Prof. Dr. Jasmin Blanchette
Jannis Limperg
Luca Maio

Ludwig-Maximilians-Universität München
Institut für Informatik
Abgabe bis 8. Mai 2024, 10:00 Uhr

Lösungsvorschlag zur Übung 3 zur Vorlesung
Formale Sprachen und Komplexität

Wenn Sie Automaten angeben, tun Sie dies immer in Form eines Zustandsgraphen. Andere Formen der Darstellung (z.B. als Liste von Übergängen) werden nicht gewertet, da sie sehr viel aufwändiger zu korrigieren sind. Vergessen Sie nicht, im Zustandsgraph Start- und Endzustände zu markieren.

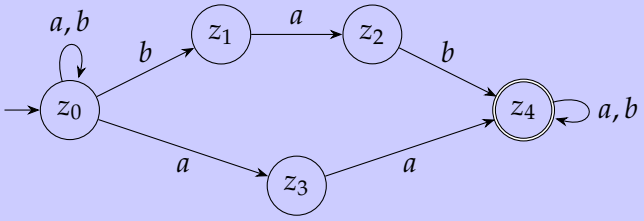
FSK3-1 ***Konstruktion von NFAs*** (2 Punkte)

Verwenden Sie in dieser Aufgabe nur NFAs *ohne* ε-Übergänge.

a) Geben Sie einen NFA an, der die folgende Sprache L über dem Alphabet $\Sigma = \{a, b\}$ akzeptiert:

$$L = \{uvw \mid u, w \in \Sigma^*, v \in \{bab, aa\}\}$$

LÖSUNGSVORSCHLAG:



b) Viele Programmiersprachen erlauben nur Variablennamen, die Regeln wie diese erfüllen:

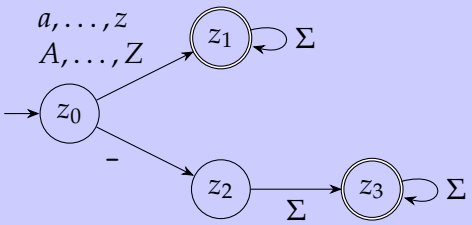
- Ein Variablenname kann Unterstriche, kleine und große Buchstaben (a–z, A–Z) und Ziffern enthalten.
- Ein Variablenname muss mindestens ein Zeichen enthalten.
- Ein Variablenname darf nicht mit einer Ziffer anfangen.
- „_“ ist kein Variablenname.

Geben Sie einen NFA an, der genau die Variablennamen erkennt, die diesen Regeln folgen.

LÖSUNGSVORSCHLAG:

Wir definieren einen NFA über dem Alphabet

$$\Sigma = \{\_, a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$$



c) Sei n eine natürliche Zahl, $\Sigma_n = \{0, \ldots, n\}$ und

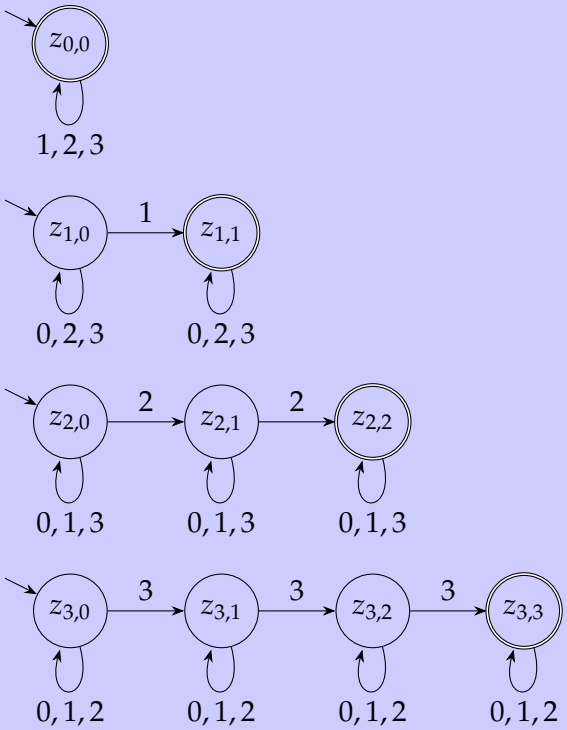
$$L_n = \{w \in \Sigma_n^* \mid \exists i \in \Sigma_n, \#_i(w) = i\}$$

Das heißt, die Sprache L enthält genau die Wörter w, für die gilt: Es gibt eine Zahl $i \in \{0, \ldots, n\}$ sodass das Wort w das Symbol i genau i-mal enthält.

