Prof. Dr. Roland Meyer
René Maseli

Theoretische Informatik 1
Große Übung 6

TU Braunschweig
Wintersemester 2024/25

Übungsaufgabe 11.6: CYK-Algorithmus

Gegeben die kontextfreie Grammatik G, nutze den CYK-Algorithmus, um zu prüfen, ob *ccdecde* von G erzeugt werden kann.

$$S \to ABC \mid CAB \qquad A \to dC \mid eBC \qquad B \to ED \mid BB \mid EA \qquad C \to c\,.$$

Für den Algorithmus aus der Vorlesung benötigen wir eine Grammatik in Chomsky-Normalform.

$$S \to AF \mid CH \quad A \to DC \mid EF \quad B \to ED \mid BB \mid EA \quad C \to c \quad D \to d \quad E \to e \quad F \to BC \quad H \to AB\,.$$

Die Tabelle wird diagonal-weise gefüllt.

	1	2	3	4	5	6	7
1	E	–	–	A	–	H	AHS
2		E	B	BF	–	B	BF
3			D	A	–	H	HS
4				C	–	–	–
5					E	B	BF
6						D	A
7							C

In der oberen rechten Zelle soll sich genau dann S befinden, wenn das Wort in der Sprache liegt. Hier können wir also $eedcedc \in \mathcal{L}(G') = \mathcal{L}(G)$ schließen.

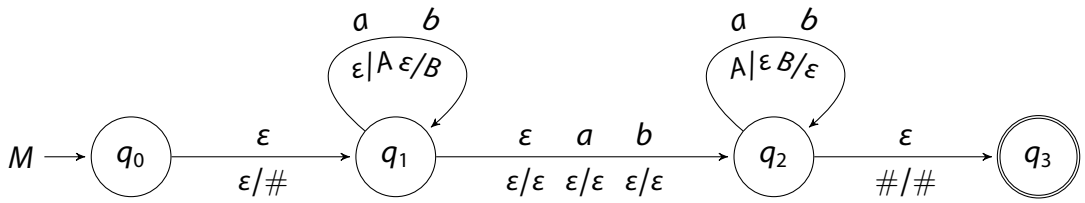
Darüber hinaus lesen wir anhand der Anzahl von S in der ganzen Tabelle, dass 2 nicht-leere Infixe des Wortes in der Sprache liegen: Neben *eedcedc* selbst gibt es noch $dcedc \in \mathcal{L}(G)$. Die oberen Zeile verrät Eigenschaften über Präfixe und die rechteste Spalte steht für die Suffixe des Wortes.

Übungsaufgabe 11.7: Konstruktion von Pushdown-Automaten

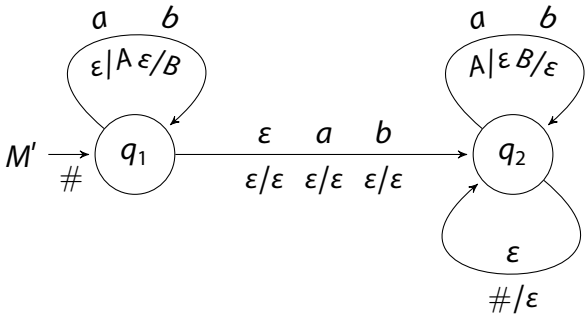
a) Konstruieren Sie einen PDA für die Palindrom-Sprache $\{ w \in \{a, b\}^* \mid w = \text{rev}(w) \}$.

Idee: Der erste Hälfte des Wortes wird in den Stack gepusht und beim Verarbeiten der zweiten Hälfte werden die Symbole abgeglichen und gepopt. Dazu nutzt man Stack-Symbole A und B. Der Automat hat keine Möglichkeit, deterministisch die Mitte des Wortes zu erkennen, also muss er sie während des Runs nicht-deterministisch erraten. Wörter können gerade oder ungerade sein, also darf in der Mitte ein Buchstabe konsumiert werden.

Runs, die falsch raten, dürfen aber nicht akzeptieren. Das könnte sonst bei jedem Wort passieren, wenn die geratene Mitte sich als das letzte Symbol der Eingabe herausstellt. Der Stack muss komplett abgebaut werden, und damit der Automat das feststellen kann, braucht man ein weiteres Stack-Symbol #, das nur einmal und nur am Bottom of Stack vorkommt.

