzeiten (exponentiell zur Wortlänge) in Kauf nehmen.

Die Konsequenz ist die Entwicklung anderer Verfahren, was uns zur Theorie der Syntaxanalyse führt. Allerdings müssen dazu die uns bekannten endlichen Automaten erweitert werden. Denn kontextfreie Grammatiken können nicht von endlichen Automaten erkannt werden, da die Klasse der von endlichen Automaten akzeptierten Sprachen **echt** kleiner ist als die Klasse der Sprachen, die von kontextfreien Grammatiken erzeugt wird.

Wir werden zeigen, daß die Klasse von Sprachen, die von (nichtdeterministischen) Kellerautomaten akzeptiert wird, genau die Klasse der kontextfreien Sprachen ist.

4.1 Nichtdeterministische Verfahren zur Syntaxanalyse

Sei $G=(N,T,P,S)$ mit:

$N = \{E,D,F\}$

$T = \{\alpha, +, *, (,)\}$

$P = \{ E ::= E+D \Leftrightarrow E+D \leftarrow E \quad (\text{Regel 1})$

$E ::= D \Leftrightarrow D \leftarrow E \quad (\text{Regel 2})$

$D ::= D * F \Leftrightarrow D * F \leftarrow D \quad (\text{Regel 3})$

$D ::= F \Leftrightarrow F \leftarrow D \quad (\text{Regel 4})$

$F ::= (E) \Leftrightarrow (E) \leftarrow F \quad (\text{Regel 5})$

$F ::= \alpha \Leftrightarrow \alpha \leftarrow F \quad (\text{Regel 6})$

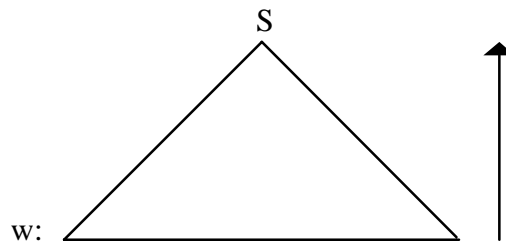
}

$S = E$

Überprüfe, ob: $w := \alpha + \alpha * \alpha \in L(G)$.

4.1.1 Bottom-up-Analyse mit Backtracking

Sei $w=w_1w_2\dots w_n$, $P=\{P_1,\dots,P_m\}$. Es wird versucht, w auf das Startsymbol S zu reduzieren.



Dabei wird das Paar (ε, w) in ein Paar (X, ε) transformiert, wobei $X \in (N \cup T)^*$.

Es gilt: $w \in L(G) \Leftrightarrow (X, \varepsilon) = (S, \varepsilon)$.

Die Transformation erfolgt folgendermaßen:

Sei $i:=0$, $w_0 := \varepsilon$, $W := w_i$.

- 1) Betrachte das Paar $(XW, w_{i+1}\dots w_n)$, $X=X_1X_2\dots X_k \in (N \cup T)^*$, $X_0 := \varepsilon$, $W \in (N \cup T)$.
Sei $\overline{X} := W$.

FOR $t:=k$ DOWNTO 0 DO BEGIN

(* es wird nun XW von rechts nach links vollständig reduziert *)

- Reduziere \overline{X} so weit wie möglich.

Dabei wird das Produktionensystem $P=\{P_1,\dots,P_m\}$ von links nach rechts nach einer passenden Reduktionsregel durchsucht; die Nummer der benutzten Regel wird für das entsprechende Paar abgespeichert, so daß die Reduktionsschritte zurückverfolgbar sind.

Es gelte nun $\overline{X} \leftarrow^* Z$, Z nicht weiter reduzierbar.

- $\overline{X} := X_t \circ Z$ (* Konkatenation *)

END.

- 2) Sei $(\overline{X}, w_{i+1} \dots w_n)$ das Paar, das durch Reduktion aus $(XW, w_{i+1} \dots w_n)$ entstanden ist.
- Falls $(\overline{X}, w_{i+1} \dots w_n) \equiv (\overline{X}, \varepsilon)$, so überführe $(\overline{X}, w_{i+1} \dots w_n)$ durch Shiften in $(\overline{X} w_{i+1}, w_{i+2} \dots w_n) \equiv (\overline{X} W, w_{i+2} \dots w_n)$, $i := i+1$ und fahre mit 1) fort.
 - Ist $(\overline{X}, \varepsilon) = (S, \varepsilon)$, so sind wir fertig. Es gilt: $w \in L(G)$.
 - Es ist $(X, \varepsilon) \neq (S, \varepsilon)$.
 - Es wird die letzte Reduktion (mit eventuell vorangehenden Shiftoperationen) rückgängig gemacht und danach mit der nächsten Reduktionsmöglichkeit bei 1) weitergemacht.. Dieses Vorgehen bezeichnet man als *Backtracking*.
Sind alle Möglichkeiten erschöpft und ist $X \neq w_1 \dots w_n$, so wird mit Punkt 2ci) weitergemacht.
 - Das Paar (ε, w) wurde durch einfaches Shiften in das Paar (X, ε) überführt, d.h. $X = w_1 \dots w_n$. Dann gilt: $w \notin L(G)$.

Beispiel

Bottom-up-Analyse für $w = \alpha + \alpha^* \alpha$; es wird versucht, w auf das Startsymbol E zu reduzieren.

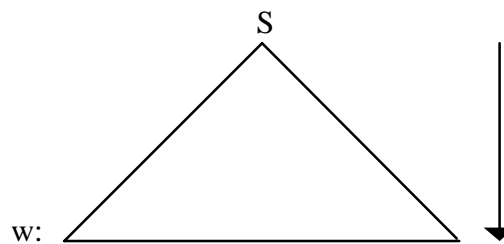
$(\varepsilon, \alpha + \alpha^* \alpha)$
 \downarrow Shift
 $(\alpha, + \alpha^* \alpha)$
 \downarrow Regel 6
 $(F, + \alpha^* \alpha)$
 \downarrow Regel 4
 $(D, + \alpha^* \alpha)$
 \downarrow Regel 2
 $(E, + \alpha^* \alpha)$
 \downarrow Shift
 $(E+, \alpha^* \alpha)$
 \downarrow Shift
 $(E+\alpha, * \alpha)$
 \downarrow Regel 6
 $(E+F, * \alpha)$
 \downarrow Regel 4

$(E+D,*\alpha)$	$(*)$	$(E+D,*\alpha)$	$(*)$	$(E+D,*\alpha)$
↓ Regel 1		↓ Regel 2		↓ Shift
$(E,*\alpha)$		$(E+E,*\alpha)$		$(E+D*,\alpha)$
↓ Shift		↓ Shift		↓ Shift
$(E*,\alpha)$		$(E+E*,\alpha)$		$(E+D*\alpha,\varepsilon)$
↓ Shift		↓ Shift		↓ Regel 6
$(E*\alpha,\varepsilon)$		$(E+E*\alpha,\varepsilon)$		$(E+D*F,\varepsilon)$
↓ Regel 6		↓ Regel 6		↓ Regel 4
$(E*F,\varepsilon)$		$(E+E*F,\varepsilon)$		$(E+D,\varepsilon)$
↓ Regel 4		↓ Regel 4		↓ Regel 1
$(E*D,\varepsilon)$		$(E+E*D,\varepsilon)$		(E,ε)
↓ Regel 2		↓ Regel 1		$\Rightarrow w \in L(G)$
$(E*E,\varepsilon)$		$(E+E*E,\varepsilon)$		
\Rightarrow keine weitere Reduktion möglich; $E*E \neq E$		\Rightarrow keine weitere Reduktion möglich; $E+E*E \neq E$		
\Rightarrow Backtracking		\Rightarrow Backtracking		

(In beiden Fällen muß insgesamt bis $(*)$ zurückgegangen werden)

4.1.2 Top-down-Analyse mit Backtracking

Es wird versucht, das Startsymbol S auf w abzuleiten.



Dabei wird das Paar (ε, S) in ein Paar (z, X) transformiert, wobei $X \in (N \cup T)^*$, $z \in T^*$.

Es gilt: $w \in L(G) \Leftrightarrow (z, X) = (w, \varepsilon)$.

Die Transformation erfolgt folgendermaßen:

Sei $i:=0$, $w_0 := \varepsilon$, $X:=S$, $\alpha := \varepsilon$.

- 1) Betrachte das Paar $(w_0 \dots w_i, X\alpha)$, $X \in N$, $\alpha \in (N \cup T)^*$.

Leite X so weit wie möglich ab.

Dabei wird das Produktionensystem $P = \{P_1, \dots, P_m\}$ von links nach rechts nach einer passenden Ableitungsregel P_i durchsucht. Passend heißt, daß

- die Ableitungsregel P_i nur Terminalsymbole enthält, die auch in w vorkommen,
- $X\alpha$ durch die Ableitung nicht länger wird als das Analysewort w .

Die Nummer der benutzten Regel wird für das entsprechende Paar abgespeichert, so daß die Ableitungsschritte zurückverfolgbar sind.

- 2) Sei $(\overline{w_0 \dots w_i}, \overline{X} \overline{\alpha})$ das Paar, das durch Ableiten aus $(w_0 \dots w_i, X\alpha)$ entstanden ist, $\overline{X} \in (N \cup T \cup \{\varepsilon\})$, $\overline{\alpha} \in (N \cup T)^*$.

- a) $\overline{X} = w_{i+1} \in T$.

Überführe $(\overline{w_0 \dots w_i}, \overline{X} \overline{\alpha})$ durch Shiften in $(\overline{w_0 \dots w_{i+1}}, \overline{\alpha})$, $i := i+1$.

- b) $\overline{X} \neq w_{i+1} \in T \vee \overline{X} \in N$.

Mache die letzte Ableitung (mit eventuell vorangehenden Shiftoperationen) rückgängig und fahre mit der nächsten passenden Ableitungsmöglichkeit bei 1) fort.

Sind alle Möglichkeiten erschöpft (d.h. $X=S$), dann gilt: $w \notin L(G)$.

- c) $\overline{X} = \varepsilon$.

$\Rightarrow \overline{\alpha} = \varepsilon$, fahre mit 3) fort.

- 3) a) $\overline{\alpha} = \varepsilon$ und $i = n$.

$\Rightarrow w \in L(G)$.

- b) $\overline{\alpha} = \varepsilon$ und $i \neq n$.

Mache die letzte Ableitung (mit eventuell vorangehenden Shiftoperationen) rückgängig, ersetze sie durch die nächste passende Ableitungsmöglichkeit und fahre mit 2) fort.

- c) Fahre mit 2) fort.

Beispiel

Top-down-Analyse für $w = \alpha + \alpha * \alpha$; es wird versucht, das Startsymbol E auf w abzuleiten.

(ϵ, E)

↓ Regel 1

($\epsilon, E+D$)

↓ Regel 1 (*)

($\epsilon, E+D+D$)

Bemerkung: $E ::= E+D$ nicht anwendbar, da (wegen ϵ -Freiheit) keine Verkürzungsregel vorhanden ist und somit $E+D+D+D$ nicht mehr auf $w = \alpha + \alpha * \alpha$ abgeleitet werden kann.

↓ Regel 2

($\epsilon, D+D+D$)

↓ Regel 4

($\epsilon, F+D+D$)

Bemerkung: $F ::= (E)$ nicht anwendbar, da "(" in w nicht vorkommt.

↓ Regel 6

($\epsilon, \alpha + D + D$)

↓ Shift

($\alpha, + D + D$)

↓ Shift

($\alpha +, D + D$)

↓ Regel 4

($\alpha +, F + D$)

↓ Regel 6

($\alpha +, \alpha + D$)

↓ Shift

($\alpha + \alpha, + D$)

$\Rightarrow + \neq *$

\Rightarrow Backtracking

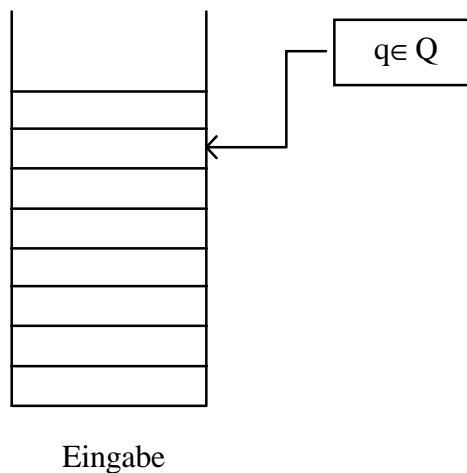
Insgesamt muß bis (*) zurückgesprungen werden. Hier die darauffolgenden Ableitungen:

$(\epsilon, E+D)$
 \downarrow Regel 2
 $(\epsilon, D+D)$
 \downarrow Regel 4
 $(\epsilon, F+D)$
 \downarrow Regel 6
 $(\epsilon, \alpha+D)$
 \downarrow Shift
 $(\alpha, +D)$
 \downarrow Shift
 $(\alpha+, D)$
 \downarrow Regel 3
 $(\alpha+, D^*F)$
 \downarrow Regel 4
 $(\alpha+, F^*F)$
 \downarrow Regel 6
 $(\alpha+, \alpha^*F)$
 \downarrow Shift
 $(\alpha+\alpha, ^*F)$
 \downarrow Shift
 $(\alpha+\alpha^*, F)$
 \downarrow Regel 6
 $(\alpha+\alpha^*, \alpha)$
 \downarrow Shift
 $(\alpha+\alpha^*\alpha, \epsilon)$
 $\Rightarrow w \in L(G)$

4.2 Kellerautomaten

Durch die Produktion $S ::= ab \mid aSb$ wird der Satz $w = a^n b^n$ erzeugt. Es existiert kein endlicher Automat A , der $a^n b^n$ für alle $n \in \mathbb{N}$ erkennt, da A nur eine feste (endliche) Anzahl von Zuständen besitzt.

Allgemeine Darstellung eines endlichen Automaten:

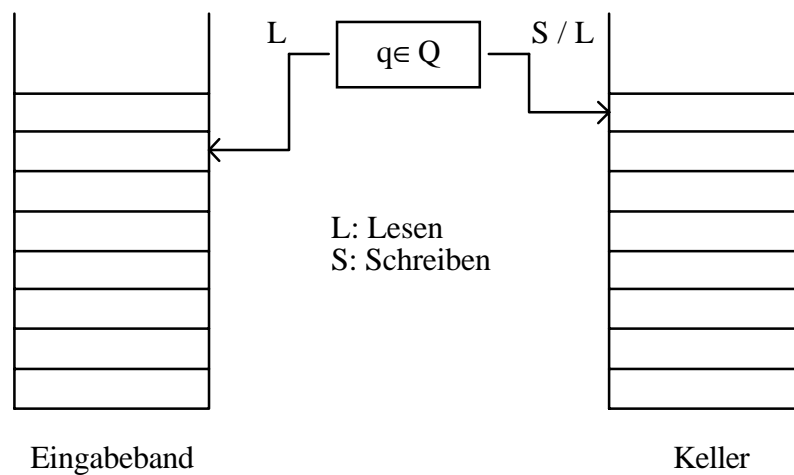


Aber der wie folgt arbeitende Kellerautomat erkennt $a^n b^n$:

Eingabe $a \Rightarrow$ gehe in Zustand q (schreibe a in den Keller)

Eingabe $b \Rightarrow$ gehe in Zustand q' (lösche ein a aus dem Keller)

Allgemeine Darstellung eines Kellerautomaten:



Es gibt 1-Weg- und 2-Weg-Kellerautomaten; sie unterscheiden sich durch die Bewegungsrichtungen des Lesekopfes auf dem Eingabeband.

4.2.1 (Nichtdeterministische) 1-Weg-Kellerautomaten

Definition 4.1 Ein *1-Weg-Kellerautomat* $(1KA, KA)$ ist ein Tupel $KA=(Q,T,\Gamma,\delta,q_0,Z_0,F)$, wobei:

Q : endliche Zustandsmenge

T : Eingabealphabet

Γ : Kelleralphabet

$\delta: Q \times \Gamma^+ \times (T \cup \{\varepsilon\}) \rightarrow \wp(Q \times \Gamma^*)$

$q_0 \in Q$: Startzustand

$Z_0 \in \Gamma$: Kellerstartsymbol

$F \subseteq Q$: Menge der Endzustände.

Bemerkung

$\delta(q,Z,\varepsilon)=(q,\alpha_1 \dots \alpha_n)$ bedeutet: Der Kellerautomat KA im Zustand q und mit Z als oberstem Kellersymbol ersetzt unabhängig von der Eingabe das oberste Kellersymbol durch $\alpha_1 \dots \alpha_n$; der Lesekopf wird nicht vorangeschoben.

Definition 4.2 $(q,\gamma,x) \in Q \times \Gamma^* \times T^*$ heißt *Konfiguration*, wobei:

q : momentaner Zustand

γ : momentaner Kellerinhalt

x : noch nicht verarbeiteter Teil der Eingabe, wobei der Lesekopf auf dessen ersten Symbol steht.

Definition 4.3 Seien $q,q' \in Q$, $\alpha \in \Gamma^+$, $\beta,\gamma \in \Gamma^*$, $a \in T \cup \{\varepsilon\}$, $x \in T^*$. Dabei bedeutet:

$a \in T$: Eingabezeiger wird bewegt

$a = \varepsilon$: keine Veränderung der Eingabe .

Dann: $(q,\gamma\alpha,ax) \vdash (q',\gamma\beta,x) :\Leftrightarrow (q',\beta) \in \delta(q,\alpha,a)$.

\vdash^+ sei die transitive Hülle von \vdash .

\vdash^* sei die transitiv-reflexive Hülle von \vdash .

Definition 4.4 Sei $KA=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ ein Kellerautomat, $f \in F$, $q \in Q$.

- a) $w \in T^*$ wird unter Erreichen eines Endzustandes akzeptiert
 $:\Leftrightarrow (q_0, Z_0, w) \vdash^* (f, \gamma, \varepsilon)$.
- b) $w \in T^*$ wird unter Leeren des Kellers akzeptiert
 $:\Leftrightarrow (q_0, Z_0, w) \vdash^* (q, \varepsilon, \varepsilon)$.

Satz 4.1 Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik. Dann existiert ein Kellerautomat K , der einen Satz unter Leeren des Kellers akzeptiert, mit $L(G) = L_e(K)$.

Beweis:

Sei $K := (\{q\}, T, N \cup T, \delta, q, S, \emptyset)$ mit:

1. $\forall A ::= \alpha \in P, \alpha = \alpha_1 \dots \alpha_n : \delta(q, A, \varepsilon)$ enthält $(q, \alpha_n \dots \alpha_1)$
 (oberstes Kellersymbol wird durch $\alpha_n \dots \alpha_1$ ersetzt (**Reihenfolge beachten!**), d.h. das am weitesten rechts stehende Symbol von α kommt als erstes Symbol auf den Keller; entspricht: Ableitung)
2. $\forall b \in T: \delta(q, b, b) = \{(q, \varepsilon)\}$
 (oberstes Kellersymbol wird gelöscht; entspricht: pop)

q.e.d.

Bemerkung

Dieser Automat erzeugt eine Linksableitung für das Analysewort.

Beispiel

Betrachte $G=(\{\alpha, \tau\}, \{V, +, *\}, P, \alpha)$ mit $P = \{ \alpha ::= \tau \mid \alpha + \tau, \tau ::= V \mid \tau * V \}$ (vergleiche auch Seite 25).

Dann existiert ein Kellerautomat K mit $L(G) = L_e(K)$.

Betrachte $w := V+V*V \in L(G)$. Dann durchläuft dieser Kellerautomat K folgende Zustände:

$$\begin{array}{ccccccc}
 \begin{bmatrix} V \\ * \\ V \\ + \\ V \end{bmatrix} & \rightarrow & \begin{bmatrix} V \\ * \\ V \ \alpha \\ + \\ V \ \tau \end{bmatrix} & \rightarrow & \begin{bmatrix} V \\ * \\ V \ \tau \\ + \\ V \ \tau \end{bmatrix} & \rightarrow & \begin{bmatrix} V \\ * \\ V \ V \\ + \\ V \ \tau \end{bmatrix} \\
 \\
 \rightarrow & \begin{bmatrix} V \\ * \\ V \ + \\ + \ \tau \end{bmatrix} & \rightarrow & \begin{bmatrix} V \\ * \\ V \ \tau \end{bmatrix} & \rightarrow & \begin{bmatrix} V \ \tau \\ * \\ V \ V \end{bmatrix} & \rightarrow & \begin{bmatrix} V \ V \\ * \\ V \ V \end{bmatrix} \\
 \\
 \rightarrow & \begin{bmatrix} V \ * \\ * \ V \end{bmatrix} & \rightarrow & \begin{bmatrix} V \ V \end{bmatrix} & \rightarrow & \begin{bmatrix} \varepsilon \ \varepsilon \end{bmatrix}
 \end{array}$$

$\Rightarrow w \in L(G)$

Satz 4.2 Gegeben sei eine kontextfreie Grammatik $G=(N,T,P,S)$. Es existiert ein Kellerautomat K , der einen Satz unter Erreichen eines Endzustandes akzeptiert, mit: $L(G) = L_b(K)$.

