1 Grammatiken

Autor: Tilman Blumenbach
Letzte Änderung: 28. Juni 2012 18:15

Ziel von Grammatiken Wollen die Struktur von Sprachen modellieren und charakterisieren.

Beispiel Ein Satz der deutschen Sprache besitzt ein Subjekt, ein Prädikat und ein Objekt (SPO).

$$
\begin{aligned}
S &\to A_1 N \\
P &\to V\,Ad \\
O &\to A_2 N
\end{aligned}
$$

$$
\begin{aligned}
A_1 &\to \text{DER} \mid \text{DIE} \mid \text{DAS} \\
N &\to \text{HAUS} \mid \text{HUND} \mid \text{BLUME} \\
V &\to \text{GEHT} \mid \text{SIEHT} \\
A_2 &\to \text{DEN} \mid \text{DIE} \mid \text{DAS} \\
Ad &\to \text{SCHNELL} \mid \varepsilon
\end{aligned}
$$

Mit Hilfe der Grammatik können wir Sätze erzeugen.

$$
\begin{aligned}
\text{Satz} &\to SPO \to A_1 NPO \\
&\to \text{DER } NPO \\
&\to \text{DER } NV\,Ad\ O \\
&\to \text{DER HUND } V\,Ad\ O \\
&\to \text{DER HUND } V\,Ad\ A_2 N \\
&\to \text{DER HUND SIEHT } Ad\ A_2 N \\
&\to \dots \\
&\to \text{DER HUND SIEHT DIE BLUME}
\end{aligned}
$$

DER, HUND etc. sind sogenannte Terminalsymbole (nicht weiter auflösbar).

Formale Definition Eine Grammatik G besteht aus:

- V: Variablen
- Σ: Alphabet/Terminalsymbole
- $S \in V$: Startvariable (oben „Satz“)
- P: Ableitungsregeln oder Produktionen. Die Elemente von P haben die Form $\alpha \to \beta$, wobei $\alpha \in (V \cup \Sigma)^+$ (mit mindestens einer Variable) und $\beta \in (V \cup \Sigma)^*$.

Es gilt außerdem:

$$V \cap \Sigma = \varnothing$$

Erzeugen von Worten (Ableitung) Grammatiken erzeugen Wörter.

- Beginnen mit dem Startsymbol S.

- Solange die aktuelle Zeichenkette x die linke Seite α einer Produktion enthält, ersetze α durch β. Sind dabei mehrere Ersetzungen möglich, wählen wir uns eine aus.
- Wiederhole, solange möglich.
- Wenn nur noch Terminalsymbole dastehen, wurde der String durch G erzeugt.

Die durch eine Grammatik G beschriebene Sprache ist:

$$\mathcal{L}(G) = \{w \in \Sigma^* \mid w \text{ wird von } G \text{ erzeugt}\}$$

Wortproblem Gegeben $w \in \Sigma^*$ und Grammatik G: Ist $w \in \mathcal{L}(G)$?

Weitere Beispiele

- Sei $\Sigma = \{(,)\}$
  Gib Grammatik für alle gültigen Klammerausdrücke an.

  $$S \to \varepsilon \mid (S) \mid S\ S$$

  Behauptung: Diese Grammatik erzeugt alle gültigen Klammerausdrücke.

