



# **Proseminar 1999/2000**

## **Thema**

**„Syntaktische Analyse von Programmiersprachen“**

## **Ausarbeitung von Kapitel II**

**„Kontextfreie Grammatiken“**

## **Autor**

**Michael Bergau**

Student der Informatik und Mathematik an der Universität des Saarlandes

## **Proseminarleitung**

**Prof. Dr. Wilhelm und Diplom Informatiker Andreas Kerren**

## **Vorwort**

Es handelt sich um einen im Rahmen eines Proseminar erstellten Vortrag mit dem Thema „Syntaktische Analyse von Programmiersprachen – Kontextfreie Grammatiken“. Es wird ein allgemeiner Überblick über die kontextfreien Sprachen geben, über die Grenzen der Notwendigkeit in der Syntaxanalyse hinaus. Die Folien des Vortrags entsprechen inhaltlich und strukturell völlig dieser Ausarbeitung. Die ergänzenden Kommentare und Erklärungen entsprechen in etwa dem Grundgerüst der Ausführung im Rahmen des Seminars. Das gesamte Proseminar hat als Grundlage das Buch

*„Parsing Theory. Vol.1 + Vol.2“*

von „Sippu, Soisalon-Soininen“. Jedoch wurden gerade in den einleitenden Vorträgen mehrmals auf Bücher aus der „theoretischen Informatik“, die die bearbeiteten Stoffe als Hauptthema haben, zurückgegriffen.

## Inhalts - Übersicht

Theoretische Grundlagen für die Syntaktische Analyse: „*Kontextfreie Grammatiken*“

### Kapitel 1. Allgemeine Grundlagen Seite 4 - 5

#### § 1.1 Definitionen

- Definition von Grammatik
- Definition von Sprache
- Definition von Satzform
- Definition von Satz

#### § 1.2 Allgemeine Spracheinteilung

- Chomsky – Hierarchie

### Kapitel 2. Kontextfreie Grammatik Seite 6 – 8

#### § 2.1 Kontextfrei

- Definition

#### § 2.2 $\epsilon$ - Sonderregelung

- Ausnahmeregelung
- Algorithmus zur  $\epsilon$  - Befreiung
- Beispiel

#### § 2.3 Syntaxanalyse

- Bedeutung
- Beispiel

### Kapitel 3. Grundstrukturen kontextfreier Grammatiken Seite 9 - 11

#### § 3.1 Ableitungen

- Ableitungsbäume
- Linksableitungen
- Rechtsableitungen

#### § 3.2 Mehrdeutigkeit

- Inhärente Mehrdeutigkeit
- Multiplizität
- Beispiel von Parikh

#### § 3.3 Nutzlose Nichtterminale

- Reduzierte Grammatik
- Produktive und erreichbare Nichtterminale
- Beispiel

### Kapitel 4. Normalformen kontextfreie Grammatiken Seite 12 - 16

#### § 4.1 Zusammenfassung

#### § 4.2 2 - Form

#### § 4.3 CNF

#### § 4.4 GNF

### Kapitel 5. Pumping – Lemma Seite 17 - 18

- Beweis
- Anwendungsbeispiel
- Ogden – Lemma

### Kapitel 6. Eigenschaften kontextfreier Grammatiken Seite 19 - 23

#### § 6.1 Abschlußigenschaften

- Beweise
- Gegenbeispiele

#### § 6.2 Entscheidbare Probleme

#### § 6.3 Unentscheidbare Probleme

#### § 6.4 Einschub: „*Komplexitätstheorie*“

### Kapitel 7. CYK – Algorithmus Seite 24 - 25

- Algorithmus
- Beispiel

## Kapitel 1. Allgemeine Grundlagen

Es handelt sich hierbei um die Bereitstellung der Theorie, die im weiteren Grundlage für alle Betrachtungen der Syntaxanalyse ist. Hierzu bedarf es einiger grundlegender Definitionen, aus der theoretischen Informatik. Es folgt eine spezialisierte Betrachtung der kontextfreien Sprachen.

Grundlegend wird erörtert: Was ist eine Grammatik und in welchem Zusammenhang steht diese mit der Sprachdefinition. Welche Beziehungen bestehen zwischen diesen einzelnen Elementen.

### § 1.1 Definitionen

#### Eine Grammatik

ist ein 4 – Tupel  $G = (V, \Sigma, P, S)$  mit

- $V$  Menge der Variablen (Nichtterminale)
- $\Sigma$  Terminalalphabet
- $P$  Menge der Produktionen, Ableitungsregeln
- $S \in V$  ist Startvariable

#### Sprache

$L(G) := \{w \in \Sigma^* \mid S \rightarrow^* w \text{ in } G\}$

#### Satzform

Ein Wort  $w \in (V \cup \Sigma)^*$ , das noch Nichtterminalsymbole enthält heißt Satzform

#### Satz

Ein Wort  $w \in \Sigma^*$ , das nur noch aus Terminalsymbolen besteht wird als Satz bezeichnet

### § 1.2 Allgemeine Spracheinteilung

Eine Sprache ist also nur eine Menge von Worten, deren Aufbau bestimmten Regeln entspricht. Diese Regeln sind in einer Grammatik zusammengefaßt. Es kann also zu einer Sprache verschiedene Grammatiken geben, die dieselbe Sprache erzeugen. Eine mögliche Einteilung der Sprachen in verschiedene Klassen liefert die von Noam Chomsky erstellte Chomsky 0 – 3 Hierarchie.

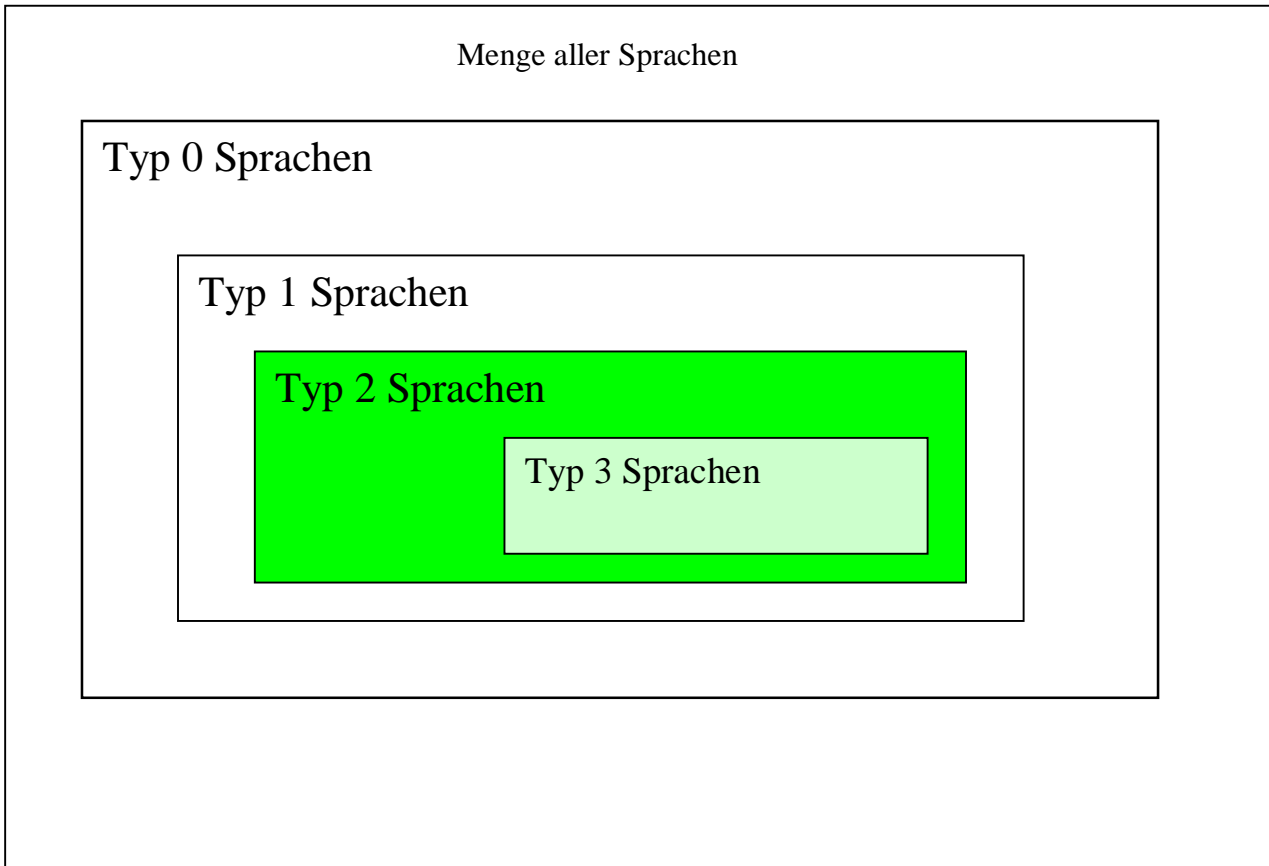
#### Chomsky – Hierarchie

Grammatik – Typ	Einschränkungen
Typ 0 oder Phasenstrukturgrammatik	Keinerlei Einschränkungen
Typ 1 oder kontextsensitive Grammatik	$\forall u \rightarrow v \in P \text{ gilt: }  u  \leq  v $
<b>Typ 2 oder kontextfreie Grammatik</b>	<b><math>\forall u \rightarrow v \in P \text{ gilt: } u \in V</math></b>
Typ 3 oder reguläre Grammatik	$\forall u \rightarrow v \in P \text{ gilt: } v \in (\Sigma \cup (\Sigma V))$

Mittels Einschränkungen entsteht eine Inklusionshierarchie von Typ 0 bis Typ 3 Sprachen. Im Bezug auf die Syntaxanalyse betrachtet man die kontextfreien Sprachen (Typ 2 Sprachen). Für die lexikalische Analyse verwendet man die regulären Sprachen. Jede Sprachklasse hat eigene spezifische Eigenschaften. Die Eigenschaften der kontextfreien Sprachen werden hier nun im weiteren erläutert und zusammengefaßt.

