Automatentheorie und Formale Sprachen

– LV 4110 –

Kellerautomaten und Kontextfreie Sprachen

- Kennenlernen der Beschreibungsmöglichkeiten von Programmiersprachen
- Klärung, was man unter der Mehrdeutigkeit einer Sprache sowie der Mehrdeutigkeit einer Grammatik versteht
- Definition von Normalformen kontextfreier Grammatiken
- Kennenlernen des Pumping-Lemmas
- Modellierung und Arbeitsweise eines Kellerautomaten
- Kennenlernen des Zusammenhangs zwischen Kellerautomaten und kontextfreien Grammatiken
- Identifizierung der Probleme bei der Syntaxanalyse

IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen
   - 1.1 Beschreibungsmittel BNF und Syntaxdiagramme
   - 1.2 Definition für eine Chomsky-Grammatik
2. Mehrdeutigkeit bei kontextfreien Grammatiken
   - 2.1 Definition von Mehrdeutigkeiten
   - 2.2 Inhärent mehrdeutige Sprachen
3. Normalformen kontextfreier Grammatiken
   - 3.1 ε-freie Grammatik
   - 3.2 Greibach-Normalform
   - 3.3 Chomsky-Normalform

Fortsetzung:

Wir erinnern uns:

Chomsky-Grammatiken vom Typ 2 haben die Form:

$$A \rightarrow \gamma \quad \text{mit} \quad A \in N; \quad \gamma \in (N \cup T)^*$$

$$\text{oder} \quad \gamma = \varepsilon$$

Diese Regelform ist charakteristisch für die meisten höheren Programmiersprachen (PASCAL, C, ...).

- Backus-Naur-Form
- Syntaxdiagramme

IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen

1.1 Beschreibungsmittel BNF und Syntaxdiagramme

1.2 Definition für eine Chomsky-Grammatik

2. Mehrdeutigkeit bei kontextfreien Grammatiken

2.1 Definition von Mehrdeutigkeiten

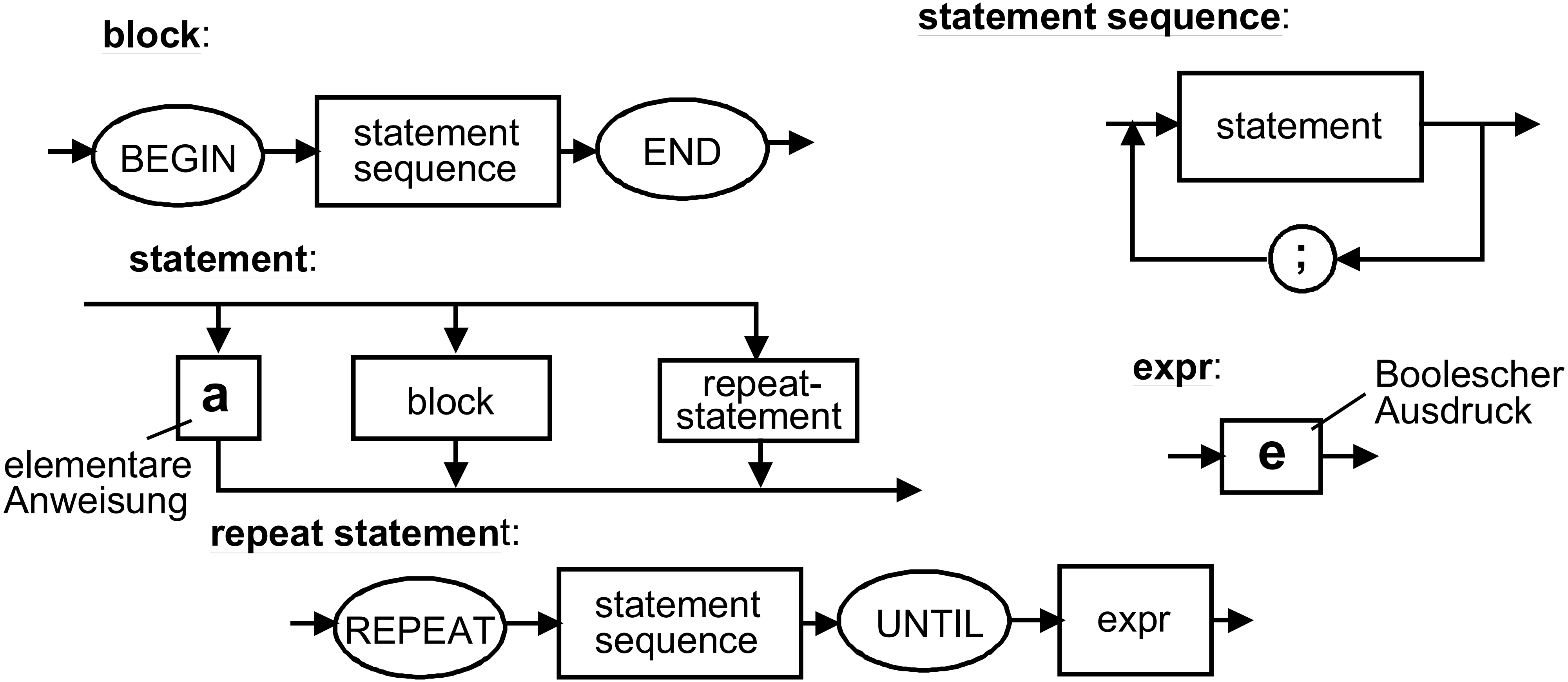
2.2 Inhärent mehrdeutige Sprachen

3. Normalformen kontextfreier Grammatiken

3.1 ε-freie Grammatik

3.2 Greibach-Normalform

3.3 Chomsky-Normalform



block:
BEGIN
statement sequence
END
statement:
a
block
repeat-statement
elementare Anweisung
repeat statement:
REPEAT
statement sequence
UNTIL
expr
statement sequence:
statement
;
expr:
e
Boolescher Ausdruck

Definition der syntaktischen Variablen:

<block>	::= **BEGIN** <statement seq.> **END**
<statement sequence>	::= <statement> { ; <statement> }
<statement>	::= a | <block> | <repeat-statement>
<repeat-statement>	::= **REPEAT** <statement sequence> **UNTIL** <expr>
<expr>	::= e

IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen
   - 1.1 Beschreibungsmittel BNF und Syntaxdiagramme
   - **1.2 Definition für eine Chomsky-Grammatik**
2. Mehrdeutigkeit bei kontextfreien Grammatiken
   - 2.1 Definition von Mehrdeutigkeiten
   - 2.2 Inhärent mehrdeutige Sprachen
3. Normalformen kontextfreier Grammatiken
   - 3.1 ε-freie Grammatik
   - 3.2 Greibach-Normalform
   - 3.3 Chomsky-Normalform

Definitionen für eine Chomsky-Grammatik:

- Terminalsymbole

  T = { **BEGIN**, **END**, **REPEAT**, **UNTIL**, **a**, **e**, **;** }
- Nonterminalsymbole

  N = { S, A, B, C, D, E } mit

  S = <block>
  A = <statement sequence>
  B = <statement>
  C = <repeat-statement>
  D = <expr>
  E = { ... } (Iteration) bzw. | (Alternative)

Ersetzt man in der BNF das Symbol **::=** durch den Ableitungspfeil $\Rightarrow$, so ergeben sich folgende Ableitungsregeln bzw. Produktionen **P**:

S $\Rightarrow$ **BEGIN** A **END**	(1)
A $\Rightarrow$ B E	(2)
E $\Rightarrow$ ε | **;** A	(3, 4)
B $\Rightarrow$ **a** | S | C	(5, 6, 7)
C $\Rightarrow$ **REPEAT** A **UNTIL** D	(8)
D $\Rightarrow$ **e**	(9)