  Beweis

  1. Die Grammatik erzeugt nur gültige Klammerausdrücke, nichts anderes.
     Beweis durch Induktion nach n, der Anzahl der Ableitungsschritte.
     IA
     $n = 1$. Kann nur $S \to \varepsilon$ anwenden; gültig.
     IS
     IV: Kann in höchstens n Schritten nur gültige Klammerausdrücke erzeugen.
     Zu zeigen: Kann in $\leq n+1$ Schritten nur gültige Klammerausdrücke erzeugen.
     Betrachten eine Ableitung mit $n+1$ Schritten. Unterscheide den ersten Schritt.
     - $S \to (S) \xrightarrow{n \text{ Schritte}} (x)$
       x wird aus S in n Schritten abgeleitet, nach IV ist x gültig. Folgerung: (x) ist gültig.
     - $S \to S\ S \xrightarrow{n \text{ Schritte}} x_1\ x_2$
       Also: $S \xrightarrow{\leq n \text{ Schritte}} x_1$ und $S \xrightarrow{\leq n \text{ Schritte}} x_2$
       Nach IV sind x_1 und x_2 gültige Klammerausdrücke, also auch $x_1 x_2$.
  2. Sei α gültiger Klammerausdruck. Dann können wir α erzeugen. Induktion nach $|\alpha|$.
     IA
     $|\alpha| = 0$, d. h. $\alpha = \varepsilon$. $S \to \varepsilon$, d. h. α kann erzeugt werden.
     IS
     IV: Können alle Klammerausdrücke der Länge $\leq n$ erzeugen.
     Zu zeigen: Können alle Klammerausdrücke der Länge $\leq n+1$ erzeugen. Sei α Klammerausdruck mit $|\alpha| = n+1$. Wissen: $\alpha = (\beta)\gamma$, mit β, γ gültige Klammerausdrücke mit $|\beta|, |\gamma| \leq n$.
     Nach IV können β, γ erzeugt werden. Erzeugen α folgendermaßen:

     $$S \to S\ S \to (S)\ S \xrightarrow{\text{IV}} (\beta)S \xrightarrow{\text{IV}} (\beta)\gamma = \alpha$$

- Sei $\Sigma = \{0, 1, \#\}$.
  Gib Grammatik für $L = \{w\#w \mid w \in \{0,1\}^*\}$ an.

$$
\begin{aligned}
S &\to A\ S \mid B\ S \mid \# \\
A\ \# &\to 0\ \#\ 0 \\
B\ \# &\to 1\ \#\ 1 \\
A\ 0 &\to 0\ 0\ \overline{A} \\
A\ 1 &\to 0\ 1\ \overline{A} \\
\overline{A}\ 0 &\to 0\ \overline{A} \\
\overline{A}\ 1 &\to 1\ \overline{A} \\
\overline{A}\ \# &\to \#\ 0 \\
B\ 0 &\to 1\ 0\ \overline{B} \\
B\ 1 &\to 1\ 1\ \overline{B} \\
\overline{B}\ 0 &\to 0\ \overline{B} \\
\overline{B}\ 1 &\to 1\ \overline{B} \\
\overline{B}\ \# &\to \#\ 1
\end{aligned}
$$

- Sei $\Sigma = \{\square, \heartsuit\}$. Wollen: $L = \{w \in \Sigma^* \mid |w| \text{ ist gerade}\}$

$$
\begin{aligned}
S &\to \varepsilon \mid \square\ T \mid \heartsuit\ T \\
T &\to \square\ S \mid \heartsuit\ S
\end{aligned}
$$

Chomsky-Hierarchie Wir klassifizieren Grammatiken dadurch, welche Art von Produktionen erlaubt ist. Die Chomsky-Hierarchie ist echt („proper“), d. h. alle Typen sind unterschiedlich.

Typen

0. Keine Beschränkung (alle Produktionen erlaubt).
1. Kontextsensitiv: Für alle Produktionen $\alpha \to \beta$ gilt: $|\beta| \geq |\alpha|$.
2. Kontextfrei: Alle Produktionen haben die Form $V \to (V \cup \Sigma)^*$, d. h. links steht immer eine Variable.
3. Regulär: Alle Produktionen haben die Form:

$$
\begin{aligned}
V &\to \Sigma \\
V &\to \Sigma\ V
\end{aligned}
$$

Definition Wir nennen eine Sprache L Typ i (mit $0 \leq i \leq 3$) genau dann, wenn $L \setminus \{\varepsilon\}$ durch eine Grammatik vom Typ i erzeugt werden kann.

Fakt 1 Typ 0 $\supseteq$ Typ 1 $\supseteq$ Typ 2 $\supseteq$ Typ 3; alles, was mit Typ i geht, geht auch mit Typ $i-1$.

Satz 1 Die Menge der Typ 0-Sprachen ist genau die Menge der rekursiv aufzählbaren/semientscheidbaren Sprachen (ohne Beweis).

