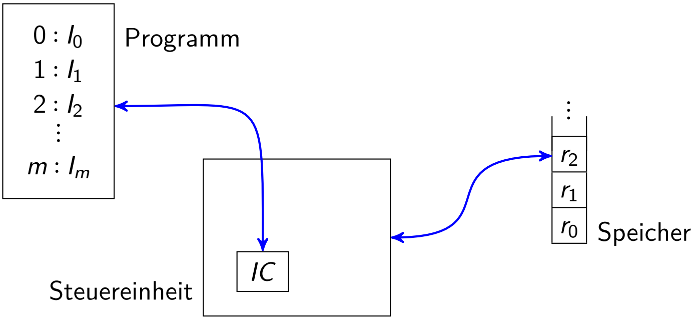
Einführung in die Theoretische Informatik

Johannes Köbler

Institut für Informatik
Humboldt-Universität zu Berlin

WS 2011/12

Die Registermaschine (random access machine, RAM)



- führt ein Programm $P = I_0, \ldots, I_m$ aus, das aus einer endlichen Folge von Befehlen (instructions) I_i besteht,
- hat einen Befehlszähler (instruction counter) IC, der die Nummer des nächsten Befehls angibt (zu Beginn ist $IC = 0$),
- verfügt über einen frei adressierbaren Speicher (random access memory) mit unendlich vielen Speicherzellen (Registern) r_i, die beliebig große natürliche Zahlen aufnehmen können.

Eine Programmiersprache für RAMs

In GOTO-Programmen sind folgende Befehle zulässig ($i, j, c \in \mathbb{N}$):

Befehl	Semantik
$r_i := r_j + c$	setzt Register r_i auf den Wert $r_j + c$
$r_i := r_j \dot{-} c$	setzt Register r_i auf den Wert $\max(0, r_j - c)$
GOTO j	setzt den Befehlszähler IC auf den Wert j
IF $r_i = c$ **THEN GOTO** j	setzt IC auf j, falls r_i den Wert c hat
HALT	beendet die Programmausführung

GOTO-Berechenbarkeit

Definition

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt GOTO-berechenbar, falls es ein GOTO-Programm $P = (I_0, \ldots, I_m)$ mit folgender Eigenschaft gibt:

Wird P auf einer RAM mit den Werten $r_i = n_i$ für $i = 1, \ldots, k$, sowie $IC = 0$ und $r_i = 0$ für $i = 0, k+1, k+2, \ldots$ gestartet, so gilt:

- P hält genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
- falls P hält, hat r_0 nach Beendigung von P den Wert $f(n_1, \ldots, n_k)$.

Beispiel

Folgendes GOTO-Programm berechnet die Funktion $f(x, y) = xy$:

0 **IF** $r_1 = 0$ **THEN GOTO** 4

1 $r_1 := r_1 \dot{-} 1$

2 $r_0 := r_0 + r_2$

3 **GOTO** 0

4 **HALT**

◁

Numerische Repräsentation von Wörtern

- Da Turingmaschinen auf Wörtern und GOTO-Programme auf Zahlen operieren, müssen wir Wörter durch Zahlen (und umgekehrt) kodieren.
- Sei $\Sigma = \{a_0, \ldots, a_{m-1}\}$ ein Alphabet. Dann können wir jedes Wort $x = a_{i_1} \ldots a_{i_n} \in \Sigma^*$ durch eine natürliche Zahl $num_\Sigma(w)$ kodieren:

$$num_\Sigma(x) = \sum_{j=0}^{n-1} m^j + \sum_{j=1}^{n} i_j m^{j-1} = \frac{m^n - 1}{m - 1} + (i_n \ldots i_1)_m.$$

- Da die Abbildung $num_\Sigma : \Sigma^* \to \mathbb{N}$ bijektiv ist, können wir umgekehrt jede natürliche Zahl n durch das Wort $str_\Sigma(n) = num_\Sigma^{-1}(n)$ kodieren.

