Prof. Dr. Jasmin Blanchette
Jannis Limperg
Luca Maio

Ludwig-Maximilians-Universität München
Institut für Informatik
Abgabe bis 26. Juni 2024, 10:00 Uhr

Lösungsvorschlag zur Übung 9 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.

Erlaubte Konstrukte für

LOOP-Programme	WHILE-Programme	GOTO-Programme
$x_i := x_j + c$	$x_i := x_j + c$	$M_k : x_i := x_j + c$
$x_i := x_j - c$	$x_i := x_j - c$	$M_k : x_i := x_j - c$
$x_i := c$	$x_i := c$	$M_k : x_i := c$
$P_1; P_2$	$P_1; P_2$	$P_1; P_2$
LOOP x_i **DO** P **END**	**LOOP** x_i **DO** P **END**	$M_k :$ **GOTO** M_j
	WHILE $x_i \neq 0$ **DO** P **END**	$M_k :$ **IF** $x_i = c$ **THEN GOTO** M_j
		$M_k :$ **HALT**
		Dabei darf jede Marke M_k nur einmal vorkommen

FSK9-1 ***WHILE-, LOOP- und GOTO-Programme*** (2 Punkte)

In den folgenden Teilaufgaben sollen Sie jeweils ein Programm angeben. Verwenden Sie dabei nur die erlaubten Anweisungen aus der obigen Tabelle. Kommentieren Sie außerdem Ihre Programme, sodass klar wird, wie sie funktionieren sollen. Nicht kommentierte Programme werden eventuell nicht gewertet.

a) Schreiben Sie ein GOTO-Programm, das der Variablen x_0 den Wert von $3 \cdot x_1 \cdot x_2$ zuweist.

LÖSUNGSVORSCHLAG:

```
// Initialisiere x_0 = 0
M_1 :   x_0 := 0;
M_2 :   x_4 := x_1 + 0;
// x_1-mal:
M_3 :   IF x_4 = 0 THEN GOTO M_11;
M_4 :     x_3 := x_2 + 0;
          // x_2-mal:
M_5 :     IF x_3 = 0 THEN GOTO M_9;
            // Inkrementiere x_0
M_6 :       x_0 := x_0 + 1;
M_7 :       x_3 := x_3 - 1;
M_8 :       GOTO M_5;
M_9 :     x_4 := x_4 - 1;
M_10 :    GOTO M_3;
M_11 :  x_4 := 2;
M_12 :  x_5 := x_0 + 0;
// 2-mal, da bereits einmal in x_0:
M_13 :  IF x_4 = 0 THEN GOTO M_21;
M_14 :    x_3 := x_5 + 0;
          // Altes x_0-mal:
M_15 :    IF x_3 = 0 THEN GOTO M_19;
            // Inkrementiere x_0
M_16 :      x_0 := x_0 + 1;
M_17 :      x_3 := x_3 - 1;
M_18 :      GOTO M_15;
M_19 :    x_4 := x_4 - 1;
M_20 :    GOTO M_13;
M_21 :  HALT
```

b) Schreiben Sie ein WHILE-Programm, das die folgende Funktion berechnet:

$$f : \mathbb{N}^2 \to \mathbb{N}, \quad f(x, y) = 5 * (x + y)$$

LÖSUNGSVORSCHLAG:
Gemäß der Definition von WHILE-Berechenbarkeit erhalten wir die Einga-

ben x in x_1 und y in x_2 und speichern die Ausgabe in x_0.

```
// Initialisiere x_0 als x
x_0 := x_1 + 0;
x_3 := x_2 + 0;
// Addiere y:
WHILE x_3 ≠ 0 DO
   x_0 := x_0 + 1;
   x_3 := x_3 − 1
END;
x_4 := x_0 + 0;
x_5 := 4;
// 4-mal, da bereits einmal in x_0:
WHILE x_5 ≠ 0 DO
   // Addiere alten Wert von x_0 zum Ergebnis:
   // Initialisiere x_3 = Altes x_0
   x_3 := x_4 + 0;
   // Altes x_0-mal:
   WHILE x_3 ≠ 0 DO
      // Inkrementiere das Ergebnis
      x_0 := x_0 + 1;
      x_3 := x_3 − 1
   END;
   x_5 := x_5 − 1
END
```

c) Schreiben Sie ein LOOP-Programm, das den folgenden Pseudocode implementiert:

FOR $x_i := 0$ **TO** x_1 **DO** $x_0 := x_0 + x_i$ **END**

LÖSUNGSVORSCHLAG:

```
// Initialisiere x_i mit 0:
x_i := 0;
// x_1 mal:
LOOP x_1 DO
  // Addiere x_i zu x_0:
  LOOP x_i DO
    x_0 := x_0 + 1
  END;
  // Inkrementiere x_i:
  x_i := x_i + 1
END;
// Addiere ein letztes mal x_i zu x_0, da wir es zuletzt
// mit dem Wert x_1 − 1 addiert hatten und noch nicht mit dem Wert x_1:
LOOP x_i DO
  x_0 := x_0 + 1
END
```

FSK9-2 ***LOOPY-Programme*** (0 Punkte)

Wir betrachten LOOPY-Programme, die aus folgenden Anweisungen bestehen:

- $x_i := x_j + c$, $x_i := x_j - c$ und P_1; P_2 wie bei LOOP-Programmen.
- $(P_1 \mid P_2)$, wobei P_1 und P_2 LOOPY-Programme sind. Diese Anweisung führt nichtdeterministisch entweder P_1 oder P_2 aus.
- **LOOPY** P **END**. Diese Anweisung führt das LOOPY-Programm P nichtdeterministisch 0 oder mehr Male aus.

Beispielsweise führt das LOOPY-Programm

$$\textbf{LOOPY}\ (x_0 := x_0 + 2 \mid x_0 := x_0 - 1)\ \textbf{END}$$

beliebig oft entweder die Anweisung $x_0 := x_0 + 2$ oder die Anweisung $x_0 := x_0 - 1$ aus, wobei es in jeder Iteration eine andere Entscheidung treffen kann. Der Wert von x_0 am Ende des Programms kann also jede natürliche Zahl sein.

a) Die Semantik eines LOOPY-Programms können wir als eine Relation $\xrightarrow[\text{LOOPY}]{}$ definieren. Diese Relation ist ähnlich der für WHILE-Programme, aber LOOPY-Programme können nichtdeterministisch verschiedene Ergebnisse haben. Dementsprechend kann ein LOOPY-Programm P mit einer Variablenbelegung ρ mehrere Nachfolge-Variablenbelegungen ρ' und mehrere Nachfolge-Programme P' haben,

je nachdem, welche nichtdeterministischen Entscheidungen das Programm trifft. Für alle diese ρ' und P' soll gelten:

$$(\rho, P) \xrightarrow[\text{LOOPY}]{} (\rho', P')$$

Definieren Sie die Semantik von LOOPY-Programmen.

LÖSUNGSVORSCHLAG:
Zuweisungen zu Variablen funktionieren genau so wie bei LOOP-Programmen:

$$(\rho, x_i := x_j + c) \xrightarrow[\text{LOOPY}]{} (\rho\{x_i \mapsto \rho(x_j) + c\}, \varepsilon)$$
$$(\rho, x_i := x_j - c) \xrightarrow[\text{LOOPY}]{} (\rho\{x_i \mapsto \max(0, \rho(x_j) - c)\}, \varepsilon)$$

Sequenzierung ebenfalls wie bei LOOP-Programmen. Wir wählen eine etwas andere Formulierung als im Skript, aber beide Formulierungen sind äquivalent.

$$(\rho, P_1;\ P_2) \xrightarrow[\text{LOOPY}]{} (\rho', P_2) \quad \text{falls } (\rho, P_1) \xrightarrow[\text{LOOPY}]{}{}^{*} (\rho', \varepsilon)$$

Für die nichtdeterministische Ausführung zweier Programme brauchen wir eine Regel für jede Alternative:

$$(\rho, (P_1 \mid P_2)) \xrightarrow[\text{LOOPY}]{} (\rho, P_1)$$
$$(\rho, (P_1 \mid P_2)) \xrightarrow[\text{LOOPY}]{} (\rho, P_2)$$

