$P_1; P_2$ wird so interpretiert, daß zuerst P_1 und dann P_2 ausgeführt wird.

LOOP x_i DO P END wird so interpretiert, daß P sooft ausgeführt wird, wie der Wert der Variable x_i zu Beginn angibt.

Das Resultat ergibt sich als Wert von x_0.

Definition.
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt *LOOP–berechenbar*, falls es ein LOOP–Programm P gibt, das gestartet mit $n_1, \ldots, n_k$ in den Variablen $x_1, \ldots, x_k$ mit dem Wert $f(n_1, \ldots, n_k)$ in x_0 stoppt.

Bemerkung
Alle LOOP–berechenbaren Funktionen sind total definiert.

- IF $x = 0$ THEN A END

  kann simuliert werden durch

  $y := 1$;
  LOOP x DO $y := 0$ END;
  LOOP y DO A END

Beispiel.

- Die Addition "$x_0 := x_1 + x_2$" ist LOOP–berechenbar:

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

  mit Verallgemeinerung auf "$x_0 := x_i + x_j$"

- Die Multiplikation "$x_0 := x_1 * x_2$" ist LOOP–berechenbar:

  $x_0 := 0$;
  LOOP x_2 DO $x_0 := x_0 + x_1$ END

Man kann auch DIV, MOD durch LOOP–Programme beschreiben
$\leadsto$ x:=(y DIV z) + (x MOD 5) * z

Erweiterung der LOOP–Programme durch die WHILE–Schleife
Ergebnis: WHILE–Programme:

- Ist P ein WHILE–Programm, x_i Variable, dann ist auch

  WHILE $x \neq 0$ DO P END;

  ein WHILE–Programm.

Semantik.
P wird solange ausgeführt, wie $x_i \neq 0$ gilt.

Man kommt in WHILE–Programmen ohne LOOP aus:

LOOP x DO P END;

wird simuliert durch

$y := x$;
WHILE $y \neq 0$ DO $y := y - 1; P$ END;

Definition.
Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt *WHILE–berechenbar*, falls es ein WHILE–Programm P gibt, das gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 sonst) mit dem Wert $f(n_1, \ldots, n_k)$ in x_0 stoppt.
Sonst stoppt P nicht.

Satz.
TM können WHILE–Programme simulieren.
D.h. jede WHILE–berechenbare Funktion ist auch TM –berechenbar.

Beweis.
Man kann die Wertzuweisungen, Sequenzen und WHILE–Schleifen mit einer Mehrband TM simulieren.
(Wobei das i-te Band der i-ten Variablen (binär) entspricht.)
Man kann eine Mehrband TM mit einer 1–Band TM simulieren. ■

Beweis der Umkehrung:
Betrachte GOTO–Programme.

GOTO–Programme bestehen aus Folgen von markierten Anweisungen

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

Als Anweisungen sind zugelassen:

Wertzuweisungen: $x_i := x_j \pm c$
unbedingter Sprung: GOTO M_i
bedingter Sprung: IF x_i=c THEN GOTO M_i

Stopanweisung: HALT

Vereinbarung: Marken von nicht angesprungen Anweisungen werden weggelassen.
Die Semantik ist klar.

Definition.
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt *GOTO–berechenbar*, falls es ein GOTO–Programm P gibt, das gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 sonst) mit dem Wert $f(n_1, \ldots, n_k)$ in x_0 stoppt.
Sonst stoppt P nicht.

Satz.
Jedes WHILE–Programm kann durch ein GOTO–Programm simuliert werden.

Beweis.
WHILE $x_i \neq 0$ DO P END

kann simuliert werden durch

M_1: IF $x_i = 0$ THEN GOTO M_2;
  P;
  GOTO M_1;
M_2: ... ■

Korollar
Jede WHILE–berechenbare Funktion ist GOTO–berechenbar.

Satz.
Jedes GOTO–Programm kann durch ein WHILE–Programm (mit nur einer WHILE–Schleife) simuliert werden.