Z.B. ist $2012323 \in L_3$, da dieses Wort genau 1-mal das Symbol 1 enthält. Ebenso ist $20311233 \in L_3$, da dieses Wort genau 2-mal das Symbol 2 enthält. Hingegen ist $0112223 \notin L_3$.

Geben Sie für jedes n einen NFA A_n an, der L_n erkennt. Beschreiben Sie ausnahmsweise A_n nicht durch einen Zustandsgraph, sondern geben Sie die Zustandsmenge, Start- und Endzustände und Übergänge (in Abhängigkeit von n) explizit an. Geben Sie außerdem den Zustandsgraph von A_3 an.

LÖSUNGSVORSCHLAG:
Der folgende Automat A_3 erkennt L_3:

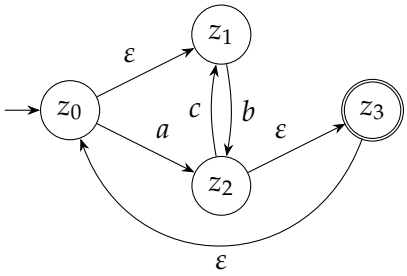


Die Konstruktion für beliebige n verallgemeinert diesen Automaten:

$$
\begin{aligned}
A_n &= (Z_n, \Sigma_n, \delta_n, S_n, E_n) \\
Z_n &= \{z_{i,0}, \ldots, z_{i,i} \mid i \in \Sigma_n\} \\
S_n &= \{z_{i,0} \mid i \in \Sigma_n\} \\
E_n &= \{z_{i,i} \mid i \in \Sigma_n\} \\
\delta_n(z_{i,j}, k) &= \begin{cases} \{z_{i,j+1}\} & \text{für } i \in \Sigma_n,\ k = i,\ j \in \{0, \ldots, i-1\} \\ \{z_{i,j}\} & \text{für } i, j \in \Sigma_n,\ k \in \Sigma_n \setminus \{i\} \\ \emptyset & \text{sonst} \end{cases}
\end{aligned}
$$

FSK3-2 ***Entfernen von ε-Übergängen und Potenzmengenkonstruktion*** (2 Punkte)

a) Sei A_1 der folgende NFA über dem Alphabet $\{a, b, c\}$:



Geben Sie einen NFA A_1' ohne ε-Übergänge mit $L(A_1') = L(A_1)$ an. Verwenden Sie den Algorithmus zum Entfernen von ε-Übergängen aus der Vorlesung. Geben Sie die Zwischenschritte Ihrer Berechnung an. Das erlaubt uns, Ihnen für Folgefehler Teilpunkte zu geben.

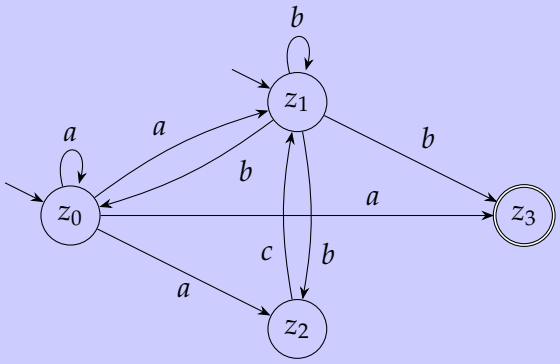
LÖSUNGSVORSCHLAG:
Die Startzustände von A_1' sind die Zustände von A_1, die von z_0 aus nur mit ε-Übergängen erreichbar sind. Diese Bedingung wird (außer von z_0) nur von z_1 erfüllt. Die Endzustände von A_1' sind die von A_1.
Als Übergänge von A_1' verwenden wir zunächst alle Übergänge aus A_1 außer den ε-Übergängen. Weiterhin betrachten wir alle Übergangsfolgen in A_1 von der Form

$$z_{i_1} \xrightarrow{d} z_{i_2} \xrightarrow{\varepsilon} z_{i_3} \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} z_{i_n}$$