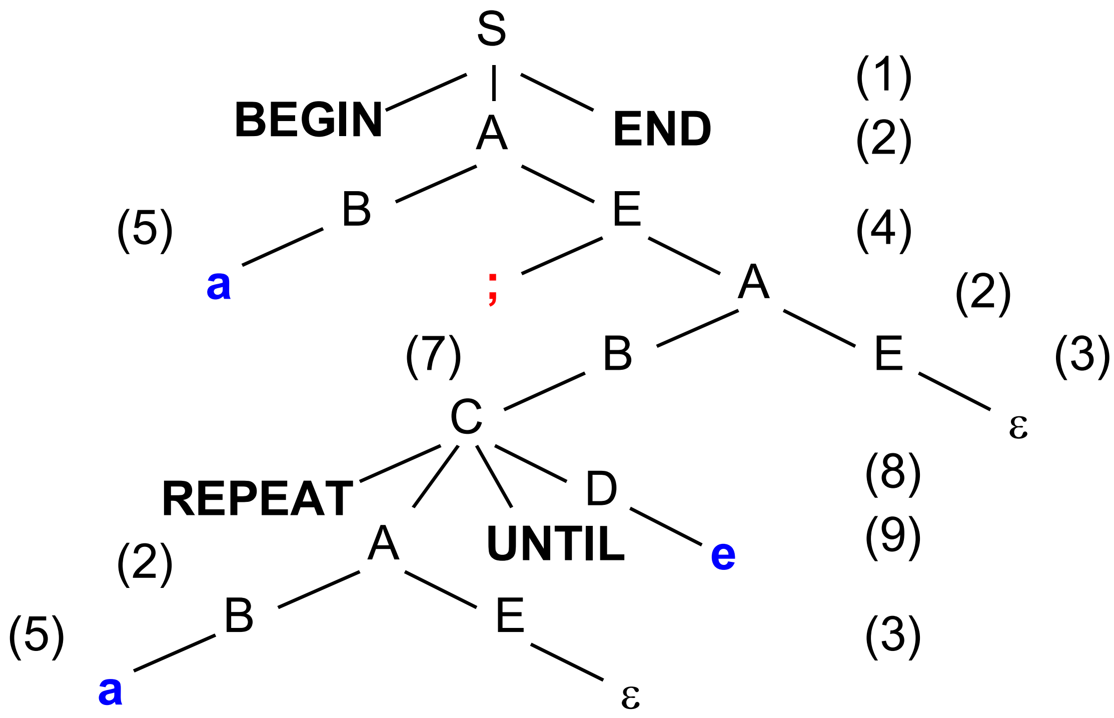
Ergebnis:

G = (**T**, **N**, **P**, S)

Beispiel:

BEGIN a ; Repeat a UNTIL e END

Für dieses Beispiel ergibt sich nebenstehender **Ableitungsbaum**



IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen
   - 1.1 Beschreibungsmittel BNF und Syntaxdiagramme
   - 1.2 Definition für eine Chomsky-Grammatik
2. **Mehrdeutigkeit bei kontextfreien Grammatiken**
   - **2.1 Definition von Mehrdeutigkeiten**
   - 2.2 Inhärent mehrdeutige Sprachen
3. Normalformen kontextfreier Grammatiken
   - 3.1 ε-freie Grammatik
   - 3.2 Greibach-Normalform
   - 3.3 Chomsky-Normalform

Definition:

Eine Grammatik heißt ***mehrdeutig***, wenn es ein Wort in **L(G)** gibt, zu dem Ableitungsbäume von **unterschiedlicher Struktur** existieren.

Bemerkung:

Der Ableitungsbaum ist ein **statisches** Gebilde, zu dem man auf verschiedenen Wegen, d. h. durch in der Reihenfolge der Schritte unterschiedliche Ableitungen, kommen kann. Wenn das Endresultat immer gleich ist, bedeutet dies noch keine Mehrdeutigkeit der Grammatik. Mit anderen Worten: Für die Mehrdeutigkeit sind verschiedene Ableitungsbäume maßgebend. Es reicht nicht aus, dass für ein Wort verschiedene Ableitungen existieren, denn diese können denselben Ableitungsbaum festlegen.

Eindeutige Grammatik:

Wir betrachten folgende **eindeutige** Grammatik G = (**T**, **N**, **P**, S) mit dem Startsymbol S:

$\mathbf{T} = \{\, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{+}, \mathbf{*}, \mathbf{(}, \mathbf{)} \,\}$

$\mathbf{N} = \{\, S, T, F, Z \,\}$

$\mathbf{P} = \{\, S \Rightarrow T \mid S \mathbf{+} T,$ (1, 2)

$T \Rightarrow F \mid F \mathbf{*} T,$ (3, 4)

$F \Rightarrow Z \mid \mathbf{(}\, S \,\mathbf{)},$ (5, 6)

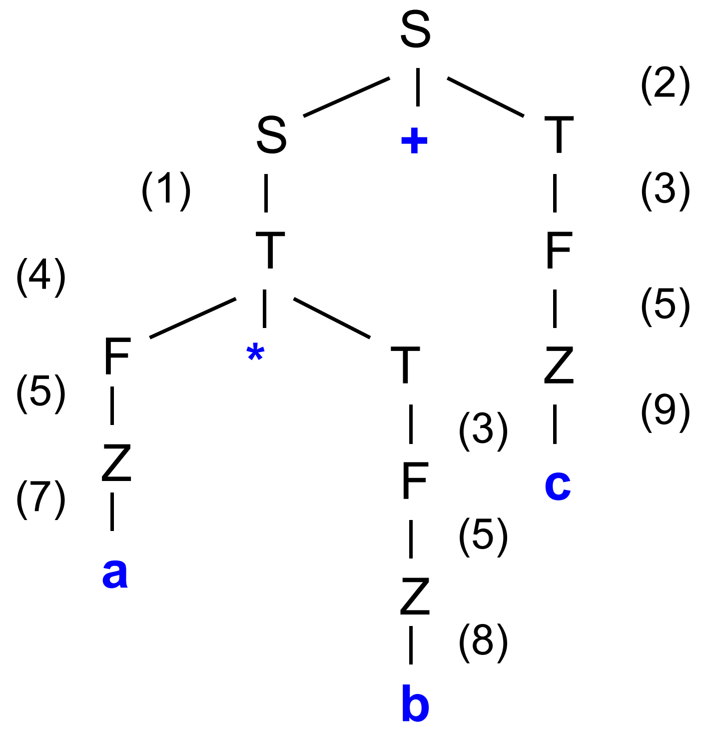
$Z \Rightarrow \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \,\}$ (7, 8, 9)

Beispiel:

a * b + c

Ableitung:

$$S \Rightarrow_{(2)} \underline{S} + T \Rightarrow_{(1)} \underline{T} + T \Rightarrow_{(4)} \underline{F} * T + T$$
$$\Rightarrow_{(5)} \underline{Z} * T + T \Rightarrow_{(7)} a * \underline{T} + T$$
$$\Rightarrow_{(3)} a * \underline{F} + T \Rightarrow_{(5)} a * \underline{Z} + T$$
$$\Rightarrow_{(8)} a * b + \underline{T} \Rightarrow_{(3)} a * b + \underline{F}$$
$$\Rightarrow_{(5)} a * b + \underline{Z} \Rightarrow_{(9)} a * b + c$$



IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen
   - 1.1 Beschreibungsmittel BNF und Syntaxdiagramme
   - 1.2 Definition für eine Chomsky-Grammatik
2. Mehrdeutigkeit bei kontextfreien Grammatiken
   - 2.1 Definition von Mehrdeutigkeiten
   - **2.2 Inhärent mehrdeutige Sprachen**
3. Normalformen kontextfreier Grammatiken
   - 3.1 ε-freie Grammatik
   - 3.2 Greibach-Normalform
   - 3.3 Chomsky-Normalform

Folgerung aus vorangegangenem Beispiel:

Zu jeder kontextfreien Grammatik gibt es unendlich viele äquivalente kontextfreie Grammatiken.

Schlußbemerkung:

Mehrdeutige Grammatiken sind unerwünscht, weil sie Interpretationsschwierigkeiten verursachen können. Mehrdeutigkeit kann aber u. U. nicht vermieden werden. Man spricht von einer ***inhärent mehrdeutigen*** *Sprache,* wenn **jede** Grammatik, die die Sprache erzeugt, mehrdeutig ist.
Aus der Mehrdeutigkeit einer Sprache läßt sich auch auf die Mehrdeutigkeit der zugehörigen Grammatik schließen, aber nicht umgekehrt.

IV. Kellerautomaten und Kontextfreie Sprachen

1. Grammatiken von Programmiersprachen
   - 1.1 Beschreibungsmittel BNF und Syntaxdiagramme
   - 1.2 Definition für eine Chomsky-Grammatik
2. Mehrdeutigkeit bei kontextfreien Grammatiken
   - 2.1 Definition von Mehrdeutigkeiten
   - 2.2 Inhärent mehrdeutige Sprachen
3. **Normalformen kontextfreier Grammatiken**
   - **3.1 ε-freie Grammatik**
   - 3.2 Greibach-Normalform
   - 3.3 Chomsky-Normalform

Definition:

Eine kontextfreie Grammatik G = (**N**, **T**, **P**, S) heißt *ε-frei*, wenn es in **P** keine Regel der Form **A → ε** mit A ∈ **N** gibt.

Satz:

Zu jeder kontextfreien Grammatik G mit der Sprache L(G) gibt es eine ε-freie mit der Sprache L(G´) = L(G) - {ε}.

Hinweis zur Konstruktion:

Regel A → ε kann wegfallen, wenn man z. B. die Regel B → aA durch B → a ergänzt.

