Grundlagen der Theoretischen Informatik

4. Kellerautomaten und kontextfreie Sprachen (III)

9.06.2016

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Übersicht

1. Motivation
2. Terminologie
3. Endliche Automaten und reguläre Sprachen
4. Kellerautomaten und kontextfreie Sprachen
5. Turingmaschinen und rekursiv aufzählbare Sprachen
6. Berechenbarkeit, (Un-)Entscheidbarkeit
7. Komplexitätsklassen P und NP

Umformung von Grammatiken

- **Startsymbol nur links**

  Ist das bei einer Grammatik nicht gegeben, kann man es wie folgt erreichen:
  – Führe ein neues Startsymbol S_{neu} ein
  – Füge die Regel $S_{neu} \rightarrow S$ hinzu.

- **Keine nutzlose Symbole**

Theorem (cf-Grammatik ohne nutzlose Symbole)
Ist $G = (V, T, R, S)$ eine cf-Grammatik mit $L(G) \neq \emptyset$,
dann existiert eine cf-Grammatik $G' = (V', T', R', S')$ mit:

- G' ist äquivalent zu G.
- Jedes $x \in (V \cup T)$ ist erreichbar und co-erreichbar.

Normalform für Regeln

Theorem (Normalform)
Zu jeder Grammatik G (beliebigen Typs) existiert eine äquivalente Grammatik G', bei der für alle Regeln $P \to Q \in R'$ gilt:

- $Q \in V^*$ und P beliebig
- $Q \in T$ und $P \in V$

Für alle Typen außer den linearen hat G' denselben Typ wie G.

Beweis: Für jedes $t \in T$ erzeuge man eine neue Variable V_t.

- $V' = V \cup \{V_t \mid t \in T\}$
- R' entsteht aus R, indem für jede Regel $P \to Q \in R$ in Q alle Vorkommen eines Terminals t durch die zugehörige Variable V_t ersetzt werden. Außerdem enthält R' für jedes $t \in T$ eine neue Regel $V_t \to t$.

Elimination von ε-Regeln

Definition (ε-Regel, nullbare Variablen)

Eine Regel der Form $P \to \varepsilon$ (P eine Variable) heißt **ε-Regel**.

Eine Variable A heißt **nullbar**, falls $A \Longrightarrow^* \varepsilon$

Theorem (ε-Regeln sind eliminierbar)

Zu jeder cf-Grammatik G existiert eine äquivalente cf-Grammatik G'

- ohne ε-Regeln und nullbare Variablen,
  falls $\varepsilon \notin L(G)$,
- mit der einzigen ε-Regel $S \to \varepsilon$ und der einzigen nullbaren Variablen S,
  falls $\varepsilon \in L(G)$ und S das Startsymbol ist.

Elimination von ε-Regeln

- Algorithmus zur Berechnung der Menge Neu der nullbaren Variablen
- Ausgangsgrammatik G habe die Normalform, bei der für jede Regel $P \to Q$: $Q \in V^*$ oder $Q \in T$.

  Für jede Regel $P \to A_1 \ldots A_n$ generiere alle möglichen Kombinationen

  $$P \to \alpha_1 \ldots \alpha_n$$

  mit
  - $\alpha_i \in \{\varepsilon, A_i\}$ falls A_i nullbar
  - $\alpha_i = A_i$ falls A_i nicht nullbar
  - Füge alle diese neuen Regeln zur Grammatik hinzu
  - Entferne alle Regeln der Form $A \to \varepsilon$ mit $A \neq S$

Korollar. $\mathbf{L}_2 \subseteq \mathbf{L}_1$
Das heißt, jede kontextfreie Sprache ist auch kontextsensitiv

Elimination von Kettenproduktionen

Definition. Eine Regel der Form

$$A \to B \qquad \text{mit } A,\ B \in V$$

heißt **Kettenproduktion**.

Theorem. (Kettenproduktionen sind eliminierbar)
Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik ohne Kettenproduktionen.

Elimination von Kettenproduktionen

- Sei $G = (V, T, R, S)$ eine kontextfreie Grammatik ohne ε-Regeln, außer ggf. $S \to \varepsilon$.

