

5. Registermaschinen

Ein formales Berechnungsmodell auf den natürlichen Zahlen

Registermaschinen

Die Registermaschinen formalisieren Berechnungsverfahren über den natürlichen Zahlen:

- Der SPEICHER besteht aus einer festen Anzahl von *Registern*, die je eine natürliche Zahl aufnehmen können.
- Die elementaren (TRANSFORMATIONS-)OPERATIONEN erlauben das *Inkrementieren* (+1) und das *Dekrementieren* (-1) der Zahlen in den Registern.
(Hierbei: $(n + 1) - 1 = n$ und $0 - 1 = 0$)
- Die elementaren TESTOPERATIONEN erlauben festzustellen, ob die Register *leer* sind, d.h. die 0 enthalten.

Turingmaschinen vs. Registermaschinen

Turingmaschinen:

- Daten = Wörter
- Speicher = 1 Wort (mit Blanks) und Zeiger auf einen der Buchstaben
- Operationen = elementare Wortoperationen (Ersetzen, Löschen und Anfügen (an den Wortenden) von Buchstaben)
- Tests = Lesen eines Buchstabens

Registermaschinen:

- Daten = natürliche Zahlen
- Speicher = 1 Zahlenvektor
- Operationen = elementare Zähloperationen (Hoch-/Runterzählen)
- Tests = Nulltests

Bei der Registermaschine wird von der Darstellung der Zahlen abstrahiert (vgl. die Idee des Datentyps).

REGISTERMASCHINE (INTUITIV): SPEICHERSTRUKTUR, EIN-/AUSGABE

Eine Registermaschine M zur Berechnung einer n -stelligen (partiellen) Funktion

$$\varphi : \mathbb{N}^n \rightarrow \mathbb{N}$$

verfügt über $k \geq n + 1$ Register:

- Register 1 - n : *Eingaberegister*
- Register $n + 1$: *Ausgaberegister*
- Register $n + 2 - k$: eventuelle *Hilfsregister*

EINGABE: Zu Beginn der Rechnung wird die i -te Komponente x_i der Eingabe \vec{x} in das i -te Register geschrieben ($1 \leq i \leq n$). Die übrigen Register sind leer.

AUSGABE: Am Ende der Rechnung wird die Ausgabe dem $n + 1$ -ten Register entnommen.

Hilfsregister (aber auch die Ein- und das Ausgaberegister) dienen während der Rechnung zum Speichern von Zwischenergebnissen.

REGISTERMASCHINE (INTUITIV): SPEICHEROPERATIONEN

- Inkrementieren des i -ten Registers

$$a_i(x_1, \dots, x_k) = (x_1, \dots, x_{i-1}, x_i + 1, x_{i+1}, \dots, x_k)$$

- Dekrementieren des i -ten Registers

$$s_i(x_1, \dots, x_k) = (x_1, \dots, x_{i-1}, x_i - 1, x_{i+1}, \dots, x_k)$$

- Nulltest für das i -te Register

$$t_i(x_1, \dots, x_k) = \begin{cases} 1 & \text{falls } x_i = 0 \\ 0 & \text{sonst} \end{cases}$$

(Hierbei gilt jeweils $1 \leq i \leq k$.)

REGISTERMASCHINE (INTUITIV): PROGRAMM

Das Programm ist eine endliche Folge von Operations- und Testinstruktionen.

- Operationsinstruktion

(z, a_i, z') bzw. (z, s_i, z')

“ im Zustand z in-(bzw. de-)krementiere das i -te Register und gehe in Zustand z' ”

- Testinstruktion

(z, t_i, z', z'')

“ im Zustand z teste, ob das i -te Register leer ($=0$) ist; falls nein, gehe in Zustand z' , falls ja, gehe in Zustand z'' ”

Dabei gibt es zu jedem Zustand z höchstens eine mit z beginnende Instruktion (\rightarrow Determinismus). Ein Zustand, mit dem keine Instruktion beginnt, ist ein *Stoppzustand*. Beginnt die Instruktion I mit dem Zustand z , so heisst z die *Adresse* von I .