Satz 2 Das Wortproblem für Typ 1-Sprachen ist entscheidbar.

Satz 3 Die Menge der Typ 3-Sprachen ist genau die Menge der regulären Sprachen. Beweis: Übung.

Fakt 2 Die Menge aller gültigen Klammerausdrücke ist nicht regulär (Pumping-Lemma!), aber kontextfrei.

Fakt 3 $L = \{w\#w \mid w \in \{0,1\}^*\}$ ist nicht kontextfrei, aber kontextsensitiv.

Weitere Klassifizierungen

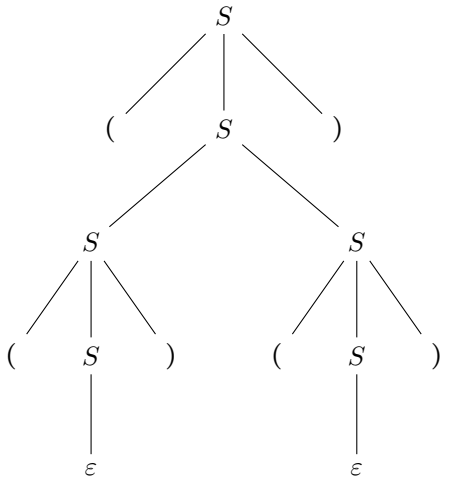
- Typ 0: Halteproblem.
- Typ 1: Siehe Fakt 3, $\{w\#w\}$.
- Typ 2: Klammerausdrücke.
- Typ 3: $\mathcal{L}((01)^*)$
- Keiner (nicht in Chomsky-Hierarchie): Diagonalsprache.

Kontextfreie Sprachen bzw. kontextfreie Grammatiken Kontextfreie Sprachen sind interessant da, sie ausdrucksstark sind und sich effizient analysieren (parsen) lassen.

Syntaxbäume Die Ableitung aus einer kontextfreien Grammatik kann man als Syntaxbaum darstellen. Wurzel ist Startvariable (hier S), innere Knoten sind Variablen, Blätter sind Terminalsymbole (oder ε) und die Kinder eines inneren Knotens entsprechen einer angewandten Produktion.
Beispiel:

$$S \to \varepsilon \mid (S) \mid S\ S$$
$$S \to (S) \to (S\ S) \to ((S)\,S) \to (()\,S) \to \cdots \to (()\,())$$

Das ergibt den folgenden Syntaxbaum:



Zu jeder Ableitung gibt es genau einen Syntaxbaum, aber zu einem Syntaxbaum kann es mehrere Ableitungen geben.

Mehrdeutigkeit von kontextfreien Grammatiken Es kann sein, dass es zu einem Wort verschiedene Syntaxbäume gibt.

Eine Grammatik heißt mehrdeutig, wenn ein $w \in \mathcal{L}(G)$ existiert, so dass w zwei verschiedene Syntaxbäume hat.

Beispiel:

$$\Sigma = \{(,), \cdot, +, 0, 1, \ldots, 9\}$$
$$S \to S + S \mid S \cdot S \mid (S) \mid Z$$
$$Z \to 0 \mid 1 \mid \ldots \mid 9$$

Mit $w = 1 + 4 \cdot 5$.

Diese Grammatik ist mehrdeutig, was schlecht für den Bau von Übersetzern ist. Wollen eindeutige Grammatik, welche die Rechenregeln befolgt. Geht folgendermaßen:

$$S \to S + T \mid T$$
$$T \to T \cdot F \mid F$$
$$F \to (S) \mid Z$$
$$Z \to 0 \mid 1 \mid \ldots \mid 9$$

Fakt Es gibt inhärent mehrdeutige Sprachen, d. h. kontextfreie Sprachen, für die keine eindeutige Grammatik existiert.