Beweis:

Sei $K := (\{q,f\}, T, \Gamma, \delta, q, \$, \{f\})$ mit:

$\Gamma := N \cup T \cup \{\$, \# \}$, $\$ \notin (N \cup T)$

- δ :
1. $\forall b \in T, X \in \Gamma: \delta(q, X, b) = \{(q, Xb)\}$ (entspricht: Shiften)
 2. $\forall A ::= \alpha \in P: \delta(q, \alpha, \varepsilon)$ enthält (q, A) (entspricht: Reduktion)
 3. $\delta(q, \$S, \varepsilon) = \{(f, \varepsilon)\}$

q.e.d.

Bemerkung

Dieser Automaten erzeugt eine Rechtsreduktion für das Analysewort.

Beispiel

Betrachte $G=(\{\alpha,\tau\},\{V,+,*\},P,\alpha)$ mit $P=\{ \alpha::=\tau \mid \alpha+\tau, \tau::=V \mid \tau*V \}$ (vergleiche auch Seite 25).

Dann existiert ein Kellerautomat K mit $L(G) = L_b(K)$.

Betrachte $w:=V+V*V \in L(G)$. Dann durchläuft dieser Kellerautomat K folgende Zustände:

$$\begin{array}{ccccccc}
 \left[\begin{array}{c} V \\ * \\ V \\ + \\ V \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} V \\ * \\ V \\ + \\ V \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} V \\ * \\ V \\ + \\ \tau \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} V \\ * \\ V \\ + \\ \alpha \\ \$ \end{array} \right] \\
 \rightarrow & \left[\begin{array}{c} V \\ * \\ V \\ + \\ \alpha \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} V \\ + \\ V \\ * \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} \tau \\ + \\ V \\ * \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} * \\ \tau \\ + \\ \alpha \\ V \\ \$ \end{array} \right] \\
 \rightarrow & \left[\begin{array}{c} V \\ * \\ \tau \\ + \\ \alpha \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} \tau \\ + \\ \alpha \\ \$ \end{array} \right] & \rightarrow & \left[\begin{array}{c} \alpha \\ \$ \end{array} \right]
 \end{array}$$

$\Rightarrow w \in L(G)$

Satz 4.3 Sei $L_e(K)$ die von einem nichtdeterministischen Kellerautomaten K akzeptierte Sprache, wobei K Sätze unter Leeren des Kellers akzeptiert. Dann ist L eine kontextfreie Sprache.

Beweisskizze:

Sei $K=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ der Kellerautomat. Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik mit N als einer Menge von Objekten der Form $[q,A,p]$ mit p,q aus Q und A aus Γ und einem neuen Symbol S . P sei die Menge folgender Produktionen (in Greibach-Normalform):

- 1) $S ::= [q_0, Z_0, q]$ für jedes $q \in Q$.
- 2) $[q, A, q_{m+1}] ::= a[q_1, B_1, q_2] [q_2, B_2, q_3] \dots [q_m, B_m, q_{m+1}]$ für $q, q_1, \dots, q_{m+1} \in Q$, $a \in T \cup \{\epsilon\}$ und $A, B_1, \dots, B_m \in \Gamma$, so daß $(q_1, B_1 \dots B_m) \in \delta(q, A, a)$ (gilt $m=0$, so lautet die Produktion $[q, A, q_1] ::= a$).

Für den Beweis des Satzes ist die Kenntnis hilfreich, daß die Variablen und Produktionen von G so definiert worden sind, daß eine Linksableitung in G von einem Satz x eine Simulation des Kellerautomaten K ist, wenn dieser mit der Eingabe x gestartet wurde. Insbesondere entsprechen die Variablen, die in einem Schritt einer Linksableitung in G auftauchen, jeweils den Symbolen auf dem Keller von K zu dem Zeitpunkt, an dem K genausoviel von der Eingabe gesehen hat, wie die Grammatik bereits erzeugt hat. Anders gesagt: Die Intention ist es, daß aus $[q, A, p]$ genau dann x ableitbar ist, wenn K infolge x ein A durch eine Folge von Bewegungen, die im Zustand q anfangen und im Zustand p enden, von seinem Keller löscht.

Um zu zeigen, daß $L(G) = L(K)$ gilt, beweisen wir durch Induktion über die Schrittzahl in einer Ableitung von G bzw. über die Anzahl der Bewegungen in K , daß

$$[q, A, p] \xrightarrow{*} x \quad \Leftrightarrow \quad (q, A, x) \vdash^* (p, \epsilon, \epsilon)$$

(kompletter Beweis: Hopcroft / Ullman).

q.e.d.

Satz 4.4 Für die Sprache L sind folgende Aussagen äquivalent:

- 1) $L = L(K)$ für einen Kellerautomaten K , der nur auf den obersten Kellereintrag zugreifen kann und einen Satz unter Erreichen eines Endzustandes akzeptiert.
- 2) $L = L_e(K)$ für einen Kellerautomaten K , der nur auf den obersten Kellereintrag zugreifen kann und einen Satz unter Leeren des Kellers akzeptiert.
- 3) $L = L(K)$ für einen beliebigen Kellerautomaten K , der einen Satz unter Erreichen eines Endzustandes akzeptiert.
- 4) $L = L_e(K)$ für einen beliebigen Kellerautomaten K , der einen Satz unter Leeren des Kellers akzeptiert.
- 5) $L = L(G)$ für eine kontextfreie Grammatik.

Beweis:1) \Rightarrow 3) trivial

3) \Rightarrow 1) Simulation eines Kellerautomaten K mit Zugriff auf beliebig viele Kellersymbole durch einen Kellerautomaten K' mit Zugriff nur auf das oberste Kellersymbol:
 Sei $K=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ gegeben. Da im Keller eine beliebige Anzahl von endlich vielen Symbolen steht, existiert eine Zahl N der maximal bei einem Arbeitsschritt gelesenen Kellersymbole. Sei $\$ \notin \Gamma$, $\Gamma' := (\Gamma \cup \{\$\})^N$.

Durch $f: \bigcup_{i=0}^N \Gamma^i \rightarrow \Gamma'$,

$$f(\gamma_1, \dots, \gamma_i) = (\gamma_1, \dots, \gamma_i, \$, \dots, \$), \text{ wobei } \text{Anz}(\$) + i = N,$$

ist jedem gelesenen Kellereintrag von K ein Kellereintrag von K' zugeordnet (der Kellereintrag von K' ist also ein N -Tupel). Somit läßt sich K' definieren als:

$K' := (Q, T, \Gamma', \delta', q_0, f(Z_0), F)$ mit:

f, Γ' wie oben,

$$\delta'(q, \gamma, a) := \{(p, f(\alpha) \mid (p, \alpha) \in \delta(q, f^{-1}(\gamma), a)\}$$

2) \Rightarrow 4) trivial4) \Rightarrow 2) Analog zu: 3) \Rightarrow 1)

3) \Rightarrow 4) Simulation eines Kellerautomaten K' , der Worte unter Erreichen eines Endzustandes akzeptiert, durch einen Kellerautomaten K , der Worte unter Leeren des Kellers akzeptiert:

Sei $K=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ gegeben. Setze $K' := (Q', T, \Gamma', \delta', q_A, \perp, F')$ mit:

$$Q' := Q \cup \{q_A, q_E\}, \quad q_A, q_E \notin Q,$$

$$F' := \{q_E\},$$

$$\Gamma' := \Gamma \cup \{\perp\}, \quad \perp \notin \Gamma,$$

$$\delta' := \delta \cup \{((q_A, \perp, \varepsilon), (q_0, \perp Z_0))\} \cup \bigcup_{\substack{q \in Q \\ q \neq q_A, q_E}} \{((q, \perp, \varepsilon), (q, \varepsilon))\}.$$

K' hat \perp als unterstes Kellersymbol. Nachdem Z_0 auf den Keller geschrieben wurde, arbeitet K' genauso wie K . Wenn K ein Wort durch Leeren des Kellers akzeptiert hat, steht nur noch \perp im Keller, so daß K' nur noch in den Endzustand zu springen braucht.

4) \Rightarrow 3) Simulation eines Kellerautomaten K' , der Worte unter Leeren des Kellers akzeptiert, durch einen Kellerautomaten K , der Worte unter Erreichen eines Endzustandes akzeptiert:

Sei $K=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ gegeben. Setze $K' := (Q',T,\Gamma',\delta',q_0,Z_0,F')$ mit:

$$Q' := Q \cup \{q'\}, q' \in Q,$$

$$F' := \emptyset,$$

$$\delta' := \delta \cup \bigcup_{f \in F, \gamma \in \Gamma} \{((f,\gamma,\epsilon),(q',\gamma))\} \cup \bigcup_{\gamma \in \Gamma} \{((q',\gamma,\epsilon),(q',\epsilon))\} .$$

Wurde ein Endzustand nach Abarbeitung des Eingabewortes erreicht, geht der Kellerautomat K' in den Zustand q' über und löscht alle Kellersymbole.

5) \Rightarrow 2) Satz 4.1

5) \Rightarrow 3) Satz 4.2

2) \Rightarrow 5) Satz 4.3

q.e.d.

4.2.2 Deterministische 1-Weg-Kellerautomaten

Definition 4.5 Ein Kellerautomat $K=(Q,T,\Gamma,\delta,q_0,Z_0,F)$ heißt *deterministisch (DKA)*, wenn er folgende Eigenschaften hat:

Für alle $q \in Q$, $a \in T \cup \{\epsilon\}$, $b \in \{a,\epsilon\}$ und für alle $\alpha, \beta \in \Gamma^*$ mit:

i) β ist Suffix von α oder α ist Suffix von β

ii) $(\alpha,a) \neq (\beta,b)$

ist

$$|\delta(q,\alpha,a)| + |\delta(q,\beta,b)| \leq 1$$

Definition 4.6 Eine kontextfreie Sprache heißt *deterministisch*, wenn sie von einem deterministischen 1KA **unter Erreichen eines Endzustandes** akzeptiert wird.

Bemerkung

Eine Sprache, die von einem deterministischen Kellerautomaten unter Leeren des Kellers erkannt wird, ist deterministisch, aber nicht jede deterministische Sprache wird von einem deterministischen Kellerautomaten unter Leeren des Kellers akzeptiert.

Definition 4.7 Eine deterministische Sprache L wird von einem deterministischen KA unter *Leeren des Kellers* akzeptiert $:\Leftrightarrow xy \in L, x \in L \Rightarrow y = \varepsilon$ (**Präfixfreiheit**).

Bemerkung:

1. Für eine beliebige kontextfreie Sprache ist nicht entscheidbar, ob sie deterministisch ist.
2. Präfixfreiheit kann man zum Beispiel durch Einführen eines Wort- (Satz-) Endesymbols erreichen (so zum Beispiel das \$-Zeichen im Beweis von Satz 4.2 auf Seite 53).

Definition 4.8 Ein *1-Weg-Kellertransduktor* $KT=(Q,T,U,\Gamma,\delta,q_0,Z_0,F)$ ist ein 1KA mit Ausgabe, wobei U das Ausgabealphabet ist.

Die von einem Transduktor M unter Übergang in einen Endzustand definierte *Übersetzung* ist gegeben durch:

$$\ddot{U}(M) := \{(x,y) \mid x \in T^*, y \in U^*, (q_0, Z_0, x, \varepsilon) \vdash^* (f, \gamma, \varepsilon, y), f \in F\}.$$

Die von einem Transduktor M unter Leeren des Kellers definierte *Übersetzung* ist gegeben durch:

$$\ddot{U}(M) := \{(x,y) \mid x \in T^*, y \in U^*, (q_0, Z_0, x, \varepsilon) \vdash^* (q', \varepsilon, \varepsilon, y), q' \in Q\}.$$

4.2.3 Leistungsfähigkeit von Kellerautomaten

- $L(1NKA) \subset L(2NKA)$
 Beispiel: Sei $L := \{1^n 0^n 2^n \mid n \geq 1\}$, dann $L \notin L(1NKA)$, aber $L \in L(2NKA)$.
 Allerdings ist L keine kontextfreie Sprache (siehe auch Beispiel auf Seite 19).
- $L(1DKA) \subset L(1NKA)$
 Beispiel: $L = \{0^n 1^n \mid n \geq 1\} \cup \{0^n 1^{2n} \mid n > 1\}$.
 Bemerkung: Der 1NKA ist abgeschlossen gegenüber Vereinigung, der DKA nicht.
- Nicht geklärt ist die Frage, ob es kontextfreie Sprachen gibt, die nicht von 2DKA erkannt werden.

Wichtig für die Syntaxanalyse:

Der deterministische 1-Weg-Kellerautomat akzeptiert zwar nur eine Teilmenge aller kontextfreien Sprachen, allerdings umfaßt diese Teilmenge die Syntax der meisten Programmiersprachen.

4.3 Analysestrategien

Bei Anwendung eines Erkennungs- oder Zerteilungsalgorithmus auf ein Wort w werden die einzelnen Symbole von w jeweils in einer bestimmten Reihenfolge gelesen, im allgemeinen von links nach rechts.

Wird dabei nach jedem Lesen eines Symbols von w festgestellt, ob das bisher gelesene Teilstück von w in $L(G)$ liegt und welche syntaktische Struktur es in diesem Fall hat, so spricht man von einem *sequentiellen Erkennungs-* bzw. *Zerteilungsalgorithmus* (auch *On-line-Algorithmus*). Wird dagegen nur ermittelt, ob w in $L(G)$ liegt und welche syntaktische Struktur w in diesem Fall hat, so spricht man von einem *Off-line-Erkennungs-* bzw. *-Zerteilungsalgorithmus*.

Die verschiedenen Methoden zur syntaktischen Analyse können danach klassifiziert werden, in welcher Reihenfolge verschiedene Teilstrukturen der syntaktischen Struktur, d.h. des Strukturbaums eines Wortes erkannt werden. Jeder solchen Klasse von Analysemethoden liegt somit eine jeweils charakteristische Analysestrategie zugrunde; darunter verstehen wir eine bestimmte einheitliche Vorgehensweise bezüglich der Reihenfolge, in der bestimmte Teilstrukturen erkannt werden. Die einer Analysestrategie zugehörigen Analysemethoden unterscheiden sich im wesentlichen dadurch, in welchem Maße Informationen über bereits bekannte Teilstrukturen zum weiteren Aufbau der Struktur herangezogen werden.

Im folgenden soll zunächst ein Überblick über die wichtigsten Analysestrategien gegeben werden. Wir unterscheiden dabei zwischen

- **Tabellenstrategien**
(z.B. der Analysealgorithmus von Cocke-Kasami-Younger oder das Analyseverfahren von Earley)
und
- **ableitungsorientierten Strategien.**

4.3.1 Tabellenorientierte Analysestrategien

Bei den sogenannten *Tabellenmethoden* erfolgt die Buchhaltung über die bereits erkannten Teilstrukturen in Tabellenform. Charakteristisch für die hier einzuordnenden Methoden ist folgendes Prinzip:

Man gewinnt systematisch Informationen über mögliche Ableitungsstrukturen für umfassendere Teilstücke des zu analysierenden Wortes aus in entsprechender Weise ermittelten Informationen über kleinere Teilstücke.

Diese Methoden arbeiten somit nach dem Prinzip der "Dynamischen Programmierung".

Eine mögliche Vorgehensweise ist zum Beispiel, daß man bei der Analyse eines Wortes w zuerst Ableitungsbäume zu allen Teilworten der Länge 1 und dann induktiv fortschreitend nacheinander Ableitungsbäume für immer längere Teilworte von w bestimmt, bis man entweder einen Strukturbaum von w findet oder feststellt, daß es keinen gibt.

4.3.1.1 Analysealgorithmus von Cocke-Kasami-Younger

Eingabe: Grammatik G in Chomsky-Normalform, $w=t_1 \dots t_n$.

Ausgabe: Analysematrix und Meldung: $w \in L(G)$ oder $w \notin L(G)$

```

for i:=0 to n-1 do
   $m_{i,i+1} := \{A \in N \mid A ::= t_{i+1} \in P\}$ ;
for d:=2 to n do
  for i:=0 to n-d do
    begin
      j:=i+d;
       $m_{i,j} := \{A \in N \mid \exists k, i < k < j: \exists B \in m_{i,k} : \exists C \in m_{k,j} : A ::= BC \in P\}$ 
    end;
  if  $S \in m_{0,n}$  then
     $w \in L(G)$ 
  else
     $w \notin L(G)$ ;

```

Beispiel

Sei $G=(\{S,A,B\},\{a,b,c\},P,S)$ mit $P=\{S::=SA \mid a, A::=BS, B::=BB \mid BS \mid b \mid c\}$.

Betrachte: $w:=abacba$.

Die zugehörige Analysematrix lautet:

	0	1	2	3	4	5	6
0	-	S	-	S	-	-	S
1	-	-	B	A,B	B	B	A,B
2	-	-	-	S	-	-	S
3	-	-	-	-	B	B	A,B
4	-	-	-	-	-	B	A,B
5	-	-	-	-	-	-	S
6	-	-	-	-	-	-	-

$S \in m_{0,6} \Rightarrow w \in L(G)$

Satz 4.5 Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik in Chomsky-Normalform und $w=t_1 \dots t_n \in T^*$. Sei $M_w=(m_{ij})$ die konstruierte Analysematrix. Dann gilt für alle $A \in N$:

$$A \in m_{ij} \quad \Leftrightarrow \quad A \xrightarrow{*} t_{i+1} t_{i+2} \dots t_j.$$

Beweis:

Induktion über $d=j-i$:

1. Schleife: Betrachte m_{ij} mit $j-i=1$.

Für diese Elemente ist die Aussage sicher richtig, da in dieser Diagonalen genau diese Nichtterminalen gesammelt werden.

2. Schleife: Sei $d \geq 2$ und Aussage ist richtig für alle $m_{\bar{i}\bar{j}}$ mit: $1 \leq \bar{j} - \bar{i} < d$, $d=j-i > 0$:

$$A \in m_{ij} \Leftrightarrow \exists k, i < k < j: \exists B \in m_{ik} : \exists C \in m_{kj} : A ::= BC \in P.$$

Aussage gilt für B und C. Daher:

$$A \rightarrow BC \xrightarrow{*} t_{i+1} \dots t_k C \xrightarrow{*} t_{i+1} \dots t_k t_{k+1} \dots t_j$$

q.e.d.

Aufwandabschätzung

Satz 4.6 Die Berechnung von M_w für ein gegebenes Wort w mit $|w|=n$ benötigt $O(n^3)$ Schritte.

Beweis:

1. Schleife: Die Bestimmung einer Komponente $m_{i-1,i} = \{A \in N \mid A ::= t_i \in P\}$ ist unabhängig von n

\Rightarrow Aufwand = c_1

2. Schleife: Eine Komponente $m_{i,j}$ mit $j-i=d \geq 2$:

$\left. \begin{array}{l} m_{ik} \\ m_{kj} \end{array} \right\} (d-1) \text{ Paare überprüfen (unabhängig von } n)$

\Rightarrow Aufwand = c_2

Anzahl der m_{ij} ist $(n+1-d)$.

Der Gesamtaufwand beträgt somit:

$$c_1 * n + c_2 * \sum_{d=2}^n (n+1-d)(d-1)$$

$$\leq c_1 * n + c_2 * \sum_{d=2}^n O(n^2)$$

$$\leq c_1 * n + c_2 * O(n^3)$$

$$= O(n^3)$$

q.e.d.