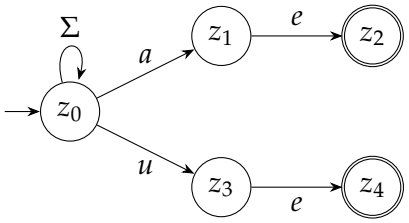
Für jede solche Folge fügen wir die Übergänge $z_{i_1} \xrightarrow{d} z_{i_2}$, $z_{i_1} \xrightarrow{d} z_{i_3}$, …, $z_{i_1} \xrightarrow{d} z_{i_n}$ zu A_1' hinzu. Es ergeben sich folgende Übergänge:

- Aus der Übergangsfolge $z_0 \xrightarrow{a} z_2 \xrightarrow{\varepsilon} z_3 \xrightarrow{\varepsilon} z_0 \xrightarrow{\varepsilon} z_1$ erhalten wir $z_0 \xrightarrow{a} z_3$, $z_0 \xrightarrow{a} z_0$ und $z_0 \xrightarrow{a} z_1$.
- Aus der Übergangsfolge $z_1 \xrightarrow{b} z_2 \xrightarrow{\varepsilon} z_3 \xrightarrow{\varepsilon} z_0 \xrightarrow{\varepsilon} z_1$ erhalten wir $z_1 \xrightarrow{b} z_3$, $z_1 \xrightarrow{b} z_0$ und $z_1 \xrightarrow{b} z_1$.

Insgesamt ist A_1' also:



b) Der folgende NFA A_2 über einem Alphabet $\Sigma \supseteq \{a, e, u\}$ kann verwendet werden, um in einem Text nach den Zeichenfolgen *ae* und *ue* zu suchen.

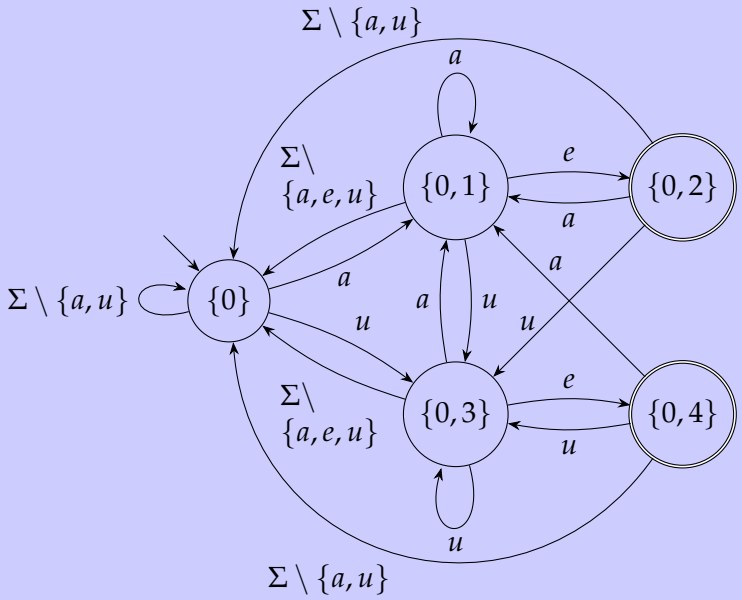


Die Suche wird wesentlich beschleunigt, wenn wir A_2 in einen DFA umwandeln. Verwenden Sie deshalb die Potenzmengenkonstruktion, um einen DFA A_2' mit $L(A_2') = L(A_2)$ zu konstruieren. Geben Sie außer dem Zustandsgraph von A_2' auch die Rechenschritte an, die Sie bei der Potenzmengenkonstruktion ausgeführt haben. Das erlaubt uns, Ihnen bei Folgefehlern noch Teilpunkte zu geben.

LÖSUNGSVORSCHLAG:
Potenzmengenkonstruktion:

Start	$\longrightarrow$	Ziel	Start	$\longrightarrow$	Ziel
$\{0\}$	a	$\{0,1\}$	$\{0,2\}$	a	$\{0,1\}$
$\{0\}$	u	$\{0,3\}$	$\{0,2\}$	u	$\{0,3\}$
$\{0\}$	$\Sigma \setminus \{a,u\}$	$\{0\}$	$\{0,2\}$	$\Sigma \setminus \{a,u\}$	$\{0\}$
$\{0,1\}$	a	$\{0,1\}$	$\{0,4\}$	a	$\{0,1\}$
$\{0,1\}$	u	$\{0,3\}$	$\{0,4\}$	u	$\{0,3\}$
$\{0,1\}$	e	$\{0,2\}$	$\{0,4\}$	$\Sigma \setminus \{a,u\}$	$\{0\}$
$\{0,1\}$	$\Sigma \setminus \{a,e,u\}$	$\{0\}$			
$\{0,3\}$	a	$\{0,1\}$			
$\{0,3\}$	u	$\{0,3\}$			
$\{0,3\}$	e	$\{0,4\}$			
$\{0,3\}$	$\Sigma \setminus \{a,e,u\}$	$\{0\}$			