  Konstruiere neue Grammatik wie folgt:

  1. Für alle
     - Variablenpaare $A, B \in V, \quad A \neq B \quad$ mit $A \Longrightarrow^* B$
     - Regeln $B \to \alpha \in R, \quad \alpha \notin V$

     füge zu R hinzu:

     $$A \to \alpha$$

  2. Lösche alle Kettenproduktionen

Normalform für cf-Grammatiken

Theorem. Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik

- ohne ε-Regeln
  (bis auf $S \to \varepsilon$, falls ε zur Sprache gehört;
  in diesem Fall darf S in keiner Regelconclusio vorkommen),
- ohne nutzlose Symbole,
- ohne Kettenproduktionen,
- so dass für jede Regel $P \to Q$ gilt: entweder $Q \in V^*$ oder $Q \in T$.

Normalform für cf-Grammatiken

Beweis.

1. Man teste zunächst, ob S nullbar ist. Falls ja, dann verwende man S_{neu} als neues Startsymbol und füge die Regeln $S_{neu} \to S \mid \varepsilon$ zum Regelsatz hinzu.
2. Man eliminiere nutzlose Symbole.
3. Man eliminiere alle ε-Regeln außer $S_{neu} \to \varepsilon$.
4. Man bringe die Grammatik in die Normalform,
   bei der für jede Regel $P \to Q$ gilt: entweder $Q \in V^*$ oder $Q \in T$.
5. Man eliminiere Kettenproduktionen.
6. Zum Schluss eliminiere man noch einmal alle nutzlosen Symbole
   (wg. Schritt 3)

Normalformen

Zwei Normalformen

Chomsky-Normalform: Baut auf den Umformungen des vorigen Teils auf.

Greibach-Normalform: Ähnlich den rechtslinearen Grammatiken.

Chomsky-Normalform

Definition. Eine cf-Grammatik $G = (V, T, R, S)$ ist in **Chomsky-Normalform (CNF)**, wenn gilt:

- G hat nur Regeln der Form

$$A \to BC \quad \text{mit } A, B, C \in V \text{ und}$$
$$A \to a \quad \text{mit } A \in V,\ a \in T \quad (\text{nicht } \varepsilon!)$$

- Ist $\varepsilon \in L(G)$, so darf G zusätzlich die Regel $S \to \varepsilon$ enthalten. In diesem Fall darf S in keiner Regelconclusio vorkommen.
- G enthält keine nutzlosen Symbole.

Theorem. Zu jeder cf-Grammatik existiert eine äquivalente cf-Grammatik in Chomsky-Normalform.

Chomsky-Normalform

Schritt 1: Wende auf G die Umformungen des letzten Abschnitts an.

Ergebnis:

- G hat keine nutzlosen Symbole
- Alle Regeln haben die Form:
  - $A \to \alpha$ mit $A \in V$ und $\alpha \in V^*$, $|\alpha| \geq 2$, und
  - $A \to a$ mit $A \in V$, $a \in T$

Schritt 2: Regeln so umformen, dass keine Conclusio eine Länge größer 2 hat.

Ersetze jede Regel $A \to A_1 \ldots A_n$ mit $A, A_i \in V, n \geq 3$ durch:

$$\begin{aligned} A &\to A_1 C_1 \\ C_1 &\to A_2 C_2 \\ &\vdots \\ C_{n-2} &\to A_{n-1} A_n \end{aligned}$$

Dabei sind die C_i neue Variablen in V.

Greibach-Normalform

Definition.

Eine cf-Grammatik $G = (V, T, R, S)$ ist in **Greibach-Normalform (GNF)**, wenn gilt:

- G hat nur Regeln der Form

  $$A \to a\alpha \text{ mit } A \in V \text{ und } a \in T \text{ und } \alpha \in V^*$$

- Ist $\varepsilon \in L(G)$, so darf G zusätzlich die Regel $S \to \varepsilon$ enthalten. In diesem Fall darf S in keiner Regelconclusio vorkommen.
- G enthält keine nutzlosen Symbole.

Pumping-Lemma für kontextfreie Sprachen

Wiederholung

Theorem (Pumping-Lemma für L_3-Sprachen)
Sei $L \in$ **RAT**.
Dann existiert ein $n \in \mathbb{N}$, so dass:

Für alle

$$x \in L \quad \text{mit} \quad |x| \geq n$$

existiert eine Zerlegung

$$x = uvw \qquad u, v, w \in \Sigma^*$$

mit

- $|v| \geq 1$
- $|v| < n$
- $uv^m w \in L \quad$ für alle $m \in \mathbb{N}$

Pumping-Lemma für kontextfreie Sprachen

Theorem (Pumping-Lemma für kontextfreie Sprachen)

Sei L kontextfrei.