Beispiel

Für das Alphabet $\Sigma = \{a, b, c\}$ erhalten wir folgende Kodierung:

w	ε	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc	aaa	$\ldots$
$num_\Sigma(w)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	$\ldots$

Äquivalenz von Turing- und GOTO-Berechenbarkeit

- Ist $\Sigma = \{0,1\}$, so lassen wir den Index weg und schreiben einfach *num* und *str* anstelle von num_Σ und str_Σ.
- Zudem erweitern wir die Kodierungsfunktion $str : \mathbb{N} \to \{0,1\}$ zu einer Kodierungsfunktion $str_k : \mathbb{N}^k \to \{0,1,\#\}$ wie folgt:

  $$str_k(n_1, \ldots, n_k) = str(n_1)\#\ldots\#str(n_k).$$

- Nun können wir eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ durch folgende partielle Wortfunktion $\hat{f} : \{0,1,\#\}^* \to \{0,1\}^* \cup \{\uparrow\}$ repräsentieren:

  $$\hat{f}(w) = \begin{cases} str(n), & w = str_k(n_1, \ldots, n_k) \text{ und } f(n_1, \ldots, n_k) = n \in \mathbb{N}, \\ \uparrow, & \text{sonst.} \end{cases}$$

- Es ist klar, dass f durch $\hat{f}$ eindeutig bestimmt ist. Wir nennen f die numerische Repräsentation von $\hat{f}$.

Satz

Sei f die numerische Repräsentation einer partiellen Funktion $\hat{f}$.
Dann ist f genau dann GOTO-berechenbar, wenn $\hat{f}$ berechenbar ist.

Simulation eines GOTO-Programms durch eine DTM

- Sei $\hat{f}$ eine partielle Funktion, deren numerische Repräsentation f von einem GOTO-Programm P auf einer RAM R berechnet wird.
- Dann existiert eine Zahl k', so dass P nur Register r_i mit $i \leq k'$ benutzt.
- Daher lässt sich eine Konfiguration von R durch Angabe der Inhalte des Befehlszählers IC und der Register $r_0, \ldots, r_{k'}$ beschreiben.
- Wir konstruieren eine $(k'+2)$-DTM M, die
  - den Inhalt von IC in ihrem Zustand,
  - die Registerwerte $r_1, \ldots, r_{k'}$ auf den Bändern $1, \ldots, k'$ und
  - den Wert von r_0 auf dem Ausgabeband $k'+2$ speichert.
- Ein Registerwert r_i wird hierbei in der Form $str(r_i)$ gespeichert.
- Band $k'+1$ wird zur Ausführung von Hilfsberechnungen benutzt.

Simulation eines GOTO-Programms durch eine DTM

- Die Aufgabe von M ist es, bei Eingabe $w = str_k(n_1, \dots, n_k)$ das Wort $str(f(n_1, \dots, n_k))$ auszugeben, wenn $(n_1, \dots, n_k) \in dom(f)$ ist, und andernfalls nicht zu halten.
- Zuerst kopiert M die Teilwörter $str(n_i)$ für $i = 2, \dots, k$ auf das i-te Band und löscht auf dem 1. Band alle Eingabezeichen bis auf $str(n_1)$.
- Da das leere Wort den Wert $num(\varepsilon) = 0$ kodiert, sind nun auf den Bändern $1, \dots, k'$ und auf Band $k' + 2$ die der Startkonfiguration von R bei Eingabe $(n_1, \dots, n_k)$ entsprechenden Registerinhalte gespeichert.
- Danach führt M das Programm P Befehl für Befehl aus.
- Es ist leicht zu sehen, dass sich jeder Befehl I in P durch eine Folge von Anweisungen realisieren lässt, die die auf den Bändern gespeicherten Registerinhalte bzw. den im Zustand von M gespeicherten Wert von IC entsprechend modifizieren.
- Sobald P stoppt, hält auch M und gibt das Wort $str(r_0) = str(f(n_1, \dots, n_k))$ aus.

Simulation einer DTM durch ein GOTO-Programm

- Sei $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ eine partielle Funktion und sei $M = (Z, \Sigma, \Gamma, \delta, q_1, E)$ eine DTM mit Eingabealphabet $\Sigma = \{0, 1, \#\}$, die die zugehörige Wortfunktion $\hat{f} : \Sigma^* \to \{0, 1\}^* \cup \{\uparrow\}$ berechnet.
- M gibt also bei Eingabe $str_k(n_1, \ldots, n_k)$ das Wort $str(f(n_1, \ldots, n_k))$ aus, falls $f(n_1, \ldots, n_k)$ definiert ist, und hält andernfalls nicht.
- Wir konstruieren ein GOTO-Programm P, das bei Eingabe $(n_1, \ldots, n_k)$ die DTM M bei Eingabe $w = str_k(n_1, \ldots, n_k)$ simuliert und im Fall, dass $M(w)$ hält, den Wert $num(M(w)) = f(n_1, \ldots, n_k)$ berechnet.
- Wir können annehmen, dass M eine 1-DTM ist.
- Sei $Z = \{q_1, \ldots, q_r\}$ und $\Gamma = \{a_0, \ldots, a_{m-1}\}$, wobei wir annehmen, dass $a_0 = \sqcup$, $a_1 = 0$, $a_2 = 1$ und $a_3 = \#$ ist.