Die Tabelle enthält die Zustände und Übergänge von A_2'. Startzustand ist $\{0\}$. Endzustände sind die Zustände, die 2 und 4 enthalten (da q_2 und q_4 im ursprünglichen Automaten Endzustände waren), also $\{0,2\}$ und $\{0,4\}$. Daraus ergibt sich der Zustandsgraph von A_2':



Nebenbei: A_2 erkennt nicht die Sprache der Wörter, die *ae* oder *ue* enthalten, sondern die Sprache der Wörter, die mit *ae* oder *ue* enden. Wir können A_2 aber trotzdem verwenden, um Wörter zu erkennen, die *ae* oder *ue* enthalten, indem wir A_2 auf einem gegebenen Wort ausführen und bestimmen, ob A_2 jemals in einen Endzustand kommt.

FSK3-3 ***Tokenizer*** (0 Punkte)

Ein Einsatzgebiet für endliche Automaten sind Tokenizer. Diese werden verwendet, um den Quelltext einer Programmiersprache in syntaktische Einheiten (Tokens) zu zerlegen. Ein Token ist beispielsweise ein Schlüsselwort, ein Bezeichner oder ein Operator.

Zum Beispiel wird das Programm

```
if(x==y){z=x};
```

zerlegt in

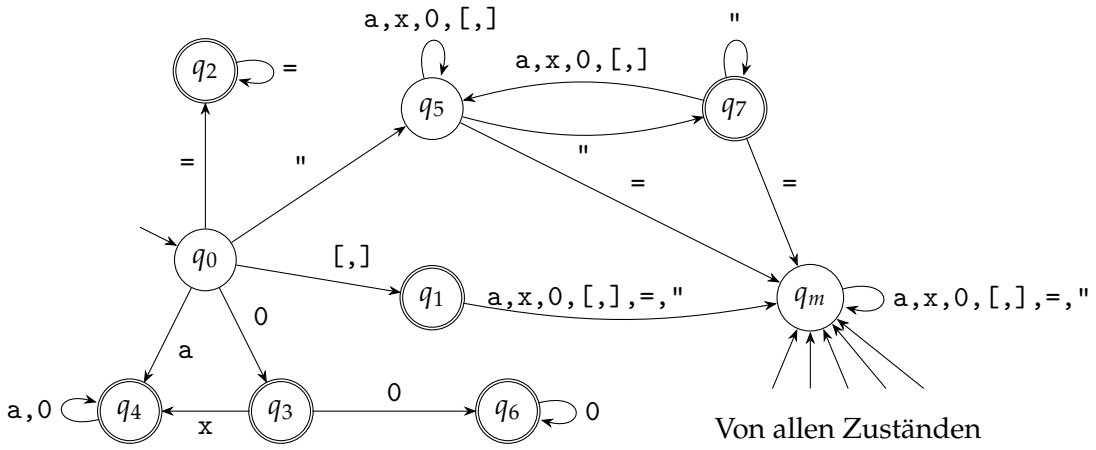
„`if`“ „`(`“ „`x`“ „`==`“ „`y`“ „`)`“ „`{`“ „`z`“ „`=`“ „`x`“ „`}`“ „`;`“

In dieser Aufgabe erstellen wir einen Tokenizer, indem wir die möglichen Tokens als reguläre Sprache auffassen.

a) Um alle Schritte sinnvoll per Hand rechnen zu können, arbeiten wir mit einem reduzierten Alphabet (`[]` statt `()` oder {}, weniger Buchstaben aus dem Alphabet,

nur eine Ziffer, ...):
$\Sigma = \{\texttt{a},\texttt{x},\texttt{0},\texttt{[},\texttt{]},\texttt{=},\texttt{"}\}$

Die Sprache der möglichen Tokens ist als DFA A gegeben:



Um das erste Token aus einem String zu identifizieren, wird A vom Anfang des Strings aus laufen gelassen. Wenn der Lauf nie in einen Endzustand kommt, meldet der Tokenizer einen Fehler. Ansonsten wird die *letzte* Position, in welcher der Automat in einem Endzustand war, als Token-Ende genommen.