Beispiel:

Gegeben sei G = (**N**, **T**, **P**, S) mit

P = { $S \Rightarrow AB$, $A \Rightarrow a$, $B \Rightarrow \varepsilon$ } (1, 2, 3) $\Rightarrow$ L(G) = { a }

gezeigt: $S \Rightarrow_{(1)} \underline{A}B \Rightarrow_{(2)} a\underline{B} \Rightarrow_{(3)} a\varepsilon = a$

Es sei nun G' gegeben mit

P = { $S \Rightarrow AB$, $A \Rightarrow a$, $S \Rightarrow A$ } (1, 2, 4) $\Rightarrow$ L(G') = { a }

gezeigt: $S \Rightarrow_{(4)} \underline{A} \Rightarrow_{(2)} a$

Ergebnis:

Durch Hinzufügen der Regel (4) haben wir nun eine **ε-freie** Grammatik G' zum Erzeugen der **gleichen Sprache**.

Anmerkung:

Gegeben sei G = (**N**, **T**, **P**, S) mit

$\mathbf{P} = \{ S \Rightarrow AB,\ A \Rightarrow a,\ B \Rightarrow \varepsilon \}$ (1, 2, 3) $\Rightarrow$ $L(G) = \{ a \}$

gezeigt: $S \Rightarrow_{(1)} \underline{A}B \Rightarrow_{(2)} a\underline{B} \Rightarrow_{(3)} a\varepsilon = a$

Hingegen würde das Entfernen der Regel (3) in G keinen Sinn ergeben, da für G'' mit

$\mathbf{P} = \{ S \Rightarrow AB,\ A \Rightarrow a \}$ (1, 2) $\Rightarrow$ $L(G'') = \{ \ \}$

gezeigt: $S \Rightarrow_{(1)} \underline{A}B \Rightarrow_{(2)} aB$ und B nicht weiter ersetzbar!

IV. Kellerautomaten und Kontextfreie Sprachen

Definition:

Eine kontextfreie Grammatik G = (**N**, **T**, **P**, S) ist in der **Greibach-Normalform** (kurz GNF), wenn **P** nur Regeln der Form $\mathbf{A \rightarrow a\, \varphi}$ mit $A \in \mathbf{N}$, $a \in \mathbf{T}$ und $\varphi \in \mathbf{N}^*$ enthält.

Satz:

Zu jeder kontextfreien Grammatik G mit der Sprache L(G) gibt es eine kontextfreie Grammatik G´ in Greibach-Normalform mit

$$L(G') = L(G) - \{\varepsilon\}.$$

Bemerkung:

Die Fragestellung, ob ein gegebenes Wort oder Programm zum Sprachumfang einer kontextfreien Sprache gehört, ist in der Regel einfacher zu untersuchen, wenn die Grammatik in Greibach-Normalform vorliegt, weil dann das Programm in der Reihenfolge **von links nach rechts** analysiert und abgearbeitet werden kann und der Ableitungsbaum systematischer in dieser Richtung aufgebaut werden kann.

IV. Kellerautomaten und Kontextfreie Sprachen

Definition:

Eine kontextfreie Grammatik G = (**N**, **T**, **P**, S) ist in der **Chomsky-Normalform** (kurz CNF), wenn **P** nur Regeln der Form **A → BC** oder **A → a** mit A, B, C ∈ **N** und a ∈ **T** enthält.

Satz:

Zu jeder kontextfreien Grammatik G mit der Sprache L(G) gibt es eine kontextfreie Grammatik G´ in Chomsky-Normalform mit

$$L(G') = L(G) - \{\varepsilon\}.$$

Beispiel:

Gegeben sei G = (**N**, **T**, **P**, S) mit **N** = { S }, **T** = { **(**, **)** } und

P = { $S \Rightarrow S\,S$, $S \Rightarrow (\,S\,)$, $S \Rightarrow \varepsilon$ } (1, 2, 3)

Mögliche Ableitung:

$S \Rightarrow_{(2)} \mathbf{(}\,\underline{S}\,\mathbf{)} \Rightarrow_{(1)} \mathbf{(}\,\underline{S}\,S\,\mathbf{)} \Rightarrow_{(2)} \mathbf{(\,(}\,S\,\mathbf{)}\,\underline{S}\,\mathbf{)} \Rightarrow_{(2)} \mathbf{(\,(}\,S\,\mathbf{)\,(}\,S\,\mathbf{)\,)}$

$\Rightarrow_{(3)} \mathbf{(\,(}\,\varepsilon\,\mathbf{)\,(}\,\underline{S}\,\mathbf{)\,)} \Rightarrow_{(2)} \mathbf{(\,(}\,\varepsilon\,\mathbf{)\,(\,(}\,\underline{S}\,\mathbf{)\,)\,)} \Rightarrow_{(3)} \mathbf{(\,(}\,\varepsilon\,\mathbf{)\,(\,(}\,\varepsilon\,\mathbf{)\,)\,)}$

d. h.

(( )(( )))

Nun Herleitung einer Grammatik in **CN** $\Rightarrow$ 2 Schritte

1. Schritt: Umformung von G in ε-freies G'.

$\mathbf{P} = \{\, S \Rightarrow S\,S,\ S \Rightarrow K_a\, S\, K_z,\ S \Rightarrow K_a\, K_z,\ K_a \Rightarrow \mathbf{(},\ K_z \Rightarrow \mathbf{)}\, \}$

neue Regel (3) (1, 2, 3, 4, 5)

2. Schritt: Nonterminalsymbol **H** einführen mit der Regel $\mathbf{H} \Rightarrow S\,K_z$.

$\mathbf{P} = \{\, S \Rightarrow S\,S,\ S \Rightarrow K_a\, \mathbf{H},\ \mathbf{H} \Rightarrow S\,K_z,\ S \Rightarrow K_a\, K_z,\ K_a \Rightarrow \mathbf{(},\ K_z \Rightarrow \mathbf{)}\, \}$

(1, 2, 3, 4, 5, 6)

Dies entspricht nun der **Chomsky-Normalform**, da nur Regeln der Form: $A \Rightarrow B\,C$ oder $A \Rightarrow a$.

Chomsky-Normalform

Definition

<u>Mögliche Ableitung gemäß **CN**:</u>

$$\mathbf{P} = \{\, S \Rightarrow S\,S,\; S \Rightarrow K_a\,\mathbf{H},\; \mathbf{H} \Rightarrow S\,K_z,\; S \Rightarrow K_a\,K_z,\; K_a \Rightarrow \mathbf{(},\; K_z \Rightarrow \mathbf{)} \,\}$$

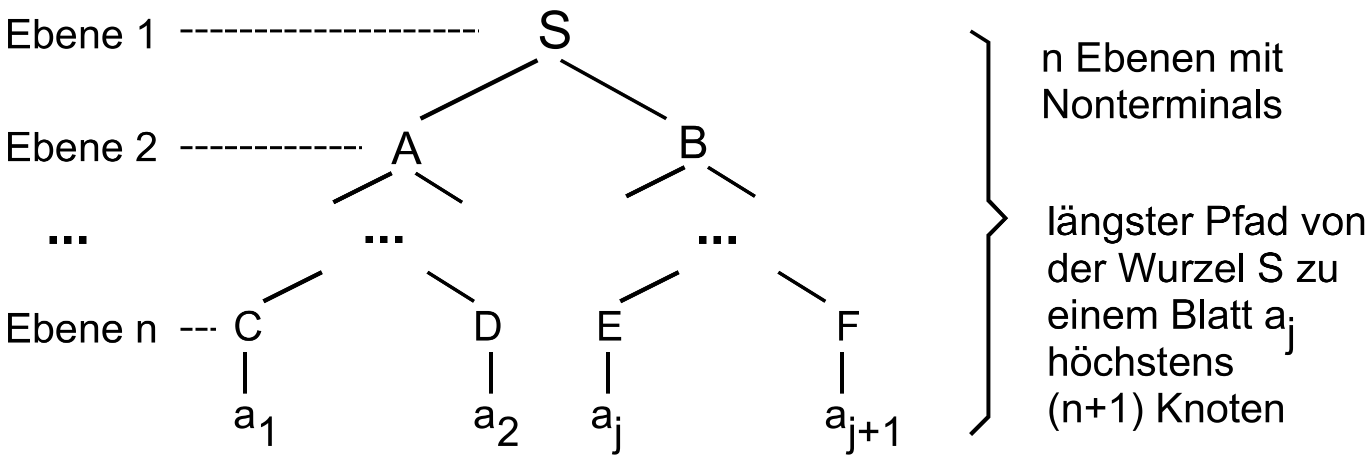
(1, 2, 3, 4, 5, 6)