LOOPY führt das gegebene Programm beliebig oft aus. Wir kodieren das hier mit zwei Regeln, eine für „0-mal“ und eine für „$(n+1)$-mal“:

$$(\rho, \textbf{LOOPY } P \textbf{ END}) \xrightarrow[\text{LOOPY}]{} (\rho, \varepsilon)$$
$$(\rho, \textbf{LOOPY } P \textbf{ END}) \xrightarrow[\text{LOOPY}]{} (\rho, P;\ \textbf{LOOPY } P \textbf{ END})$$

b) Zeigen Sie, dass mit Ihrer Semantik gilt:

$$(\{x_0 \mapsto 0\}, \textbf{LOOPY } (x_0 := x_0 + 2 \mid x_0 := x_0 - 1) \textbf{ END}) \xrightarrow[\text{LOOPY}]{}{}^{*} (\{x_0 \mapsto 1\}, \varepsilon)$$

LÖSUNGSVORSCHLAG:

$$(\{x_0 \mapsto 0\}, \textbf{LOOPY } (x_0 := x_0 + 2 \mid x_0 := x_0 - 1) \textbf{ END}) \xrightarrow[\text{LOOPY}]{}$$

$(\{x_0 \mapsto 0\},$
$\quad (x_0 := x_0 + 2 \mid x_0 := x_0 - 1);$ **LOOPY** $(x_0 := x_0 + 2 \mid x_0 := x_0 - 1)$ **END**$) \xrightarrow[\text{LOOPY}]{(1)}$
$(\{x_0 \mapsto 2\},$ **LOOPY** $(x_0 := x_0 + 2 \mid x_0 := x_0 - 1)$ **END**$) \xrightarrow[\text{LOOPY}]{}$
$(\{x_0 \mapsto 2\},$
$\quad (x_0 := x_0 + 2 \mid x_0 := x_0 - 1);$ **LOOPY** $(x_0 := x_0 + 2 \mid x_0 := x_0 - 1)$ **END**$) \xrightarrow[\text{LOOPY}]{(2)}$
$(\{x_0 \mapsto 1\},$ **LOOPY** $(x_0 := x_0 + 2 \mid x_0 := x_0 - 1)$ **END**$) \xrightarrow[\text{LOOPY}]{}$
$(\{x_0 \mapsto 1\}, \varepsilon)$

Aussage (1) gilt, da

$$(\{x_0 \mapsto 0\}, (x_0 := x_0 + 2 \mid x_0 := x_0 - 1)) \xrightarrow[\text{LOOPY}]{} (\{x_0 \mapsto 0\}, x_0 := x_0 + 2) \xrightarrow[\text{LOOPY}]{} (\{x_0 \mapsto 2\}, \varepsilon)$$

Analog für Aussage (2):

$$(\{x_0 \mapsto 2\}, (x_0 := x_0 + 2 \mid x_0 := x_0 - 1)) \xrightarrow[\text{LOOPY}]{} (\{x_0 \mapsto 2\}, x_0 := x_0 - 1) \xrightarrow[\text{LOOPY}]{} (\{x_0 \mapsto 1\}, \varepsilon)$$

c) Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ ist LOOPY-berechenbar, wenn es ein LOOPY-Programm P gibt, sodass es für alle $n_1, \ldots, n_k \in \mathbb{N}$ eine Variablenbelegung ρ gibt, für die gilt:

$$(\{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}, P) \xrightarrow[\text{LOOPY}]{}{}^* (\rho, \varepsilon)$$

und $\rho(x_0) = f(n_1, \ldots, n_k)$.
Zeigen Sie: Jede Funktion, die LOOP-berechenbar ist, ist auch LOOPY-berechenbar.

LÖSUNGSVORSCHLAG:
Angenommen $f\colon \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar, d.h. es gibt ein LOOP-Programm P, sodass

$$(\{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}, P) \xrightarrow[\text{LOOP}]{}{}^* (\rho, \varepsilon)$$

mit $\rho(x_0) = f(n_1, \ldots, n_k)$. Wir zeigen, dass es dann auch ein LOOPY-Programm P' gibt, für das gilt:

$$(\{x_1 \mapsto n_1, \ldots, x_k \mapsto n_k\}, P') \xrightarrow[\text{LOOPY}]{}{}^* (\rho, \varepsilon)$$

Daraus folgt direkt, dass f LOOPY-berechenbar ist.
Um die Behauptung zu zeigen, geben wir für jedes mögliche LOOP-Programm ein entsprechendes LOOPY-Programm an. Für dieses LOOPY-Programm muss gelten, dass eine seiner möglichen (nichtdeterministischen) Ausführungen der Ausführung des LOOP-Programms entspricht.

