

Inhaltsverzeichnis

1. Reguläre Ausdrücke, Typ-3-Grammatiken, Endliche Automaten	2
1.1. Typ-3-Grammatiken	2
1.1.1. Äquivalenz rechtslinearer und linkslinearer Grammatiken	2
1.1.2. Eliminierung von ϵ -Regeln.....	3
1.2. Reguläre Ausdrücke.....	3
1.2.1. Pumping-Lemma für reguläre Sprachen.....	3
1.3. Endliche Automaten	4
1.3.1. Nicht totaler deterministischer endlicher Automat.....	4
1.3.2. Totaler (vollständiger) deterministischer endlicher Automat.....	4
1.3.3. Nichtdeterministischer endlicher Automat.....	5
1.3.4. Endlicher Automat mit ϵ -Übergänge	6
1.3.5. Komplementärer Endlicher Automaten	7
1.3.6. Minimierung Endlicher Automaten.....	8
2. Kontextfreie Grammatiken, Kellerautomaten.....	9
2.1. Typ-2-Grammatiken (Kontextfrei)	9
2.1.1. Eliminierung von ϵ -Regeln.....	9
2.1.2. Chomsky-Normalform	9
2.1.3. Greibach-Normalform.....	10
2.1.4. Pumping-Lemma für kontextfreie Sprachen	10
2.2. Kellerautomaten	10
2.2.1. Konstruktion eines Kellerautomaten aus einer Grammatik	11
3. Kontextsensitive Grammatiken, Turingautomaten.....	13
3.1. Typ-1-Grammatiken (Kontextsensitiv)	13
3.2. Typ-0-Grammatiken (rekursiv-aufzählbare Sprachen)	13
3.3. Turingautomaten	14

1. Reguläre Ausdrücke, Typ-3-Grammatiken, Endliche Automaten

1.1. Typ-3-Grammatiken

Definition: $G = (\Sigma, N, P, S)$
 Σ ist das Terminalalphabet
 N ist das disjunktive Nonterminalalphabet
 P ist die Produktionsmenge
 S ist das Startsymbol ($S \in N$)
 $P \subseteq N \times (\Sigma \cup N \cup \{\epsilon\})$

Erlaubt sind als Regeln der Art $A \rightarrow \epsilon \mid a \mid Ba$ ($a \in \Sigma, A \in N, B \in N$)

$A \rightarrow Aa$ ist eine linkslineare Grammatik

$A \rightarrow aA$ ist eine rechtslineare Grammatik

1.1.1 Äquivalenz rechtslinearer und linkslinearer Grammatiken

Verfahren mit dem eine rechtslineare in eine linkslineare Grammatik umgewandelt werden kann:

1. Ende der Ableitung muss $T \rightarrow \epsilon$ sein: T wird Startsymbol, $T \rightarrow \epsilon$ fällt weg.
2. Regeln der Art $A \rightarrow aB$ werden zu $B \rightarrow Aa$.
3. Regel $S \rightarrow \epsilon$: hinzufügen.
4. Wenn $S \rightarrow \epsilon$: existierte, dann $T \rightarrow \epsilon$: hinzufügen.

Beispiel:

$G_1 = (\{0, 1\}, \{S, A, B, T\}, P_1, S)$
 $P_1 = \{ S \rightarrow 0S \mid 1S \mid 0A,$
 $A \rightarrow 0B \mid 1B,$
 $B \rightarrow 0T \mid 1T,$
 $T \rightarrow \epsilon \}$

Ableitung von 011010:

$S \Rightarrow 0S \Rightarrow 01S \Rightarrow 011S \Rightarrow 0110A \Rightarrow 01101B \Rightarrow 011010T \Rightarrow 011010$

$G_2 = (\{0, 1\}, \{S, A, B, T\}, P_2, T)$

$P_2 = \{ S \rightarrow S0 \mid S1,$
 $A \rightarrow S0,$
 $B \rightarrow A0 \mid A1,$
 $T \rightarrow B0 \mid B1,$
 $S \rightarrow \epsilon \}$

T und S wieder vertauschen und A und B ebenfalls (nur der Übersichtlichkeit wegen):