Dann existiert ein $n \in \mathbb{N}$, so dass:

Für alle

$$z \in L \quad \text{mit} \quad |z| \geq n$$

existiert eine Zerlegung

$$z = uvwxy \qquad u, v, w, x, y \in \Sigma^*$$

mit

- $|vx| \geq 1$
- $|vwx| < n$
- $uv^m wx^m y \in L \quad$ für alle $m \in \mathbb{N}$

Pumping-Lemma für kontextfreie Sprachen

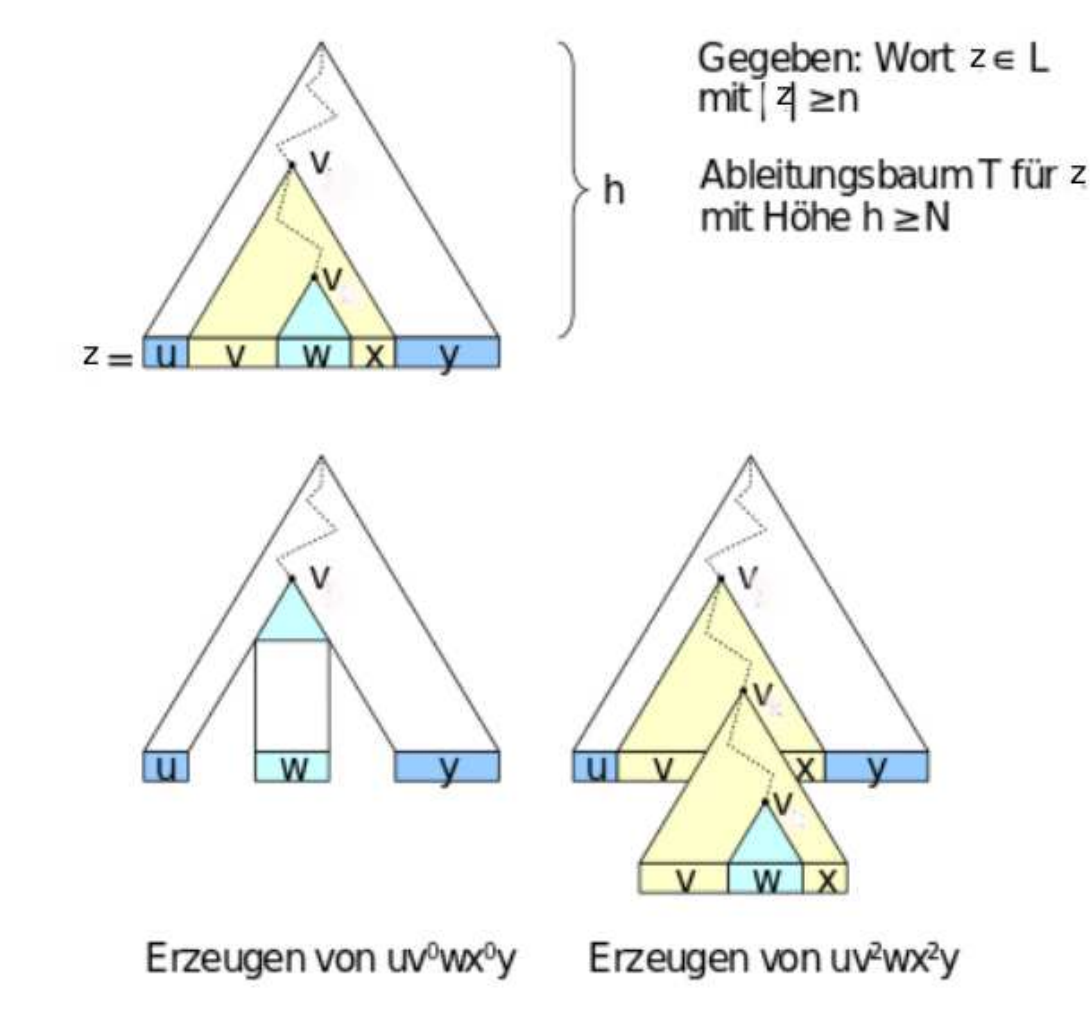
Beweisidee:

Bei der Ableitung eines hinreichend langen Wortes muss es eine Variable geben, die mehr als einmal auftaucht.

