Technische Universität München
Fakultät für Informatik
M. Bauer, D. Golias, S. Riesner,
Dr. M. Schneider

WS 2002/2003
Lösungsvorschläge zu Blatt 13
29. Januar 2003

Übungen zu Einführung in die Informatik I

Aufgabe 44 Sprache und Grammatik (Lösungsvorschlag)

Um zu zeigen, dass $L = L(G)$ gilt, muss man die Inklusionen $L \subseteq L(G)$ und $L(G) \subseteq L$ zeigen.

- $L \subseteq L(G)$

  Zum Nachweis von $L \subseteq L(G)$ muss man $a^m b^n \in L(G)$ für gegebenes m, n zeigen, d.h. eine Herleitung von $a^m b^n$ aus S angeben:

  $$\begin{array}{llll} S & \stackrel{m-1}{\rightarrow} & a^{m-1}S & (m-1) \text{ Anwendungen von (P1)} \\ & \rightarrow & a^m U & 1 \text{ Anwendung von (P2)} \\ & \stackrel{n-1}{\rightarrow} & a^m b^{n-1} U & (n-1) \text{ Anwendungen von (P3)} \\ & \rightarrow & a^m b^n & 1 \text{ Anwendung von (P4)} \end{array}$$

- $L(G) \subseteq L$

  Zum Nachweis von $L(G) \subseteq L$ charakterisieren wir (durch Induktion nach k) die in k Schritten aus S herleitbaren Wörter (Sätze und Satzformen):

  $$S \stackrel{k}{\rightarrow} w \quad \Rightarrow \quad \begin{array}{rl} w = a^k S & \text{oder} \\ (\exists m \geq 1, n \geq 0, m+n=k)(w = a^m b^n U) & \text{oder} \\ (\exists m \geq 1, n \geq 0, m+n=k)(w = a^m b^n) & \end{array}$$

  Für terminales w zeigt dies, dass $S \stackrel{k}{\rightarrow} w$ impliziert, dass $w \in L$ gilt.

  - **Induktionsanfang**

    Für $k = 0$ ist die Behauptung klar, da $S \stackrel{0}{\rightarrow} w$ nur für $w = S = a^0 S$ gilt.

  - **Induktionsschritt**

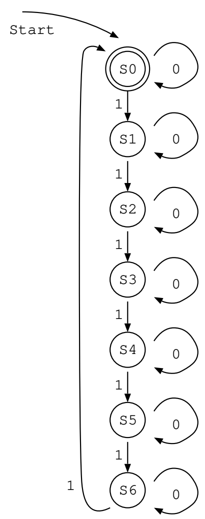
    Im Induktionsschritt von k nach $k+1$ betrachtet man den letzten Schritt in der Herleitung $S \stackrel{k+1}{\rightarrow} w$, d.h. $S \stackrel{k}{\rightarrow} w' \rightarrow w$.

    Erfolgt dieser vermöge der Regeln (P1) oder (P2), so muss w' die Variable S enthalten, also nach Induktionsvoraussetzung $w' = a^k S$ gelten. Es ist dann aber $w = a^{(k+1)} S$ bzw. $w = a^{(k+1)} U$ von der verlangten Gestalt.

    Wurde die Regel (P3) oder (P4) angewendet, muss entsprechend U in w' vorkommen und daher $w' = a^m b^n U$ für geeignete $m \geq 1$, $n \geq 0$ mit $m + n = k$ gelten. Hier gilt dann $w = a^m b^{(n+1)} U$ oder $w = a^m b^{(n+1)}$, weshalb wegen $m + (n+1) = k+1$ das Wort w wieder eine der gewünschten Gestalten hat.

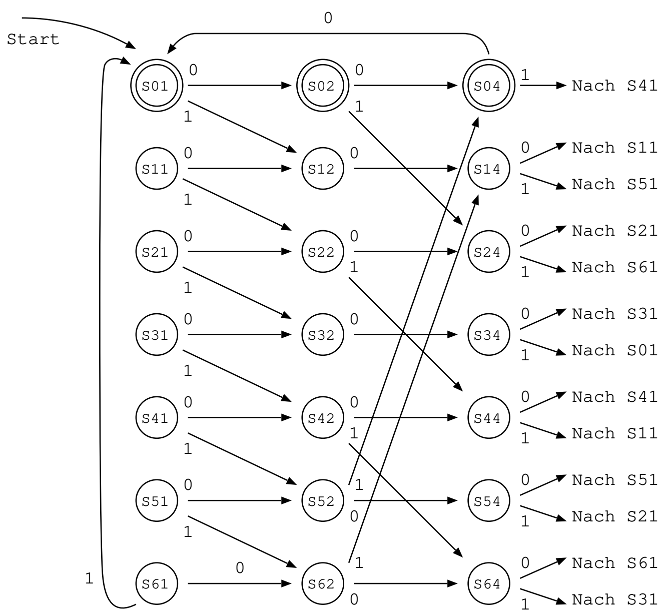
Aufgabe 45 Endliche Automaten (Lösungsvorschlag)

b)



