PD Dr. Jan Johannsen
Elisabeth Lempa
Luca Maio

Ludwig-Maximilians-Universität München
Institut für Informatik
Besprechung 07.05.2026 bis 11.05.2026
Abgabe bis 19.05.2026, 14:00 Uhr

Übung 3 zur Vorlesung

Formale Sprachen und Komplexität

Hinweis:
Die letzte Aufgabe auf diesem Blatt ist eine Aufgabe zur Klausurvorbereitung. Diese Aufgabe orientiert sich in Form und inhaltlichen Schwerpunkten an den Klausuraufgaben. Dies bedeutet jedoch nicht, dass die anderen Aufgaben nicht klausurrelevant sind.

Die Lösungen der Klausurvorbereitungs-Aufgaben werden am Ende der Beabeitungszeit gesondert veröffentlicht, aber **nicht** im Tutorium besprochen. Die Lösungen der anderen Aufgaben werden bereits zu Beginn der Bearbeitungszeit veröffentlicht und können Ihnen bei der Bearbeitung helfen.

Wenn Sie Ihre Lösung innerhalb der Bearbeitungszeit über Moodle abgeben, erhalten Sie eine individuelle Korrektur. Die Abgabe ist freiwillig (aber nachdrücklich empfohlen).

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*

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

a) 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.

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

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

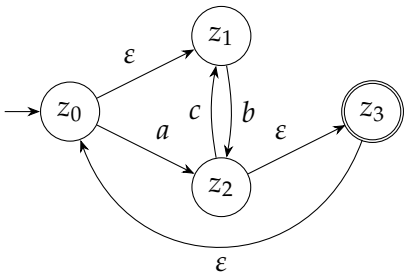
Das heißt, die Sprache L enthält genau die Wörter w, für die gilt: Es gibt eine Zahl $i \in \{0, \ldots, n\}$ sodass das Wort w das Symbol i genau i-mal enthält.

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 die Übergangsfunktion (in Abhängigkeit von n) explizit an. Geben Sie außerdem den Zustandsgraph von A_3 an.

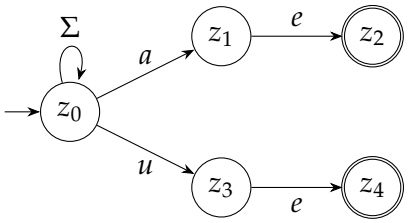
FSK3-2 ***Entfernen von ε-Übergängen und Potenzmengenkonstruktion***

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.

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.

FSK3-3 ***Tokenizer***

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};`

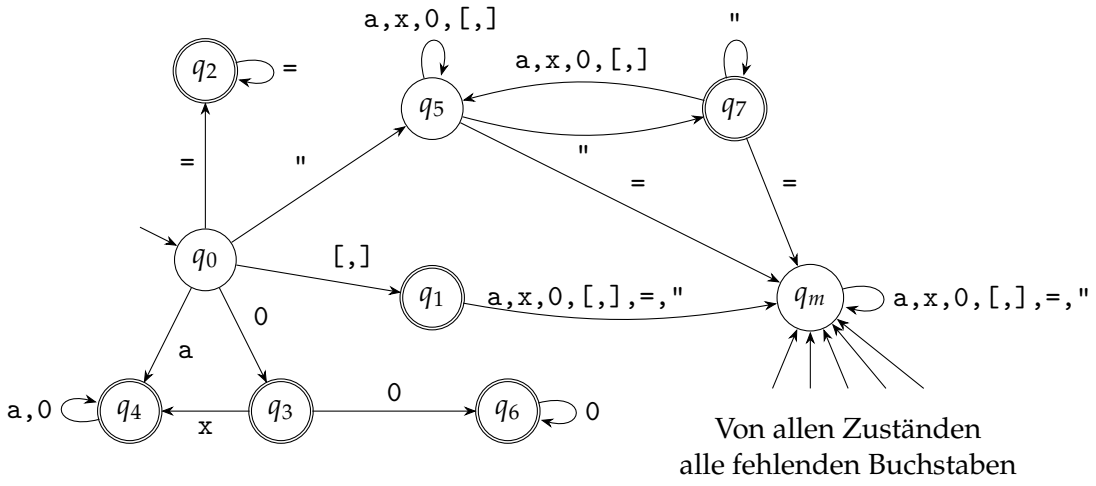
zerlegt in

„`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, ...):
$\Sigma = \{$`a,x,0,[,],=,"`$\}$

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"`

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.

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.

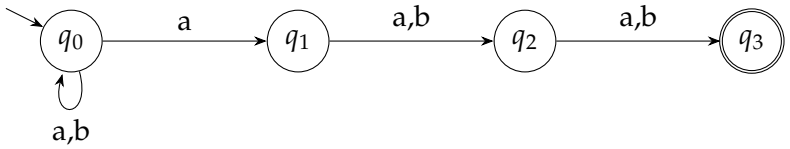
d) Tatsächlich müssen wir den String nicht verändern, sondern, wenn ein Token gefunden wurde, nur den Automat an der nächsten Position im String starten. Wir „kürzen“ den String also in $O(1)$.

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.)

FSK3-4 *Umgedrehte Sprache*

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:



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

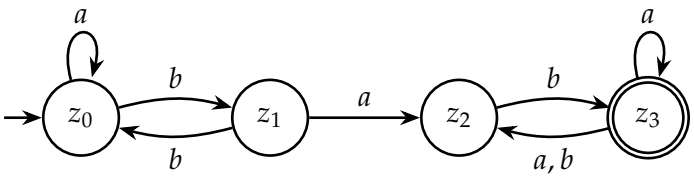
c) Zeigen Sie: Für jeden NFA A ist $L(T(A)) = \{w \mid \overline{w} \in L(A)\}$. Dabei steht $\overline{w}$ wie in der Vorlesung für das rückwärts gelesene Wort w.

Klausurvorbereitung FSK-3-K

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

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

b) Für diese Aufgabe betrachten wir folgenden NFA C über dem Alphabet $\{a, b\}$:



Berechnen Sie zum NFA C einen äquivalenten DFA D mit der Potenzmengenkonstruktion. Geben Sie nur den Zustandsgraphen des vom Startzustand erreichbaren Teils des Automaten an. Vergessen Sie nicht, den Startzustand und die Endzustände zu markieren.