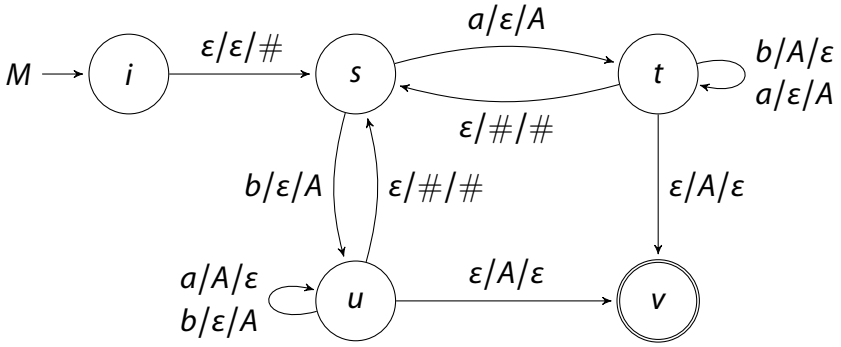


Eine Variante, die mit leerem Stack akzeptiert, statt mit akzeptierenden Zuständen wie oben, kann wiefolgt aussehen:

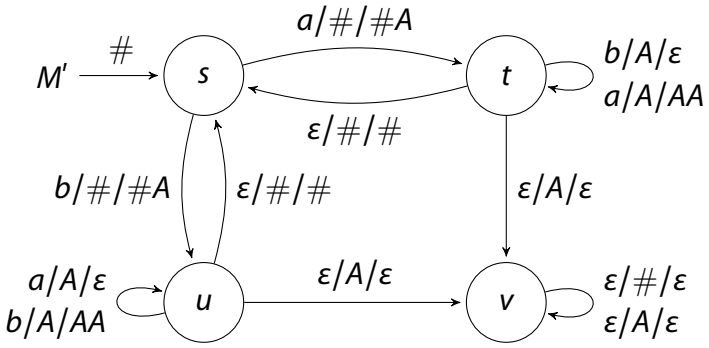


b) Gegeben die Sprache $L = \{ w \in \{a, b\}^* \mid |w|_a \neq |w|_b \}$. Konstruieren Sie einen PDA für L.

Idee: Laufend wird die Differenz $|w|_a - |w|_b$ gezählt. Dabei übernimmt der Stack den absoluten Wert und entsprechende Zustände übernehmen das Vorzeichen. Der PDA akzeptiert in Ziel-Zuständen $Q_F = \{v\}$.



Ein PDA, der mit leerem Stack akzeptiert (und in jeder Transition ein Stacksymbol entfernt), sieht so aus:



Erklärung: Der Automat befindet sich in Zustand t, wenn die Differenz positiv ist. Er ist in u, wenn sie negativ ist. Zustand s steht für neutrale Differenz. s ist gleichzeitig initial und erlaubt spätere Vorzeichenwechel. Akzeptieren kann der Automat nur, indem

Übungsaufgabe 11.8: Tripelkonstruktion

Finden Sie eine kontextfreie Grammatik für L aus Übungsaufgabe 11.7 b).

Wir nutzen M'. Zunächst sollte man versuchen, die Menge produktiver Tripel einzuschränken: Jede Beobachtung erspart uns nachher Regeln, die wir sonst algorithmisch durchlaufen müssten.

1. Zustand s ist nie Ziel einer pop-Transition. xYs ist nie produktiv.
2. Alle ausgehenden Transitionen von s erwarten $\#$ auf dem Top-of-Stack. sYx mit $Y \neq \#$ ist nie produktiv.
3. Stack-Symbol $\#$ wird in jeder Transition weder ersetzt, noch dupliziert. Alle Transitionen, die es poppen, führen nach v. $x\#y$ mit $y \neq v$ ist nie produktiv.
4. Zustand v kann Zustände s, t, u nicht mehr erreichen. vYx mit $x \neq v$ ist nie produktiv.

Nun kann man direkt die Produktionen erzeugen. Wann immer ein Tripel neu entdeckt wird, wird es sofort unterstrichen. Nachträglich unproduktive Tripel wurden eingeklammert. Eine entsprechende Beobachtung wäre

5. Der PDA kann erst dann zwischen t und u wechseln, nachdem alle A's aus dem Stack entfernt wurden. uAt und tAu sind nicht produktiv.