$$S \Rightarrow_{(2)} \underline{K_a}\,\mathbf{H} \Rightarrow_{(5)} \mathbf{(}\,\underline{\mathbf{H}} \Rightarrow_{(3)} \mathbf{(}\,S\,\underline{K_z} \Rightarrow_{(6)} \mathbf{(}\,\underline{S}\,\mathbf{)} \Rightarrow_{(1)} \mathbf{(}\,\underline{S}\,S\,\mathbf{)} \Rightarrow_{(4)} \mathbf{(}\,\underline{K_a}\,K_z\,\underline{S}\,\mathbf{)}$$

$$\Rightarrow_{(2)} \mathbf{(}\,\underline{K_a}\,K_z\,K_a\,\mathbf{H}\,\mathbf{)} \Rightarrow_{(5)} \mathbf{((}\,\underline{K_z}\,K_a\,\mathbf{H}\,\mathbf{)} \Rightarrow_{(6)} \mathbf{(()}\,\underline{K_a}\,\mathbf{H}\,\mathbf{)} \Rightarrow_{(5)} \mathbf{(()(}\,\underline{\mathbf{H}}\,\mathbf{)}$$

$$\Rightarrow_{(3)} \mathbf{(()(}\,\underline{S}\,K_z\,\mathbf{)} \Rightarrow_{(4)} \mathbf{(()(}\,K_a\,K_z\,\underline{K_z}\,\mathbf{)} \Rightarrow_{(6)} \mathbf{(()(}\,\underline{K_a}\,K_z\,\mathbf{))}$$

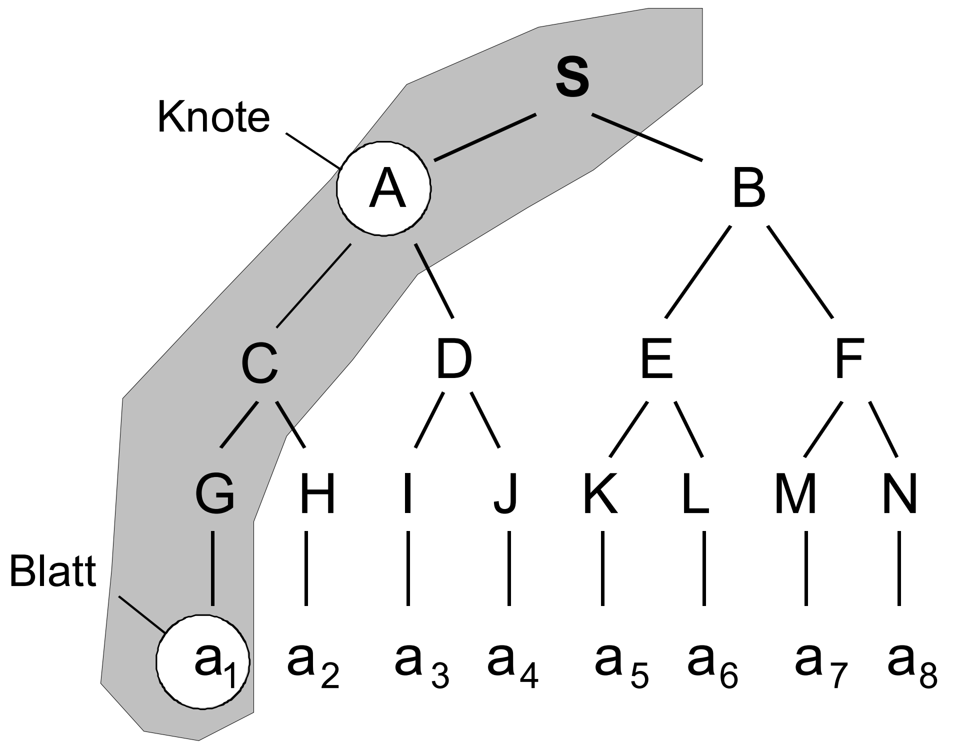
$$\Rightarrow_{(6)} \mathbf{(()((}\,\underline{K_z}\,\mathbf{))} \Rightarrow_{(6)} \boxed{\mathbf{(()(()))}}$$



- Es handelt sich um einen **Binärbaum**, bei dem jeder Knoten genau zwei Sohnknoten hat (mit Ausnahme der letzten Ebene).
- Da sich beim Übergang auf die nächste Ebene (mit Ausnahme der letzten Ebene) die Zeichenmenge **höchstens verdoppeln** kann, gilt für die **Länge** des abgeleiteten Wortes $|w| \leq 2^{n-1}$.

Beispiel:

$n = 4$



Ebene 1

Ebene 2

Ebene 3

Ebene **n = 4**

$| w | = 8 = 2^3$

hier:

längster Pfad von S bis a_1 umfasst $5 = (n + 1)$ Knoten

hier: $\mathbf{n} - 1 = 3$

Satz:

Es sei **L(G)** eine reguläre Sprache. Dann gibt es eine von **G** abhängige Zahl **k**, so dass jedes Wort $\mathbf{w} \in \mathbf{L(G)}$ mit der Länge $|\mathbf{w}| \geq \mathbf{k}$ in der Form

$$\mathbf{w = xyz} \qquad \text{mit} \qquad \mathbf{x, y, z} \in \mathbf{T^*}$$

geschrieben werden kann, wobei gilt:

a) $|xy| \leq k$

b) $y \neq \varepsilon$

c) $xy^iz \in L(G)$ für $i \geq 0$.

Satz:

Es sei **L(G)** eine kontextfreie Sprache. Dann gibt es eine von **G** abhängige Zahl **k**, so dass jedes Wort $\mathbf{w} \in \mathbf{L(G)}$ mit der Länge $|\mathbf{w}| \geq \mathbf{k}$ in der Form

$$\mathbf{w = xuyvz} \qquad \text{mit} \qquad \mathbf{x, u, y, v, z} \in \mathbf{T^*}$$

geschrieben werden kann, wobei gilt:

a) $|uyv| \leq k$

b) $uv \neq \varepsilon$

c) $xu^iyv^iz \in L(G)$ für $i \geq 0$.

<u>Beweisvorbereitung</u>:

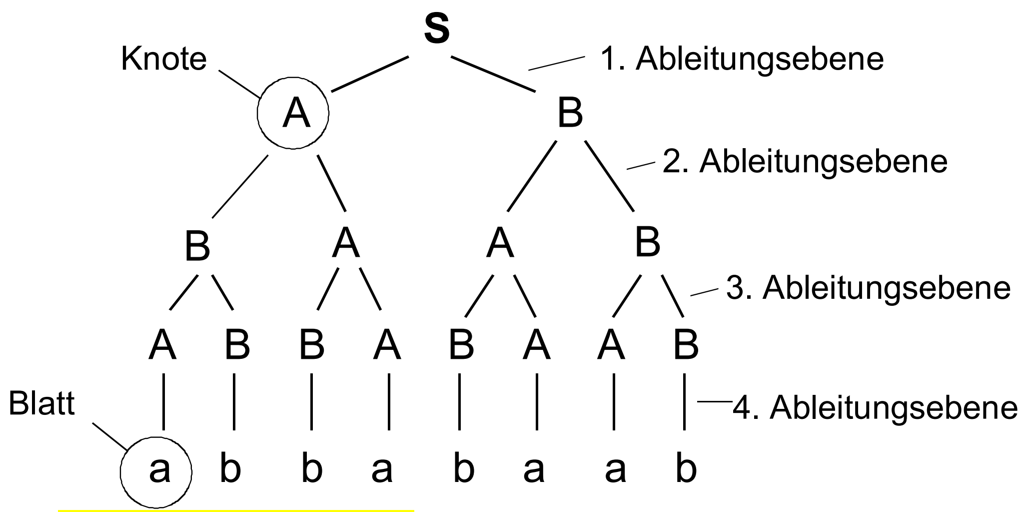
Ausgangspunkt: Sei **G** in der Chomsky-Normalform (CNF), d. h. o. E. nur Regeln der Form:

$\mathbf{A} \rightarrow \mathbf{a}$, wobei $\mathbf{a} \in \mathbf{T}$

oder $\mathbf{A} \rightarrow \mathbf{BC}$, wobei $\mathbf{B}, \mathbf{C} \in \mathbf{N}$.