Für die LOOP-Programme $x_i := x_j + c$, $x_i := x_j - c$ und $P_1;\ P_2$ ist das trivial, da diese Programme auch LOOPY-Programme sind.
Für jedes LOOP-Programm $x_i := c$ wählen wir ein LOOPY-Programm $x_i := x_j + c$, wobei x_j eine Variable ist, die sonst nirgends vorkommt. Das LOOP-Programm hat die Ausführung

$$(\rho, x_i := c) \xrightarrow[\text{LOOP}]{} (\rho\{x_i \mapsto c\}, \varepsilon)$$

Das LOOPY-Programm hat die gleiche Ausführung

$$(\rho, x_i := x_j + c) \xrightarrow[\text{LOOPY}]{} (\rho\{x_i \mapsto 0 + c\}, \varepsilon)$$

da stets $x_j = 0$ gilt.
Für das LOOP-Programm **LOOP** x_i **DO** P **END** wählen wir das LOOPY-Programm **LOOPY** P' **END**. Dabei ist P' ein LOOPY-Programm, das eine mögliche Ausführung hat, die der Ausführung von P entspricht. Dass so ein Programm P' existiert, ergibt sich aus der Induktionshypothese.
Wenn wir das LOOP-Programm ausführen, erhalten wir

$$(\rho, \textbf{LOOP } x_i \textbf{ DO } P \textbf{ END}) \xrightarrow[\text{LOOP}]{} (\rho, \underbrace{P;\ \ldots;\ P}_{\rho(x_i)\text{-mal}})$$

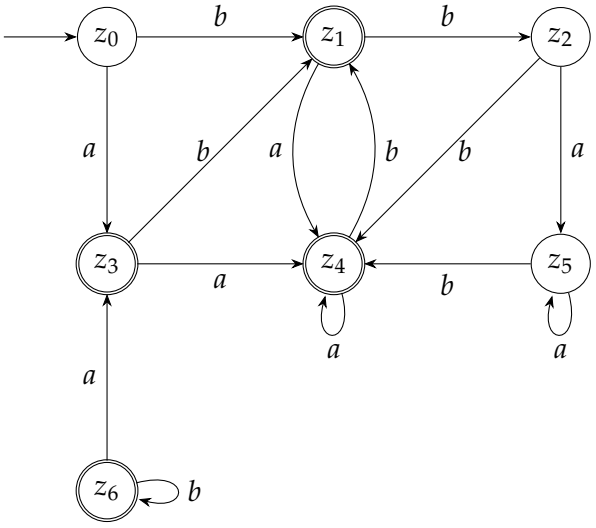
Das LOOPY-Programm generiert die Ausführungen

$$(\rho, \textbf{LOOPY } P' \textbf{ END}) \xrightarrow[\text{LOOPY}]{}{}^{*} (\rho, \underbrace{P';\ \ldots;\ P'}_{k\text{-mal}})$$

für beliebige k und damit insbesondere für $k = \rho(x_i)$. Dieses Programm hat als eine mögliche Ausführung die Ausführung des LOOP-Programms.
(Hier braucht das LOOPY-Programm mehrere $\xrightarrow[\text{LOOPY}]{}$-Schritte, um einen $\xrightarrow[\text{LOOP}]{}$-Schritt zu simulieren, aber das schadet nicht. Wir hätten die Semantik von **LOOPY** auch analog zu der von **LOOP** definieren können, um diese Diskrepanz zu vermeiden.)

FSK9-3 ***Algorithmen Wiederholung*** (2 Punkte)

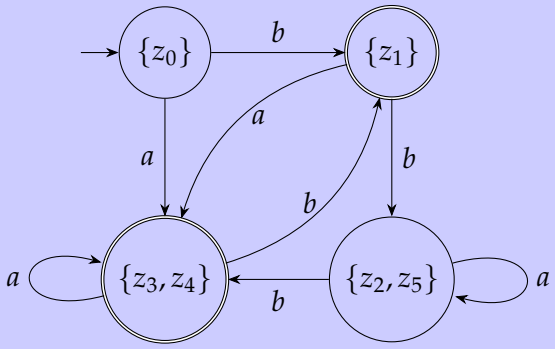
a) Minimieren Sie den folgenden DFA. Verwenden Sie die tabellarische Variante des Algorithmus zur Minimierung von DFAs aus der Vorlesung (nicht die graphische Variante und nicht den Algorithmus von letztem Jahr). Geben Sie die Partitionstabelle und den minimalen DFA an.



