

# FORMALE SYSTEME

## 15. Vorlesung: Einleitung Kellerautomaten

Hannes Straß

Folien: © Markus Krötzsch, <https://iccl.inf.tu-dresden.de/web/FS2020>, CC BY 3.0 DE

TU Dresden, 29. November 2021

## Kellerautomaten

## (Nicht-)Abschlusseigenschaften für Typ 2

**Satz:** Wenn  $L$ ,  $L_1$  und  $L_2$  kontextfreie Sprachen sind, dann sind auch die folgenden Sprachen kontextfrei:

- (1)  $L_1 \cup L_2$  (Abschluss unter Vereinigung)
- (2)  $L_1 \circ L_2$  (Abschluss unter Konkatenation)
- (3)  $L^*$  (Abschluss unter Kleene-Stern)

Aber:

**Satz:** Es gibt kontextfreie Sprachen  $L$ ,  $L_1$  und  $L_2$ , so dass die folgenden Sprachen nicht kontextfrei sind:

- (1)  $L_1 \cap L_2$  (Nichtabschluss unter Schnitt)
- (2)  $\bar{L}$  (Nichtabschluss unter Komplement)

## Ein Berechnungsmodell für Typ-2-Sprachen

Eine typische Problemsprache (kontextfrei, aber nicht regulär):

$$\{a^i b^i \mid i \geq 0\}$$

Warum können endliche Automaten diese Sprache nicht erkennen?

Weil sie die Zahl der gelesenen **a**s nicht **speichern** können.

↪ Wir wollen endlichen Automaten mehr Speicher geben.

## Pseudoalgorithmus $a^i b^i$

```
function isWordFromAiBi() {
  let state = "readA", anum = 0, symbol = "";
  while ( hasNextSymbol() ) {
    symbol = getNextSymbol();
    if ( state == "readA" ) {
      if ( symbol == a ) {
        anum = anum + 1;
      } else if ( symbol == b ) {
        anum = anum - 1; state = "readB";
      }
    } else if ( state == "readB" ) {
      if ( symbol == a ) {
        return false;
      } else if ( symbol == b ) {
        anum = anum - 1;
      }
    }
  }
  return anum == 0;
}
```

### Speicherbedarf?

- Zustandsvariable state (1 Bit – zwei mögliche Werte)
- Zähler anum (beliebig große ganze Zahl!)

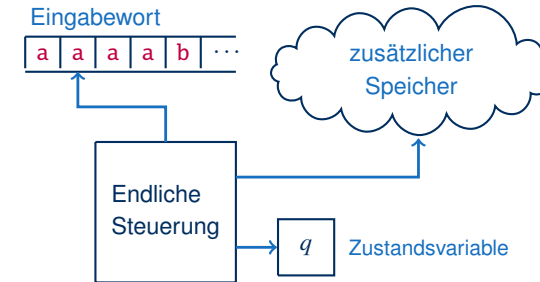
Der Algorithmus benötigt eine unbeschränkte Menge an Speicher, je nach Eingabe

~> kein endlicher Automat

## Endliche Automaten + Speicher

Das vorige Beispiel verwendet eine Zustandsvariable, fast wie ein endlicher Automat ...

Wir müssten endliche Automaten irgendwie durch einen unbegrenzt großen „Speicher“ ergänzen. Aber welche Form soll dieser Speicher haben?



## Welche Form hat der Speicher?

Das vorige Beispiel verwendet eine beliebig große Zahl.

Im Allgemeinen sind Zahlen aber nicht das sinnvollste Format für die Speicherung der Informationen, die beim Erkennen kontextfreier Sprachen anfallen.

Beispiel: Die folgende kontextfreie Grammatik erkennt Palindrome über  $\Sigma = \{a, b\}$ , deren Mitte mit  $c$  markiert ist:

$$S \rightarrow aSa \mid bSb \mid c$$

Die Grammatik generiert zum Beispiel die Wörter  $abcb$  und  $abaabcb$ .

Diese Sprache ist ebenfalls nicht regulär (Übung).

Wie würde man die Sprache aus diesem Beispiel erkennen?