Eigenschaften kontextfreier Sprachen

- Sind L_1 und L_2 kontextfrei, so auch $L_1 \cup L_2$, $L_1 \circ L_2$ und L_1^*.
- Gilt nicht für $L_1 \cap L_2$; kontextfreie Sprachen sind unter Schnitt nicht zwingend abgeschlossen (gibt Gegenbeispiel).
- Daraus folgt: Kontextfreie Sprachen sind ebenfalls nicht unter Komplementbildung abgeschlossen.
  Beweis: Wenn $\Sigma^* \setminus L_1$ kontextfrei wäre, dann wäre auch
  $$L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$$
  kontextfrei, was aber nicht der Fall ist.
- Aber immerhin gilt: L_1 kontextfrei, L_2 regulär $\Rightarrow L_1 \cap L_2$ kontextfrei.

Chomsky-Normalform (CNF) Algorithmisch und mathematisch ist es vorteilhaft, kontextfreie Grammatiken auf eine einheitliche Form zu bringen.

Definition Eine Grammatik ist in CNF genau dann, wenn alle Produktionen die Form

$$A \to \sigma, \quad A \in V, \sigma \in \Sigma$$

oder

$$A \to B\ C, \quad A, B, C \in V$$

haben.

Satz Sei G eine kontextfreie Grammatik mit $\varepsilon \notin \mathcal{L}(G)$. Dann existiert eine zu G äquivalente Grammatik in CNF.

Überführung einer kontextfreien Grammatik in CNF Siehe `http://www.inf.fu-berlin.de/lehre/SS12/GTI/cnf.pdf` bzw. chomsky-normalform.pdf in diesem Verzeichnis.

Algorithmus von Cocke, Younger & Kasami (CYK) [1] Anwendung der CNF; ist ein Algorithmus für das Wortproblem.

Gegeben: Grammatik G in CNF und Wort $w \in \Sigma^*$.

Frage: Ist $w \in \mathcal{L}(G)$? Wenn ja, finde Syntaxbaum für w.

Strategie: Dynamisches Programmieren. Schreibe: $w = \sigma_1\sigma_2 \dots \sigma_n$ mit $\sigma \in \Sigma$. Teilprobleme: $V[i,j]$, das sind alle Variablen, aus denen sich der String $\sigma_i\sigma_{i+1} \dots \sigma_j$ (mit $1 \le i \le j \le n$) ableiten lässt.

Rekursion:

$$V[i,i] = \{A \in V \mid A \to \sigma_i \text{ ist Produktion}\}$$
$$V[i,j] = \bigcup_{i \le k \le j} \{A \in V \mid A \to B\ C \text{ ist Produktion} \wedge B \in V[i,k] \wedge C \in V[k+1,j]\}$$

Also: Berechnen zuerst $V[1,1], V[2,2], V[3,3], \dots, V[n,n]$, dann $V[1,2], V[2,3], V[3,4], \dots$, dann $V[1,3], V[2,4], \dots$. Immer so weiter, bis man $V[1,n]$ berechnet.

Es gilt:

$$w \in \mathcal{L}(G) \Leftrightarrow S \in V[1,n]$$

Beispiel

Sei folgende Grammatik gegeben:

$$V = \{S, A, B, C, D, E, F\}$$
$$\Sigma = \{a, b, c\}$$

$$S \to A\ B$$
$$A \to C\ D \mid C\ F$$
$$B \to c \mid E\ B$$
$$C \to a$$
$$D \to b$$
$$E \to c$$
$$F \to A\ D$$

Möchten feststellen, ob Wort $w = aaabbbcc$ durch die Grammatik erzeugt wird. Anwenden des CYK-Algorithmus ergibt folgende Tabelle:

S	—	—	—	—	—	—	—
S	x	—	—	—	—	—	—
A	x	x	—	—	—	—	—
x	F	x	x	—	—	—	—
x	A	x	x	x	—	—	—
x	x	F	x	x	x	—	—
x	x	A	x	x	x	B	—
C	C	C	D	D	D	B, E	B, E
a	a	a	b	b	b	c	c

[1] siehe auch `http://de.wikipedia.org/wiki/Cocke-Younger-Kasami-Algorithmus`

Jede Zelle enthält die Variablen, aus denen das Teilwort abgeleitet werden kann, dessen Anfang und Ende wie folgt bestimmt werden:

- Das gesamte Wort steht in der letzten Zeile.
- Der Anfang des Teilwortes wird durch die Spalte der aktuellen Zelle bezeichnet („zeigt“ auf einen Buchstaben in der letzten Zeile).
- Das Ende des Wortes findet man heraus, indem man von der aktuellen Zelle diagonal nach rechts unten geht, bis man in der vorletzten Zeile landet und von der so erreichten Zelle noch eins nach unten. Trifft nicht auf die vorletzte Zeile zu, denn dort stehen nur Teilworte, die aus einem Buchstaben bestehen.