Dies führt zu einer Schleife in der Ableitung, die aufgepumpt werden kann.

Pumping-Lemma für kontextfreie Sprachen

Beweisidee:



Pumping-Lemma für kontextfreie Sprachen

Anwendung des Pumping-Lemmas für cf-Sprachen

Wenn das cf-Pumping-Lemma für eine Sprache nicht gilt,
dann kann sie nicht kontextfrei sein.

Pumping-Lemma für kontextfreie Sprachen

Anwendung des Pumping-Lemmas für cf-Sprachen

Wenn das cf-Pumping-Lemma für eine Sprache nicht gilt,
dann kann sie nicht kontextfrei sein.

Beispiel (Sprachen, die nicht kontextfrei sind)

Für folgende Sprachen kann man mit Hilfe des cf-Pumping-Lemmas zeigen, dass sie nicht kontextfrei sind:

- $\{a^p \mid p \text{ prim}\}$
- $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- $\{zzz \mid z \in \{a, b\}^*\}$.

Pumping-Lemma für kontextfreie Sprachen

$L_1 = \{a^p \mid p \text{ prim}\}$ ist nicht kontextfrei.

Beweis: Wir nehmen an, L_1 sei kontextfrei.

Sei dann n die zugehörige Konstante aus dem Pumping-Lemma.

Wir betrachten das Wort $z = a^p$, wobei p prim und $p \geq n + 2$.

Es muss dann eine Zerlegung $z = uvwxy$ geben, so dass:
$|vx| \geq 1, \quad |vwx| < n, \quad uv^i wx^i y \in L_1$ für alle $i \geq 0$.

Dann $u = a^{i_1}, v = a^{i_2}, w = a^{i_3}, x = a^{i_4}, y = a^{i_5}$ mit

- $i_1 + i_2 + i_3 + i_4 + i_5 = p$
- $i_2 + i_4 \geq 1, \quad i_2 + i_3 + i_4 < n$
- $i_1 + mi_2 + i_3 + mi_4 + i_5$ prim für alle $m \geq 0$.

Sei $m = i_1 + i_3 + i_5$. Dann kann $i_1 + mi_2 + i_3 + mi_4 + i_5 = (i_1 + i_3 + i_5)(1 + i_2 + i_4)$ nicht prim sein, da $i_1 + i_3 + i_5 = p - (i_2 + i_4) \geq p - n \geq 2$ und $1 + i_2 + i_4 \geq 2$.

Also $uv^m wx^m y \notin L_1$. Widerspruch.

Also kann L_1 nicht kontextfrei sein.

Pumping-Lemma für kontextfreie Sprachen

$L_2 = \{a^m b^m c^m \mid m \in \mathbb{N}\}$ ist nicht kontextfrei

Beweis:

Wir nehmen an, L_2 sei kontextfrei.

Sei dann n die zugehörige Konstante aus dem Pumping-Lemma.

Wir betrachten das Wort $z = a^n b^n c^n$.

Es muss dann eine Zerlegung $z = uvwxy$ geben, so dass:
$|vx| \geq 1, \quad |vwx| < n, \quad uv^i wx^i y \in L_2$ für alle $i \geq 0$.

Da $|vwx| < n$, enthält das Wort vwx höchstens zwei verschiedene Buchstaben.

Da $|vx| \geq 1$, kann $uv^2 wx^2 y$ nicht von allen drei Buchstaben gleich viele enthalten.

Also kann L_2 nicht kontextfrei sein.

Pumping-Lemma für kontextfreie Sprachen

$L_3 = \{zzz \mid z \in \{a, b\}^*\}$ ist nicht kontextfrei.

Beweis: Wir nehmen an, L_3 sei kontextfrei. Sei dann n die zugehörige Konstante aus dem Pumping-Lemma.

Wir betrachten das Wort $z = a^n b a^n b a^n b$.