## Pseudoalgorithmus für markierte Palindrome

```
function isMarkedPalindrome() {
  let state = "start", list = [], symbol = "";
  while ( hasNextSymbol() ) {
    symbol = getNextSymbol();
    if ( state == "start" ) {
      if ( symbol == c ) {
        state = "end";
      } else {
        list.append(symbol);
      }
    } else if ( state == "end" ) {
      if ( !list.isEmpty()
        && symbol == list.lastEntry() ) {
        list.removeLastEntry();
      } else {
        return false;
      }
    }
  }
  return list.isEmpty();
}
```

### Speicherformat?

- Liste von Symbolen
- Zugriffsmethoden:
  - list.append(symbol)
  - list.lastEntry()
  - list.removeLastEntry()
  - list.isEmpty()

~> Stack (Keller)

## Keller sind Stapel

Ein **Stack** (oder auch **Keller** oder auch **Stapel**) ist eine Datenstruktur, die eine Liste von Einträgen speichert.

Wir denken uns die Liste vertikal, mit den letzten Einträgen oben (daher der Name).

Wichtigste Zugriffsoperationen:

- **push**: Ein Element wird oben auf dem Stapel abgelegt,
- **pop**: das oberste Element wird vom Stapel entfernt und zurückgeliefert (wenn der Stapel nicht leer ist).

Keller sind demnach **LIFO-Speicher** (last-in/first-out).

**Idee**: Verwende Keller als zusätzliche Speicher in endlichen Automaten.

## Kellerautomaten definieren

**Übergangsfunktion von Kellerautomaten:**

**Eingabe**: Zustand + Eingabesymbol + Kellersymbol (pop)

**Ausgabe**: Zustand + Kellersymbol (push)

**Weitere Designentscheidungen:**

- Wir unterscheiden mögliche Eingabesymbole  $\Sigma$  von möglichen Kellersymbolen  $\Gamma$  (Gamma).
- Wir wollen erlauben, ...
  - (1) ... dass der Automat manchmal kein Eingabesymbol liest ( $\epsilon$ -Übergänge);
  - (2) ... dass der Automat manchmal kein Kellersymbol liest;
  - (3) ... dass der Automat manchmal nichts auf den Keller schreibt.
- Wir wollen Kellerautomaten als nichtdeterministische Automaten definieren.  
~> Die Übergangsfunktion liefert eine Menge von Möglichkeiten.

## Automaten mit Kellern

Wie kann ein Automat einen Keller lesen oder schreiben?

**Zustandsänderungen bei endlichen Automaten:**

Abhängig von  
– aktuellem Zustand  
– Eingabesymbol

Bestimme neuen Zustand

**Beispiel:** „Wenn in Zustand  $q_1$  Symbol **a** gelesen wird, wechsele in Zustand  $q_2$ .“

**Zustandsänderungen bei endlichen Automaten mit Keller:**

Abhängig von  
– aktuellem Zustand  
– Eingabesymbol  
– oberstem Kellersymbol

Bestimme neuen Zustand  
Schreibe auf Keller

**Beispiel:** „Wenn in Zustand  $q_1$  Symbol **a** gelesen wird und Symbol **C** vom Keller gelesen wird, dann wechsele in Zustand  $q_2$  und schreibe **D** auf den Keller.“

## Kellerautomaten

Zur Vereinfachung der folgenden Definition schreiben wir  $\Sigma_\epsilon$  für  $\Sigma \cup \{\epsilon\}$  (analog für  $\Gamma_\epsilon$ ).

Ein **Kellerautomat** (international: „PDA“, „Pushdown Automaton“)  $\mathcal{M}$  ist ein Tupel  $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, Q_0, F \rangle$  mit den folgenden Bestandteilen:

- $Q$ : endliche Menge von **Zuständen**
- $\Sigma$ : **Eingabealphabet**
- $\Gamma$ : **Kelleralphabet**
- $\delta$ : **Übergangsfunktion**, eine totale Funktion

$$Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow 2^{Q \times \Gamma_\epsilon},$$

wobei  $2^{Q \times \Gamma_\epsilon}$  die Potenzmenge von  $Q \times \Gamma_\epsilon$  ist.