Zum Beispiel ist bei Eingabe `==aa[` der Lauf $q_0 \xrightarrow{=} q_2 \xrightarrow{=} q_2 \xrightarrow{a} q_m \xrightarrow{a} q_m \xrightarrow{[} q_m$. Da q_2 akzeptierend ist (aber q_m nicht), ist das erkannte Token `==`.

Notieren Sie bei folgenden Strings die Zustände, die A bei Verarbeitung dieser Strings annehmen wird (die Läufe) und geben Sie je die Ausgabe des Tokenizers an. Bezüglich der Ausgabe reicht es, sofern der Tokenizer keinen Fehler zurückgibt, nur das erste erkannte Token anzugeben.

- `aa==a`
- `a[0]`
- `a[[[[]`
- `"a[0]"ax"`
- `"a=[0]"ax"`
- `"a[0]"a=x"`

LÖSUNGSVORSCHLAG:
`aa==a`: $\Rightarrow q_0 \xrightarrow{a} q_4 \xrightarrow{a} q_4 \xrightarrow{=} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m \Rightarrow$ Token: `aa`
`a[0]`: $\Rightarrow q_0 \xrightarrow{a} q_4 \xrightarrow{[} q_m \xrightarrow{0} q_m \xrightarrow{]} q_m \Rightarrow$ Token: `a`
`a[[[[]`: $\Rightarrow q_0 \xrightarrow{a} q_4 \xrightarrow{[} q_m \xrightarrow{[} q_m \xrightarrow{[} q_m \xrightarrow{[} q_m \xrightarrow{]} q_m \Rightarrow$ Token: `a`

`"a[0]"ax`: $\Rightarrow q_0 \xrightarrow{"} q_5 \xrightarrow{a} q_5 \xrightarrow{[} q_5 \xrightarrow{0} q_5 \xrightarrow{]} q_5 \xrightarrow{"} q_7 \xrightarrow{a} q_5 \xrightarrow{x} q_5 \xrightarrow{"} q_7$
$\Rightarrow$ Token: `"a[0]"ax`
`"a=[0]"ax`: $\Rightarrow q_0 \xrightarrow{"} q_5 \xrightarrow{a} q_5 \xrightarrow{=} q_m \xrightarrow{[} q_m \xrightarrow{0} q_m \xrightarrow{]} q_m \xrightarrow{"} q_m \xrightarrow{a} q_m \xrightarrow{x} q_m \xrightarrow{"} q_m$
$\Rightarrow$ Fehler
`"a[0]"a=x`: $\Rightarrow q_0 \xrightarrow{"} q_5 \xrightarrow{a} q_5 \xrightarrow{[} q_5 \xrightarrow{0} q_5 \xrightarrow{]} q_5 \xrightarrow{"} q_7 \xrightarrow{a} q_5 \xrightarrow{=} q_m \xrightarrow{x} q_m \xrightarrow{"} q_m$
$\Rightarrow$ Token: `"a[0]"`

b) Bestimmen Sie asymptotisch (in O-Notation), wie viele Schritte der Tokenizer-Automat braucht, um ein Token aus einem String der Länge n zu extrahieren.

LÖSUNGSVORSCHLAG: $O(n)$, da die Verarbeitung von einem Zeichen $O(1)$ ist und zweimal über den String gelaufen werden muss: Einmal zur Verarbeitung des Strings und einmal bei der Suche nach dem letzten Zustand, der ein Endzustand ist. (Man kann sich den jeweils letzten Endzustand natürlich auch merken, dann muss man nur einmal über den String laufen. Das ändert aber an der asymptotischen Laufzeit nichts.)

c) Um mehrere Tokens zu extrahieren, wird das gefundene Token von dem String entfernt und wieder von vorne ein Token gesucht. Wenn der verbleibende String leer ist, ist der Tokenizer fertig.

