Registermaschinen

- "Realistischeres" Maschinenmodell als die Turing-Maschine.
- Begriff wurde 1961 von Marvin Minsky eingeführt (Annals of Mathematics, Bd. 74).
- Abarbeitung von Programmen in elementarer, imperativer Programmiersprache.

Definition: Eine **Registermaschine (random access machine, RAM)** besteht aus folgenden Elementen:

- endlich vielen Registern x_i, $i \geq 0$;
  - jedes Register kann eine beliebig große natürliche Zahl aufnehmen;
  - zu Beginn enthalten die Register standardmäßig Zahl 0, sofern nicht ein Initialwert explizit zugeordnet wurde;
- ein LOOP-, WHILE- oder GOTO-Programm.

LOOP-Programme

Definition: Ein **LOOP-Befehl** bzw. ein **LOOP-Programm** ist induktiv wie folgt definiert:

- Induktionsbeginn: Für jedes Register x_i ist
  - $x_i := x_i + 1$,
  - $x_i := x_i - 1$

  sowohl ein LOOP-Befehl als auch ein LOOP-Programm.
- Induktionsschritt: Sind P_1 und P_2 LOOP-Programme, dann gilt:
  - $P_1; P_2$ ist ein LOOP-Programm;
  - LOOP x_i DO P_1 END ist sowohl ein LOOP-Befehl als auch ein LOOP-Programm, für jedes Register x_i.

Ein Befehl wird auch **Programmzeile** genannt.

Definition (Semantik von LOOP-Programmen): Sei M eine Registermaschine. M führt LOOP-Programme folgendermaßen aus:

$x_i := x_i + 1$: M inkrementiert den Inhalt des Registers x_i.

$x_i := x_i - 1$: Falls $x_i > 0$, so dekrementiert M den Inhalt von x_i; ansonsten enthält x_i weiterhin den Wert 0.

LOOP x_i DO P END : M führt P n-mal hintereinander aus, wenn n der Inhalt von x_i vor Beginn der ersten Ausführung des Schleifenrumpfes ist.

$P_1; P_2$: M führt zunächst P_1 aus und dann, unter Übernahme aller aktuellen Registerwerte, das Programm P_2.

Wenn keine nächste auszuführende Programmzeile existiert, dann bricht M die Programmausführung ab.

Definition: Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt **LOOP-berechenbar**, falls es eine Registermaschine M mit LOOP-Programm P gibt, s.d. für alle $(n_1, \ldots, n_k) \in \mathbb{N}^k$ und alle $m \in \mathbb{N}$ gilt:

$f(n_1, \ldots, n_k) = m \iff$ Wenn M gestartet wird mit

- $x_i = n_i$, für $1 \leq i \leq k$ und
- $x_i = 0$ für $i > k$,

so bricht die Programmausführung schließlich ab mit

- $x_i = n_i$, für $1 \leq i \leq k$,
- $x_{k+1} = m$ und
- $x_i = 0$, für $i > k + 1$.

Wir bezeichnen die Menge aller LOOP-berechenbaren Funktionen mit $LOOP$.

Falls M (mit Programm P) den Funktionswert $f(n_1, \ldots, n_k) = m$ berechnet, dann ist $n_1, \ldots, n_k$ der **Input** und m der **Output** von M (bzw. P).

Beachte:

- Die Input-Argumente $n_1, \ldots, n_k$ müssen laut Definition am Ende der Rechnung in den ersten k Registern vorhanden sein.

  $\Longrightarrow$ Anderenfalls berechnet ein LOOP-Programm keine Funktion!

Beispiel: Das Programm P, gegeben durch

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

berechnet keine Funktion:

- es gilt nach Programmende immer $x_1 = 0$ und $x_2 = 1$.

  $\Longrightarrow$ P kann keine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ berechnen, egal wie k gewählt wird!

Satz: Jede LOOP-berechenbare Funktion ist total.

Beweis:

- Jedes LOOP-Programm hat nur endlich viele Zeilen, und
- jede Schleife LOOP x_i DO P END wird nur endlich oft durchlaufen.

$\Longrightarrow$ Es gibt kein Programmkonstrukt, das in eine Endlosschleife geraten kann.

$\Longrightarrow$ Wenn ein LOOP-Programm eine Funktion berechnet, muß diese total sein. ■

Im folgenden zeigen wir, wie man mit Hilfe der elementaren Operationen von LOOP-Programmen komplexere Befehle simulieren kann.

Dafür ist folgende Bemerkung relevant:

- Für ein Programm kann man immer die Menge der Register ermitteln, auf die zugegriffen wird.