Es muss dann eine Zerlegung $z = uvwxy$ geben, so dass:
$|vx| \geq 1, \quad |vwx| < n, \quad uv^i wx^i y \in L_3$ für alle $i \geq 0$.

Da $|vwx| < n$, enthält das Wort vwx höchstens ein b.

Fall 1: $vwx = a^k$. Dann wird durch aufpumpen von v, x eine a Sequenz länger als die anderen, so $uv^2 wx^2 y \notin L_3$.

Fall 2: $vwx = a^k b a^m$.

Fall 2.1: b kommt nicht in v oder x vor. Dann sind in $uv^2 wx^2 y$ zwei a Sequenzen länger als die dritte, so $uv^2 wx^2 y \notin L_3$.

Fall 2.2: b kommt in v oder x vor. Dann enthält $uv^0 wx^0 y$ nur zwei b's, d.h. $uv^0 wx^0 y \notin L_3$.

Also kann L_3 nicht kontextfrei sein.

Pushdown-Automaten (PDAs)

Grammatiken vs. Automaten

Erzeugende Grammatiken– akzeptierende Automaten

Erinnerung: Reguläre Sprachen

- werden erzeugt von rechtslinearen Grammatiken
- werden akzeptiert von endlichen Automaten

Grammatiken vs. Automaten

Erzeugende Grammatiken– akzeptierende Automaten

Erinnerung: Reguläre Sprachen

- werden erzeugt von rechtslinearen Grammatiken
- werden akzeptiert von endlichen Automaten

Jetzt: Kontextfreie Sprachen

- werden erzeugt von kontextfreien Grammatiken
- werden akzeptiert von **Pushdown-Automaten**

Idee des Push-Down-Automaten

Beispiel

Die „prototypische“ cf-Sprache

$$\{a^n b^n \mid n \in \mathbb{N}\}$$

- Endliche Automaten reichen nicht aus.
- Sie können sich nicht merken, wie oft sie einen Zustand durchlaufen haben.
- Für $a^n b^n$ muss man aber **mitzählen**.

Idee des Push-Down-Automaten

Idee: Wie kann man diese Sprache akzeptieren?

- Weitere Informationen auf dem Stack sichern
- Später zurückholen
- Ähnlich einem "Prozeduraufruf"
- Grammatikregel wie $S \to aAb$ entspricht Aufruf einer Prozedur für das A.

Idee des Push-Down-Automaten

Idee: Wie kann man diese Sprache akzeptieren?

- Weitere Informationen auf dem Stack sichern
- Später zurückholen
- Ähnlich einem "Prozeduraufruf"
- Grammatikregel wie $S \to aAb$ entspricht Aufruf einer Prozedur für das A.

Stack, Stapel, Keller

- Last in, first out
- Zuletzt gespeicherte Information liegt immer "obenauf"
- Beliebig viel Information kann gespeichert werden
  (Aber kein beliebiger Zugriff!)

Push-Down-Automat

Push-Down-Automat (PDA): Informell

- Wie endlicher Automat, aber **zusätzlicher** einen Stack
- Übergangsrelation bezieht das oberste Stacksymbol in den Übergang ein
- Bei Zustandsübergang: lesen und schreiben auf Stack

Push-Down-Automat

Definition(Push-Down-Automat)

Ein **Push-Down-Automat (PDA)** ist ein Tupel

$$\mathcal{M} = (K, \Sigma, \Gamma, \Delta, s_0, Z_0, F)$$

Dabei ist:

K	eine endliche Menge von Zuständen
Σ	das Eingabealphabet
Γ	das Stack- oder Kelleralphabet
$s_0 \in K$	der Startzustand
$Z_0 \in \Gamma$	das Anfangssymbol im Keller
$F \subseteq K$	eine Menge von finalen Zuständen
Δ	die Zustandsübergangsrelation, eine endliche Relation: $\Delta \subset (K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (K \times \Gamma^*)$

Push-Down-Automat

Arbeitsschritt eines PDA

In Abhängigkeit

- vom aktuellen Zustand
- vom nächsten Eingabezeichen (oder auch unabhängig davon)
- vom obersten Kellersymbol

geschieht folgendes

- nächstes **Eingabezeichen** wird **gelesen oder nicht** (bei ε),
- das oberste **Kellersymbol** wird **entfernt**,
- der **Zustand** wird **geändert**,
- es werden null oder mehr **Zeichen auf den Keller** geschoben
  Bei neuen Keller-Wort $\gamma = A_1 \dots A_n$ wird A_n zuerst auf den Keller geschoben usw., so dass am Schluss A_1 obenauf liegt.

