Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Wintersemester 2007/2008

Dank

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-Progamme
- Beziehungen zwischen LOOP, WHILE, GOTO
- Beziehungen zwischen Registermaschinen und Turingmaschinen

Teil II

Registermaschinen

Registermaschinen: Motivation

Im Vergleich zu Turingmaschinen

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

Ähnlich wie ...

- Imperativer Kern von Programmiersprachen
- Pseudo-Code

Beispiel Pseudo-Code

Berechnung von $a \mod b$

```
r := a;
while r ≥ b do
  r := r − b
end;
return r
```


Registermaschinen: Fragen der Definition

Fragen

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

Registermaschinen

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 $x_1, x_2, x_3, \ldots$,
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 $n+1$

Registermaschinen: Syntax von LOOP-Programmen

Definition 4.1 (LOOP-Programm)

- $\mathrm{x}_i := \mathrm{x}_i + 1$
- $\mathrm{x}_i := \mathrm{x}_i - 1$

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

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_i ein Register, dann ist

- $\mathtt{loop}\ \mathrm{x}_i\ \mathtt{do}\ P\ \mathtt{end}$

ein LOOP-Programm (und ein LOOP-Befehl)

Registermaschinen: Definition

Definition 4.2 (Zustand einer Registermaschine)

Ein Zustand s einer Registermaschine ist eine Abbildung

$$s : \{\mathrm{x}_i \mid i \in \mathbb{N}\} \longrightarrow \mathbb{N}$$

die jedem Register eine natürliche Zahl als Wert zuordnet.

Registermaschinen: Definition

Definition 4.3 (Anfangszustand, Eingabe)

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

- $\mathrm{x}_i = m_i$ für $1 \leq i \leq k$
- $\mathrm{x}_i = 0$ für $i > k$

Definition 4.4 (Ausgabe)

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

$$s_{term}(\mathrm{x}_{k+1})$$

die Ausgabe der Maschine.

Registermaschinen: Semantik

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.

Registermaschinen: Berechnete Funktion

Definition 4.6

Eine Registermaschine P berechnet eine Funktion

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

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

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

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

Registermaschinen: Berechnete Funktion

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 $\mathtt{x}_1, \ldots, \mathtt{x}_k$ auf die Eingabewerte
- Register $\mathtt{x}_i$ für $i > k+1$ auf 0

Folge

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

Registermaschinen: Berechenbare 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

Semantik von LOOP

Definition 5.1 (Semantik von LOOP-Programmen)

Sei P ein LOOP-Programm.
Induktive Definition von $\Delta(P)$ wie folgt:

$\Delta(\mathrm{x}_i := \mathrm{x}_i + 1)(s_1, s_2)$ genau dann, wenn:

$$s_2(\mathrm{x}_i) = s_1(\mathrm{x}_i) + 1$$
$$s_2(\mathrm{x}_j) = s_1(\mathrm{x}_j) \text{ für } j \neq i$$

$\Delta(\mathrm{x}_i := \mathrm{x}_i - 1)(s_1, s_2)$ genau dann, wenn:

$$s_2(\mathrm{x}_i) = \begin{cases} s_1(\mathrm{x}_i) - 1 & \text{falls } s_1(\mathrm{x}_i) > 0 \\ 0 & \text{falls } s_1(\mathrm{x}_i) = 0 \end{cases}$$
$$s_2(\mathrm{x}_j) = s_1(\mathrm{x}_j) \text{ für } j \neq i$$

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

$$\Delta(P_1)(s_1, s')$$
$$\Delta(P_2)(s', s_2)$$

Semantik von LOOP

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

$\Delta(\texttt{loop } \mathrm{x}_i \texttt{ do } P \texttt{ end})(s_1, s_2)$ genau dann, wenn:

es gibt Zustände $s'_0, \ldots, s'_n$ mit

- $s_1(\mathrm{x}_i) = n$
- $s_1 = s'_0$
- $s_2 = s'_n$
- $\Delta(P)(s'_k, s'_{k+1})$ 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**

LOOP-berechenbare Funktionen

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 Konkatenation: 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

$\mathrm{x}_i := 0$

$\texttt{loop } \mathrm{x}_i \texttt{ do } \mathrm{x}_i := \mathrm{x}_i - 1 \texttt{ end}$

