

# Theoretische Informatik

Prof. Dr. Barbara Staehle

HTWG Konstanz  
Fakultaet für Informatik

WS 2021/2022

## Teil IV

# Kontextfreie Sprachen und Kellerautomaten

# Teil IV Typ 2 Sprachen und PDA

## 1. Kontextfreie Sprachen

- 1.1 Definition und Eigenschaften
- 1.2 Normalformen und Pumping-Lemma für kontextfreie Sprachen
- 1.3 Entscheidungsprobleme und Abschlusseigenschaften

## 2. Kellerautomaten

- 2.1 Kellerautomaten
- 2.2 Kellerautomaten und kontextfreie Sprachen
- 2.3 Deterministische Kellerautomaten

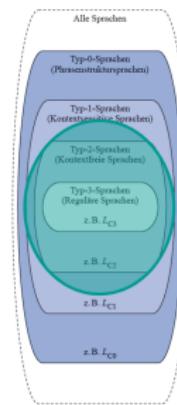
# Abschnitt 1

## Kontextfreie Sprachen

# Kontextfreie Sprachen - Einführung

## Kontextfreie Sprachen, Typ 2 Sprachen, Sprachen der Klasse $\mathcal{L}_2$

- Erweiterung (echte Obermenge) von  $\mathcal{L}_3$  in der Chomsky-Hierarchie
- mächtiger und komplizierter strukturiert als reguläre Sprachen, aber für Computer immer noch gut zu verarbeiten
- Modell der meisten Programmiersprachen und natürlichen Sprachen
- Erzeugung: kontextfreie Grammatiken
- Akzeptanz: Kellerautomaten (siehe Abschnitt 2)



Quelle:  
[Hoffmann, 2011]

## Definition

Eine formale Sprache heißt **kontextfrei**, falls Sie von einer kontextfreien Grammatik erzeugt wird.

# Beispiel: Erzeugung der Sprache $L_{C2}$

Erinnerung:  $L_{C2} = \{a^n b^n \mid n \in \mathbb{N}\}$ .

Wieso kann  $L_{C2}$  nicht von einer regulären Grammatik erzeugt werden?

Weil  $L_{C2}$  nicht regulär ist. Siehe Beweis via Pumping-Lemma bzw. gescheiterter Versuch der Konstruktion eines akzeptierenden DEAs.

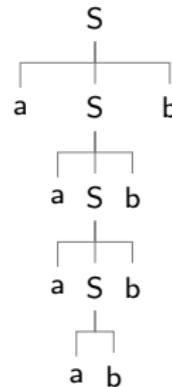
Kontextfreie Grammatik für  $L_{C2}$ :

$G_{C2} = (\{S\}, \{a, b\}, P, S)$  mit der Regelmenge  $P : S \rightarrow aSb|ab$

Ableitung des Wortes „aaaabbbb“:

$$\begin{aligned} S &\Rightarrow aSb \\ &\Rightarrow aaSbb \\ &\Rightarrow aaaSbbb \\ &\Rightarrow aaaabbbb \end{aligned}$$

Syntaxbaum zur Ableitung „aaaabbbb“



Auffällig: Der Syntaxbaum ist symmetrisch. Allgemein sind kontextfreie (und nicht reguläre Grammatiken) meist nicht linear.

# Beispiele aus der Welt der Programmiersprachen

**Dyck-Sprache  $D_n$ :** Menge der korrekt geklammerten Ausdrücke für  $n$  verschiedene Klammerpaare.

$D_2$  wird von der Grammatik  $G_2$  erzeugt:

- $G_2 = \{N, \Sigma, P, S\} = \{\{S, T\}, \{(,), [, ]\}, P, S\}$
- Die Produktionsmenge  $P$  besteht aus den Regeln:

$$S \rightarrow T \mid SS \mid [S] \mid (S)$$

$$T \rightarrow \varepsilon$$

- $D_2$  ist **kontextfrei**. (Struktur der Regeln, Pumping-Lemma)

**If-Else-Sprache  $IE$ :** z.B. in Java, C/C++, MATLAB, ..., kann, aber muss auf ein `if`, nicht unbedingt ein `else` folgen.

$IE$  wird von der Grammatik  $G_{IE}$  erzeugt:

- $G_{IE} = \{N, \Sigma, P, S\} = \{\{S, T\}, \{\text{if, else}\}, P, S\}$
- Die Produktionsmenge  $P$  besteht aus den Regeln:

$$S \rightarrow T \mid SS \mid \text{if} S \mid \text{if} S \text{else} S$$

$$T \rightarrow \varepsilon$$

- $IE$  ist **kontextfrei**. (Struktur der Regeln, Pumping-Lemma)

# Beispiel zum Mitdenken

Beispiel:  $K = \{a^i b^{i-1} c^j \mid i, j \in \mathbb{N}\}$

- Grammatik  $G$  erzeugt  $K$ :  $\mathcal{L}(G) = K$ .
- $G = (\{S, B, C\}, \{a, b, c\}, P, S)$

$$S \rightarrow BC$$

- $P : B \rightarrow aBb \mid a$
- $C \rightarrow cC \mid \varepsilon$

- Ableitung des Wortes „aabbc“:

$$S \Rightarrow BC$$

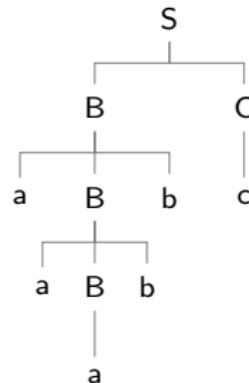
$$\Rightarrow aBbC$$

$$\Rightarrow aaBbbC$$

$$\Rightarrow aaabbC$$

$$\Rightarrow aaabbc$$

Syntaxbaum zur Ableitung  
„aabbc“:



**Beobachtung:** Struktur von Grammatik und Ableitungsbaum nicht sehr regelmäßig. Daher für Computer nicht sehr gut zu verarbeiten, Ableitungsschritte nicht sehr vorhersagbar.

**Lösung:** Optimierung von Grammatiken durch Normalformen

# Chomsky-Normalform

## Definition

Eine Grammatik  $G = (N, \Sigma, P, S)$  liegt in **Chomsky-Normalform** (CNF) vor, wenn alle Regeln die Form

$$\begin{array}{l} A \rightarrow \sigma \quad \text{oder} \\ A \rightarrow BC \end{array}$$

besitzen, mit  $A, B, C \in N$  und  $\sigma \in \Sigma$ .

**Verdeutlichung:** Eine Grammatik ist in CNF, wenn auf **jeder linken Regelseite nur ein Nonterminal steht**, auf **jeder rechten Regelseite entweder genau ein Terminal oder genau zwei Nonterminale**.

## Satz

Jede kontextfreie Grammatik  $G$  mit  $\varepsilon \notin \mathcal{L}(G)$  lässt sich in die Chomsky Normalform überführen.

# Erzeugung einer Chomsky-Normalform

1. **Elimination der  $\varepsilon$ -Regeln** Alle Regeln der Form  $A \rightarrow \varepsilon$  werden eliminiert, indem die Ersetzung von  $A$  durch  $\varepsilon$  in allen anderen Regel vorweggenommen wird.
2. **Elimination von Kettenregeln** Jede Kettenregel der Form  $A \rightarrow B$  mit  $A, B \in N$  trägt nicht zur Produktion von Terminalzeichen bei. Dieser werden eliminiert, indem man auf der rechten Seite die Ersetzung vorweg nimmt.
3. **Separation von Terminalzeichen** Jedes Terminalzeichen  $\sigma$ , das in Kombination mit anderen Symbolen auftaucht, wird durch ein neues Nonterminal  $N_\sigma$  ersetzt und die Menge der Produktionen durch die Regel  $N_\sigma \rightarrow \sigma$  ergänzt.
4. **Elimination von mehrelementigen Nonterminalketten** Alle Produktionen der Form  $A \rightarrow B_1B_2\dots B_n$  werden in die Produktionen  $A \rightarrow A_{n-1}B_n, A_{n-1} \rightarrow A_{n-2}B_{n-1}, \dots, A_2 \rightarrow B_1B_2$  zerteilt. Nach der Ersetzung sind alle längeren Nonterminalketten vollständig beseitigt und die CNF erreicht.