$G_3 = (\{0, 1\}, \{S, A, B, T\}, P_3, S)$

$P_3 = \{ S \rightarrow A0 \mid A1,$
 $A \rightarrow B0 \mid B1,$
 $B \rightarrow T0,$
 $T \rightarrow T0 \mid T1 \mid \epsilon \}$

Ableitung von 011010:

$S \Rightarrow A0 \Rightarrow B10 \Rightarrow T010 \Rightarrow T1010 \Rightarrow T11010 \Rightarrow T011010 \Rightarrow 011010$

1.1.2 Eliminierung von ε -Regeln

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

$$P = \{ \begin{array}{l} S \rightarrow aA \mid bS, \\ A \rightarrow aB \mid bC, \\ B \rightarrow aB \mid bC, \\ C \rightarrow aA \mid bS, \\ B \rightarrow \varepsilon, \\ C \rightarrow \varepsilon, \\ \end{array} \}$$

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

$$P = \{ \begin{array}{l} S \rightarrow aA \mid bS, \\ A \rightarrow aB \mid bC \mid a, \\ B \rightarrow aB \mid bC \mid b, \\ C \rightarrow aA \mid bS \end{array} \}$$

1.2. Reguläre Ausdrücke

Als Alternative zur Grammatik können auch die Regulären Ausdrücke verwendet werden.

$a|b$ heisst *Alternative* (a oder b)
 ab heisst *Konkatenation* (Verkettung) (a gefolgt von b)
 a^* heisst *Kleene-Star* (Kleene-Abschluss) (a Stern)
 a kann 0-mal oder beliebig oft vorkommen

Beispiele:

$(a|b)(a|b)$ $\{aa, ab, ba, bb\}$
 a^* $\{a^n \mid n \leq 0\} = \{\varepsilon, a, aa, aaa, \dots\}$

1.2.1. Pumping-Lemma für reguläre Sprachen

Um zu zeigen, dass eine bestimmte Sprache L nicht regulär ist, kann man so vorgehen:

1. Wähle n (höher als Anzahl Nichtterminalsymbole)
2. Wähle ein Wort $z \in L$, so dass $|z| \geq n$
3. Teile z auf in $z = uvw$, $|uv| \leq n$, $|v| \geq 1$
4. Zeige, dass für jedes $uv^i w$ ein i existiert, so dass $uv^i w \notin L$
5. Wenn das klappt, kann L nicht regulär sein.

1.3. Endliche Automaten

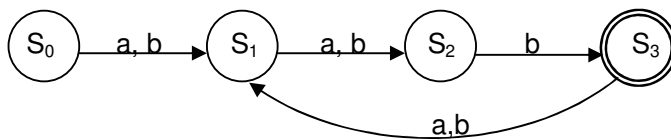
Definition: $A = (\Sigma, S, \delta, s_0, F)$
 Σ ist das Eingabealphabet
 S ist die Zustandsmenge
 δ ist die Überföhrungsfunktion
 s_0 ist der Startzustand
 F ist die Menge der Endzustände

Anmerkungen:

Σ^* alle erzeugbaren W6rter, Σ_+ alle erzeugbaren W6rter ohne das leere Wort ϵ .
 DFA_Σ ist die Klasse der von endlichen Automaten akzeptierten Sprache über Σ .
 REG_Σ ist die Klasse der regulären Sprache über Σ .
 $DFA_\Sigma = REG_\Sigma$

1.3.1. Nicht totaler deterministischer endlicher Automat

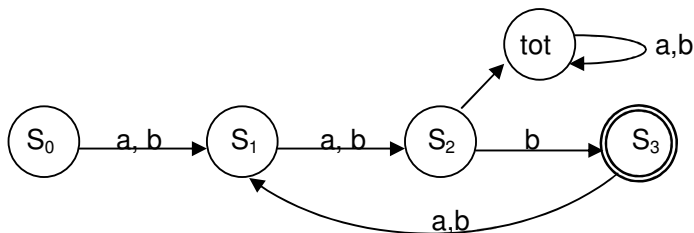
$A = (\{a,b\}, \{S_0, S_1, S_2, S_3\}, \delta, S_0, \{S_3\})$
 $\delta = \{(S_0, a, S_1), (S_0, b, S_1), (S_1, a, S_2), (S_1, b, S_2), (S_2, b, S_3), (S_3, a, S_1), (S_3, b, S_1)\}$



