

LOOP-Programme: Syntaktische Komponenten

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

- **Variablen:** x_0 x_1 x_2 ...
- **Konstanten:** 0 1 2 ...
- **Operationssymbole:** + -
- **Trennsymbole:** ; :=
- **Schlüsselwörter:** LOOP DO END

LOOP-Programme: Syntax

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

(i) Jede **Wertzuweisung** der Form

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

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

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

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

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 **LOOP** x_i **DO** P_1 **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 $f : \mathbb{N}^k \rightarrow \mathbb{N}$, $k \in \mathbb{N}$, heißt **LOOP-berechenbar**, falls es ein LOOP-Programm P gibt, das f in dem Sinne berechnet, dass P ,

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

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

Schreibweise: $f = f_P^k$

Erstes Beispiel einer LOOP-berechenbaren Funktion

Gegeben sei das LOOP-Programm

```
 $x_0 := x_1 + 0;$   
LOOP  $x_2$  DO  $x_0 := x_0 + 1$  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

$$+ : \mathbb{N}^2 \rightarrow \mathbb{N} \text{ verm\"oge } (x_1, x_2) \mapsto +(x_1, x_2) = x_1 + x_2.$$

Also ist die **Addition LOOP-berechenbar**.

Zweites Beispiel einer LOOP-berechenbaren Funktion

Gegeben sei das LOOP-Programm

```
LOOP  $x_2$  DO
    LOOP  $x_1$  DO  $x_0 := x_0 + 1$  END
END
```

Eine genaue Betrachtung des Programms zeigt, dass damit die Funktion

$$\cdot : \mathbb{N}^2 \rightarrow \mathbb{N} \text{ verm\"oge } (x_1, x_2) \mapsto \cdot(x_1, x_2) = x_1 \cdot x_2,$$

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

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

Drittes Beispiel einer LOOP-berechenbaren Funktion

Das Konstrukt

IF $x_1 = 0$ THEN A ELSE B END

wird durch das LOOP-Programm

```
 $x_2 := 1; x_3 := 0;$   
LOOP  $x_1$  DO  $x_2 := 0; x_3 := 1$  END;  
LOOP  $x_2$  DO  $A$  END;  
LOOP  $x_3$  DO  $B$  END
```

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 x_1 x_2 ...
- **Konstanten:** 0 1 2 ...
- **Trennsymbole:** ; := \neq
- **Operationssymbole:** + -
- **Schlüsselwörter:** LOOP WHILE DO END

WHILE-Programme: Syntax

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

(i) Jede Wertzuweisung der Form

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

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

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

$P_1; P_2$ und **LOOP** x_i **DO** P_1 **END** und **WHILE** $x_i \neq 0$ **DO** P_1 **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 **LOOP** x_i **DO** P_1 **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 **WHILE** $x_i \neq 0$ **DO** P_1 **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_i statt.

WHILE-berechenbare Funktionen

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

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

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

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

Schreibweise: $f = f_P^k$

Folgerung: Jede LOOP-berechenbare Funktion ist WHILE-berechenbar.

1. Beispiel einer WHILE-berechenbaren Funktion

Das WHILE-Programm

```
 $x_3 := x_1 - 5;$   
WHILE  $x_3 \neq 0$  DO  $x_1 := x_1 + 1$  END;  
LOOP  $x_1$  DO  $x_0 := x_0 + 1$  END;  
LOOP  $x_2$  DO  $x_0 := x_0 + 1$  END
```

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

```
 $x_1 := x_1 + 1;$   
WHILE  $x_1 \neq 0$  DO  
     $x_0 := x_0 + 1;$   
    LOOP  $x_2$  DO  $x_1 := x_1 - 1$  END  
END;  
 $x_0 := x_0 - 1$ 
```

berechnet die **ganzzahlige Division** $\text{div} : \mathbb{N}^2 \rightarrow \mathbb{N}$ vermöge

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

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 $Z = \{z_1, z_2, \dots, z_k\}$, $\Gamma = \{a_1, a_2, \dots, a_m\}$. Sei außerdem b eine Zahl mit $b > m$.

Eine **Turingmaschinen-Konfiguration**

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

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

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

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

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

Simulation: Turingmaschine durch WHILE-Programm – 2

Struktur des WHILE-Programmes mit Eingabe und Ausgabe auf y

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

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

Simulation: Turingmaschine durch WHILE-Programm – 3

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

```
z := i';  
y := y div b;  
y := b * y + j';  
y := b * y + (x mod b);  
x := x div b
```

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.