BEMERKUNG. Im Gegensatz zu unserem Turingmaschinenmodell verzichten wir hier bei dem Programm auf bedingte Anweisungen. Wir könnten diese hier jedoch auch einführen, z.B. in der Form des *if – then – else*-Formats:

if Register i leer *then* P_1 *else* P_2

Ebenso könnten wir bei Turingmaschinen auf bedingte Anweisungen verzichten und getrennte Operations- und Testinstruktionen verwenden.

Auf die Mächtigkeit der beiden Konzepte hat die Wahl des Instruktionsformats keinen Einfluss.

BEISPIEL

Eine Registermaschine M zur Berechnung der Addition $f(x, y) = x + y$ kommt mit den beiden Eingabe- und dem Ausgaberegister aus. M arbeitet wie folgt:

Zunächst wird das erste Register dekrementiert bis es leer ist und dabei gleichzeitig das dritte Register entsprechend inkrementiert. Dann wird entsprechend mit dem zweiten Register verfahren.

(z_0, t_1, z_1, z_3)	$z_0 : \text{if } r_1 = 0 \text{ then goto } z_3 \text{ else goto } z_1;$
(z_1, s_1, z_2)	$z_1 : r_1 := r_1 - 1; \text{ goto } z_2;$
(z_2, a_3, z_0)	$z_2 : r_3 := r_3 + 1; \text{ goto } z_0;$
(z_3, t_2, z_4, z_6)	$z_3 : \text{if } r_2 = 0 \text{ then goto } z_6 (= \text{stop}) \text{ else goto } z_4;$
(z_4, s_1, z_5)	$z_4 : r_2 := r_2 - 1; \text{ goto } z_5;$
(z_5, a_3, z_3)	$z_5 : r_3 := r_3 + 1; \text{ goto } z_3.$

Am Ende der Rechnung sind die beiden Eingaberegister geleert und das Ausgaberegister enthält die Summe der beiden Eingaben.

FORMALE SPEZIFIKATION EINER k -REGISTERMASCHINE (zur Berechnung einer Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$):

$$M = (k, n, Z, z_0, \delta)$$

Komponenten von M :

- $k =$ Anzahl der Register von M wobei $k \geq n + 1$.
- n ist die Stelligkeit der berechneten Funktion.
- Z ist die endliche Menge der (Programm-)Zustände.
- $z_0 \in Z$ ist der Startzustand.
- δ , das Programm, ist eine partielle Funktion

$$\delta : Z \rightarrow (OPER \times Z) \cup (TEST \times Z \times Z)$$

wobei $OPER = \{a_1, \dots, a_k, s_1, \dots, s_k\}$ und $TEST = \{t_1, \dots, t_k\}$.

(δ ordnet der Adresse einer Instruktion deren Rumpf zu.)

BEISPIEL

Formale Spezifikation der zuvor angegebenen Registermaschine M zur Berechnung der Summe zweier Zahlen: $M = (3, 2, Z, z_0, \delta)$ wobei:

- $Z = \{z_0, \dots, z_6\}$
- δ durch folgende Tabelle bestimmt ist:

$$Z \rightarrow (OPER \times Z) \cup (TEST \times Z \times Z)$$

z_0			t_1	z_1	z_3
z_1	s_1	z_2			
z_2	a_3	z_0			
z_3			t_2	z_4	z_6
z_4	s_2	z_5			
z_5	a_3	z_3			

Meist geben wir δ durch Auflisten der Instruktionen an (wie im ursprünglichen Beispiel.)

FORMALE BESCHREIBUNG DER ARBEITSWEISE DER k -REGISTERMASCHINE $M = (k, n, Z, z_0, \delta)$

Wir gehen entsprechend wie bei Turingmaschinen vor und benutzen die dort eingeführten Begriffe.