# Beispiel zum Mitdenken: Erzeugen einer CNF

Gegeben: Grammatik  $G$  mit Regeln  $P$  und  $\mathcal{L}(G) = K$ .

Startpunkt ( $P$ )	Ergebnis 1: ( $P_1$ )	Ergebnis 2: ( $P_2$ )	Ergebnis 3: ( $P_3$ )	Ergebnis 4: ( $P_4$ )
$S \rightarrow BC$	$S \rightarrow BC$	$S \rightarrow BC$	$S \rightarrow BC$	$S \rightarrow BC$
$B \rightarrow aBb$	<del><math>S \rightarrow BC</math></del>	<del><math>S \rightarrow aBb</math></del>	<del><math>S \rightarrow NaBN_b</math></del>	$S \rightarrow S_2N_b$
$B \rightarrow a$	$B \rightarrow aBb$	$S \rightarrow a$	$S \rightarrow a$	$S_2 \rightarrow NaB$
$C \rightarrow cC$	$B \rightarrow a$	$B \rightarrow aBb$	$B \rightarrow NaBN_b$	$S \rightarrow a$
<del><math>\emptyset \rightarrow \emptyset</math></del>	$C \rightarrow cC$	$B \rightarrow a$	$B \rightarrow a$	$B \rightarrow S_2N_b$
Schritt 1: Streiche $\varepsilon$ -Regeln	Schritt 2: Streiche Kettenregeln	Schritt 3: Eliminiere Terminale		Schritt 4: Eliminiere Non- terminalketten

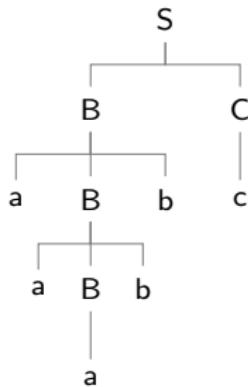
Produktionsregeln  $P_4$  sind die CNF der ursprünglichen Regeln  $P$  der Grammatik  $G$ . Offensichtlich gilt für die Grammatik  $G_C = (\{S, B, C, N_a, N_b, N_c, S_2\}, \{a, b\}, P_4, S)$ :  $\mathcal{L}(G_C) = \mathcal{L}(G) = K$ .

# Vergleich der Ableitungen nach $G$ und $G_C$

Ableitung von „aaabbc“ nach  $G$ :

$$\begin{aligned} S &\Rightarrow BC \\ &\Rightarrow aBbC \\ &\Rightarrow aaBbbC \\ &\Rightarrow aaabbC \\ &\Rightarrow aaabbc \end{aligned}$$

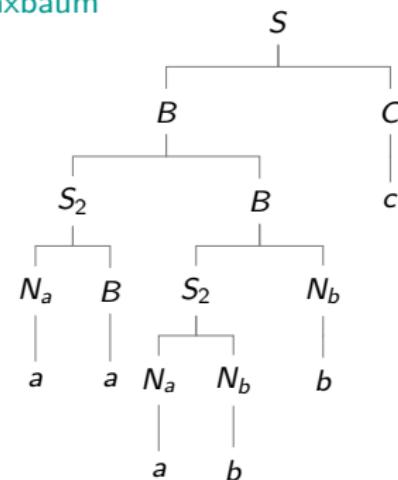
Syntaxbaum



Ableitung von „aaabbc“ nach  $G_C$ :

$$\begin{aligned} S &\Rightarrow S_2 A \\ &\Rightarrow BC \\ &\Rightarrow S_2 N_b C \\ &\Rightarrow N_a B N_b N_b C \\ &\Rightarrow N_a N_a B N_b N_b C \\ &\Rightarrow^* aaabbc \end{aligned}$$

Syntaxbaum



# Eigenschaften der Chomsky-Normalform

Bemerkung:

- Der Syntaxbaum einer Ableitung nach einer Grammatik in CNF ist immer ein **Binärbaum**.
- **Vorteile:** vereinfachter Beweis von Eigenschaften, vereinfachte Arbeitsweise für CYK-Algorithmus und Parser.

**Grund:** Jedes Nonterminal  $N$  wird abgebildet entweder auf

- zwei Nonterminale  $\Rightarrow N$  bleibt innerer Knoten und hat zwei Kinder,
- ein Nonterminal und ein Terminal  $\Rightarrow N$  bleibt innerer Knoten und hat zwei Kinder,
- ein Terminal  $\Rightarrow N$  hat das Terminal als Blatt und hat ein Kind

**Exkurs: Alternative Definition der CNF** von  $G = (N, \Sigma, P, S)$ :

- alle Produktionen haben die Form  $S \rightarrow \varepsilon$ ,  $A \rightarrow \sigma$ , oder  $A \rightarrow BC$
- $A \in N, B, C \in N \setminus \{S\}$  und  $\sigma \in \Sigma$
- Alle kontextfreien Sprachen (auch solche, die  $\varepsilon$  enthalten), lassen sich in solch eine Form überführen.

# Exkurs: Greibach-Normalform und Backus-Naur-Form

## Greibach-Normalform (GNF) (nach [Hedtstück, 2012])

- Sheila Greibach 1965: Verschärfung der CNF
- Jede Regel hat die Form  $X \rightarrow aY_1 \dots Y_n$  mit  $X, Y_1, \dots, Y_n \in N, a \in \Sigma, n \in \mathbb{N}_0$
- Folgerung: reguläre Grammatiken sind Teilmenge der kontextfreien (für alle regulären Regeln gilt  $n \in \{0, 1\}$ ).
- Aus der GNF kann ein sehr schneller Parser hergeleitet werden, da bei jedem Schritt ein Terminalsymbol entsteht.
- Aus der GNF kann einfach ein Kellerautomat abgeleitet werden.

## Backus-Naur-Form (BNF)

- John Backus 1959, Peter Naur 1963
- Zweck: korrekte Spezifikation kontextfreier Programmiersprachen
- Notation
  - ▶ „ ::= “ statt „ $\rightarrow$ “
  - ▶ „<>“ kennzeichnen Nonterminale
  - ▶ „|“ steht für Alternativen
- **Erweiterte Backus-Naur-Form:** Nikolaus Wirth 1977, ISO/IEC 14977

