PD Dr. Jan Johannsen
Elisabeth Lempa
Luca Maio

Ludwig-Maximilians-Universität München
Institut für Informatik
Besprechung 25.06.2026 bis 29.06.2026
Abgabe bis 07.07.2026, 14:00 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- und GOTO-Programme***

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
```

FSK9-2 ***LOOPY-Programme***

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:

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

Aussage (1) gilt, da

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

Analog für Aussage (2):

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

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 ***μ-Rekursion***

Nehmen Sie für diese Aufgabe an, dass Addition, Subtraktion, Multiplikation, Division, Exponentiation und absolute Differenz $\mathit{absdiff}(x_1, x_2) = |x_1 - x_2|$ primitiv rekursiv sind.

Zur Erinnerung: Wir unterscheiden hier die (modifizierte) Subtraktion und die absoluten Differenz *absdiff*. (Wir schreiben in der folgenden Aufgabe zur besseren Unterscheidung $\dot{-}$ für die modifizierte Subtraktion.) Bei der (modifizierten) Subtraktion $x_1 \dot{-} x_2$ wird im Fall $x_2 > x_1$ auf 0 abgebildet, d.h. $3 \dot{-} 4 = 0$. Bei der absoluten Differenz werden dagegen x_1 und x_2 als ganze Zahlen subtrahiert, d.h. $|3 - 4| = 1$.

a) Berechnen Sie μg für folgende Funktionen g.

- $g\colon \mathbb{N} \to \mathbb{N}$, $g(x) = 3x^2 + 5x + 3$

  LÖSUNGSVORSCHLAG:
  Die Funktion g hat keine Nullstellen, also ist μg undefiniert:

  $$\mu g\colon \mathbb{N}$$
  $$\mu g = \text{undefiniert}$$

- $g\colon \mathbb{N} \to \mathbb{N}$, $g(x) = (|x/2 - 4|)^2 \dot{-} 2$.

  LÖSUNGSVORSCHLAG:
  Die Funktion g hat Nullstellen zwischen 6 und 11, von denen wir die kleinste nehmen:

  $$\mu g\colon \mathbb{N}$$
  $$\mu g = 6$$

- $g\colon \mathbb{N}^2 \to \mathbb{N}$, $g(x_1, x_2) = (x_1 + 1) \cdot x_2$

  LÖSUNGSVORSCHLAG:
  Die Funktion g nimmt den Wert 0 an g.d.w. $x_1 + 1 = 0$ oder $x_2 = 0$. Für $x_2 = 0$ ist μg also 0, da der Wert von x_1 hier beliebig ist und wir somit den kleinsten Wert, also 0, wählen; für alle anderen Werte von x_2 ist es undefiniert, da $x_1 \geq 0$ und damit nie gelten kann, dass $x_1 + 1 = 0$.

  $$\mu g\colon \mathbb{N} \to \mathbb{N}$$
  $$(\mu g)(x_2) = \begin{cases} 0 & \text{falls } x_2 = 0 \\ \text{undefiniert} & \text{sonst} \end{cases}$$

b) Zeigen Sie, dass die Quadratwurzelfunktion

$$sqrt\colon \mathbb{N} \to \mathbb{N}, \quad sqrt(x_1) = \sqrt{x_1}$$

μ-rekursiv ist. Beachten Sie, dass die Quadratwurzel für natürliche Zahlen nicht überall definiert ist. Es gilt:

$$sqrt(x_1) = n \iff n^2 = x_1$$

LÖSUNGSVORSCHLAG:
Für *sqrt* gilt:

$$sqrt(x_1) = n \iff n^2 = x_1 \iff |n^2 - x_1| = 0$$

Wir definieren also die primitiv rekursiv Funktion $sqrt'$:

$$\begin{aligned} sqrt'(n, x_1) &= absdiff(exp(n, 2), x_1) \\ &= absdiff(exp(\pi_1^2(n, x_1), g()), \pi_2^2(n, x_1)) \end{aligned}$$

wobei

$$g() = 2$$

Die Quadratwurzelfunktion ist dann

$$sqrt = \mu sqrt'$$

FSK9-4 ***Primitiv rekursive Prädikate***

Primitiv rekursive Funktionen, die auf $\{0, 1\}$ abbilden, können auch als Prädikate aufgefasst werden. Wir betrachten in dieser Aufgabe nur einstellige Prädikate p, wobei $p(x) = 0$ bedeutet, dass x die Eigenschaft p nicht besitzt, und $p(x) = 1$ bedeutet, dass x die Eigenschaft p besitzt.

a) Zeigen Sie, dass die Funktion *iszero* ein primitiv rekursives Prädikat ist.