Regulärer Ausdruck: $(a|b)(a|b)b((a|b)(a|b)b)^*$

1.3.2. Totaler (vollständiger) deterministischer endlicher Automat

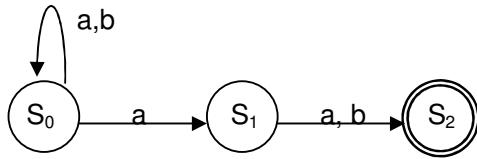
$A = (\{a,b\}, \{S_0, S_1, S_2, S_3, \text{tot}\}, \delta, S_0, \{S_3\})$
 $\delta = \{(S_0, a, S_1), (S_0, b, S_1), (S_1, a, S_2), (S_1, b, S_2), (S_2, a, \text{tot}), (S_2, b, S_3), (S_3, a, S_1), (S_3, b, S_1), (\text{tot}, a, \text{tot}), (\text{tot}, b, \text{tot})\}$



1.3.3. Nichtdeterministischer endlicher Automat

$$A = (\{a,b\}, \{S_0, S_1, S_2\}, \delta, S_0, \{S_2\})$$

$$\delta = \{ (S_0, a, S_0), (S_0, a, S_1), (S_0, b, S_0), (S_1, a, S_2), (S_1, b, S_2) \}$$



Entsprechende Grammatik:

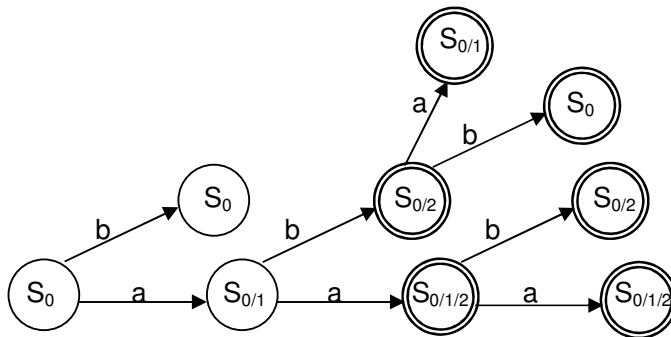
$$G = (\{a, b\}, \{S, A\}, P, S)$$

$$P = \{ S \rightarrow aS \mid bS \mid aA, A \rightarrow a \mid b \}$$

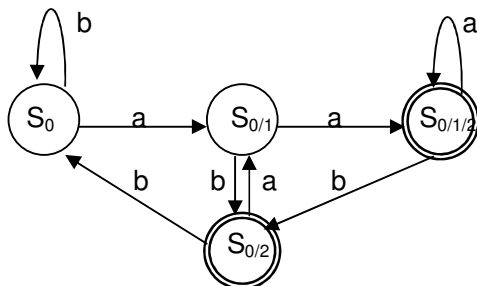
Regulärer Ausdruck: $(a|b)^*a(a|b)$

Umwandeln in einen deterministischen Automaten

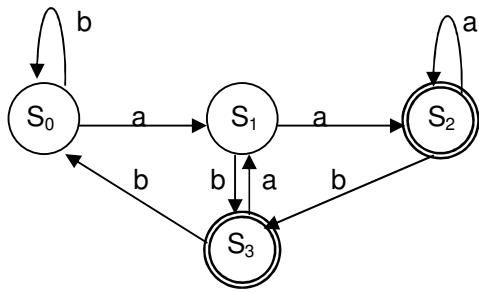
1. Alle Möglichkeiten aufzeichnen. Wenn ein Zustand bereits vorhanden ist, endet der Zweig.



2. Gleiche Zustände zusammenfassen



3. Eventuell Beschriftung Zustände ändern



$A = (\{a,b\}, \{S_0, S_1, S_2, S_3\}, \delta, S_0, \{S_2, S_3\})$

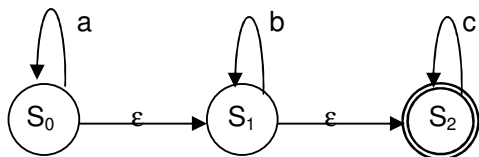
$\delta = \{(S_0, a, S_1), (S_0, b, S_0), (S_1, a, S_2), (S_1, b, S_3), (S_2, a, S_2), (S_2, b, S_3), (S_3, a, S_1), (S_3, b, S_0)\}$