# Beispiel: Auszug aus der Algol60 Syntax in BNF

## Algol-Grammatik (Auszug)

```
<program>      ::= <block> | <compound statement>
<block>        ::= <unlabelled block> | <label>: <block>
<unlabelled block> ::= <block head> ; <compound tail>
<block head>   ::= 'BEGIN' <declaration> | <block head> ; <declaration>
<compound statement> ::= <unlabelled compound> |
                         <label>: <compound statement>
<unlabelled compound> ::= 'BEGIN' <compound tail>
<compound tail>  ::= <statement> 'END' | <statement> ; <compound tail>
<declaration>    ::= <type declaration> | <array declaration> |
                         <switch declaration> | <procedure declaration>
<type declaration> ::= <local or own type> <type list>
<local or own type> ::= <type> | 'OWN' <type>
<type>          ::= 'REAL' | 'INTEGER' | 'BOOLEAN'
<type list>      ::= <simple variable> | <simple variable> , <type list>
<array declaration> ::= 'ARRAY' <array list> |
                         <local or own type> 'ARRAY' <array list>
<array list>     ::= <array segments> | <array list> , <array segment>
<array segment>   ::= <array identifier> [ <bound pair list> ] | |
                         <array identifier> , <array segment>
<array identifier> ::= <identifier>
<bound pair list> ::= <bound pair> | <bound pair list> , <bound pair>
<bound pair>       ::= <lower bound> : <upper bound>
<upper bound>     ::= <arithmetic expression>
<lower bound>     ::= <arithmetic expression>
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24

Quelle: [Hoffmann, 2011]

## Etwas moderner: Python, Full Grammar specification

# Beispiel zum Mitdenken I

**Beispiel:**  $L_P = \{w \in \{a, b, \dots, z\}^* \mid w \text{ ist ein Palindrom}\}$

- Palindrome sind Worte, die von vorne und hinten gelesen gleich sind.
- Beispiele:  $\varepsilon$ , anna, otto, xx, z, reliefpfeiler, abcxyzycba, vitalernebelmitsinnistimlebenrelativ, ...

**Gesucht:** Grammatik  $G_P$  mit  $\mathcal{L}(G_P) = L_P$ .

**Lösung:** Sei  $G_P = (N, \Sigma, P, S)$  mit

- $\Sigma = \{ a, b, \dots, z \}$
- $N = \{S\}$
- $P : S \rightarrow a \mid b \mid \dots \mid z \mid a \, S \, a \mid b \, S \, b \mid \dots \mid z \, S \, z \mid \varepsilon$
- $\mathcal{L}(G_P) = \{\varepsilon, a, \dots, z, aa, \dots, zz, aaa, \dots, zzz, \dots\}$

# Beispiel zum Mitdenken II

**Beispiel:** Ableitung der Palindrome gnudung, gnuuduung und gnuuuduuung

Ableitung von

„gnudung“:

$$\begin{aligned} S &\Rightarrow gSg \\ &\Rightarrow gnSng \\ &\Rightarrow gnuSung \\ &\Rightarrow gnudung \end{aligned}$$

Ableitung von

„gnuuduung“:

$$\begin{aligned} S &\Rightarrow gSg \\ &\Rightarrow gnSng \\ &\Rightarrow gnuSung \\ &\Rightarrow gnuuSuung \\ &\Rightarrow gnuuduung \end{aligned}$$

Ableitung von

„gnuuuduuung“:

$$\begin{aligned} S &\Rightarrow gSg \\ &\Rightarrow gnSng \\ &\Rightarrow gnuSung \\ &\Rightarrow gnuuSuung \\ &\Rightarrow gnuuuSuuung \\ &\Rightarrow gnuuuduuung \end{aligned}$$

**Beobachtung:** gnudung  $\in L_P$ , genauso wie gnu $^x$ du $^x$ ng  $\in L_P$ , für  $x \in \mathbb{N}_0$ .

**Begründung:** Für das Ziel „gnudung“, kann das Nonterminal  $S$  in „gnuSung“ z.B. zum Terminal „d“, oder zu „uSu“ abgeleitet werden. „uSu“ kann wiederum z.B. zu „d“ oder zu „uSu“ abgeleitet werden ... Man sagt, die **Mittelteile** „u“ des Wortes „gnudung“ können unendlich oft **aufgepumpt** werden.

# Eine formellere Herleitung des Pumpinglemmas I

- Wir betrachten eine Grammatik  $G = (N, \Sigma, P, S)$  in Chomsky-Normalform.
- Wir setzen  $j := 2^{|N|}$  und wählen ein beliebiges Wort  $\omega \in \mathcal{L}(G)$  mit  $|\omega| \geq j$ .
- $G$  ist in CNF.  $\Rightarrow$  Syntaxbaum für  $\omega$  ist Binärbaum mit mindestens  $j = 2^{|N|}$  Blättern.
- Auf einem Pfad mit Länge  $\geq |N|$  müssen mindestens  $|N| + 1$  NTs sein
  - $\Rightarrow$  ein NT muss mehrfach auftauchen. Dieses sei  $A$ .
  - $\Rightarrow \omega$  lässt sich in Segmente  $\omega = uvwxy$  zerlegen (siehe Syntaxbaum)
- aus  $A$  kann nur über Regel der Form  $A \rightarrow BC$  ein weiteres  $A$  werden ( $G$  ist in CNF).
  - $\Rightarrow$  mindestens eines der Segmente  $v$  oder  $x$  muss ein Zeichen enthalten, d.h.,  $|vx| \geq 1$  (da jedes NT zu einem T wird).

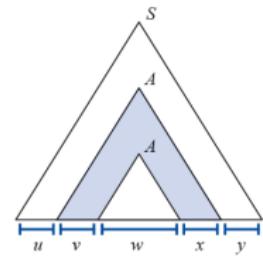


Bild: Struktur kontextfreier Worte [Hoffmann, 2011]

# Eine formellere Herleitung des Pumpinglemmas II

- Die Position der As im Syntaxbaum kann so gewählt werden, dass alle weiteren Vorkommen von A, näher an der Wurzel liegen und alle anderen NTs, die näher an den Blättern liegen, paarweise verschieden sind.  
⇒ Das obere A ist maximal  $|N|$  Schritte von den Blättern entfernt und der von diesem A aufgespannte Syntaxbaum für  $vwx$  kann höchstens  $j = 2^{|N|}$  Blätter haben.
- Durch das doppelte A kann „aufgepumpt“ werden: genauso wie das Wort  $uvwxy$  können  $uv^2wx^2y$ ,  $uv^3wx^3y$ , ..., bzw. auch  $uv^0wx^0y$  abgeleitet werden.

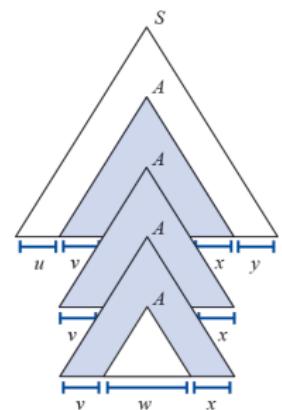
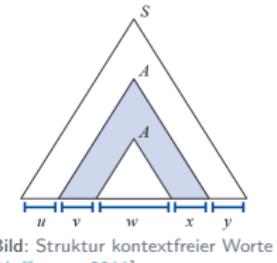


Bild: Aufpumpen der Mittelstücke  
[Hoffmann, 2011]

# Das Pumping-Lemma (PL) für kontextfreie Sprachen

## Satz

Für jede kontextfreie Sprache  $L$  existiert ein  $j \in \mathbb{N}$ , so dass sich alle Wörter  $\omega \in L$  mit  $|\omega| \geq j$  in der folgenden Form darstellen lassen:

$$\omega = uvwxy \quad \text{mit } |vx| \geq 1 \text{ und } |vwx| \leq j .$$

Weiterhin ist für alle  $i \in \mathbb{N}_0$  mit  $\omega$  auch das Wort  $uv^iwx^iy$  in  $L$  enthalten.

### Bemerkungen:

- Für  $\omega = uvwxy$  gilt, dass  $v \neq \varepsilon \vee x \neq \varepsilon$  (entweder  $v$  oder  $x$  dürfen das leere Wort sein, aber nicht beide).  $u = \varepsilon$ ,  $w = \varepsilon$  oder  $y = \varepsilon$  ist zulässig.
- Für  $x = y = \varepsilon$  ergibt sich das PL für reguläre Sprachen als Spezialfall des PL für kontextfreie Sprachen.
- Alle Sprachen  $L$  die nur Wörter mit maximal Länge  $n$  haben, sind kontextfrei. (Wähle  $j = n + 1$ , dann ist  $\{\omega \in L \mid |\omega| \geq j\} = \emptyset$ .)

# Anwendungen des Pumping Lemma

Zu beachten:

- Nicht jede Sprache die das PL erfüllt ist kontextfrei.
- Jede Sprache, die das PL **nicht** erfüllt, ist **nicht** kontextfrei.
- Das PL ist also ein gutes Instrument, um nachzuweisen, dass eine Sprache nicht kontextfrei ist.

# Beispiel zum Mitdenken

**Behauptung:** Die Sprache  $L_{C1} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$  ist nicht kontextfrei

**Beweis durch Widerspruch:**

- **Nehmen wir an**,  $L_{C1}$  wäre kontextfrei.
- **Dann muss es nach dem Pumping-Lemma** ein  $j \in \mathbb{N}_0$  existieren, so dass sich jedes Wort  $\omega \in L_{C1}$  mit  $|\omega| \geq j$  in der Form  $uvwxy$  darstellen lässt, wobei  $|vx| \geq 1$  und  $|vwx| \leq j$ .
- Betrachten wir **das spezielle Wort**  $\omega = a^j b^j c^j$ . Aus  $|vx| \geq 1$  folgt, dass  $v$  oder  $x$  aus genau einem Buchstaben ( $a$ ,  $b$ , oder  $c$ ) bestehen müssen. Ansonsten würde ein Aufpumpen von z.B.  $x = ab$  zu Wörter führen die  $(ab)^n$  beinhalten und nicht in  $L_{C1}$  sind.
- **Nehmen wir also an**,  $x = a^k$ ,  $0 < k \leq j$ . Damit ist das Wort  $uv^0wx^0y = a^{j-k}b^j c^j$  (da das Wort  $x$  entfernt wurde) kein Element von  $L_{C1}$ .
- Daher ist  $L_{C1}$  **nicht kontextfrei**.

# Exkurs: Entscheidungsprobleme kontextfreier Sprachen

Problem	Eingabe	Fragestellung	Entscheidbar?
Wortproblem	$L$ , Wort $\omega \in \Sigma^*$	Ist $\omega \in L$ ?	✓ Ja
Leerheitsproblem	$L$	Ist $L = \emptyset$	✓ Ja
Endlichkeitsproblem	$L$	Ist $ L  < \infty$ ?	✓ Ja
Äquivalenzproblem	$L_1$ und $L_2$	Ist $L_1 = L_2$ ?	✗ Nein

Tabelle: Entscheidungsprobleme für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_2$ 

## Bemerkung:

- Das Wortproblem lässt sich mit dem **CYK-Algorithmus** (Autoren: Cocke, Younger, Kasami) effizient lösen.  
Schlüsselidee: dynamische Programmierung.

# CYK-Algorithmus I

Gegeben sei eine Grammatik  $G = (N, \Sigma, P, S)$  in **Chomsky-Normalform** (CNF) und  $\omega \in \Sigma^*$ .

**Wortproblem:** Lässt sich  $\omega$  aus  $S$  ableiten, gilt also  $S \in \mathcal{L}(G)$ ?

**Beobachtung:** Wann lässt sich  $\omega$  aus einem Nonterminal  $A \in N$  ableiten?

- Wenn  $|\omega| = 1$  dann muss eine Regel  $A \rightarrow \omega$  in  $P$  existieren.
- Wenn  $|\omega| > 1$ 
  - ▶ dann lässt sich  $\omega$  als Zeichenkette  $\sigma_1, \dots, \sigma_n$  mit  $n \geq 2$  schreiben und es kann aus dem Nonterminal  $A$  nur mit Hilfe einer Regel  $A \rightarrow BC$  entstanden sein.
  - ▶ Es gibt also ein  $k$  mit  $1 \leq k < n$ , so dass das Teilwort  $\sigma_1, \dots, \sigma_k$  von  $B$  abgeleitet werden kann und das Teilwort  $\sigma_{k+1}, \dots, \sigma_n$  von  $C$ .

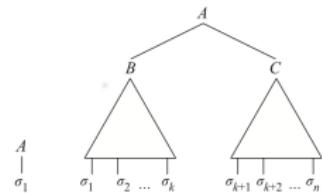


Bild: Struktur von CNF-Syntaxbäumen  
[Hoffmann, 2011]

# CYK-Algorithmus II

- Definiere mit Hilfe dieser Eigenschaft einen einfachen rekursiven Algorithmus für das Wortproblem.
- Nachteil: exponentielle (sehr lange) Laufzeit der rekursiven Implementierung
- Reduktion der Laufzeit durch dynamische Programmierung
- Idee:
  - ▶ Berechne Zwischenergebnisse nur einmal und speichere sie in einer Matrix  $cyk$ , wobei  $cyk[i][j]$  alle Non-Terminale enthält, aus denen das Teilwort  $\sigma_i, \dots, \sigma_{i+j-1}$  abgeleitet werden kann.  
**Achtung:** im Unterschied zur normalen Matrixnotation speichert  $cyk[i][j]$  den Wert in **Zeile  $j$  und Spalte  $i$**  der Matrix
  - ▶ Die erste Zeile der Matrix kann mit den Ableitungsregeln für Terminate in CNF gefüllt werden:  $cyk[i][1] = \{A | (A \rightarrow \sigma_i) \in P\}$
  - ▶ Die restlichen Zeilen ergeben sich rekursiv aus den vorhergehenden:  
$$cyk[i][j] = \{A | (A \rightarrow BC) \in P, B \in cyk[i][k], C \in cyk[i+k][j-k], 1 \leq k < j\}$$
  - ▶  $\sigma \in L(G)$  genau dann wenn  $S \in cyk[1, n]$ .

# Beispiele zum Mitdenken

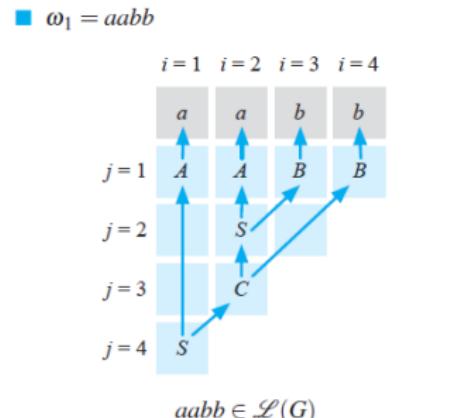
Beispiel:  $G = (\{S, A, B, C\}, \{a, b\}, P, S)$  mit

$$P = \begin{array}{ll} S & \rightarrow AB|AC \\ & \quad A \rightarrow a \\ C & \rightarrow SB \\ & \quad B \rightarrow b \end{array}$$

Ist das Wort  $aabb$  in der von der Grammatik  $G$  erzeugten Sprache  $\mathcal{L}(G)$ ?

$cyk$	$i=1$	$i=2$	$i=3$	$i=4$
	a	a	b	b
$j = 1$	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$
$j = 2$	$\{\}$	$\{S\}$	$\{\}$	
$j = 3$	$\{\}$	$\{C\}$		
$j = 4$	$\{S\}$			

$S$  ist in  $cyk[1, 4]$  enthalten, damit gilt:  $aabb \in \mathcal{L}(G)$



Quelle: [Hoffmann, 2015]

# Beispiele zum Mitdenken

Beispiel:  $G = (\{S, A, B, C\}, \{a, b\}, P, S)$  mit

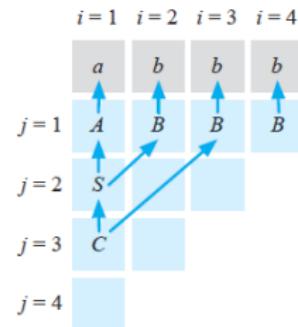
$$P = \begin{array}{ll} S & \rightarrow AB|AC \\ & \quad A \rightarrow a \\ C & \rightarrow SB \\ & \quad B \rightarrow b \end{array}$$

Ist das Wort  $abbb$  in der von der Grammatik  $G$  erzeugten Sprache  $\mathcal{L}(G)$ ?

$cyk$	$i=1$	$i=2$	$i=3$	$i=4$
	a	b	b	b
$j = 1$	$\{A\}$	$\{B\}$	$\{B\}$	$\{B\}$
$j = 2$	$\{S\}$	$\{\}$	$\{\}$	
$j = 3$	$\{C\}$	$\{\}$		
$j = 4$	$\{\}$			

$S$  ist **nicht** in  $cyk[1, 4]$  enthalten,  
damit gilt:  $abbb \notin L(G)$

■  $w_2 = abbb$



$abbb \notin \mathcal{L}(G)$

Quelle: [Hoffmann, 2015]

# CYK-Algorithmus Zusammenfassung

## CYK-Algorithmus