## Graphische Darstellung

Zur Verdeutlichung des Sachverhalt hier die graphische Umsetzung der Chomsky – Hierarchie. Diese Graphik zeigt nur daß die Menge aller regulären Sprachen in der Menge aller kontextfreien Sprachen liegen und so weiter. Man kann jedoch keine Aussage über die Mächtigkeit der einzelnen Sprachklassen machen. So ist die Menge aller Sprachen bei weitem die größte Klasse.



## Kapitel 2. Kontextfreie Grammatik

### § 2.1 Kontextfrei

#### Begriffserklärung „kontextfrei“

**Kontextfrei**  $\Leftrightarrow$  eine Variable kann unabhängig von ihrem Kontext ersetzt werden  
 Realisierung durch die Bedingung:  $\forall w_1 \rightarrow w_2 \in P \text{ gilt } w_1 \in V$

In einigen Büchern wird diese Sachverhalt mit der Monotonie der Ableitung bezeichnet. Mit jedem Ableitungsschritt wird die erzeugte Satzform länger oder bleibt gleich. Es kann auch nur noch eine kontinuierliche Änderung statt finden, da jeweils nur ein Nichtterminalsymbol mittels einer Produktion ersetzt wird.

#### Beispiel

Für Typ 1 Sprachen ist es möglich auf eine Ableitung der Form

$$uAv \rightarrow uav$$

zu stoßen, es wird  $A \in V$  nur zu  $a \in \Sigma$  abgeleitet in der Verbindung (im Kontext) mit  $u$  und  $v$

$\Rightarrow$  In kontextfreien Grammatiken kann  $A$  in jedem Kontext durch  $a$  ersetzt werden, wenn es eine Produktionsregel  $A \rightarrow a \in P$  gibt.

### § 2.2 $\epsilon$ - Sonderregelung

Diese erzwungene Konstruktion der Kontextfreiheit führt jedoch auch auf zwei Probleme.

1. Durch die Bedingung  $\forall u \rightarrow v \in P \text{ gilt: } |u| \leq |v|$  gibt es schon bei den Typ 1 Sprachen keine Möglichkeit mehr das leere Wort abzuleiten.  
 Eine Ableitung  $S \rightarrow \epsilon$  widerspricht mit  $|S| = 1 \leq 0 = |\epsilon|$  der gestellten Bedingung  
 $\Rightarrow G$  kontextfreie Grammatik  
 $\Rightarrow \exists (A \rightarrow \epsilon) \in P$   
 $\Rightarrow \epsilon \notin L(G)$
2. Die Produktionenmenge wird umfangreicher (siehe Beispiel), da es keine  $\epsilon$  - Produktionen mehr geben darf.

#### Ausnahmeregelung

Falls  $\epsilon \in L(G)$  erwünscht  $\Rightarrow (S \rightarrow \epsilon) \in P$  erlaubt,  $S$  nie auf einer rechten Ableitungsseite

Nun gibt es also wieder die Möglichkeit das leere Wort abzuleiten und somit ist die Vollständigkeit der kontextfreien Sprachen wieder gegeben. Es wird auch  $A \rightarrow \epsilon$  erlaubt, sogenannte  $\epsilon$ -Produktionen. Aber auch diese Regelung zerstört unsere Definition nicht, wie der folgende Satz mit Algorithmus zeigt.

#### Satz

Jede kontextfreie Grammatik kann  $\epsilon$  - frei gemacht werden

Ein mögliches Verfahren liefert der folgende Algorithmus.

## Algorithmus

### I. Bestimmung der nullierbaren Variablen

1.  $\exists A \rightarrow \varepsilon \Rightarrow A$  ist nullierbar
2.  $\exists B \rightarrow \alpha$  und  $\forall$  Variablen aus  $\alpha$  sind nullierbar  $\Rightarrow B$  ist nullierbar

### II. Ergänze Produktionen $A \rightarrow x_1 \dots x_n$ mit $A \rightarrow \alpha_1 \dots \alpha_n$ mit

1.  $x_i$  nicht nullierbar  $\Rightarrow \alpha_i = x_i$
2.  $x_i$  nullierbar  $\Rightarrow \alpha_i = x_i$  oder  $\alpha_i = \varepsilon$
3. Nicht alle  $\alpha_i$  sind  $\varepsilon$

## Beispiel zur $\varepsilon$ - Befreiung einer kontextfreien Grammatik

### Grammatik mit $\varepsilon$ - Produktionen

$$G = (V, \Sigma, P, S) \text{ mit } V = \{S, A, B, C\}$$

$$\Sigma = \{0, 1\}$$

$$P = \{S \rightarrow AB,$$

$$A \rightarrow 0C1 \mid 1C0 \mid \varepsilon,$$

$$B \rightarrow BB \mid 0,$$

$$C \rightarrow \mid 0A1 \mid 1A0 \mid \varepsilon\}$$

Erzeugte Sprache  
 $L(G) = \{w \in \{0,1\}^* \mid 0^+\}$

### $\varepsilon$ - Produktionen freie Grammatik

$$G = (V, \Sigma, P, S) \text{ mit } V = \{S, A, B, C\}$$

$$\Sigma = \{0, 1\}$$

$$P = \{S \rightarrow AB \mid B,$$

$$A \rightarrow 0A1 \mid 1A0 \mid 01 \mid 10,$$

$$B \rightarrow BB \mid 0\}$$

Man sieht sehr schön, welche Fälle abgefangen werden müssen um die Grammatik  $\varepsilon$  - frei zu machen. Es werden die Produktionen, so reduziert das die  $\varepsilon$  - Ableitungen, in ihrer Wirkung schon vorausgenommen werden, d.h. es entstehen mehr zusätzliche Fälle ( $A \rightarrow 01 \mid 10$ ) und die  $\varepsilon$  - Produktionen können dann ohne Einschränkung wieder gestrichen werden, waren also nur Hilfsmittel zur Konstruktion.

## § 2.3 Syntaxanalyse

### **Bedeutung für die Syntaxanalyse**

Höhere Programmiersprachen lassen geschachtelte Strukturen (wie u.a. Blöcke (begin ... end) und Klammernungen in imperativen, funktionalen und logischen Programmiersprachen) zu.

- ⇒ Ihre Syntax kann deswegen nicht mit regulären Ausdrücken spezifiziert und mit endlichen Automaten erkannt werden.
- ⇒ Zur Definition der Syntax höherer Programmiersprachen werden kontextfreie Grammatiken verwendet, zur Syntaxanalyse von Programmen werden Kellerautomaten eingesetzt.

Die Theorie der Kellerautomaten wird in diesem Vortrag weder ausgebreitet noch erläutert. Der Kellerautomat und seine große Bedeutung in der Syntaxanalyse wird in einer eigenständigen Ausarbeitung abgehandelt. Dort werden dann die Grundlagen der kontextfreien Sprachen verwandt.

### Beispielgrammatik

#### **Kontextfreie Grammatik für Mini – SML**

$G = (V, \Sigma, P, S)$  mit  $V = \{\text{EXP, OPR, PAT}\}$   
 $\Sigma = \{ \text{id, const, if, (, ), let, val, =, in, end, fn, =>, rec, [, ], ::, nil, +, -, *, div, <, >, <=, >=, \_ , , } \}$   
 $P = \{ \text{EXP} \rightarrow \text{id}$   
 $\quad \text{const}$   
 $\quad \text{EXP OPR EXP}$   
 $\quad \text{if EXP then EXP else EXP}$   
 $\quad \text{let val PAT = EXP in EXP end}$   
 $\quad (\text{EXP, EXP, ..., EXP})$   
 $\quad \text{fn PAT} \Rightarrow \text{EXP}$   
 $\quad \text{let val rec id = fn PAT} \Rightarrow \text{EXP in EXP end}$   
 $\quad [\text{EXP, ..., EXP}]$   
 $\quad \text{nil}$   
 $\quad \text{EXP} :: \text{EXP},$   
 $\quad \text{OPR} \rightarrow + \mid - \mid * \mid \text{div} \mid < \mid > \mid <= \mid >= \mid = ,$   
 $\quad \text{PAT} \rightarrow \text{id} \mid \text{const} \mid \_ \mid (\text{PAT, ..., PAT}) \}$   
 $S = \{\text{EXP}\}$

An dieser einfachen Programmiersprache ist nun gut zu erkennen, welche Erweiterungen die Definition der kontextfreien Grammatiken im Gegensatz zu den regulären Grammatiken zulassen. Die strukturierte Klammerung wie z.B. bei if EXP then EXP else EXP ist mit einer regulären Grammatik nicht möglich. Es gibt nur Produktionen der Form  $A \rightarrow a$  und  $A \rightarrow aB$ .

