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

Inhalt von Teil II

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

Inhalt von Teil II

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

Inhalt von Teil II

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

Inhalt von Teil II

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

Inhalt von Teil II

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

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

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

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

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

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

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: 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: 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: 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: 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

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

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

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

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)

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

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

- `loop` x_i `do` P `end`

ein LOOP-Programm (und ein LOOP-Befehl)

Registermaschinen: Syntax von LOOP-Programmen

Definition 4.1 (LOOP-Programm)

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

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

- `loop` x_i `do` P `end`

ein LOOP-Programm (und ein LOOP-Befehl)

Registermaschinen: Syntax von LOOP-Programmen

Definition 4.1 (LOOP-Programm)

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

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

- `loop` x_i `do` P `end`

ein LOOP-Programm (und ein LOOP-Befehl)

Registermaschinen: Definition

Definition 4.2 (Zustand einer Registermaschine)

Ein Zustand s einer Registermaschine ist eine Abbildung

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

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: 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: 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

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

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

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 $\mathrm{x}_1, \ldots, \mathrm{x}_k$ 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.

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 $\mathrm{x}_1, \ldots, \mathrm{x}_k$ 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.

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

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

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

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

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

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

$$\begin{aligned}
\textbf{LOOP} &= \text{Menge aller LOOP-berechenbaren Funktionen}\\
\textbf{WHILE} &= \text{Menge aller WHILE-berechenbaren Funktionen}\\
\textbf{GOTO} &= \text{Menge aller GOTO-berechenbaren Funktionen}\\
\textbf{TM} &= \text{Menge aller TM-berechenbaren Funktionen}
\end{aligned}$$

Teil II

Registermaschinen

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:

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

$$s_2(x_i) = s_1(x_i) + 1$$
$$s_2(x_j) = s_1(x_j) \text{ für } j \neq i$$

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

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

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

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

Semantik von LOOP

Definition 5.1 (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:

$$s_2(x_i) = s_1(x_i) + 1$$
$$s_2(x_j) = s_1(x_j) \text{ für } j \neq i$$

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

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

$\Delta(P_1; P_2)(s_1, s_2)$ 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.))

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

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

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

Merke

Die Anzahl der Schleifendurchläufe ist der Wert von $\texttt{x}_i$ **zu Beginn der Schleife**.

Etwaige Änderungen an $\texttt{x}_i$ innerhalb der Schleifendurchläufe sind **irrelevant**

Semantik von LOOP

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

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

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

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

Merke

Die Anzahl der Schleifendurchläufe ist der Wert von $\texttt{x}_i$ **zu Beginn der Schleife**.

Etwaige Änderungen an $\texttt{x}_i$ innerhalb der Schleifendurchläufe sind **irrelevant**

Semantik von LOOP

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

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

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

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

Merke

Die Anzahl der Schleifendurchläufe ist der Wert von $\texttt{x}_i$ **zu Beginn der Schleife**.

Etwaige Änderungen an $\texttt{x}_i$ 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-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-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-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-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-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

$x_i := 0$

$\texttt{loop } x_i \texttt{ do } x_i := x_i - 1 \texttt{ end}$

$x_i := c$ für $c \in \mathbb{N}$

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

$x_i := x_j$

$\texttt{loop } x_j \texttt{ do } x_n := x_n + 1 \texttt{ end};$
$x_i := 0;$
$\texttt{loop } x_n \texttt{ do } x_i := x_i + 1 \texttt{ end}$

LOOP: Definition von Zusatzbefehlen

$x_i := 0$

$$\texttt{loop } x_i \texttt{ do } x_i := x_i - 1 \texttt{ end}$$

$x_i := c$ für $c \in \mathbb{N}$

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

$x_i := x_j$

$$\begin{array}{l} \texttt{loop } x_j \texttt{ do } x_n := x_n + 1 \texttt{ end}; \\ x_i := 0; \\ \texttt{loop } x_n \texttt{ do } x_i := x_i + 1 \texttt{ end} \end{array}$$

LOOP: Definition von Zusatzbefehlen

$x_i := 0$

$\texttt{loop } x_i \texttt{ do } x_i := x_i - 1 \texttt{ end}$

$x_i := c$ für $c \in \mathbb{N}$

$x_i := 0;$

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

$x_i := x_j$

$\texttt{loop } x_j \texttt{ do } x_n := x_n + 1 \texttt{ end};$

$x_i := 0;$

$\texttt{loop } x_n \texttt{ do } x_i := x_i + 1 \texttt{ end}$

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

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

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

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
$x_n := 0$

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

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

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)

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)

Registermaschinen: Syntax von WHILE-Programmen

Definition 6.1 (WHILE-Programm)

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

sind WHILE-Programme (und WHILE-Befehle) für alle Register $\mathtt{x}_i$

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 $\mathtt{x}_i$ ein Register, dann ist

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

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:

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

$$s_2(x_i) = s_1(x_i) + 1$$
$$s_2(x_j) = s_1(x_j) \text{ für } j \neq i$$

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

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

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

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

Semantik von WHILE

Definition 6.2 (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.))