Push-Down-Automat

Notation

- a, b, c für Buchstaben aus Σ
- u, v, w für Wörter aus Σ^*
- A, B für Stacksymbole aus Γ
- γ, η für Stackinhalte aus Γ^*

Push-Down-Automat: Konfiguration

Konfiguration eines PDA: Informell

- Konfiguration beschreibt die aktuelle Situation des PDA **komplett**
- Bestandteile:
  - **aktueller Zustand**
  - **noch zu lesendes Restwort**
  - **kompletter Stackinhalt**
- Für Konfigurationen C_1, C_2 bedeutet

$$C_1 \vdash C_2$$

  dass der PDA in einem Schritt von C_1 nach C_2 gelangen kann.

Push-Down-Automat: Konfiguration

Definition (Konfiguration eines PDA)

Eine **Konfiguration** C eines PDA $\mathcal{M} = (K, \Sigma, \Gamma, \Delta, s_0, Z_0, F)$ ist ein Tripel

$$(q, w, \gamma) \in K \times \Sigma^* \times \Gamma^*.$$

- q der aktuelle Zustand
- w der noch zu lesendes Restwort
- γ der komplette Stackinhalt

Push-Down-Automat: Konfiguration

Definition (Konfiguration eines PDA)

Eine **Konfiguration** C eines PDA $\mathcal{M} = (K, \Sigma, \Gamma, \Delta, s_0, Z_0, F)$ ist ein Tripel

$$(q, w, \gamma) \in K \times \Sigma^* \times \Gamma^*.$$

- q der aktuelle Zustand
- w der noch zu lesendes Restwort
- γ der komplette Stackinhalt

Definition (Startkonfiguration)

Bei Eingabewort w ist die **Startkonfiguration**:

$$(s_0, w, Z_0)$$

Push-Down-Automat: Konfiguration

Definition (Nachfolgekonfiguration)

C_2 heißt **Nachfolgekonfiguration** von C_1,

$$C_1 \vdash C_2$$

falls

$$\exists a \in \Sigma \; \exists A \in \Gamma \; \exists w \in \Sigma^* \; \exists \gamma, \eta \in \Gamma^*$$

so dass

entweder $C_1 = (q_1, aw, A\gamma)$, $C_2 = (q_2, w, \eta\gamma)$, und $(q_1, a, A) \; \Delta \; (q_2, \eta)$,

oder $C_1 = (q_1, w, A\gamma)$, $C_2 = (q_2, w, \eta\gamma)$, und $(q_1, \varepsilon, A) \; \Delta \; (q_2, \eta)$,

Push-Down-Automat: Rechnung

Definition (Rechnung eines PDA)

Sei $\mathcal{M}$ ein Push-Down-Automat.

$$C \vdash_M^* C'$$

gdw es eine Reihe von Konfigurationen

$$C_0, C_1, \ldots, C_n \qquad (n \geq 0)$$

so dass

- $C = C_0$,
- $C' = C_n$,
- $C_i \vdash_M C_{i+1}$ für alle $0 \leq i < n$

Dann heißt $C_0, C_1, \ldots, C_n$ eine **Rechnung** von $\mathcal{M}$

Push-Down-Automat: Akzeptierte Sprache

Definition (von PDA akzeptierte Sprache)

Ein PDA $\mathcal{M}$ kann auf zwei verschiedene Arten eine Sprache akzeptieren:

- über **finale Zustände**
- über **leeren Keller**

$$L_f(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in F\ \exists \gamma \in \Gamma^*\ \left((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \gamma)\right)\}$$

$$L_l(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in K\ \left((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \varepsilon)\right)\}$$

Push-Down-Automat: Akzeptierte Sprache

Definition (von PDA akzeptierte Sprache)

Ein PDA $\mathcal{M}$ kann auf zwei verschiedene Arten eine Sprache akzeptieren:

- über **finale Zustände**
- über **leeren Keller**

$$L_f(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in F \; \exists \gamma \in \Gamma^* \; \big((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \gamma)\big)\}$$

$$L_l(\mathcal{M}) = \{w \in \Sigma^* \mid \exists q \in K \; \big((s_0, w, Z_0) \vdash_{\mathcal{M}}^* (q, \varepsilon, \varepsilon)\big)\}$$

Bemerkung

Das zu akzeptierende Wort w muss von $\mathcal{M}$ ganz gelesen werden:

$$(s_0, w, Z_0) \vdash^* (q, \varepsilon, \cdot)$$

ist gefordert.