Bemerkung:

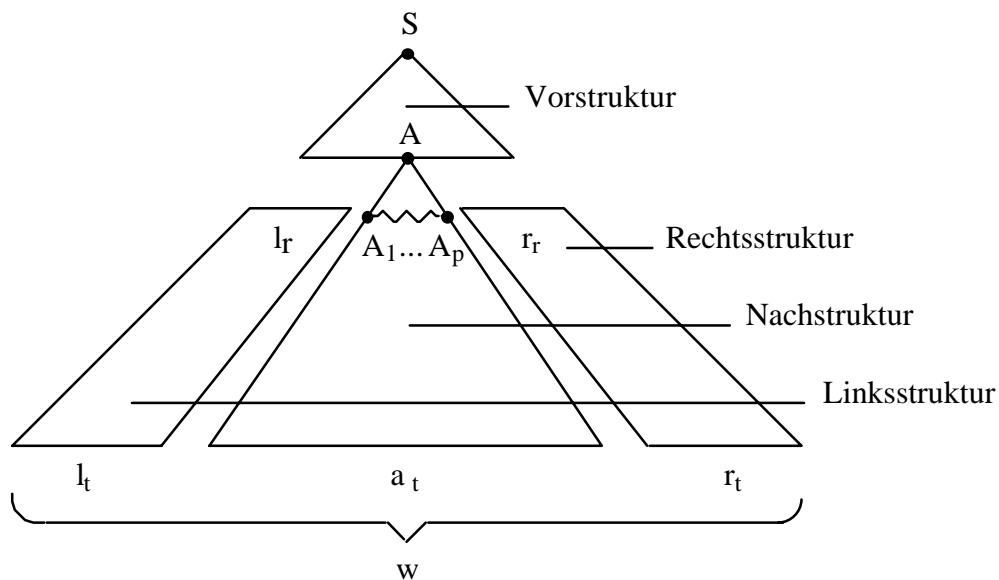
Es kann gezeigt werden, daß der Aufwand zur Berechnung von $M_w \leq O(n^{2,5})$ ist.

Satz 4.7 Der Speicherbedarf beträgt $\frac{n^2 + n}{2}$ Elemente. Jedes Element enthält höchstens $|N|$ Zeichen.

4.3.2 Ableitungsorientierte Analysestrategien

Ableitungsorientierte Analysestrategien sind dadurch gekennzeichnet, daß sie unmittelbar auf die Konstruktion einer Ableitung des zu analysierenden Wortes hinzielen. Ihnen liegen feinere Unterteilungen von Strukturbäumen in Teilstrukturen zugrunde.

Um dieses zu erläutern, betrachten wir zunächst ein von der kontextfreien Grammatik $G=(N,T,P,S)$ erzeugtes Wort w , einen zugehörigen Strukturbaum τ und in diesem eine feste Verzweigung, bestehend aus einem Knoten und seinen direkten Nachfolgern. Diese Verzweigung beschreibt die Anwendung einer Produktion, etwa $A::=\alpha$, mit $\alpha=A_1\dots A_p$ für ein $p \geq 0$ und $A_i \in (N \cup T)$ für $1 \leq i \leq p$. Die herausgegriffene Verzweigung besteht also aus einem Knoten, der mit A markiert, ist und seinen direkten Nachfolgern, die mit A_1 bis A_p markiert sind.



Durch diese Fixierung einer festen Verzweigung und der zugehörigen Anwendung einer Produktion wird τ in Teilstrukturen zerlegt. Dazu betrachten wir die Klasse τ_A aller Strukturbäume zu G , die Anfangsteilbäume von τ sind und den fest herausgegriffenen Knoten A als Endknoten haben. Diese haben Endschnittbilder der Form: lAr mit $l,r \in (N \cup T)^*$ und beschreiben Ableitungen für $S^* \rightarrow lAr$.

Sei τ_{\min} der eindeutig bestimmte Strukturbaum in τ_A mit minimaler Knotenzahl, und $l_r A r_r$ sei sein Endschnittbild. Wir nennen

$$\begin{aligned} l_r & : \text{ den } \textit{reduzierten Linkskontext} \\ r_r & : \text{ den } \textit{reduzierten Rechtskontext} \end{aligned}$$

zur betrachteten Anwendung der Produktion $A ::= \alpha$.

Sei τ_{\max} der eindeutig bestimmte Strukturbaum in τ_A mit maximaler Knotenzahl, und $l_t A r_t$ sei sein Endschnittbild. Dann sind $l_t, r_t \in T^*$. Wir nennen

$$\begin{aligned} l_t & : \text{ den } \textit{terminalen Linkskontext} \\ r_t & : \text{ den } \textit{terminalen Rechtskontext} \end{aligned}$$

zur betrachteten Anwendung der Produktion $A ::= \alpha$.

Sei τ_{bel} ein beliebig herausgegriffener Strukturbaum in τ_A , und $l A r$ sei sein Endschnittbild. Dann gilt:

$$\begin{aligned} l_r^* & \rightarrow l^* \rightarrow l_t \\ \text{und} \\ r_r^* & \rightarrow r^* \rightarrow r_t . \end{aligned}$$

Die fest herausgegriffene Anwendung einer Produktion $A ::= \alpha$ bestimmt in der betrachteten syntaktischen Struktur von w somit 4 Teilstrukturen:

- die Vorstruktur. Sie entspricht der Ableitung: $S^* \rightarrow l_r A r_r$
- die Linksstruktur. Sie entspricht der Ableitung: $l_r^* \rightarrow l_t$
- die Rechtsstruktur. Sie entspricht der Ableitung: $r_r^* \rightarrow r_t$
- die Nachstruktur. Sie entspricht der Ableitung: $\alpha^* \rightarrow \alpha_t$

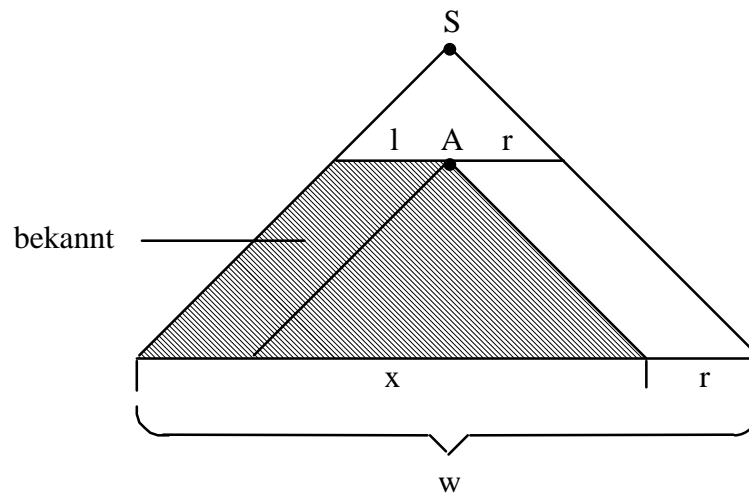
Die folgenden Analysestrategien sind dadurch charakterisiert, inwieweit die zu einer festen Anwendung einer Produktion gehörigen Teilstrukturen für das Erkennen der Anwendung dieser Produktion jeweils erkannt sein müssen:

- Linksstruktur total erkannt: Analyse von links nach rechts
- Rechtsstruktur total erkannt: Analyse von rechts nach links
- Vorstruktur total erkannt: Top-down
- Nachstruktur total erkannt: Bottom-up

(Kombinationen sind möglich)

Wir beschränken uns im folgenden auf die Analyse von links nach rechts.

4.3.2.1 Deterministische Bottom-up-Analyse (kanonische Reduktion)



Betrachte folgende Situation:

Bisher sei ein Präfix x von $w= xr$ auf $\gamma=lA$ reduziert worden, wobei:

- 1) $\gamma \neq \epsilon$ (mindestens eine Reduktion)
- 2) $\gamma=lA$, $A \in N$, A ist beim letzten Reduktionsschritt entstanden
- 3) Analyse ist noch nicht abgeschlossen

Für die zu reduzierende Teilzeichenreihe von γr gilt dann:

1. Sie ist ein Suffix von γ
oder
2. Sie ist ein Suffix von γ , gefolgt von einem nichtleeren Präfix von r (sonst würde gegen die kanonische Vorgehensweise von links nach rechts verstoßen).

Definition 4.9 (γ, r) heißt *aktuelles Paar*, wobei:

γ : Kellerinhalt
 r : noch nicht gelesene Eingabe.

Eine Rechtsreduktion von $w \in L(G)$ kann beschrieben werden durch Überführung von (ϵ, w) auf (S, ϵ) durch die Schritte "Shiften" und "Reduzieren". Dabei bedeutet:

- "Shiften" : nächstes Zeichen von w lesen und auf den Keller legen.
"Reduzieren" : eine Produktion auf die obersten Kellersymbole anwenden.

Wir definieren:

Definition 4.10 Sei G eine reduzierte kontextfreie Grammatik.

- 1) Zu jedem $A ::= \alpha \in P$ definiert man als die zugehörige *Reduktionsklasse* die Paarmenge:

$$R_{A ::= \alpha} := \{ (l\alpha, r) \mid l \in (N \cup T)^* \wedge r \in T^* \wedge S^* \rightarrow lAr \rightarrow l\alpha r \}$$

- 2) Als *Shiftklasse* bezeichnet man die Paarmenge

$$R_{\text{shift}} := \{ (l, r) \mid l \in (N \cup T)^* \wedge r \in T^* \wedge \\ \exists z \in T^+ : \exists \bar{r} \in T^* : \exists A ::= a \in P : (lr = lz \bar{r} \wedge (lz, \bar{r}) \in R_{A ::= a}) \}$$

Wenn die Reduktionsklassen einer Grammatik bekannt sind, kann eine deterministische Bottom-up-Analyse anhand dieser Klassen durchgeführt werden:

Bei einem durch ein aktuelles Paar (γ, r) gekennzeichneten Stand der Analyse eines Wortes w muß nur ermittelt werden, ob ein $p \in P \cup \{\text{shift}\}$ existiert, mit $(\gamma, r) \in R_p$, und wenn ja, welche Elemente p diese Eigenschaft haben. Aus diesen muß eines ausgewählt werden, und zwar nach einer fest vorgegebenen Vorschrift, die für jede Teilmenge von $P \cup \{\text{shift}\}$ das jeweils auszuwählende Element festlegt. Das ausgewählte Element p bestimmt dann, wie die Analyse fortzusetzen ist. Sie ist im Falle $p \in P$ mit einer Reduktion bezüglich der Produktion p und im Falle $p = \text{shift}$ mit einem Shift des ersten Symbols von r an das rechte Ende von γ fortzusetzen.

Auf diese Art und Weise erhält man ein Verfahren zur deterministischen Bottom-up-Analyse. Voraussetzung ist, daß man die Reduktionsklassen kennt und für jedes aktuelle Paar (γ, r) und jedes $p \in P \cup \{\text{shift}\}$ (in genügend einfacher Weise) entscheiden kann, ob (γ, r) in R_p liegt oder nicht. Der Aufwand zur Entscheidung dieser Frage ist jedoch im allgemeinen vergleichbar mit dem Aufwand für die Analyse des Wortes w selbst. Damit erscheint auch die Realisierung der deterministischen Bottom-up-Analyse mit einer Steuerung durch die Reduktionsklassen für beliebige kontextfreie Grammatiken nicht mit vertretbarem Aufwand realisierbar.

Der Entwicklung deterministischer Methoden zur Bottom-up-Analyse soll deshalb folgende Idee zurgrunde gelegt werden:

Man versucht, Obermengen O_p der Reduktionsklassen R_p , $p \in P \cup \{\text{shift}\}$, so zu finden, daß die Steuerung der Analyse anhand dieser Obermengen statt anhand der Reduktionsklassen durchgeführt werden kann. Bei einem durch ein aktuelles Paar (γ, r) gekennzeichneten Stand der Analyse eines Wortes soll also jeweils in einfacher Weise ermittelt werden können, für welche $p \in P \cup \{\text{shift}\}$ das Paar (γ, r) in O_p liegt. Mit einer entsprechenden Aktion soll die Analyse dann jeweils fortgesetzt werden.

Diese Steuerung anhand der Obermengen ermöglicht eine deterministische Analyse nur dann, wenn ausgeschlossen ist, daß ein aktuelles Paar gleichzeitig zwei Obermengen O_p und $O_{p'}$ für verschiedene Elemente $p, p' \in P \cup \{\text{shift}\}$ angehört, aber nur in einer der zugehörigen Reduktionsklassen R_p und $R_{p'}$ liegt. In diesem Fall könnte die zur korrekten Fortsetzung der Analyse durchzuführende Aktion anhand der Obermengen nicht eindeutig identifiziert und somit die Analyse nicht deterministisch durchgeführt werden. Derartige Fälle sollen ausgeschlossen werden durch die Forderung, daß die Obermengen disjunkt sind, d.h. :

$$\forall p, p' \in P \cup \{\text{shift}\}, p \neq p': O_p \cap O_{p'} = \emptyset.$$

Hier ergibt sich die Frage, unter welchen Umständen zu den Reduktionsklassen R_p Obermengen O_p existieren, welche paarweise disjunkt sind. Der folgende Satz bereitet die Antwort vor.

Satz 4.8 Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik.

$$G \text{ ist eindeutig} \quad \Leftrightarrow \quad \begin{array}{l} \text{i) } \quad \neg(S \xrightarrow{+} S) \\ \text{und} \\ \text{ii) } \quad \forall p, p' \in P \cup \{\text{shift}\}, p \neq p': R_p \cap R_{p'} = \emptyset. \end{array}$$

Beweis:

" \Rightarrow " G eindeutig $\Rightarrow \neg(S \xrightarrow{+} S)$, denn sonst: $S \xrightarrow{+} S \begin{cases} \rightarrow w \\ \xrightarrow{+} S \end{cases}$, Widerspruch zur Eindeutigkeit.

Annahme: $\exists p, p' \in P \cup \{\text{shift}\}, p \neq p', (\alpha, r): (\alpha, r) \in R_p \wedge (\alpha, r) \in R_{p'}$, dann Widerspruch zur Eindeutigkeit.

Insgesamt folgt i) und ii).

" \Leftarrow " Beweis durch Kontraposition:

Sei G ist mehrdeutig (dann muß eine der beiden Bedingungen verletzt sein).

Sei $s \in (N \cup T)^*$ mit $S \xrightarrow{*}_R u \xrightarrow{R} s$ und $S \xrightarrow{*}_R v \xrightarrow{R} s$. s ist eine Satzform mit 2 verschiedenen Rechtsableitungen.

$\Rightarrow s = s_1 s_2$ mit $(s_1, s_2) \in R_p \cap R_{p'}$, also $R_p \cap R_{p'} \neq \emptyset$.

Insgesamt folgt " \Leftarrow ".

q.e.d.

Korollar 4.1 Für eine beliebige kontextfreie Grammatik ist nicht entscheidbar, ob die zu verschiedenen Elementen $p, p' \in P \cup \{\text{shift}\}$ gehörenden Reduktionsklassen R_p und $R_{p'}$ disjunkt sind.

Bemerkung:

Sei $G = (N, T, P, S)$ eine reduzierte kontextfreie Grammatik, deren Reduktionsklassen R_p , $p \in P \cup \{\text{shift}\}$, paarweise disjunkt sind. Dann ist das Bestehen einer Beziehung $A \xrightarrow{+} A$ für ein $A \in N$ nicht ausgeschlossen. Doch folgt aus $A \xrightarrow{+} A$ für ein $A \neq S$ stets auch $S \xrightarrow{+} S$. Liegt ein solcher Fall vor, dann ist die Grammatik G zwar mehrdeutig, aber die von ihr erzeugte Sprache $L(G)$ ist eindeutig. Denn:

Aus der Disjunktheit der Reduktionsklassen folgt, daß jede Ableitung eines Wortes w , in der ein Zyklus $A \xrightarrow{+} A$ oder $S \xrightarrow{+} S$ auftritt, die Form:

$$S \xrightarrow{+} A \xrightarrow{+} S \xrightarrow{+} A \xrightarrow{+} w \quad \text{oder} \quad S \xrightarrow{+} S \xrightarrow{+} S \xrightarrow{+} w$$

hat. Weiter gilt, daß man aus G eine äquivalente eindeutige Grammatik bereits durch Streichen aller Produktionen erhält, die das Startsymbol S auf der rechten Seite enthalten.

Ein einfaches Beispiel liefert die Grammatik:

$$G = (\{S, A\}, \{a\}, \{S ::= A, A ::= S, S ::= a\}, S) \text{ mit } L(G) = \{a\}.$$

Kommen wir zurück auf die Frage nach der Existenz disjunkter Obermengen O_p ; Bedingung (ii) des Satzes ist offensichtlich erfüllt, wenn Obermengen $O_p \supseteq R_p$, $p \in P \cup \{\text{shift}\}$, existieren mit:

$$O_p \cap O_{p'} = \emptyset \text{ für } p, p' \in P \cup \{\text{shift}\}, p \neq p'.$$

Wir erhalten damit das folgende Korollar:

Korollar 4.2 Sei $G=(N,T,P,S)$ eine *reduzierte kontextfreie Grammatik*.

G ist *eindeutig* \Leftrightarrow

- i) $\neg(S^+ \rightarrow S)$
- und
- ii) *es gibt Obermengen $O_p \supseteq R_p$, $p \in P \cup \{shift\}$, derart, daß $\forall p, p' \in P \cup \{shift\}, p \neq p': O_p \cap O_{p'} = \emptyset$.*

Dieses Korollar zeigt, daß eine deterministische Bottom-up-Analyse auf der Grundlage disjunkter Obermengen der Reduktionsklassen nur für **eindeutige** kontextfreie Grammatiken möglich ist.

Algorithmus: Deterministische Bottom-up-Analyse

Eingabe: - reduzierte kontextfreie Grammatik, die Regeln sind mit 1 bis $|P|$ numeriert
 - disjunkte Obermengen O_p derart, daß für jedes Paar (γ, r) und für jedes p entscheidbar ist, ob $(\gamma, r) \in O_p$
 - $w \in T^*$

Ausgabe: - $w \in L(G) \Rightarrow$ Reduktionsschritte
 - $w \notin L(G) \Rightarrow \begin{cases} \text{Endlosschleife} \\ \text{Fehlermeldung} \end{cases}$

var ap, ap_{alt} ; (* ap = aktuelles Paar *)

begin

$ap := (\epsilon, w)$;

repeat

$ap_{alt} := ap$;

if " $ap \in O_{\text{shift}} \wedge ap = (x, tr)$ für ein $x \in (N \cup T)^*$, $t \in T$, $r \in T^*$ " then

$ap := (xt, r)$ fi;

if "P enthält ein $A ::= \alpha$ mit: $ap \in O_{A ::= \alpha} \wedge ap = (x\alpha, r)$ " then

begin

"Ausgabe der Markierung von $A ::= \alpha$ ";

$ap := (xA, r)$

end

fi

until $(ap = (S, \epsilon))$ or $(ap = ap_{alt})$;

if $ap = (S, \epsilon)$ then

$w \in L(G)$

else

Fehlermeldung:

$ap = (\gamma, r)$, wobei ap in keiner Obermenge liegt

oder

$ap = (\gamma, r) \in O_p$, aber die entsprechenden Operationen sind nicht ausführbar, d.h.

entweder ($p = A ::= \alpha$ und γ hat α nicht als Suffix) oder ($p = \text{shift}$ und $r = \epsilon$)

fi

end.

Satz 4.9 Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik, und seien O_p , $p \in P \cup \{\text{shift}\}$, disjunkte Obermengen der Reduktionsklassen R_p mit folgenden Eigenschaften:

- 1) $\neg(S \xrightarrow{+} S)$
- 2) Für jedes $p \in P \cup \{\text{shift}\}$ und jedes Paar (γ, r) mit $\gamma \in (N \cup T)^*$ und $r \in T^*$ ist entscheidbar, ob $(\gamma, r) \in O_p$ oder $(\gamma, r) \notin O_p$.

Sei $w \in T^*$, dann gilt:

- a) $\forall w \in L(G)$ liefert der Algorithmus eine Rechtsreduktion von w .
- b) $\forall w \notin L(G)$ hält der Algorithmus entweder mit Fehlermeldung an oder er hält überhaupt nicht an.

Beispiel

Sei $G=(\{S,A,C\}, \{a,b\}, P, S)$ mit: $P=\{S::=A \mid C, A::=ab, C::=\epsilon\}$. Es ist $L(G)=\{ab, \epsilon\}$.

Reduktionsklassen

$$R_{S::=A} = \{(A, \epsilon)\}$$

$$R_{S::=C} = \{(C, \epsilon)\}$$

$$R_{A::=ab} = \{(ab, \epsilon)\}$$

$$R_{C::=\epsilon} = \{(\epsilon, \epsilon)\}$$

$$R_{\text{Shift}} = \{(\epsilon, ab), (a, b)\}$$

Obermengen

$$O_p = R_p \text{ für } p \in \{S::=A, S::=C, A::=ab\}$$

$$O_{C::=\epsilon} = R_{C::=\epsilon} \cup \{(bC^n, \epsilon) \mid n \geq 0\}$$

$$O_{\text{Shift}} = R_{\text{Shift}} \cup \{(\epsilon, b)\}$$

Sei $w:=b$.

Dann: $(\epsilon, b) \vdash (b, \epsilon) \vdash (bC, \epsilon) \vdash (bC^2, \epsilon) \vdash \dots$

\Rightarrow Algorithmus terminiert nicht (wegen ϵ -Produktion)

$\Rightarrow w \notin L(G)$.

Bemerkung:

Eigenschaft b) ist recht unbefriedigend. Deshalb ist man bemüht, zu Obermengen überzugehen, bei denen der Algorithmus auf alle Fälle anhält. Genau diese Eigenschaft erfüllen die LR(k)-Obermengen.

