

1.2 WHILE-Berechenbarkeit

WHILE-Programme sind wie folgt definiert:

Variablen: x_1, x_2, x_3, \dots

Konstanten: $0, 1, 2, \dots$

Trennsymbole: $;$ $:=$

Operatoren: $+$ $-$ \neq

Schlüsselwörter: WHILE DO END

Der Aufbau von WHILE-Programmen:

- $x_i := c$, $x_i := x_j + c$, $x_i := x_j - c$ sind WHILE-Programme
Die Interpretation dieser Ausdrücke erfolgt, wie üblich, mit der Einschränkung, dass $x_j - c$ als 0 gewertet wird, falls $c > x_j$.
- Sind P_1 und P_2 WHILE-Programme, so ist auch

$$P_1; P_2$$

ein WHILE-Programm.

Interpretation: Führe zuerst P_1 und dann P_2 aus.

- Ist P ein WHILE-Programm, so ist auch

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

ein WHILE-Programm.

Interpretation: Führe P solange aus, bis x_i den Wert 0 hat.

Achtung: Zuweisungen an x_i im Innern von P beeinflussen dabei den Wert von x_i !

Satz 117

Ist eine Funktion WHILE-berechenbar, so ist sie auch Turing-berechenbar.

Beweis:

Die Turingmaschine merkt sich den Programmzähler des WHILE-Programms sowie die aktuellen Werte aller Variablen. Der Platz dafür muss notfalls immer wieder angepasst werden. □

Wir werden später auch die umgekehrte Aussage des obigen Satzes zeigen.

1.3 GOTO-Berechenbarkeit

GOTO-Programme sind wie folgt definiert:

Variablen: x_1, x_2, x_3, \dots

Konstanten: 0, 1, 2, ...

Trennsymbole: ; :=

Operatoren: + - =

Schlüsselwörter: IF THEN GOTO HALT

Ein GOTO-Programm ist eine Folge von markierten Anweisungen

$$M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$$

wobei die A_i Anweisungen und die M_i Zielmarken für Sprünge sind.

Als Anweisungen können auftreten:

- Wertzuweisung: $x_i := x_j \pm c$
- unbedingter Sprung: GOTO M_i
- bedingter Sprung: IF $x_j = c$ THEN GOTO M_i
- Stopanweisung: HALT

Dabei ist c jeweils eine (beliebige) Konstante.

Satz 118

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

Beweis:

Ersetze jede WHILE-Schleife WHILE $x_i \neq 0$ DO P END durch folgendes Konstrukt:

$$\begin{aligned} M_1: & \text{ IF } x_i = 0 \text{ THEN GOTO } M_2 \\ & \quad P; \\ & \quad \text{GOTO } M_1 \\ M_2: & \dots \end{aligned}$$


Satz 119

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis:

Gegeben sei das GOTO-Programm

$$M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$$

Wir simulieren dies durch ein WHILE-Programm mit genau einer WHILE-Schleife:

```
c := 1;
WHILE c ≠ 0 DO
  IF c = 1 THEN A'_1 END;
  IF c = 2 THEN A'_2 END;
  ⋮
  IF c = k THEN A'_k END;
END
```

Beweis:

wobei

$$A'_i := \begin{cases} x_j := x_l \pm b; c := c + 1 & \text{falls } A_i = x_j := x_l \pm b \\ c := \ell & \text{falls } A_i = \text{GOTO } M_\ell \\ \text{IF } x_j = b \text{ THEN } c := \ell & \text{falls } A_i = \text{IF } x_j = b \text{ THEN} \\ \text{ELSE } c := c + 1 \text{ END} & \text{GOTO } M_\ell \\ c := 0 & \text{falls } A_i = \text{HALT} \end{cases}$$

Es bleibt als Übungsaufgabe überlassen, die IF-Anweisungen ebenfalls durch WHILE-Schleifen zu ersetzen. □

Satz 120

Aus Turing-Berechenbarkeit folgt GOTO-Berechenbarkeit.

Beweis:

Die Konfiguration (α, q, β) einer (det.) 1-Band-TM wird in den Variablen x_l, x_Q, x_r codiert. Die Zeichenreihen α und β werden als Zahlen (zu einer geeigneten Basis) aufgefasst, mit der niedrigstwertigen Ziffer bei der Position des Lese-/Schreibkopfes.

Jedes Tupel der Übergangsfunktion der TM wird durch ein geeignetes kleines Programmstück simuliert. Dabei werden Operationen wie Multiplikation mit, Division durch und Modularechnung zur Basis benötigt. □

1.4 Primitiv-rekursive Funktionen

Betrachte die kleinste Klasse der Funktion $\mathbb{N}_0^k \rightarrow \mathbb{N}_0$, für die gilt:

- 1 Sie enthält die konstanten Funktionen.
- 2 Sie enthält die Nachfolgerfunktionen: $n \mapsto n + 1$.
- 3 Sie enthält die Projektionsfunktionen:

$$\text{proj}_{k,i} : \mathbb{N}_0^k \ni (x_1, \dots, x_k) \mapsto x_i \in \mathbb{N}_0$$

- 4 Sie ist abgeschlossen unter Komposition.
- 5 Sie ist abgeschlossen unter primitiver Rekursion, d.h. mit

$$g : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$$

$$h : \mathbb{N}_0^{n+2} \rightarrow \mathbb{N}_0$$

ist auch

$$f(0, y_1, \dots, y_n) := g(y_1, \dots, y_n)$$

$$f(m+1, y_1, \dots, y_n) := h(f(m, y_1, \dots, y_n), m, y_1, \dots, y_n)$$

(primitive Rekursion) in der Klasse (und sonst nichts).