- Menge der M -Konfigurationen:

$$KON_M = \mathbb{N}^k \times Z$$

- Startkonfiguration bei Eingabe $\vec{m} = (m_1, \dots, m_n)$:

$$\alpha_M(\vec{m}) = (in(\vec{m}), z_0) = (m_1, \dots, m_n, 0, \dots, 0, z_0) = (\vec{m}, 0^{k-(n+1)}, z_0)$$

- Die 1-Schrittrelation $\Rightarrow_M \subseteq KON_M \times KON_M$:

$$(\vec{r}, z) \Rightarrow_M (\vec{r}', z') \Leftrightarrow$$

$$(\vec{r}', z') = \begin{cases} (o(\vec{r}), z') & \text{falls } \delta(z) = (o, z') \\ (\vec{r}, z') & \text{falls } \delta(z) = (t, z', z'') \text{ und } t(\vec{r}) = 0 \\ (\vec{r}, z'') & \text{falls } \delta(z) = (t, z', z'') \text{ und } t(\vec{r}) = 1 \end{cases}$$

- Ausgabefunktion $out_M : \mathbb{N}^k \times Z \rightarrow \mathbb{N}$:

$$out_M(r_1, \dots, r_k, z) = out_{k,n}(r_1, \dots, r_k) = r_{n+1}$$

- Hiermit lassen sich dann *Nachfolgekonfigurationen*, *Stoppkonfigurationen*, *Konfigurationenfolgen*, *Rechnungen* sowie die von M berechnete *partielle Funktion* wie bei den TMs definieren.

REGISTEROPERATOREN

In Analogie zu den Turingoperatoren, können wir über die elementaren Operationen und Tests der k -Registermaschinen spezielle partielle Speichertransformationen $P : \mathbb{N}^k \rightarrow \mathbb{N}^k$, die sogenannten *k -Registeroperatoren* definieren. Diese erhält man induktiv ausgehend aus den elementaren Operationen durch Hintereinanderausführung bzw. Iteration (wobei so lange iteriert wird, bis ein ausgewähltes Register leer ist.)

INDUKTIVE DEFINITION DER k -REGISTEROPERATOREN (k -ROs):

1. $P \in \{a_1, \dots, a_k, s_1, \dots, s_k\}$, wobei

$$\begin{aligned} a_l(r_1, \dots, r_k) &= (r_1, \dots, r_{l-1}, r_l + 1, r_{l+1}, \dots, r_k) \\ s_l(r_1, \dots, r_k) &= (r_1, \dots, r_{l-1}, r_l - 1, r_{l+1}, \dots, r_k) \end{aligned}$$

„inkrementiere bzw. dekrementiere Register l “

2. Sind P_1 und P_2 k -ROs, so auch P_1P_2 , wobei

$$P_1P_2(\vec{r}) = P_2(P_1(\vec{r}))$$

„erst P_1 , dann P_2 “

3. Ist P ein k -RO und $1 \leq l \leq k$, so ist auch $[P]_l$ ein k -RO, wobei

$$[P]_l(\vec{r}) = \text{Iter}_{t_l}(P)(\vec{r})$$

„iteriere P so lange, bis das l te Register leer ist“

BESCHRÄNKTE ITERATION UND PRIMITIVE REGISTEROPERATOREN

Bei einem Iterationsoperator $[P]_l$ kann man i.a. nicht a priori sagen, wie oft der innere Operator P ausgeführt wird (*while*-Schleife). Dies können wir jedoch durch folgende Einschränkung der Iteration vermeiden:

3'. Ist P ein k -RO, in dem die Operatoren a_l und s_l nicht vorkommen, so ist auch $[s_l P]_l$ ein k -RO.

Wird hier $[s_l P]_l$ auf $\vec{r} = (r_1, \dots, r_k)$ angewandt, so wird der innere Operator P r_l -mal hintereinander ausgeführt (und das l te Register ist nach Ausführung von $[s_l P]_l$ leer) (*for*-Schleife).

Ersetzt man in der Definition der k -Registeroperatoren die Iterationsklausel 3. durch die beschränkte Iteration 3'., so erhält man die *primitiven k -Registeroperatoren* (k -PROs).

