

Vorlesung Theoretische Informatik II

Bernhard Beckert

Institut für Informatik



Wintersemester 2007/2008

Diese Vorlesungsmaterialien basieren zum Teil auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Christoph Kreitz (gehalten an der Universität Potsdam)

Ihnen gilt mein herzlicher Dank.

– Bernhard Beckert, Oktober 2007

Inhalt von Teil II

- Registermaschinen (Random Access Machines)
- LOOP-Programme
- WHILE-Programme
- GOTO-Programme
- Beziehungen zwischen LOOP, WHILE, GOTO
- Beziehungen zwischen Registermaschinen und Turingmaschinen

Teil II

Registermaschinen

1 Registermaschinen

2 LOOP-Programme

3 WHILE-Programme

4 GOTO-Programme

5 Beziehungen zwischen LOOP, WHILE, GOTO und Turingmaschinen

Im Vergleich zu Turingmaschinen

- Als Berechnungsmodell gleichwertige Grundlage für Berechenbarkeitstheorie (weniger auch Komplexitätstheorie)
- Vorteil: abstrakter, einfacher zu verstehende Programme

Ähnlich wie ...

- Imperativer Kern von Programmiersprachen
- Pseudo-Code

Beispiel Pseudo-Code

Berechnung von $a \bmod b$

```
r := a;  
while  $r \geq b$  do  
  r := r - b  
end;  
return r
```

Fragen

- Welche Konstrukte: if? while? goto?
- Welche Datentypen: Integers? Strings?
- Welche Datenstrukturen: Objekte? Arrays?
- Welche atomaren Befehle?
- Wie funktioniert Ein- und Ausgabe?

Festlegungen (informell)

Konstrukte: Unterschiedliche Varianten

loop oder while oder if+goto

Datentyp: Die natürlichen Zahlen

(wesentlicher Unterschied zu realen Computern)

Datenstrukturen: Unbeschränkte aber endliche Zahl von Registern

bezeichnet mit x_1, x_2, x_3, \dots ,

enthalten jeweils eine natürliche Zahl

(keine Arrays, Objekte o. ä)

Atomare Befehle: Inkrementieren und Dekrementieren eines Registers

Eingabe/Ausgabe: n Eingabewerte in den ersten n Registern

(alle anderen Register am Anfang 0),

Ausgabe im Register $n + 1$

Definition 4.1 (LOOP-Programm)

- $x_j := x_j + 1$
- $x_j := x_j - 1$

sind LOOP-Programme (und LOOP-Befehle) für alle Register x_j

Wenn P_1, P_2 LOOP-Programme sind, dann ist auch

- $P_1; P_2$

ein LOOP-Programm

Wenn P ein LOOP-Programm ist und x_j ein Register, dann ist

- `loop x_j do P end`

ein LOOP-Programm (und ein LOOP-Befehl)

Definition 4.2 (Zustand einer Registermaschine)

Ein Zustand s einer Registermaschine ist eine Abbildung

$$s : \{x_i \mid i \in \mathbb{N}\} \longrightarrow \mathbb{N}$$

die jedem Register eine natürliche Zahl als Wert zuordnet.

Registermaschinen: Definition

Definition 4.3 (Anfangszustand, Eingabe)

Seien natürliche Zahlen $m_1, \dots, m_k \in \mathbb{N}$ als Eingabe gegeben. Im Anfangszustand gilt:

- $x_i = m_i$ für $1 \leq i \leq k$
- $x_i = 0$ für $i > k$

Definition 4.4 (Ausgabe)

Terminiert eine Registermaschine, die mit Eingabe m_1, \dots, m_k gestartet wurde, in einem Zustand s_{term} , dann ist

$$s_{term}(x_{k+1})$$

die Ausgabe der Maschine.

Definition 4.5

Semantik einer Registermaschine Die Semantik $\Delta(P)$ einer Registermaschine P ist eine Relation

$$\Delta(P) : S \times S$$

auf der Menge aller Zustände.

Dabei bedeutet

$$\Delta(P)(s_1, s_2)$$

dass man durch Ausführen von P vom Zustand s_1 in den Zustand s_2 gelangt.

