Prof. Dr. Jasmin Blanchette
Jannis Limperg
Luca Maio

Ludwig-Maximilians-Universität München
Institut für Informatik
Abgabe bis 12. Juni 2024, 10:00 Uhr

Lösungsvorschlag zur Übung 7 zur Vorlesung

Theoretische Informatik für Studierende der Medieninformatik

TIMI7-1 ***Sprachen einordnen*** (Punkte)

Die formalen Sprachen L_i, $i \in \{0, \ldots, 2\}$, seien definiert als

$$\begin{aligned} L_0 &:= \{ab^i \mid i \in \mathbb{N}\} \cup \{c^i a \mid i \in \mathbb{N}\} && \subseteq \{a, b, c\}^* \\ L_1 &:= \{w\$ \mid \#_a(w) < \#_b(w) + \#_c(w)\} && \subseteq \{a, b, c, \$\}^* \\ L_2 &:= \{(ab)^j \$ c^i \mid i, j \in \mathbb{N}\} && \subseteq \{a, b, c, \$\}^* \end{aligned}$$

Für die i-fache Wiederholung des Worts w schreiben wir manchmal $(w)^i$ statt nur w^i, um Anfang und Ende von w zu markieren. Die Klammern sind daher *nicht* Teil des Alphabets der jeweiligen Sprachen.

Bearbeiten Sie die folgenden Arbeitsaufträge für jede der Sprachen L_i.

a) Beweisen oder widerlegen Sie, dass L_i regulär ist.

b) Falls L_i deterministisch kontextfrei ist, beweisen Sie dies. Falls nicht, begründen Sie, warum das so ist.

c) Falls L_i kontextfrei ist, beweisen Sie dies. Falls nicht, begründen Sie, warum das so ist.

Hinweis: Nutzen Sie, dass manche Aussagen direkt aus anderen Aussagen folgen. Um zu beweisen, dass L_i regulär/deterministisch kontextfrei/kontextfrei ist, genügt es, ein geeignetes Konstrukt K_i (Grammatik, Automat oder regulärer Ausdruck) anzugeben und kurz zu begründen, warum $L(K_i) = L_i$ gilt.

LÖSUNGSVORSCHLAG:

- $L_0 = \{ab^i \mid i \in \mathbb{N}\} \cup \{c^i a \mid i \in \mathbb{N}\} = L(ab^*) \cup L(c^*a)$ ist regulär (und damit auch kontextfrei und deterministisch kontextfrei), da die Sprache von einer Vereinigung regulärer Ausdrücke erzeugt wird. Jedes Wort der Sprache ist entweder aus ab^* oder aus c^*a, da beide i's unabhängig voneinander sind. Wir nutzen hierbei, dass reguläre Sprachen unter Vereinigung abgeschlossen sind.
- Gemeint (aber nicht klar spezifiziert) war hier, dass w nur a, b und c enthält, d.h. $L_1 = \{w\$ \mid w \in \{a, b, c\}^* \text{ und } \#_a(w) < \#_b(w) + \#_c(w)\}$.

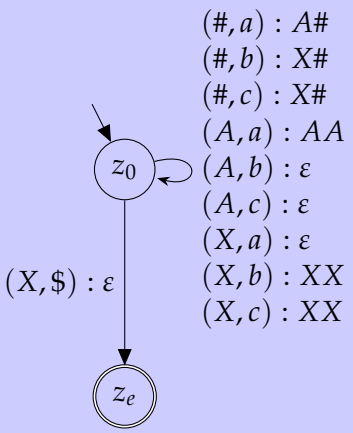
L_1 ist nicht regulär. Ein Beweis mit dem Pumping-Lemma für reguläre Sprachen folgt:

Der Beweis ist durch Widerspruch. Wir nehmen an, L_1 ist regulär. D.h. die Pumping-Eigenschaft gilt für L_1.

Gegeben sei $n \in \mathbb{N}_{>0}$. Wir wählen das Wort $z = a^n b^n c^n \$ \in L_1$ mit $|z| \geq n$.