Die vorletzte Zeile ist leicht auszufüllen, hier müssen nur die Variablen gefunden werden, aus denen sich der jeweilige Buchstabe der letzten Zeile ableiten lässt.

Die restlichen Zellen leiten sich, um es mit Prof. Mulzers Worten zu sagen, wie folgt ab:

„Den linken Finger auf die Zelle unter der aktuellen Zelle, den rechten Finger auf die Zelle in der vorletzten Zeile, die sich über dem letzten Buchstaben des Teilwortes befindet. Dann alle Variablen finden, aus denen sich die beiden durch den linken und rechten Finger (in dieser Reihenfolge) bezeichneten Variablen ableiten lassen. Anschließend den linken Finger um eine Zelle nach unten verschieben, den rechten diagonal nach links oben (um eine Zelle), erneut Zellen vergleichen, Variablen finden, wieder Finger verschieben usw.“

Findet man einmal keine Variablen, schreibt man ein „x“ (o. ä.) in die Zelle. Enthält die Zelle in der obersten Zeile am Ende (auch) die Startvariable, erzeugt die Grammatik das Wort.

Indem man die Schritte, die man gemacht hat, um die oberste Zelle auszufüllen, rückwärts nachvollzieht (quasi von oben nach unten), kann man einen Syntaxbaum für die Ableitung des Wortes konstruieren.

Laufzeit

Der CYK-Algorithmus ist effizient bzw. läuft in Polynomialzeit ($\mathcal{O}(n^3)$).

Aber: Ist in der Praxis zu langsam. Deswegen benutzt man im Übersetzerbau eingeschränkte, spezielle kontextfreie Grammatiken, die sich schneller analysieren lassen ($LL(k)$ und $LR(k)$).

Pumping-Lemma Sei G kontextfrei und in CNF. Sei $w \in \mathcal{L}(G)$ und T Syntaxbaum für w.

Löschen alle Terminalsymbole aus T. Erhalten einen binären Baum (wegen CNF), d. h. Wurzelbaum, in dem jeder innere Knoten genau zwei Kinder hat.

Höhe eines binären Baumes ist Länge eines längsten Pfades von der Wurzel zu einem Blatt. Länge eine Pfades ist Anzahl der Kanten. Binärer Baum mit Höhe h hat höchstens 2^h Blätter. Folgerung: Binärer Baum mit 2^h Blättern hat Höhe $\geq h$.

Nun: Sei G in CNF mit k Variablen. Sei $w \in \mathcal{L}(G)$ mit $|w| \geq 2^k$. Sei T ein Syntaxbaum für w ohne Terminalsymbole. T ist ein binärer Baum mit mindestens 2^k Blättern, hat also Höhe $\geq k$.

Daraus folgt: T hat Pfad P von Wurzel zu einem Blatt mit $\geq k{+}1$ Knoten. Mindestens eine Variable aus G taucht mindestens zweimal in P auf. Nenne eine solche Variable A. Betrachte Teilbäume von A:

- T_1: Unterster Teilbaum mit Wurzel A entlang P.
- T_2: Nächstoberer Teilbaum mit Wurzel A entlang P.

Können T_2 entlang von P beliebig oft wiederholen und erhalten jedes Mal einen gültigen Syntaxbaum für G. Alle diese Syntaxbäume entsprechen einem Wort in $\mathcal{L}(G)$.

Zerlege $w = rstuv$:

- r ist Teil vor T_2.
- s ist Teil nach r, aber vor T_1 und in T_2.
- t ist Teil unter T_1.
- u ist Teil nach T_1, aber in T_2.

- v ist Teil nach T_2.

Dann: $|stu| \leq 2^k$ und $su \neq \varepsilon$ und für alle $i = 0, 1, 2, \ldots$ gilt $rs^i tu^i v \in \mathcal{L}(G)$.