```
// Eingabe: Grammatik  $G = (V, \Sigma, P, S)$ 
//          Wort  $\omega = \sigma_1, \dots, \sigma_n$ 
// Ausgabe: true , wenn  $\omega \in \mathcal{L}(G)$ , false wenn  $\omega \notin \mathcal{L}(G)$ 

boolean cyk(G, ω)
{
    // Berechne die erste Zeile ...
    for (i = 1; i ≤ n; i++) {
        cyk[i][1] = {A | (A → σi) ∈ P};
    }

    // Berechne alle restlichen Zeilen ...
    for (j = 2; j ≤ n; j++) {
        for (i = 1; i ≤ n+1-j; i++) {
            cyk[i][j] = ∅;
            for (k = 1; k < j; k++) {
                cyk[i][j] = cyk[i][j] ∪ {A | (A → BC) ∈ P, B ∈ cyk[i][k], C ∈ cyk[i+k][j-k]};
            }
        }
    }
    return S ∈ cyk[1][n];
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22

Bild: CYK Algorithmus in Pseudocode [Hoffmann, 2015]

Animierte gif: Parsen eines Satzes ([Wikipedia](#))

# Exkurs: Abschlusseigenschaften kontextfreier Sprachen

Operation	Eingabe	Fragestellung	Antwort
Vereinigung	$L_1, L_2$	Ist $L_1 \cup L_2 \in \mathcal{L}_2$ ?	✓ Ja
Schnitt	$L_1, L_2$	Ist $L_1 \cap L_2 \in \mathcal{L}_2$ ?	✗ Nein
Komplement	$L$	Ist $\Sigma^* \setminus L \in \mathcal{L}_2$ ?	✗ Nein
Produkt	$L_1, L_2$	Ist $L_1 L_2 \in \mathcal{L}_2$ ?	✓ Ja
Stern	$L$	Ist $L^* \in \mathcal{L}_2$ ?	✓ Ja

Tabelle: Abschlusseigenschaften für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_3$ 

## Bemerkung:

- Beweis der positiven Fragen über Grammatikkonstruktion.
- Gegenbeispiele zu Fragen „Schnitt“ und „Komplement“: Seien  $L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}_0\}$ ,  $L_2 = \{a^m b^n c^n \mid n, m \in \mathbb{N}_0\} \in \mathcal{L}_2$ .
  - ▶  $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}_0\} \notin \mathcal{L}_2$  (siehe Beispiel für PL).
  - ▶  $\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2 \notin \mathcal{L}_2$  (De Morgan).