Beispiel: Bei der oben genannten Eingabe `==aa[` mit dem ersten Token `==`, ist der Reststring nach dem Entfernen `aa[`, das zweite Token dann also `aa`.

Zerlegen Sie mit diesem Algorithmus den String `a="ax0"aa[0]=a` in alle Tokens.

LÖSUNGSVORSCHLAG:
String: `a="ax0"aa[0]=a`
Automat $q_0 \xrightarrow{a} q_4 \xrightarrow{=} q_m \xrightarrow{"} q_m \xrightarrow{a} q_m \xrightarrow{x} q_m \xrightarrow{0} q_m \xrightarrow{"} q_m \xrightarrow{a} q_m \xrightarrow{a} q_m \xrightarrow{[} q_m \xrightarrow{0} q_m \xrightarrow{]} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `a` Reststring: `="ax0"aa[0]=a`
Automat $q_0 \xrightarrow{=} q_2 \xrightarrow{"} q_m \xrightarrow{a} q_m \xrightarrow{x} q_m \xrightarrow{0} q_m \xrightarrow{"} q_m \xrightarrow{a} q_m \xrightarrow{a} q_m \xrightarrow{[} q_m \xrightarrow{0} q_m \xrightarrow{]} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `=` Reststring: `"ax0"aa[0]=a`
Automat $q_0 \xrightarrow{"} q_5 \xrightarrow{a} q_5 \xrightarrow{x} q_5 \xrightarrow{0} q_5 \xrightarrow{"} q_7 \xrightarrow{a} q_5 \xrightarrow{a} q_5 \xrightarrow{[} q_5 \xrightarrow{0} q_5 \xrightarrow{]} q_5 \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `"ax0"` Reststring: `aa[0]=a`

Automat $q_0 \xrightarrow{a} q_4 \xrightarrow{a} q_4 \xrightarrow{[} q_m \xrightarrow{0} q_m \xrightarrow{]} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `aa` Reststring: `[0]=a`
Automat $q_0 \xrightarrow{[} q_1 \xrightarrow{0} q_m \xrightarrow{]} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `[` Reststring: `0]=a`
Automat $q_0 \xrightarrow{0} q_3 \xrightarrow{]} q_m \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `0` Reststring: `]=a`
Automat $q_0 \xrightarrow{]} q_1 \xrightarrow{=} q_m \xrightarrow{a} q_m$
Token: `]` Reststring: `=a`
Automat $q_0 \xrightarrow{=} q_2 \xrightarrow{a} q_m$
Token: `=` Reststring: `a`
Automat $q_0 \xrightarrow{a} q_4$
Token: `a` Reststring: ε
Damit ist die Liste der Tokens: „`a`“, „`=`“, „`"ax0"`“, „`aa`“, „`[`“, „`0`“, „`]`“, „`=`“, „`a`“

d) Tatsächlich müssen wir den String nicht verändern, sondern, wenn ein Token gefunden wurde, nur den Automat an der nächsten Position im String starten. Wir „kürzen“ den String also in $O(1)$.

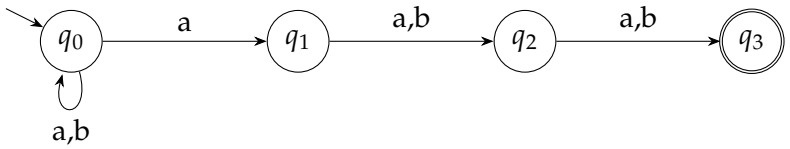
Wie viele Schritte brauchen wir dann asymptotisch, um alle Tokens aus einem String der Länge n zu finden? (Hinweis: Es ist nicht $O(n)$. Man könnte das Verfahren aber optimieren, um eine Laufzeit von $O(n)$ zu erreichen.)

LÖSUNGSVORSCHLAG: $O(n^2)$, da es bis zu n Tokens geben kann und jedes in $O(n)$ Schritten gefunden wird.