Lemma Sei L kontextfrei. Dann gilt:

$\exists n \in \mathbb{N} : \forall w \in L, |w| \geq n : \exists$ Zerlegung $w = rstuv$ mit $|stu| \leq n, su \neq \varepsilon : \forall i = 0, 1, 2, \ldots$ ist $rs^i tu^i v \in L$

Kontraposition

$\forall n \in \mathbb{N} : \exists w \in L, |w| \geq n : \forall$ Zerlegungen $w = rstuv, |stu| \leq n, su \neq \varepsilon : \exists i \in \{0, 1, \ldots\}$ mit $rs^i tu^i v \notin L$

Dann ist L nicht kontextfrei.

Beispiele

- $L = \{a^m b^m c^m \mid m \in \mathbb{N}\}, \Sigma = \{a, b, c\}$ ist nicht kontextfrei.
  Sei $n = 3$. Dann kann man $w = aaabbbccc \in L$ zerlegen:

$$w = \underbrace{a}_{r} \underbrace{aa}_{s} \underbrace{b}_{t} \underbrace{\varepsilon}_{u} \underbrace{bbccc}_{v}$$

  Sei nun $i = 0$:

$$rtv = abbbccc \notin L$$

  Allgemeine Strategie für $i = 0$: Sei $n \in \mathbb{N}$. Wähle $w = a^n b^n c^n$. Sei $w = rstuv$ Zerlegung mit $|stu| \leq n$ und $su \neq \varepsilon$.
  Wissen: Da $|stu| \leq n$ ist, kann stu nicht a, b und c enthalten. Wähle $i = 0$; das ergibt Wort rtv, das nicht in L ist, da nicht gleich viele as wie bs wie cs vorkommen.
  L ist damit nicht kontextfrei.
- $L = \{w\#w \mid w \in \{0, 1\}^m\}, \Sigma = \{0, 1, \#\}$ ist nicht kontextfrei.
  Sei $n \in \mathbb{N}$. Wähle $w = 0^n 1^n \# 0^n 1^n, w \in L, |w| \geq n$. Sei $w = rstuv, |stu| \leq n, su \neq \varepsilon$.
  - Fall 1: # ist nicht in stu. Dann ist $rsstuuv \notin L$, da vor # etwas anderes kommt als nach #.
  - Fall 2: # ist in s oder u. Dann ist $rssstuuuv \notin L$, da es # drei Mal enthält.
  - Fall 3: # ist in t. Dann besteht s nur aus 1en und u nur aus 0en, also ist $rsstuuv \notin L$, da vor und hinter # verschiedene Dinge stehen.

  L ist also nicht kontextfrei.
  Folgerung: Es existiert eine kontextsensitive Sprache, die nicht kontextfrei ist! Die Chomsky-Hierarchie ist also echt.

Kellerautomaten Auch genannt Stapelautomaten oder auf Englisch „Pushdown automata“ (PDA).
Haben gelernt:

- Turingmaschinen erkennen (semi-)entscheidbare Sprachen.
  Haben Zustände und unbeschränkten Speicher (Band) mit beliebigem Zugriff (Schreib-/Lesekopf).
- DEAs und NEAs erkennen reguläre Sprachen.
  Haben beschränkten Speicher (Zustände).

PDAs erkennen kontextfreie Sprachen (Maschinenmodell für kontextfreie Sprachen).

Beschreibung Haben Zustände und unbeschränkten Speicher (Keller, Stapel), auf den man nur nach dem LIFO-Prinzip zugreifen kann (PUSH, POP).

Können Eingabe nur von links nach rechts lesen. PDAs sind nichtdeterministisch.

In jedem Schritt sieht der PDA den aktuellen Zustand, das aktuelle Eingabezeichen und das oberste Stacksymbol. Abhängig davon kann der PDA in einen neuen Zustand wechseln, ein Zeichen vom Keller entfernen oder neue Zeichen auf den Keller schreiben und außerdem zum nächsten Eingabezeichen gehen.