4.3.2.2 LR(k)-Analyse

Definition 4.11 Sei $x \in (N \cup T)^*$, $k \in \mathbf{N}$.

$$k(x) := \begin{cases} x_1 & , \text{ falls } x = x_1 x_2 \text{ und } |x_1| = k \text{ und } x_2 \in (N \cup T)^* \\ x & , \text{ falls } |x| \leq k \end{cases}$$

heißt *k-Anfang* von x ("look-ahead").

Definition 4.12 Sei $G = (N, T, P, S)$ eine reduzierte kontextfreie Grammatik und $k \in \mathbf{N}$. Zu jedem $p \in P \cup \{\text{shift}\}$ definiert man als *LR(k)-Obermenge* der Reduktionsklassen R_p die Menge

$$O_p^k := \{(\gamma, y) \mid \gamma \in (N \cup T)^* \wedge y \in T^* \wedge \exists (\gamma, r) \in R_p : k(y) = k(r)\}.$$

Definition 4.13 Sei $G = (N, T, P, S)$ eine reduzierte kontextfreie Grammatik. G heißt *LR(k)-Grammatik*, falls gilt:

- i) $\neg(S \xrightarrow{+} S)$
und
- ii) $p, q \in P \cup \{\text{shift}\}, p \neq q \Rightarrow O_p^k \cap O_q^k = \emptyset$.

Bemerkung:

Zwischenstufen zwischen LR(0) und LR(1):

SLR: simple-LR

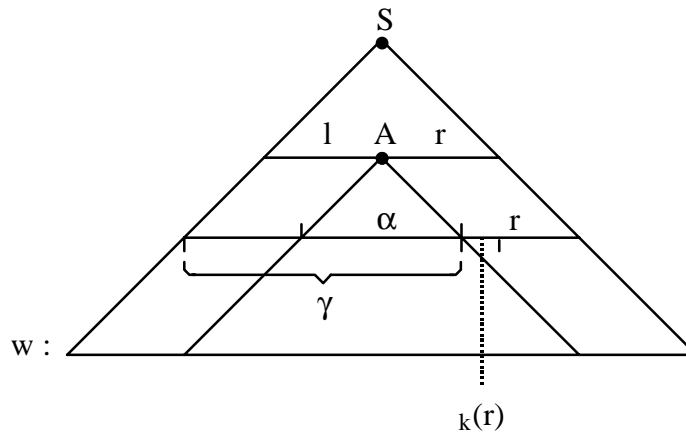
und

LALR: look-ahead LR

Definition 4.14 Eine Sprache heißt *LR(k)-Sprache*, wenn sie von einer LR(k)-Grammatik erzeugt wird.

Eine *LR(k)-Analyse* ist die deterministische Bottom-up-Analyse (gemäß Algorithmus) unter Zugrundelegung der LR(k)-Obermengen.

Strukturbaum τ zum Analysewort w :



Dabei bedeutet: γ : gekellte Information
 r : noch nicht gelesener Teil der Eingabe .

Zu jedem Zeitpunkt benötigt man zur Bestimmung der nächsten Aktion γ (= Vorgeschichte der bisherigen Aktionen) und den look-ahead der ersten k Symbole der noch nicht verarbeiteten Eingabe.

Bemerkung:

Für $LR(k)$ -Obermengen hält der Algorithmus für $w \notin L(G)$ immer mit Fehlermeldung an.

Satz 4.10 Sei G eine $LR(k)$ -Grammatik, $k \in \mathbb{N}$. Dann ist G eindeutig.

Beweis:

G ist $LR(k)$ -Grammatik

$$\Leftrightarrow \left\{ \begin{array}{l} \neg(S \rightarrow S) \\ \text{und} \\ p, p' \in P \cup \{\text{shift}\}, p \neq p' : O_p^k \cap O_{p'}^k = \emptyset \\ \left. \begin{array}{l} O_p^k \supseteq R_p \\ O_{p'}^k \supseteq R_{p'} \end{array} \right\} \Rightarrow R_p \cap R_{p'} = \emptyset \Rightarrow G \text{ eindeutig} \end{array} \right.$$

q.e.d.

Satz 4.11 Sei G eine reduzierte kontextfreie Grammatik, $k \in \mathbf{N}$.

G ist LR(k)-Grammatik \Leftrightarrow

i) $\neg(S \xrightarrow{+} S)$

und

$$\text{ii) } \left\{ \begin{array}{l} S \xrightarrow{\dagger} lAr \quad R \rightarrow l\alpha r \\ S \xrightarrow{\dagger} \bar{l} \bar{A} \bar{r} \quad R \rightarrow \bar{l} \bar{\alpha} \bar{r} = l\alpha r \\ k(r) = k(\tilde{r}) \end{array} \right\} \Rightarrow l = \bar{l} \wedge A = \bar{A} \wedge \alpha = \bar{\alpha} .$$

Bemerkung:

Dieser Satz wird auch zur Definition von LR(k)-Grammatiken herangezogen.

Satz 4.14 Für jede kontextfreie Grammatik $G=(N,T,P,S)$ und jedes $k \in \mathbf{N}$ ist entscheidbar, ob G eine LR(k)-Grammatik ist.

Bemerkung:

Für eine beliebige kontextfreie Grammatik $G=(N,T,P,S)$ ist nicht entscheidbar, ob es ein $k \in \mathbf{N}$ gibt mit: G ist LR(k)-Grammatik.

Jede Aktion ist abhängig von γ und $k(y)$; $k(y)$ ist beschränkt, γ im allgemeinen nicht. Aufgabe ist es also, die beschränkten Informationen herauszufinden, die zusammen mit $k(y)$ die nächste Aktion bestimmen:

Definition 4.15 Sei $G=(N,T,P,S)$ eine kontextfreie Grammatik, $\gamma \in (N \cup T)^*$, $k \in \mathbf{N}$

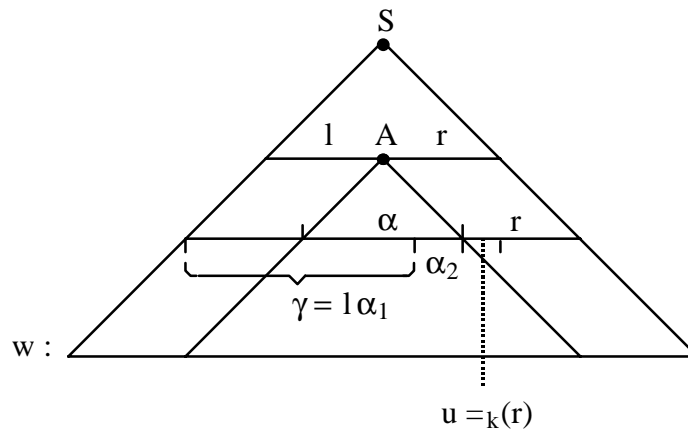
a) $[A := \alpha_1 \alpha_2, u]$ heißt LR(k)-Auskunft zu γ , wenn:

$$\exists A := \alpha_1 \alpha_2 \in P: \exists l \in (N \cup T)^*, r \in T^* :$$

$$S \xrightarrow{\dagger} lAr \quad R \rightarrow l\alpha_1\alpha_2 r, \text{ mit } \gamma = l\alpha_1 \wedge u = k(r).$$

b) Die Menge aller LR(k)-Auskünfte zu γ bezeichnet man als LR(k)-Information zu γ (Schreibweise: $I_k(\gamma)$).

c) Als kanonische Kollektion von LR(k)-Informationen zu γ bezeichnet man die Menge $J_k := \{I_k(\gamma) \neq \emptyset \mid \gamma \in (N \cup T)^*\}$.



Eine LR(k)-Information gibt darüber Auskunft, welche Reduktionen am rechten Rand von γ möglich sind bzw. nach eventuellem Shiften von Eingabesymbolen an das rechte Ende von γ möglich werden können und wie diese Reduktion vom look-ahead abhängt.

Satz 4.12 Für jedes $\gamma \in (N \cup T)^*$ ist $I_k(\gamma)$ eine endliche Menge. Die kanonische Kollektion J_k ist eine endliche Menge.

Definition 4.16 Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik, $k \in \mathbf{N}$.

Für jedes $\gamma \in (N \cup T)^*$ sei $L(\gamma) := \{ w \in T^* \mid \gamma \xrightarrow{*} w \}$. $L_k(\gamma) := {}_k(L(\gamma))$ heißt k -Anfang zu γ (First $_k(\gamma)$).

Für jedes $A \in N$ sei $F(A) := \{ w \in T^* \mid \exists l \in (N \cup T)^* : S \xrightarrow{*} lAw \}$. $F_k(A) := {}_k(F(A))$ heißt k -Folgemenge von A (Follow $_k(A)$).

Satz 4.13 (Hauptsatz über LR(k)-Grammatiken) Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik, $k \in \mathbf{N}$

Dann gilt für jedes Paar (γ, y) mit $\gamma \in (N \cup T)^*$ und $y \in T^*$:

$$a) \quad (\gamma, y) \in O_{A::=a}^k \Leftrightarrow [A::=\alpha, \varepsilon, {}_k(y)] \in I_k(\gamma)$$

$$b) \quad (\gamma, y) \in O_{\text{Shift}}^k \Leftrightarrow \exists [A::=\alpha_1, \alpha_2, v] \in I_k(\gamma) : v \in {}_k(T^*) \wedge {}_1(\alpha_2) \in T \wedge {}_k(y) \in L_k(\alpha_2 v).$$

Bemerkung:

Der Satz besagt, daß für jedes aktuelle Paar (γ, y) und $p \in P \cup \{\text{shift}\}$ die Zugehörigkeit von (γ, y) zu O_p^k bereits durch p , die LR(k)-Information $I_k(\gamma)$ und $k(y)$ bestimmt ist.

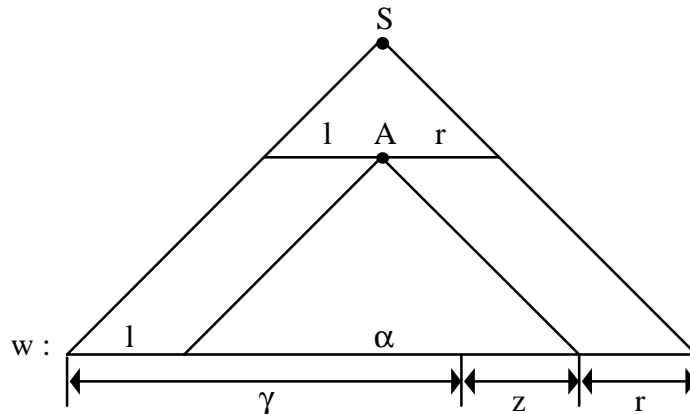
Beweis:

$$\begin{aligned}
 \text{a) } (\gamma, y) \in O_{A::=a}^k & \stackrel{\text{Definition}}{\Leftrightarrow} \exists (\gamma, r) \in R_{A::=\alpha} : k(r) = k(y) \\
 & \stackrel{\text{Definition}}{\Leftrightarrow} \exists (l\alpha, r) : l \in (N \cup T)^* \wedge r \in T^* \wedge S \xrightarrow{*} lAr \rightarrow l\alpha r \\
 & \stackrel{\text{Definition}}{\Leftrightarrow} [A::=\alpha, \varepsilon, k(y)] \in I_k(\gamma)
 \end{aligned}$$

b) " \Leftarrow " Sei $[A::=\alpha_1.\alpha_2, v] \in I_k(\gamma)$ mit obigen Eigenschaften
 $\Rightarrow \exists l \in (N \cup T)^*, r \in T^* : S \xrightarrow{*} lAr \rightarrow l\alpha_1\alpha_2 r = \gamma\alpha_2 r$ und $v = k(r)$
 \Rightarrow Ein Paar (γ, y) mit $k(y) \in L_k(\alpha_2 r) = L_k(\alpha_2 v)$ ist aus O_{shift}^k .

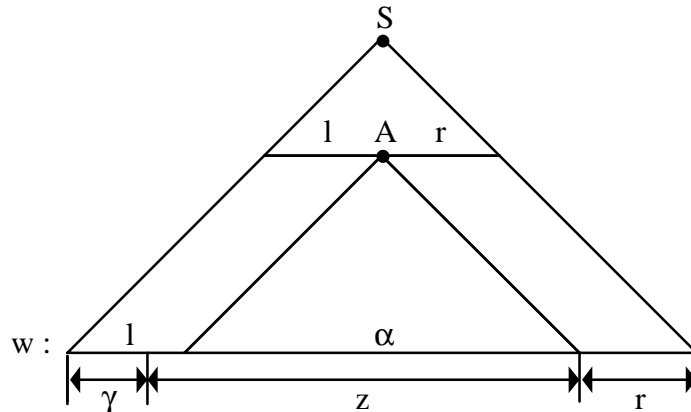
" \Rightarrow " Sei $(\gamma, y) \in O_{\text{shift}}^k \Rightarrow \exists r \in T^*, l \in (N \cup T)^*, z \in T^+, A::=\alpha \in P : S \xrightarrow{*} lAr \xrightarrow{R} l\alpha r = \gamma z r$ mit $k(y) = k(zr)$.

1. Fall:



z ist Suffix von α . Wähle als Zerlegung von α : $\alpha = \alpha_1.\alpha_2$ mit $\alpha_2 = z$
 $\Rightarrow [A::=\alpha_1.\alpha_2, k(r)] \in I_k(\gamma)$
 $z \in T^+ \Rightarrow l(\alpha_2) \in T$ und außerdem $L_k(\alpha_2 k(r)) = \{k(zr)\} = \{k(y)\}$.

2. Fall:



α ist echtes Suffix von z

$\Rightarrow \gamma$ ist echtes Präfix von l

$\Rightarrow l \neq \epsilon$ und $S \xrightarrow{R} lAr, lAr \xrightarrow{R} S$

$\Rightarrow \exists n \geq 1: \exists w_i \in (N \cup T)^*, 1 \leq i \leq n: S = w_0 \xrightarrow{R} w_1 \xrightarrow{R} \dots \xrightarrow{R} w_n = lAr \xrightarrow{R} l\alpha r.$

Seien $l_i \in (N \cup T)^*, A_i \in N, r_i \in T^*$ derart, daß $w_i = l_i A_i r_i$ für $1 \leq i \leq n$. Sei nun j die kleinste Zahl derart, daß γ echtes Präfix von l_i ist für alle i mit $j \leq i \leq n$. Dieses j existiert, da zumindest die Zeichenfolge $l_n = l \gamma$ als echtes Präfix enthält. Offensichtlich gilt $1 \leq j \leq n$.

\Rightarrow Jedes $w_i, j \leq i \leq n$, ist darstellbar durch: $w_i = \gamma \delta_i A_i r_i = l_i A_i r_i$ für ein $\delta_i \in (N \cup T)^+, A_i \in N$ und $r_i \in T^*$.

Betrachte nun:

$$S \xrightarrow{R} w_{j-1} = l'A'r' \xrightarrow{R} w_j = l'\alpha'r' = \gamma\delta_j A_j r_j \xrightarrow{R} lAr \xrightarrow{R} l\alpha r = \gamma z r$$

$$\Rightarrow |l'| \leq |\gamma|.$$

Da $|l'\alpha'| > |\gamma\delta_j| > |\gamma|$, existiert eine Zerlegung von α' in $\alpha' = \alpha'_1 \alpha'_2$ mit $l'\alpha'_1 = \gamma$ und $|\alpha'_2| > 0$

$$\Rightarrow [A'::=\alpha'_1 \cdot \alpha'_2, k(r')] \in I_k(\gamma) \text{ und es gilt: } k(y) = k(zr) \in L_k(\alpha'_2 k(r')).$$

Bleibt zu zeigen: ${}_1(\alpha'_1) \in T$. Dazu betrachten wir:

$$w_j = \gamma\alpha'_2 r' = \gamma\delta_j A_j r_j \xrightarrow{R} w_{j+1} = \gamma\delta_{j+1} A_{j+1} r_{j+1}$$

$$\rightarrow \dots \rightarrow w_n = \gamma\delta_n Ar \xrightarrow{R} \gamma z r$$

$$\forall j \leq i \leq n: |\delta_i| > 0 \Rightarrow {}_1(\alpha'_2) = {}_1(\delta_j) = \dots = {}_1(\delta_n) = {}_1(z) \in T.$$

q.e.d.

Definition 4.17 Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik, $k \in \mathbf{N}$. Eine Menge J von LR(k)-Informationen heißt *konsistent*, wenn für jedes I aus J gilt:

$$[A::=\alpha.\varepsilon,u], [A'::=\alpha'_1.\alpha'_2,v] \in I \text{ mit } {}_1(\alpha'_2) \notin N \text{ und } u \in L_k(\alpha'_2 v) \\ \Rightarrow [A::=\alpha.\varepsilon,u] = [A'::=\alpha'_1.\alpha'_2,v].$$

Bemerkung:

1. ${}_1(\alpha'_2) \notin N$ bedeutet: ${}_1(\alpha'_2) \in T$ oder ${}_1(\alpha'_2) = \varepsilon$, d.h. $\alpha'_2 = \varepsilon$.
2. Konsistenz von J heißt: Kein I in J enthält verschiedene LR(k)-Auskünfte der Form:
 $[A::=\alpha.\varepsilon,u]$ und $[A'::=\alpha'.\varepsilon,v]$ mit $u=v$
 oder
 $[A::=\alpha.\varepsilon,u]$ und $[A'::=\alpha'_1.\alpha'_2,v]$ mit ${}_1(\alpha'_2) \in T$ und $u \in L_k(\alpha'_2 v)$.

Aus dem Hauptsatz folgt: Für jedes (γ,y) gilt entweder $(\gamma,y) \in O_{A::=a}^k$
 oder $(\gamma,y) \in O_{\text{Shift}}^k$.

Korollar 4.3 $G=(N,T,P,S)$ ist LR(k)-Grammatik \Leftrightarrow

- i) $\neg(S^+ \rightarrow S)$
 und
- ii) die kanonische Kollektion J_k ist konsistent.

Daraus ergibt sich ein Entscheidungsverfahren, ob für gegebenes k die Grammatik G eine LR(k)-Grammatik ist:

1. Überprüfung, ob $\neg(S^+ \rightarrow S)$.
2. Bestimmung von J_k zu G .
3. Überprüfung der Konsistenz von J_k .

1. Schritt: Überprüfung, ob $\neg(S \xrightarrow{+} S)$.

a) **Algorithmus: "ε-Ableitungen"**

Eingabe: kontextfreie Grammatik $G=(N,T,P,S)$

Ausgabe: Menge aller $A \in N$ mit: $A^* \rightarrow \epsilon$

```
begin
  E:={A | A::=ε ∈ P}
  while " ∃ A::=B1 ...Bm ∈ P : m≥1 ∧ A∉E ∧ ∨ 1≤i≤m : Bi ∈ E " do
    E := E ∪ {A}
  end.
```

b) **Algorithmus: "zyklisches Axiom"**

Eingabe: - kontextfreie Grammatik $G=(N,T,P,S)$,

- $E=\{A \in N \mid A^* \rightarrow \epsilon\}$,

- S

Ausgabe: "S zyklisch" oder "S nicht zyklisch"

```
begin
  var W, Walt ;
  W:=∅;
  repeat
    Walt :=W;
    W := W ∪ {C ∈ N | ∃ B::=βCγ : B=S ∨ B ∈ W,
      β,γ ∈ (N ∪ T)*,
      β* → ε ∧ γ* → ε}
  until W=Walt ;
  if S ∈ W then
    write("S zyklisch")
  else
    write("S nicht zyklisch")
  end.
```

Mit den Algorithmen a) und b) ist entscheidbar, ob: $\neg(S \xrightarrow{+} S)$.

2. Schritt: Berechnung der LR(k)-Informationen und der kanonischen Kollektion J_k

a) Algorithmus: Berechnung von $I_k(\gamma)$

Man ermittelt $I_k(\gamma)$ zunächst für den leeren Keller $\gamma=\varepsilon$ und dann induktiv für Keller wachsender Länge.