- ⇒ kontextfreie Grammatiken bilden die kleinste Menge an Sprachen, in denen eine Klammerung möglich ist (Programmiersprachen); deshalb dienen sie als Grundlage für die Syntaxanalyse

## Kapitel 3. Grundstrukturen kontextfreier Grammatiken

### § 3.1 Ableitungen

#### Ableitungsbäume

Die kontextfreien Grammatiken erzeugen in ihren Ableitungen eine Baumstruktur. (siehe Abbildung unten)

#### Links- und Rechtsableitungen

##### Definition

Sei  $S \xrightarrow{*} w_1 \xrightarrow{*} w_n$  eine Ableitung (gemäß einer kontextfreien Grammatik  $G$ )

Die Ableitung ist eine Linksableitung (bzw. Rechtsableitung)

$\Leftrightarrow$  es für alle  $i \geq 1$  Symbolfolgen  $x_i, y_i$  und  $z_i$  gibt mit:

- $w_i = x_i A y_i$ ,
- $w_{i+1} = x_i z_i y_i$ ,
- $A \rightarrow z_i$  ist eine Produktion
- $x_i$  (bzw.  $y_i$ ) in  $\Sigma^*$  (d.h.  $x_i$  (bzw.  $y_i$ ) enthält nur Terminalsymbole).

##### Satz

Sei  $G = (V, \Sigma, P, S)$  eine kontextfreie Grammatik. Für alle  $\gamma \in V^*$  gilt

$$\{w \in \Sigma^* \mid \gamma \xrightarrow{*}_L w\} = \{w \in \Sigma^* \mid \gamma \rightarrow^* w\} = \{w \in \Sigma^* \mid \gamma \xrightarrow{*}_R w\}$$

##### Beachte

Zu einem gegebenen Ableitungsbaum existiert genau eine Linksableitung und Rechtsableitung

#### Beispiele zur Links- und Rechtsableitung

a. Kontextfreie Grammatik für die regulären Ausdrücke der Kleen ´schen Algebra

$$G = (\{S\}, \{a, b, (, ), \epsilon, \emptyset, +, \cdot, *\}, P, S) \quad \text{mit } P = \{ S \rightarrow \emptyset \mid a \mid b \mid \epsilon \mid (S+S) \mid (S \cdot S) \mid (S^*) \}$$

$  \begin{array}{c}  S \\  / \quad \backslash \\  (S + S) \\  / \quad \backslash \\  (S \cdot S) \quad a \\    \quad   \\  a \quad b  \end{array}  $	$  \begin{array}{c}  S \\  / \quad \backslash \\  (S + S) \\  / \quad \backslash \\  (S \cdot S) \quad a \\    \quad   \\  a \quad b  \end{array}  $
Linksableitung	Rechtsableitung

b. Kontextfreie Grammatik für  $L = \{ w = a^* \mid |w| \geq 1 \}$

$$G = (\{S\}, \{a\}, P, S) \quad \text{mit } P = \{ S \rightarrow SS \mid a \}$$

$  \begin{array}{c}  S \\  / \quad \backslash \\  S \quad S \\  / \quad \backslash \quad \backslash \\  S \quad S \quad a \\    \quad   \\  a \quad a  \end{array}  $	$  \begin{array}{c}  S \\  / \quad \backslash \\  S \quad S \\  / \quad / \quad \backslash \\  a \quad S \quad S \\    \quad   \\  a \quad a  \end{array}  $
Linksableitung	Rechtsableitung

Diese statische Betrachtung der Ergebnisse bringt einem nicht die Unterschiede in der Konstruktion vor Augen. Es soll nur zeigen, daß es für ein Wort verschiedene Ableitungsbäume geben kann. Dies führt zu folgender Betrachtung.

## § 3.2 Mehrdeutigkeit

Das obige Beispiel führt uns auf folgende grundlegende Definitionen und Begriffsbildungen

### Definition

Eine Grammatik  $G$  heißt **mehrdeutig**

⇔ es gibt für ein und dasselbe **Wort  $x$  mindestens zwei verschiedene Syntaxbäume**

⇔ es gibt **zwei voneinander verschiedene Linksableitungen** für ein Wort  $x$

⇔ es gibt **zwei voneinander verschiedene Rechtsableitungen** für ein Wort  $x$

### Definition

Eine Sprache  $L$  heißt **mehrdeutig**

⇔ es gibt **nur mehrdeutige Grammatiken  $G$**  für  $L$

Man nennt die Sprache dann auch **inhärent mehrdeutig**.

### Bemerkung

**Multiplizität** gibt die Anzahl der möglichen Linksableitungen eines Wortes in einer Grammatik an.

### Beispiel

Eine inhärent mehrdeutige, kontextfreie Sprache

$$L = \{a^i b^j c^k \mid i = j \text{ oder } j = k\} \quad (\text{Parikh (1966)})$$

Es gibt nun mehrere Beweisansätze. Hier nun die Beweisideen von Parikh, die die Abgeschlossenheit der komplexen Sprachen verwendet. Diese Eigenschaft der kontextfreien Sprachen werden wir in Kapitel 6 im einzelnen betrachten und beweisen.

### Beweis zu dem Beispiel von Parikh

#### Behauptung

$L = \{a^i b^j c^k \mid i = j \text{ oder } j = k\}$  ist eine inhärent mehrdeutige, kontextfreie Sprache

#### Beweisidee

- $L$  ist Vereinigung von  $L_1 = \{a^i b^i \mid i \geq 0\} \{c\}^*$  und  $L_2 = \{a\}^* \{b^i c^i \mid i \geq 0\}$
- $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 0\}$

#### Vermutung

Es gibt für die Wörter die im Schnitt liegen mehrere Ableitungsbäume

- Suche für eine beliebige kontextfreie Grammatik  $G$  für  $L$  ein  $m \in \mathbb{N}$  so daß  $a^m b^m c^m$  zwei Ableitungsbäume besitzt

#### Warum gibt es diese verschiedenen Syntaxbäume?

- Ableitungsbaum A basiert auf den Regeln, die testen ob  $|a| = |b|$
- Ableitungsbaum B basiert auf den Regeln, die testen ob  $|b| = |c|$

Anschaulich ist dies klar, da in jeder Grammatik unabhängig von einander Wörter der Art  $a^m b^m c^m$  erzeugt werden (abgeleitet werden). Da dies unter verschiedenen Bedingungen geschieht ist anzunehmen, daß diese Ableitungen verschiedenen Charakter haben. Diese Vermutung beweist sich als richtig bei einer Analyse der verschiedenen Ableitungsbäume.

### § 3.3 Nutzlose Nichtterminale

#### Entfernung nutzloser Nichtterminale

Oftmals hat man eine Sprache vorgegeben und sucht dann eine kontextfreie Grammatik, die diese Sprache erzeugt. Meistens entsteht bei dieser Grammatiksuche, mittels verschiedener Algorithmen oder durch die Realisierung gegebener Einschränkungen, eine nicht sehr effektive Sprache. Nun gibt uns folgende Definition eine Möglichkeit Grammatiken zu optimieren und in ihrer Größe zu reduzieren, ohne daß sich die erzeugte Sprache ändert.