$$
\begin{aligned}
S &\to \underline{\texttt{s\#v}} \\
\texttt{s\#v} &\to a\,\underline{\texttt{tAt}}\,\underline{\texttt{t\#v}} \mid a\,\underline{\texttt{(tAu)}}\,\underline{\texttt{u\#v}} \mid a\,\underline{\texttt{tAv}}\,\underline{\texttt{v\#v}} \mid b\,\underline{\texttt{(uAt)}}\,\texttt{t\#v} \mid b\,\underline{\texttt{uAu}}\,\texttt{u\#v} \mid b\,\underline{\texttt{uAv}}\,\texttt{v\#v} \\
\texttt{tAt} &\to b \mid a\,\texttt{tAt}\,\texttt{tAt} \mid a\,\texttt{tAu}\,\texttt{uAt} \\
\texttt{t\#v} &\to \varepsilon\,\texttt{s\#v} \\
\texttt{(tAu)} &\to a\,\texttt{tAt}\,\texttt{(tAu)} \mid a\,\texttt{(tAu)}\,\texttt{uAu} \\
\texttt{u\#v} &\to \varepsilon\,\texttt{s\#v} \\
\texttt{tAv} &\to \varepsilon \mid a\,\texttt{tAt}\,\texttt{tAv} \mid a\,\texttt{(tAu)}\,\texttt{uAv} \mid a\,\texttt{tAv}\,\underline{\texttt{vAv}} \\
\texttt{v\#v} &\to \varepsilon \\
\texttt{(uAt)} &\to a\,\texttt{(uAt)}\,\texttt{tAt} \mid a\,\texttt{uAu}\,\texttt{(uAt)} \\
\texttt{uAu} &\to a \mid b\,\texttt{(uAt)}\,\texttt{(uAu)} \mid b\,\texttt{uAu}\,\texttt{uAu} \\
\texttt{uAv} &\to \varepsilon \mid b\,\texttt{(uAt)}\,\texttt{tAv} \mid b\,\texttt{uAu}\,\texttt{uAv} \mid b\,\texttt{uAv}\,\texttt{vAv}
\end{aligned}
$$

(hat in der großen Übung gefehlt.)

$$
\texttt{vAv} \to \varepsilon
$$

Addendum: Jeder PDA kann in eine kontextfreie Grammatik überführt werden. Die starken Linksableitungen werden dabei Läufe des Automaten simulieren.

Mit einem Nichtterminal fassen wir einen Teil der PDA-Berechnung zusammen: Nichtterminale haben die Form $\langle p, A, q\rangle \in Q \times \Gamma \times Q$ und beschreiben alle Teil-Läufe, die in p mit A auf dem Stack starten, und irgendwann im Zustand q erstmals das darunterliegende Stack-Symbol freilegen, bzw. den Stack leeren.

In der naiven Grammatik sind üblicherweise die meisten Produktionen nutzlos, also entweder nicht erreichbar oder nicht produktiv. Der folgende Algorithmus berechnet einen nennenswert kleineren erreichbaren Teil:

Require: PDA $M = \langle Q, \Sigma, \Gamma, q_0, \#, \delta\rangle$ (akz. mit leerem Stack)
Ensure: $\mathcal{L}(G) = \mathcal{L}(M)$

$P \leftarrow \varnothing$
$T \leftarrow \{ \langle p, A, q\rangle \mid p \xrightarrow[A/\beta]{\sigma} q' \text{ und } p' \xrightarrow[B/\varepsilon]{s} q \}$ (mit den richtigen Beobachtungen kleiner)
$N \leftarrow \{S\}$
$N_{\text{done}} \leftarrow \{S\}$
for $\langle p, A, q\rangle \in T$ **do**
  P.add($S \to \langle q_0, \#, q\rangle$)
  N.add($\langle q_0, \#, q\rangle$)
end for
while $N \neq N_{\text{done}}$ **do**
  Sei $\langle p, A, q\rangle \in N \setminus N_{\text{done}}$
  for $p \xrightarrow[A/B_n \ldots B_1]{s} q'$ und $\langle p_1, B_1, q_1\rangle \ldots \langle p_n, B_n, q_n\rangle \in T^n$ **do**
    if $p_1 = q'$ und $q_n = q$ und $q_i = p_{i+1}$ für alle $i \in \{1, \ldots, n-1\}$ **then**
      P.add($\langle p, A, q\rangle \to s\langle p_1, B_1, q_1\rangle \ldots \langle p_n, B_n, q_n\rangle$)
      N.addAll($\langle p_1, B_1, q_1\rangle, \ldots, \langle p_n, B_n, q_n\rangle$)
    end if
  end for
  N_{done}.add($\langle p, A, q\rangle$)
end while
return $\langle N, \Sigma, S, P\rangle$

Bemerkung

Achtet darauf, dass die Reihenfolge nach LIFO-Prinzip im Stack verkehrt herum gelesen wird: In $p \xrightarrow[A/B_n \ldots B_1]{s} q'$ wird B_n als erstes gepusht und daher als letztes verarbeitet. B_1 wird als letztes gepusht und muss deshalb zuerst verarbeitet werden.