Wir wählen: $\mathbf{k = 2^n}$, $\mathbf{n = \#N}$ (Anzahl der Nichtterminale einschließlich des Startsymbols **S**)

Ableitung von **w**: Der Ableitungsbaum für ein Wort $\mathbf{w} \in \mathbf{L(G)}$ mit voraussetzungsgemäß $\mathbf{|w| \geq k = 2^n > 2^{n-1}}$ Blätter ist ein <u>Binärbaum</u> (mit dem Startsymbol **S**)



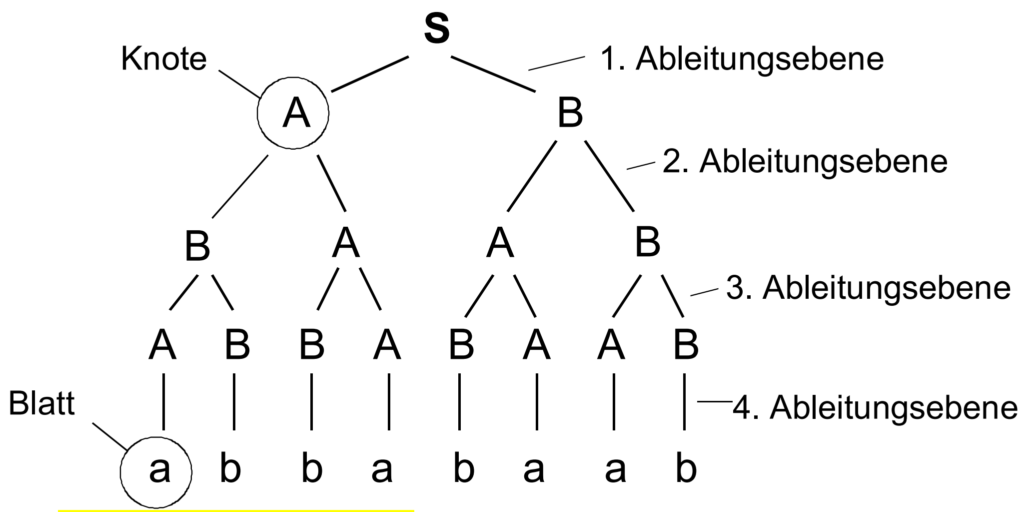
$|w| \geq k = 2^3 = 8 \;\Rightarrow\;$ **4 Ableitungsebenen**

Knoten mit **genau einem** Blatt repräsentieren eine Regel der Form **$A \to a$**.

Alle anderen Knoten mögen für eine Regel der Form **$A \to BC$** stehen und haben **genau zwei** Blätter.

hier: **$n = 3$** , weil **$N = \{S, A, B\}$**

Um in einem solchen <u>Binärbaum</u> **2^n** Blätter zu erhalten, benötigt man mindestens **$n + 1$ Ableitungsebenen**.
Um von dem Startsymbol **S** aus zu einem Blatt zu gelangen, ergibt sich eine Knotenfolge von mindestens **$n + 1$ Nonterminals** (einschließlich dem Startsymbol **S**).



$A \rightarrow a$.

$A \rightarrow BC$

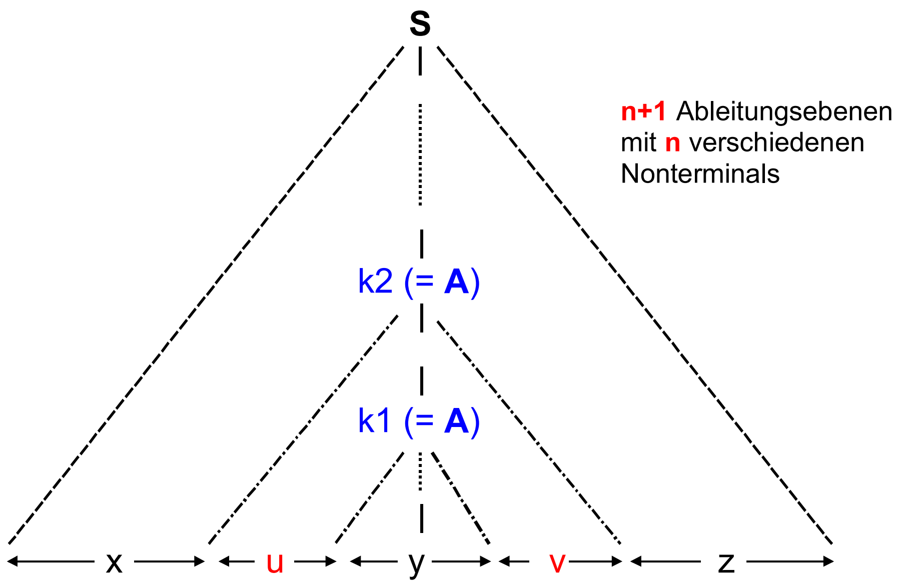
$|w| \geq k = 2^n$

$|w| \geq k = 2^3 = 8 \Rightarrow$ **4 Ableitungsebenen** hier: **n = 3** , und **N = {S, A, B}**

Folgerung:

Da definitionsgemäß jedoch nur **n** verschiedene Nonterminals existieren (vgl. S. 37 **n = #N**), tritt mindestens **ein** Nonterminal mehrfach auf.

Wir bezeichnen dieses Nonterminal im folgenden mit **A**.



Bezeichnung:

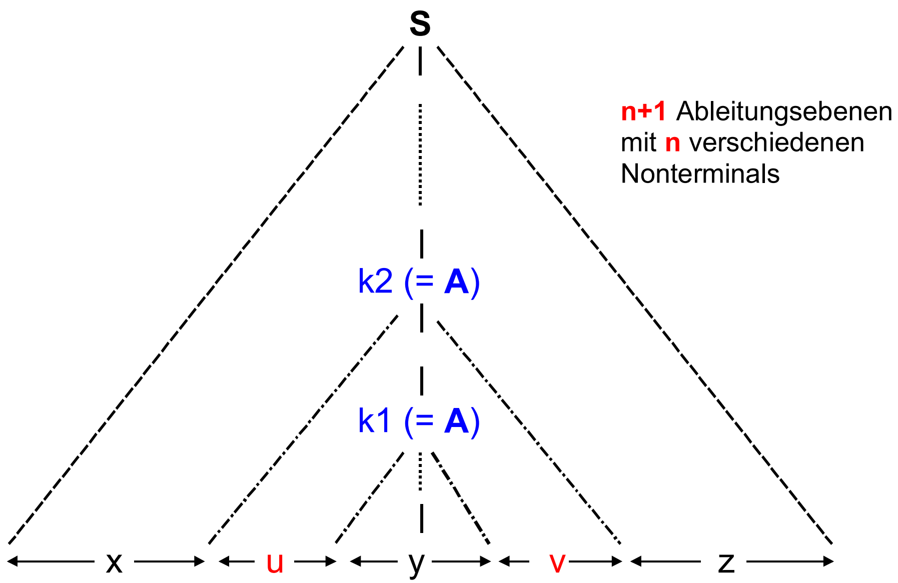
Wir bezeichnen die letzte Wiederholung von **A** mit k1, die vorletzte mit k2 und erhalten nebenstehenden den Ableitungsbaum:

Es ist also:

y aus k1
uyv aus k2 und
w = xuyvz aus **S** abgeleitet

1. Folgerung:

Da uyv aus k2 erzeugt wird und für diese Ableitung höchstens **n+1** Ableitungsschritte benötigt werden, gilt: $|uyv| \leq \mathbf{2^n}$ bzw. $|uyv| \leq \mathbf{k}$.



Es sind:

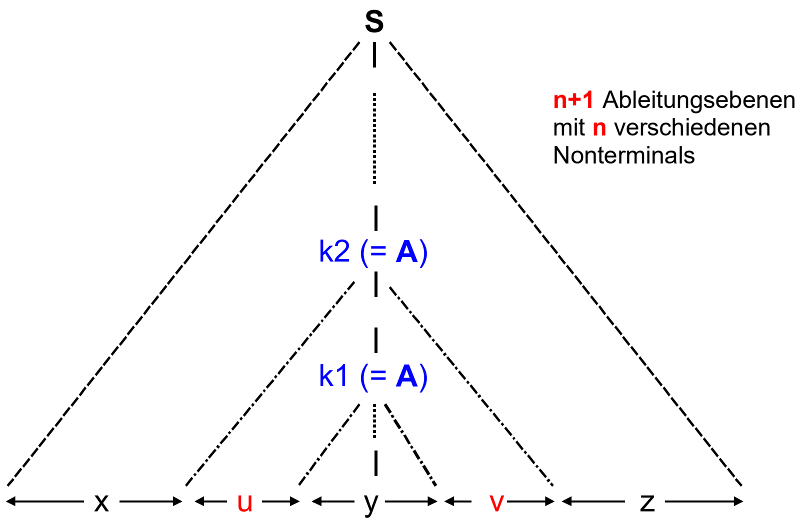
y aus k1

uyv aus k2 und

w = xuyvz aus **S** abgeleitet

2. Folgerung:

k2 hat <u>**genau**</u> <u>**zwei**</u> Nachfolger. Aus dem einen Nachfolger geht später y hervor, aus dem anderen u oder v. Also ist: $uv \neq \varepsilon$.