- $Q_0$ : Menge möglicher **Startzustände**  $Q_0 \subseteq Q$
- $F$ : Menge von **Endzuständen**  $F \subseteq Q$

## Kellerautomaten: Intuition (1)

$\delta$ : Übergangsfunktion, eine totale Funktion  $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow 2^{Q \times \Gamma_\epsilon}$ .

### Beispiele:

- $\langle q_2, D \rangle \in \delta(q_1, a, C)$  bedeutet:  
„Wenn der PDA im Zustand  $q_1$  das Symbol **a** einliest und **C** oben vom Keller nimmt (pop), dann kann er in den Zustand  $q_2$  wechseln und dabei **D** auf den Keller legen (push).“
- $\langle q_2, \epsilon \rangle \in \delta(q_1, a, \epsilon)$  bedeutet:  
„Wenn der PDA im Zustand  $q_1$  das Symbol **a** einliest, dann kann er in den Zustand  $q_2$  wechseln und den Keller unverändert lassen.“
- $\langle q_2, D \rangle \in \delta(q_1, \epsilon, C)$  bedeutet:  
„Wenn der PDA im Zustand  $q_1$  das Symbol **C** oben vom Keller nimmt (pop), dann kann er in den Zustand  $q_2$  wechseln und dabei **D** auf den Keller legen (push), ohne ein Symbol zu lesen.“

## Beispiel $a^i b^i$

Ein PDA für  $\{a^i b^i \mid i \geq 0\}$  könnte wie folgt aussehen:

$$\begin{aligned} \Sigma &= \{a, b\} \\ \Gamma &= \{A, S\} \\ Q &= \{q_s, q_a, q_b, q_f\} \\ Q_0 &= \{q_s\} \\ F &= \{q_s, q_f\} \end{aligned}$$

Wir definieren  $\delta$  wie folgt:

$$\begin{aligned} \delta(q_s, \epsilon, \epsilon) &= \{(q_a, S)\} && \text{(markiere Kellerboden)} \\ \delta(q_a, a, \epsilon) &= \{(q_a, A)\} && \text{(speichere a)} \\ \delta(q_a, b, A) &= \{(q_b, \epsilon)\} && \text{(erstes b)} \\ \delta(q_b, b, A) &= \{(q_b, \epsilon)\} && \text{(weitere b)} \\ \delta(q_b, \epsilon, S) &= \{(q_f, \epsilon)\} && \text{(mögliches Wortende)} \end{aligned}$$

Dies sind alle Übergänge (d.h. alle anderen Fälle liefern  $\emptyset$ ).

## Kellerautomaten: Intuition (2)

$\delta$ : Übergangsfunktion, eine totale Funktion  $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow 2^{Q \times \Gamma_\epsilon}$ .

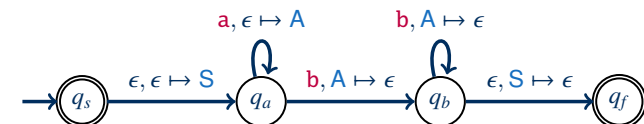
### Beispiele:

- $\langle q_2, D \rangle \in \delta(q_1, \epsilon, \epsilon)$  bedeutet:  
„Wenn der PDA im Zustand  $q_1$  ist, dann kann er spontan in den Zustand  $q_2$  wechseln und dabei **D** auf den Keller legen (push), ohne dabei eine Eingabe oder den Keller zu lesen.“
- $\delta(q_1, a, C) = \emptyset$  bedeutet:  
„Wenn der PDA im Zustand  $q_1$  das Symbol **a** einliest und **C** oben vom Keller nimmt (pop), dann kann er nichts mehr tun (kein erfolgreicher Lauf).“
- ... und viele andere Varianten.

## Grafische Darstellung von PDAs

Wir können PDAs ähnlich zu NFAs darstellen, wenn wir die Änderungen des Kellerinhalts zusätzlich an die Übergänge schreiben.