$\mathrm{x}_i := c$ für $c \in \mathbb{N}$

$$\begin{array}{l} \mathrm{x}_i := 0; \\ \left.\begin{array}{l} \mathrm{x}_i := \mathrm{x}_i + 1; \\ \vdots \\ \mathrm{x}_i := \mathrm{x}_i + 1; \end{array}\right\} c \text{ mal} \end{array}$$

$\mathrm{x}_i := \mathrm{x}_j$

$\texttt{loop } \mathrm{x}_j \texttt{ do } \mathrm{x}_n := \mathrm{x}_n + 1 \texttt{ end;}$
$\mathrm{x}_i := 0;$
$\texttt{loop } \mathrm{x}_n \texttt{ do } \mathrm{x}_i := \mathrm{x}_i + 1 \texttt{ end}$

LOOP: Definition von Zusatzbefehlen

Im folgenden bezeichen $x_n, x_{n+1}, \ldots$ 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

Registermaschinen: Syntax von WHILE-Programmen

Definition 6.1 (WHILE-Programm)

- $x_i := x_i + 1$
- $x_i := x_i - 1$

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

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_i ein Register, dann ist

- while $x_i \neq 0$ do P end

ein WHILE-Programm (und ein WHILE-Befehl)

Semantik von WHILE

Definition 6.2 (Semantik von WHILE-Programmen)

Sei P ein WHILE-Programm.
Induktive Definition von $\Delta(P)$ wie folgt:

$\Delta(\mathrm{x}_i := \mathrm{x}_i + 1)(s_1, s_2)$ genau dann, wenn:

$$s_2(\mathrm{x}_i) = s_1(\mathrm{x}_i) + 1$$
$$s_2(\mathrm{x}_j) = s_1(\mathrm{x}_j) \text{ für } j \neq i$$

$\Delta(\mathrm{x}_i := \mathrm{x}_i - 1)(s_1, s_2)$ genau dann, wenn:

$$s_2(\mathrm{x}_i) = \begin{cases} s_1(\mathrm{x}_i) - 1 & \text{falls } s_1(\mathrm{x}_i) > 0 \\ 0 & \text{falls } s_1(\mathrm{x}_i) = 0 \end{cases}$$
$$s_2(\mathrm{x}_j) = s_1(\mathrm{x}_j) \text{ für } j \neq i$$

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

$$\Delta(P_1)(s_1, s')$$
$$\Delta(P_2)(s', s_2)$$

Semantik von WHILE

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

$\Delta(\mathtt{while}\ \mathrm{x}_i \neq 0\ \mathtt{do}\ P\ \mathtt{end})(s_1, s_2)$ genau dann, wenn:

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

- $s_1 = s'_0$
- $s_2 = s'_n$
- $\Delta(P)(s'_i, s'_{i+1})$ für $0 \leq i < n$
- $s_k(\mathrm{x}_i) \neq 0$ für $0 \leq k < n$
- $s_n(\mathrm{x}_i) = 0$

Merke

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

Der Schleifenrumpf kann die Anzahl der Durchläufe beeinflussen

Endlosschleifen sind möglich

WHILE und LOOP

Theorem 6.4

LOOP $\subseteq$ **WHILE** ,

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

WHILE und LOOP

Beweis

Ein LOOP-Befehl

$\mathtt{loop}\ \mathrm{x}_i\ \mathtt{do}\ P\ \mathtt{end}$

kann in einem WHILE-Programm simuliert werden durch

$\mathtt{while}\ \mathrm{x}_i \neq 0\ \mathtt{do}$
$\quad \mathrm{x}_n := \mathrm{x}_n + 1; \quad \mathrm{x}_{n+1} := \mathrm{x}_{n+1} + 1; \quad \mathrm{x}_i := \mathrm{x}_i - 1$
$\mathtt{end};$
$\mathtt{while}\ \mathrm{x}_{n+1} \neq 0\ \mathtt{do}$
$\quad \mathrm{x}_i := \mathrm{x}_i + 1; \quad \mathrm{x}_{n+1} := \mathrm{x}_{n+1} - 1$
$\mathtt{end};$
$\mathtt{while}\ \mathrm{x}_n \neq 0\ \mathtt{do}$
$\quad P; \quad \mathrm{x}_n := \mathrm{x}_n - 1$
$\mathtt{end};$

Partielle WHILE-berechenbare Funktionen

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)

Teil II

Registermaschinen

Registermaschinen: Syntax von GOTO-Programmen

Definition 7.1 (Index)

Ein **Index** (Zeilennummer) ist eine Zahl $j \geq 0$.