#### Definition

Nichtterminale heißen nützlich, falls

1. eine Ableitung zu einem Terminalwort existiert (produktive Nichtterminale)

$$V_{\text{Nutz}_1} = \{ U \in V \mid \exists U \rightarrow^* w \text{ mit } w \text{ Terminalwort} \}$$

2. eine Ableitung von S zu dem Nichtterminalsymbol existiert (erreichbare Nichtterminale)

$$V_{\text{Nutz}_2} = \{ U \in V_{\text{Nutz}_1} \mid \exists S \rightarrow \alpha U \beta \text{ mit } \alpha, \beta \in (V \cup \Sigma)^* \setminus \{U\} \}$$

$$\Rightarrow V_{\text{Nutz}} = V_{\text{Nutz}_2}$$

#### Definition

Eine **kontextfreie Grammatik** G heißt **reduziert**, wenn sie nur **produktive** und **erreichbare Nichtterminale** enthält

#### Warnung $\eta$

Aufstellen von  $V_{\text{Nutz}_1}$  und  $V_{\text{Nutz}_2}$  nur in dieser Reihenfolge möglich

#### Bemerkung

Weitere Vereinfachung der Grammatik durch Entfernung von Kettenproduktionen

#### Beispiel Elemination nutzloser Variablen

##### Unbereinigte Grammatik

$G = (V, \Sigma, P, S)$  mit  $V = \{S, A, B, C, D, E, F, H\}$ ,  $\Sigma = \{a, b\}$

$$P = \{ S \rightarrow AB \mid D, \\ A \rightarrow BD \mid EF \mid H, \\ B \rightarrow ED \mid b, \\ C \rightarrow AB, \\ E \rightarrow B, \\ D \rightarrow F, \\ H \rightarrow a \}$$

##### Bereinigte Grammatik

$G_b = (V, \Sigma, P, S)$  mit  $V = \{S, A, B, H\}$ ,  $\Sigma = \{a, b\}$

$$P = \{ S \rightarrow AB, \\ A \rightarrow H \\ H \rightarrow a, \\ B \rightarrow b \}$$

erzeugte Sprache  $L = \{ab\}$

Es werden alle Produktionen entfernt, die keine Terminalableitung gestatten und die vom Startsymbol aus nicht erreichbar sind. Erhalten bleiben jedoch die erwähnten Kettenproduktionen. Im Beispiel ist das Nichtterminalsymbol H ein nicht benötigtes Glied einer solchen Kettenproduktion. Eine direkte Ableitung von  $A \rightarrow a$  erzeugt die gleiche Sprache. Im Beispiel erkennt man auch die Notwendigkeit dieser Umformung, denn aus der ersten Grammatik läßt sich die erzeugte Sprache nicht ohne weiteres erkennen.

## Kapitel 4. Normalformen kontextfreie Grammatiken

### § 4.1 Zusammenfassung

#### Verschiedenen Formen für kontextfreie Grammatiken

Es gibt verschiedene Grammatikstrukturen innerhalb der kontextfreien Sprachen um diese systematischer aufzufassen. Hier wollen wir nun die wichtigsten und bekanntesten Formen einführen.

**1. (E)BNF (erweiterte Backus Naur Form)**

Beschränkung auf folgende Produktionen

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n \quad \text{d.h.} \quad A \rightarrow \alpha_1, \dots, A \rightarrow \alpha_n$$

$$A \rightarrow \alpha [\beta] \gamma \quad \text{d.h.} \quad A \rightarrow \alpha \gamma \text{ und } A \rightarrow \alpha \beta \gamma$$

$$A \rightarrow \alpha \{ \beta \} \gamma \quad \text{d.h.} \quad A \rightarrow \alpha \gamma, A \rightarrow \alpha \beta \gamma, B \rightarrow \beta \text{ und } B \rightarrow \beta B$$

**2. Kanonische 2 - Form Grammatiken**

Beschränkung auf folgende Produktionen

$$A \rightarrow BC$$

$$A \rightarrow B$$

$$A \rightarrow a$$

**3. CNF (Chomsky Normalform)**

Beschränkung auf folgende Produktionen

$$A \rightarrow BC$$

$$A \rightarrow a$$

**4. GNF (Greibach Normalform)**

Beschränkung auf folgende Produktionen

$$A \rightarrow aB_1B_2 \dots B_k \quad \text{für } k \geq 0$$

Jede dieser Grammatikformen hat ihre eigenen Vor- und Nachteile. Es wäre nun wünschenswert, wenn all diese Strukturen äquivalent in einander umgewandelt werden könnten, daß genau dies geht wollen wir nun in den nächsten Schritten zeigen.

Die (E)BNF ist nur eine andere Schreibweise mit Meta - Regeln, sie ist schon nach Definition äquivalent mit den anderen Formen.

Es bleiben also nur noch die kanonische 2 - Form, CNF und die GNF. Hierbei zeigen wir, daß die kanonische 2 - Form Grammatik in eine CNF - Grammatik und weiter in eine GNF - Grammatik umgewandelt werden kann.

Zu den einzelnen Strukturen bleibt zu sagen, daß die CNF in vielen Beweisen hilfreich ist, da ihre Ableitungsbäume eine für Anwendungen sehr nützliche Binärstruktur aufweisen. Die GNF ist eine Erweiterung der regulären Sprachen. Setzt man  $k \leq 1$  so erhält man die Typ 2 Sprachen.

Es sollen nicht nur die Beweise für die Existenz der einzelnen Umformungsalgorithmen geführt werden, sondern es werden drei Methoden aufgezeigt, wie man solche Algorithmen angeben kann.

1. Direkt Implementation, indem man die allgemeine Ergebnis - Grammatik angibt
2. Als Transformationsverfahren mit einer angegebenen Schrittfolge ohne genaue Verlaufsangabe (Schleifendurchläufe, Komplexität usw.)
3. Als Algorithmus in einer Pseudo - Programmiersprache mit genauer Schleifenkomplexität

## § 4.2 Kanonische 2 – Form

### Kanonische 2-Form Grammatik für jede kontextfreie Grammatik

kontextfreie Grammatik  $G \Leftrightarrow \exists$  kanonische 2 – Form Grammatik  $G'$  mit  $L(G) = L(G')$

#### Beweisidee

Durch Produktionentransformationen erreicht man die gewünschten Strukturen. Hierbei muß man dafür sorgen, daß jede Produktion von der Form  $A \rightarrow X_1 \dots X_m$  oder  $A \rightarrow a$  ist. Dann werden die Produktionen  $A \rightarrow X_1 \dots X_m$  nach folgendem Verfahren transformiert. Alle anderen Produktionen haben schon eine der gewünschten Formen.

$$\begin{array}{l}
 A \rightarrow X_1 \dots X_m (m \geq 2) \Rightarrow \quad A \rightarrow [X_1] [X_2 \dots X_m] \\
 \quad \quad \quad \quad \quad \quad \quad [X_2 \dots X_m] \rightarrow [X_2] [X_3 \dots X_m] \\
 \quad \quad \quad \quad \quad \quad \quad \vdots \quad \quad \quad \quad \quad \quad \quad \vdots \\
 \quad \quad \quad \quad \quad \quad \quad [X_{m-1} X_m] \rightarrow [X_{m-1}] [X_m] \\
 \text{und} \\
 \quad \quad \quad \quad \quad \quad \quad [X_i] \rightarrow X_i
 \end{array}$$

Es ergibt sich dann folgende Grammatik  $G'$

$$V' = V \cup \{[\beta] \mid \beta \in V^+ \text{ und } (A \rightarrow \alpha\beta) \in P\} \cup \{[X] \mid X \in V \text{ und } (A \rightarrow \alpha X\beta) \in P\}$$

$$\begin{aligned}
 P' = & \{A \rightarrow \alpha \mid (A \rightarrow \alpha) \in P \text{ und } |\alpha| \leq 1\} \\
 & \cup \{A \rightarrow [X][\beta] \mid A \rightarrow X\beta \in P, X \in V, \beta \in V^+\} \\
 & \cup \{[X\gamma] \rightarrow [X][\gamma] \mid [X\gamma] \in V', X \in V \text{ und } \gamma \in V^+\} \\
 & \cup \{[X] \rightarrow X \mid [X] \in V' \text{ und } X \in V\}
 \end{aligned}$$

**Ziel**

$A \rightarrow a$

$A \rightarrow B$

$A \rightarrow BC$

Dies ist nicht die einfachste Grammatik in kanonischer 2 – Form, jedoch ist dies für den Beweis auch nicht notwendig. Da es nur um die Existenz einer solchen Grammatik geht.

### Beispiel (Kanonische 2-Form Grammatik)

#### kontextfreie Grammatik

$$\begin{aligned}
 G = (V, \Sigma, P, S) \text{ mit } V = \{S, A, B\}, \Sigma = \{0, 1\} \text{ und } P = \{ & S \rightarrow A \mid B \\
 & B \rightarrow 0B1 \mid 01 \\
 & A \rightarrow 1A0 \mid 10 \}
 \end{aligned}$$

$$\Rightarrow L(G) = \{w \in a^n b^n \mid a, b \in \{0, 1\}, n > 0\}$$

#### Umgeformte Grammatik in kanonischer 2 – Form

$$G = (V^*, \Sigma, P^*, S) \text{ mit } V^* = \{S, A, B, N, E, C, D\}, \Sigma = \{0, 1\} \text{ und}$$

$$\begin{aligned}
 P^* = \{ & S \rightarrow A \mid B \\
 & N \rightarrow 0 \\
 & E \rightarrow 1 \\
 & B \rightarrow NE \mid NC \\
 & A \rightarrow ED \mid EN \\
 & D \rightarrow AN \\
 & C \rightarrow BE \}
 \end{aligned}$$

**Ziel**

$A \rightarrow a$

$A \rightarrow B$

$A \rightarrow BC$

## § 4.3 CNF

Betrachtet man nun die Ergebnisgrammatik aus § 4.2, so erkennt man, daß der nächste Schritt zur Chomsky Normalform, nur noch in der Elimination der Produktionen  $A \rightarrow B$  besteht.