Grundlegende Idee:
Die Zustandsnummer gibt an, welcher Rest bei einer Division durch 7 verbleibt. Rest 0 bedeutet, dass die Zahl durch 7 teilbar ist. Akzeptierender Zustand S0.
Wird eine 0 eingelesen, so wird der aktuelle Zustand beibehalten.
Wird eine 1 eingelesen, so geht der Zähler einen Zustand weiter, da sich ja auch der Divisionsrest erhöht.
Mit dem Übergang von S6 auf S0 wird Modulo realisiert, d.h. bei Teilung der aktuellen Summe durch 7 bleibt Rest 0.

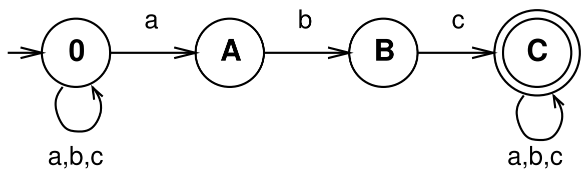
c)



Grundlegende Idee:
Das Prinzip der Restbildung gem. Teilabschnitt a bleibt erhalten. Da hier jedoch noch eine Gewichtung stattfindet, muss auch vermerkt werden, an welcher Gewichtungsstelle sich die aktuelle Binärstelle befindet. Dafür die zusätzlichen Zustandsspalten. Damit gilt:
Wird eine 0 eingelesen, so bleibt die Restzählung erhalten, jedoch wird die Gewichtungsspalte gewechselt.
Wird eine 1 eingelesen, so wird sowohl die Gewichtungsspalte gewechselt, als auch die Restzählung durchgeführt.
Alle Zustände mit Rest 0 akzeptieren, da Modulo 0 gilt.
Zu beachten ist, dass die Binärzahl mit der niedrigsten Stelle beginnend einzulesen ist, da sonst der Startzustand nicht S01 ist.

Aufgabe 46 Nichtdeterministische Automaten (Lösungsvorschlag)

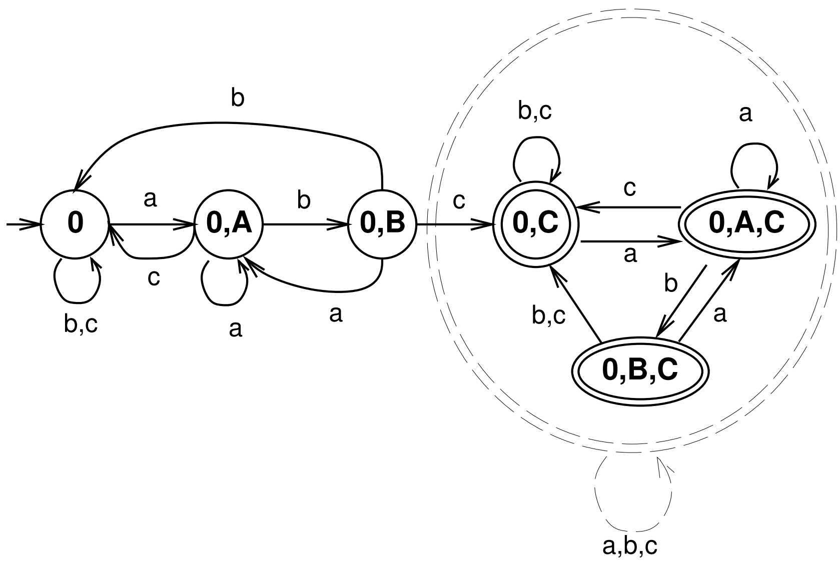
a) Ein nichtdeterministischer Automat, der ein Wort, das das Teilwort `abc` enthält, akzeptiert, hat die Form:



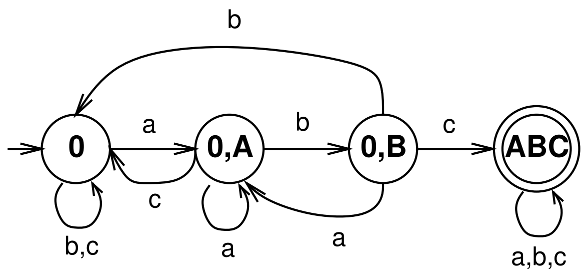
b) Um diesen nichtdeterministischen endlichen Automaten deterministisch zu machen, kann man die sog. „Myhill-Konstruktion“ durchführen. Dabei geht man so vor, daß ausgehend vom Startzustand die Zustandsmengen errechnet werden, die mit den Eingabezeichen erreichbar sind. Dieses Verfahren wird für jede neu entstehende Zustandsmenge ausgeführt, bis keine neue mehr entsteht. Diese Zustandsmengen bilden dann die Zustände des deterministischen endlichen Automaten. Jede Zustandsmenge, die einen Endzustand (hier Zustand *C*) enthält, wird zum Endzustand im deterministischen Automaten.

	a	b	c
0	**0,A**	0	0
0,A	0,A	**0,B**	0
0,B	0,A	0	**0,C**
0,C	**0,A,C**	0,C	0,C
0,A,C	0,A,C	**0,B,C**	0,C
0,B,C	0,A,C	0,C	0,C

Also ergibt sich folgende graphische Darstellung des endlichen Automaten:



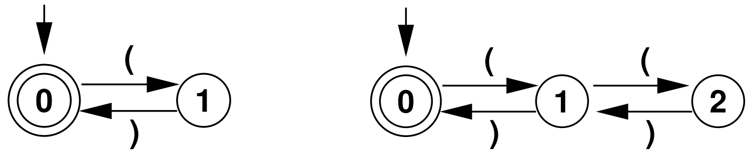
Wie man jedoch sehen kann, können nach Erreichen des Zustands **0,C** nur noch Endzustände erreicht werden (jeder Endzustand hat zudem Übergänge für sämtliche Zeichen des Eingabealphabets, d.h. der Fangzustand wird ebenfalls nicht von dort aus erreicht). Deshalb kann man die 3 Endzustände **0,C**, **0,A,C** und **0,B,C** zu einem Endzustand **ABC** zusammenfassen, der erreicht wird, nachdem *abc* gelesen wurde und dann nicht mehr verlassen wird. Durch diese „Minimierung“ ergibt sich folgender Automat:



Aufgabe 47 Pumping–Lemma für reguläre Sprachen (Lösungsvorschlag)

Für Programmiersprachen sind Klammerausdrücke ein wesentliches Hilfsmittel zur Strukturierung von Ausdrücken. Dazu muss im Prinzip die Anzahl öffnender Klammern in einem Ausdruck mit der Anzahl schließender Klammern übereinstimmen. Außerdem darf die Anzahl der schließenden Klammern bei korrekter Klammerung in keinem Anfangsteilwort größer sein als die der schließenden Klammern.

a) Beschränken wir uns bei der Differenz von öffnenden und schließenden Klammern auf eine feste Zahl n, so können wir dafür jeweils Automaten definieren, die „zählen" wie groß die Differenz ist:



Ein endlicher Automat verfügt allerdings nur über eine endliche Zustandsmenge, um die Differenz von öffnenden und schließenden Klammern zählen zu können. Will man eine beliebige Differenz erlauben, so ist das Konzept der endlichen Automaten nicht mehr mächtig genug, um genau die gewüschte Sprache zu akzeptieren. Dies führt uns zum Beweis in der nächsten Aufgabe, in der wir zeigen dass die Sprache der beliebig tief verschachtelten Klammerausdrücke nicht regulär ist bzw. nicht von einem endlichen Automaten akzeptiert wird.

b) Um mit dem Pumping–Lemma zu zeigen, daß eine Sprache L nicht regulär ist, wählen wir uns aus L zu der gegebenen Konstante n ein geeignetes Wort $x \in L$ mit $|x| \geq n$, von dem wir zeigen können, daß bei **jeder möglichen** Zerlegung $x = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq n$ mindestens eines der Wörter aus $\{uv^iw \mid i \geq 0\}$ nicht in L liegt.

Sei nun n die Konstante aus dem Pumping–Lemma. Wir wählen das Wort a^nb^n, das offensichtlich in L liegt. Bei jeder Zerlegung $a^nb^n = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq n$ gilt, daß v ausschließlich aus a besteht, d. h. $v = a^k$ für ein $k > 0$.

„Pumpen" wir nun das Wort x, so folgt

$$x' = uv^2w = a^{2k}a^{n-k}b^n = a^{n+k}b^n$$

Ebenso gilt:

$$uv^0w = uw = a^{n-k}b^n.$$

Beide Worte liegen also nicht in der Sprache L. Wäre L regulär, so müssten allerdings alle Wörter $x' = uv^iw, i \in \mathbb{N}$ in L liegen. Deshalb kann L nicht regulär sein!