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

Teil II

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

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

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

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

wenn $B =$ $\mathtt{if}\ \mathtt{x}_i = 0\ \mathtt{goto}\ j_{\mathtt{goto}}$:

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

Semantik von GOTO

Merke

Die Anzahl der Rücksprünge ($\approx$ Schleifendurchläufe)
ist **nicht** zu Anfang festgelegt

Endlosschleifen sind möglich

Notation

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

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

Gleichmächtigkeit von WHILE und GOTO

Theorem 7.4

WHILE = **GOTO**

WHILE^{part} = **GOTO**^{part}

Beweis

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

Es genügt zu zeigen, dass $\mathtt{while}\ \mathtt{x}_i \neq 0\ \mathtt{do}\ P\ \mathtt{end}$
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.)

$\mathtt{while}\ \mathtt{x}_i \neq 0\ \mathtt{do}\ P\ \mathtt{end}$

wird ersetzt durch

$j_1 :\ \mathtt{if}\ \mathtt{x}_i = 0\ \mathtt{goto}\ j_3;$
$P';$
$j_2 :\ \mathtt{if}\ \mathtt{x}_n = 0\ \mathtt{goto}\ j_1;$ *// unbedingter Sprung, da* $\mathtt{x}_n = 0$
$j_3 :\ \mathtt{x}_n := \mathtt{x}_n - 1$ *// NOP, nur Sprungziel*

Dabei:

- $\mathtt{x}_n$ 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:

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

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

Dabei für $1 \leq i \leq k$:

Falls $B_i =$ $x_i := x_i \pm 1$**:**

$$B'_i = \quad x_i := x_i \pm 1;\ x_{index} = j_{i+1}$$

Falls $B_i =$ `if` $x_i = 0$ `goto` j_{goto}**:**

$$B'_i = \quad \texttt{if } x_i = 0 \texttt{ then } x_{index} = j_{goto} \texttt{ else } x_{index} = j_{i+1}$$

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

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

$$\textit{GOTO} \subseteq \textit{TM}$$

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

$\textbf{TM} \subseteq \textbf{GOTO}$ und damit $\textbf{TM} = \textbf{GOTO} = \textbf{WHILE}$ 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:

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

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}