$$iszero(x_1) = \begin{cases} 0 & \text{für } x_1 > 0 \\ 1 & \text{für } x_1 = 0 \end{cases}$$

LÖSUNGSVORSCHLAG:
Dass *iszero* auf {0,1} abbildet, ist an der Fallunterscheidung bereits zu sehen. Daher müssen wir nur noch zeigen, dass *iszero* primitiv rekursiv ist. Das kann man tun, indem man *iszero* wie folgt im primitiv rekursiven Schema angibt:

$$iszero(x_1) = \begin{cases} 1 & \text{für } x_1 = 0 \\ 0 & \text{sonst} \end{cases}$$

Dies entspricht dem Schema, denn es ist gleich

$$iszero(x_1) = \begin{cases} g() & \text{für } x_1 = 0 \\ h(iszero(x_1 - 1), x_1 - 1) & \text{sonst} \end{cases}$$

mit $g() = 1$ und $h(y_1, y_2) = 0$ (konstante Funktionen).

b) Zeigen Sie, dass die Funktion *even* ein primitiv rekursives Prädikat ist.

$$even(x_1) = \begin{cases} 1 & \text{für } x_1 \text{ gerade} \\ 0 & \text{für } x_1 \text{ ungerade} \end{cases}$$

LÖSUNGSVORSCHLAG:
Auch hier genügt es, *even* im primitiv rekursiven Schema anzugeben:

$$even(x_1) = \begin{cases} 1 & \text{für } x_1 = 0 \\ iszero(even(x_1 - 1)) & \text{sonst} \end{cases}$$

Die verwendeten Funktionen sind $g() = 1$ (konstante Funktion) und $h(y_1, y_2) = iszero(\pi_1^2(y_1, y_2))$ (Komposition).
Diese sind ebenfalls wieder in das primitiv rekursive Schema einzusetzen.

c) Zeigen Sie, dass die Funktion *ifnotzero* primitiv rekursiv ist.

$$ifnotzero(x_1, x_2, x_3) = \begin{cases} x_2 & \text{falls } x_1 > 0 \\ x_3 & \text{sonst} \end{cases}$$

LÖSUNGSVORSCHLAG:
Mit $g(y_1, y_2) = \pi_2^2(y_1, y_2)$, $h(y_1, y_2, y_3, y_4) = \pi_3^4(y_1, y_2, y_3, y_4)$ im primitiv rekursiven Schema erhält man

$$ifnotzero(x_1, x_2, x_3) = \begin{cases} g(x_2, x_3) & \text{falls } x_1 = 0 \\ h(ifnotzero(x_1 - 1, x_2, x_3), x_1 - 1, x_2, x_3) & \text{sonst} \end{cases}$$

oder, besser lesbar:

$$ifnotzero(x_1, x_2, x_3) = \begin{cases} x_3 & \text{falls } x_1 = 0 \\ x_2 & \text{sonst} \end{cases}$$

d) Zeigen Sie, dass die Funktion *ifthenelse* primitiv rekursiv ist, angenommen, dass $p(x)$ ein primitiv rekursives Prädikat ist.

$$ifthenelse(x_1, x_2, x_3) = \begin{cases} x_2 & \text{falls } p(x_1) \\ x_3 & \text{sonst} \end{cases}$$

LÖSUNGSVORSCHLAG: $ifthenelse(x_1, x_2, x_3) = ifnotzero(p(x_1), x_2, x_3)$
Dies entspricht dem primitiv rekursiven Schema, denn es ist gleich

$$ifthenelse(x_1, x_2, x_3) = ifnotzero(h_1(x_1, x_2, x_3), h_2(x_1, x_2, x_3), h_3(x_1, x_2, x_3))$$

wobei gilt:

$$\begin{aligned} h_1(x_1, x_2, x_3) &= p(\pi^3_1(x_1, x_2, x_3)) \\ h_2(x_1, x_2, x_3) &= \pi^3_2(x_1, x_2, x_3) \\ h_3(x_1, x_2, x_3) &= \pi^3_3(x_1, x_2, x_3) \end{aligned}$$

e) Zeigen Sie, dass, gegeben zwei primitiv rekursive Prädikate p und q, auch die Konjunktion $p \wedge q$ primitiv rekursiv ist.

LÖSUNGSVORSCHLAG:
Mit Multiplikation oder *ifnotzero*.

Wir zeigen es hier sogar für beliebig-stellige Prädikate:

$$(p \wedge q)(x_1, \ldots, x_k) = ifnotzero(p(x_1, \ldots, x_k), q(x_1, \ldots, x_k), z(x_1, \ldots, x_k))$$

Dabei ist $z(x_1, \ldots, x_k) = 0$ eine konstante Funktion.

Da *iszero* auf $\{0, 1\}$ wie die Negation wirkt, ist diese auch primitiv rekursiv. Da $\{\neg, \wedge\}$ funktional vollständig ist, können wir damit beliebige logische Verknüpfungen herstellen.