BEMERKUNGEN zu den primitiven Registeroperatoren:

1. k -PROs sind total! (Triviale Induktion nach Aufbau der k -PROs.) k -ROs können dagegen partiell sein (Beispiel: $a_1[a_1]_1$).
2. Jeder k -PRO ist ein k -RO.
3. In der beschränkten Iteration $[s_l P]_l$ lassen wir auch den leeren Operator P (= Identitätsfunktion) zu. D.h. $[s_l]_l$ ist ein k -PRO. Offensichtlich gilt $[s_l]_l = [s_l a_n s_n]_l$ (für $n \neq l$).
4. Ist $[s_l P]_l$ ein primitiver RO, so ist $[s_l P]_l$ zu $[P s_l]_l$ äquivalent: Da der Teiloperator P nicht auf Register l zugreift, spielt es keine Rolle, ob wir dieses vor oder nach Ausführung von P dekrementieren. Wir akzeptieren daher i.a. auch $[P s_l]_l$ als PRO.

VON (P)ROs BERECHNETE FUNKTIONEN

Für $n < k$ ist die von dem k -(P)RO P berechnete partielle n -stellige Funktion

$$\text{res}_P^n : \mathbb{N}^n \rightarrow \mathbb{N}$$

die Funktion

$$\text{res}_P^n(\vec{m}) = (P(\vec{m}, 0^{k-(n+1)}))_{n+1},$$

wobei $(r_1, \dots, r_k)_{n+1} = r_{n+1}$ sei.

Gilt hierbei für alle $\vec{m} \in \text{Db}(\text{res}_P^n)$

$$P(\vec{m}, 0^{k-(n+1)}) = (\vec{m}, \text{res}_P^n(\vec{m}), 0^{k-(n+1)}),$$

so berechnet P res_P^n konservativ.

NB. Bei der Definition der von dem RO P berechneten Funktion wird dieser mit der Ein- und Ausgabefunktion für Registermaschinen kombiniert.

BEISPIELE

1. Der primitive 3-Registeroperator

$$Sum = [s_1 a_3]_1 [s_2 a_3]_2$$

berechnet die Summe zweier natürlicher Zahlen, wobei er wie die im vorangegangenen Beispiel beschriebene RM vorgeht. *Sum* ist nicht konservativ, da die Eingaben durch die Rechnung zerstört werden:

$$Sum(m_1, m_2, 0) = (0, 0, m_1 + m_2).$$

2. Der PRO $[s_i]_i$ leert das *i*te Register.

3. Der PRO

$$TL_{i \rightarrow j} = [s_j]_j [s_i a_j]_i,$$

kopiert Register *i* in Register *j* ($j \neq i$), wobei Register *i* gelöscht wird (*Registertransfer* mit Löschen):

$$TL_{i \rightarrow j}(\dots r_i \dots r_j \dots) = (\dots 0 \dots r_i \dots)$$

4. Der primitive Registeroperator

$$T_{i \rightarrow j, h} = [s_j]_j [s_h]_h [s_i a_j a_h]_i [s_h a_i]_h$$

beschreibt den *Registertransfer* (ohne Löschen) von Register i in Register j unter Verwendung von Register h als Hilfsregister (i, j, h paarweise verschieden):

$$T_{i \rightarrow j, h}(\dots r_i \dots r_j \dots r_h \dots) = (\dots r_i \dots r_i \dots 0 \dots)$$

5. Mit Hilfe von Registertransfers kann man den PRO Sum in einen PRO zur konservativen Berechnung der Addition umwandeln, indem man Kopien der Summanden in zwei zusätzlichen Hilfsregistern zwischenspeichert:

$$Sum_{kon} = T_{1 \rightarrow 4, 3} T_{2 \rightarrow 5, 3} Sum TL_{4 \rightarrow 1} TL_{5 \rightarrow 2}$$

Analog kann man jede (P)RO-Berechnung in eine konservative (P)RO-Berechnung überführen.

BEZIEHUNGEN

Wie bei dem Turingmaschinenmodell sind Registermaschinen und Registeroperatoren gleichmächtig. Weiterhin sind Register- und Turingmaschinen gleichmächtig. Das Konzept der primitiven Registeroperatoren wird sich dagegen als schwächer herausstellen.

Hier beobachten wir zunächst nur die folgenden Inklusionen

$$(1) F(\text{RO}) \subseteq F(\text{RM})$$

$$(2) F(\text{RO}) \subseteq F(\text{TO})$$

$$(3) F(\text{RM}) \subseteq F(\text{TM})$$

wobei (3) in den Übungen gezeigt wird. Die Umkehrungen werden sich später aus einem Ringschluss ergeben.

