LOOP-Programme: Syntaktische Komponenten

LOOP-Programme bestehen aus folgenden Zeichen (syntaktischen Komponenten):

- Variablen: $x_0 \quad x_1 \quad x_2 \quad \ldots$
- Konstanten: $0 \quad 1 \quad 2 \quad \ldots$
- Operationssymbole: $+ \quad -$
- Trennsymbole: $; \quad :=$
- Schlüsselwörter: $\mathrm{LOOP} \quad \mathrm{DO} \quad \mathrm{END}$

LOOP-Programme: Syntax

Die Syntax von LOOP-Programmen wird wie folgt induktiv definiert.

(i) Jede Wertzuweisung der Form

$$x_i := x_j + c \qquad \text{bzw.} \qquad x_i := x_j - c$$

ist ein LOOP-Programm, wobei c eine Konstante ist.

(ii) Sind P_1, P_2 LOOP-Programme, dann sind auch

$$P_1; P_2 \qquad \text{sowie} \qquad \text{LOOP } x_i \text{ DO } P_1 \text{ END}$$

LOOP-Programme.

LOOP-Programme: Semantik, Teil i)

(i) Jede Wertzuweisung der Form $x_i := x_j + c$ wird wie "üblich" interpretiert: der neue Wert der Variablen x_i berechnet sich als Summe des Wertes der Variablen x_j und der Konstanten c, wobei der Wert in der Variablen x_j erhalten bleibt.

Die Wertzuweisung $x_i := x_j - c$ wird analog interpretiert, wobei sich aber die Werte nach der sogenannten modifizierten Differenz "$\dot{-}$", die wie folgt definiert ist

$$n_1 \dot{-} n_2 = \begin{cases} n_1 - n_2 & \text{falls } n_1 \geq n_2, \\ 0 & \text{sonst,} \end{cases}$$

berechnen.

LOOP-Programme: Semantik, Teil ii)

ii) Ein LOOP-Programm der Form $P_1; P_2$ soll die Hintereinanderausführung der Programme P_1 und P_2 bedeuten, also zuerst wird das Programm P_1, dann das Programm P_2 ausgeführt.

Ein LOOP-Programm der Form $\mathrm{LOOP}\ x_i\ \mathrm{DO}\ P_1\ \mathrm{END}$ bedeutet, dass das Programm P_1 sooft ausgeführt wird, wie der Wert der Variablen x_i zu Beginn angibt. Änderungen des Wertes der Variablen x_i haben also keinen Einfluss auf die Anzahl der Wiederholungen.

LOOP-berechenbare Funktionen

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$, heißt LOOP-berechenbar, falls es ein LOOP-Programm P gibt, das f in dem Sinne berechnet, dass P,

gestartet mit $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_k$ und 0 in den restlichen Variablen,

mit dem Wert $f(n_1, n_2, \ldots, n_k)$ in der Variablen x_0 stoppt.

Erstes Beispiel einer LOOP-berechenbaren Funktion

Gegeben sei das LOOP-Programm

$$x_0 := x_1 + 0;$$
$$\text{LOOP } x_2 \text{ DO } x_0 := x_0 + 1 \text{ END}$$

Man erkennt leicht, dass das Programm mit dem Wert der Summe der Anfangsbelegungen der Variablen x_1 und x_2 in der Variablen x_0 stoppt. Es berechnet also die Addition

$$+ : \mathbb{N}^2 \to \mathbb{N} \text{ vermöge } (x_1, x_2) \mapsto +(x_1, x_2) = x_1 + x_2.$$

Also ist die Addition LOOP-berechenbar.

Zweites Beispiel einer LOOP-berechenbaren Funktion

Gegeben sei das LOOP-Programm

$$\begin{aligned}&\text{LOOP } x_2 \text{ DO}\\ &\qquad\text{LOOP } x_1 \text{ DO } x_0 := x_0 + 1 \text{ END}\\ &\qquad\text{END}\end{aligned}$$

Eine genaue Betrachtung des Programms zeigt, dass damit die Funktion

$$\cdot : \mathbb{N}^2 \to \mathbb{N} \text{ vermöge } (x_1, x_2) \mapsto \cdot(x_1, x_2) = x_1 \cdot x_2,$$

berechnet wird. Die Multiplikation ist damit also LOOP-berechenbar.

Man beachte, dass die Anfangsbelegung der Variablen x_0 natürlich laut Definition 0 ist. Das wird hier gebraucht und verwendet.

Drittes Beispiel einer LOOP-berechenbaren Funktion

Das Konstrukt

$$\text{IF } x_1 = 0 \text{ THEN } A \text{ ELSE } B \text{ END}$$

wird durch das LOOP-Programm

$$\begin{aligned}
&x_2 := 1;\ x_3 := 0;\\
&\text{LOOP } x_1 \text{ DO } x_2 := 0;\ x_3 := 1 \text{ END};\\
&\text{LOOP } x_2 \text{ DO } A \text{ END};\\
&\text{LOOP } x_3 \text{ DO } B \text{ END}
\end{aligned}$$

simuliert. Dabei sind die Variablen x_2 und x_3 natürlich nicht in den Programmen A und B enthalten.