  $\Longrightarrow$ Es gibt ein eindeutiges Register x_ℓ das den höchsten Index aller verwendeten Register hat.

Zusatzbefehle für LOOP-Programme:

Die folgenden Befehle können mittels der angegebenen LOOP-Befehle simuliert werden.

Dazu seien werden die folgenden Annahmen getroffen:

- $\{x_1, \ldots, x_\ell\}$ ist die Menge der verwendeten Register eines gegebenen LOOP-Programms.
- Register x_n und x_{n+1} sind jeweils neue (und daher mit 0 belegte) Register mit $n > \ell$.
- c ist eine Konstante aus $\mathbb{N}$.

1. $x_i := c$:

   % x_i auf 0 setzen

   LOOP x_i DO $x_i := x_i - 1$ END;

   $x_i := x_i + 1$;
   $\vdots$
   $x_i := x_i + 1$ } c mal

2. $x_i := x_j \pm c$:

   % x_n auf x_j setzen

   LOOP x_j DO $x_n := x_n + 1$ END;
   $x_i := 0$;
   LOOP x_n DO $x_i := x_i + 1$ END;
   $x_n := c$;
   LOOP x_n DO $x_i := x_i \pm 1$ END;
   $x_n := 0$

3. IF $x_i = 0$ THEN P_1 ELSE P_2 END :

   % Falls $x_i = 0$, soll $x_n = 1$ und
   % $x_{n+1} = 0$ sein, sonst umgekehrt.

   $x_n := 1$; $x_{n+1} := 1$;
   LOOP x_i DO $x_n := x_n - 1$ END;
   LOOP x_n DO $x_{n+1} := x_{n+1} - 1$ END;
   % P_1 bzw. P_2 ausführen
   LOOP x_n DO P_1 END;
   LOOP x_{n+1} DO P_2 END;
   $x_n := 0$; $x_{n+1} := 0$

4. IF $x_i > c$ THEN P END :

   $x_n := x_i - c$;

   LOOP x_n DO $x_{n+1} := 1$ END;

   LOOP x_{n+1} DO P END;

   $x_n := 0$; $x_{n+1} := 0$

5. $x_i := x_j \pm x_k$:

   $x_i := x_j + 0$;

   LOOP x_k DO $x_i := x_i \pm 1$ END

6. $x_i := x_j * x_k$:

   $x_i := 0$;

   LOOP x_k DO $x_i := x_i + x_j$ END

7. NOP :

   % Dieser Befehl tut nichts.

   $x_n := x_n - 1$

8. $x_i := x_j$ DIV x_k; $x_{i'} := x_j$ MOD x_k :

   % x_n zählt jeweils bis x_k hoch und

   % x_{n+1} prüft ob $x_n = x_k$. x_i speichert

   % die Durchgänge wenn $x_n = x_k$.

   % Am Ende gilt $x_j = x_i \cdot x_k + x_{i'}$

   % mit $x_n = x_{i'}$.

   $x_i := 0$;

   LOOP x_j DO

     $x_n := x_n + 1$;

     $x_{n+1} := x_k - x_n$;

     IF $x_{n+1} = 0$

       THEN $x_i := x_i + 1$; $x_n := 0$

       ELSE NOP;

     END;

   END;

   $x_{i'} := x_n$;

   $x_n := 0$; $x_{n+1} := 0$

Weitere natürliche Varianten wie z.B. $x_i := x_j$ oder IF $x_i = c$ THEN P END ebenfalls definierbar.

WHILE-Programme

Definition: Ein **WHILE-Befehl** bzw. ein **WHILE-Programm** ist induktiv wie folgt definiert:

- Induktionsbeginn: Für jedes Register x_i ist
  - $x_i := x_i + 1$,
  - $x_i := x_i - 1$

  sowohl ein WHILE-Befehl als auch ein WHILE-Programm.
- Induktionsschritt: Sind P_1 und P_2 WHILE-Programme, dann gilt:
  - $P_1; P_2$ ist ein WHILE-Programm;
  - WHILE $x_i \neq 0$ DO P_1 END ist sowohl ein WHILE-Befehl als auch ein WHILE-Programm, für jedes Register x_i.

Definition (Semantik von WHILE-Programmen): Sei M eine Registermaschine. M führt WHILE-Programme folgendermaßen aus:

$x_i := x_i \pm 1$: Analog zur Ausführung in LOOP-Programmen.

WHILE $x_i \neq 0$ DO P END :

1. Falls der Wert in x_i ungleich 0 ist, führt M das Programm P aus, sonst geht sie zu 3.
2. M wiederholt 2.
3. M führt die Programmzeile nach dem WHILE-Befehl aus.