Beispiel des vorigen PDA für  $\{a^i b^i \mid i \geq 0\}$ :



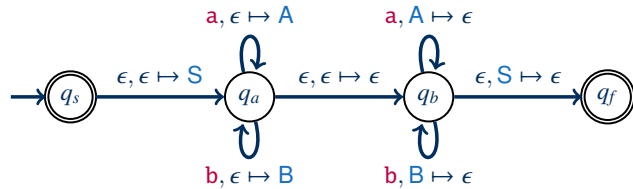
**Kodierungstrick:** Wir implementieren hier die Funktion isEmpty indem wir den Kelleranfang mit einem speziellen Symbol S markieren, welches den (fast) leeren Keller signalisiert.

## Beispiel: Palindrome ohne Markierung

Wir betrachten die Sprache aller Palindrome über  $\{a, b\}$  mit gerader Wortlänge, d.h. die Sprache  $\{ww^r \mid w \in \{a, b\}^*\}$  wobei  $w^r$  die „rückwärts gelesene“ Version von  $w$  ist.

(Für  $w = a_1 \dots a_n$  ist also  $w^r = a_n \dots a_1$ .)

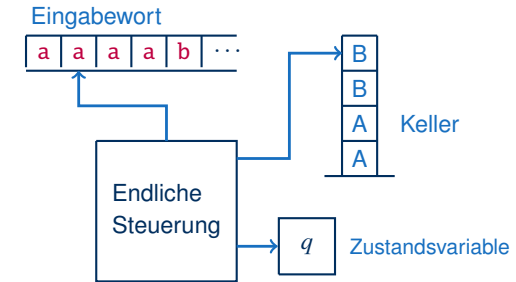
Wir können diese mit folgendem PDA erkennen ( $\Gamma = \{A, B, S\}$ ):



Anmerkung: Wir erraten hier die Wortmitte nichtdeterministisch.

## Konfigurationen eines PDA

Wir können die Architektur von PDAs wie folgt veranschaulichen:



Eine mögliche Konfiguration des PDA während der Erkennung ist gegeben durch den Zustand  $q \in Q$ , den Inhalt des Kellers  $\gamma \in \Gamma^*$  und das noch zu lesende Restwort  $w \in \Sigma^*$ .

## Die Sprache eines PDA (formal)

Sei  $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, Q_0, F \rangle$  ein PDA.

Eine Konfiguration von  $\mathcal{M}$  ist ein Tripel  $\langle q, w, \gamma \rangle \in Q \times \Sigma^* \times \Gamma^*$ .

Wir definieren eine Übergangsrelation zwischen Konfigurationen wie folgt: Die Konfiguration  $\langle q, vw, \psi\gamma \rangle$  kann immer dann in die Konfiguration  $\langle q', w, \psi'\gamma \rangle$  übergehen, in Symbolen  $\langle q, vw, \psi\gamma \rangle \vdash \langle q', w, \psi'\gamma \rangle$ , falls  $\langle q', \psi' \rangle \in \delta(q, v, \psi)$  gilt.<sup>a</sup>

Mit  $\vdash^*$  bezeichnen wir den reflexiven, transitiven Abschluss von  $\vdash$ .

<sup>a</sup>Anmerkung: In diesem Fall ist  $v \in \Sigma_\epsilon$  und  $\psi, \psi' \in \Gamma_\epsilon$ .

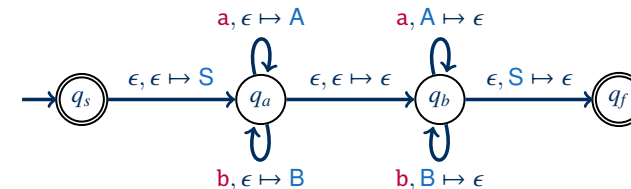
Dies entspricht der intuitiven Idee von möglichen Übergängen.

Der PDA  $\mathcal{M}$  akzeptiert ein Wort  $w$  genau dann, wenn es eine Folge von Übergängen  $\langle q_s, w, \epsilon \rangle \vdash^* \langle q_f, \epsilon, \gamma \rangle$  für ein  $q_s \in Q_0$  und  $q_f \in F$  gibt.