Entsprechende Grammatik (Elimierung von ϵ -Regeln siehe Kap. 1):

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

$P = \{ S \rightarrow aA \mid bS, \\ A \rightarrow aB \mid bC \mid a, \\ B \rightarrow aB \mid bC \mid b, \\ C \rightarrow aA \mid bS \}$

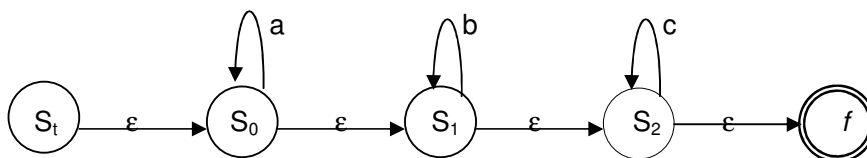
1.3.4. Endlicher Automat mit ϵ -Übergänge



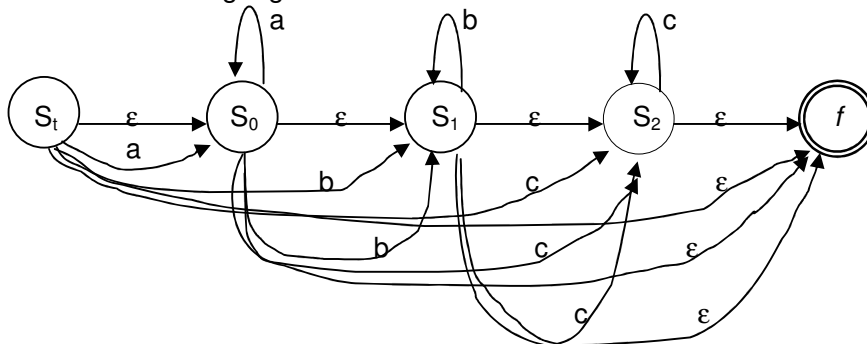
Regulärer Ausdruck: $a^*b^*c^*$

Umwandeln zu einem nichtdeterministischen endlichen Automaten

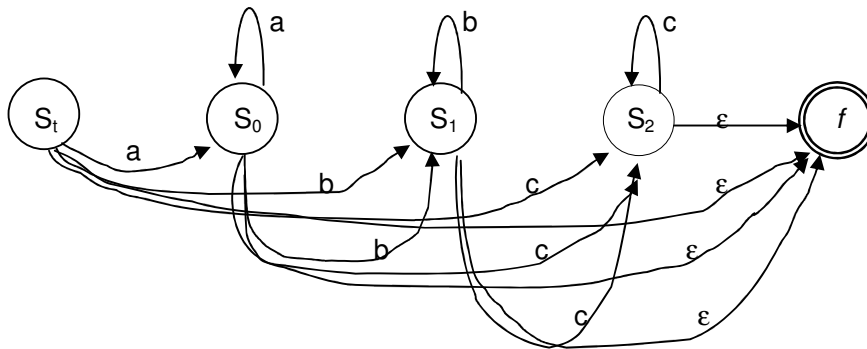
1. Transformation, dass Automat genau einen Start- und Endzustand hat