$P_1; P_2$ und Programmende: Analog zu LOOP-Programmen.

Definition: Eine Funktion $f : \mathbb{N}^* \to \mathbb{N}$ heißt **WHILE-berechenbar**, falls es eine Registermaschine M mit WHILE-Programm P gibt, s.d. für alle $(n_1, \ldots, n_k) \in \mathbb{N}^k$ und alle $m \in \mathbb{N}$ gilt:

1. $f(n_1, \ldots, n_k) = m \iff$ Wenn M gestartet wird mit
   - $x_i = n_i$, für $1 \leq i \leq k$ und
   - $x_i = 0$ für $i > k$,

   so bricht die Programmausführung schließlich ab mit
   - $x_i = n_i$, für $1 \leq i \leq k$,
   - $x_{k+1} = m$ und
   - $x_i = 0$, für $i > k + 1$.
2. $f(n_1, \ldots, n_k)$ ist undefiniert $\iff M$, gestartet mit
   - $x_i = n_i$, für $1 \leq i \leq k$ und
   - $x_i = 0$ für $i > k$,

   hält nie.

Wir bezeichnen die Menge aller totalen (bzw. partiellen) WHILE-berechenbaren Funktionen mit $WHILE$ (bzw. $WHILE^{part}$).

Satz: $LOOP \subseteq WHILE$.

Beweis: Wir müssen zeigen: jede LOOP-berechenbare Funktion ist WHILE-berechenbar.

Dazu zeigen wir: der LOOP-Befehl kann durch WHILE simuliert werden.

- Sei P ein LOOP-Programm das keine weiteren LOOP-Befehle enthält.
- Seien x_n und x_{n+1} neue (und somit mit 0 belegte) Register.

$\Longrightarrow$ LOOP x_i DO P END wird folgendermaßen simuliert:

% $x_n := x_i$ simulieren.

WHILE $x_i \neq 0$ DO

$x_n := x_n + 1;\ x_{n+1} := x_{n+1} + 1;\ x_i := x_i - 1$

END;

WHILE $x_{n+1} \neq 0$ DO

$x_i := x_i + 1;\ x_{n+1} := x_{n+1} - 1$

END;

% LOOP-Befehl selbst simulieren.

WHILE $x_n \neq 0$ DO $P;\ x_n := x_n - 1$ END ■

Bemerkungen:

1. Da der LOOP-Befehl mittels WHILE simuliert werden kann, können alle Zusatzbefehle auf Basis von LOOP auch in WHILE-Programmen verwendet werden.
2. Es gilt $WHILE \subset WHILE^{part}$.

   Beweis dazu: Das WHILE-Programm

   WHILE $x_1 \neq 0$ DO $x_1 := x_1 + 1$ END

   berechnet die nichttotale Funktion $f : \mathbb{N} \to \mathbb{N}$ mit

$$f(n) := \begin{cases} 0 & \text{falls } n = 0; \\ \text{undefiniert} & \text{falls } n \neq 0. \end{cases}$$

Problem: Sind alle *totalen* WHILE-berechenbaren Funktionen in $LOOP$, oder gilt $LOOP \subset WHILE$?

$\Rightarrow$ Wir werden später zeigen, daß es eine totale Funktion gibt, die in $WHILE$ liegt aber durch kein LOOP-Programm berechnet werden kann.

GOTO-Programme

Definition:

1. Ein **Index** ist eine natürliche Zahl $j > 0$.
2. Für jedes Register x_i und jeden Index j ist jeder der folgenden Ausdrücke ein **GOTO-Befehl**:
   - $x_i := x_i + 1$,
   - $x_i := x_i - 1$,
   - IF $x_i = 0$ GOTO j.
3. Ein **GOTO-Programm** ist induktiv wie folgt definiert:
   - Für jeden Index j und jeden GOTO-Befehl B ist $j : B$ ein GOTO-Programm.
   - Sind P_1 und P_2 GOTO-Programme, dann ist $P_1; P_2$ ein GOTO-Programm.

Vereinbarung:

- Programmzeilen seien mit Indizes so durchnumeriert, daß der i-ten Programmzeile der Index i voransteht.

Definition (Semantik von GOTO-Programmen): Sei M eine Registermaschine. M führt GOTO-Programme folgendermaßen aus:

$j : B$: Dies Programm wird ausgeführt wie B.

$x_i := x_i \pm 1$ und $P_1; P_2$: Analog zur Ausführung in LOOP- und WHILE-Programmen.