## Abschnitt 2

### Kellerautomaten

# Können [DN]EAs Typ 2 Sprachen erkennen?

**Erinnerung:** Für jede reguläre Sprache  $L$  existiert ein DEA  $A$  mit  $\mathcal{L}(A) = L$ .

**Frage:** Was ist mit kontextfreien Sprachen? Lässt sich z.B. ein DEA oder NEA finden, der  $L_{C_2} = \{a^n b^n \mid n \in \mathbb{N}\}$  akzeptiert?

**Antwort:** Nein.

**Veranschaulichung:** Konstruiere DEAs  $A_n$ , die Teilmengen von  $L_{C_2}$  für **feste** maximale Wortlängen akzeptieren:  $\mathcal{L}(A_n) = \{a^i b^i \mid 1 \leq i \leq n\}$ .

**Beobachtung 1:**  $A_n$  hat  $2n + 1$  Zustände.

**Beobachtung 2:**  $L_{C_2} = \bigcup_{n=1}^{\infty} \mathcal{L}(A_n)$

**Folgerung:** DEA  $A$  mit  $\mathcal{L}(A) = L_{C_2}$  hätte unendlich viele Zustände ↴

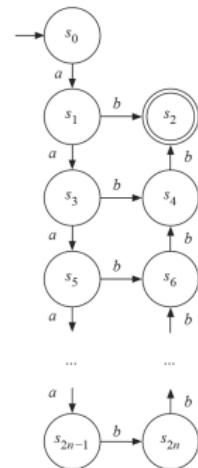


Bild:  $A_n$  [Hoffmann, 2011]

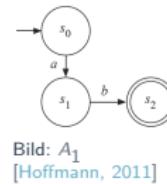


Bild:  $A_1$   
[Hoffmann, 2011]

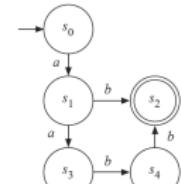


Bild:  $A_2$   
[Hoffmann, 2011]

# Kellerautomat (PDA)

## Definition

Ein **Kellerautomat**, kurz PDA, ist ein 5-Tupel  $P = (Q, \Sigma, \Gamma, \delta, q_0)$  mit

- der endlichen **Zustandsmenge**  $Q$ ,
- dem endlichen **Eingabealphabet**  $\Sigma$  mit  $\varepsilon \notin \Sigma$ , sowie  $\Sigma \cap \Gamma = \emptyset$
- dem endlichen **Kelleralphabet**  $\Gamma$  mit dem **Kellerzeichen**  $\# \in \Gamma$
- der **Zustandsübergangsfunktion**  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  mit  $|\delta(q, \omega, \gamma)| < \infty$  für alle  $q, \omega, \gamma$ .
- dem **Startzustand**  $q_0$ .

## Bemerkungen:

- PDA kommt vom englischen „pushdown automaton“
- Zustandsübergänge sind **nichtdeterministisch** und können auch ohne ein gelesenes Eingabe-Zeichens passieren ( **$\varepsilon$ -Übergänge**).
- Alternative (gleichwertige) Definitionen für PDAs beinhalten eine Finalmenge. Unsere Definition **akzeptiert bei leerem Keller**.

# Weitere Bemerkungen zu PDAs

- Im **Kellerspeicher, Stack, Stapel** werden neue Zeichen immer oben drauf gelegt (Push) und können auch nur von oben wieder entnommen werden (Pop).

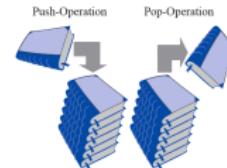


Bild: Stacks arbeiten nach dem LIFO (Last-in-first-out) Prinzip [Hoffmann, 2011]

- Das Eingabealphabet  $\Sigma$  und das Kelleralphabet  $\Gamma$  sind disjunkt. Elemente aus  $\Sigma$  werden meist als Klein-, Elemente aus  $\Gamma$  als Großbuchstaben dargestellt.
- In jedem Verarbeitungsschritt liest der PDA ein Eingabezeichen  $\sigma$  ein (oder wählt einen  $\varepsilon$ -Übergang und liest nichts) und entfernt das oberste Kellerzeichen  $\gamma$ .
- In Abhängigkeit von  $\sigma, \gamma$  und des aktuellen Zustands  $q$  geht der PDA in einen Folgezustand  $q'$  über und schreibt (ggf. mithilfe mehrerer Push-Operationen) eine Zeichenkette  $\gamma'_0 \dots \gamma'_i \in \Gamma^*$  (oder nichts, in dem Fall wird  $\gamma$  einfach entfernt) in den Keller.
- Ein PDA startet im Zustand  $q_0$  und mit dem Kellerzeichen  $\#$  im Keller.

# Arbeitsweise eines PDA I

- Der PDA  $P$  beginnt im Startzustand  $q_0$  das Eingabewort  $\omega$ , sowie das Kellerzeichen  $\#$  zu verarbeiten
- Für das Eingabewort  $\omega = \sigma_1\sigma_2\dots\sigma_n$  durchläuft  $P$  in Abhängigkeit des jeweiligen obersten Kellerzeichens  $\gamma_i$  **mehrere mögliche Zustandsfolgen**  $q_0, \dots, q_n$  mit  $q_i \in \delta(q_{i-1}, \sigma_i, \gamma_i)$ .
- $P$  stoppt (auf dem jeweiligen Pfad), wenn es für  $\sigma_i$ ,  $q_{i-1}$  und  $\gamma_i$  keinen nächsten Zustand gibt, (bzw. wenn der Keller leer ist).
- Das Wort  $\omega$  gilt genau dann als **akzeptiert**, wenn ein Pfad existiert, auf dem  $\omega$  vollständig eingelesen wurde und der Keller leer ist.

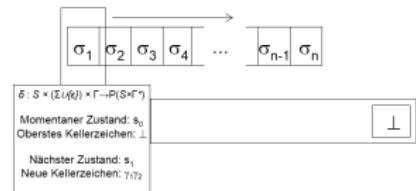


Bild: Start der Verarbeitung von  $\omega$

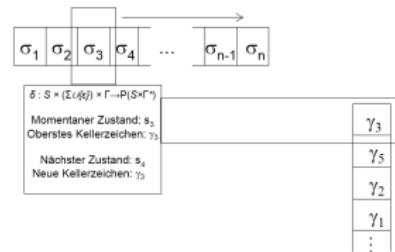


Bild: Eine Option mitten in der Verarbeitung von  $\omega$

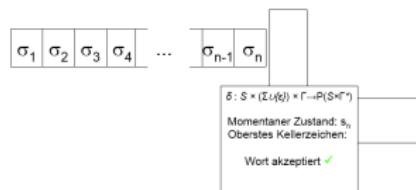


Bild: Eine Option am Ende der Verarbeitung von  $\omega$

# Arbeitsweise eines PDA II

Beispiel: PDA  $P_1 = (Q, \Sigma, \Gamma, \delta, q_0)$

- $Q := \{q_0, q_1\}$
- $\Sigma := \{a, b\}$
- $\Gamma := \{A, \#\}$
- $\delta(q_0, a, \#) := \{(q_0, A\#)\}$
- $\delta(q_0, a, A) := \{(q_0, AA)\}$
- $\delta(q_0, b, A) := \{(q_1, \varepsilon)\}$
- $\delta(q_1, b, A) := \{(q_1, \varepsilon)\}$
- $\delta(q_1, \varepsilon, \#) := \{(q_1, \varepsilon)\}$

Frage: Akzeptiert  $P_1$  das Wort  $\omega = aabb$ ?