Eingabe: - reduzierte kontextfreie Grammatik $G=(N,T,P,S)$,
 - $\gamma \in (N \cup T)^*$,
 - $k \in \mathbf{N}$ fest

Ausgabe: $I_k(\gamma)$

(1) Berechnung von $I_k(\varepsilon)$:

$I_k(\varepsilon) := \{ [S::=\varepsilon.\delta,\varepsilon] \mid S::=\delta \in P \};$

repeat

$z := I_k(\varepsilon);$

 for " jedes $A, B \in N, \beta, \delta \in (N \cup T)^*, u \in_k(T^*)$ " do

$I_k(\varepsilon) := I_k(\varepsilon) \cup \{ [B::=\varepsilon.\beta,v] \mid [A::=\varepsilon.B\delta,u] \in I_k(\varepsilon),$

$B::=\beta \in P,$

$v \in L_k(\delta u) \}$

until $z = I_k(\varepsilon);$

(2) Induktionsschritt zur Berechnung von $I_k(\overline{\gamma X})$ aus gegebenem $I_k(\overline{\gamma})$ für ein $\overline{\gamma} \in (N \cup T)^*, X \in (N \cup T), \overline{\gamma X}$ Präfix von $\overline{\gamma}$:

$I_k(\overline{\gamma X}) := \{ [A::=\alpha X.\delta,u] \mid A \in N, \alpha, \delta \in (N \cup T)^*, u \in_k(T^*) \text{ mit:}$

$[A::=\alpha.X\delta,u] \in I_k(\overline{\gamma}) \};$

repeat

$z := I_k(\overline{\gamma X});$

 for " jedes $A, B \in N, \alpha, \beta, \delta \in (N \cup T)^*, u \in_k(T^*)$ " do

$I_k(\overline{\gamma X}) := I_k(\overline{\gamma X}) \cup \{ [B::=\varepsilon.\beta,v] \mid [A::=\alpha.B\delta,u] \in I_k(\overline{\gamma X}),$

$B::=\beta \in P,$

$v \in L_k(\delta u) \}$

until $z = I_k(\overline{\gamma X});$

b) **Algorithmus: Berechnung der kanonischen Kollektion J_k**

1. Man bestimmt $I_k(\epsilon)$.

$$I_k(\epsilon) \neq \emptyset \Rightarrow J_k = I_k(\epsilon)$$

2. Hüllenbildung

Zu jedem $I_k(\gamma) \in J_k$ und $X \in (N \cup T)$ bestimmt man $I_k(\gamma X)$.

$$I_k(\gamma X) \neq \emptyset \Rightarrow J_k := J_{k_{\text{alt}}} \cup I_k(\gamma X)$$

Eingabe: - reduzierte kontextfreie Grammatik $G=(N,T,P,S)$,
- $k \in \mathbf{N}$

Ausgabe: J_k

(1) $J_k := \emptyset$;

"Berechnung von $I_k(\epsilon)$ ";

if $I_k(\epsilon) \neq \emptyset$ then

$$J_k := I_k(\epsilon)$$

(* $I_k(\epsilon)$ noch unmarkiert *)

(2) repeat

if " J_k enthält eine unmarkierte Information $I_k(\gamma)$ " then

begin

for "jedes $X \in (N \cup T)$ " do

begin

"Berechnung von $I_k(\gamma X)$ ";

if $(I_k(\gamma X) \neq \emptyset)$ and $(I_k(\gamma X) \notin J_k)$ then

$$J_k := J_k \cup I_k(\gamma X)$$

end;

"markiere die Information $I_k(\gamma)$ in J_k "

end

until "alle LR(k)-Informationen sind markiert";

Beispiel

Sei $G = (\{S\}, \{a, b\}, P, S)$ mit $P = \{S ::= SaSb, S ::= \epsilon\}$.

Behauptung: G ist LR(1)-Grammatik.

Beweis:

1. Schritt: Offenbar: $\neg(S \xrightarrow{+} S)$.

2. Schritt: a) Berechnung der LR(k)-Informationen

Berechnung von $I_1(\epsilon)$:

Initialisierung: $I_1(\epsilon) := \{[S ::= \epsilon.SaSb, \epsilon], [S ::= \epsilon.\epsilon, \epsilon]\}$

Repeatschleife:

1. Durchlauf: $I_1(\epsilon) := I_1(\epsilon) \cup \{[S ::= \epsilon.SaSb, a], [S ::= \epsilon.\epsilon, a]\}$, da $L_1(aSb) = \{a\}$

2. Durchlauf: keine Veränderung

$\Rightarrow I_1(\epsilon) = \{[S ::= \epsilon.SaSb, \epsilon], [S ::= \epsilon.\epsilon, \epsilon], [S ::= \epsilon.SaSb, a], [S ::= \epsilon.\epsilon, a]\}$

Berechnung von $I_1(\epsilon X)$ für alle $X \in (N \cup T) = \{S, a, b\}$:

Berechnung von $I_1(S)$:

Initialisierung: $I_1(S) := \{[S ::= S.aSb, \epsilon], [S ::= S.aSb, a]\}$

Repeatschleife: keine Veränderung

Berechnung von $I_1(a)$:

Initialisierung: $I_1(a) := \emptyset$

Repeatschleife: keine Veränderung

Berechnung von $I_1(b)$:

Initialisierung: $I_1(b) := \emptyset$

Repeatschleife: keine Veränderung

Berechnung von $I_1(\gamma X)$ für alle $X \in \{S, a, b\}$, $\gamma = S$:

(Bemerkung: Die Berechnungen für $\gamma = a$, $\gamma = b$ entfallen, da für diese γ gilt: $I_1(\gamma) = \emptyset$)

Berechnung von $I_1(SS)$:

Initialisierung: $I_1(SS) := \emptyset$
 Repeatschleife: keine Veränderung

Berechnung von $I_1(Sa)$:

Initialisierung: $I_1(Sa) := \{[S ::= Sa.Sb, \epsilon], [S ::= Sa.Sb, a]\}$
 Repeatschleife:
 1. Durchlauf: $I_1(Sa) := I_1(Sa) \cup \{[S ::= \epsilon.SaSb, b], [S ::= \epsilon.\epsilon, b]\}$,
 da $L_1(b) = \{b\}$, $L_1(ba) = \{b\}$
 2. Durchlauf: $I_1(Sa) := I_1(Sa) \cup \{[S ::= \epsilon.SaSb, a], [S ::= \epsilon.\epsilon, a]\}$
 3. Durchlauf: keine Veränderung
 $\Rightarrow I_1(Sa) = \{ [S ::= Sa.Sb, \epsilon], [S ::= Sa.Sb, a], [S ::= \epsilon.SaSb, b],$
 $[S ::= \epsilon.\epsilon, b], [S ::= \epsilon.SaSb, a], [S ::= \epsilon.\epsilon, a] \}$

Berechnung von $I_1(Sb)$:

Initialisierung: $I_1(Sb) := \emptyset$
 Repeatschleife: keine Veränderung

Berechnung von $I_1(\gamma X)$ für alle $X \in \{S, a, b\}$, $\gamma = Sa$:

Berechnung von $I_1(SaS)$:

Initialisierung: $I_1(SaS) := \{ [S ::= SaS.b, \epsilon], [S ::= SaS.b, a],$
 $[S ::= S.aSb, a], [S ::= S.aSb, b] \}$
 Repeatschleife: keine Veränderung

Berechnung von $I_1(Saa)$:

$I_1(Saa) := \emptyset$

Berechnung von $I_1(Sab)$:

$I_1(Sab) := \emptyset$

Berechnung von $I_1(\gamma X)$ für alle $X \in \{S, a, b\}$, $\gamma = SaS$:

Berechnung von $I_1(SaSS)$:

$$I_1(SaSS) := \emptyset$$

Berechnung von $I_1(SaSa)$:

Initialisierung: $I_1(SaSa) := \{[S::=Sa.Sb,a],[S::=Sa.Sb,b]\}$

Repeatschleife:

1. Durchlauf: $I_1(SaSa) := I_1(SaSa) \cup \{[S::=\epsilon.SaSb,b],[S::=\epsilon.\epsilon,b]\}$

2. Durchlauf: $I_1(SaSa) := I_1(SaSa) \cup \{[S::=\epsilon.SaSb,a],[S::=\epsilon.\epsilon,a]\}$

3. Durchlauf: keine Veränderung

$$\Rightarrow I_1(SaSa) = \{ [S::=Sa.Sb,a],[S::=\epsilon.SaSb,b],[S::=\epsilon.\epsilon,b],[S::=\epsilon.SaSb,a], \\ [S::=\epsilon.\epsilon,a],[S::=Sa.Sb,b] \}$$

Berechnung von $I_1(SaSb)$:

Initialisierung: $I_1(SaSb) := \{[S::=SaSb.\epsilon,\epsilon],[S::=SaSb.\epsilon,a]\}$

Repeatschleife: keine Veränderung

Berechnung von $I_1(\gamma X)$ für alle $X \in \{S, a, b\}$, $\gamma = SaSa \wedge \gamma = SaSb$:

Berechnung von $I_1(SaSaS)$:

Initialisierung: $I_1(SaSaS) := \{ [S::=S.aSb,a],[S::=S.aSb,b], \\ [S::=SaS.b,a],[S::=SaS.b,b] \}$

Repeatschleife: keine Veränderung

Berechnung von $I_1(SaSaa)$:

$$I_1(SaSaa) := \emptyset$$

Berechnung von $I_1(SaSab)$:

$$I_1(SaSab) := \emptyset$$

Berechnung von $I_1(SaSbS)$:

$$I_1(SaSbS) := \emptyset$$

Berechnung von $I_1(SaSba)$:

$$I_1(SaSba) := \emptyset$$

Berechnung von $I_1(\text{SaSbb})$:

$$I_1(\text{SaSbS}) := \emptyset$$

Berechnung von $I_1(\gamma X)$ für alle $X \in \{S, a, b\}$, $\gamma = \text{SaSaS}$:

Berechnung von $I_1(\text{SaSaSS})$:

$$I_1(\text{SaSaSS}) := \emptyset$$

Berechnung von $I_1(\text{SaSaSa})$:

$$I_1(\text{SaSaSa}) := \{[S::=\text{Sa.Sb,a}], [S::=\text{Sa.Sb,b}]\}$$

Repeatschleife:

$$1. \text{ Durchlauf: } I_1(\text{SaSaSa}) := I_1(\text{SaSaSa}) \cup \{[S::=\epsilon.\text{SaSb,b}], [S::=\epsilon.\epsilon,b]\}$$

$$2. \text{ Durchlauf: } I_1(\text{SaSaSa}) := I_1(\text{SaSaSa}) \cup \{[S::=\epsilon.\text{SaSb,a}], [S::=\epsilon.\epsilon,a]\}$$

$$3. \text{ Durchlauf: } \text{keine Veränderung}$$

$$\Rightarrow I_1(\text{SaSaSa}) = \{ [S::=\text{Sa.Sb,a}], [S::=\text{Sa.Sb,b}], [S::=\epsilon.\text{SaSb,a}], \\ [S::=\epsilon.\epsilon,a], [S::=\epsilon.\text{SaSb,b}], [S::=\epsilon.\epsilon,b] \}$$

Berechnung von $I_1(\text{SaSaSb})$:

$$\text{Initialisierung: } I_1(\text{SaSaSb}) := \{[S::=\text{SaSb.\epsilon,a}], [S::=\text{SaSb.\epsilon,b}]\}$$

$$\text{Repeatschleife: } \text{keine Veränderung}$$

Die von der leeren Menge verschiedenen LR(k)-Informationen lauten:

$$I_1(\epsilon) = \{ [S::=\epsilon.\text{SaSb,\epsilon}], [S::=\epsilon.\epsilon,\epsilon], [S::=\epsilon.\text{SaSb,a}], [S::=\epsilon.\epsilon,a] \}$$

$$I_1(S) = \{ [S::=\text{S.aSb,\epsilon}], [S::=\text{S.aSb,a}] \}$$

$$I_1(\text{Sa}) = \{ [S::=\text{Sa.Sb,\epsilon}], [S::=\text{Sa.Sb,a}], [S::=\epsilon.\text{SaSb,b}], [S::=\epsilon.\epsilon,b], \\ [S::=\epsilon.\text{SaSb,a}], [S::=\epsilon.\epsilon,a] \}$$

$$I_1(\text{SaS}) = \{ [S::=\text{SaS.b,\epsilon}], [S::=\text{SaS.b,a}], [S::=\text{S.aSb,a}], [S::=\text{S.aSb,b}] \}$$

$$I_1(\text{SaSa}) = \{ [S::=\text{Sa.Sb,a}], [S::=\text{Sa.Sb,b}], [S::=\epsilon.\text{SaSb,b}], [S::=\epsilon.\epsilon,b], \\ [S::=\epsilon.\text{SaSb,a}], [S::=\epsilon.\epsilon,a] \}$$

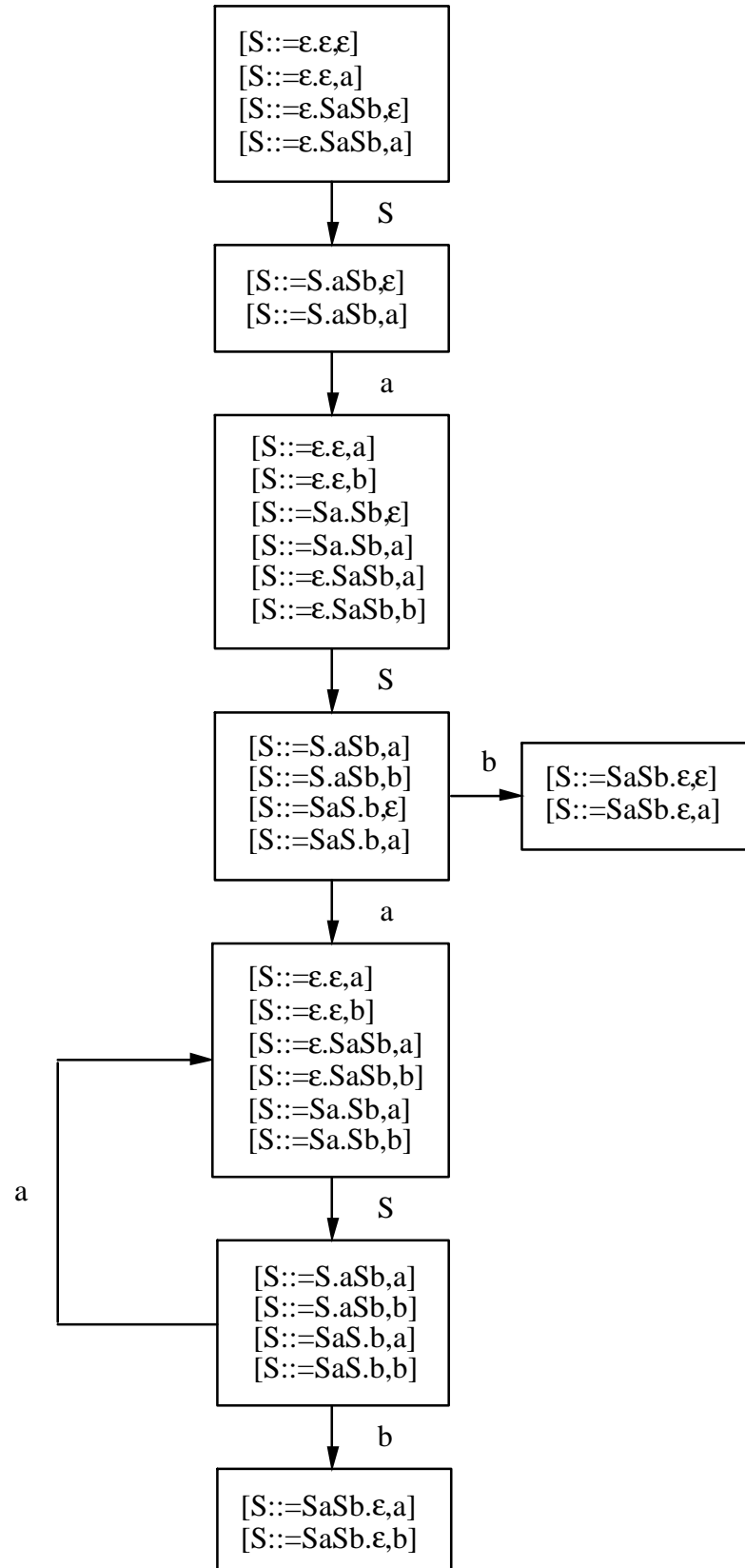
$$I_1(\text{SaSb}) = \{ [S::=\text{SaSb.\epsilon,a}], [S::=\text{SaSb.\epsilon,\epsilon}] \}$$

$$I_1(\text{SaSaS}) = \{ [S::=\text{SaS.b,a}], [S::=\text{SaS.b,b}], [S::=\text{S.aSb,a}], [S::=\text{S.aSb,b}] \}$$

$$I_1(\text{SaSaSa}) = I_1(\text{SaSa})$$

$$I_1(\text{SaSaSb}) = \{ [S::=\text{SaSb.\epsilon,a}], [S::=\text{SaSb.\epsilon,b}] \}$$

Das Ergebnis kann man auch graphisch darstellen mittels eines **Übergangsgraphen**:



2. Schritt: b) Berechnung der kanonischen Kollektion:

$$J_1 := I_1(\epsilon)$$

Repeat-Schleife;

1. Durchlauf: $J_1 := J_1 \cup I_1(S)$
 $I_1(\epsilon)$ wird markiert
2. Durchlauf: $J_1 := J_1 \cup I_1(Sa)$
 $I_1(S)$ wird markiert
3. Durchlauf: $J_1 := J_1 \cup I_1(SaS)$
 $I_1(Sa)$ wird markiert
4. Durchlauf: $J_1 := J_1 \cup I_1(SaSa)$
 $J_1 := J_1 \cup I_1(SaSb)$
 $I_1(SaS)$ wird markiert
5. Durchlauf: $J_1 := J_1 \cup I_1(SaSaS)$
 $I_1(SaSa)$ wird markiert
6. Durchlauf: $I_1(SaSb)$ wird markiert
7. Durchlauf: $J_1 := J_1 \cup I_1(SaSaSb)$
 $I_1(SaSaS)$ wird markiert
8. Durchlauf: keine Veränderung für J_1
 $I_1(SaSaSb)$ wird markiert

\Rightarrow alle LR(k)-Informationen sind markiert

$$J_1 = \{ [S::=\epsilon.SaSb,\epsilon], [S::=\epsilon.\epsilon,\epsilon], [S::=\epsilon.SaSb,a], [S::=\epsilon.\epsilon,a], [S::=S.aSb,\epsilon], [S::=S.aSb,a], [S::=Sa.Sb,\epsilon], [S::=Sa.Sb,a], [S::=Sa.Sb,a], [S::=\epsilon.SaSb,b], [S::=\epsilon.\epsilon,b], [S::=SaS.b,\epsilon], [S::=SaS.b,a], [S::=\epsilon.SaSb,b], [S::=\epsilon.SaSb,a], [S::=SaSb.\epsilon,\epsilon], [S::=SaSb.\epsilon,a], [S::=SaSb.\epsilon,b], [S::=S.aSb,a], [S::=S.aSb,b], [S::=SaS.b,b] \}$$

3. Schritt: Überprüfung, ob J_1 konsistent:

1) Zu zeigen: $\neg(\exists [A::=\alpha.\varepsilon,u], [A':=\alpha'.\varepsilon,v] \in I: u=v)$.

Offensichtlich erfüllt.

2) Zu zeigen: $\neg(\exists [A::=\alpha.\varepsilon,u], [A':=\alpha'_1.\alpha'_2,v] \in I: {}_1(\alpha'_2) \in T, u \in L_1(\alpha'_2 v))$.

$I_1(\varepsilon)$: da $\neg(\exists [A':=\alpha'_1.\alpha'_2,v] \in I_1(\varepsilon): {}_1(\alpha'_2) \in T)$, ist Bedingung erfüllt

$I_1(S)$: da $\neg(\exists [A::=\alpha.\varepsilon,u] \in I_1(S))$, ist Bedingung erfüllt

$I_1(Sa)$: da $\neg(\exists [A':=\alpha'_1.\alpha'_2,v] \in I_1(Sa): {}_1(\alpha'_2) \in T)$, ist Bedingung erfüllt

$I_1(SaS)$: da $\neg(\exists [A::=\alpha.\varepsilon,u] \in I_1(SaS))$, ist Bedingung erfüllt

$I_1(SaSa)$: da $\neg(\exists [A':=\alpha'_1.\alpha'_2,v] \in I_1(SaSa): {}_1(\alpha'_2) \in T)$, ist Bedingung erfüllt

$I_1(SaSb)$: da $\neg(\exists [A':=\alpha'_1.\alpha'_2,v] \in I_1(SaSb): {}_1(\alpha'_2) \in T)$, ist Bedingung erfüllt

$I_1(SaSaS)$: da $\neg(\exists [A::=\alpha.\varepsilon,u] \in I_1(SaSaS))$, ist Bedingung erfüllt

$I_1(SaSaSb)$: da $\neg(\exists [A':=\alpha'_1.\alpha'_2,v] \in I_1(SaSaSb): {}_1(\alpha'_2) \in T)$, ist Bedingung erfüllt

$\Rightarrow J_1$ ist konsistent

$\Rightarrow G$ ist LR(1)-Grammatik

q.e.d.