Betrachte nun Zerlegungen $z = uvw$ mit $|uv| \leq n$, $|v| > 0$ und $uv^i w \in L_1$ für jedes $i \in \mathbb{N}$. Wir zeigen, dass es stets ein $i \in \mathbb{N}$ gibt, sodass $uv^i w \notin L_1$ ist. Aus den ersten zwei Eigenschaften der Zerlegungen folgt, dass jede Zerlegung die Form $u = a^j, v = a^k, w = a^l b^n c^n \$$ hat mit $j + k + l = n$, $j + k \leq n$ und $k > 0$. Dies bedeutet, v enthält immer mindestens ein a. Wählen wir nun $i = 2n = \#_b(z) + \#_c(z)$. Dann gilt $\#_a(uv^i w) = j + 2nk + l \geq 2n = \#_b(z) + \#_c(z)$, da $k > 0$ und somit $uv^i w \notin L_1$. Dies verletzt aber die Annahme, dass $uv^i w \in L_1$ für jedes i. Widerspruch.

L_1 ist deterministisch kontextfrei (und damit auch kontextfrei), da der folgende DPDA M mit Startsymbol # die Sprache L_1 erkennt:



M speichert die aktuelle Balance von a zu b und c, indem es für jedes a entweder ein A hinzufügt oder ein X löscht, und umgekehrt für jedes b oder c ein X hinzufügt oder ein A löscht. Wenn am Ende ein X als oberstes Symbol im Keller ist können wir das Wort akzeptieren, da ein Überschuss an b und c vorhanden gewesen sein muss.

- $L_2 = \{(ab)^j \$ c^i \mid i, j \in \mathbb{N}\} = L((ab)^* \$ c^*)$ ist regulär (und damit auch deterministisch kontextfrei und kontextfrei), da die Sprache von einem regulären Ausdruck erzeugt wird. Jedes Wort der Sprache besteht erst aus beliebig vielen ab's gefolgt von \$ und endet mit beliebig vielen c.

TIMI7-2 ***Turingmaschinen erstellen*** (2 Punkte)

Wir betrachten die Sprache $L = \{w \mid \#_b(w) > \#_c(w)\}$ über dem Alphabet $\{b, c\}$.

a) Erstellen Sie auf `https://turingmachinesimulator.com/` eine TM, die L erkennt. Geben Sie sowohl einen Link zur Maschine an als auch den „Programmtext“ der Maschine.

LÖSUNGSVORSCHLAG:
TM: `http://turingmachinesimulator.com/shared/ndqrvoyaon`
Idee: Wir gehen von links nach rechts über das Wort. Für jedes b, das wir dabei finden, suchen wir nach rechts ein entsprechendes c, und für jedes c suchen wir ein b. Die von links abgearbeiteten b's und c's ersetzen wir durch X und markieren so den Teil des Wortes, den wir nicht mehr besuchen müssen. Die entsprechenden c's und b's ersetzen wir durch C's und B's, um zu markieren, dass sie bereits verwendet wurden. Wenn wir nach einem „Partner“ zu einem b suchen, aber keinen finden, also $\square$ lesen, akzeptieren wir das Wort, da es somit mindestens ein mehr b's als c's geben muss.
Genauer führen wir folgende Schritte in einer Schleife aus:

i) (z_0, Startzustand) Laufe nach rechts über das Wort und halte nach dem ersten b oder c an. Das gefundene b oder c wird dabei durch X ersetzt. Ist das erste Zeichen bereits ein b oder c, halte auf dem zweiten Zeichen.

ii) (z_1) Ist das gefundene Zeichen ein b, laufe nach rechts und ersetze das erste c durch ein C.