Die von  $\mathcal{M}$  akzeptierte Sprache  $L(\mathcal{M})$  ist die Menge aller von  $\mathcal{M}$  akzeptierten Wörter.

## Beispiel

Der Automat



erlaubt z.B. die folgenden Konfigurationsübergänge:

$\langle q_s, abba, \epsilon \rangle \vdash \langle q_a, abba, S \rangle \vdash \langle q_a, bba, AS \rangle \vdash \langle q_a, ba, BAS \rangle \vdash \langle q_a, a, BBAS \rangle \vdash \langle q_a, \epsilon, ABBAS \rangle$   
 $\leadsto$  kein akzeptierender Lauf

$\langle q_s, abba, \epsilon \rangle \vdash \langle q_a, abba, S \rangle \vdash \langle q_a, bba, AS \rangle \vdash \langle q_a, ba, BAS \rangle \vdash \langle q_b, ba, BAS \rangle \vdash \langle q_b, a, AS \rangle \vdash \langle q_b, \epsilon, S \rangle \vdash \langle q_f, \epsilon, \epsilon \rangle$   
 $\leadsto$  akzeptierender Lauf

## Quiz: Kellerautomat

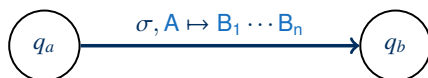
**Konfigurationsübergangsrelation:**  $\langle q, vw, \psi\gamma \rangle \vdash \langle q', w, \psi'\gamma \rangle$  gdw.  $\langle q', \psi' \rangle \in \delta(q, v, \psi)$ .  
**PDA  $\mathcal{M}$  akzeptiert Wort  $w$  gdw.**  $\langle q_s, w, \epsilon \rangle \vdash^* \langle q_f, \epsilon, \gamma \rangle$  für ein  $q_s \in Q_0$  und  $q_f \in F$ .

**Quiz:** Gegeben sei Kellerautomat  $\mathcal{M} = \langle Q, \{\emptyset, \mathbf{1}\}, \{S, X\}, \delta, \{q_0\}, \{q_4\} \rangle$  mit  $\delta$  wie folgt:

...

## Wörter pushen

Wir können Übergänge der folgenden Form erlauben:



„Wenn in Zustand  $q_a$  das Zeichen (oder leere Wort)  $\sigma$  gelesen wird und wenn  $A$  oben vom Keller geholt werden kann (pop), dann wechsele in den Zustand  $q_b$  und lege die Zeichen  $B_1 \cdots B_n$  auf den Keller (wobei  $B_1$  ganz oben zu liegen kommt).“

Dies kann mittels neuer Zustände kodiert werden:



## PDA: Variationen

Es gibt auch hier viele äquivalente Definitionen:

- Ein Startzustand reicht auch aus. (Übung: Warum?)
- Alternative Akzeptanzbedingung: Ersetze „PDA ist im Endzustand“ durch „PDA hat leeren Keller“. (Keine Endzustände nötig, dafür manchmal aufwändigere Kodierung.)
- Die Übergangsfunktion  $\delta$  kann auch als Relation  $\Delta \subseteq (Q \times \Sigma_\epsilon \times \Gamma_\epsilon) \times (Q \times \Gamma_\epsilon)$  beschrieben werden (wie bei NFAs).
- Das Startsymbol des Kellers kann Teil der Definition sein.
- Man kann erlauben, dass in einem Schritt mehrere Symbole auf den Keller geschrieben werden.  
 Dann kann man auch verlangen, dass in jedem Schritt ein Symbol entnommen wird (pop), da man es bei Bedarf zurücklegen kann.

Jede dieser Varianten führt zur gleichen Ausdrucksstärke (akzeptierten Sprachklasse).

## Ausdrucksstärke von PDAs

Man kann nun formal untersuchen, welche Sprachen durch PDAs akzeptiert werden können. Wir erhalten das erhoffte Resultat:

**Satz:**

Eine Sprache ist genau dann kontextfrei, wenn sie von einem PDA akzeptiert wird.