### CNF für jede kontextfreie Grammatik

G kontextfreie Grammatik  $\Leftrightarrow \exists$  Grammatik  $G'$  in CNF mit  $L(G) = L(G')$

### Transformationsverfahren

Transformiere G in eine kanonische 2 – Form Grammatik

$\Rightarrow A \rightarrow BC$   
 $A \rightarrow B$   
 $A \rightarrow a$

<u>Ziel</u>
$A \rightarrow a$
$A \rightarrow BC$

$\Rightarrow$  Eliminiere Produktionen der Form  $A \rightarrow B$  (Einheitsableitungen)

1. Ergänze P mit  $A \rightarrow B' C'$  falls  $A \rightarrow BC \in P$  mit  $B \xrightarrow{*} B'$  und  $C \xrightarrow{*} C'$
2. Lösche alle Produktionen der Form  $A \rightarrow B$

Man ergänzt also ganz analog zur Entfernung der  $\epsilon$  - Produktionen alle Einheitsableitungen, indem man sie in eine Produktion der Form  $A \rightarrow BC$  einsetzt und in die Produktionsmenge aufnimmt.

### Beispiel (CNF – Grammatik)

#### Umgeformte Grammatik in kanonischer 2 – Form

$G = (V, \Sigma, P, S)$  mit  $V = \{S, A, B, N, E, C, D\}$ ,  $\Sigma = \{0, 1\}$  und

$$P = \{ S \rightarrow A \mid B$$

$$N \rightarrow 0$$

$$E \rightarrow 1$$

$$B \rightarrow NE \mid NC$$

$$A \rightarrow ED \mid EN$$

$$D \rightarrow AN$$

$$C \rightarrow BE \}$$

#### Grammatik in CNF

$G = (V^*, \Sigma, P^*, S)$  mit  $V^* = \{S, A, B, N, E, C, D\}$ ,  $\Sigma = \{0, 1\}$  und

$$P^* = \{ S \rightarrow ED \mid EN \mid NE \mid NC$$

$$N \rightarrow 0$$

$$E \rightarrow 1$$

$$B \rightarrow NE \mid NC$$

$$A \rightarrow ED \mid EN$$

$$D \rightarrow AN$$

$$C \rightarrow BE \}$$

<u>Ziel</u>
$A \rightarrow a$
$A \rightarrow BC$

Natürlich kann die Grammatik durch diese Operation um einiges anwachsen, da mehr Fälle aufgeführt werden müssen. Die Mächtigkeit steigt mit der Anzahl der Einschränkungen stetig an. Dieser Effekt wird bei der Umformung in Greibach - Normalform noch deutlicher werden.

## § 4.4 GNF

### GNF für jede kontextfreie Grammatik

G kontextfreie Grammatik  $\Leftrightarrow \exists$  Grammatik  $G'$  in CNF mit  $L(G) = L(G')$

In drei Schritten werden wir nun eine Grammatik von CNF in GNF umformen. Im ersten Schritt schaffen wir eine künstliche beliebige, aber feste Ordnung auf der Menge der Nichtterminale und formen die Produktionen so um, damit wir eine sortierte Folge erhalten ( $A_i \rightarrow A_j\alpha$  mit  $i < j$ ).

#### Algorithmisches Verfahren zur Umformung einer CNF Grammatik

1. Durchnummerierung der Variablen von  $A_1$  bis  $A_m$

Modifiziere Regeln:  $A_i \rightarrow A_j\alpha$  mit  $i < j$

FOR  $i = 1$  TO  $m$  DO

FOR  $j = 1$  TO  $i - 1$  DO

FOR  $\forall A_i \rightarrow A_j\alpha \in P$  DO

Seien  $A_j \rightarrow \beta_1 | \dots | \beta_n$  alle  $A_j$  - Regeln

Füge hinzu:  $A_i \rightarrow \beta_1\alpha | \dots | \beta_n\alpha$

Streiche:  $A_i \rightarrow A_j\alpha$

END

END

IF  $\exists A_i \rightarrow A_j\alpha$  THEN

Ersetze  $A_i \rightarrow A_i\alpha_1 | \dots | A_i\alpha_k | \beta_1 | \dots | \beta_k$

Durch  $A_i \rightarrow \beta_1 | \dots | \beta_k$  und  $A_i \rightarrow \beta_1 B_i | \dots | \beta_k B_i$

$B_i \rightarrow \alpha_1 | \dots | \alpha_k$  und  $B_i \rightarrow \alpha_1 B_i | \dots | \alpha_k B_i$

END

END

Rekursive Abarbeitung der entstanden Produktionen von  $A_{m-1}$  bis  $A_1$  bringt alle Regeln auf die Form, daß sie mit einem Terminalzeichen beginnen. Dies führt auf den folgenden Algorithmus.

2. Transformiere alle  $A_i$  - Regeln, so daß alle Regeln mit einem Terminalzeichen beginnen

-  $A_m$  - Regeln beginnen schon mit einem Terminalzeichen

- Rekursive Umformung

FOR  $i = m - 1$  DOWNTO  $1$  DO

FOR  $\forall A_i \rightarrow A_j\alpha \in P, j > i$  DO

Seien  $A_j \rightarrow \beta_1 | \dots | \beta_n$  alle  $A_j$  - Regeln

Füge hinzu:  $A_i \rightarrow \beta_1\alpha | \dots | \beta_n\alpha$

Streiche:  $A_i \rightarrow A_j\alpha$

END

END

$\Rightarrow$  Alle  $A_i$  - Regeln sind in Greibach - Normalform

<p><b>Ziel</b>  <math>A \rightarrow aB_1..B_k</math>  mit <math>k \geq 0</math></p>
---

Nun bleibt nur noch die Umformung der neu erzeugten Produktionen in analoger Weise wie oben.

3. Umformung der Regeln für die  $B_i$  - Variablen in GNF

FOR  $i = 1$  TO  $m$  DO

FOR  $\forall B_i \rightarrow A_j\alpha \in P$  DO

Seien  $A_j \rightarrow \beta_1 | \dots | \beta_n$  alle  $A_j$  - Regeln in GNF

Füge hinzu:  $B_i \rightarrow \beta_1\alpha | \dots | \beta_n\alpha$

Streiche:  $B_i \rightarrow A_j\alpha$

END

END

## Beispiel (GNF – Grammatik)

### Grammatik in CNF

$G = (V, \Sigma, P, S)$  mit  $V = \{S, A, B, N, E, C, D\}$ ,  $\Sigma = \{0, 1\}$  und

$$P = \{ S \rightarrow ED \mid EN \mid NE \mid NC$$

$$N \rightarrow 0$$

$$E \rightarrow 1$$

$$B \rightarrow NE \mid NC$$

$$A \rightarrow ED \mid EN$$

$$D \rightarrow AN$$

$$C \rightarrow BE \}$$

### Grammatik in GNF

$G = (V^*, \Sigma, P^*, S)$  mit  $V^* = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ ,  $\Sigma = \{0, 1\}$  und

$$P^* = \{ A_1 \rightarrow 1A_3 \mid 1A_4 \mid 0A_2 \mid 0A_5$$

$$A_2 \rightarrow 1$$

$$A_3 \rightarrow 1A_3A_4 \mid 1A_4A_4$$

$$A_4 \rightarrow 0$$

$$A_5 \rightarrow 0A_2A_2 \mid 0A_5A_2$$

$$A_6 \rightarrow 0A_2 \mid 0A_5$$

$$A_7 \rightarrow 1A_3 \mid 1A_4 \}$$

**Ziel**

$A \rightarrow aB_1..B_k$   
mit  $k \geq 0$

# Kapitel 5. Pumping – Lemma

Das nun folgende Lemma ermöglicht es zu zeigen, daß eine Sprache nicht kontextfrei ist. Man kann hiermit nicht zeigen, daß eine Sprache kontextfrei ist (vgl. Beispiel).