Push-Down-Automat

Bemerkung

- Das unterste Symbol im Keller kann gelöscht werden.
- Dann aber **hängt** der PDA
- Er kann nicht mehr weiter rechnen
- **Es gibt keine Nachfolgekonfiguration**

Push-Down-Automat: Beispiel

Beispiel

Sprache der Palindrome über $\{a, b\}$:

$$L = \{w \in \{a, b\}^* \mid w = w^R\}$$

L wird über leeren Keller akzeptiert von dem PDA

$$\mathcal{M} := (\{s_0, s_1\}, \{a, b\}, \{Z_0, A, B\}, \Delta, s_0, Z_0, \emptyset)$$

mit ...

Push-Down-Automat: Beispiel

Beispiel (Forts.) **Idee:**

- Ein Palindrom $w = w^R$ hat die Form

  $$vv^R \quad \text{oder} \quad vav^R \quad \text{oder} \quad vbv^R$$

  für ein $v \in \{a, b\}^*$
- Der Automat $\mathcal{M}$ liest v und merkt sich jeden Buchstaben.
- **Er rät indeterminiert die Wortmitte.**

  Falls das Wort eine ungerade Anzahl von Buchstaben hat,
  also $w = vav^R$ oder $w = vbv^R$,
  dann muss dabei ein Buchstabe überlesen werden.
- Der Stack enthält nun v^R.
  $\mathcal{M}$ muss jetzt nur noch jeden weiteren gelesenen Buchstaben mit dem jeweils obersten Kellersymbol vergleichen.

Push-Down-Automat: Beispiel

Beispiel (Forts.)

$(s_0, \varepsilon, Z_0)\ \ \Delta\ (s_1, \varepsilon)$ } ε akzeptieren

$(s_0, a, Z_0)\ \ \Delta\ (s_0, A)$
$(s_0, a, A)\ \ \Delta\ (s_0, AA)$
$(s_0, a, B)\ \ \Delta\ (s_0, AB)$
$(s_0, b, Z_0)\ \ \Delta\ (s_0, B)$
$(s_0, b, A)\ \ \Delta\ (s_0, BA)$
$(s_0, b, B)\ \ \Delta\ (s_0, BB)$
} Stack aufbauen

Push-Down-Automat: Beispiel

Beispiel (Forts.)

$$\left.\begin{array}{ll} (s_0, \varepsilon, A) & \Delta\,(s_1, \varepsilon) \\ (s_0, \varepsilon, B) & \Delta\,(s_1, \varepsilon) \end{array}\right\} \text{Richtungswechsel für Palindrome mit ungerader Buchstabenanzahl}$$

$$\left.\begin{array}{ll} (s_0, a, A) & \Delta\,(s_1, \varepsilon) \\ (s_0, b, B) & \Delta\,(s_1, \varepsilon) \end{array}\right\} \text{Richtungswechsel für Palindrome mit gerader Buchstabenanzahl}$$

$$\left.\begin{array}{ll} (s_1, a, A) & \Delta\,(s_1, \varepsilon) \\ (s_1, b, B) & \Delta\,(s_1, \varepsilon) \end{array}\right\} \text{Stack abbauen}$$

Push-Down-Automat: Beispiel

Beispiel (Forts.)

Für das Eingabewort *abbabba* rechnet $\mathcal{M}$ so:

$$
\begin{array}{ccccccc}
(s_0, abbabba, Z_0) & \vdash & (s_0, bbabba, A) & \vdash & (s_0, babba, BA) & \vdash \\
(s_0, abba, BBA) & \vdash & (s_0, bba, ABBA) & \vdash & (s_1, bba, BBA) & \vdash \\
(s_1, ba, BA) & \vdash & (s_1, a, A) & \vdash & (s_1, \varepsilon, \varepsilon) &
\end{array}
$$