Registermaschinen: Syntax von GOTO-Programmen

Definition 7.2 (GOTO-Befehl, GOTO-Programm)

- $\mathtt{x}_i := \mathtt{x}_i + 1$
- $\mathtt{x}_i := \mathtt{x}_i - 1$

sind GOTO-Befehle für alle Register $\mathtt{x}_i$

Wenn $\mathtt{x}_i$ ein Register ist und j ein Index, dann ist

- $\mathtt{if}\ \mathtt{x}_i = 0\ \mathtt{goto}\ j$

ein GOTO-Befehl

Wenn $B_1, \ldots, B_k$ GOTO-Befehle sind und $j_1, \ldots, j_k$ Indizes ($k \geq 1$), dann ist

- $j_1 : B_1;\ j_2 : B_2;\ \ldots;\ j_k : B_k$

ein GOTO-Programm

Unterschied im Aufbau: WHILE und GOTO

Unterschied im Aufbau

- WHILE-Programme enthalten WHILE-Programme
  Rekursive Definition von Syntax und Semantik
- GOTO-Programme sind eine Liste von GOTO-Befehlen
  Nicht-rekursive Definition von Syntax und Semantik

Semantik von GOTO

Definition 7.3 (Semantik von GOTO-Programmen)

Sei

$$P \quad = \quad j_1 : B_1;\ j_2 : B_2;\ \ldots;\ j_k : B_k$$

ein GOTO-Programm.

Außerdem sei j_{k+1} ein Index, der nicht in P vorkommt (Programmende).

Es gilt

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

genau dann, wenn es für ein beliebiges $n \geq 0$

- Zustände $s'_0, \ldots, s'_n$
- Indizes $z_0, \ldots, z_n$

gibt, für die folgendes gilt ...

Semantik von GOTO

Definition (Forts.)

- $s'_0 = s_1$
- $s'_n = s_2$
- $z_0 = j_1$
- $z_n = j_{k+1}$

Semantik von GOTO

Definition (Forts.)

- für $0 \leq l < n$ gilt, wobei $j_s : B_s$ die Zeile in P ist mit $j_s = z_l$:

wenn $B = \quad \mathrm{x}_i := \mathrm{x}_i + 1$:

$$s'_{l+1}(\mathrm{x}_i) = s'_l(\mathrm{x}_i) + 1$$
$$s'_{l+1}(\mathrm{x}_{i'}) = s'_l(\mathrm{x}_{i'}) \text{ für } i' \neq i$$
$$z_{l+1} = j_{s+1}$$

wenn $B = \quad \mathrm{x}_i := \mathrm{x}_i - 1$:

$$s'_{l+1}(\mathrm{x}_i) = \begin{cases} s'_l(\mathrm{x}_i) - 1 & \text{falls } s'_l(\mathrm{x}_i) > 0 \\ 0 & \text{falls } s'_l(\mathrm{x}_i) = 0 \end{cases}$$
$$s'_{l+1}(\mathrm{x}_{i'}) = s'_l(\mathrm{x}_{i'}) \text{ für } i' \neq i$$
$$z_{l+1} = j_{s+1}$$

wenn $B = \quad \texttt{if } \mathrm{x}_i = 0 \texttt{ goto } j_{\texttt{goto}}$:

$$s'_{l+1} = s'_l$$
$$z_{l+1} = \begin{cases} j_{\texttt{goto}} & \text{falls } s_l(\mathrm{x}_i) = 0 \\ j_{s+1} & \text{sonst} \end{cases}$$

Semantik von GOTO

Merke

Die Anzahl der Rücksprünge (≈ Schleifendurchläufe) ist **nicht** zu Anfang festgelegt