Es sind:

y aus k1

uyv aus k2 und

w = xuyvz aus **S** abgeleitet

3. Folgerung:

Da man die Ableitungsschritte des Teilbaums ab k1 bereits früher, d. h. bereits ab k2 hätte vornehmen können und diejenigen Ableitungen ab k2 auch ab k1 hätten wiederholt werden können (und zwar beliebig oft) entstehen Wörter der Form: xyz, xuyvz, xuuyvvz, xuuuyvvvz etc. Also ist: $xu^iyv^iz \in L(G)$ für $\mathbf{i} \geq 0$.

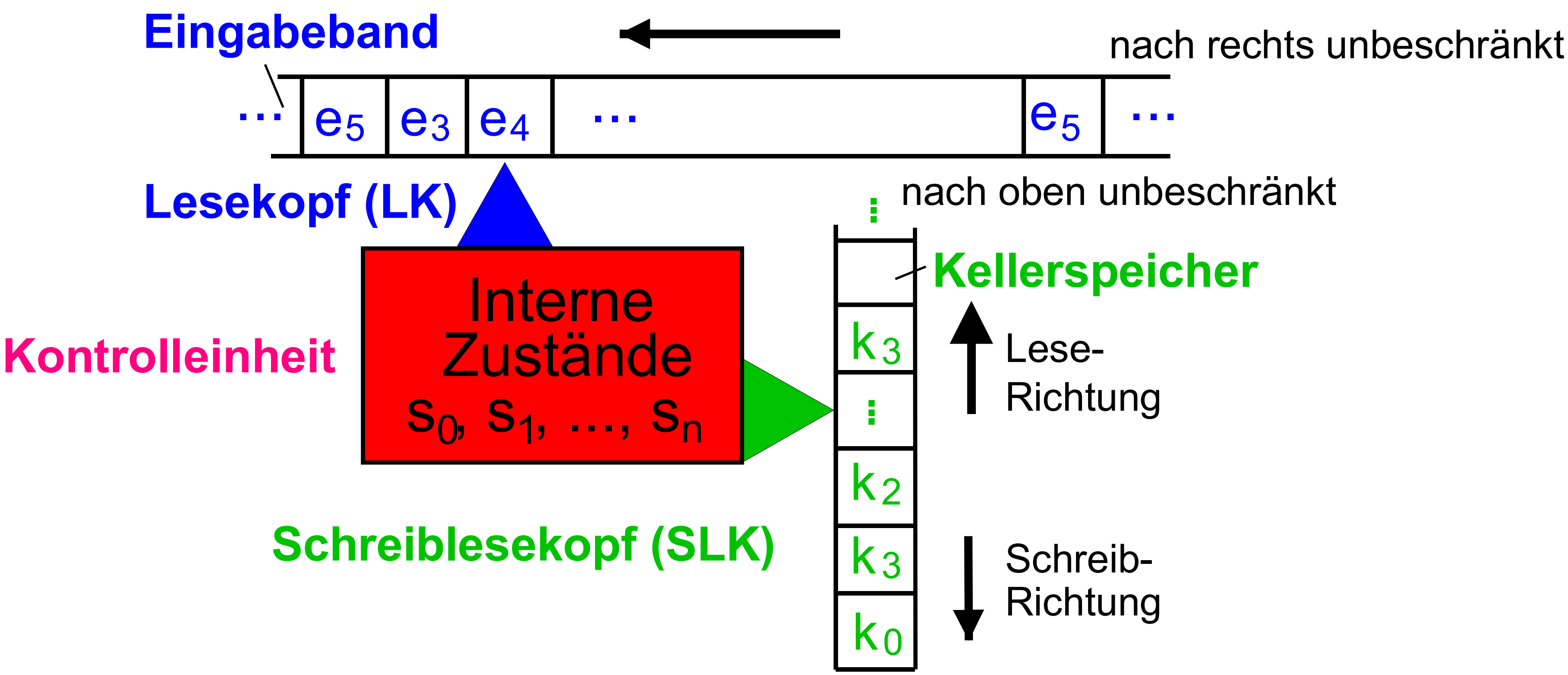
4. Das Pumping Lemma
    - 4.1 Pumping Lemma für reguläre Sprachen
    - 4.2 Pumping Lemma für kontextfreie Sprachen
5. **Kellerautomaten**
    - 5.1 Modellbildung
    - 5.2 Deterministische Kellerautomaten
    - 5.3 Sprache des deterministischen Kellerautomaten
    - 5.4 Nicht-deterministische Kellerautomaten
    - 5.5 Sprache des Nicht-deterministischen Kellerautomaten
    - 5.6 Kellerautomaten und kontextfreie Grammatiken
    - 5.7 Das Problem der Syntaxanalyse

bisher:

- Endliche Automaten betrachtet, die dadurch charakterisiert sind, dass die Anzahl ihrer Zustände endlich ist.
- Folglich war ein solcher Automat auch nicht in der Lage, **un**beschränkt viele Informationen zu speichern.

Ausweg:

- Naheliegende Erweiterung besteht darin, einen endlichen Automaten mit einem unbeschränkt großen Speicher zu versehen.
- Dieser Speicher würde es erlauben, die Vergangenheit der Verarbeitung eines Wortes in gewissem Umfang festzuhalten.


Eingabeband
nach rechts unbeschränkt
… e5 e3 e4 … e5 …
Lesekopf (LK)
nach oben unbeschränkt
Kellerspeicher
Interne Zustände
s0, s1, …, sn
Kontrolleinheit
k3
Lese-Richtung
k2
k3
k0
Schreib-Richtung
Schreiblesekopf (SLK)

Ein Verarbeitungsschritt besteht darin, dass der Automat das Zeichen unter dem Lesekopf (LK) und unter dem Schreib-/Lesekopf (SLK) liest und **– in Abhängigkeit vom aktuellen Zustand –** seinen Zustand verändert **sowie das oberste Kellerzeichen durch eine Folge von Zeichen ersetzt**. Anschließend wird der LK um eine Position nach rechts und der SLK auf das oberste Kellerzeichen positioniert.

Definition:

Das Septupel **KA** = (**S**, s_0, **F**, Σ, **K**, k_0, δ) bezeichnet einen deterministischen endlichen Kellerautomaten, wenn für die die einzelnen Komponenten gilt:

- **S** endliche Menge der *internen Zustände* des Automaten
- s_0 interner Anfangszustand, $s_0 \in \mathbf{S}$
- **F** Menge der internen Endzustände, $\mathbf{F} \subseteq \mathbf{S}$
- Σ *endliche* Menge der Eingabezeichen
- **K** endliche Menge der Kellerzeichen k_i
- k_0 Kellerstartzeichen (unterstes Zeichen auf dem Kellerband)
- δ (*deterministische*) Überführungsfunktion mit $\delta: \mathbf{S} \times (\Sigma \cup \{\varepsilon\}) \times \mathbf{K} \rightarrow \mathbf{S} \times \mathbf{K}^*$

Eigenschaften:

Ist für $s \in \mathbf{S}$, $a \in \Sigma$ und $k \in \mathbf{K}$

$\delta(s, a, k)$ definiert,

so ist

$\delta(s, \varepsilon, k)$ undefiniert.

Damit wird gewährleistet, dass es höchstens eine Möglichkeit gibt, δ anzuwenden.

Ähnlich wie ein endlicher Automat **EA** ist ein **KA** ein Akzeptator, d. h. ein Eingabewort $\mathbf{w} \in \Sigma^*$ wird akzeptiert, wenn sich der Automat nach der Verarbeitung des Wortes in einem **Endzustand** befindet, andernfalls wird es nicht akzeptiert.

Eigenschaften:

Die Überführungsfunktion

$$\delta: \mathbf{S} \times (\Sigma \cup \{\varepsilon\}) \times \mathbf{K} \rightarrow \mathbf{S} \times \mathbf{K}^*$$

bedeutet, dass ein geordnetes Tripel **(s, a, k)** in ein Paar **(s‘, v)** überführt wird, wobei

s‘ = der neue Zustand und

v = ein Wort aus Kellerzeichen ist, durch das das oberste Kellerzeichen ersetzt wird.

Deterministische Kellerautomaten

Ausgangskonfiguration: $(s_0, a_1a_2...a_n, k_0)$

Übergangsverhalten: $\rightarrow$ schrittweise Bearbeitung des Eingabewortes $a_1a_2...a_n$

Solange entweder $\delta(s, a, k)$ für das aktuelle Eingabezeichen $a \in \Sigma$ oder $\delta(s, \varepsilon, k)$ definiert ist, führe aus:

Setze $(s', w) := \delta(s, x, k)$ (mit $x = a$ oder $x = \varepsilon$)
Entferne k aus dem Keller (Pop-Funktion)
Schreibe w in den Keller (Push-Funktion)
Gehe in den internen Zustand s' über
Falls $x \neq \varepsilon$
    gehe ein Zeichen weiter auf dem Eingabeband.