Definition 4.6

Eine Registermaschine P berechnet eine Funktion

$$f_P : \mathbb{N}^k \longrightarrow \mathbb{N}$$

genau dann, wenn für alle $m_1, \dots, m_k \in \mathbb{N}$ folgendes gilt:

Wenn man P im Anfangszustand für die Eingabe m_1, \dots, m_k startet, dann gilt:

- P terminiert genau dann, wenn $f(m_1, \dots, m_k)$ definiert ist
- Falls P terminiert, dann ist die Ausgabe $f(m_1, \dots, m_k)$
- (siehe nächste Folie)

Besonderheit

Wir verlangen zusätzlich, dass eine Registermaschine, wenn sie terminiert, alle Register außer dem Ausgaberegister wieder auf ihren ursprünglichen Wert zurücksetzt (oder unverändert lässt):

- Eingaberegister x_1, \dots, x_k auf die Eingabewerte
- Register x_i für $i > k + 1$ auf 0

Folge

Eine Maschine, die auch nur für manche Eingaben diese Zusatzbedingung verletzt, berechnet gar keine Funktion.

Definition 4.7 (Berechenbare Funktion)

Eine Funktion f heißt

LOOP-berechenbar, wenn es eine Registermaschine mit einem LOOP-Programm gibt, die f berechnet

WHILE-berechenbar, wenn es eine Registermaschine mit einem WHILE-Programm gibt, die f berechnet

GOTO-berechenbar, wenn es eine Registermaschine mit einem GOTO-Programm gibt, die f berechnet

TM-berechenbar, wenn es eine Turingmaschine gibt, die f berechnet

LOOP = Menge aller LOOP-berechenbaren Funktionen

WHILE = Menge aller WHILE-berechenbaren Funktionen

GOTO = Menge aller GOTO-berechenbaren Funktionen

TM = Menge aller TM-berechenbaren Funktionen

Teil II

Registermaschinen

- 1 Registermaschinen
- 2 LOOP-Programme**
- 3 WHILE-Programme
- 4 GOTO-Programme
- 5 Beziehungen zwischen LOOP, WHILE, GOTO und Turingmaschinen

Definition 5.1 (Semantik von LOOP-Programmen)

Sei P ein LOOP-Programm.

Induktive Definition von $\Delta(P)$ wie folgt:

$\Delta(x_i := x_i + 1)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} s_2(x_i) &= s_1(x_i) + 1 \\ s_2(x_j) &= s_1(x_j) \text{ f\"ur } j \neq i \end{aligned}$$

$\Delta(x_i := x_i - 1)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} s_2(x_i) &= \begin{cases} s_1(x_i) - 1 & \text{falls } s_1(x_i) > 0 \\ 0 & \text{falls } s_1(x_i) = 0 \end{cases} \\ s_2(x_j) &= s_1(x_j) \text{ f\"ur } j \neq i \end{aligned}$$

$\Delta(P_1; P_2)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} &\Delta(P_1)(s_1, s') \\ &\Delta(P_2)(s', s_2) \end{aligned}$$

Definition 5.2 (Semantik von LOOP-Programmen (Forts.))

$\Delta(\text{loop } x_i \text{ do } P \text{ end})(s_1, s_2)$ genau dann, wenn:

es gibt Zustände s'_0, \dots, s'_n mit

$$- s_1(x_i) = n$$

$$- s_1 = s'_0$$

$$- s_2 = s'_n$$

$$- \Delta(P)(s'_k, s'_{k+1}) \quad \text{für } 0 \leq k < n$$

Merke

Die Anzahl der Schleifendurchläufe ist der Wert von x_i
zu Beginn der Schleife.

Etwaige Änderungen an x_i innerhalb der Schleifendurchläufe sind **irrelevant**

Theorem 5.3

Jedes LOOP-Programm terminiert für jede Eingabe.

Beweisskizze

Beweis durch Induktion über den Aufbau von Programmen, dass jedes Programm terminiert.

Induktionsanfang: Atomare Programme terminieren.

Induktionsschritt für Konkatination: einfach.

Induktionsschritt für Schleife: Da Anzahl der Durchläufe zu Anfang feststeht kann es keine Endlosschleife geben.

Korollar

Jede LOOP-berechenbare Funktion ist total.