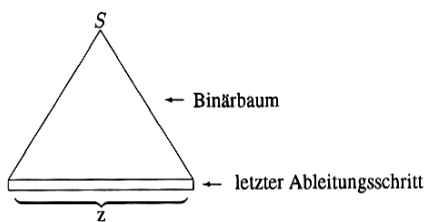
## Pumping – Lemma für kontextfreie Grammatiken

Sei  $L$  eine kontextfreie Sprache. Dann gibt es eine Zahl  $n \in \mathbb{N}$ , so daß sich alle Wörter  $z \in L$  mit  $|z| \geq n$  zerlegen lassen in  $z = uvwxy$  mit folgenden Eigenschaften

1.  $|vx| \geq 1$
2.  $|vwx| \leq n$
3. Für alle  $i \geq 0$  gilt:  $uv^iwx^iy \in L$

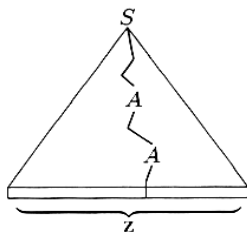
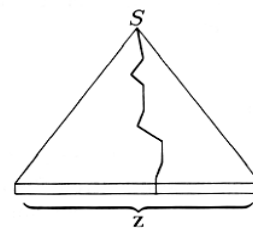
### Graphischer Darstellung der Beweisidee

$G$  in CNF,  $k = |V|$ , wähle  $n = 2^k$



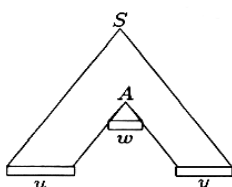
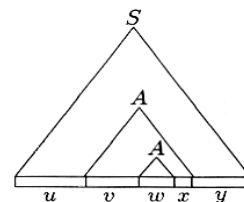
Man erhält als Ableitungsbaum einen Baum mit Binärstruktur, bis auf die letzte Ableitung (Terminalableitung)

Binärbaum hat  $\geq 2k$  Blätter und somit mindestens einen Pfad der Länge  $\geq k$ . Mit der Startvariablen ergibt sich also ein Pfad der Länge  $\geq k + 1$ .



Da die Grammatik nur  $k$  Variablen besitzt, muß mindestens eine Variable doppelt vorkommen

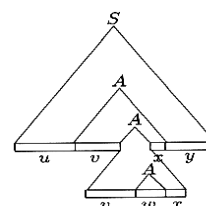
Arbeitet man diesen Pfad von den Blättern des Baumes auf so erhält man folgende Unterteilung von  $z$  in  $uvwxy$



Nun kann man bei der ersten Variable den Ableitungsbaum der Variable zwei anhängen  
Pumpen mit  $i = 0 \Rightarrow uvw \in L$

Analog erhält man natürlich bei umgekehrtem vorgehen weitere Wörter gemäß dem Pumpinglemma

z. B.  $i = 2 \Rightarrow uv^2wx^2y \in L$



## Beispiel für die Anwendung des Pumping - Lemma

$L = \{a^m b^m c^m \mid m \geq 1\}$  ist nicht kontextfrei

### Beweis

Man kann mit dem Pumping - Lemma also nur beweisen, daß eine Sprache nicht kontextfrei ist. Es gibt jedoch keine Möglichkeit zu zeigen, daß die gegebene Sprache kontextfrei ist. Somit ergibt sich der Beweis als Widerspruchsfolgerung. Man nimmt also an, daß eine Sprache kontextfrei sei und zeigt, daß die Bedingungen des Pumping - Lemma nicht erfüllt werden. daraus läßt sich dann der Widerspruch ableiten, und man erhält, daß die Sprache nicht kontextfrei ist.

**Annahme:**  $L$  ist kontextfrei

- Wähle Zerlegung  $z = a^n b^n c^n$  mit  $n$  Pumping – Zahl  
 $\Rightarrow |z| = 3n \geq n, z = uvwxy$
- $vx$  kann nicht aus  $a$ 's,  $b$ 's und  $c$ 's bestehen, da  $|vwx| \leq n$  nach Bedingung 2
- $vx$  ist nicht leer nach Eigenschaft 1
- $uv^0 wx^0 y = uwy \in L$  nach Eigenschaft 3 mit  $i = 0$   
 $\Rightarrow uwy$  hat nicht die Form  $a^m b^m c^m$   $\eta$  Widerspruch da  $uwy \notin L$   
 $\Rightarrow L$  ist nicht kontextfrei

### Anmerkung

Es gibt das Pumping – Lemma noch in einer allgemeineren Form dem Ogden - Lemma. Es sei nur zur Vollständigkeit erwähnt, da für unsere Anwendungen das Pumping - Lemma ausreicht.

## Ogden – Lemma

Für jede kontextfreie Sprache  $L$  gibt es eine Konstante  $n \in \mathbb{N}$ , so daß für jedes Wort  $z \in L$  mit  $|z| \geq n$  folgende Aussage gilt:

- Markiere  $\geq n$  Buchstaben in  $z$   
 $\Rightarrow \exists$  Zerlegung  $z = uvwxy$  mit
1.  $\exists$  mindestens ein markierter Buchstabe in  $vx$
  2.  $\exists$  höchstens  $n$  markierte Buchstaben in  $vwx$
  3.  $\forall i \geq 0$  gilt:  $uv^i wx^i y \in L$

Das Ogden - Lemma betrachtet also nur Wortausschnitte. Markiert man bei Ogden alle Buchstaben eines gegebenen Wortes, so erhalten wir wieder das Pumping – Lemma.

## Kapitel 6. Eigenschaften kontextfreier Grammatiken

### § 6.1 Abschlußeigenschaften

#### Definition

Sei  $X$  eine Klasse von Mengen (Sprachen). Dann heißt  $X$  Schnitt- (Vereinigungs-, Komplement-, Produkt-, Stern-) abgeschlossen, falls aus  $A \in X$  und  $B \in X$  folgt  $A \cap B \in X$

(bzw.  $A \cup B \in X$ ,  $\overline{A} \in X$ ,  $AB \in X$ ,  $A^* \in X$ )

Kontextfreie Sprachen sind **abgeschlossen** unter

- **Vereinigung**
- **Produkt**
- **Stern**

Kontextfreie Sprachen sind **nicht abgeschlossen** unter

- **Schnitt**
- **Komplement**

Die Beweise ergeben sich direkt durch Angabe der gesuchten Sprache oder durch einfache Gegenbeispiele. Hierbei möge sich der Leser an unser Beispiel von Pumping – Lemma (Kapitel 5) erinnern. Es entstehen nicht immer die einfachsten Grammatiken, jedoch ist dies auch nicht notwendig für den Beweise der einzelnen Aussagen.

#### Beweise zu den Abschlusseigenschaften

##### **Vereinigung**

$G_1 = (V_1, \Sigma, P_1, S_1)$  und  $G_2 = (V_2, \Sigma, P_2, S_2)$  sind kontextfreie Grammatiken mit  $V_1 \cap V_2 = \emptyset$   
 $\Rightarrow G = (V, \Sigma, P, S)$  ist kontextfreie Grammatik  
 mit  $V = V_1 \cup V_2 \cup \{S\}$  und  $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$

##### **Produkt**

$G_1 = (V_1, \Sigma, P_1, S_1)$  und  $G_2 = (V_2, \Sigma, P_2, S_2)$  sind kontextfreie Grammatiken mit  $V_1 \cap V_2 = \emptyset$   
 $\Rightarrow G = (V, \Sigma, P, S)$  ist kontextfreie Grammatik  
 mit  $V = V_1 \cup V_2 \cup \{S\}$  und  $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$

##### **Stern**

$G_1 = (V_1, \Sigma, P_1, S_1)$  ist kontextfreie Grammatik  
 $\Rightarrow G = (V, \Sigma, P, S)$  ist kontextfreie Grammatik  
 mit  $V = V_1 \cup \{S\}$  und  $P = P_1 \cup \{S \rightarrow \epsilon, S \rightarrow S_1, S_1 \rightarrow S_1 S_1\} - \{S_1 \rightarrow \epsilon\}$

#### Gegenbeispiele zu den Abschlußeigenschaften

##### **Schnitt (Gegenbeispiel)**

$L_1 = \{a^i b^j c^j \mid i, j > 0\}$  und  $L_2 = \{a^i b^i c^j \mid i, j > 0\}$   
 $\Rightarrow L_1 \cap L_2 = \{a^i b^i c^i \mid i > 0\}$  ist nicht kontextfrei (siehe Pumping – Lemma)

##### **Komplement (Mit Widerspruch)**

###### **Annahme**

Die kontextfreien Sprachen sind abgeschlossen unter Komplementbildung

$$\Rightarrow L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \quad (\text{Regeln von de Morgan})$$

$\Rightarrow$  kontextfreie Sprachen sind abgeschlossen unter Schnittbildung  $\Rightarrow$  Widerspruch

## § 6.2 Entscheidbare Probleme

### Wortproblem

Stellt die Frage ob für ein  $x \in \Sigma^*$  gilt  $x \in L(G)$  oder  $x \notin L(G)$

⇒ **Entscheidbar** da das Wortproblem schon für Typ 1 Sprachen entscheidbar ist

### Leerheitsproblem

Stellt die Frage ob eine gegebene Sprache leer ist (also  $L(G) = \emptyset$ )

⇒ **Entscheidbar** mit Markierungsalgorithmus

### Endlichkeitsproblem

Stellt die Frage ob für eine Sprache gilt  $|L| = \infty$

⇒ **Entscheidbar** mit dem Pumping - Lemma auf Wortproblem zurückführbar

Die meisten Probleme im Zusammenhang mit kontextfreien Sprachen sind unlösbar, d.h. es gibt nicht nur noch keine Lösung für diese Probleme, sondern es wurde bewiesen, daß diese Probleme keine Lösung haben. Es ist relativ einfach die entscheidbaren Probleme zu beweisen, wie wir im folgenden sehen werden. Jedoch benötigen wir für die Beweise der unentscheidbaren Probleme einiges Wissen aus der Komplexitätstheorie. Dieses wollen wir als kleinen Einschub an den entsprechenden Stellen kurz anreißen. Dem Leser sei ein genaueres Studium in [2] empfohlen. Wir beschränken uns auf die direkt notwendigen Sätze und Folgerungen in unseren Beweisen.