Definition:

Es sei **KA** = (**S**, s_0, **F**, Σ, **K**, k_0, δ) ein Kellerautomat.

Die **Sprache des KA** besteht aus **allen Worten** von Σ^*, bei denen sich der Kellerautomat nach der Verarbeitung des letzten Zeichens auf dem Eingabeband in einem internen Endzustand befindet und das Kellerband wieder in der Ausgangsposition steht. Diese Situation bezeichnet man auch als eine

→ **(akzeptierende) Endkonfiguration**: (s_f, , k_0) mit $s_f \in$ **F**

→ d. h. auch, dass das Eingabeband leer ist!

Beispiel: $\Rightarrow$ Deterministischer **KA** für $\mathbf{a^n b^n}$ mit $n > 0$

Wir wollen zeigen, dass der **KA** mit

$$\Sigma = \{a, b\} \qquad \mathbf{S} = \{S_0, S_1\} \qquad \mathbf{F} = \{S_1\} \qquad \mathbf{K} = \{k_0, a\}$$

sowie der Zustandsüberführungsfunktion δ

gemäß:

$\delta(S_0, a, k_0) = (S_0, ak_0)$ (1) $\qquad \delta(S_0, a, a) = (S_0, aa)$ (2)

$\delta(S_0, b, a) = (S_1, \varepsilon)$ (3) $\qquad \delta(S_1, b, a) = (S_1, \varepsilon)$ (4)

genau die Wörter der Sprache $L(\mathbf{KA}) = \{\mathbf{a^n b^n} \mid n \in \mathbb{N}\}$ akzeptiert.

Konfigurationsfolge beim Akzeptieren von a^3b^3:

1) Schritt 0 = **Startkonfiguration**

2) Schritt 6 = **akzept. Endkonfiguration**

$\delta(S_0, a, k_0) = (S_0, ak_0)$ (1)

$\delta(S_0, a, a) = (S_0, aa)$ (2)

$\delta(S_0, b, a) = (S_1, \varepsilon)$ (3)

$\delta(S_1, b, a) = (S_1, \varepsilon)$ (4)

Schritt	Gl(…)	Eingabeband	Kellerband	Zustand	Konfiguration
0		a a a b b b	k_0	S_0	**(S_0, a a a b b b, k_0)** 1)
1	1	a a a b b b	a k_0	S_0	(S_0, a a b b b, a k_0)
2	2	a a a b b b	a a k_0	S_0	(S_0, a b b b, a a k_0)
3	2	a a a b b b	a a a k_0	S_0	(S_0, b b b, a a a k_0)
4	3	a a a b b b	a a k_0	S_1	(S_1, b b, a a k_0)
5	4	a a a b b b	a k_0	S_1	(S_1, b, a k_0)
6	4	a a a b b b _	k_0	S_1	**(S_1, , k_0)** 2)

Definition:

Ein nicht-deterministischer Kellerautomat ist definiert durch ein Septupel **KA** = (**S**, s_0, **F**, Σ, **K**, k_0, δ), wenn alle Komponenten – außer δ – dieselbe Bedeutung haben wie beim deterministischen KA und δ eine *nicht-deterministische* Überführungsfunktion von **S** x (Σ ∪ {ε}) x **K** in die Potenzmenge von **S** x **K*** darstellt. Das bedeutet, dass es Konfigurationen des Kellerautomaten gibt, für die mehrere Nachfolgekonfigurationen existieren.

Die beim deterministischen KA getroffene Einschränkung bzgl. der Definiertheit von δ kann entfallen; man kann δ als eine **totale** Funktion annehmen.

Definition:

Die Sprache des nicht-deterministischen Kellerautomaten KA besteht aus allen Worten aus Σ^*, bei denen es *möglich ist*, daß sich nach dem Lesen des letzten Eingabezeichens der KA in einer das Wort akzeptierenden Endkonfiguration befindet.

Satz:

Im Gegensatz zum endlichen Automaten **EA** besteht zwischen nicht-deterministischen Kellerautomaten und deterministischen Kellerautomaten **keine** Äquivalenz und demzufolge auch keine Überführungsmöglichkeiten.

Satz:

Zu jeder kontextfreien Grammatik G gibt es einen nicht-deterministischen Kellerautomaten KA und umgekehrt mit

$$L(KA) = L(G).$$

Beweis:

Nur für eine Richtung durch Konstruktion des KA zu vorgegebener Grammatik $G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, S_G)$.

Idee:

Die Anwendung von Regeln wird mit Hilfe des Kellerspeichers simuliert. Dabei wird gleichzeitig das Eingabewort w verarbeitet.

Input: (gegeben!)

$\rightarrow$ kontextfreie Grammatik $G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, S_G)$.

Output: (gesucht!)

$\rightarrow$ nicht-deterministischer $\mathbf{KA} = (\mathbf{S}, s_0, \mathbf{F}, \Sigma, \mathbf{K}, k_0, \delta)$ mit $L(KA) = L(G)$.

Algorithmusbeschreibung:

- Zustände des KA: $\mathbf{S} = \{s_0, s_f\}$
- Eingabezeichen des KA: $\Sigma = \mathbf{T}$ (d. h. Terminals von G)
- Kellerzeichen des KA: $\mathbf{K} = \{k_0\} \cup \mathbf{T} \cup \mathbf{N}$
- Endzustand des KA: $\mathbf{F} = \{S_f\}$

- Überführungsfunktion des KA:

$\delta(s_0, \varepsilon, k_0) = (s_f, S_G k_0)$	;	am Anfang wird das Startsymbol S_G in den Keller geschrieben
$\delta(s_f, \varepsilon, A) = (s_f, \gamma)$ mit $A \in \mathbf{N}$	;	für jede Regel $A \to \gamma$ von G bzw. $\in$ P, wobei oberstes Kellerzeichen = 1. Zeichen von γ
$\delta(s_f, a, a) = (s_f, \varepsilon)$ mit $a \in \mathbf{T}$	;	für alle Eingabezeichen bzw. Terminals **T**

Umkehrung:

Zum Nachweis, daß L(G) = L(KA) gilt, geht man davon aus, dass ein Wort der Sprache L(G) auf dem Eingabeband des Kellerautomaten steht und der KA in der Ausgangssituation ist.

Man beachte, dass als "Eingabezeichen" auch ε erlaubt ist, falls auf dem Kellerband ein Nonterminal an oberster Stelle steht. Schritt für Schritt wird vom KA entweder eine Erzeugungsregel für das Eingabewort auf das Kellerband geschrieben oder ein Eingabezeichen abgearbeitet, bis das letzte Zeichen auf dem Eingabeband erreicht ist.

Beispiel: $\Rightarrow$ Konstruktion eines nicht-deterministischen Kellerautomaten **KA**

Gegeben sei die Grammatik $G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, S_G)$ mit $\mathbf{N} = \{S_G\}$, $\mathbf{T} = \{a, b\}$, dem Startsymbol S_G und den beiden Regeln

$$\mathbf{P} = \{ S_G \Rightarrow a\, S_G\, b,\ S_G \Rightarrow \varepsilon \},$$

welche die Sprache $L(G) = \{\mathbf{a^n b^n} \mid n \in \mathbf{IN}\}$ erzeugt.

Gesucht sei der entsprechende **KA** mit dem Startzustand S_0 in der algebraischen Form:

$$\mathbf{KA} = (\mathbf{S}, S_0, \mathbf{F}, \Sigma, \mathbf{K}, k_0, \delta).$$

Lösung:

Für den gesuchten **KA** ergibt sich:

$\Sigma = \{ a, b \}$ $\quad$ **S** $= \{ S_0 , S_f \}$ $\quad$ **F** $= \{ S_f \}$ $\quad$ **K** $= \{ S_G, k_0, a, b \}$

und die Zustandsüberführungsfunktion δ gemäß:

$\delta(S_0, \varepsilon, k_0) = (S_f, S_G\, k_0)$ (1)

$\delta(S_f, \varepsilon, S_G) = (S_f, a\, S_G\, b) \mid (S_f, \varepsilon)$ (2), (2')

$\delta(S_f, a, a) = (S_f, \varepsilon)$ (3)

$\delta(S_f, b, b) = (S_f, \varepsilon)$ (4)

Konfigurationsfolge beim Akzeptieren von **aabb**:

$\delta(S_0, \varepsilon, k_0) = (S_f, S_G\, k_0)$ (1)

$\delta(S_f, \varepsilon, S_G) = (S_f, a\, S_G\, b) \mid (S_f, \varepsilon)$ (2), (2')

$\delta(S_f, a, a) = (S_f, \varepsilon)$ (3)

$\delta(S_f, b, b) = (S_f, \varepsilon)$ (4)

$(S_0, aabb, k_0) \Rightarrow_{(1)} (S_f, aabb, S_G\, k_0) \Rightarrow_{(2)} (S_f, aabb, a\, S_G\, b\, k_0)$

$\Rightarrow_{(3)} (S_f, abb, S_G\, b\, k_0) \Rightarrow_{(2)} (S_f, abb, a\, S_G\, b\, b\, k_0)$

$\Rightarrow_{(3)} (S_f, bb, S_G\, b\, b\, k_0) \Rightarrow_{(2')} (S_f, bb, b\, b\, k_0)$

$\Rightarrow_{(4)} (S_f, b, b\, k_0) \Rightarrow_{(4)} (S_f, \quad, k_0)$ = **akzeptierter Endzustand**

aabb $\in$ L(**KA**) = L(G) q. e. d.