Zusammenfassung der LR(k)-Analyse

- 1) Zu jeder Grammatik G und jedem festen $k \in \mathbf{N}$ ist **entscheidbar**, ob G eine LR(k)-Grammatik ist.
- 2) Während der Analyse ist für jedes aktuelle Paar (γ, y) , das im Laufe der Analyse eines Wortes w auftritt, **entscheidbar**, ob (γ, y) in einer Klasse O_p^k liegt und in welcher.
- 3) Jedes (γ, y) liegt in höchstens einer O_p^k . Die Klasse bestimmt die nächste Aktion der Analyse.

Algorithmen

A. Überprüfen von G auf LR(k)-Eigenschaften

1. $\neg(S \xrightarrow{+} S)$?
Überprüfen durch den Algorithmus "Zyklisches Axiom" in Verbindung mit "ε-Ableitungen".
2. Bestimmung von J_k durch den Algorithmus "Berechnung der kanonischen Kollektion J_k " mit Hilfe des Algorithmus "Berechnung der LR(k)-Informationen I_k " für gegebenes k .
3. Überprüfung der Konsistenz von J_k .

B. Syntaxanalyse bei Vorliegen der LR(k)-Eigenschaften

1. Bestimmung von J_k gemäß A2).
2. Analyse durch "deterministische Bottom-up-Analyse" anhand der LR(k)-Obermengen O_p^k .

4.3.2.3 LL(k)-Analyse

Bevor wir uns der tabellengesteuerten LR(k)-Analyse zuwenden, wollen wir noch kurz die LL(k)-Analyse charakterisieren.

Definition 4.18 Sei $G=(N,T,P,S)$ eine reduzierte kontextfreie Grammatik und $k \geq 1$ eine ganze Zahl. Zu jeder Produktion $A::=\alpha \in P$ definieren wir als *LL(k)-Obermenge* der Ableitungsklasse $A_{A::=\alpha}$ die Menge

$$\Omega_{A::=a}^k := \{(Ar,x) \mid A \in N, r \in (N \cup T)^*, x \in T^* \wedge \exists y \in T^*: (Ar,y) \in A_{A::=\alpha} \wedge k(x)=k(y)\}.$$

Die Grammatik G heißt *LL(k)-Grammatik*, wenn die zu verschiedenen Produktionen $p,p' \in P$ gehörenden Obermengen Ω_p^k und $\Omega_{p'}^k$ disjunkt sind. Eine Sprache heißt *LL(k)-Sprache*, wenn es eine LL(k)-Grammatik gibt, die sie erzeugt.

Als *LL(k)-Analyse* bezeichnen wir die deterministische Top-down-Analyse gemäß Algorithmus unter Zugrundelegung der oben definierten LL(k)-Obermengen Ω_p^k , $p \in P$.

Nach Definition ist $\Omega_{A::=a}^k$ die Menge aller Paare (Ar,x) mit $A \in N$, $r \in (N \cup T)^*$ und $x \in T^*$, welche in der 1. Komponente und in den ersten k Symbolen der 2. Komponente mit einem Paar (Ar,y) in der Ableitungsklasse übereinstimmen. Ist $G=(N,T,P,S)$ eine LL(k)-Grammatik für ein $k \geq 0$, so gehört jedes Paar (Ar,x) mit $A \in N$, $r \in (N \cup T)^*$ und $x \in T^*$ höchstens einer Menge Ω_p^k , $p \in P$, an. Das bedeutet: Zu jedem (Ar,x) existiert höchstens eine Produktion $A::=\alpha$ mit $S \xrightarrow{*}_L \rightarrow lAr \xrightarrow{*}_L \rightarrow l\alpha r \xrightarrow{*}_L \rightarrow ly$ für ein $y \in T^*$ und $k(y) = k(x)$.

Für die LL(k)-Analyse gilt damit: Ist der Stand der Analyse eines Wortes w durch ein aktuelles Paar (Ar,x) charakterisiert, so ist die nun anzuwendende Produktion festgelegt durch Ar (d.h. die Vorgeschichte der Analyse) und den lookahead, der aus den ersten k Symbolen des noch nicht verarbeiteten Suffixes x von w besteht.

Satz 4.14 Für jedes $k \geq 1$ ist die Klasse der LL(k)-Sprachen eine echte Teilmenge der Klasse der LL(k+1)-Sprachen. Die Klassen der LL(k)-Sprachen bilden für $k \geq 1$ eine unendliche Hierarchie.

4.3.2.4 Tabellengesteuerte LR(k)-Analyse

Unser Ziel ist es, durch geschicktes Anlegen von Tabellen die LR(k)-Analyse zu optimieren.

Definition 4.19 J_k sei die kanonische Kollektion der LR(k)-Informationen für eine gegebene Grammatik G . Die *GOTO-Funktion* g zu J_k ist gegeben durch:

$$g: J_k \times (N \cup T) \rightarrow (J_k \cup \emptyset),$$

$$g(I_k(\gamma), X) = I_k(\gamma X) \text{ für alle } X \in (N \cup T), I_k(\gamma) \in J_k.$$

Die GOTO-Funktion g ergibt sich aus dem Algorithmus A2) :

g	...	X	...
\vdots			
I		$g(I, X)$	
\vdots			

und kann somit gleichzeitig durch A2) angelegt werden.

Beispiel

Sei $G = (\{S\}, \{a, b\}, P, S)$ mit $P = \{S ::= SaSb, S ::= \epsilon\}$ (vgl. auch Seite 82) .

Seien weiter Z_0, \dots, Z_7 wie folgt definiert:

$$Z_0 := I_1(\epsilon)$$

$$Z_1 := I_1(S)$$

$$Z_2 := I_1(Sa)$$

$$Z_3 := I_1(SaS)$$

$$Z_4 := I_1(SaSa)$$

$$Z_5 := I_1(SaSb)$$

$$Z_6 := I_1(SaSaS)$$

$$Z_7 := I_1(SaSaSb)$$

Dann ergibt sich GOTO-Funktion g nach Algorithmus A2) zu:

g	S	a	b
Z_0	Z_1		
Z_1		Z_2	
Z_2	Z_3		
Z_3		Z_4	Z_5
Z_4	Z_6		
Z_5			
Z_6		Z_4	Z_7
Z_7			

Betrachte nun die folgenden gekellerten Informationen:

	X_1	X_2	...	X_i	X_{i+1}	...	X_n
I^0	I^1	I^2	...	I^j	I^{j+1}	...	I^n

wobei: $I^0 := I_k(\epsilon)$, $I^j := I_k(X_1 \dots X_j)$, $X_i \in (N \cup T)$, $i, j = 1, \dots, n$.

Dann sind am oberen Kellerrand zwei Aktionen möglich: Reduktion und Shift:

1. Reduktion der Form: $A ::= X_{j+1} \dots X_n$

Wir erhalten am oberen Kellerrand:

	X_1	X_2	...	X_i	A
I^0	I^1	I^2	...	I^j	

Dieser wird durch die GOTO-Funktion g modifiziert zu:

g		X_1	X_2	...	X_i	A
\Rightarrow	I^0	I^1	I^2	...	I^j	I^{j+1}

wobei $I^{j+1} = g(I^j, A)$.

2. Shift:

Wir erhalten am oberen Kellerand:

shift		X_1	X_2	...	X_n	X_{n+1}
\Rightarrow	I^0	I^1	I^2	...	I^n	

Dieser wird durch die GOTO-Funktion g modifiziert zu:

g		X_1	X_2	...	X_n	X_{n+1}
\Rightarrow	I^0	I^1	I^2	...	I^n	I^{n+1}

wobei $I^{n+1} = g(I^n, X_{n+1})$.

Wir stellen also insgesamt fest, daß bei Vorliegen einer Wertetabelle für die GOTO-Funktion g nach jeder Veränderung des Kellers die Angabe der LR(k)-Information zu jedem Anfangsstück des Kellers vervollständigt werden kann durch die Entnahme des entsprechenden Funktionswertes aus der Wertetabelle für die GOTO-Funktion.

Definition 4.20 Als *LR(k)-Analyseaktionsfunktion* zu einer Grammatik G bezeichnet man die Funktion $f: J_k \times_k (T^*) \rightarrow P \cup \{\text{shift}\} \cup \{\text{error}\}$ mit:

$$f(I, u) = \begin{cases} A::=\alpha & , \text{ falls } [A::=\alpha, \varepsilon, u] \in I \\ \text{shift} & , \text{ falls: } \exists [A::=\alpha_1, \alpha_2, v] \in I : {}_1(\alpha_2) \in T \wedge u \in L_k(\alpha_2 v) \\ \text{error} & , \text{ sonst} \end{cases}$$

Die tabellarische Darstellung heißt *LR(k)-Analyseaktionstabelle*.

Mit Hilfe der GOTO-Funktion und der Analyseaktionsfunktion (unter Zugrundelegung der LR(k)-Informationen) haben wir nun einen deterministischen Kellerautomaten zur LR(k)-Analyse angegeben. Die $I \in J_k$ sind die LR(k)-Zustände zur Grammatik G , $I_k(\varepsilon)$ ist der Anfangszustand. Die Analyseaktionsfunktion bestimmt die nächste Aktion in Abhängigkeit vom aktuellen Zustand und den nächsten k Zeichen der Eingabe, die GOTO-Funktion den neuen Zustand. Dieses geschieht wie folgt:

Im Keller wird an Stelle von γ (beim aktuellen Paar (γ, y)) jetzt γ_z gespeichert, wobei $\gamma_z = Z_0 X_1 Z_1 X_2 Z_2 \dots X_n Z_n$, falls $\gamma = X_1 \dots X_n$ und $Z_0 = I_k(\epsilon)$, $Z_i = I_k(X_1 \dots X_i)$:

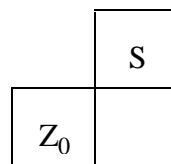
	X_1	X_2	...	X_n
Z_0	Z_1	Z_2	...	Z_n

\Rightarrow Der nächste Analyseschritt ist bestimmt durch Z_n am oberen Kellerrand und $k(y)$.

1. Fall: $f(Z_{n,k}(y)) = A ::= X_{j+1} \dots X_n$.

Die Folge $X_{j+1} Z_{j+1} \dots X_n Z_n$ am oberen Kellerrand wird ersetzt durch A.

a) Danach liegt das "aktuelle Paar" $(Z_0 S, \epsilon)$ vor:



\Rightarrow Analyse abgeschlossen

b) In allen anderen Fällen ist $g(Z_j, A)$ im Keller zu speichern.

2. Fall: $f(Z_{n,k}(y)) = \text{shift}$.

Im Keller wird dann:

- $1(y)$
und
- $g(Z_{n,1}(y))$ gespeichert.

3. Fall: $f(Z_{n,k}(y)) = \text{error}$.

\Rightarrow Fehler

Bemerkung:

Die Berechnung der LR(k)-Analyseaktionstabelle kann zusammen mit der Überprüfung der Konsistenz der LR(k)-Informationen erfolgen, so daß dadurch kein Mehraufwand entsteht (vgl. auch Seite 97).

Algorithmus: Tabellengesteuerte LR(k)-Analyse

- Eingabe:
- ganze Zahl $k \geq 0$
 - LR(k)-Grammatik $G=(N,T,P,S)$, deren Produktionen mit 1 bis $|P|$ markiert sind und $\Pi=\{1,\dots,|P|\}$
 - Wertetabellen der zur Grammatik G gehörenden Analyseaktionsfunktion f und GOTO-Funktion g
 - zu analysierendes Wort $w \in T^*$

Ausgabe: Folge der Produktionen zur Analyse von w bzw. Fehlermeldung

```

var at; (* at ist "neues aktuelles Paar" mit: at=( $\gamma_Z, y, \pi$ ), wobei  $y \in T^*$ ,  $\pi \in \Pi^*$  *)

begin
  at:=( $Z_0, w, \epsilon$ );
  repeat
    (* sei at=( $\delta Z, y, \pi$ ) mit:  $\delta Z \in Z_0(VZ_k)^*$ ,  $V=(N \cup T)$  *)
3:   if f( $Z, k(y)$ )=error then
      at:=error;
2:   if f( $Z, k(y)$ )=shift then
      begin
        if  $y=\epsilon$  then
          at:=error
        else
          if  $g(Z, t)=\emptyset$  then (*  $y=ty'$ ,  $t \in T$ ,  $y' \in T^*$  *)
            at:=error
          else
            at:=( $\delta Z t g(Z, t), y', \pi$ )
        end;
1:   if f( $Z, k(y)$ ) = A::= $\alpha$  then
      begin
        (* Sei  $\alpha=\alpha_1 \dots \alpha_m$ . Dann haben die obersten ( $2 \cdot m$ ) Elemente des *)
        (* Kellers die Form:  $\alpha_1 Z_{n-m+1} \dots \alpha_m Z_n$  *)
        "Lösche die oberen  $2 \cdot |\alpha|$  Symbole im Keller";
        (*  $\delta'Z'$  sei der verbleibende Kellerinhalt *)
        if ( $Z'=Z_0$ ) and ( $A=S$ ) and ( $y=\epsilon$ ) then
          at:=( $Z_0 S, \epsilon, i\pi$ ) (* i sei die Marke von A::= $\alpha$  *)
        else
          if  $g(Z', A)=\emptyset$  then
            at:=error
          else
            at:=( $\delta'Z' A g(Z', A), y, i\pi$ )
        end
      until (at=error) or (at=( $Z_0 S, \epsilon, \pi$ ));
      if at=error then
        "w $\notin$  L(G)"
      else
        "Ausgabe von  $\pi$ "
      end.

```

Algorithmus: LR(k)-Analyseaktionsfunktion

Eingabe: - Reduzierte kontextfreie Grammatik $G=(N,T,P,S)$ mit: $\neg(S \xrightarrow{+} S)$
 - ganze Zahl $k \geq 0$
 - J_k

Ausgabe: Wertetabelle der LR(k)-Analyseaktionsfunktion, falls G eine LR(k)-Grammatik ist bzw. eine entsprechende Meldung sonst.

```

for "jedes  $I \in J_k$  und jedes  $u_k \in (T^*)$ " do
  begin
     $f(I,u) := \text{error}$ ;
    for "jedes  $[A ::= \alpha_1 \cdot \alpha_2, v] \in I$ " do
      begin
        if  $(\alpha_2 = \epsilon)$  and  $(u=v)$  then
          begin
            if  $f(I,u) = \text{error}$  then
               $f(I,u) := A ::= \alpha_1$ 
            else
              write("keine LR(k)-Grammatik")
            end;
          if  $(\alpha_1 \in T)$  and  $(u \in L_k(\alpha_2 v))$  then
            begin
              if  $(f(I,u) = \text{error})$  or  $(f(I,u) = \text{shift})$  then
                 $f(I,u) := \text{shift}$ 
              else
                write("keine LR(k)-Grammatik")
              end
            end
          end
        end;
      end;
    end;
  end;

```

Beispiel

Sei $G = (\{S\}, \{a, b\}, P, S)$ mit $P = \{S ::= SaSb, S ::= \epsilon\}$ (vgl. auch Seite 82).

Seien Z_0, \dots, Z_7 definiert wie auf Seite 91.

Betrachte die LR(1)-Analyseaktionstabelle zur Grammatik G:

f	a	b	ϵ
Z_0	$S ::= \epsilon$	error	$S ::= \epsilon$
Z_1	shift	error	error
Z_2	$S ::= \epsilon$	$S ::= \epsilon$	error
Z_3	shift	shift	error
Z_4	$S ::= \epsilon$	$S ::= \epsilon$	error
Z_5	$S ::= SaSb$	error	$S ::= SaSb$
Z_6	shift	shift	error
Z_7	$S ::= SaSb$	$S ::= SaSb$	error

Sei $w ::= aabb$.

Dann durchläuft der Keller bei der Analyse von w die folgenden Zustände:

gezellerte Informationen:

Z_0

nächstes zu lesende Zeichen im Analysewort:

aabb

↑

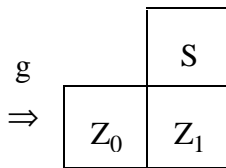
(\Rightarrow Reduktion $S ::= \epsilon$)

f

S
Z ₀

\Rightarrow

(\Rightarrow Vervollständigung der LR(1)-Informationen)

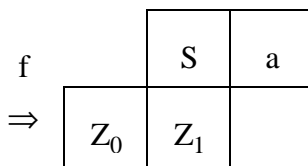


nächstes zu lesende Zeichen im Analysewort:

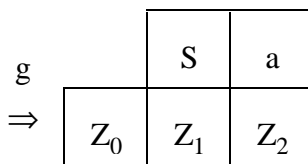
aabb

↑

(\Rightarrow Shift)



(\Rightarrow Vervollständigung der LR(1)-Informationen)

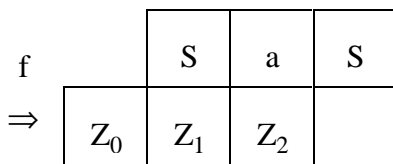


nächstes zu lesende Zeichen im Analysewort:

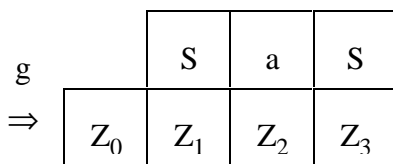
aabb

↑

(\Rightarrow Reduktion $S ::= \epsilon$)



(\Rightarrow Vervollständigung der LR(1)-Informationen)

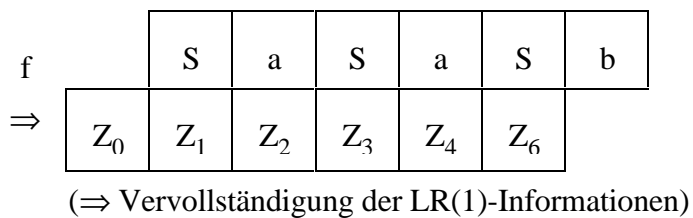
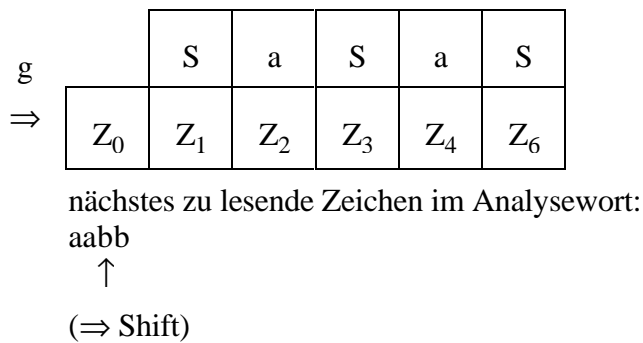
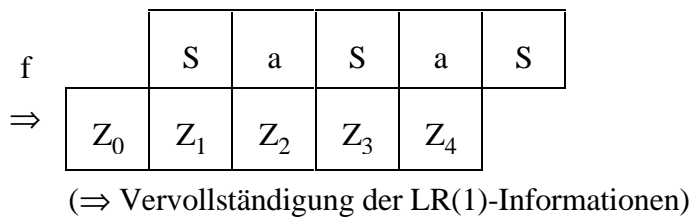
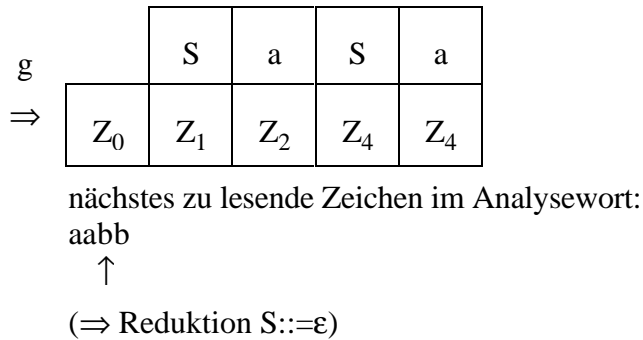
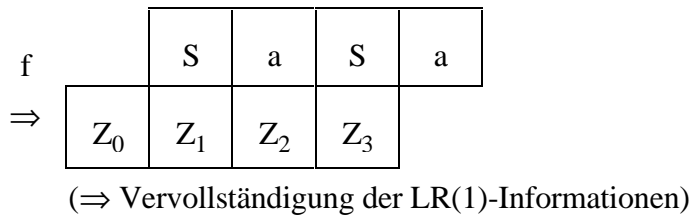


nächstes zu lesende Zeichen im Analysewort:

aabb

↑

(\Rightarrow Shift)



$$\Rightarrow$$

	S	a	S	a	S	b
Z ₀	Z ₁	Z ₂	Z ₃	Z ₄	Z ₆	Z ₇

nächstes zu lesende Zeichen im Analysewort:

aabb

↑

(\Rightarrow Reduktion $S ::= SaSb$)

$$\Rightarrow$$

	S	a	S
Z ₀	Z ₁	Z ₂	

(\Rightarrow Vervollständigung der LR(1)-Informationen)

$$\Rightarrow$$

	S	a	S
Z ₀	Z ₁	Z ₂	Z ₃

nächstes zu lesende Zeichen im Analysewort:

aabb

↑

(\Rightarrow Shift)

$$\Rightarrow$$

	S	a	S	b
Z ₀	Z ₁	Z ₂	Z ₃	

(\Rightarrow Vervollständigung der LR(1)-Informationen)

$$\Rightarrow$$

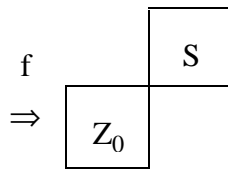
	S	a	S	b
Z ₀	Z ₁	Z ₂	Z ₃	Z ₅

nächstes zu lesende Zeichen im Analysewort:

aabbε

↑

(\Rightarrow Reduktion $S ::= SaSb$)



$\Rightarrow w \in L(G)$

Satz 4.15 Für eine kontextfreie Sprache L sind folgende Eigenschaften äquivalent:

- i) L ist deterministisch.
- ii) Es gibt eine LR(1)-Grammatik, die L erzeugt.
- iii) Es gibt eine LR(k)-Grammatik, die L erzeugt.