Beweis.
Gegeben sei ein GOTO–Programm

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

wird simuliert durch folgendes WHILE–Programm mit nur einer WHILE–Schleife:

count:=1;
WHILE count $\neq 0$ DO
  IF $count = 1$ THEN A'_1 END;
  IF $count = 2$ THEN A'_2 END;

$\vdots$
IF $count = k$ THEN A'_k END;
END

wobei

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

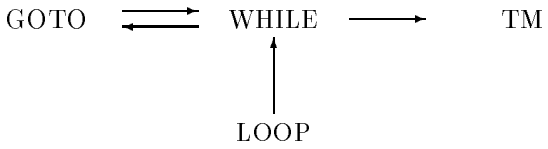
■

Korollar
Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist GOTO–berechenbar, falls f WHILE–berechenbar ist.

Satz. *(Kleenesche Normalformsatz für WHILE–Programme)*
Jede WHILE–berechenbare Funktion $f : \mathbb{N}^k \to \mathbb{N}$ kann durch ein WHILE–Programm mit nur einer WHILE–Schleife berechnet werden.

Beweis.
Sei P ein beliebiges WHILE–Programm für f.
P wird simuliert durch das GOTO–Programm P'.
P' wird simuliert durch das WHILE–Programm P'' mit nur einer WHILE–Schleife. ■

Bisher ist gezeigt worden:



Satz.
GOTO–Programme können TM simulieren.

Beweis.
Sei $M = (Q, \Sigma, \Gamma, \delta, q_1, \Box, F)$ eine TM die f berechnet.
Das simulierende GOTO–Programm hat folgende Gestalt:
$M_1 : A_1 \ ; \ M_2 : A_2 \ ; \ M_3 : A_3$

wobei:

P_1 : transformiert die gegebenen Anfangswerte in Binärdarstellung und erzeugt eine Darstellung der Starkonfiguration von M in die Variablen x, y, z.
P_2 : simuliert die Rechnung von M Schritt-für-Schritt durch Veränderung der Variablenwerte x, y, z.
P_3 : erzeugt aus der kodierten Form der Endkonfiguration in x, y, z die eigentliche Ausgabe in der Ausgabevariable x_0.

Bemerkung
P_1, P_3 hängen nicht von M ab, lediglich von P_2.

Kodierung:
Sei

$$Q = \{q_1, \ldots, q_k\}$$
$$\Gamma = \{a_1, \ldots, a_m\} \ , \ b > \#\Gamma$$

Kodierung der TM –Konfiguration:

$$a_{i_1} \ldots a_{i_p} q_l a_{j_1} \ldots a j_q$$

durch die folgenden Werte von x, y, z:

$$x = (i_1 \ldots i_p)_b$$
$$y = (j_q \ldots j_1)_b$$
$$z = l$$

GOTO–Programmstück $M_2 : P_2$:

M_2: $a := y$ MOD b;
IF $(z = 1)$ AND $(a = 1)$ THEN GOTO M_{11}
IF $(z = 1)$ AND $(a = 2)$ THEN GOTO M_{12}
$\vdots$
IF $(z = k)$ AND $(a = m)$ THEN GOTO M_{km}
M_{11}: $\otimes$
GOTO M_2
M_{12}: $\otimes$
GOTO M_2
$\vdots$
M_{km}: $\otimes$
GOTO M_2

wobei $\otimes$ bedeutet:
Man nimmt das Programmstück das mit M_{ij} markiert ist

Sei $\delta(q_i, a_j) = (q_i, a_j, L)$

und simuliert den Übergang durch:

$z := i';$
$y := y \text{ DIV } b;$
$y := b * y + j';$
$y := b * y + (x \text{ MOD } b);$
$x := x \text{ DIV } b;$

ist q_i Endzustand, kann man

für $\otimes$ GOTO M_3 setzen.

Der Rest ist klar. ■

Korollar
Eine Funktion ist
TM –berechenbar $\Leftrightarrow$ GOTO–berechenbar $\Leftrightarrow$ WHILE–berechenbar