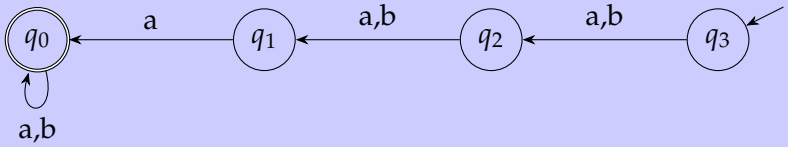
FSK3-4 ***Umgedrehte Sprache*** (0 Punkte)

Sei T die Funktion, die aus einem NFA $A = (Z, \Sigma, \delta, S, E)$ einen NFA $T(A) = (Z, \Sigma, \delta', E, S)$ erzeugt, wobei $p \in \delta'(q, a) \iff q \in \delta(p, a)$.

a) Berechnen Sie den Automaten $B = T(A)$ für folgenden Automaten A:

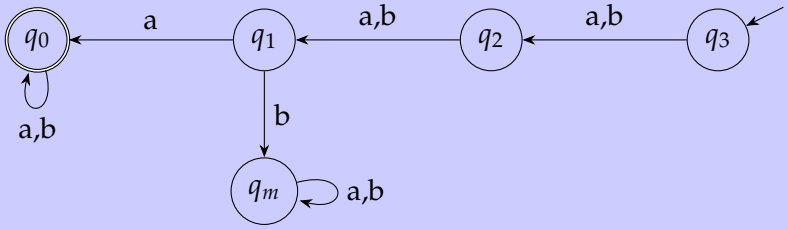


LÖSUNGSVORSCHLAG:



b) Geben Sie einen DFA C mit $L(B) = L(C)$ an. (Sie dürfen die Potenzmengenkonstruktion nutzen, müssen aber nicht.)

LÖSUNGSVORSCHLAG:
Fast wie in b), aber q_1 benötigt noch einen ausgehenden Übergang mit b. Darum Müllzustand hinzufügen.



c) Zeigen Sie: Für jeden NFA A ist $L(T(A)) = \{w \mid \overline{w} \in L(A)\}$. Dabei steht $\overline{w}$ wie in der Vorlesung für das rückwärts gelesene Wort w.

LÖSUNGSVORSCHLAG:
Wir zeigen für alle $q, p \in Z$ und alle $w \in \Sigma^*$:

$$q \in \widetilde{\delta}(p, \overline{w}) \iff p \in \widetilde{\delta'}(q, w) \tag{1}$$

durch Induktion über die Länge $|w|$.
Zur Erinnerung: Für $X \subseteq Z$, $a \in \Sigma$, $w \in \Sigma^*$ und $z \in Z$ gilt

$$\widetilde{\delta}(X, \varepsilon) = X, \quad \widetilde{\delta}(X, aw) = \widetilde{\delta}(\bigcup_{z \in X} \delta(z, a), w), \quad \widetilde{\delta}(z, w) := \widetilde{\delta}(\{z\}, w)$$

Basis: $w = \varepsilon$**:** Die Definition von $\widetilde{\delta}$ liefert sofort $\widetilde{\delta}(q, \varepsilon) = \{q\} = \widetilde{\delta'}(q, \varepsilon)$

Schritt: $a_1 \cdots a_n \to a_1 \cdots a_{n+1}$**:**

$$\begin{aligned}
& q \in \widetilde{\delta}(p, a_{n+1} \cdots a_1) \\
\text{g.d.w. } & \exists z \in Z : z \in \delta(p, a_{n+1}) \wedge q \in \widetilde{\delta}(z, a_n \cdots a_1) && \text{(Definition von } \widetilde{\delta}) \\
\text{g.d.w. } & \exists z \in Z : z \in \delta(p, a_{n+1}) \wedge z \in \widetilde{\delta'}(q, a_1 \cdots a_n) && \text{(mit IH)} \\
\text{g.d.w. } & \exists z \in Z : p \in \delta'(z, a_{n+1}) \wedge z \in \widetilde{\delta'}(q, a_1 \cdots a_n) && \text{(Definition von } \delta') \\
\text{g.d.w. } & p \in \widetilde{\delta'}(q, a_1 \cdots a_{n+1}) && \text{(Definition von } \widetilde{\delta'})
\end{aligned}$$

Schließlich zeigen wir die Behauptung:

$$
\begin{array}{ll}
 & \overline{w} \in L(A) \\
\text{g.d.w.} & \exists q \in E, p \in S : q \in \widetilde{\delta}(p, \overline{w}) \\
\text{g.d.w.} & \exists q \in E, p \in S : p \in \widetilde{\delta'}(q, w) \quad \text{(mit Gleichung (1))} \\
\text{g.d.w.} & w \in L(T(A))
\end{array}
$$