Lösung: Bestimme schrittweise angenommene Zustände und geschriebene Kellersymbole.

Beispiel 1:  $\omega = aabb$  ✓

Schritt	Zustand	$\omega$	Keller
0	$q_0$	$aabb$	#
1	$q_0$	$abb$	$A\#$
2	$q_0$	$bb$	$AA\#$
3	$q_1$	$b$	$A\#$
4	$q_1$	$\varepsilon$	#
5	$q_1$	$\varepsilon$	$\varepsilon$

Beispiel 2:  $\omega = aaabb$  ↘

Schritt	Zustand	$\omega$	Keller
0	$q_0$	$aaabb$	#
1	$q_0$	$aabb$	$A\#$
2	$q_0$	$abb$	$AA\#$
3	$q_0$	$bb$	$AAA\#$
4	$q_1$	$b$	$AA\#$
5	$q_1$	$\varepsilon$	$A\#$

# Arbeitsweise eines PDA III

Grafische Darstellung eines PDA als **erweitertes Zustandsübergangsdiagramm** möglich, aber aufwändig:

- Erweitere von DEA / NEA bekanntes Zustandsübergangsdiagramm durch Berücksichtigung des Kellers
- Beschrifte jeden Pfeil mit (**gelesenes Eingabezeichen , oberstes Kellersymbol ; auf den Keller zu schreibende Symbole**)
- Berücksichtige bei der Simulation der Verarbeitung eines Wortes auch den Keller

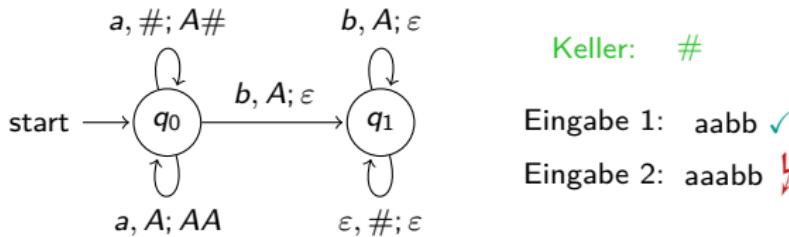


Bild: Erweitertes Zustandsübergangsdiagramm für  $P_1$

# Beschreibung eines PDA durch Konfigurationen

Beobachtung: Das Verhalten eines NEAs hängt vom aktuell gelesenen Zeichen und dem aktuellen Zustand ab.

Das Verhalten eines PDAs hängt darüber hinaus noch vom Inhalt des Stacks ab. Man verwendet daher **Konfigurationen** um einen PDA darzustellen:

## Definition

Die **Konfiguration**  $k = (q, \omega, \gamma)$  eines PDAs  $P = (Q, \Sigma, \Gamma, \delta, q_0)$  enthält

- $q \in Q$ , den aktuellen Zustand,
- $\omega \in \Sigma^*$ , das verbleibende Eingabewort,
- $\gamma \in \Gamma^*$ , den Inhalt des Stacks.

Wir nennen  $K_P$  die **Menge aller Konfigurationen für  $P$** .

**Bemerkung:** Nur die Analyse der Veränderung der Konfiguration erlaubt das Nachvollziehen der Arbeitsweise des Kellerautomats, da nicht nur Zustände, sondern auch der Inhalt des Kellers wichtig sind.

# Beschreibung eines PDA durch die Übergangsrelation

Die Arbeitsweise eines PDAs wird durch die Verkettung von Konfigurationen mittels der **Übergangsrelation** beschrieben:

## Definition

Sei  $P = (Q, \Sigma, \Gamma, \delta, q_0)$  ein beliebiger Kellerautomat.

Für beliebige  $q_1, q_2 \in Q, \omega \in \Sigma^*, a \in \Sigma, \gamma, \gamma' \in \Gamma^*, X \in \Gamma$  ist die **Übergangsrelation**  $\vdash \subseteq K_P \times K_P$  definiert wie folgt:

1.  $(q_1, a\omega, X\gamma) \vdash (q_2, \omega, \gamma'\gamma) : \Leftrightarrow (q_2, \gamma') \in \delta(q_1, a, X)$  // Lese a
2.  $(q_1, \omega, X\gamma) \vdash (q_2, \omega, \gamma'\gamma) : \Leftrightarrow (q_2, \gamma') \in \delta(q_1, \varepsilon, X)$  //  $\varepsilon$ -Übergang

## Bemerkungen:

- $\vdash$  ist **keine Funktion, sondern eine Relation**, da es für jede Konfiguration  $k$  mehrere mögliche Folgekonfigurationen geben kann.
- 1. Fall:  $P$  geht von  $q_1$  in  $q_2$  über, **liest das Eingabezeichen a** und ersetzt im Stack das oberste Zeichen  $X$  durch die Zeichenkette  $\gamma'$ .
- 2. Fall:  $P$  geht von  $q_1$  in  $q_2$  über **ohne ein Eingabezeichen zu lesen** und ersetzt im Stack das oberste Zeichen  $X$  durch die Zeichenkette  $\gamma'$ .

# Sprache eines PDA

## Definition

Die von einem PDA  $P = (Q, \Sigma, \Gamma, \delta, q_0)$  akzeptierte Sprache ist

$$\begin{aligned}\mathcal{L}(P) &= \{\omega \in \Sigma^* \mid P \text{ akzeptiert } \omega\} \\ &= \{\omega \in \Sigma^* \mid P \text{ hat mindestens eine Möglichkeit, } \omega \text{ vollständig einzulesen und in einem beliebigem Zustand mit leerem Keller zu stoppen.}\} \\ &= \{\omega \in \Sigma^* \mid (q_0, \omega, \#) \vdash^* (q_i, \varepsilon, \varepsilon)\}. \text{ (für } q_i \in Q \text{ beliebig)}\end{aligned}$$

Rekursive Definition der **erweiterten Übergangsrelation**  $\vdash^*$

$$\vdash^* \subseteq K_P \times K_P \quad \text{mit}$$

$$k \vdash^* k, \text{ für alle } k \in K_P$$

$$\begin{aligned}k_1 \vdash^* k_n, \text{ wenn es } k_1, k_2, k_n \in K_P \text{ gibt mit} \\ k_1 \vdash k_2, k_2 \vdash^* k_n\end{aligned}$$

$\Rightarrow k_1 \vdash^* k_n$ , falls Kette  $k_1 \vdash k_2, k_2 \vdash k_3, \dots, k_{n-1} \vdash k_n$  existiert.

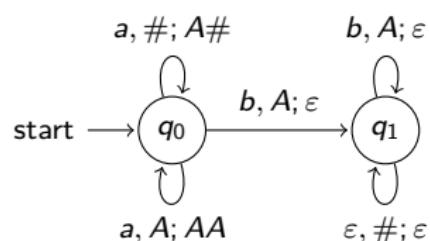
# Beispiel: Sprache des PDA $P_1$

**Frage:** Was ist die Endkonfiguration  $k_x$ , wenn  $P_1$  das Wort  $\omega = aabb$  verarbeitet?

**Formal:** suche  $k_x$  mit  $(q_0, aabb, \#) \vdash^* k_x$ .

$$\begin{aligned} (q_0, aabb, \#) &\vdash (q_0, abb, A\#) \\ &\vdash (q_0, bb, AA\#) \\ &\vdash (q_1, b, A\#) \\ &\vdash (q_1, \varepsilon, \#) \\ &\vdash (q_1, \varepsilon, \varepsilon) \end{aligned}$$

$\Rightarrow (q_0, \omega, \#) \vdash^* (q_1, \varepsilon, \varepsilon) \Rightarrow aabb$  wird akzeptiert.