LOOP: Definition von Zusatzbefehlen

$x_j := 0$

loop x_j do $x_j := x_j - 1$ end

$x_j := c$ für $c \in \mathbb{N}$

$x_j := 0;$
 $x_j := x_j + 1;$
 \vdots
 $x_j := x_j + 1;$ } c mal

$x_j := x_j$

loop x_j do $x_n := x_n + 1$ end;

$x_j := 0;$

loop x_n do $x_i := x_i + 1$ end

LOOP: Definition von Zusatzbefehlen

Im folgenden bezeichnen x_n, x_{n+1}, \dots
neue, bisher nicht verwendete Register

$x_i := e_1 \pm e_2$ (e_1, e_2 arithmetische Ausdrücke)

$x_i := e_1$;

$x_n := e_2$;

loop x_n do $x_i := x_i \pm 1$ end;

$x_n := 0$

$x_i := e_1 * e_2$ (e_1, e_2 arithmetische Ausdrücke)

$x_i := 0$;

$x_n := e_1$;

loop x_n do $x_i := x_i + e_2$ end;

$x_n := 0$

LOOP: Definition von Zusatzbefehlen

```
if  $x_i = 0$  then  $P_1$  else  $P_2$  end
```

```
 $x_n := 1 - x_i$ ;
```

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

```
loop  $x_n$  do  $P_1$  end;
```

```
loop  $x_{n+1}$  do  $P_2$  end
```

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

```
if  $x_i \leq x_j$  then  $P_1$  else  $P_2$  end
```

```
 $x_n := x_i - x_j$ ;
```

```
if  $x_n = 0$  then  $P_1$  else  $P_2$  end
```

```
 $x_n := 0$ 
```

Teil II

Registermaschinen

- 1 Registermaschinen
- 2 LOOP-Programme
- 3 WHILE-Programme**
- 4 GOTO-Programme
- 5 Beziehungen zwischen LOOP, WHILE, GOTO und Turingmaschinen

Definition 6.1 (WHILE-Programm)

- $x_j := x_j + 1$
- $x_j := x_j - 1$

sind WHILE-Programme (und WHILE-Befehle) für alle Register x_j

Wenn P_1, P_2 WHILE-Programme sind, dann ist auch

- $P_1; P_2$

ein WHILE-Programm

Wenn P ein WHILE-Programm ist und x_j ein Register, dann ist

- `while $x_j \neq 0$ do P end`

ein WHILE-Programm (und ein WHILE-Befehl)

Definition 6.2 (Semantik von WHILE-Programmen)

Sei P ein WHILE-Programm.

Induktive Definition von $\Delta(P)$ wie folgt:

$\Delta(x_i := x_i + 1)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} s_2(x_i) &= s_1(x_i) + 1 \\ s_2(x_j) &= s_1(x_j) \text{ f\"ur } j \neq i \end{aligned}$$

$\Delta(x_i := x_i - 1)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} s_2(x_i) &= \begin{cases} s_1(x_i) - 1 & \text{falls } s_1(x_i) > 0 \\ 0 & \text{falls } s_1(x_i) = 0 \end{cases} \\ s_2(x_j) &= s_1(x_j) \text{ f\"ur } j \neq i \end{aligned}$$

$\Delta(P_1; P_2)(s_1, s_2)$ genau dann, wenn:

$$\begin{aligned} &\Delta(P_1)(s_1, s') \\ &\Delta(P_2)(s', s_2) \end{aligned}$$

Definition 6.3 (Semantik von WHILE-Programmen (Forts.))

$\Delta(\text{while } x_j \neq 0 \text{ do } P \text{ end})(s_1, s_2)$ genau dann, wenn:

es gibt für ein beliebiges $n \geq 0$ Zustände s'_0, \dots, s'_n mit

$$- s_1 = s'_0$$

$$- s_2 = s'_n$$

$$- \Delta(P)(s'_i, s'_{i+1}) \quad \text{für } 0 \leq i < n$$

$$- s_k(x_j) \neq 0 \quad \text{für } 0 \leq k < n$$

$$- s_n(x_j) = 0$$

Merke

Die Anzahl der Schleifendurchläufe ist **nicht** zu Anfang festgelegt

Der Schleifenrumpf kann die Anzahl der Durchläufe beeinflussen

Endlosschleifen sind möglich

Theorem 6.4

LOOP \subseteq **WHILE** ,

d.h., jede LOOP-berechenbare Funktion ist auch WHILE-berechenbar.

Beweis

Ein LOOP-Befehl

loop x_j do P end

kann in einem WHILE-Programm simuliert werden durch

```
while  $x_j \neq 0$  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;
while  $x_n \neq 0$  do
   $P$ ;    $x_n := x_n - 1$ 
end;
```

Nicht-Terminierung

WHILE-Programme können Endlosschleifen enthalten. Daher:

- WHILE-Programme terminieren nicht immer
- WHILE-berechenbare Funktionen können für bestimmte Eingaben undefiniert sein (**partielle Funktionen sein**)

Notation

WHILE = Menge aller **totalen** WHILE-berechenbaren Funktionen

WHILE^{part} = Menge **aller** WHILE-berechenbaren Funktionen
(incl. partiellen)