Endlosschleifen sind möglich

Notation

$$\begin{aligned}\textbf{GOTO} &= \text{Menge aller } \textbf{totalen} \text{ GOTO-berechenbaren Funktionen}\\ \textbf{GOTO}^{part} &= \text{Menge } \textbf{aller} \text{ GOTO-berechenbaren Funktionen}\\ &\quad \text{(incl. partiellen)}\end{aligned}$$

Gleichmächtigkeit von WHILE und GOTO

Theorem 7.4

$$\begin{aligned}\textbf{WHILE} &= \textbf{GOTO}\\ \textbf{WHILE}^{part} &= \textbf{GOTO}^{part}\end{aligned}$$

Beweis

1. WHILE $\subseteq$ GOTO und WHILE^{part} $\subseteq$ GOTO^{part}:

Es genügt zu zeigen, dass `while` $x_i \neq 0$ `do` P `end` mit GOTO-Befehlen simuliert werden kann.

Wir können dabei annehmen, dass P kein `while` (mehr) enthält (ersetze die `while` von innen nach außen)

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

`while` $x_i \neq 0$ `do` P `end`

wird ersetzt durch

$$\begin{aligned}&j_1: \texttt{if } x_i = 0 \texttt{ goto } j_3;\\ &P';\\ &j_2: \texttt{if } x_n = 0 \texttt{ goto } j_1; \quad \textit{// unbedingter Sprung, da } x_n = 0\\ &j_3: x_n := x_n - 1 \quad \textit{// NOP, nur Sprungziel}\end{aligned}$$

Dabei:

- x_n ein bisher nicht verwendetes Register
- P' entsteht aus P, indem allen Befehlen ohne Index ein beliebiger (neuer) Index vorangestellt wird

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

2. GOTO $\subseteq$ WHILE und GOTO^{part} $\subseteq$ WHILE^{part}:

Jedes GOTO-Programm

$$j_1 : B_1;\ j_2 : B_2;\ \ldots;\ j_k : B_k$$

kann durch das folgende äquivalente WHILE-Programm ersetzt werden:

$$\begin{aligned}&x_{\texttt{index}} := j_1;\\ &\texttt{while } x_{\texttt{index}} \neq 0 \texttt{ do}\\ &\quad \texttt{if } x_{\texttt{index}} = j_1 \texttt{ then } B'_1 \texttt{ end};\\ &\quad \texttt{if } x_{\texttt{index}} = j_2 \texttt{ then } B'_2 \texttt{ end};\\ &\quad\quad \vdots\\ &\quad \texttt{if } x_{\texttt{index}} = j_k \texttt{ then } B'_k \texttt{ end};\\ &\texttt{end}\end{aligned}$$

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

Dabei für $1 \le i \le k$:

Falls $B_i = \mathtt{x}_i := \mathtt{x}_i \pm 1$**:**

$$B_i' = \mathtt{x}_i := \mathtt{x}_i \pm 1;\ \mathtt{x}_{\mathtt{index}} = j_{i+1}$$

Falls $B_i = \mathtt{if}\ \mathtt{x}_i = 0\ \mathtt{goto}\ j_{\mathtt{goto}}$**:**

$$B_i' = \begin{array}{l} \mathtt{if}\ \mathtt{x}_i = 0\ \mathtt{then}\ \mathtt{x}_{\mathtt{index}} = j_{\mathtt{goto}} \\ \qquad\qquad \mathtt{else}\ \mathtt{x}_{\mathtt{index}} = j_{i+1} \end{array}$$

Außerdem:

$$j_{k+1} = 0$$

Gleichmächtigkeit von WHILE und GOTO

Schlussfolgerung aus dem Beweis

Jede WHILE-berechenbare Funktion lässt sich durch ein **WHILE+IF-Programm** mit nur **einer Schleife** berechnen.

Weitere Schlussfolgerung

Spaghetti-Code (GOTO) ist nicht mächtiger als strukturierter Code (WHILE)

Denn:
Man kann mit dem einen das andere nachahmen.

