

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*

Teil II

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

Teil II

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

Definition 7.1 (Index)

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

Definition 7.2 (GOTO-Befehl, GOTO-Programm)

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

sind GOTO-Befehle für alle Register x_i

Wenn x_j ein Register ist und j ein Index, dann ist

- $\text{if } x_j = 0 \text{ goto } j$

ein GOTO-Befehl

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

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

ein GOTO-Programm

Definition 7.2 (GOTO-Befehl, GOTO-Programm)

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

sind GOTO-Befehle für alle Register x_i

Wenn x_j ein Register ist und j ein Index, dann ist

- $\text{if } x_i = 0 \text{ goto } j$

ein GOTO-Befehl

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

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

ein GOTO-Programm

Definition 7.2 (GOTO-Befehl, GOTO-Programm)

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

sind GOTO-Befehle für alle Register x_j

Wenn x_j ein Register ist und j ein Index, dann ist

- $\text{if } x_j = 0 \text{ goto } j$

ein GOTO-Befehl

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

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

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

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

Definition 7.3 (Semantik von GOTO-Programmen)

Sei

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

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 $s'_0, \dots, s'_n$
- Indizes $z_0, \dots, z_n$

gibt, für die folgendes gilt ...

Definition 7.3 (Semantik von GOTO-Programmen)

Sei

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

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 $s'_0, \dots, s'_n$
- Indizes $z_0, \dots, z_n$

gibt, für die folgendes gilt ...

Definition 7.3 (Semantik von GOTO-Programmen)

Sei

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

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 $s'_0, \dots, s'_n$
- Indizes $z_0, \dots, z_n$

gibt, für die folgendes gilt ...

Definition (Forts.)

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

Definition (Forts.)

- $s'_0 = s_1$

- $s'_n = s_2$

- $z_0 = j_1$

- $z_n = j_{k+1}$

Definition (Forts.)

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

Definition (Forts.)

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

Definition (Forts.)

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

$$\begin{aligned} \text{wenn } B = \quad & x_i := x_i + 1 : \\ & s'_{l+1}(x_i) = s'_l(x_i) + 1 \\ & s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i \\ & z_{l+1} = j_{s+1} \end{aligned}$$

$$\begin{aligned} \text{wenn } B = \quad & x_i := x_i - 1 : \\ & s'_{l+1}(x_i) = \begin{cases} s'_l(x_i) - 1 & \text{falls } s'_l(x_i) > 0 \\ 0 & \text{falls } s'_l(x_i) = 0 \end{cases} \\ & s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i \\ & z_{l+1} = j_{s+1} \end{aligned}$$

$$\begin{aligned} \text{wenn } B = \quad & \text{if } x_i = 0 \text{ goto } j_{\text{goto}} : \\ & s'_{l+1} = s'_l \\ & z_{l+1} = \begin{cases} j_{\text{goto}} & \text{falls } s_l(x_i) = 0 \\ j_{s+1} & \text{sonst} \end{cases} \end{aligned}$$

Definition (Forts.)

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

wenn $B = x_i := x_i + 1$:

$$s'_{l+1}(x_i) = s'_l(x_i) + 1$$

$$s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i$$

$$z_{l+1} = j_{s+1}$$

wenn $B = x_i := x_i - 1$:

$$s'_{l+1}(x_i) = \begin{cases} s'_l(x_i) - 1 & \text{falls } s'_l(x_i) > 0 \\ 0 & \text{falls } s'_l(x_i) = 0 \end{cases}$$

$$s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i$$

$$z_{l+1} = j_{s+1}$$

wenn $B = \text{if } x_i = 0 \text{ goto } j_{\text{goto}}$:

$$s'_{l+1} = s'_l$$

$$z_{l+1} = \begin{cases} j_{\text{goto}} & \text{falls } s_l(x_i) = 0 \\ j_{s+1} & \text{sonst} \end{cases}$$

Definition (Forts.)

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

wenn $B = x_i := x_i + 1$:

$$s'_{l+1}(x_i) = s'_l(x_i) + 1$$

$$s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i$$

$$z_{l+1} = j_{s+1}$$

wenn $B = x_i := x_i - 1$:

$$s'_{l+1}(x_i) = \begin{cases} s'_l(x_i) - 1 & \text{falls } s'_l(x_i) > 0 \\ 0 & \text{falls } s'_l(x_i) = 0 \end{cases}$$

$$s'_{l+1}(x_{i'}) = s'_l(x_{i'}) \text{ für } i' \neq i$$

$$z_{l+1} = j_{s+1}$$

wenn $B = \text{if } x_i = 0 \text{ goto } j_{\text{goto}}$:

$$s'_{l+1} = s'_l$$

$$z_{l+1} = \begin{cases} j_{\text{goto}} & \text{falls } s_l(x_i) = 0 \\ j_{s+1} & \text{sonst} \end{cases}$$

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)

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)

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 `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

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 `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

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 `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

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 `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

`j_1 : if $x_i = 0$ goto j_3 ;
 P' ;`

`$j_2$  : if  $x_n = 0$  goto  $j_1$ ;    // unbedingter Sprung, da  $x_n = 0$`

`$j_3$  :  $x_n := x_n - 1$            // NOP, nur Sprungziel`

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.)

while $x_i \neq 0$ do P end

wird ersetzt durch

j_1 : if $x_i = 0$ goto j_3 ;
 P' ;

j_2 : if $x_n = 0$ goto j_1 ; // *unbedingter Sprung, da $x_n = 0$*

j_3 : $x_n := x_n - 1$ // *NOP, nur Sprungziel*

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.)