Nichtdeterminismus: Kann mehrere solche Operationen zur Verfügung haben und wählt automatisch die richtige aus.

Formale Definition

- Σ: Eingabealphabet
- Γ: Kelleralphabet
- Q: Zustandsmenge
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: Akzeptierende Endzustände
- $\perp \in \Gamma$: Kellerbodensymbol (liegt anfangs auf dem Stack)

Nichtdeterministische Überführungsfunktion:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$$

PDA liest die Eingabe von links nach rechts (Schreib-/Lesekopf), kann dabei Zustände wechseln und den Stapel PUSHen und POPen.

Formal: Konfiguration des PDAs ist ein Tripel $Q \times \Sigma^* \times \Gamma^*$, also (q, w, x) mit q aktueller Zustand, w verbleibende Eingabe und x Kellerinhalt.

Sei $(q, \sigma w', \gamma x')$ mit $\sigma \in \Sigma, w' \in \Sigma^*, \gamma \in \Gamma, x' \in \Gamma^*$ die aktuelle Konfiguration des PDA und $(q', y) \in \delta(q, \sigma, \gamma)$, dann ist (q', w', yx') eine gültige Nachfolgekonfiguration von $(q, \sigma w', \gamma x')$.

Alternativ: Sei $(q, w, \gamma x')$ die aktuelle Konfiguration und $(q', y) \in \delta(q, \varepsilon, \gamma)$. Dann ist (q', w, yx') eine gültige Nachfolgekonfiguration von $(q, w, \gamma x')$.

Sei $w \in \Sigma^*$. Ein PDA akzeptiert w genau dann, wenn eine gültige Konfigurationsfolge von $(q_0, w, \perp)$ nach (q_F, ε, y) mit $q_F \in F$ existiert.

Funktionsweise Erklärung anhand eines Beispiels.

$$\Sigma = \{(,)\}$$
$$L = \{\text{alle gültigen Klammerausdrücke}\}$$

Erkenne L mit Hilfe eines Stapels. Das ist einfach: PUSH bei öffnender Klammer, POP bei schließender Klammer, am Ende muss Stapel leer sein.

Listing 1: Pseudocode

```
while input is not finished
    c ← next symbol
    if c = ( then
        PUSH (
    else if stack is not empty
        POP
    else
```

```
        return false

return stackIsEmpty
```

Beispiel Sei $\Sigma = \{0,1\}$ und $L = \{ww^R \mid w \in \{0,1\}^*\}$, wobei w^R das umgedrehte Wort w ist. Geben Sie einen PDA für L an.

$$\Gamma = \{0, 1, \bot\}$$
$$Q = \{q_0, q_1, q_2, q_3\}$$
$$F = \{q_3\}$$

$$\delta(q_0, 0, \gamma) = \{(q_0, 0\gamma)\}$$
$$\delta(q_0, 1, \gamma) = \{(q_0, 1\gamma)\}$$
$$\delta(q_0, \varepsilon, \gamma) = \{(q_1, \gamma)\}$$
$$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, 0, 1) = \{(q_2, 1)\}$$
$$\delta(q_1, 0, \bot) = \{(q_2, \bot)\}$$
$$\delta(q_1, 1, 1) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, 1, 0) = \{(q_2, 0)\}$$
$$\delta(q_1, 1, \bot) = \{(q_2, \bot)\}$$
$$\delta(q_1, \varepsilon, \bot) = \{(q_3, \bot)\}$$

Sei nun $w = 0110$. Das ergibt folgende Konfigurationsfolge:

$$\begin{aligned}
&(q_0, 0110, \bot) \\
\vdash\ &(q_0, 110, 0\,\bot) \\
\vdash\ &(q_0, 10, 10\,\bot) \\
\vdash\ &(q_1, 10, 10\,\bot) \\
\vdash\ &(q_1, 0, 0\,\bot) \\
\vdash\ &(q_1, \varepsilon, \bot) \\
\vdash\ &(q_3, \varepsilon, \bot)
\end{aligned}$$

Satz Die Sprachen, die von Kellerautomaten akzeptiert werden, sind genau die kontextfreien Sprachen. Kein Beweis in der Vorlesung.