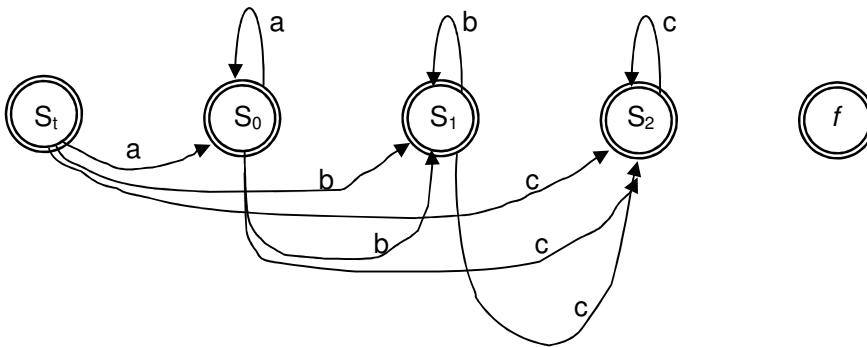
2. Ersatz zu ϵ -Übergängen einzeichnen



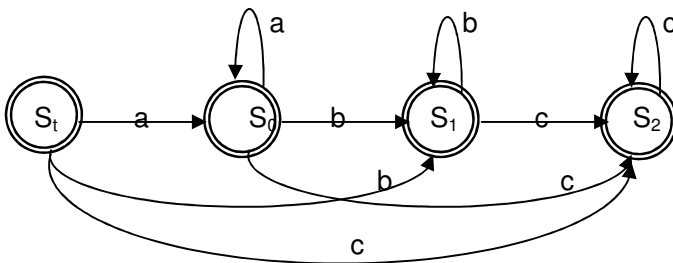
3. Alle ϵ -Übergängen eliminieren die nicht auf den Endzustand zeigen



4. Alle ϵ -Übergängen eliminieren die auf den Endzustand zeigen (Startpunkt wird zu Endzustand)



5. Neu zeichnen

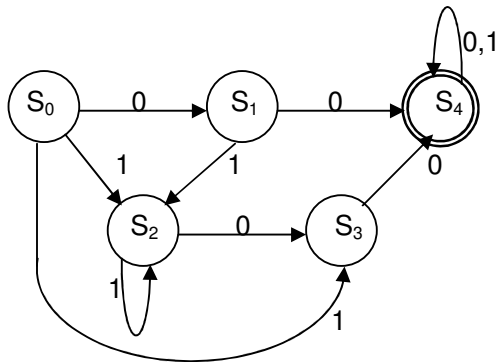


1.3.5. Komplementärer Endlicher Automaten

Um einen komplementären endlichen Automaten zu konstruieren muss wie folgt vorgegangen werden:

1. Automat muss deterministisch gemacht werden.
2. Automat muss vollständig (total) gemacht werden.
3. Endzustände und Nicht-Endzustände tauschen.

1.3.6. Minimierung Endlicher Automaten



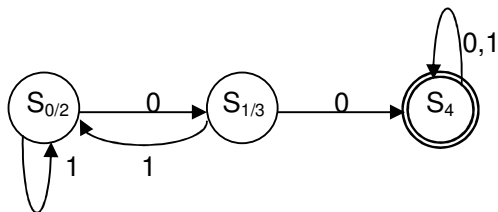
Zusammenfassen aller Endzustände und aller Nicht-Endzustände
 Neue Zustände in Tabelle erfassen:

	S _{0/1/2/3}				S ₄
	S ₀	S ₁	S ₂	S ₃	S ₄
0	S _{0/1/2/3}	S ₄	S _{0/1/2/3}	S ₄	S ₄
1	S _{0/1/2/3}	S _{0/1/2/3}	S _{0/1/2/3}	S _{0/1/2/3}	S ₄

Da der Zustand S_{0/1/2/3} bei den Unterzuständen ungleiches Verhalten aufweist nochmals teilen:

	S _{0/2}		S _{1/3}		S ₄
	S ₀	S ₂	S ₁	S ₃	S ₄
0	S _{1/3}	S _{1/3}	S ₄	S ₄	S ₄
1	S _{0/2}	S _{0/2}	S _{0/2}	S _{0/2}	S ₄

Da jetzt alle Unterzustände gleiches Verhalten aufweisen, kann der minimale Automat konstruiert werden.



Regulärer Ausdruck: $(1^*(01))^*00(0|1)$

2. Kontextfreie Grammatiken, Kellerautomaten

2.1. Typ-2-Grammatiken (Kontextfrei)

Definition: $G = (\Sigma, N, P, S)$
 Σ ist das Terminalalphabet
 N ist das disjunktive Nonterminalalphabet
 P ist die Produktionsmenge
 S ist das Startsymbol ($S \in N$)
 $P \subseteq N \times (\Sigma \cup N)^*$

Erlaubt sind also auf der linken Seite ein Nonterminal und auf der rechten Seite mehrere Nonterminal und Terminal in beliebiger Reihenfolge.