Aussagen über LOOP-berechenbarer Funktionen

- Jede von einem LOOP-Programm berechnete Funktion ist total.
  (Da die Anzahl der Abläufe einer LOOP-Schleife endlich ist, stoppt das Programm bei jeder Eingabe.)
- Es gibt (intuitiv) berechenbare Funktionen, die nicht LOOP-berechenbar sind.
  (z.B. jede berechenbare Funktion, die nicht total ist)
- Es gibt totale und (intuitiv) berechenbare Funktionen, die nicht LOOP-berechenbar sind.

WHILE-Programme: Syntaktische Komponenten

WHILE-Programme bestehen aus folgenden Zeichen (syntaktischen Komponenten):

- Variablen: $x_0 \quad x_1 \quad x_2 \quad \ldots$
- Konstanten: $0 \quad 1 \quad 2 \quad \ldots$
- Trennsymbole: $; \quad := \quad \neq$
- Operationssymbole: $+ \quad -$
- Schlüsselwörter: $\mathrm{LOOP} \quad \mathrm{WHILE} \quad \mathrm{DO} \quad \mathrm{END}$

WHILE-Programme: Syntax

Die Syntax von WHILE-Programmen wird wie folgt induktiv definiert.

(i) Jede Wertzuweisung der Form

$$x_i := x_j + c \qquad \text{bzw.} \qquad x_i := x_j - c$$

ist ein WHILE-Programm, wobei c eine Konstante ist.

(ii) Sind P_1, P_2 WHILE-Programme, dann sind auch

$$P_1; P_2 \quad \text{und} \quad \text{LOOP } x_i \text{ DO } P_1 \text{ END} \quad \text{und} \quad \text{WHILE } x_i \neq 0 \text{ DO } P_1 \text{ END}$$

WHILE-Programme.

WHILE-Programme: Semantik, Teil i)

Jede Wertzuweisung der Form $x_i := x_j + c$ wird wie "üblich" interpretiert: der neue Wert der Variablen x_i berechnet sich als Summe des Wertes der Variablen x_j und der Konstanten c, wobei der Wert in der Variablen x_j erhalten bleibt.

Die Wertzuweisung $x_i := x_j - c$ wird analog interpretiert, wobei sich aber die Werte nach der sogenannten modifizierten Differenz " $\dot{-}$ ", die wie folgt definiert ist

$$n_1 \dot{-} n_2 = \begin{cases} n_1 - n_2 & \text{falls } n_1 \geq n_2, \\ 0 & \text{sonst,} \end{cases}$$

berechnen.

WHILE-Programme: Semantik, Teil ii)

Ein WHILE-Programm der Form $P_1; P_2$ soll die Hintereinanderausführung der Programme P_1 und P_2 bedeuten, also zuerst wird das Programm P_1, dann das Programm P_2 ausgeführt.

Ein WHILE-Programm der Form $\mathrm{LOOP}\ x_i\ \mathrm{DO}\ P_1\ \mathrm{END}$ bedeutet, dass das Programm P_1 sooft ausgeführt wird, wie der Wert der Variablen x_i zu Beginn angibt. Änderungen des Wertes der Variablen x_i haben also keinen Einfluss auf die Anzahl der Wiederholungen.

Ein WHILE-Programm der Form $\mathrm{WHILE}\ x_i \neq 0\ \mathrm{DO}\ P_1\ \mathrm{END}$ bedeutet, dass das Programm P_1 solange ausgeführt wird, wie der Wert der Variablen x_i ungleich Null ist. Es findet also vor jedem erneuten Durchlauf des Programms P_1 eine Abfrage der Variablen x_1 statt.

WHILE-berechenbare Funktionen

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$, heißt WHILE-berechenbar, falls es ein WHILE-Programm P gibt, das f in dem Sinne berechnet, dass P,

gestartet mit $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_k$ und 0 in den restlichen Variablen,

mit dem Wert $f(n_1, n_2, \ldots, n_k)$ in der Variablen x_0 stoppt.

Ist $f(n_1, n_2, \ldots, n_k)$ dagegen nicht definiert, so stoppt P nicht.

Folgerung:

Jede LOOP-berechenbare Funktion ist WHILE-berechenbar.