Teil II

Registermaschinen

Rück- und Ausblick

Aus dem bisherigen wissen wir schon:

$$\textbf{LOOP} \subseteq \textbf{WHILE} = \textbf{GOTO} \subsetneq \textbf{WHILE}^{part} = \textbf{GOTO}^{part}$$

Noch zu zeigen:

- $\textbf{LOOP} \neq \textbf{WHILE}$
- $\textbf{WHILE} = \textbf{TM}$ und $\textbf{WHILE}^{part} = \textbf{TM}^{part}$

GOTO $\subseteq$ TM

Theorem 8.1

GOTO $\subseteq$ ***TM***

Beweisskizze

Es genügt zu zeigen, dass zu jedem GOTO-Programm

$$P = \quad j_1 : B_1;\ j_2 : B_2;\ \ldots;\ j_k : B_k$$

eine äquivalente Turingmaschine konstruiert werden kann.

GOTO $\subseteq$ TM

Beweisskizze (Forts.)

Sei l die Zahl der in P vorkommenden Register

Konstruiere eine TM M mit l Halbbändern
über dem Alphabet $\Sigma = \{\#, |\}$

Das i-te Band enthält immer soviele $|$, wie der Wert von x_i ist

M hat für jede Zeile $j_r : B_r$ von P einen Zustand s_r

Wenn M in s_r ist, macht sie das, was B_r entspricht:

- Register inkrementieren oder dekrementieren
- Sprungbedingung auswerten
- In entsprechenden Folgezustand gehen

All dies kann eine TM offensichtlich leisten

GOTO $\subseteq$ TM

Beweisskizze (Forts.)

In „Theoretische Informatik I“ haben wir gezeigt:
Zu jeder TM mit mehreren Halbbändern gibt es eine äquivalente Standard-TM mit nur einem Halbband.

Also gibt es auch eine Standard-TM, die das Programm P simuliert.

Bemerkung

TM $\subseteq$ **GOTO** und damit **TM** $=$ **GOTO** $=$ **WHILE** beweisen wir später

LOOP $\neq$ TM

Lemma 8.2

Die Menge aller LOOP-Programme ist rekursiv aufzählbar.

Beweisskizze

Man kann eine Grammatik angeben, die die LOOP-Programme erzeugt.

Bemerkung

Das gilt genauso für WHILE-Programme, GOTO-Programme und Turingmaschinen.

LOOP $\neq$ TM

Lemma 8.3

Es gibt eine Turingmaschine M_{LOOP}, die alle LOOP-Programme simuliert.

Genauer:
Gegeben sei eine Aufzählung $P_1, P_2, P_3, \ldots$ aller LOOP-Programme.

Wenn P_i auf Eingabe m, die Ausgabe o berechnet,
dann berechnet M_{LOOP} auf Eingabe $\langle i, m \rangle$ die Ausgabe o.

Beweisskizze

Der Beweis kannn genauso geführt werden, wie der Beweis, dass es eine universelle TM gibt, die alle Turingmaschinen simuliert.

Bemerkung

Auch das gilt genauso für WHILE-Programme, GOTO-Programme und TMen.

LOOP $\neq$ TM

Theorem 8.4

LOOP $\neq$ TM

Beweis

Die Funktion $\psi : \mathbb{N} \to \mathbb{N}$ sei definiert durch:

$$\psi(i) = P_i(i) + 1 \qquad \textit{(Ausgabe von } P_i \textit{ auf Eingabe } i \textit{ plus 1)}$$

1. $\psi \in$ TM

Ändere M_{LOOP} so ab, dass nach der Berechnung von $P_i(i)$ noch 1 addiert wird.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin$ LOOP

Angenommen, es gebe ein LOOP-Programm P_{i_0}, das ψ berechnet.

Aber:
Die Ausgabe von P_{i_0} auf Eingabe von i_0 ist

$$P_{i_0}(i_0) \neq P_{i_0}(i_0) + 1 = \psi(i_0)$$

Widerspruch!

Bemerkung

Dies gilt **nicht** für WHILE-Programme, GOTO-Programme und TMen.
Warum?Beweis beruht auf Totalität von ψ,
denn sonst kann $P_{i_0}(i_0) = P_{i_0}(i_0) + 1$ undefiniert sein.

Rück- und Ausblick

Aus dem bisherigen wissen wir:

- **LOOP $\subsetneq$ WHILE $=$ GOTO $\subseteq$ TM**
- **WHILE^{part} $=$ GOTO^{part} $\subseteq$ TM^{part}**

Noch zu zeigen:

TM $\subseteq$ WHILE und **TM^{part} $\subseteq$ WHILE^{part}**