**Sprache von  $P_1$ :**

- Alle von  $P_1$  akzeptierten Worte  $\omega$ .
- Formal: Alle Worte  $\omega$ , für die gilt  $(q_0, \omega, \#) \vdash^* (q, \varepsilon, \varepsilon)$ , mit  $q \in Q$ .
- $\mathcal{L}(P_1) = \{a^n b^n \mid n \in \mathbb{N}\} = L_{C_2}$

**Alternative Notation:** Nutzung von Tabellen

# Beispiel: Automatenkonstruktion I

**Erinnerung:** Die Palindromsprache ist die Menge aller Worte, die von vorne und hinten gelesen gleich sind. Vereinfachend betrachten wir  $L_{Pab} = \{\varepsilon\} \cup \{\sigma_1 \dots \sigma_n \sigma_n \dots \sigma_1 \mid n \in \mathbb{N}, \sigma_i \in \{a, b\}\}$ .

**Gesucht:** PDA  $P_P$  mit  $\mathcal{L}(P_P) = L_{Pab}$ .

**Lösung:**  $P_P = (Q, \Sigma, \Gamma, \delta, q_0) = (\{q_0, q_1\}, \{a, b\}, \{A, B, \#\}, \delta, q_0)$  mit

1.  $\delta(q_0, a, \gamma) := \{(q_0, A\gamma)\}$
2.  $\delta(q_0, b, \gamma) := \{(q_0, B\gamma)\}$
3.  $\delta(q_0, \varepsilon, \gamma) := \{(q_1, \gamma)\}$
4.  $\delta(q_1, a, A) := \{(q_1, \varepsilon)\}$
5.  $\delta(q_1, b, B) := \{(q_1, \varepsilon)\}$
6.  $\delta(q_1, \varepsilon, \#) := \{(q_1, \varepsilon)\}$

wobei  $\gamma \in \{A, B, \#\}$  ein beliebiges Kellerzeichen ist.

- Regeln 1&2 speichern „vorwärts“ gelesene Buchstaben im Stack.
- Regeln 4&5 prüfen ob „rückwärts“ gelesene Buchstaben auch vorwärts da waren.
- Regel 3 kehrt die Leserichtung um.
- Regel 6 sorgt für einen leeren Keller, falls das gelesene Wort vorwärts und rückwärts vorkam

# Beispiel: Automatenkonstruktion II

Bemerkung:  $P_P$  „errät“ die Mitte des Wortes (Übergang nach  $q_1$  immer möglich). Klappt, da nur ein Bearbeitungspfad erfolgreich sein muss.

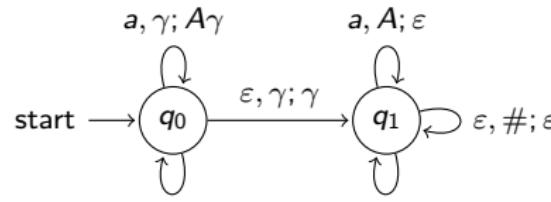
Beispiel:  $\omega = bbaabb$

Ein erfolgreicher Berechnungspfad:

Schritt	Zustand	$\omega$	Keller
0	$q_0$	$bbaabb$	#
1	$q_0$	$baabb$	$B\#$
2	$q_0$	$aabb$	$BB\#$
3	$q_0$	$abb$	$ABB\#$
4	$q_1$	$abb$	$ABB\#$
5	$q_1$	$bb$	$BB\#$
6	$q_1$	$b$	$B\#$
7	$q_1$	$\epsilon$	#
8	$q_1$	$\epsilon$	$\epsilon$

Erweitertes Übergangsdiagramm:

mit  $\gamma \in \Gamma$  beliebig um Pfeile zu sparen



Keller: #

Eingabe: bbaabb ✓

Bemerkung: Zeige Berechnung auf dem erfolgreichen Pfad.

# Kellerautomaten und kontextfreie Sprachen

## Erinnerung:

- Für jede reguläre Sprache  $L \in \mathcal{L}_3$  existiert ein NEA, der  $L$  akzeptiert.
- $\mathcal{L}_2$ , die Sprache der kontextfreien Sprachen ist eine Oberklasse von  $\mathcal{L}_3$ , also ausdrucksstärker, mächtiger
- PDAs sind eine Erweiterung von NEAs, aber viel mächtiger

**Folgerung:** Die von PDAs erkannte Sprachklasse ist eine Obermenge der von DEAs/NEAs erkannten.

## Satz

*Die folgenden Aussagen für eine kontextfreie Sprache  $L$  sind äquivalent:*

- Es gibt eine kontextfreie Grammatik  $G$  mit  $L = \mathcal{L}(G)$ .
- Es gibt einen PDA  $P$  mit  $L = \mathcal{L}(P)$ .

# Kontextfreie Grammatiken und Kellerautomaten I

Folgerung aus dem Äquivalenzsatz: Jede kontextfreie Grammatik  $G = (N, \Sigma, P, S)$  lässt sich in einen Kellerautomaten  $P_G = (Q, \Sigma, \Gamma, \delta, q_0)$  überführen mit  $\mathcal{L}(P_G) = \mathcal{L}(G)$ .

Konstruktion des PDAs  $P_G$  für  $G$ :

- $P_G$  braucht nur einen Zustand,  $Q = \{q_0\}$ .
- Auf dem Keller wird die Ableitung des Wortes aus den Regeln nachvollzogen. Daher:  $\Gamma = \Sigma \cup N$
- Für jedes  $A \in N$  werden die Regeln  $A \rightarrow \omega_1 \mid \dots \mid \omega_n$  in die Zustandsübergangsfunktion übersetzt:

$$\delta(q_0, \varepsilon, A) := \{(q_0, \omega_1), \dots, (q_0, \omega_n)\}$$

- Akzeptanz des Eingabewortes folgt durch Kellerabbau:

$$\delta(q_0, \sigma, \sigma) := \{(q_0, \varepsilon)\}$$

- Start der Wörterkennung durch Ablage des Startsymbols im Keller:

# Ein Kellerautomat für die Dyck-Sprache $D_2$

**Erinnerung:**  $D_2 = \text{Menge aller korrekt geklammerten Ausdrücke mit } () \text{ und } []$ .  $\mathcal{L}(G_2) = D_2$  mit Regelmenge  $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

**Konstruktion von  $P_{G_2}$ :**

- $P_{G_2} = (\{q_0\}, \Sigma, \Gamma, \delta, q_0)$
- Ableitung von  $\delta$  aus den Regeln von  $G$ :
  1.  $\delta(q_0, \varepsilon, S) := \{(q_0, \varepsilon)\}$
  2.  $\delta(q_0, \varepsilon, S) := \{(q_0, SS)\}$
  3.  $\delta(q_0, \varepsilon, S) := \{(q_0, [S])\}$
  4.  $\delta(q_0, \varepsilon, S) := \{(q_0, (S))\}$
  5.  $\delta(q_0, \sigma, \sigma) := \{(q_0, \varepsilon)\}, \text{ für } \sigma \in \Sigma$
  6.  $\delta(q_0, \varepsilon, \perp) := \{(q_0, S)\}$

**Bemerkungen:**

- Regeln 6 startet die Ableitung
- Regeln 1-4 modellieren die Regeln von  $G$
- Regel 5 akzeptiert erzeugte Terminalzeichen

Beispiel:  $\omega = ()[0]()$

Zustand	$\omega$	Keller
(6) $s_0$	$()[0]()$	$\perp$
(6) $s_0$	$()[0]()$	$S$
(2) $s_0$	$()[0]()$	$SS$
(4) $s_0$	$()[0]()$	$(S)S$
(5) $s_0$	$)[]()$	$S)S$
(1) $s_0$	$)[]()$	$)S$
(5) $s_0$	$[]()$	$S$
(2) $s_0$	$[]()$	$SS$
(3) $s_0$	$[]()$	$[S]S$
(5) $s_0$	$]()$	$S]S$
(4) $s_0$	$]()$	$(S)]S$
(5) $s_0$	$]()$	$S)]S$
(1) $s_0$	$]()$	$]S$
(5) $s_0$	$]()$	$]S$
(5) $s_0$	$($	$S$
(4) $s_0$	$($	$(S)$
(5) $s_0$	$)$	$S)$
(1) $s_0$	$)$	$)$
(5) $s_0$	$\varepsilon$	$\varepsilon$

✓ akzeptiert

Bild: Verarbeitung von  $\omega$  durch  $P_{G_2}$   
[Hoffmann, 2011]

# Kontextfreie Grammatiken und Kellerautomaten II

## Bemerkungen:

- Jeder Kellerautomat lässt sich auf einen äquivalenten Automaten mit **einem einzigen Zustand** reduzieren.
- Trotzdem ist die Verwendung mehrerer Zustände nicht sinnlos, viele Sprachen lassen sich dadurch kompakter und übersichtlicher formulieren.
- Insbesondere lässt sich **jede kontextfreie Grammatik durch einen PDA mit nur einem Zustand abbilden**, der die einzelnen Ableitungsschritte in seinem Kellerspeicher abbildet.
- Die Ausdrucksstärke von Kellerautomaten beruht also ausschließlich auf ihrem **(unendlich großen) Kellerspeicher** im Vergleich zu den endlichen Zuständen der DEAs und NEAs.
- **Allerdings** ist der Nichtdeterminismus für die Mächtigkeit von PDAs ebenso wichtig! Im Unterschied zu endlichen Automaten erkennen **deterministische Kellerautomaten** weniger Sprachen als **(nichtdeterministische) Kellerautomaten**.