$A \rightarrow \alpha, \alpha \in (\Sigma \cup N)^*$

2.1.1. Eliminierung von ε -Regeln

$G = (\{a, b\}, \{S, A, B, C\}, P, S)$
 $P = \{$
 $\quad S \rightarrow aAa \mid \varepsilon,$
 $\quad A \rightarrow bBb \mid \varepsilon,$
 $\quad B \rightarrow aCa,$
 $\quad C \rightarrow aA \mid bS,$
 $\quad B \rightarrow \varepsilon,$
 $\quad C \rightarrow \varepsilon,$
 $\quad \}$

Verfahren siehe Lehrbrief 4 Seite 6

1. $N_1 = \{S, A, B, C\}$
2. $N_2 = \{S, A, B, C\}$
3. $A \rightarrow bb, B \rightarrow aa$

$G = (\{a, b\}, \{S, A, B, C\}, P, S)$
 $P = \{$
 $\quad S \rightarrow aAa \mid bS,$
 $\quad A \rightarrow bBb \mid bb,$
 $\quad B \rightarrow aCa \mid aa\}$

Erzeugt die gleiche Sprache ohne das leere Wort

2.1.2. Chomsky-Normalform

Kontextfreie Regeln in Chomsky-Normalform haben folgende Gestalt:

$A \rightarrow BC$ oder $A \rightarrow a$

Umwandeln in die Chomsky-Normalform:

$G = (\{a, (,), +, -, *, / \}, \{E, O\}, P, E)$
 $P = \{$
 $\quad E \rightarrow a \mid E O E \mid (E),$
 $\quad O \rightarrow + \mid - \mid * \mid / \}$

1. Alle Terminale in Nichtterminalen wandeln:
 $E \rightarrow KEL, K \rightarrow (, L \rightarrow)$
2. Regeln mit mehr als zwei Nichtterminalen auflösen

$$E \rightarrow KI, I \rightarrow EL$$

$$E \rightarrow EJ, J \rightarrow OE$$

$$G = (\{a, (,), +, -, *, /, \}, \{E, O, K, L, I, J\}, P, E)$$

$$P = \{ \begin{array}{l} E \rightarrow a \mid KI \mid KJ, \\ O \rightarrow + \mid - \mid * \mid /, \\ E \rightarrow KI, I \rightarrow EL, \\ E \rightarrow EJ, J \rightarrow OE, \\ K \rightarrow (, L \rightarrow) \end{array} \}$$

2.1.3. Greibach-Normalform

Kontextfreie Regeln in Greibach-Normalform haben folgende Gestalt:

$$A \rightarrow a \mid aB \mid aBC \mid aBCD \dots \quad \text{also } A \rightarrow aB_1 \dots B_m, m \geq 0$$

2.1.4. Pumping-Lemma für kontextfreie Sprachen

Um zu zeigen, dass eine bestimmte Sprache L nicht kontextfrei ist, kann man so vorgehen:

1. Wähle n (2^m , m ist die Anzahl Nichtterminalsymbole)
2. Wähle ein Wort $z \in L$, so dass $|z| \geq n$
3. Teile z auf in $z = uvwxy$, $|ux| \geq 1$, $|vwx| \leq n$
4. Zeige, dass für jedes $uv^iwx^i y \notin L$
5. Wenn das klappt, kann L nicht kontextfrei sein.

2.2. Kellerautomaten

Definition: $K = (\Sigma, S, \Gamma, \delta, s_0, \perp, F)$
 Σ ist das Eingabealphabet
 S ist die Zustandsmenge
 Γ ist das Kelleralphabet
 δ ist die Überföhrungsfunktion
 s_0 ist der Startzustand
 \perp ist das Keller-Bottomsymbol
 F ist die Menge der Endzustände

Anmerkungen:

- PDA_Σ ist die Klasse der von nicht deterministischen Kellerautomaten akzeptierten Sprachen über Σ .
- $DPDA_\Sigma$ ist die Klasse der von deterministischen Kellerautomaten akzeptierten Sprachen über Σ .
- Anders als bei endlichen Automaten sind im allgemeinen nichtdeterministische Kellerautomaten nicht in deterministische äquivalent transformierbar.
 Es gilt $DPDA_\Sigma \subset PDA_\Sigma$

Beispiel:

Kellerautomat der die Sprache $L = \{0^n 1^n \mid n \geq 0\}$ erkennt

$K = (\{a, b\}, \{s_0, s_1, s_i\}, \{a, \perp\}, \delta, s_0, \perp, \{s_i\})$
 $\delta = \{$