IF $x_i = 0$ GOTO j :

1. Falls der aktuelle Wert in x_i ungleich 0 ist:
   - M arbeitet in der Programmzeile nach diesem GOTO-Befehl weiter, falls es so eine Zeile gibt,
   - ansonsten bricht M die Programmausführung ab.
2. Falls der aktuelle Wert in x_i gleich 0 ist:
   - M führt als nächstes die Programmzeile aus, vor der der Index j steht,
   - falls es keine Zeile mit Index j gibt, bricht M die Programmausführung ab.

Definition: Eine Funktion $f : \mathbb{N}^* \to \mathbb{N}$ heißt **GOTO-berechenbar**, falls es eine Registermaschine M mit GOTO-Programm P gibt, s.d. die Bedingungen analog wie für WHILE-berechenbare Funktionen erfüllt sind.

Wir bezeichnen die Menge aller totalen (bzw. partiellen) GOTO-berechenbaren Funktionen mit $GOTO$ (bzw. $GOTO^{part}$).

Satz: Es gilt:

- $WHILE = GOTO$.
- $WHILE^{part} = GOTO^{part}$.

Beweis: Wir zeigen zunächst: jede WHILE-berechenbare Funktion ist GOTO-berechenbar.

Der einzige Befehl, der in WHILE-, aber nicht in GOTO-Programmen vorkommt, ist

WHILE $x_i \neq 0$ DO P END.

Wir simulieren diesen Befehls mittels einem GOTO-Programm.

- Es sei P ein WHILE-Programm das selbst keine weiteren WHILE-Befehle enthält.
- Weiters seien j_1, j_2, j_3 neue Indizes und x_n ein bislang unbenutztes Register.
- Dann wird WHILE $x_i \neq 0$ DO P END durch folgendes GOTO-Programm simuliert:

j_1 : IF $x_i = 0$ GOTO j_3;
$\hat{P}$;
% Unbedingter Sprung, da $x_n = 0$.
j_2 : IF $x_n = 0$ GOTO j_1;
% NOP, j_3 *ist nur Sprungziel.*
j_3 : $x_n := x_n - 1$

  - $\hat{P}$ bezeichnet das Programm, das entsteht, wenn in P jede Programmzeile mit einem Index versehen wird.

- Weiters gilt:
  - Die Eigenschaft der Totalität bleibt bei der Umformung enthalten.
    $\Longrightarrow$ Wenn das WHILE-Programm total war, so ist auch das simulierende Programm total.

Jetzt zeigen wir: Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

- Sei P ein GOTO-Programm der Form

1 : B_1;
2 : B_2;
$\vdots$
t : B_t.

- Wir simulieren P mittels eines WHILE-Programms das eine neue Variable x_{count} verwendet mit folgender Bedeutung:
  - falls $x_{count} = i$, dann soll in P als nächstes die Programmzeile $i : B_i$ ausgeführt werden.

- Das simulierende WHILE-Programm hat folgende Form:

  $x_{count} := 1;$
  WHILE $x_{count} \neq 0$ DO
    IF $x_{count} = 1$ THEN P'_1 END;
    IF $x_{count} = 2$ THEN P'_2 END;
    $\vdots$
    IF $x_{count} = t$ THEN P'_t END;
    IF $x_{count} > t$ THEN $x_{count} = 0$ END
  END

  wobei P'_i wie folgt definiert ist:

  – Falls B_i gleich $x_i := x_i \pm 1$, dann ist P'_i:

    $$x_i := x_i \pm 1;\ x_{count} := x_{count} + 1$$

  – Falls B_i gleich IF $x_i = 0$ GOTO j, dann ist P'_i:

    IF $x_i = 0$ THEN $x_{count} := j$
    ELSE $x_{count} := x_{count} + 1$ END;

- Die Simulation erhält wieder die Totalität. ■

Bemerkung:

- Aus diesem Resultat folgt wieder, daß die Zusatzbefehle auf Basis von LOOP auch in GOTO-Programmen verwendet werden können.

Im folgenden bezeichne WHILE^{if} die Erweiterung von WHILE-Programmen in denen Befehle der Form

$$\text{IF } x_i = n \text{ THEN } P \text{ END}$$

als primitive Ausdrücke vorkommen können.

Korollar: Jede WHILE-berechenbare Funktion läßt sich durch ein WHILE^{if}-Programm mit höchstens einer WHILE-Schleife berechnen.

Beweis:

- Aus vorigem Satz folgt, daß jedes WHILE-Programm W in ein äquivalentes GOTO-Programm G umgeschrieben werden kann.
- Laut zweitem Teil des Beweises desselben Satzes folgt, daß G in ein WHILE-Programm W' mit nur einer WHILE-Schleife und mehreren IF-Befehlen übersetzt werden kann. ■