Simulation einer DTM durch ein GOTO-Programm

- Eine Konfiguration $K = uq_iv$ von M wird wie folgt in den Registern r_0, r_1, r_2 gespeichert. Sei $u = a_{i_1} \dots a_{i_n}$ und $v = a_{j_1} \dots a_{j_{n'}}$.
  - $r_0 = (i_1 \dots i_n)_m$,
  - $r_1 = i$,
  - $r_2 = (j_{n'} \dots j_1)_m$.
- P besteht aus 3 Programmteilen $P = P_1, P_2, P_3$.
- P_1 berechnet in Register r_2 die Zahl $(j_n \dots j_1)_m$, wobei $(j_1, \dots, j_n)$ die Indexfolge der Zeichen von $str_k(n_1, \dots, n_k) = a_{j_1} \dots a_{j_n}$ ist.
- Die übrigen Register setzt P_1 auf den Wert 0.
- P_1 stellt also die Startkonfiguration $K_w = q_1w$ von M bei Eingabe $w = str_k(n_1, \dots, n_k)$ in den Registern r_0, r_1, r_2 her.
- Anschließend führt P_2 eine schrittweise Simulation von M aus.
- Hierzu überführt P_2 solange die in r_0, r_1, r_2 gespeicherte Konfiguration von M in die zugehörige Nachfolgekonfiguration, bis M hält.

Simulation einer DTM durch ein GOTO-Programm

- Das Programmstück P_2 hat die Form

$$
\begin{array}{ll}
M_2 & r_3 := r_2 \textbf{ MOD } m \\
& \textbf{IF } r_1 = 1 \wedge r_3 = 0 \textbf{ THEN GOTO } M_{1,0} \\
& \quad \vdots \\
& \textbf{IF } r_1 = r \wedge r_3 = m-1 \textbf{ THEN GOTO } M_{r,m-1}
\end{array}
$$

- Die Befehle ab Position $M_{i,j}$ hängen von $\delta(q_i, a_j)$ ab. Wir betrachten exemplarisch den Fall $\delta(q_i, a_j) = \{(q_{i'}, a_{j'}, L)\}$:

$$
\begin{array}{ll}
M_{i,j} & r_1 := i' \\
& r_2 := r_2 \textbf{ DIV } m \\
& r_2 := r_2 m + j' \\
& r_2 := r_2 m + (r_0 \textbf{ MOD } m) \\
& r_0 := r_2 \textbf{ DIV } m \\
& \textbf{GOTO } M_2
\end{array}
$$

- Im Fall $\delta(q_i, a_j) = \varnothing$ erfolgt ein Sprung an den Beginn von P_3.
- P_3 transformiert den Inhalt $r_0 = (j_1 \ldots j_n)_m$ von Register r_0 in die Zahl $num(a_{j_1} \ldots a_{j_n})$ und hält.

WHILE- und LOOP-Programme

- Die Syntax von WHILE-Programmen ist induktiv wie folgt definiert ($i, j, c \in \mathbb{N}$):
  - Jede Wertzuweisung der Form $x_i := x_j + c$ oder $x_i := x_j \dot{-} c$ ist ein WHILE-Programm.
  - Falls P und Q WHILE-Programme sind, so auch
    - $P; Q$ und
    - **IF** $x_i = c$ **THEN** P **ELSE** Q **END**
    - **WHILE** $x_i \neq c$ **DO** P **END**
- Die Syntax von LOOP-Programmen ist genauso definiert, nur dass Schleifen der Form **LOOP** x_i **DO** P **END** an die Stelle von WHILE-Schleifen treten.
- Die Semantik von WHILE-Programmen ist selbsterklärend.
- Eine LOOP-Schleife **LOOP** x_i **DO** P **END** wird sooft ausgeführt, wie der Wert von x_i zu Beginn der Schleife angibt.