## § 6.3 Unentscheidbare Probleme

Für  $L(G)$  sind folgende Fragestellungen **unentscheidbar**

- **Mehrdeutigkeit**
- **regulär**
- **deterministisch kontextfrei**

Für  $L(G_1), L(G_2)$  sind folgende Fragestellungen **unentscheidbar**

- **Leerheitsproblem**  
⇒  $L(G_1) \cap L(G_2) = \emptyset$  ?
- **Endlichkeitsproblem**  
⇒  $|L(G_1) \cap L(G_2)| = \infty$  ?
- **Kontextfreiheit**  
⇒  $L(G_1) \cap L(G_2)$  kontextfrei ?
- **Teilmengenproblem**  
⇒  $L(G_1) \subset L(G_2)$  ?
- **Äquivalenzproblem**  
⇒  $L(G_1) = L(G_2)$  ?

## Beweise zu den entscheidbaren Problemen

### Wortproblem

Das Wortproblem für Typ 1 Sprachen (und damit für Typ 2, 3 Sprachen) ist entscheidbar. Es gibt einen Algorithmus der bei Eingabe einer kontextsensitiven Grammatik  $G = (V, \Sigma, P, S)$  und eines Wortes  $x \in \Sigma^*$  in endlicher Zeit entscheidet, ob  $x \in L(G)$  oder  $x \notin L(G)$

#### **Beweis**

```

INPUT (G, x); { |x| = n }
  T := {S}
REPEAT
  T1 := T
  T := Abln(T1)
UNTIL (x ∈ T) OR (T = T1)
IF x ∈ T
  THEN WriteString ('x liegt in L(G)')
  ELSE WriteString ('x liegt nicht in L(G)')
END

```

Dieser Algorithmus soll nur zur Vollständigkeit Erwähnung finden, denn für die kontextfreien Sprachen gibt es einen effektiveren und spezialisierteren Algorithmus zur Lösung des Wortproblems. Den CYK – Algorithmus benannt nach seinen Entdeckern Cocker, Younger und Kasami. Er hat nur noch eine Komplexität von  $O(n^3)$  im Gegensatz zum exponentiellen Aufwand des obigen Algorithmus (siehe Kapitel 7).

### **CYK – Algorithmus**

Entscheidet ob ein Wort  $w$  in einer Sprache  $L$  liegt oder nicht

#### **Algorithmusidee**

- Nichtdeterministische Suche nach einem Ableitungsbaum

#### **Verfahren**

- Verwende CNF
  - ⇒ Ableitungsbäume haben Binärstruktur
- Markiere alle möglichen Ableitungsschritte
- verwerfe Ansätze die in einer Sackgasse enden
- Rückwärtssimulation eines Ableitungsbaums
  - ⇒ Durchforstung des Baums von den Blättern zur Wurzel
  - ⇒  $w \in L(G) \Leftrightarrow$  kompletter Ableitungsbaum konnte generiert werden

### **Leerheitsproblem**

Sei  $G$  kontextfreie Grammatik in CNF

Markiere alle Variablen die Terminalwörter ableiten können

1. Markiere alle  $A$  falls  $(A \rightarrow a) \in P$
2. Markiere alle  $A$  falls  $(A \rightarrow BC) \in P$  und  $B, C$  schon markiert

⇒  $L(G) = \emptyset \Leftrightarrow S$  ist unmarkiert

### **Endlichkeitsproblem**

Mit Pumping – Lemma

$|L| = \infty \Leftrightarrow \exists z \in L(G)$  mit  $n \leq |z| < 2n$  (mit  $n$  Pumping – Zahl)

⇒ Reduktion auf Wortproblem

( $\Leftarrow$ )  $\exists w \in L$  mit  $|w| \geq n \Rightarrow L$  enthält nach dem Pumping – Lemma unendlich viele Wörter

( $\Rightarrow$ )  $|L| = \infty$  und  $|z| \geq n$  mit  $z \in L$  mit minimaler Länge

wenn  $|z| \geq 2n \Rightarrow z = uvwxy$  mit  $uwxy \in L$  und  $|uwxy| \geq n$

⇒ Widerspruch zur Minimalität von  $z$

⇒  $\exists z \in L$  mit  $n \leq |z| < 2n$

## Beweis zu den unentscheidbaren Problemen

Man kann dem Postschen Korrespondenzproblem zwei kontextfreie Grammatiken zuordnen. Auf diesen Grammatiken können dann die verschiedenen Probleme allgemein simuliert werden, und es kann dank des Postschen Korrespondenzproblems ihre Unentscheidbarkeit gefolgert werden. Wir geben nur die beiden Grammatiken an und erläutern beispielhaft die weiteren Schlußfolgerungen. Für diese Schlußfolgerungen ist die Anmerkung aus dem Kasten von großer theoretischer Bedeutung. Als Einschub folgt in § 6.4 eine kurze Einführung in die Komplexitätstheorie.

### Reduktion auf Postsches Korrespondenzproblem

$$K = ((x_1, y_1), \dots, (x_k, y_k)) \text{ mit } x_i, y_i \in \{0, 1\}^*$$

**Erinnerung**

$$K \leq H \leq \text{MPCP} \leq \text{PCP}$$

Ordne dem PCP folgende kontextfreie Grammatiken zu ( $\tilde{w}$  ist  $w$  gespiegelt)

$$G_1 = (V_1, \Sigma, P_1, S) \text{ mit } V_1 = \{S, A, B\}$$

$$\Sigma = \{0, 1, \$, a_1, \dots, a_k\}$$

$$P_1 = \{ S \rightarrow A\$B,$$

$$A \rightarrow a_1Ax \mid \dots \mid a_kAx_k$$

$$A \rightarrow a_1x_1 \mid \dots \mid a_kx_k$$

$$B \rightarrow \tilde{y}_1Ba_1 \mid \dots \mid \tilde{y}_kBa_k$$

$$B \rightarrow \tilde{y}_1a_1 \mid \dots \mid \tilde{y}_ka_k\}$$

$$\Rightarrow L_1 = \{ a_{i_n} \dots a_{i_1} x_{i_1} \dots x_{i_n} \$ \tilde{y}_{j_m} \dots \tilde{y}_{j_1} a_{j_1} \dots a_{j_m} \mid n, m \geq 1, i_\mu, j_k \in \{1, \dots, k\}\}$$

$$G_2 = (V_2, \Sigma, P_2, S) \text{ mit } V_2 = \{S, T\}$$

$$\Sigma = \{0, 1, \$, a_1, \dots, a_k\}$$

$$P_2 = \{ S \rightarrow a_1 Sa_1 \mid \dots \mid a_k Sa_k \mid T$$

$$T \rightarrow OT0 \mid 1T1 \mid \$\}$$

$$\Rightarrow L_2 = \{ uv \$ \tilde{v} \tilde{u} \mid u \in \{a_1, \dots, a_k\}^*, v \in \{0, 1\}^* \}$$

### Beispiel Leerheitsproblem

Es gilt nun:  $K$  besitzt die Lösung  $i_1, \dots, i_n \Leftrightarrow L_1 \cap L_2$  ist leer

$\Rightarrow$  Es ist nicht entscheidbar, ob  $K$  eine Lösung besitzt

$\Rightarrow$  Durch die Äquivalenz folgt nun, daß auch nicht entscheidbar ist ob  $L_1 \cap L_2$  leer ist!

Die nun folgenden Anmerkungen zur Komplexitätstheorie sollen nur dem Leser, der schon Erfahrungen auf diesem Gebiet hat, als Erinnerung dienen und ansonsten einen schemenhaften Überblick über die Materie geben, welche hier im Hintergrund Anwendung findet.