Die soeben definierte Funktionenklasse sind die **primitiv-rekursiven** Funktionen.

Beispiel 121

Die folgenden Funktionen sind primitiv-rekursiv:

- 1 $(x, y) \mapsto x + y$;
- 2 $(x, y) \mapsto x * y$;
- 3 $(x, y) \mapsto \max\{x - y, 0\}$;
- 4 $x \mapsto 2^x$;
- 5 $x \mapsto 2^{2^{\dots^2}}$ (Turm der Höhe x).

Satz 122

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Satz 123

Jede primitiv-rekursive Funktion ist berechenbar.

Beweis:

Induktion über den Aufbau einer primitiv-rekursiven Funktion. □

Korollar 124

*Die primitiv-rekursiven Funktionen sind eine **echte** Teilklasse der berechenbaren totalen Funktionen.*

Es gibt nicht-totale berechenbare Funktionen.

Definition 125

Sei $P(x)$ ein Prädikat, d.h. ein logischer Ausdruck, der in Abhängigkeit von $x \in \mathbb{N}_0$ den Wert **true** oder **false** liefert. Dann können wir diesem Prädikat in natürlicher Weise eine 0-1 Funktion

$$\hat{P} : \mathbb{N}_0 \rightarrow \{0, 1\}$$

zuordnen, indem wir definieren, dass $\hat{P}(x) = 1$ genau dann, wenn $P(x) = \mathbf{true}$ ist.

Wir nennen $P(x)$ **primitiv-rekursiv** genau dann, wenn $\hat{P}(x)$ primitiv-rekursiv ist.

Definition 126

Beschränkter max-Operator: Zu einem Prädikat $P(x)$ definieren wir

$$q : \mathbb{N}_0 \rightarrow \mathbb{N}_0$$

$$n \mapsto \begin{cases} 0 & \text{falls } \neg P(x) \text{ für alle } x < n \\ \max\{x < n; P(x)\} & \text{sonst} \end{cases}$$

Dann gilt: Ist P primitiv-rekursiv, so auch q , denn:

$$\begin{aligned} q(0) &= 0 \\ q(n+1) &= \begin{cases} n & \text{falls } P(n) \\ q(n) & \text{sonst} \end{cases} \\ &= q(n) + \hat{P}(n) * (n - q(n)) \end{aligned}$$

Definition 127

Beschränkter Existenzquantor: Zu einem Prädikat $P(x)$ definieren wir ein neues Prädikat $Q(x)$ mittels:

$Q(n)$ ist genau dann **true**, wenn ein $x < n$ existiert, so dass $P(x) = \mathbf{true}$.

Dann gilt: Ist P primitiv-rekursiv, so auch Q , denn:

$$\begin{aligned}\hat{Q}(0) &= 0 \\ \hat{Q}(n+1) &= \hat{P}(n) + \hat{Q}(n) - \hat{P}(n) * \hat{Q}(n)\end{aligned}$$

Beispiel 128

Zur bijektiven Abbildung von 2-Tupeln (bzw. n -Tupeln bei iterierter Anwendung der Paarbildung) natürlicher Zahlen in die Menge der natürlichen Zahlen verwendet man eine **Paarfunktion**, z.B.:

	0	1	2	3	4	...	n_2
0	0	2	5	9	14		
1	1	4	8	13			
2	3	7	12				
3	6	11					
⋮							
n_1							

Beispiel 128

Zur bijektiven Abbildung von 2-Tupeln (bzw. n -Tupeln bei iterierter Anwendung der Paarbildung) natürlicher Zahlen in die Menge der natürlichen Zahlen verwendet man eine **Paarfunktion**, z.B.:

$$\text{Betrachte: } p : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0 \quad \text{mit}$$
$$p(n_1, n_2) := \frac{(n_1+n_2)(n_1+n_2+1)}{2} + n_2$$

$$c_1 : \mathbb{N}_0 \rightarrow \mathbb{N}_0$$
$$c_1(n) := s - \left(n - \frac{s(s+1)}{2} \right); \quad \text{wobei}$$
$$s := \max \left\{ i; \frac{i(i+1)}{2} \leq n \right\}$$

$$c_2 : \mathbb{N}_0 \rightarrow \mathbb{N}_0$$
$$c_2(n) := n - \frac{s(s+1)}{2}, \quad s \text{ wie oben}$$

Satz 129

- ① p stellt eine primitiv-rekursive, bijektive Paarfunktion von \mathbb{N}_0^2 nach \mathbb{N}_0 mit den Umkehrfunktionen c_1 und c_2 dar.
- ② Die Umkehrfunktionen c_1, c_2 sind ebenfalls primitiv-rekursiv.

Beweis:

Übungsaufgabe.