WHILE- und LOOP-Berechenbarkeit

- Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt WHILE-berechenbar, falls es ein WHILE-Programm P mit folgender Eigenschaft gibt:

  Wird P mit den Werten $x_i = n_i$ für $i = 1, \ldots, k$ gestartet, so gilt:
  - P hält genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
  - falls P hält, hat x_0 nach Beendigung von P den Wert $f(n_1, \ldots, n_k)$.
- Die LOOP-Berechenbarkeit von f ist entsprechend definiert.

Beispiel

Die Funktion $f(x_1, x_2) = x_1 x_2$ wird von dem WHILE-Programm

WHILE $x_1 \neq 0$ **DO**
  $x_0 := x_0 + x_2;$
  $x_1 := x_1 \dot{-} 1$
END

sowie von folgendem LOOP-Programm berechnet:

LOOP x_1 **DO** $x_0 := x_0 + x_2$ **END** $\triangleleft$

Äquivalenz von WHILE- und GOTO-Berechenbarkeit

Satz

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ ist genau dann GOTO-berechenbar, wenn sie WHILE-berechenbar ist.

Simulation von WHILE- durch GOTO-Programme

- Sei P ein WHILE-Programm, das f berechnet.
- Wir übersetzen P wie folgt in ein äquivalentes GOTO-Programm P'.
- P' speichert den Variablenwert x_i im Register r_i.
- Damit lassen sich alle Wertzuweisungen von P direkt in entsprechende Befehle von P' transformieren.
- Eine Schleife der Form **WHILE** $x_i \neq c$ **DO** Q **END** simulieren wir durch folgendes GOTO-Programmstück:

  M_1 **IF** $r_i = c$ **THEN GOTO** M_2
  Q'
  GOTO M_1
  M_2 $\vdots$

- Ähnlich lässt sich die Verzweigung **IF** $x_i = c$ **THEN** Q_1 **ELSE** Q_2 **END** in ein GOTO-Programmstück transformieren.
- Zudem fügen wir ans Ende von P' den HALT-Befehl an.

Simulation von GOTO- durch WHILE-Programme

- Sei $P = (I_0, \ldots, I_m)$ ein GOTO-Programm, das f berechnet, und sei r_z, $z > k$, ein Register, das in P nicht benutzt wird.
- Wir übersetzen P wie folgt in ein äquivalentes WHILE-Programm P':

  $x_z := 0;$
  WHILE $x_z \neq m + 1$ **DO**
    IF $x_z = 0$ **THEN** P'_0 **END**;
      $\vdots$
    IF $x_z = m$ **THEN** P'_m **END**
  END

- Dabei ist P'_j abhängig vom Befehl I_j folgendes WHILE-Programm:
  - $r_i := r_k + c$: $x_i := x_k + c$; $x_z := x_z + 1$,
  - $r_i := r_k \dot{-} c$: $x_i := x_k \dot{-} c$; $x_z := x_z + 1$,
  - **GOTO** k: $x_z := k$,
  - **IF** $r_i = c$ **THEN GOTO** k:

    IF $x_i = c$ **THEN** $x_z := k$ **ELSE** $x_z := x_z + 1$ **END**
  - **HALT**: $x_z := m + 1$.
- Man beachte, dass P' nur eine WHILE-Schleife enthält.

Vergleich von LOOP- und WHILE-Berechenbarkeit

- Es ist leicht zu sehen, dass sich jedes LOOP-Programm durch ein WHILE-Programm simulieren lässt.
- Offensichtlich können LOOP-Programme nur totale Funktionen berechnen.
- Daher kann nicht jedes WHILE-Programm durch ein LOOP-Programm simuliert werden.
- Mittels Diagonalisierung lässt sich eine totale WHILE-berechenbare Funktion f angeben, die nicht LOOP-berechenbar ist.
- Ein bekanntes Beispiel einer totalen WHILE-berechenbaren Funktion, die nicht LOOP-berechenbar ist, ist die Ackermannfunktion $a(x, y)$, die induktiv wie folgt definiert ist:

$$a(x,y) = \begin{cases} y+1, & x = 0, \\ a(x-1, 1), & x \geq 1, y = 0, \\ a(x-1, a(x, y-1)), & x, y \geq 1. \end{cases}$$