$\Delta(\texttt{while x}_i \neq 0 \texttt{ do } P \texttt{ end})(s_1, s_2)$ 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 $0 \leq i < n$
- $s_k(\texttt{x}_i) \neq 0$ für $0 \leq k < n$
- $s_n(\texttt{x}_i) = 0$

Merke

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

Der Schleifenrumpf kann die Anzahl der Durchläufe beeinflussen

Endlosschleifen sind möglich

Semantik von WHILE

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

$\Delta(\texttt{while x}_i \neq 0 \texttt{ do } P \texttt{ end})(s_1, s_2)$ 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 $0 \leq i < n$
- $s_k(\texttt{x}_i) \neq 0$ für $0 \leq k < n$
- $s_n(\texttt{x}_i) = 0$

Merke

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

Der Schleifenrumpf kann die Anzahl der Durchläufe beeinflussen

Endlosschleifen sind möglich

Semantik von WHILE

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

$\Delta(\texttt{while x}_i \neq 0 \texttt{ do } P \texttt{ end})(s_1, s_2)$ 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 $0 \leq i < n$
- $s_k(\texttt{x}_i) \neq 0$ für $0 \leq k < n$
- $s_n(\texttt{x}_i) = 0$

Merke

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

Der Schleifenrumpf kann die Anzahl der Durchläufe beeinflussen

Endlosschleifen sind möglich

Semantik von WHILE

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

$\Delta(\texttt{while x}_i \neq 0 \texttt{ do } P \texttt{ end})(s_1, s_2)$ 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 $0 \leq i < n$
- $s_k(\texttt{x}_i) \neq 0$ für $0 \leq k < n$
- $s_n(\texttt{x}_i) = 0$

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

$$\textbf{LOOP} \subseteq \textbf{WHILE} \ ,$$

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

WHILE und LOOP

Beweis

Ein LOOP-Befehl

$$\texttt{loop } x_i \texttt{ do } P \texttt{ end}$$

kann in einem WHILE-Programm simuliert werden durch

$$\texttt{while } x_i \neq 0 \texttt{ do}$$
$$\quad x_n := x_n + 1; \quad x_{n+1} := x_{n+1} + 1; \quad x_i := x_i - 1$$
$$\texttt{end;}$$
$$\texttt{while } x_{n+1} \neq 0 \texttt{ do}$$
$$\quad x_i := x_i + 1; \quad x_{n+1} := x_{n+1} - 1$$
$$\texttt{end;}$$
$$\texttt{while } x_n \neq 0 \texttt{ do}$$
$$\quad P; \quad x_n := x_n - 1$$
$$\texttt{end;}$$

WHILE und LOOP

Beweis

Ein LOOP-Befehl

$$\texttt{loop } x_i \texttt{ do } P \texttt{ end}$$

kann in einem WHILE-Programm simuliert werden durch

$$\begin{array}{l}
\texttt{while } x_i \neq 0 \texttt{ do} \\
\quad x_n := x_n + 1; \quad x_{n+1} := x_{n+1} + 1; \quad x_i := x_i - 1 \\
\texttt{end;} \\
\texttt{while } x_{n+1} \neq 0 \texttt{ do} \\
\quad x_i := x_i + 1; \quad x_{n+1} := x_{n+1} - 1 \\
\texttt{end;} \\
\texttt{while } x_n \neq 0 \texttt{ do} \\
\quad P; \quad x_n := x_n - 1 \\
\texttt{end;}
\end{array}$$

Beweis

Ein LOOP-Befehl

$$\texttt{loop } x_i \texttt{ do } P \texttt{ end}$$

kann in einem WHILE-Programm simuliert werden durch

$$\begin{array}{l}
\texttt{while } x_i \neq 0 \texttt{ do} \\
\quad x_n := x_n + 1; \quad x_{n+1} := x_{n+1} + 1; \quad x_i := x_i - 1 \\
\texttt{end;} \\
\texttt{while } x_{n+1} \neq 0 \texttt{ do} \\
\quad x_i := x_i + 1; \quad x_{n+1} := x_{n+1} - 1 \\
\texttt{end;} \\
\texttt{while } x_n \neq 0 \texttt{ do} \\
\quad P; \quad x_n := x_n - 1 \\
\texttt{end;}
\end{array}$$

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

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

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

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

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

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

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

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