1. Beispiel einer WHILE-berechenbaren Funktion

Das WHILE-Programm

$$
\begin{array}{l}
x_3 := x_1 - 5; \\
\text{WHILE } x_3 \neq 0 \text{ DO } x_1 := x_1 + 1 \text{ END}; \\
\text{LOOP } x_1 \text{ DO } x_0 := x_0 + 1 \text{ END}; \\
\text{LOOP } x_2 \text{ DO } x_0 := x_0 + 1 \text{ END}
\end{array}
$$

berechnet die Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ vermöge

$$
f(x_1, x_2) = \begin{cases} x_1 + x_2 & \text{falls } x_1 \leq 5, \\ \textit{nicht definiert} & \text{sonst.} \end{cases}
$$

Folgerung: Es gibt WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.

2. Beispiel einer WHILE-berechenbaren Funktion

Das WHILE-Programm

$$
\begin{array}{l}
x_1 := x_1 + 1; \\
\text{WHILE } x_1 \neq 0 \text{ DO} \\
\qquad x_0 := x_0 + 1; \\
\qquad \text{LOOP } x_2 \text{ DO } x_1 := x_1 - 1 \text{ END} \\
\text{END}; \\
x_0 := x_0 - 1
\end{array}
$$

berechnet die ganzzahlige Division $\mathrm{div} : \mathbb{N}^2 \to \mathbb{N}$ vermöge

$$
x_1 \mathrm{div}\ x_2 = \begin{cases} \lfloor \frac{x_1}{x_2} \rfloor & \text{falls } x_2 > 0, \\ \textit{nicht definiert} & \text{sonst.} \end{cases}
$$

Äquivalenz von WHILE-Programmen und Turingmaschinen

Satz:

1. Jede WHILE-berechenbare Funktion ist Turing-berechenbar.
2. Jede Turing-berechenbare Funktion ist WHILE-berechenbar.

Simulation: WHILE-Programm durch Turingmaschine

Mehrband-Turingmaschinen können

- Wertzuweisungen ausführen (wobei ein Band einer Variablen entspricht),
- Konstanten addieren und subtrahieren,
- hintereinander ausgeführt werden,
- WHILE-Schleifen ausführen.

Damit kann man ein WHILE-Programm (mit k Variablen) durch eine (k-Band-)Turingmaschine simulieren.

Simulation: Turingmaschine durch WHILE-Programm – 1

Gegeben sei TM $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, \{z_k\})$, wobei $Z = \{z_1, z_2, \ldots, z_k\}$, $\Gamma = \{a_1, a_2, \ldots, a_m\}$. Sei außerdem b eine Zahl mit $b > m$.

Eine Turingmaschinen-Konfiguration

$$a_{i_1} a_{i_2} \ldots a_{i_p} z_\ell a_{j_1} a_{j_2} \ldots a_{j_q}$$

wird durch drei Programmvariablen x, y, z mit den Werten

$$x = (i_1 i_2 \ldots i_p)_b, \qquad y = (j_q j_{q-1} \ldots j_1)_b, \qquad z = \ell$$

repräsentiert; dabei bedeutet $(i_1 i_2 \ldots i_p)_b$ die Zahl $i_1 i_2 \ldots i_p$ in b-närer Darstellung, also

$$x = \sum_{\mu=1}^{p} i_\mu \cdot b^{p-\mu}, \qquad y = \sum_{\mu=1}^{q} j_\mu \cdot b^{\mu-1}$$

Simulation: Turingmaschine durch WHILE-Programm – 2

Stuktur des WHILE-Programmes mit Eingabe und Ausgabe auf y

$z := 1;$
$\text{WHILE } z < k \text{ DO}$
$\quad a := y \bmod b; z' = z;$
$\quad \text{IF}(z' = 1 \text{ AND } a = 1) \text{ THEN } P_{1,1} \text{ END};$
$\quad \text{IF}(z' = 1 \text{ AND } a = 2) \text{ THEN } P_{1,2} \text{ END};$
$\quad \vdots$
$\quad \text{IF}(z' = k-1 \text{ AND } a = m) \text{ THEN } P_{k-1,m} \text{ END};$
END

Das Teilprogramm $P_{i,j}$ simuliert die Konfigurationsänderung für Zustand z_i und Bandsymbol a_j.

Simulation: Turingmaschine durch WHILE-Programm – 3

Stuktur des Teilprogrammes $P_{i,j}$ für $\delta(z_i, a_j) = (z_{i'}, a_{j'}, L)$

$$
\begin{aligned}
&z := i';\\
&y := y \text{ div } b;\\
&y := b * y + j';\\
&y := b * y + (x \bmod b);\\
&x := x \text{ div } b
\end{aligned}
$$

Entsprechend kann man sich die anderen Fälle vorstellen.

Die Churchsche These

Jede intuitiv berechenbare Funktion ist Turing-berechenbar.

- Die Churchsche These kann naturgemäß nicht bewiesen werden.
- Sie wird aber durch die Tatsache gestützt, dass zahlreiche weitere Modelle der Berechenbarkeit äquivalent zur Turing-Berechenbarkeit sind, z.B.
  - Post- und Markov-Algorithmen,
  - Registermaschinen,
  - partiell-rekursive Funktionen.