Bemerkung:

Wenn das Wortende ohne look-ahead erkannt werden kann, dann gilt sogar:

- ii') Es gibt eine LR(0)-Grammatik, die L erzeugt.

Satz 4.19 Die kanonische LR(k)-Analyse (vergleiche obigen Algorithmus) hat linearen Zeitbedarf.

4.3.3 Zusammenfassung der verschiedenen Analyseverfahren

Nichtdeterministische Analyse-Algorithmen

- **Bottom-up-Analyse mit Backtracking**
Die Bottom-up-Analyse entspricht einer Rechtsreduktion.
- **Top-down-Analyse mit Backtracking**
Die Top-down-Analyse entspricht einer Linksableitung.

Deterministische Analysealgorithmen

Hier haben wir zwei verschiedene Strategien betrachtet:

1. Tabellenorientierte Analysestrategien:
 - **Analysealgorithmus von Cocke-Kasami-Younger**

 2. Ableitungsorientierte Analysestrategien:
 - i) Deterministische Bottom-up-Analyse
 - ii) LR(k)-Analyse
 - iii) Tabellengesteuerte LR(k)-Analyse
-
- i) **Deterministische Bottom-up-Analyse**
Reduktionsklassen (Shiftklassen) gegeben
 \Rightarrow Deterministische Bottom-up-Analyse kann durchgeführt werden. Zum aktuellen Paar (γ, r) wird ermittelt, ob es ein $p \in P \cup \{\text{shift}\}$ gibt mit:
 $(\gamma, r) \in R_p$.
 Für $w \in L(G)$ existiert ein solches p .
 \Rightarrow (deterministische) Auswahl und Ausführung der Operation, die durch p beschrieben wird.

Problem: Zu gegebenem (γ, y) muß entschieden werden, ob $(\gamma, y) \in R_p$. Aufwand zur Bestimmung der R_p ist im allgemeinen groß.

\Rightarrow Übergang zu Obermengen O_p von R_p .

Anforderung an die reduzierte kontextfreie Grammatik $G=(N, T, P, S)$ und die Obermengen O_p :

i) $\neg(S \xrightarrow{+} S)$

und

ii) Für $p, p' \in P \cup \{\text{shift}\}$, $p \neq p'$ gilt: $O_p \cap O_{p'} = \emptyset$

\Leftrightarrow die reduzierte kontextfreie Grammatik $G=(N, T, P, S)$ ist eindeutig (i.a. ist diese Eigenschaft nicht entscheidbar).

Ergebnis: Algorithmus für deterministische Bottom-up-Analyse.

Verfeinerung der deterministischen Bottom-up-Analyse:

ii) **LR(k)-Analyse**

Definition der LR(k)-Obermenge O_p^k

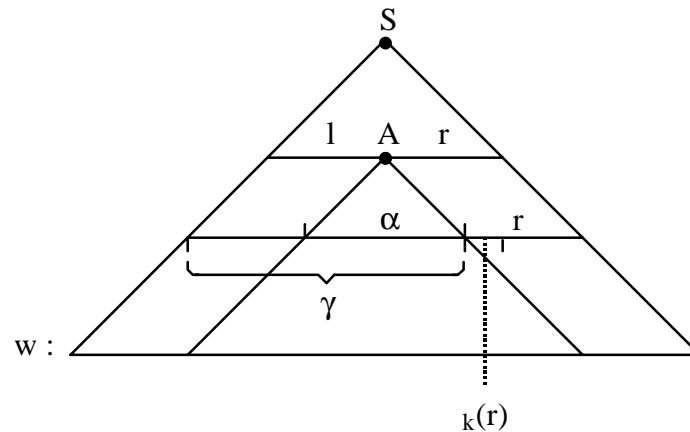
\Rightarrow Zu jedem Zeitpunkt der Analyse benötigt man zu Bestimmung der nächsten Aktion nur γ und den look-ahead der ersten k Symbole des noch nicht gelesenen Teils der Eingabe. D.h. γ und ${}_k(r)$ geben Auskunft, ob $(\gamma, r) \in O_p^k$ für ein $p \in P \cup \{\text{shift}\}$.

Aufgabe: Finde beschränkte Information von γ , die mit ${}_k(r)$ die Zugehörigkeit zu O_p^k bestimmt.

\Rightarrow Definition der:

- LR(k)-Auskunft zu γ : $[A ::= \alpha_1 \cdot \alpha_2, u]$
- LR(k)-Information zu γ : $I_k(\gamma) =$ Menge aller LR(k)-Auskünfte zu γ
- kanonischen Kollektion $J_k = \{I_k(\gamma) \neq \emptyset \mid \gamma \in (N \cup T)^*\}$.

Strukturbaum τ zum Analysewort w :



$I_k(\gamma)$ gibt darüber Auskunft, welche Reduktionen am rechten Rand von γ möglich sind bzw. nach eventuellem Shiften möglich werden.

Hauptsatz über LR(k)-Grammatiken:

Der Hauptsatz besagt, daß für (γ, r) und $p \in P \cup \{\text{shift}\}$ die Zugehörigkeit von (γ, r) zu O_p^k bereits durch p , $I_k(\gamma)$ und $k(r)$ bestimmt ist.

Definition von Konsistenz:

Konsistenz von J heißt: Kein I in J enthält verschiedene LR(k)-Auskünfte der Form:

$[A ::= \alpha. \varepsilon, u]$ und $[A' ::= \alpha'. \varepsilon, v]$ mit $u \neq v$

oder

$[A ::= \alpha. \varepsilon, u]$ und $[A' ::= \alpha'_1. \alpha'_2, v]$ mit $\alpha'_2 \in T$ und $u \in L_k(\alpha'_2 v)$.

Diese Kenntnisse erlauben eine Modifizierung der deterministischen Bottom-up-Analyse; durch einfachere Kriterien verringert sich der Aufwand.

Zu überprüfen bleibt:

- 1) $\neg(S \xrightarrow{+} S)$?
- 2) Bestimmung von J_k
- 3) J_k konsistent ?

$\neg(S \xrightarrow{+} S)$ und J_k konsistent

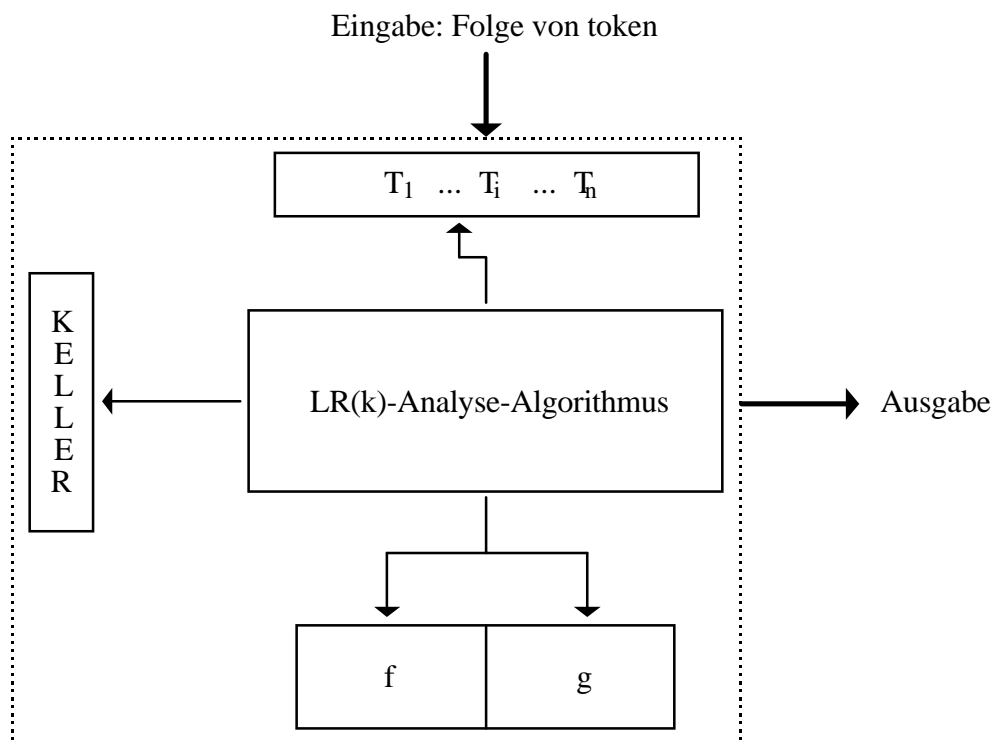
\Rightarrow deterministische Bottom-Up-Analyse

iii) **Tabellengesteuerte LR(k)-Analyse**

Optimierung des Analyseverfahrens durch Einführen der GOTO- und Analyseaktionsfunktion.

Die LR(k)-Informationen entsprechen Zuständen.

Wir erhalten also folgende Abhängigkeiten:



Wir wollen nun abschließend anhand eines Beispiels alle gewonnenen Kenntnisse über die optimierte LR(k)-Analyse zusammenfassen:

Beispiel

Sei $G = (\{A, B, S\}, \{0, 1\}, \{S ::= AB, A ::= 0A1 \mid \epsilon, B ::= 1B \mid 1\}, S)$.

Behauptung: G ist LR(1)-Grammatik.

Beweis:

a) ϵ -Ableitungen : $E = \{A\}$

b) zyklisches Axiom :

$$W = \emptyset$$

Repeatschleife:

1. Durchlauf:

$$W_{\text{alt}} = \emptyset$$

$$W = \emptyset \cup \{C \in N \mid \exists B ::= \beta C \gamma : B = S \vee B \in W, \beta, \gamma \in (N \cup T)^* \wedge \beta^* \rightarrow \epsilon \wedge \gamma^* \rightarrow \epsilon\} = \{B\}$$

2. Durchlauf:

$$W_{\text{alt}} = \emptyset$$

$$W = \{B\} \cup \emptyset = \{B\}$$

$$W = W_{\text{alt}} \Rightarrow \text{Abbruch}$$

Ausgabe: S nicht zyklisch.

Aus a) und b) folgt: $\neg(S^+ \rightarrow S)$.

c) Bestimmung der kanonischen Kollektion J_1 zu G :

$$I_1(\epsilon) = \{[S::=\epsilon.AB,\epsilon]\} \cup \{[A::=\epsilon.0A1,1],[A::=\epsilon.\epsilon,1]\}$$

$$J_1 := I_1(\epsilon)$$

Repeatschleife (im Algorithmus zur Bestimmung von J_1) :

1. Durchlauf :

$$I_1(A) = \{[S::=A.B,\epsilon]\} \cup \{[B::=\epsilon.1B,\epsilon],[B::=\epsilon.1,\epsilon]\}$$

$$J_1 := J_1 \cup I_1(A)$$

$$I_1(B) = \emptyset$$

$$I_1(S) = \emptyset$$

$$I_1(1) = \emptyset$$

$$I_1(0) = \{[A::=0.A1,1]\} \cup \{[A::=\epsilon.0A1,1],[A::=\epsilon.\epsilon,1]\}$$

$$J_1 := J_1 \cup I_1(0)$$

$I_1(\epsilon)$ wird markiert

2. Durchlauf :

$$I_1(AA) = \emptyset$$

$$I_1(AB) = \{[S::=AB.\epsilon,\epsilon]\}$$

$$J_1 := J_1 \cup I_1(AB)$$

$$I_1(AS) = \emptyset$$

$$I_1(A1) = \{[B::=1.B,\epsilon],[B::=1.\epsilon,\epsilon]\} \cup \{[B::=\epsilon.1B,\epsilon],[B::=\epsilon.1,\epsilon]\}$$

$$J_1 := J_1 \cup I_1(A1)$$

$$I_1(A0) = \emptyset$$

$I_1(A)$ wird markiert

3. Durchlauf :

$$I_1(0A) = \{[A::=0A.1,1]\}$$

$$J_1 := J_1 \cup I_1(0A)$$

$$I_1(0B) = \emptyset$$

$$I_1(0S) = \emptyset$$

$$I_1(01) = \emptyset$$

$$I_1(00) = \{[A::=0.A1,1]\} \cup \{[A::=\epsilon.0A1,1],[A::=\epsilon.\epsilon,1]\}$$

J_1 wird nicht verändert, da $I_1(00) = I_1(0)$

$I_1(0)$ wird markiert, und damit auch $I_1(00)$

4. Durchlauf :

$$\forall X \in \{A,B,S,0,1\}: I_1(ABX) = \emptyset$$

$I_1(AB)$ wird markiert

$$I_1(A1A) = \emptyset$$

$$I_1(A1B) = \{[B::=1B.\epsilon,\epsilon]\}$$

$$J_1 := J_1 \cup I_1(A1B)$$

$$I_1(A1S) = \emptyset$$

$$I_1(A11) = \{[B::=1.B,\epsilon],[B::=1.\epsilon,\epsilon]\} \cup \{[B::=\epsilon.1B,\epsilon],[B::=\epsilon.1,\epsilon]\}$$

J_1 wird nicht verändert, da $I_1(A11) = I_1(A1)$

$$I_1(A10) = \emptyset$$

$I_1(A1)$ wird markiert, und damit auch $I_1(A11)$

$$I_1(0AA) = \emptyset$$

$$I_1(0AB) = \emptyset$$

$$I_1(0AS) = \emptyset$$

$$I_1(0A1) = \{[A::=0A1.\varepsilon, 1]\}$$

$$J_1 := J_1 \cup I_1(0A1)$$

$$I_1(0A0) = \emptyset$$

$I_1(0A)$ wird markiert

5. Durchlauf :

$$\forall X \in \{A, B, S, 0, 1\}: I_1(A1BX) = \emptyset$$

$I_1(A1B)$ wird markiert

$$\forall X \in \{A, B, S, 0, 1\}: I_1(0A1X) = \emptyset$$

$I_1(0A1)$ wird markiert

Da $I_1(A11)$ schon markiert, bricht Repeatschleife ab.

$$J_1 = I_1(\varepsilon) \cup I_1(A) \cup I_1(0) \cup I_1(AB) \cup I_1(A1) \cup I_1(0A) \cup I_1(A1B) \cup I_1(0A1)$$

J_1 ist konsistent $\Rightarrow G$ ist LR(1)-Grammatik.

q.e.d.

Es werden nun die zur Analyse eines Wortes w benötigten Tabellen angelegt:

1. Berechnung der GOTO-Funktion g

$$Z_0 := I_1(\epsilon)$$

$$Z_1 := I_1(A)$$

$$Z_2 := I_1(0)$$

$$Z_3 := I_1(AB)$$

$$Z_4 := I_1(A1)$$

$$Z_5 := I_1(0A)$$

$$Z_6 := I_1(A1B)$$

$$Z_7 := I_1(0A1)$$

frei = error

g	A	B	S	0	1
Z_0	Z_1			Z_2	
Z_1		Z_3			Z_4
Z_2	Z_5			Z_2	
Z_3					
Z_4		Z_6			Z_4
Z_5					Z_7
Z_6					
Z_7					

2. Berechnung der LR(k)-Analyseaktionsfunktion f

f	0	1	ϵ
Z ₀	shift	A::= ϵ	error
Z ₁	error	shift	error
Z ₂	shift	A::= ϵ	error
Z ₃	error	error	S::=AB
Z ₄	error	shift	B::=1
Z ₅	error	shift	error
Z ₆	error	error	B::=1B
Z ₇	error	A::=0A1	error

Damit liegen alle Informationen zur Analyse eines Worte w vor.

Sei $w:=001111$.

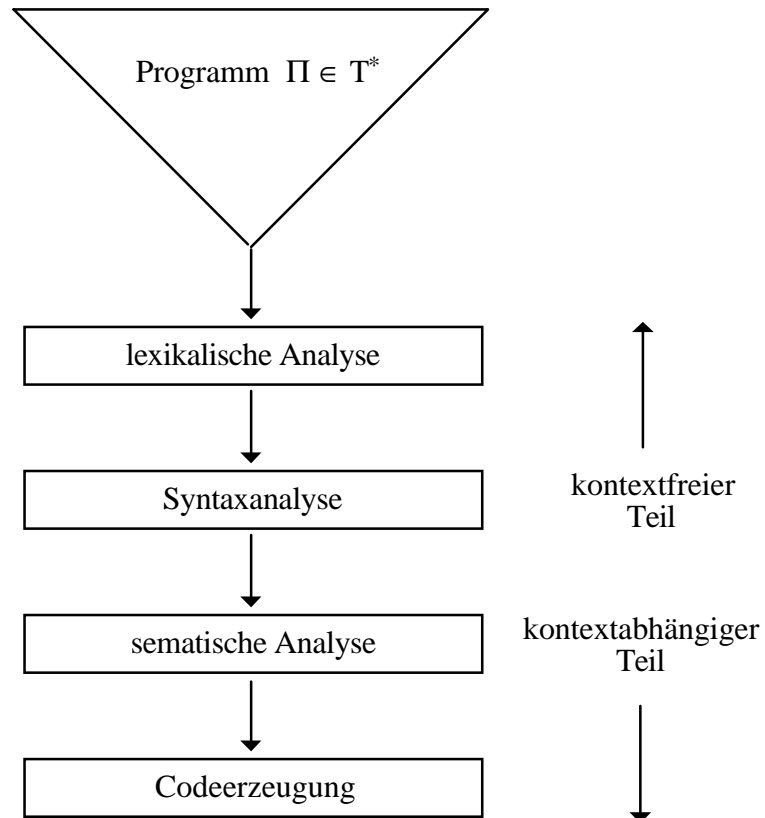
Dann durchläuft der Keller bei der Analyse von w die folgenden Zustände:

$(Z_0, 001111, \epsilon)$
 $\vdash (Z_0 0 Z_2, 01111, \epsilon)$
 $\vdash (Z_0 0 Z_2 0 Z_2, 1111, \epsilon)$
 $\vdash (Z_0 0 Z_2 0 Z_2 A Z_5, 1111, (A::=\epsilon))$
 $\vdash (Z_0 0 Z_2 0 Z_2 A Z_5 1 Z_7, 111, (A::=\epsilon))$
 $\vdash (Z_0 0 Z_2 A Z_5, 111, (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 0 Z_2 A Z_5 1 Z_7, 11, (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 A Z_1, 11, (A::=0A1) (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 A Z_1 1 Z_4, 1, (A::=0A1) (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 A Z_1 1 Z_4 1 Z_4, \epsilon, (A::=0A1) (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 A Z_1 1 Z_4 B Z_6, \epsilon, (B::=1) (A::=0A1) (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 A Z_1 B Z_3, \epsilon, (B::=1B) (B::=1) (A::=0A1) (A::=0A1) (A::=\epsilon))$
 $\vdash (Z_0 S, \epsilon, (S::=AB) (B::=1B) (B::=1) (A::=0A1) (A::=0A1) (A::=\epsilon))$

$\Rightarrow w \in L(G)$

5 Semantische Analyse

5.1 Aufgaben der semantischen Analyse



Während der semantischen Analyse werden die kontextabhängigen Rahmenbedingungen eines Programms Π geprüft. Hierbei kann natürlich nur die statische Semantik überprüft werden, alles andere muß zur Laufzeit geprüft werden.