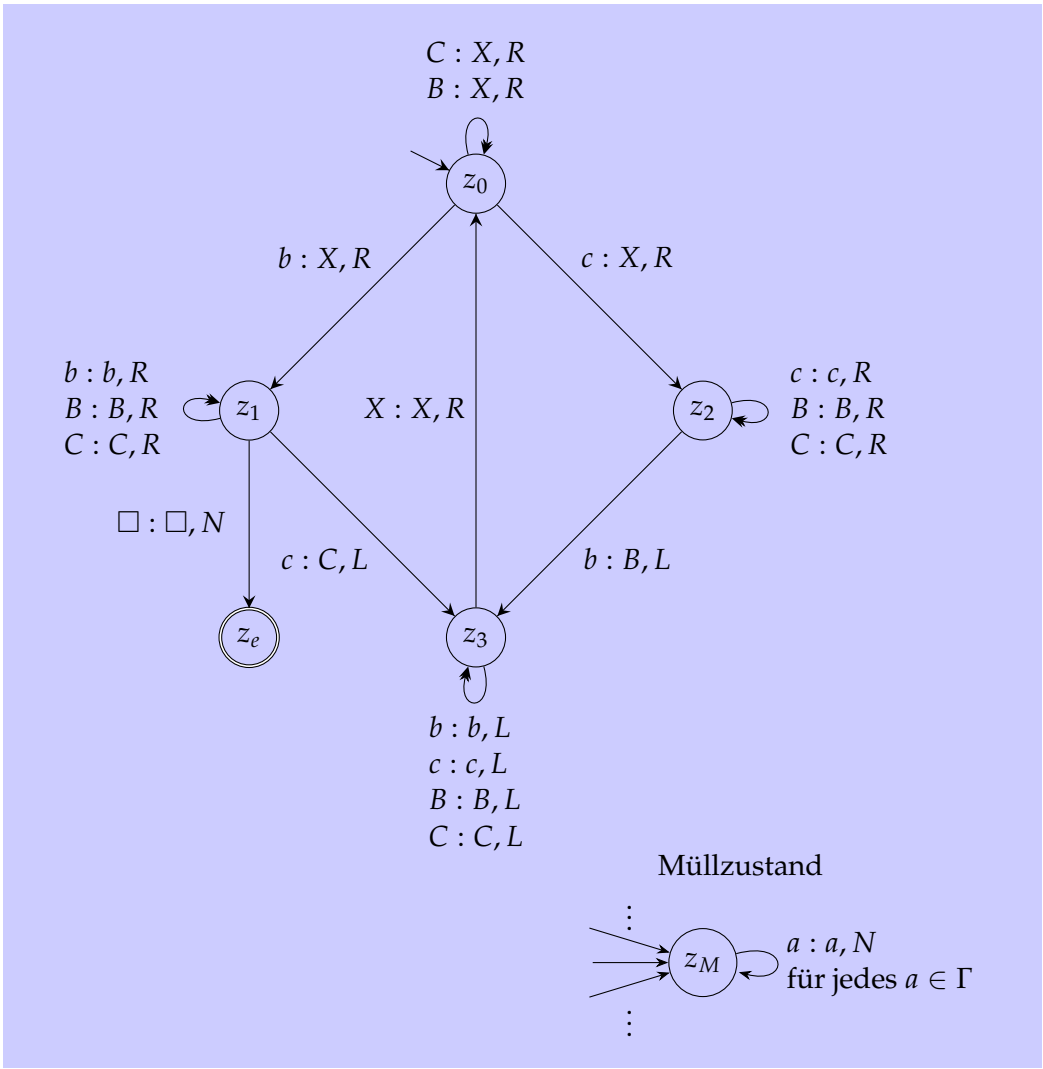
iii) (z_2) Ist das gefundene Zeichen ein c, suche entsprechend nach dem ersten b und ersetze es durch ein B.

iv) (z_3) Laufe nach links über das Wort und halte rechts des ersten X.

Wir akzeptieren, wenn wir in z_1 $\square$ lesen, es also kein c zum bereits gelesenen b gibt. Als Optimierung ersetzen wir im ersten Schritt B und C durch X und verkürzen so den Teil des Wortes, den wir in den nächsten Durchläufen betrachten müssen. Das ist aber nicht nötig.

b) Geben Sie für Ihre TM aus Teilaufgabe a) einen Zustandsgraphen an.

LÖSUNGSVORSCHLAG:



c) Geben Sie die Läufe der folgenden Wörter auf Ihre TM aus Teilaufgabe a) an: ε, c, bcc, $cbcbb$.

Hinweis: Wörter, die nicht in L liegen, erzeugen eventuell unendliche Läufe. Geben Sie in solchen Fällen ein Präfix an, aus dem ersichtlich wird, dass der Lauf unendlich ist.

LÖSUNGSVORSCHLAG:
Für ε:

$$z_0\square \vdash z_M\square \vdash z_M\square \vdash \dots$$

Für c:

$$z_0c \vdash Xz_2\square \vdash Xz_M\square \vdash Xz_M\square \vdash \dots$$

Für *bcc*:

$$z_0bcc \vdash Xz_1cc \vdash z_3XCc \vdash Xz_0Cc \vdash XXz_0c \vdash XXXz_2\square \vdash XXXz_M\square \vdash XXXz_M\square \vdash \ldots$$

Für *cbcbb*:

$$z_0cbcbb \vdash Xz_2bcbb \vdash z_3XBcbb \vdash Xz_0Bcbb \vdash XXz_0cbb \vdash XXXz_2bb \vdash XXz_3XBb \vdash XXXz_0Bb \vdash XXXXz_0b \vdash XXXXXz_1\square \vdash XXXXXz_e\square$$