LÖSUNGSVORSCHLAG:
Wir entfernen den unerreichbaren Zustand z_6.
Partitionstabelle:

z_0	z_2	z_5	z_1	z_3	z_4	
z_0	z_2	z_5	z_1	z_3	z_4	mit a
z_0	z_2	z_5	z_1	z_3	z_4	mit b

Aus der Partitionierung ergibt sich der Minimalautomat:



Sei G die Grammatik $(\{A_1, A_2, A_3, A_4, A_5\}, \{\$, \#\}, P, A_1)$ mit

$$\begin{aligned} P = \{ & A_1 \to A_2A_5 \mid A_3A_5, \\ & A_2 \to A_4A_3 \mid \$, \\ & A_3 \to A_2A_5, \\ & A_4 \to \$, \\ & A_5 \to A_5A_1 \mid \# \} \end{aligned}$$

Prüfen Sie mit dem CYK-Algorithmus, ob das Wort \$##\$## in $L(G)$ ist. Erstellen

Sie dazu die entsprechende Tabelle des Algorithmus und erklären Sie anhand der Tabelle, ob das Wort in $L(G)$ ist.

LÖSUNGSVORSCHLAG:

Wort:	\$	#	#	\$	#	#
j\ i	1	2	3	4	5	6
1	A_2, A_4	A_5	A_5	A_2, A_4	A_5	A_5
2	A_1, A_3			A_1, A_3		
3	A_1		A_5	A_1		
4			A_5			
5	A_1					
6	A_1					

Da das Startsymbol A_1 in Zeile 6, Spalte 1 enthalten ist, ist \$\#\#\$\#\# $\in L(G)$.

FSK9-4 ***Kontextfreie Sprachen*** (0 Punkte)

Sei L die formale Sprache aller Wörter $w \in \{a,b\}^*$, sodass w in der zweiten Hälfte mindestens ein b enthält:

$$L = \{ubv \mid u, v \in \{a,b\}^*, |u| > |v|\}$$

b) Geben Sie eine kontextfreie Grammatik an, die L erzeugt.

LÖSUNGSVORSCHLAG:
$G = (\{S, U, V\}, \{a, b\}, P, S)$ mit

$$P = \{S \to USV \mid Ub, U \to a \mid b \mid UU, V \to a \mid b\}$$

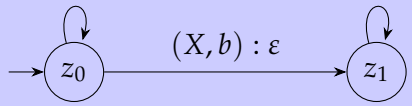
Erläuterung: S erzeugt eine Satzform der Form $U^i bV^{i-1}$. Dann können beliebig viele weitere U links von b erzeugt werden mit $U \to UU$, wobei die Anzahl an V gleich bleibt, damit gilt $|u| > |v|$. Die Nichtterminale U und V können dann beliebig a's und b's erzeugen.

b) Geben Sei einen Kellerautomaten an, der L akzeptiert. Entsprechend der Definition aus der Vorlesung soll Ihr Kellerautomat durch leeren Keller akzeptieren.

Hinweis: Nutzen Sie den Nichtdeterminismus des Automaten aus.

LÖSUNGSVORSCHLAG:

$(\#, a) : X\#$ $\quad$ $(\#, \varepsilon) : \varepsilon$
$(\#, b) : X\#$ $\quad$ $(X, \varepsilon) : \varepsilon$
$(X, a) : XX$ $\quad$ $(X, a) : \varepsilon$
$(X, b) : XX$ $\quad$ $(X, b) : \varepsilon$



Erläuterung: Im Startzustand z_0 werden beliebige a und b gelesen, wobei jeweils ein X pro Zeichen in den Keller gelegt wird. Der Nichtdeterminismus wird verwendet, um das richtige b zu lesen und in den Zustand z_1 zu wechseln. Mit diesem Wechseln können nur noch soviele a und b gelesen werden, wie zuvor gelesen wurden, da jedes Lesen ein X vom Keller abbaut. Folgen weniger Zeichen, so werden die verbleibenden X durch ε-Übergänge abgebaut. Das Startsymbol im Keller wird durch einen weiteren ε-Übergang entfernt.