Die folgende Aufgaben fallen in den Bereich der semantischen Analyse (sind also statisch überprüfbar):

1. Auffinden von fehlenden Deklarationen oder Mehrfachdeklarationen.
2. Bestimmung des Typs von "dicken" Ausdrücken.
3. Typkonformität der linken und rechten Seite einer Ergibt-Anweisung.
4. Überprüfung der Indexgrenzen (bei statischen Grenzen).
5. Bei (nicht formalen) Unterprogrammaufrufen: Überprüfung, ob
 - die Anzahl der Argumente
 - und
 - die Typen der Argumente (soweit spezifiziert)übereinstimmen.
6. Überprüfung des Typs an Stellen, an denen ein boolescher Ausdruck nötig ist.

5.2 Hilfsmittel zur semantischen Analyse

Zur sematischen Analyse eines Programms Π gibt es prinzipiell zwei Hilfsmittel:

- 1) Attributierte Grammatiken
Sie sind Erweiterungen von kontextfreien Grammatiken und werden vor allem bei compilererzeugenden Systemen eingesetzt. Sie haben den Vorteil, daß man gleichzeitig Syntaxanalyse und semantische Analyse behandeln kann.
- 2) Tabellen
Wie oben schon bemerkt, muß im Rahmen der semantischen Analyse unter anderem überprüft werden:
 - i) Ob zu jedem angewandten Vorkommen eines Identifikators genau ein entsprechendes deklarierendes Vorkommen existiert (d.h. ob das Programm Π gebunden ist).
 - ii) Typkonformität.

Um die dazu nötigen Informationen festzuhalten, wird eine Identifikorentabelle in Form eines Kellers angelegt. Die Einträge haben die folgende Form:

Eintrag[Name, Typ, Relativadresse, statisches Blockniveau, Blockzähler]

Während der Analyse müssen wir dann zwei Fälle betrachten:

a) Treffen auf Deklaration

Es wird überprüft, ob im aktuellen Block ein Eintrag mit gleichem Namen vorhanden ist.

JA \Rightarrow Fehlermeldung (Doppeldeklaration)

NEIN \Rightarrow Eintrag in die Identifikorentabelle

b) Treffen auf angewandtes Auftreten

Es wird überprüft, ob ein Eintrag mit gleichem Namen in der Identifikorentabelle vorhanden ist (in Abhängigkeit vom Blockniveau und dem Blockzähler)

JA \Rightarrow Entnahme des Typs

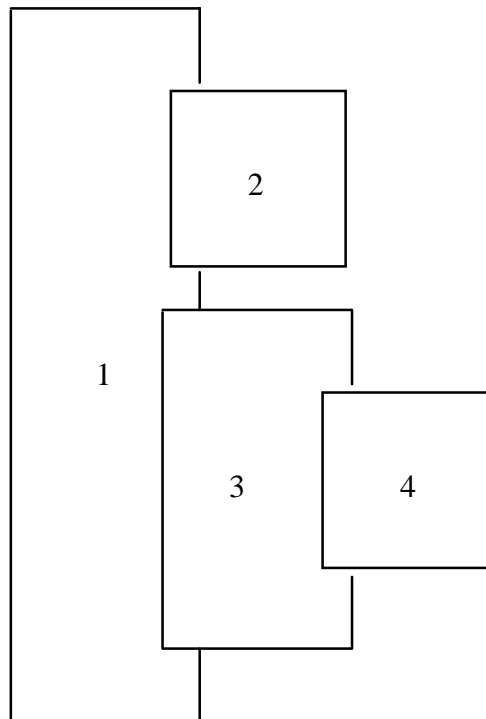
NEIN \Rightarrow Fehlermeldung

Es bleibt nur noch zu klären, wie die Identifikorentabelle nach einem Eintrag für ein angewandtes Auftreten durchsucht wird.

Schauen wir uns dazu einmal ein Programm mit folgender Blockstruktur an:

```
begin
  ...
  begin
    ...
  end;
  begin
    ...
    begin
      ...
    end;
    ...
  end;
  ...
end.
```

Graphisch läßt sich dies wie folgt darstellen:



Dann haben die zu den einzelnen Blöcken gehörenden Einträge in der Identifikortabelle folgende Form:

Block 1: Eintrag[Name, Typ, Relativadresse, 0, 1]

Block 2: Eintrag[Name, Typ, Relativadresse, 1, 2]

Block 3: Eintrag[Name, Typ, Relativadresse, 1, 3]

Block 4: Eintrag[Name, Typ, Relativadresse, 2, 4]

Die Zeichnung zeigt sofort:

Das zu einem angewandten Auftreten eines Identifikators in Block 4 gehörende deklarierende Auftreten darf nur in den Blöcken 4, 3 und 1 gesucht werden, nicht jedoch in Block 2.

Somit ergibt sich als Suchkriterium für unsere Identifikorentabelle:

1. Suche im selben Block nach einem Eintrag für das angewandte Auftreten eines Identifikators.
2. Konnte kein Eintrag gefunden werden, so setze die Suche in Blöcken mit niedrigerem Blockniveau fort.

Werden hierbei Blöcke mit gleichem Blockniveau aber unterschiedlichen Blockzählern gefunden, so darf nur im Block mit dem höchsten Blockzähler weitergesucht werden.

5.3 Typbestimmung für "dicke Ausdrücke" mit Hilfe eines Kellers

linke Spalte = oberster Kellereintrag
 K = Typ in Keller eintragen
 E = Kellereintrag ersetzen

oberste Zeile = Typ des nächsten zu lesenden
 F = Fehler
 freies Feld = kann nach Syntax nicht vorkommen

	int	real	bool	true false	not	and or	+ -	*	/	Vergleichs- operatoren
ϵ	K	K	K	K_{bool}						
int						F	K	K	F	K
real						F	K	K	K	K
bool						K	F	F	F	F
(int						F	K	K	F	K
(real						F	K	K	K	K
(bool						K	F	F	F	F
(+	E_{int}	E_{real}	F							
(-	E_{int}	E_{real}	F							
(not	F	F	E_{bool}	E_{bool}						
(K	K	K	K_{bool}	K		K			
int +	E_{int}	F	F							
int -	E_{int}	F	F							
int *	E_{int}	F	F							
real +, -, *, /	F	E_{real}	F							
int vgl.	E_{bool}	F	F							
bool and	F	F	E_{bool}	E_{bool}						
bool or	F	F	E_{bool}	E_{bool}						

Bemerkung: Am Ende eines Blocks können die für diesen Block vergebenen Speicherplätze neu vergeben werden.

5.4 Algorithmen zur semantischen Analyse (eines Programms in MMS)

Aufbau eines Programms in MMS:

```
program Name =  
  begin  
    <Deklarationsteil>;  
    <Anweisungsteil>  
  end.
```

5.4.1 Überprüfe Anweisungsteil

```
while "Element  $\neq$  (T2,.)" do  
  begin  
    "Überprüfe Anweisung";  
    "Lese nächstes Element"  
  end;
```

5.4.2 Überprüfe Anweisung

```
if "Element = (T2,if)" then  
  "Überprüfe bedingte Anweisung"  
else  
  if "Element = (T2,while)" then  
    "Überprüfe Wiederholungsanweisung"  
  else  
    "Überprüfe Wertzuweisung";
```

5.4.3 Überprüfe bedingte Anweisung

```
"Lese nächstes Element"  
"Überprüfe Bedingung"  
"Überprüfe then-Teil"  
"Überprüfe else-Teil"  
"Lese nächstes Element"
```

5.4.4 Überprüfe Bedingung

```
"Bestimme Typ des Ausdrucks"; (* Ausdrucksende: (T2,then) *)  
if Typ ≠ boolean then  
  "Fehlermeldung";
```

5.4.5 Überprüfe then-Teil

```
while "Element ≠ (T2,else)" do  
  begin  
    "Lese nächstes Element";  
    "Überprüfe Anweisung"  
  end
```

5.4.6 Überprüfe else-Teil

```
while "Element ≠ (T2,fi)" do  
  begin  
    "Lese nächstes Element";  
    "Überprüfe Anweisung"  
  end;
```

5.4.7 Überprüfe Wiederholungsanweisung

```
"Lese nächstes Element";  
"Überprüfe Bedingung"; (* Abbruch bei Element (T2,do) *)  
"Überprüfe do-Teil";  
"Lese nächstes Element";
```

5.4.8 Überprüfe do-Teil

```
while "Element  $\neq$  (T2,od)" do
  begin
    "Lese nächstes Element";
    "Überprüfe Anweisung"
  end;
```

5.4.9 Überprüfe Wertzuweisung

```
"Suche 2(Element) in Identifikatorentabelle";
if "nicht gefunden" then
  "Fehlermeldung"
else
  "Hilfsvariable:=Typ";
"Lese nächstes Element";
"Lese nächstes Element";
"Überprüfe Ausdruck";
if "Typ  $\neq$  Hilfsvariable" then
  "Fehlermeldung";
```

5.4.10 Modifikationen für "in" und "out"

```
...
else
  if "Element = (T2,in)" then
    "Überprüfe Eingabe"
  else
    if "Element = (T2,out)" then
      "Überprüfe Ausgabe"
    else
      ... (* Überprüfe Prozeduren, Funktionen, etc. *)
```

5.4.11 Überprüfe Eingabe

```
"Lese nächstes Element";
"Lese nächstes Element";
if " $T_1(\text{Element}) \neq T_1$ " then
  "Fehler"
else
  begin
    "Suche  $T_2(\text{Element})$  in Identifikatorentabelle";
    if "nicht gefunden" then
      "Fehler"
    else
      begin
        "Lese nächstes Element";
        if " $\text{Element} \neq (T_8, )$ " then
          "Fehler"
        else
          "Lese nächstes Element"
        end
      end
    end;
end;
```

5.4.11 Überprüfe Ausgabe

Analog zu "Überprüfe Eingabe".

Allerdings ist auch die Ausgabe von Ausdrücken möglich, also: out(Ausdruck).

Anhang A

Münsteraner Mini-Sprache (MMS)

<Programm>	::= program <Programmname> = <block>.
<Programmname>	::= <Identifikator>
<Identifikator>	::= <Buchstabe> <Buchstabe><Idf.ende>
<Idf.ende>	::= <B.o.Z.> <B.o.Z.><Idf.ende> _<B.o.Z.> _<B.o.Z.><Idf.ende>
<B.o.Z.>	::= <Buchstabe> <Ziffer>
<Buchstabe>	::= A ... Z a ... z
<Ziffer>	::= 0 ... 9
<block>	::= begin <Deklarationsteil> <Anweisungsteil> end begin <Anweisungsteil> end
<Deklarationsteil>	::= <Variablendeklarationen> <Prozedurdeklarationen> <Variablendeklarationen><Prozedurdeklarationen>
<Variablendeklarationen>	::= <Variablendeklaration>; <Variablendeklaration>;<Variablendeklarationen>
<Variablendeklaration>	::= var <Variablenidentifikator>:<Typ>
<Variablenidentifikator>	::= <Identifikator>
<Typ>	::= <einf. Typ> <strukturiertes Typ>
<einf. Typ>	::= integer real boolean
<strukturiertes Typ>	::= array[<Feldgrenzen>] of <einf. Typ>
<Feldgrenzen>	::= <untere Grenze>:<obere Grenze> <untere Grenze>:<obereGrenze>,<Feldgrenzen>
<untere Grenze>	::= <arithm. Ausdruck>
<obere Grenze>	::= <arithm. Ausdruck>
<Prozedurdeklarationen>	::= <Prozedurdeklaration>; <Prozedurdeklaration>;<Prozedurdeklarationen>
<Prozedurdeklaration>	::= proc <Prozedurname><Parameterliste> = <block>
<Prozedurname>	::= <Identifikator>
<Parameterliste>	::= () (<form. Parameterfolge>)
<form. Parameterfolge>	::= <Identifikator>:<Typ 2> <Identifikator>:<Typ 2>,<form. Parameterfolge>
<Typ 2>	::= <Typ> proc

<Anweisungsteil>	::= <Anweisung> <Anweisung>;<Anweisungsteil>
<Anweisung>	::= <unmarkierte Anweisung> <markierte Anweisung>
<markierte Anweisung>	::= <Marke>:<unmarkierte Anweisung>
<Marke>	::= <Identifikator>
<unmarkierte Anweisung>	::= <unbedingte Anweisung> <bedingte Anweisung>
<unbedingte Anweisung>	::= <Wertzuweisung> <Sprunganweisung> <Prozeduraufruf> <Wiederholungsanweisung> <block>
<Wertzuweisung>	::= <Variable>:=<Ausdruck>
<Sprunganweisung>	::= goto <Marke>
<Prozeduraufruf>	::= <Prozedurname><Argumentliste>
<Wiederholungsanweisung>	::= while <Bedingung> do <Anweisungsteil> od
<Ausdruck>	::= <log. Ausdruck> <arithm. Ausdruck>
<log. Ausdruck>	::= true false <Variable> (<unärer log. Operator> <log. Ausdruck>) (<log. Ausdruck> <bin.log. Operator> <log. Ausdruck>) (<log. Ausdruck>) (<arithm. Ausdruck> <Vergleichsoperator> <arithm. Ausdruck>)
<arithm. Ausdruck>	::= <real-Zahl> <int.-Zahl> <Variable> (<arithm. Ausdruck>) (<unärer arithm. Operator><arithm. Ausdruck>) (<arithm. Ausdruck> <binärer arithm.Operator> <arithm. Ausdruck>)
<Variable>	::= <Variablenidentifikator> <Variablenidentifikator>[<Indizes>]
<Indizes>	::= <arithm. Ausdruck> <arithm. Ausdruck>,<Indizes>
<Argumentliste>	::= () (<akt. Parameterfolge>)
<akt. Parameterfolge>	::= <akt. Parameter> <akt. Parameter>,<akt. Parameterfolge>
<akt. Parameter>	::= <Ausdruck> <Prozedurname>
<Bedingung>	::= <log. Ausdruck>
<bedingte Anweisung>	::= if <Bedingung> then <Anweisungsteil> else <Anweisungsteil> fi
<int.-Zahl>	::= <Ziffer> <Ziffer><int.-Zahl>
<real-Zahl>	::= <int.-Zahl>.<int.-Zahl> .<int.-Zahl> <int.-Zahl>.

<Vergleichsoperator> ::= <|> | = | <= | => | <>
<unärer log. Operator> ::= not
<binärer log. Operator> ::= and | or
<unärer arithm. Operator> ::= + | -
<binärer arithm. Operator> ::= + | - | * | /

Literaturverzeichnis

Hans Zima, *Compilerbau I + II*, BI-Wissenschaftsverlag (1982 + 1985)

David Gries, *Compiler-Construction for Digital Computers*, McGraw-Hill (1971)

Otto Mayer, *Syntaxanalyse*, BI-Wissenschaftsverlag (1978)

Lewis, Rosenkrantz, Stearns, *Compiler Design Theory*, Addison-Wesley (1976)

H.A. Maurer, *Theoretische Grundlagen der Programmiersprachen - Theorie der Syntax*,
BI-Wissenschaftsverlag (1969)

Nikolaus Wirth, *Compilerbau*, Teubner Studienbücher (1977)

Aho, Ullman, *The Theory of Parsing, Translation and Compiling*, Prentice Hall (1972)

Hopcroft, Ullmann, *Einführung in die Automatentheorie, formale Sprachen und
Komplexitätstheorie*, Addison-Wesley (1979)

Stichwortverzeichnis

A

ableiten 17

 in mindestens einem Schritt 17

 unmittelbar 17

aktuelles Paar 66

akzeptieren

 Satz (endlicher Automat) 32

 Wort (endlicher Automat) 32

 Wort unter Erreichen eines Endzustandes (Kellerautomat) 52

 Wort unter Leeren des Kellers (Kellerautomat) 52

Algorithmus

 Berechnung der kanonischen Kollektion J_k 82

 Berechnung von $I_k(\gamma)$ 81

 Bottom-up-Analyse mit Backtracking 44

 deterministische Bottom-up-Analyse 71

ϵ -Ableitungen 80

 LR(k)-Analyseaktionsfunktion 98

 Minimierung der Zustandsmenge eines endlichen Automaten 38

 Modifikationen für "in" und "out" 122

 Tabellengesteuerte LR(k)-Analyse 96

 Top-down-Analyse mit Backtracking 47

 Überprüfe Anweisung 120

 Überprüfe Anweisungsteil 120

 Überprüfe Ausgabe 123

 Überprüfe bedingte Anweisung 120

 Überprüfe Bedingung 121

- Überprüfe do-Teil 122
- Überprüfe Eingabe 123
- Überprüfe else-Teil 121
- Überprüfe then-Teil 121
- Überprüfe Wertzuweisung 122
- Überprüfe Wiederholungsanweisung 121
- zyklisches Axiom 80

Analyse

- Bottom-up- 65
- Bottom-up- mit Backtracking 44
- Cocke-Kasami-Younger- 61
- deterministische Bottom-up- 66; 71
- LL(k)- 91
- LR(k)- 73
- LR- 27; 65
- RL- 65
- Tabellengesteuerte LR(k)- 92
- Top-down- 65
- Top-down- mit Backtracking 46

Analyseaktionsfunktion 94

Analysematrix 62

Automat

- deterministischer endlicher 34
- endlicher 31
- vollständiger 34

Axiom 16

B

Backtracking 45

Backus-Naur-Form 16

- erweiterte 20

Baum

- abstrakter 12
- attributierter abstrakter 13

Blockniveau 116

Blockzähler 116

BNF 16

Bootstrapping 4

C

Chomsky-Hierarchie 19
Chomsky-Normalform 22
Compiler 3

E

Earley, Analyseverfahren von 60
EBNF 20
Eingabealphabet 31; 51
Erkennungsalgorithmus 60

G

Gesamtzeichenvorrat 16
GOTO-Funktion 92
Grammatik
 attributierte 115
 ϵ -freie 21
 eindeutige 25
 kontextfreie 16
 kontextsensitive 16
 linkslinere 16
 LL(k)- 91
 LR(k)- 73
 nicht eingeschränkte 16
 rechtslineare 16
 reduzierte 21
 reguläre 16
 Typ-0- 16
 Typ-1- 16
 Typ-2- 16
 Typ-3- 16
 zyklenfreie 21
Greibach-Normalform 22

H

Hauptsatz über LR(k)-Grammatiken 76

I

Identifikatorentabelle 115

Interpreter 8

K

k-Anfang 73; 76

k-Folgemenge 76

kanonisch 27

kanonische Kollektion 75

Kelleralphabet 51

Kellerautomat 50

1-Weg 51

deterministischer 57

Kellertransduktor, 1-Weg 58

Kettenproduktion 21

Konfiguration 51

konsistent 79

Kontrollwort 18

L

LALR 73

Laufzeitsystem 9

Linksableitung 28

Linksableitungsschritt 28

Linkskontext

reduzierter 65

terminaler 65

Linksstruktur 65

LL(k)-

Analyse 91

Grammatik 91

Obermenge 91

Sprache 91

look-ahead 73

LR(k)-

Analyse 73

Analyseaktionsfunktion 94

Analyseaktionstabelle 94

Auskunft 75
GOTO-Funktion 92
Information 75
Obermenge 73
Sprache 73
Zustände 94

M

Mehrphasen-Compilation 3
MMS 124
Münsteraner Mini-Sprache 124

N

Nachstruktur 65

O

Obermenge 68; 69; 70
 LL(k)- 91
 LR(K)- 73
Off-line-Zerteilungsalgorithmus 60
On-line-Algorithmus 60

P

Produktionensystem 16
Programm 18
 syntaktisch korrektes 18
 syntaktisches 18
Pumping-Lemma
 für kontextfreie Sprachen 19

Q

Quellsprache 3

R

Rechtsableitung 28
Rechtsableitungsschritt 28
Rechtskontext
 reduzierter 65

- terminaler 65
- Rechtsstruktur 65
- Reduktionsfolge 23
 - kanonische 28
- Reduktionsklasse 26; 67
- Reduktionsschritt 23
- reduzieren 17
 - in mindestens einem Schritt 17
 - unmittelbar 17
- reguläre Menge 30
- regulärer Ausdruck 30

S

- Satz 18
 - eindeutiger 25
- Satzform 18
- schleifeneinleitend 58
- schleifenfrei 58
- sequentieller Erkennungsalgorithmus 60
- sequentieller Zerteilungsalgorithmus 60
- Shiftklasse 67
- SLR 73
- Sprache 18
 - deterministische 57
 - kontextfreie 18
 - kontextsensitive 18
 - LL(k)- 91
 - LR(k) 73
 - reguläre 18
 - Typ-0- 18
 - Typ-1- 18
 - Typ-2- 18
 - Typ-3- 18
- Startsymbol 16
- Strukturbaum 24
- Symbol
 - nichtterminales 16
 - nützlich 21

nutzlos 21

terminales 16

T

T-Diagramm 3

Token 10

U

unmittelbar kanonischer 27

unwesentlich verschieden 27

V

Vorstruktur 65

Z

Zielcode 14

Zielsprache 3

Zwischencode 14