René Maseli
Thomas Haas
Prof. Dr. Roland Meyer

Theoretische Informatik 1
Große Übung 6

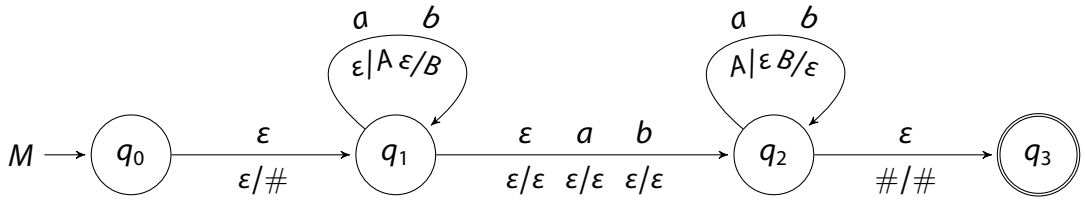
TU Braunschweig
Wintersemester 2022/23

Automaten-Konstruktion

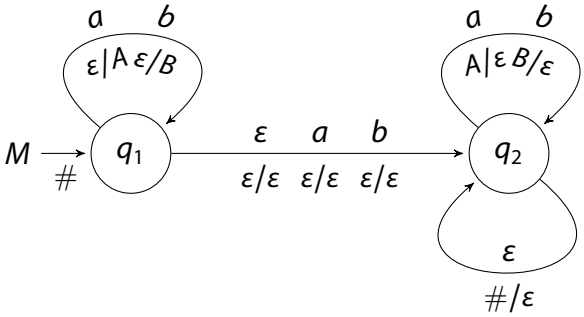
Die Copy-Sprache $L = \{ w \in \{a, b\}^* \mid w = \text{reverse}(w) \}$ ist kontextfrei. Lass uns einen PDA dafür konstruieren.

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:

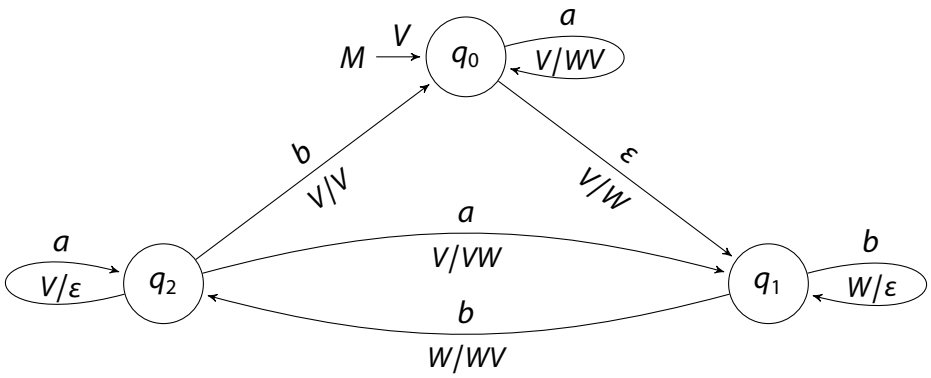


Tripel-Konstruktion

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 den 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 ← ∅
T ← { ⟨p, A, q⟩ | p --σ/A/β--> q' und p' --s/B/ε--> q }  (mit den richtigen Beobachtungen kleiner)
N ← {S}
N_done ← {S}
for ⟨p, A, q⟩ ∈ T do
    P.add(S → ⟨q_0, #, q⟩)
    N.add(⟨q_0, #, q⟩)
end for
while N ≠ N_done do
    Sei ⟨p, A, q⟩ ∈ N \ N_done
    for p --s/A/B_n...B_1--> q' und ⟨p_1, B_1, q_1⟩ ... ⟨p_n, B_n, q_n⟩ ∈ T^n do
        if p_1 = q' und q_n = q und q_i = p_{i+1} für alle i ∈ {1, ..., n − 1} then
            P.add(⟨p, A, q⟩ → s⟨p_1, B_1, q_1⟩ ... ⟨p_n, B_n, q_n⟩)
            N.addAll(⟨p_1, B_1, q_1⟩, ..., ⟨p_n, B_n, q_n⟩)
        end if
    end for
    N_done.add(⟨p, A, q⟩)
end while
return ⟨N, Σ, S, P⟩
```

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.

Auf das Beispiel angewendet, ergibt das die folgende Grammatik: Tripel-Kandidaten T sind, welche auf $q_1\rangle$ und $q_2\rangle$ enden, sowie mit $\langle q_0, V$, $\langle q_1, W$ oder $\langle q_2, V$ anfangen. Also kann q_0 kein Zwischen-Zustand sein.

$$
\begin{array}{rcll}
S & \to & \langle q_0, V, q_1\rangle \mid \langle q_0, V, q_2\rangle & \\
\langle q_0, V, q_1\rangle & \to & a\,\langle q_0, V, q_1\rangle\langle q_1, W, q_1\rangle \mid \langle q_1, W, q_1\rangle & q_2 \text{ kommt hier nicht in Frage} \\
\langle q_0, V, q_2\rangle & \to & a\,\langle q_0, V, q_1\rangle\langle q_1, W, q_2\rangle \mid \langle q_1, W, q_2\rangle & \\
\langle q_1, W, q_1\rangle & \to & b \mid b\,\langle q_2, V, q_1\rangle\langle q_1, W, q_1\rangle & \\
\langle q_1, W, q_2\rangle & \to & b\,\langle q_2, V, q_1\rangle\langle q_1, W, q_2\rangle & \\
\langle q_2, V, q_1\rangle & \to & b\,\langle q_0, V, q_1\rangle \mid a\langle q_1, W, q_2\rangle\langle q_2, V, q_1\rangle & q_1 \text{ kommt hier nicht in Frage}
\end{array}
$$

Es ist gewährleistet, dass alle so berechneten Nichtterminale und Produktionen erreichbar sind. Nachträglich kann man unproduktive Nichtterminale entfernen. Das erledigt der nächste Algorithmus:

Require: Kontextfreie Grammatik $G = \langle N, \Sigma, S, P\rangle$
Ensure: $\mathcal{L}(G') = \mathcal{L}(G)$

```
P' ← ∅
N' ← ∅
while erste Iteration oder N' ≠ N_done do
    N_done ← N'
    for (A → β_1 ... β_n) ∈ P \ P' do
        if β_i ∈ N_done ∪ Σ für alle i ∈ {1, ..., n} then
            P'.add(A → β_1 ... β_n)
            N'.add(A)
        end if
    end for
end while
return ⟨N' ∪ {S}, Σ, S, P'⟩
```

Im Beispiel können wir die Produktionen in dieser Reihenfolge als produktiv markieren:

$$
\begin{array}{rcl}
S\,(2) & \to & \langle q_0, V, q_1\rangle\,(2) \mid \langle q_0, V, q_2\rangle \\
\langle q_0, V, q_1\rangle\,(1) & \to & a\,\langle q_0, V, q_1\rangle\langle q_1, W, q_1\rangle\,(3) \mid \langle q_1, W, q_1\rangle\,(1) \\
\langle q_0, V, q_2\rangle & \to & a\,\langle q_0, V, q_1\rangle\langle q_1, W, q_2\rangle \mid \langle q_1, W, q_2\rangle \\
\langle q_1, W, q_1\rangle\,(0) & \to & b\,(0) \mid b\,\langle q_2, V, q_1\rangle\langle q_1, W, q_1\rangle\,(5) \\
\langle q_1, W, q_2\rangle & \to & b\,\langle q_2, V, q_1\rangle\langle q_1, W, q_2\rangle \\
\langle q_2, V, q_1\rangle\,(3) & \to & b\,\langle q_0, V, q_1\rangle\,(4) \mid a\langle q_1, W, q_2\rangle\langle q_2, V, q_1\rangle
\end{array}
$$

Streicht man den unnützen Rest weg, bleibt

$$
\begin{array}{rcl}
S & \to & \langle q_0, V, q_1\rangle \\
\langle q_0, V, q_1\rangle & \to & a\,\langle q_0, V, q_1\rangle\langle q_1, W, q_1\rangle \mid \langle q_1, W, q_1\rangle \\
\langle q_1, W, q_1\rangle & \to & b \mid b\,\langle q_2, V, q_1\rangle\langle q_1, W, q_1\rangle \\
\langle q_2, V, q_1\rangle & \to & b\,\langle q_0, V, q_1\rangle
\end{array}
$$