Der Beweis erfolgt in zwei Schritten:

- (1) Umwandlung Typ-2-Grammatik  $\rightsquigarrow$  PDA
- (2) Umwandlung PDA  $\rightsquigarrow$  Typ-2-Grammatik

## Grammatik $\rightsquigarrow$ PDA

Wir wollen zunächst eine Richtung zeigen:

**Satz:**  
Für jede kontextfreie Grammatik  $G$  gibt es einen PDA  $\mathcal{P}_G$ , so dass  $\mathbf{L}(G) = \mathbf{L}(\mathcal{P}_G)$ .

### Beweisidee:

- Der PDA simuliert eine Ableitung in der Grammatik.
- Der Keller speichert das Zwischenergebnis der Ableitung (Terminale und Nichtterminale der rechten Seiten von Ableitungsregeln), linke Zeichen zuoberst.
- Der PDA arbeitet das Zwischenergebnis von links ab:
  - Terminale werden mit der Eingabe verglichen und entfernt;
  - Nichtterminale werden durch Grammatikregel ersetzt.
- Die Ableitung gelingt, wenn die gesamte Eingabe gelesen werden konnte und der Keller leer ist.

## Beispiel

Die Grammatik  $S \rightarrow aSa \mid bSb \mid \epsilon$  ergibt den folgenden PDA:

$\epsilon, S \mapsto aSa$   
 $\epsilon, S \mapsto bSb$   
 $\epsilon, S \mapsto \epsilon$   
 $a, a \mapsto \epsilon$   
 $b, b \mapsto \epsilon$



Mögliche Konfigurationsfolge (Wörter werden zur Vereinfachung in einem Schritt auf Stack geschrieben):

$\langle q_s, abba, \epsilon \rangle \vdash \langle q_l, abba, S\# \rangle \vdash \langle q_l, abba, aSa\# \rangle \vdash \langle q_l, bba, Sa\# \rangle \vdash \langle q_l, bba, bSba\# \rangle \vdash$   
 $\langle q_l, ba, Sba\# \rangle \vdash \langle q_l, ba, ba\# \rangle \vdash \langle q_l, a, a\# \rangle \vdash \langle q_l, \epsilon, \# \rangle \vdash \langle q_f, \epsilon, \epsilon \rangle$

(akzeptierender Lauf)

## Grammatik $\rightsquigarrow$ PDA

Wir wollen zunächst eine Richtung zeigen:

**Satz:**  
Für jede kontextfreie Grammatik  $G$  gibt es einen PDA  $\mathcal{P}_G$ , so dass  $\mathbf{L}(G) = \mathbf{L}(\mathcal{P}_G)$ .

**Beweis:** Sei  $G = \langle V, \Sigma, P, S \rangle$ . Wir definieren einen erweiterten PDA:

$\epsilon, A \mapsto w$  für jede Regel  $A \rightarrow w$   
 $a, a \mapsto \epsilon$  für jedes Symbol  $a \in \Sigma$



Wir verwenden dabei also:

$Q = \{q_s, q_l, q_f\}$      $Q_0 = \{q_s\}$      $F = \{q_f\}$      $\Gamma = \Sigma \cup V \cup \{\#\}$

Durch Entfernen der Wort-push-Operationen wie zuvor gezeigt entsteht der gesuchte PDA  $\mathcal{P}_G$ . □

## Zusammenfassung und Ausblick

**Kellerautomaten** (PDAs) erweitern endliche Automaten um einen unbeschränkt großen Speicher, der aber nur nach dem LIFO-Prinzip verwendet werden kann.

PDAs erkennen genau die kontextfreien Sprachen.

Kontextfreie Grammatiken können durch sehr einfache PDAs (nichtdeterministisch) implementiert werden.

### Offene Fragen:

- Wie geht der Beweis weiter?
- Wie ist es mit deterministischen Kellerautomaten?
- Welche Probleme auf kontextfreien Grammatiken kann man lösen?
- Was gibt es zu Typ 1 und Typ 0 zu sagen?