$(s_0, \varepsilon, \perp, s_i, \varepsilon),$	// akzeptiert leeres Wort
$(s_0, a, \perp, s_0, a\vee),$	// Erstes a in Keller schreiben
$(s_0, a, a, s_0, aa),$	// Weiterer a's in Keller schreiben
$(s_0, b, a, s_1, \varepsilon),$	// Erstes a aus Keller löschen
$(s_1, b, a, s_1, \varepsilon),$	// Weiterer a's aus Keller löschen
$(s_1, \varepsilon, \perp, s_i, \varepsilon)\}$	// Keller leer und Wort abgearbeitet

2.2.1. Konstruktion eines Kellerautomaten aus einer Grammatik

1. Das Keller-Bottomsymbol ist das Startsymbol der Grammatik
2. Ist Eingabesymbol gleich dem Keller-Topsymbol wird dieses gelöscht.
3. Ist das Keller-Topsymbol ein Nichtterminal, wird es durch die rechte Seite der Grammatik ersetzt.

Anmerkung:

Der Kellerautomat besitzt nur einen Zustand und akzeptiert mit leerem Keller

Beispiel:

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

$P = \{$

$S \rightarrow aA,$
$A \rightarrow bB,$
$B \rightarrow bB,$
$B \rightarrow c \}$

$K = (\{a, b, c\}, \{q\}, \{a, b, c, A, B, S\}, \delta, q, S)$ // Regel 1 (Keller-Bottomsymbol ist S)

$\delta = \{$

$(q, a, a, q, \varepsilon),$	// Regel 2
$(q, b, b, q, \varepsilon),$	// Regel 2
$(q, c, c, q, \varepsilon),$	// Regel 2
$(q, \varepsilon, S, q, aA),$	// Regel 3
$(q, \varepsilon, A, q, bB),$	// Regel 3
$(q, \varepsilon, B, q, bB),$	// Regel 3
$(q, \varepsilon, B, q, c) \}$	// Regel 3

Dieser Kellerautomat ist nicht deterministisch!

Ableitung des Wortes "abbc":

a	b	b	c	(blank)
Keller:				
S	A	B	B	
--	--	--	--	
a	b	b	c	
A	B	B		

Alternative: Deterministischer Kellerautomat

$$K = (\{a, b, c\}, \{s_0, s_f\}, \{a, b, c, \perp\}, \delta, s_0, \perp, \{s_f\})$$

$$\delta = \{ \begin{array}{ll} (s_0, a, \perp, s_0, a\perp), & // \text{ Erster Buchstabe muss ein a sein} \\ (s_0, b, a, s_0, ba), & // \text{ Darauf muss ein b folgen} \\ (s_0, b, b, s_0, b), & // \text{ Beliebig viele b's} \\ (s_0, c, b, s_0, cb), & // \text{ Am Schluss ein c} \\ (s_0, \varepsilon, c, s_f, \varepsilon) \} & // \text{ Jetzt muss das Wort zu Ende sein} \end{array}$$

Dieser Kellerautomat akzeptiert mit Endzustand, der Keller wird nicht geleert!

Ableitung des Wortes "abbc":

a	b	b	c	(blank)	
Zustand:					
s_0	s_0	s_0	s_0	s_0	s_f
Keller:					
\perp	a	b	b	c	b
	\perp	a	a	b	a
		\perp	\perp	a	\perp
				\perp	

Anderer Darstellungsart:

$$\begin{array}{ll} (s_0, abbc, \perp) & \rightarrow (s_0, bbc, a\perp) \\ & \rightarrow (s_0, bc, ba\perp) \\ & \rightarrow (s_0, c, ba\perp) \\ & \rightarrow (s_0, \varepsilon, cba\perp) \\ & \rightarrow (s_f, \varepsilon, ba\perp) \end{array}$$

3. Kontextsensitive Grammatiken, Turingautomaten

3.1. Typ-1-Grammatiken (Kontextsensitiv)

Definition: $G = (\Sigma, N, P, S)$
 Σ ist das Terminalalphabet
 N ist das disjunktive Nonterminalalphabet
 P ist die Produktionsmenge
 S ist das Startsymbol ($S \in N$)
 $P \subseteq (\Sigma \cup N)^+ - \Sigma^* \times (\Sigma \cup N)^*$
 wobei gilt: $\alpha \rightarrow \beta \in P$, dann ist $|\alpha| \leq |\beta|$

Erlaubt sind also auf der linken Seite und auf der rechten Seite mehrere Nonterminal und Terminal in beliebiger Reihenfolge, wobei die linke Seite nicht länger sein darf.