## § 6.4 Einschub: „Komplexitätstheorie“

### Beweis zum Speziellen Halteproblem

Das „Spezielle Halteproblem  $K = \{w \in \{0, 1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\}$ “ ist nicht entscheidbar

#### **Beweis**

Angenommen  $K$  ist entscheidbar. Dann ist  $\chi_K$  berechenbar mittels einer TM  $M$ . Baue diese Maschine um zu  $M'$ . Das heißt  $M'$  stoppt genau dann, wenn  $M$  0 ausgeben würde. Falls  $M$  1 ausgibt, gerät  $M'$  in eine Endlosschleife. Sei  $w'$  ein Codewort der Maschine  $M'$ . Nun gilt:

$$\begin{aligned} M' \text{ angesetzt auf } w' \text{ hält} &\Leftrightarrow M \text{ angesetzt auf } w' \text{ gibt } 0 \text{ aus} \\ &\Leftrightarrow \chi_K(w') = 0 \\ &\Leftrightarrow w' \notin K \\ &\Leftrightarrow M_w' \text{ angesetzt auf } w' \text{ hält nicht} \\ &\Leftrightarrow M' \text{ angesetzt auf } w' \text{ hält nicht} \quad \text{WIDERSPRUCH} \end{aligned}$$

### Hilfssatz

Falls  $A \leq B$  und  $B$  entscheidbar (bzw. semi-entscheidbar) ist, so ist auch  $A$  entscheidbar (bzw. semi-entscheidbar)

#### **Beweis**

Es gelte  $A \leq B$  mittels Funktion  $f$ . Ferner sei  $\chi_B$  berechenbar  $\Rightarrow \chi_{B \circ f}$  ist eine berechenbare Funktion

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x))$$

### Halteproblem

Das Halteproblem  $H$  ist nicht entscheidbar

#### **Beweis**

Es genügt,  $K \leq H$  nachzuweisen. Wähle  $f(w) = w\#w$ . Dann gilt:  $w \in K \Leftrightarrow f(w) \in H$

### PCP

$MPCP \leq PCP$

#### **Beweis**

$K = ((x_1, y_1), \dots, (x_k, y_k))$  mit  $f(K) = ((x_1^l, y_1^l), (x_2^l, y_2^l), \dots, (x_k^l, y_k^l), (\$, \#\$))$

$f$  ist berechenbar und  $f$  vermittelt eine Reduktion von  $MPCP \leq PCP$

$K$  besitzt eine Lösung mit  $i_1 = 1 \Leftrightarrow f(K)$  besitzt irgendeine Lösung

**Beweis**  $K$  hat die Lösung  $(1, i_2, \dots, i_n) \Rightarrow (1, i_2+1, \dots, i_n+1, k+2)$  ist Lösung von  $f(K)$

$f(K)$  hat Lösung  $i_1, \dots, i_n \in \{1, \dots, k+2\} \Rightarrow (1, i_2-1, \dots, i_n-1)$  ist eine Lösung von  $K$

### MPCP

$H \leq MPCP$

#### **Beweis**

Gegeben TM  $M = (Z, \Sigma, \Gamma, \delta, z_0, \hat{\cdot}, E)$  (deren Codierung), ein Eingabewort  $w \in \Sigma^*$

Überführe jedes Paar  $(M, w)$  in eine Folge  $(x_1, y_1), \dots, (x_k, y_k)$  so daß gilt:

$M$  angesetzt auf  $w$  stoppt  $\Leftrightarrow (x_1, y_1), \dots, (x_k, y_k)$  besitzt eine Lösung mit  $i_1 = 1$

Konstruiere MPCP mit Alphabet  $\Gamma \cup Z \cup \{\#\}$ , erstes Wortpaar ist  $(\#, \#z_0w\#)$

Gruppen von weiteren Wortpaaren

1. Kopierregeln:  $(a, a)$  für alle  $a \in \Gamma \cup \{\#\}$
2. Überführungsregeln:
 

$(za, z'c)$	, falls $\delta(z, a) = (z', c, N)$
$(za, cz')$	, falls $\delta(z, a) = (z', c, R)$
$(bza, z'bc)$	, falls $\delta(z, a) = (z', c, L)$ für alle $b \in \Gamma$
$(\#za, \#z'c)$	, falls $\delta(z, a) = (z', c, L)$
$(z\#, z'c\#)$	, falls $\delta(z, \hat{\cdot}) = (z', c, N)$
$(z\#, cz'\#)$	, falls $\delta(z, \hat{\cdot}) = (z', c, R)$
$(bz\#, z'bc\#)$	, falls $\delta(z, \hat{\cdot}) = (z', c, L)$ für alle $b \in \Gamma$
3. Löseregeln:  $(az_e, z_e)$  und  $(z_ea, z_e)$  für alle  $a \in \Gamma$  und  $z_e \in E$
4. Abschlußregeln:  $(z_e\#\#, \#)$

- Falls TM  $M$  bei Eingabe  $w$  stoppt so gibt es Konfiguration  $(k_0, \dots, k_t)$   
mit  $k_0 = z_0w$  und  $k_t = uz_e v$  Endkonfiguration

Lösung von MPCP ist dann  $\#k_0\#k_1\#\dots\#k_t\#k_t'\#\dots\#z_e\#\#$

$(k_t', k_t''$  entstehen durch Löschen von Nachbarsymbolen)

- Falls MPCP eine Lösung besitzt mit  $i_1 = 1$  so läßt sich eine stoppende Rechnung von  $M$  ablesen

**Kapitel 7. CYK – Algorithmus**

Sei  $G$  in CNF mit  $V = \{S_1, \dots, S_K\}$ ,  $\Sigma = \{a_1, \dots, a_H\}$  und  $S_1$  Startsymbol

Produktionensystem  $P = \begin{cases} \text{nichtterminierende } N = \{N_1, \dots, N_L\} \text{ der Form } N_i = (S_{N_{i,1}} \rightarrow S_{N_{i,2}} \cdot S_{N_{i,3}}) \\ \text{terminierende } Q = \{Q_1, \dots, Q_M\} \text{ der Form } Q_m = (S_{Q_{m,1}} \rightarrow a_{Q_{m,2}}) \end{cases}$

Frage: Ist  $w = w_1w_2\dots w_n \in \Sigma^*$  aus der gegebenen Sprache?

Der CYK – Algorithmus entsteht direkt aus der Anwendung des dynamischen Programmierens. Er ist in verschiedenen Versionen in vielen Buch zur theoretischen Informatik zu finden. Hier ist nun eine Version, dessen Aufbau es dem theoretischen Anwender leicht macht, das Verfahren zu implementieren (siehe hierzu das angegebene Beispiel mit Erläuterung).

**Algorithmus**

Eingabe: Wort  $w$

```

for i from 1 to n do
  for j from 1 to n do
    for k from 1 to K do
      X[i, j, k] ← 0
for i from 1 to n do
  for m from 1 to M do
    if w[i] = a[Q[m, 2]] then X[i, i, Q[m, 1]] ← 1
for j from 1 to n – 1 do
  for i from 1 to n – j do
    for k from i to j + i – 1 do
      if X[i, k, N[l, 2]] = 1 and X[k+1, i+j, N[l, 3]] = 1 then X[i, i + j, N[l, 1]] ← 1
return X[1, n, 1]
    
```

**Anmerkung**  
 Dem interessierten Leser sei noch die Invariante des Algorithmus angegeben

$$X_{ijk} = 1 \Rightarrow S_k \rightarrow^* w_i \dots w_j$$

**Beispiel für den CYK – Algorithmus**

Gegeben sei die Grammatik  $G = (\{S_1, S_2, S_3, S_4\}, \{0,1\}, P, \{S_1\})$  mit  $P = \{ \begin{matrix} S_1 \rightarrow S_1S_1 \mid S_3S_2 \mid S_3S_4 \\ S_2 \rightarrow S_1S_4 \\ S_3 \rightarrow 1 \\ S_4 \rightarrow 0 \end{matrix} \}$

und ein Wort  $w = 101010$ .

Mittels nebenstehendem Schema (auf der Hauptdiagonale steht das gesuchte Wort, übersetzt in die Nichtterminalsymbole) reduziert sich die Ausführung auf eine einfache „Pseudo“-Vektor - Multiplikation  $S_3 \cdot S_4 = S_1$ , da es eine Produktion gibt mit  $S_1 \rightarrow S_3S_4$ . Fährt man so fort, erhält man im rechten oberen Eck die Lösung des CYK – Algorithmus. Denn falls das Startsymbol hier erscheint, hat man eine Ableitungsbaum für das gesuchte Wort in der Sprache gefunden. Die Anzahl der gefunden Startsymbole gibt die Multiplizität des Wortes in der Sprache an.

1 = S <sub>3</sub>	S <sub>1</sub>		S <sub>1</sub>		S <sub>1</sub> , S <sub>1</sub>
	0 = S <sub>4</sub>				
		1 = S <sub>3</sub>	S <sub>1</sub>		S <sub>1</sub>
			0 = S <sub>4</sub>		
				1 = S <sub>3</sub>	S <sub>1</sub>
					0 = S <sub>4</sub>

⇒  $w \in L(G)$  und Multiplizität von  $w$  ist Zwei

## Literaturliste

- ☛ I. Wegener: „*Theoretische Informatik*“ Teubener, 1993
- ☛ U. Schöning: „*Theoretische Informatik – kurzgefaßt*“ Spektrum, 1999
- ☛ K. Estenfeld: „*Formale Sprachen*“
- ☛ Sippu, Soisalon-Soininen: „*Parsing Theory. Vol.1 + Vol.2*“
- ☛ G. Hotz: „*Skript Informatik III*“ , 1999