while $x_i \neq 0$ do P end

wird ersetzt durch

j_1 : if $x_i = 0$ goto j_3 ;
 P' ;

j_2 : if $x_n = 0$ goto j_1 ; // *unbedingter Sprung, da $x_n = 0$*

j_3 : $x_n := x_n - 1$ // *NOP, nur Sprungziel*

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.)

while $x_i \neq 0$ do P end

wird ersetzt durch

j_1 : if $x_i = 0$ goto j_3 ;
 P' ;

j_2 : if $x_n = 0$ goto j_1 ; // *unbedingter Sprung, da $x_n = 0$*

j_3 : $x_n := x_n - 1$ // *NOP, nur Sprungziel*

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

Jedes GOTO-Programm

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

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.)

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

Jedes GOTO-Programm

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

kann durch das folgende äquivalente WHILE-Programm ersetzt werden:

```
x_index := j_1;  
while x_index ≠ 0 do  
  if x_index = j_1 then B'_1 end;  
  if x_index = j_2 then B'_2 end;  
  ⋮  
  if x_index = j_k then B'_k end;  
end
```

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

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

Jedes GOTO-Programm

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

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;  
  :  
  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 $B_i = x_i := x_i \pm 1$:

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

Falls $B_i = \text{if } x_i = 0 \text{ goto } j_{\text{goto}}$:

$$B'_i = \begin{array}{l} \text{if } x_i = 0 \text{ then } x_{\text{index}} = j_{\text{goto}} \\ \text{else } x_{\text{index}} = j_{i+1} \end{array}$$

Außerdem:

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

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

Dabei für $1 \leq i \leq k$:

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

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

Falls $B_i = \text{if } x_i = 0 \text{ goto } j_{\text{goto}}$:

$$B'_i = \begin{array}{l} \text{if } x_i = 0 \text{ then } x_{\text{index}} = j_{\text{goto}} \\ \text{else } x_{\text{index}} = j_{i+1} \end{array}$$

Außerdem:

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

Gleichmächtigkeit von WHILE und GOTO

Beweis (Forts.)

Dabei für $1 \leq i \leq k$:

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

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

Falls $B_i = \text{if } x_i = 0 \text{ goto } j_{\text{goto}}$:

$$B'_i = \begin{array}{l} \text{if } x_i = 0 \text{ then } x_{\text{index}} = j_{\text{goto}} \\ \text{else } x_{\text{index}} = j_{i+1} \end{array}$$

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.

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.

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

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

Teil II

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

Aus dem bisherigen wissen wir schon:

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

Noch zu zeigen:

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

Aus dem bisherigen wissen wir schon:

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

Noch zu zeigen:

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

Theorem 8.1

$$\mathbf{GOTO} \subseteq \mathbf{TM}$$

Beweisskizze

Es genügt zu zeigen, dass zu jedem GOTO-Programm

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

eine äquivalente Turingmaschine konstruiert werden kann.

Theorem 8.1

$$\mathbf{GOTO} \subseteq \mathbf{TM}$$

Beweisskizze

Es genügt zu zeigen, dass zu jedem GOTO-Programm

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

eine äquivalente Turingmaschine konstruiert werden kann.

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

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

Beweisskizze (Forts.)

Sei I die Zahl der in P vorkommenden Register

Konstruiere eine TM M mit I 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

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

Beweisskizze (Forts.)

Sei I die Zahl der in P vorkommenden Register

Konstruiere eine TM M mit I 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

Beweisskizze (Forts.)

Sei I die Zahl der in P vorkommenden Register

Konstruiere eine TM M mit I 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

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

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

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.

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.

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 $P_1, P_2, P_3, \dots$ 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 kann 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

Lemma 8.3

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

Genauer:

Gegeben sei eine Aufzählung $P_1, P_2, P_3, \dots$ 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 kann 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

Lemma 8.3

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

Genauer:

Gegeben sei eine Aufzählung $P_1, P_2, P_3, \dots$ 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 kann 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

Lemma 8.3

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

Genauer:

Gegeben sei eine Aufzählung $P_1, P_2, P_3, \dots$ 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 kann 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

Lemma 8.3

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

Genauer:

Gegeben sei eine Aufzählung $P_1, P_2, P_3, \dots$ 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 kann 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.

Theorem 8.4

$$\text{LOOP} \neq \text{TM}$$

Beweis

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

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

1. $\psi \in \text{TM}$

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

LOOP $\neq$ TM

Theorem 8.4

$$\text{LOOP} \neq \text{TM}$$

Beweis

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

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

1. $\psi \in \text{TM}$

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

Theorem 8.4

$$\text{LOOP} \neq \text{TM}$$

Beweis

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

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

1. $\psi \in \text{TM}$

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

Theorem 8.4

$$LOOP \neq TM$$

Beweis

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

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

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. $\psi \notin \text{LOOP}$

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.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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.

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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?

LOOP $\neq$ TM

Beweis (Forts.)

2. $\psi \notin \text{LOOP}$

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.

Beweis beruht auf Totalität von ψ ,

denn sonst kann $P_{i_0}(i_0) = P_{i_0}(i_0) + 1$ undefiniert sein.

Aus dem bisherigen wissen wir:

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

Noch zu zeigen:

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

Aus dem bisherigen wissen wir:

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

Noch zu zeigen:

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