Hierbei sind $F(\text{RO})$ und $F(\text{RM})$ die Klassen der von Registeroperatoren bzw. Registermaschinen berechneten partiellen Funktionen.

REGISTEROPERATOREN VS. REGISTERMASCHINEN

SATZ. $F(\text{RO}) \subseteq F(\text{RM})$.

Da Ein- und Ausgabe bei Registeroperatoren und Registermaschinen gleich definiert sind, genügt es folgendes Lemma zu zeigen.

LEMMA. Zu jedem k -RO P gibt es eine k -RM M mit Startzustand α_M und ausgezeichnetem (Stopp-)Zustand ω_M , sodass für alle $\vec{r} \in \mathbb{N}^k$ gilt:

(i) Ist $P(\vec{r})$ definiert, so ist die mit (\vec{r}, α_M) beginnende Rechnung von M endlich und endet mit der Stoppkonfiguration $(P(\vec{r}), \omega_M)$.

(ii) Ist $P(\vec{r})$ undefiniert, so ist die mit (\vec{r}, α_M) beginnende Rechnung von M unendlich.

Der Beweis ist sehr ähnlich zu dem Beweis der entsprechenden Aussage für Turingoperatoren bzw. -maschinen.

BEWEIS DES LEMMAS:

Der Beweis ist durch Induktion nach Aufbau (= Länge) des RO P .

1. $P = a_i$ oder $P = s_i$ ($1 \leq i \leq k$). Dann besteht das Programm δ von M aus der Instruktion $(\alpha_M, a_i, \omega_M)$ bzw. $(\alpha_M, s_i, \omega_M)$.

2. $P = P_1P_2$. Nach I.V. gibt es dann die die ROs P_i simulierenden Maschinen M_i mit Zustandsmengen Z_i und ausgezeichneten Zuständen α_{M_i} und ω_{M_i} ($i = 1, 2$). Durch eventuelles Umbenennen der Zustände können wir erreichen, dass $Z_1 \cap Z_2 = \{\omega_{M_1}\}$ und $\omega_{M_1} = \alpha_{M_2}$ gilt. Das Programm von M ist dann die Vereinigung der Programme von M_1 und M_2 , $\alpha_M = \alpha_{M_1}$ und $\omega_M = \omega_{M_2}$.

3. $P = [P_1]_i$ ($1 \leq i \leq k$). Nach I.V. gibt es dann eine den RO P_1 simulierende Maschine M_1 mit ausgezeichneten Zuständen α_{M_1} und ω_{M_1} . Das Programm der Maschine M erhält man aus dem Programm von M_1 durch Hinzufügen der Instruktion

$$(\alpha_M, t_i, \alpha_{M_1}, \omega_M),$$

wobei $\alpha_M = \omega_{M_1}$ während ω_M neu ist.

REGISTEROPERATOREN VS. TURINGOPERATOREN

SATZ. $F(\text{RO}) \subseteq F_{\text{kon}}(\text{TO})$.

Kern des Beweises ist das folgende Lemma.

LEMMA. Zu jedem k -RO P gibt es eine TO P' , sodass für $\vec{n} \in \mathbb{N}^k$ gilt:

$$P'(\dots \underset{\uparrow}{b\vec{n}b} \dots) = \begin{cases} \dots \underset{\uparrow}{bP(\vec{n})b} \dots & \text{falls } \vec{n} \in \text{Db}(P) \\ \uparrow & \text{sonst.} \end{cases}$$