# Deterministische Kellerautomaten

## Definition

Ein Kellerautomat  $P = (Q, \Sigma, \Gamma, \delta, q_0)$  heißt **deterministisch** (abgekürzt DPDA), wenn für alle Zustände  $q \in Q$ , Eingabezeichen  $\sigma \in \Sigma$  und Kellersymbole  $\gamma \in \Gamma$  gilt:

$$|\delta(q, \sigma, \gamma) \cup \delta(q, \varepsilon, \gamma)| \leq 1$$

## Bemerkungen:

- Ein PDA  $P$  ist also genau dann ein DPDA, wenn es in jedem Zustand für jedes gelesene Zeichen maximal einen nächsten Übergang gibt.
- In einem DPDA existieren  $\varepsilon$ -Übergänge nur dann, wenn es keinen Übergang für ein mögliches Eingabezeichen gibt.
- Genauso wie bei PDAs wird ein Eingabewort genau dann akzeptiert, wenn nach der Verarbeitung des letzten Zeichens ein leerer Keller zurückbleibt.

# Beispiel: ein DPDA für Palindromsprachen I

**Erinnerung:** Die Palindromsprache

$L_{Pab} = \{\varepsilon\} \cup \{\sigma_1 \dots \sigma_n \sigma_n \dots \sigma_1 \mid n \in \mathbb{N}, \sigma_i \in \{a, b\}\}$  ist die Menge aller Worte, die von vorne und hinten gelesen gleich sind.

- Wir hatten konstruiert: PDA  $P_P$  mit  $\mathcal{L}(P_P) = L_{Pab}$ .
- Kernidee von  $P_P$ : Um zu prüfen, ob Palindrombedingung erfüllt ist, errate die Mitte des Wortes **nichtdeterministisch**.

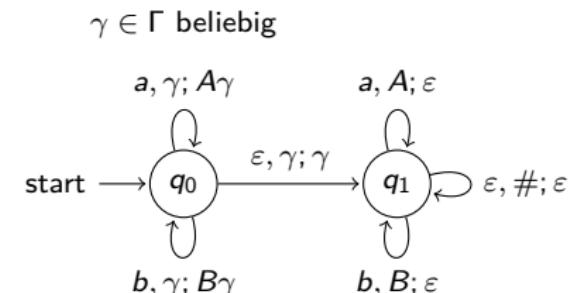


Bild: Erweitertes Übergangsdiagramm von  $P_P$

**Folgerung:** Ein DPDA kann  $L_{Pab}$  nicht erkennen, da er die Mitte des Wortes nicht kennt.

**Lösung:** Markiere Mitte des Wortes mit einem Sonderzeichen. Solche „Fast-Palindromsprachen“ können durch einen DPDA erkannt werden.

# Beispiel: ein DPDA für Palindromsprachen II

**Beispiel:**  $L_{P_{ab-}} := \{-\} \cup \{\sigma_1 \dots \sigma_n - \sigma_n \dots \sigma_1 \mid n \in \mathbb{N}, \sigma_i \in \{a, b\}\}$  ist eine Fast-Palindromsprache, die vom DPDA  $P_{P-}$  erkannt werden kann.

## Verwandlung des PDAs $P_P$ zum DPDA $P_{P-}$

$$P_P = (\{q_0, q_1\}, \{a, b\}, \{A, B\}, \delta, q_0)$$

1.  $\delta(q_0, a, \gamma) := \{(q_0, A\gamma)\}$
2.  $\delta(q_0, b, \gamma) := \{(q_0, B\gamma)\}$
3.  $\delta(q_0, \varepsilon, \gamma) := \{(q_1, \gamma)\}$
4.  $\delta(q_1, a, A) := \{(q_1, \varepsilon)\}$
5.  $\delta(q_1, b, B) := \{(q_1, \varepsilon)\}$
6.  $\delta(q_1, \varepsilon, \#) := \{(q_1, \varepsilon)\}$

$$P_{P-} = (\{q_0, q_1\}, \{a, b, -\}, \{A, B\}, \delta, q_0)$$

1.  $\delta(q_0, a, \gamma) := \{(q_0, A\gamma)\}$
2.  $\delta(q_0, b, \gamma) := \{(q_0, B\gamma)\}$
3.  $\delta(q_0, -, \gamma) := \{(q_1, \gamma)\}$
4.  $\delta(q_1, a, A) := \{(q_1, \varepsilon)\}$
5.  $\delta(q_1, b, B) := \{(q_1, \varepsilon)\}$
6.  $\delta(q_1, \varepsilon, \#) := \{(q_1, \varepsilon)\}$

**Vorgehensweise:** Ersetze nichtdeterministisches Erraten der Mitte durch deterministischen Übergang, falls Markierungszeichen - gefunden wird.

# Beispiel: ein DPDA für Palindromsprachen II

$\gamma \in \Gamma$  beliebig

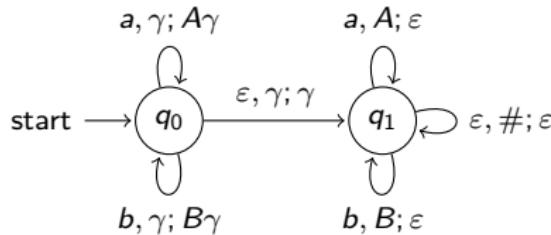


Bild: Erweitertes Übergangsdiagramm von  $P_p$

$\gamma \in \Gamma$  beliebig

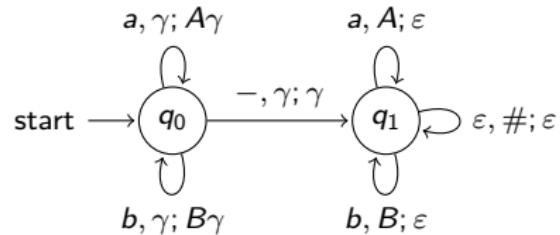


Bild: Erweitertes Übergangsdiagramm von  $P_{p-}$

**Folgerung:** Im Unterschied zu NEAs und DEAs sind PDAs und DPDAs **nicht äquivalent**.

## Satz

*Die Sprachklasse der von deterministischen Kellerautomaten akzeptierten Sprachen ist eine echte Teilmenge der von nichtdeterministischen Kellerautomaten akzeptierten Sprachen.*

# Verwendete oder empfohlene Literatur I

[Hedtstück, 2012] Hedtstück, U. (2012).

*Einführung in die theoretische Informatik: formale Sprachen und Automatentheorie.*

Oldenbourg Verlag.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2011] Hoffmann, D. W. (2011).

*Theoretische Informatik.*

Carl Hanser Verlag GmbH & Co. KG, 2. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2015] Hoffmann, D. W. (2015).

*Theoretische Informatik.*

Carl Hanser Verlag GmbH & Co. KG, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

## Verwendete oder empfohlene Literatur II

[Hopcroft et al., 2011] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2011).

*Einführung in die Automatentheorie, formale Sprachen und Berechenbarkeit (bzw. Komplexitätstheorie); engl.: Introduction to automata theory, languages and computation.*

Pearson, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Wagenknecht and Hielscher, 2014] Wagenknecht, C. and Hielscher, M. (2014).

*Formale Sprachen, abstrakte Automaten und Compiler.*

Springer Vieweg, 2. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.