$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, $\alpha_1, \alpha_2, \beta \in (\Sigma \cup N)^*$, $\beta \neq \varepsilon$, $A \in \Sigma$

Beispiel:

$G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$
 $P = \{$
 $S \rightarrow aSBC \mid aBC,$
 $CB \rightarrow BC,$
 $aB \rightarrow ab,$
 $bB \rightarrow bb,$
 $bC \rightarrow bc,$
 $cC \rightarrow cc \}$

Erzeugt wird die Sprache $L = \{a^k b^k c^k \mid k \geq 1\}$

3.2. Typ-0-Grammatiken (rekursiv-aufzählbare Sprachen)

Definition: $G = (\Sigma, N, P, S)$
 Σ ist das Terminalalphabet
 N ist das disjunktive Nonterminalalphabet
 P ist die Produktionsmenge
 S ist das Startsymbol ($S \in N$)
 $P \subseteq (\Sigma \cup N)^+ - \Sigma^* \times (\Sigma \cup N)^*$

Der Unterschied zur Typ-1-Grammatik ist dass die linke Seite länger sein darf als die rechte. Die Länge des abgeleiteten Wortes in einem Ableitungsschritt kleiner sein als die des abzuleitenden Wortes (Nicht monoton).

$\alpha \rightarrow \beta$, $\alpha, \beta \in (\Sigma \cup N)^*$, $\alpha \neq \varepsilon$

Beispiel:

Die folgende Grammatik erzeugt die Sprache $\{a^i \mid i \text{ ist pos. Zweierpotenz}\}$
 $G = (\{a\}, \{S, A, B, C, D, E\}, P, S)$
 $P = \{$
 $S \rightarrow ACaB,$
 $Ca \rightarrow aaC,$
 $CB \rightarrow DB \mid E,$
 $aD \rightarrow Da,$
 $AD \rightarrow AC,$
 $aE \rightarrow Ea,$
 $AE \rightarrow \varepsilon \}$

3.3. Turingautomaten

Definition: $T = (\Sigma, S, \Gamma, \delta, s_0, \#, F)$
 Σ ist das Eingabealphabet
 S ist die Zustandsmenge
 Γ ist das Arbeitsalphabet (Bandalphabet)
 δ ist die Überföhrungsfunktion
 s_0 ist der Startzustand
 $\#$ ist Blanksymbol ($\# \in \Gamma - \Sigma$)
 F ist die Menge der Endzustände

Anmerkungen:

- TA_Σ ist die Klasse der vom Turingautomaten akzeptierten Sprachen über Σ .
- DTA_Σ ist die Klasse der vom deterministischen Turingautomaten akzeptierten Sprachen über Σ .
- Deterministische Turingautomaten und Nichtdeterministische Turingautomaten sind äquivalent. Es gilt $TA_\Sigma = DTA_\Sigma$
- Turingautomaten akzeptieren Typ-0-Grammatiken
- Linear beschränkte Turingautomaten akzeptieren Typ-1-Grammatiken

Beispiel:

Turingautomat der die Nachfolgezahl einer binären Zahl berechnet.

$T = (\{0, 1\}, \{s_0, s_1, s_i\}, \{0, 1, \#\}, \delta, s_0, \#, \{s_i\})$
 $\delta = \{$

$(s_0, 0, s_0, 0, r),$	// über alle 0's hinweg
$(s_0, 1, s_0, 1, r),$	// über alle 1's hinweg
$(s_0, \#, s_1, \#, l),$	// auf das erste Bit setzen
$(s_1, 0, s_i, 1, -),$	// 0 durch 1 ersetzen und Ende
$(s_1, \#, s_i, 1, -),$	// # durch 1 ersetzen und Ende
$(s_1, 1, s_i, 0, l)\}$	// 1 durch 0 ersetzen und nächstes Bit prüfen

Beispiel:

$\#1011\# \Rightarrow \#1011\# \Rightarrow \#1010\# \Rightarrow \#1000\# \Rightarrow \#1100\#$
 $\uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow$