FORMALE SYSTEME

15. Vorlesung: Kellerautomaten

Markus Krötzsch
Lehrstuhl Wissensbasierte Systeme

TU Dresden, 1. Dezember 2016

(Nicht)Abschlusseigenschaften für Typ 2

Satz: Wenn $\mathbf{L}$, $\mathbf{L_1}$ und $\mathbf{L_2}$ kontextfreie Sprachen sind, dann beschreiben auch die folgenden Ausdrücke kontextfreie Sprachen:

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

Aber:

Satz: Es gibt kontextfreie Sprachen $\mathbf{L}$, $\mathbf{L_1}$ und $\mathbf{L_2}$, so dass die folgenden Ausdrücke keine kontextfreien Sprachen sind:

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

Vorschau

Plan

- Grammatiken
- Reguläre Sprachen
  - DFA und NFA
  - Umformungen, Abschlusseigenschaften, Minimalisierung
  - Pumping Lemma
- Kontextfreie Sprachen
  - Wortproblem (CYK)
  - Pumping Lemma
  - Kellerautomaten
- Typ 1 und Typ 0
  - Turingmaschinen
  - (Un)Entscheidbarkeit
- Aussagenlogik
  - Syntax und Semantik
  - Algorithmen zum logischen Schließen
  - Die Komplexitätsklasse NP

Kellerautomaten

Ein Berechnungsmodell für Typ-2-Sprachen

Eine typische Problemsprache (kontextfrei aber nicht regulär):

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

Warum können endliche Automaten diese Sprache nicht erkennen?

Weil sie die Zahl der gelesenen a nicht speichern können.

$\leadsto$ Wir wollen endlichen Automaten mehr Speicher geben.

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

```
function isAiBi():
 state = "readA"
 anum = 0
 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` (1bit – zwei mögliche Werte)
- Zähler `anum` (beliebig große Zahl!)

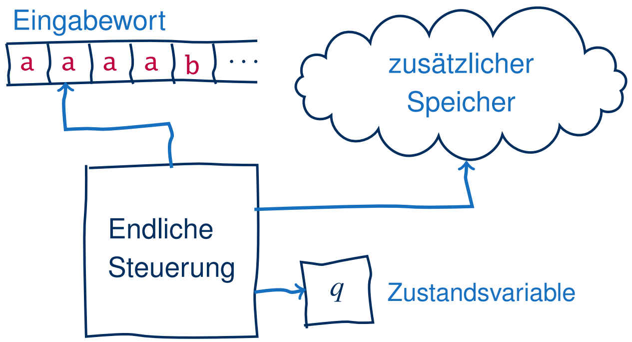
Der Algorithmus benötigt eine unbeschränkte Menge an Speicher, je nach Eingabe

$\leadsto$ 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 Information, die beim Erkennen kontextfreier Sprachen anfallen

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

$$S \to aSa \mid bSb \mid c$$

Die Grammatik generiert zum Beispiel die Wörter bacab und abaabcbaaba.

Diese Sprache ist ebenfalls nicht regulär (Übung).

Wie würde man die Sprache aus diesem Beispiel erkennen?

Pseudoalgorithmus für markierte Palindrome

```
function isMarkedPalindrome():
 state = "start"
 list = new Array()
 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()

$\rightsquigarrow$ Stack (Keller)

Keller sind Stapel

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

Wir denken uns die Liste vertikal, letzte Einträge oben (daher der Name).

Wichtigste Zugriffsoperationen:

- push: ein Element wird oben auf dem Stapel abgelegt
- pop: das oberste Element wird vom Stapel entfernt und zurückgegeben (wenn der Stapel nicht leer ist)

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

Idee: Verwende Keller als zusätzliche Speicher in endlichen Automaten

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, wechsle 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 wechsle in Zustand q_2 und schreibe D auf den Keller."

Lehrvaluation

15min zum Ausfüllen der Fragebögen

Kellerautomaten definieren

Übergangsfunktion von Kellerautomaten:

Eingabe: Zustand + Eingabesymbol + Kellersymbol ($\mathtt{pop}$)

Ausgabe: Zustand + Kellersymbol ($\mathtt{push}$)

Weitere Designentscheidungen:

- Wir unterscheiden mögliche Eingabesymbole Σ von möglichen Kellersymbolen Γ (Gamma)
- Wir wollen erlauben:
  (1) dass der Automat manchmal kein Eingabesymbol liest (ϵ-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
  $\leadsto$ Übergangsfunktion liefert Menge von Möglichkeiten

Kellerautomaten

Zur Vereinfachung der folgenden Definition schreiben wir Σ_ϵ für $\Sigma \cup \{\epsilon\}$ (und analog für Γ_ϵ).

Ein Kellerautomat (international: „PDA"/„Push-Down 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
- Σ: Eingabealphabet
- Γ: Kelleralphabet
- δ: Übergangsfunktion, eine totale Funktion
  $$Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to 2^{Q\times\Gamma_\epsilon},$$
  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)

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

Beispiele:

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

Kellerautomaten: Intuition (2)

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

Beispiele:

- $\langle q_2, \mathsf{D}\rangle \in \delta(q_1, \epsilon, \epsilon)$ bedeutet:
  „Wenn der PDA in Zustand q_1 ist, dann kann er spontan in Zustand q_2 wechseln und dabei D auf den Keller legen (`push`), ohne dabei eine Eingabe oder den Keller zu lesen."
- $\delta(q_1, \mathsf{a}, \mathsf{C}) = \emptyset$ bedeutet:
  „Wenn der PDA in 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

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

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

- $\Sigma = \{\mathsf{a}, \mathsf{b}\}$
- $\Gamma = \{\mathsf{A}, \mathsf{S}\}$
- $Q = \{q_s, q_a, q_b, q_f\}$
- $Q_0 = \{q_s\}$
- $F = \{q_s, q_f\}$

Wir definieren δ wie folgt:

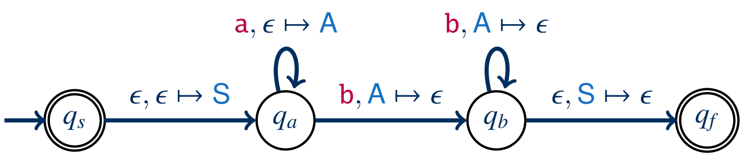
- $\delta(q_s, \epsilon, \epsilon) = \{\langle q_a, \mathsf{S}\rangle\}$ (markiere Kellerboden)
- $\delta(q_a, \mathsf{a}, \epsilon) = \{\langle q_a, \mathsf{A}\rangle\}$ (speichere a)
- $\delta(q_a, \mathsf{b}, \mathsf{A}) = \{\langle q_b, \epsilon\rangle\}$ (erstes b)
- $\delta(q_b, \mathsf{b}, \mathsf{A}) = \{\langle q_b, \epsilon\rangle\}$ (weitere b)
- $\delta(q_b, \epsilon, \mathsf{S}) = \{\langle q_f, \epsilon\rangle\}$ (mögliches Wortende)

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

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

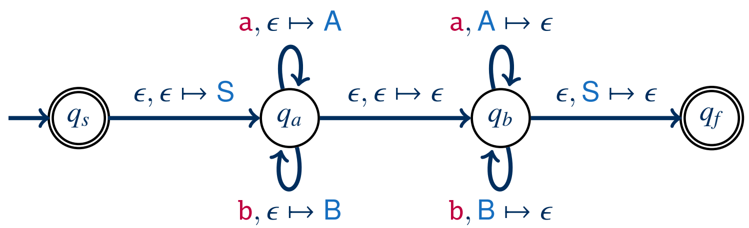


Kodierungstrick: Wir implementieren hier die Funktion `isEmtpy` indem wir den Kelleranfang mit einem speziellen Symbol S markieren, welches den (fast) leeren Keller signalisiert.

Beispiel: Palindrome ohne Markierung

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

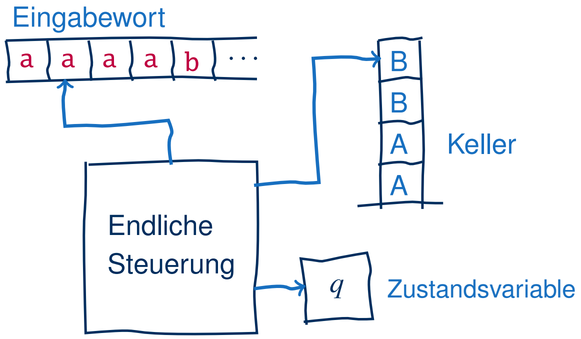
Wir können diese mit folgendem PDA erkennen ($\Gamma = \{\mathsf{A}, \mathsf{B}, \mathsf{S}\}$):



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 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 gilt $\langle q', \psi'\rangle \in \delta(q, v, \psi)$.[a] Mit $\vdash^*$ bezeichnen wir den reflexiven, transitiven Abschluss von $\vdash$.

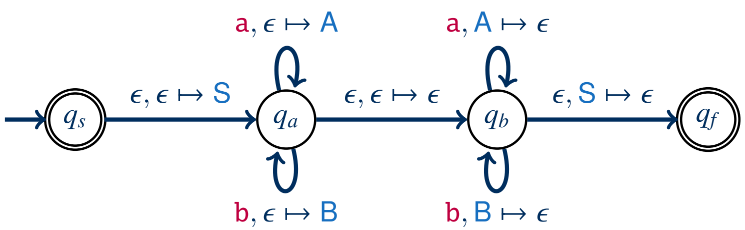
[a] Anmerkung: In diesem Fall ist $v \in \Sigma_\epsilon$ und $\psi, \psi' \in \Gamma_\epsilon$.

Dies entspricht der intuitiven Idee von möglichen Übergängen.

Der PDA $\mathcal{M}$ akzeptiert ein Wort w wenn es Übergänge $\langle q_s, w, \epsilon\rangle \vdash^* \langle q_f, \epsilon, \gamma\rangle$ gibt für ein $q_s \in Q_0$ und $q_f \in F$. Die von $\mathcal{M}$ akzeptierte Sprache $\mathbf{L}(\mathcal{M})$ ist die Menge aller von $\mathcal{M}$ akzeptierten Wörter.

Beispiel

Der Automat



erlaubt z.B. die folgenden Konfigurationsübergänge:

$\langle q_s, \text{abba}, \epsilon\rangle \vdash \langle q_a, \text{abba}, \text{S}\rangle \vdash \langle q_a, \text{bba}, \text{AS}\rangle \vdash \langle q_a, \text{ba}, \text{BAS}\rangle \vdash \langle q_a, \text{a}, \text{BBAS}\rangle \vdash \langle q_a, \epsilon, \text{ABBAS}\rangle$ (kein akzeptierender Lauf)

$\langle q_s, \text{abba}, \epsilon\rangle \vdash \langle q_a, \text{abba}, \text{S}\rangle \vdash \langle q_a, \text{bba}, \text{AS}\rangle \vdash \langle q_a, \text{ba}, \text{BAS}\rangle \vdash \langle q_b, \text{ba}, \text{BAS}\rangle \vdash \langle q_b, \text{a}, \text{AS}\rangle \vdash \langle q_b, \epsilon, \text{S}\rangle \vdash \langle q_f, \epsilon, \epsilon\rangle$ (akzeptierender Lauf)

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 PDA $\rightsquigarrow$ Typ-2-Grammatik

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

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.

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?