- höhere Programmiersprachen können als kontextfreie Sprachen angesehen werden
- syntaktische Korrektheit kann auf das Wortproblem bei formalen Sprachen zurückgeführt werden
- Fragestellung: gehört ein gegebenes Wort **w** $\in$ **T*** zur kontext-freien Sprache L(G)?

Merke:

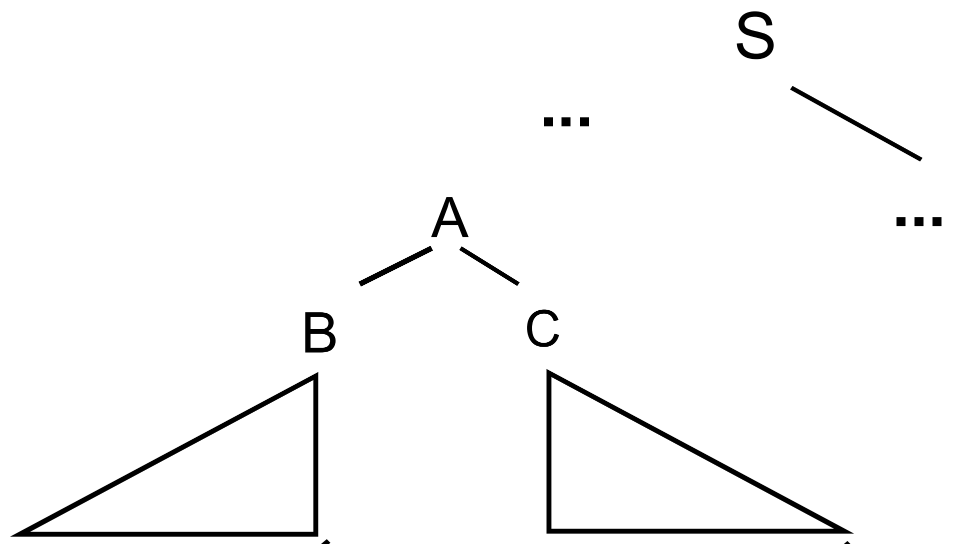
Zur Grammatik einer höheren Programmiersprache gibt es i. a. kei-nen zugehörigen deterministischen Kellerautomaten

$\rightarrow$ infolge dessen ist auch kein deterministisches Analyseverfahren herleitbar $\rightarrow$ theoretische Lösung ist das **CYK**-Verfahren

- Wenn ein Wort **w = a** der Länge 1 abgeleitet werden kann, dann sicherlich nur aufgrund der Regel **A → a**.
- Ist aber **$w = a_1a_2 \dots a_n$** ($n \geq 2$), dann kann **w** aus **A** nur deshalb ableitbar sein, weil eine Regel der Form **A → BC** angewandt worden ist.
- Von **B** aus wird dann ein Anfangsstück von **w** abgeleitet und von **C** aus das Endstück.

Folgerung:

Es muss also ein **k** mit **$1 \leq k \leq n$** geben, so dass gilt:



$$w = \underbrace{a_1\, a_2 \ldots a_k}\quad \underbrace{a_{k+1} \ldots a_n} := \mathbf{w_1\, w_2}$$

$\Rightarrow$ Damit ist es möglich, das Wortproblem für **w** der Länge **n** auf zwei entsprechende Entscheidungen für die Wörter $\mathbf{w_1}$ der Länge **k** und $\mathbf{w_2}$ der Länge **n-k** zurückzuführen.

Beschreibung des Algorithmus:

- Grammatik sei in der **Chomsky-Normalform** gegeben, d.h. es gibt nur Produktionen der Form $\mathbf{A \rightarrow a}$ oder $\mathbf{A \rightarrow BC}$.
- Sei $\mathbf{w = a_1 a_2 \ldots a_n}$ zu untersuchen.
- Betrachte Teilwort $a_i a_{i+1} \ldots a_j$ mit $1 \leq i \leq j \leq n$.
- Fasse alle Nonterminals A, aus denen $a_i a_{i+1} \ldots a_j$ ableitbar ist zur Menge $M_{ij} = \{ A \mid A \Rightarrow a_i a_{i+1} \ldots a_j \}$ zusammen.
- Dann gilt: $\mathbf{w \in L(G) \Leftrightarrow M_{1,n}}$ **enthält S**.

Konstruktion der Mengen $M_{i,j}$ - es gibt n(n+1)/2 davon - schrittweise:

0. Stufe: i=1,2,...n; j=i; $M_{1,1} = \{ A \mid A \rightarrow a_1\}$, $M_{2,2} = \{ A \mid A \rightarrow a_2\}$, $M_{3,3} = \ldots$

1. Stufe: i=1,2,...n-1; j=i+1;

$M_{1,2} = \{ A \mid A \rightarrow a_1a_2\} = \{ A \rightarrow BC \mid B \in M_{1,1}$ und $C \in M_{2,2} \}$;

$M_{2,3} = \ldots$

....

s. Stufe: i=1,2,...n-s; j=i+s: $M_{1,s+1} = \{ A \mid A \rightarrow a_1a_2\ldots a_{s+1} \}$

$= \{ A \rightarrow BC \mid B \in M_{1,k}$ und $C \in M_{k+1,s+1}$; $k = 1,2, \ldots s \}$; ...

Entscheidung nach **O(n^3)** Zeit-Schritten, ob **$S \in M_{1,n}$**

CYK-Algorithmus: // Eingabewort sei $w = a_1 a_2 \dots a_n$

FOR i = 1, …, n **DO**

  $M[i, i] := \{A \in M: \exists A \to a_i\}$;

ENDDO

FOR s = 1, …, n – 1 **DO**

  FOR i = 1, …, n – s **DO**

    $M[i, i+s] := \{\ \}$;

    FOR k = i, …, i + s – 1 **DO**

      $M[i, i+s] := M[i, i+s] \cup \{ A \in M: \exists A \to B\,C$ mit $B \in M[i, k] \ \wedge\ C \in M[k+1, i+s] \}$;

Fortsetzung:

```
            ENDDO
      ENDDO
ENDDO
IF S ∈ M[1, n] THEN
      RETURN "JA, w ∈ L(G)."
ELSE
      RETURN "NEIN, w ∉ L(G)."
ENDIF
```

Dreieckstabelle:

A → □= = = = = = = **B** = = = = = = = >

$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{1,4}$	...	$\mathbf{M_{1,n}}$
↑ $\mathbf{a_1}$	$M_{2,2}$	$M_{2,3}$	$M_{2,4}$	...	$M_{2,n}$
	↑ $\mathbf{a_2}$	$M_{3,3}$	$M_{3,4}$	...	$M_{3,n}$
		↑ $\mathbf{a_3}$	$M_{4,4}$	...	$M_{4,n}$
			↑ $\mathbf{a_4}$	...	
				...	$M_{n,n}$
					↑ $\mathbf{a_n}$

□ || || **C** || || V

Beispiel: L(**G**) = {$\mathbf{a^n b^n}$ | n ∈ **IN**} und w = **aabb**

S → AC | AB ; C → SB ; A → a ; B → b

A → □= = = = **B** = = = >

{A}	{-}	{-}	**{S}**	□
↑ **a**	{A}	{S}	{C}	‖ **C**
	↑ **a**	{B}	{-}	‖
		↑ **b**	{B}	V
			↑ **b**	

Der **CYK**-Algorithmus terminiert also mit der Antwort “**JA**” ⇒ w ∈ L(**G**).

Prinzipien der Syntaxanalyse

- Abarbeitung des Wortes bzw. des Programms grundsätzlich von links nach rechts
- Parsing-Verfahren:
  - bottom-up → Ableitungsbaum von unten nach oben aufgebaut
  - top-down → Ableitungsbaum von oben nach unten aufgebaut
  - oder gemischt