BEWEIS DES SATZES unter Verwendung des Lemmas

Sei $\varphi^{(m)} \in F(\text{RO})$, sei P ein k -RO ($k \geq m + 1$), der φ (o.B.d.A.) konservativ berechnet, und sei P' zugehöriger TO gemäß Lemma. Definiere:

$$P_\alpha := R_{bb}(ROR)^{k-m}LL_{bb}$$

“schreibe $(k - m)$ -mal $\underline{0}$ (= 0) an das rechte Bandende”

$$P_\omega := (RR_b)^m$$

“gehe um m Zahldarstellungen nach rechts”

$$P_\varphi := P_\alpha P' P_\omega$$

Dann gilt für $\vec{n} = (n_1, \dots, n_m) \in Db(\varphi)$ und $\vec{0} = (0, \dots, 0) \in \mathbb{N}^{k-(m+1)}$:

$$\begin{aligned} P_\varphi(\dots \underset{\uparrow}{b\vec{n}b} \dots) &= P_\alpha P' P_\omega(\dots \underset{\uparrow}{b\vec{n}b} \dots) \\ &= P' P_\omega(\dots \underset{\uparrow}{b(\vec{n}, \vec{0}, 0)b} \dots) \\ &= P_\omega(\dots \underset{\uparrow}{bP(\vec{n}, \vec{0}, 0)b} \dots) = P_\omega(\dots \underset{\uparrow}{b(\vec{n}, \varphi(\vec{n}), \vec{0})b} \dots) \\ &= P_\omega(\dots \underset{\uparrow}{b\vec{n}b\varphi(\vec{n})b\vec{0}b} \dots) \\ &= \dots \underset{\uparrow}{b\vec{n}b\varphi(\vec{n})b\vec{0}b} \dots \end{aligned}$$

\Rightarrow Der TO P_φ berechnet φ .

BEWEIS DES LEMMAS durch Induktion nach Aufbau des k -RO P .

$P = a_i$:

$$\text{Ziel: } P'(\dots \underline{bn_1b} \dots \underline{n_ib} \dots \underline{n_kb} \dots) = \dots \underline{bn_1b} \dots \underline{n_i + 1b} \dots \underline{n_kb} \dots$$

Der Turingoperator P' arbeitet wie folgt: Die ersten i Zahlen (d.h. genauer Zahldarstellungen) werden an das Bandende kopiert und die i -te Zahl dort um 1 erhöht. Dann werden die restlichen $k - i$ Zahlen kopiert und schließlich die Originale gelöscht:

$$P' = K^i R_{bb} 0 (L_b L)^k R K^{k-i} E^k$$

$P = s_i$: analog

$$P' = K^i R_{bb} L L [R b]_b R_{bb} L (L_b L)^k R K^{k-i} E^k$$

$$P = P_1 P_2:$$

$$\text{Ziel: } P'(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots) = \dots \underbrace{bP(\vec{n})b}_{\uparrow} \dots$$

Seien P'_1 und P'_2 die nach I.V. existierenden TOs, die P_1 und P_2 simulieren. Dann gilt:

$$\begin{aligned} \dots \underbrace{bP(\vec{n})b}_{\uparrow} \dots &= \dots \underbrace{bP_2(P_1(\vec{n}))b}_{\uparrow} \dots && \text{(nach Def. von } P) \\ &= P'_2(\dots \underbrace{bP_1(\vec{n})b}_{\uparrow} \dots) && \text{(nach I.V.)} \\ &= P'_2(P'_1(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots)) && \text{(nach I.V.)} \end{aligned}$$

$\Rightarrow P' := P'_1 P'_2$ geeignet

$$P = [P_1]_i:$$

$$\text{Ziel: } P'(\dots \underset{\uparrow}{b\vec{n}b} \dots) = \dots \underset{\uparrow}{bP(\vec{n})b} \dots$$

Sei P'_1 der nach I.V. existierende TO, der P_1 simuliert. Dann gilt

$$\begin{aligned} \dots \underset{\uparrow}{bP(\vec{n})b} \dots &= \dots \underset{\uparrow}{bP_1^m(\vec{n})b} \dots \\ &= P_1'^m(\dots \underset{\uparrow}{b\vec{n}b} \dots) \quad (\text{nach I.V.}) \end{aligned}$$

für das kleinste (von \vec{n} abhängende) $m \geq 0$, sodass die i -te Zahldarstellung auf dem Band Null ist.

$$\Rightarrow P' := (R R_b)^i L L [L_{bb} P'_1 (R R_b)^i L L]_b L_{bb} \text{ geeignet.}$$

(Ende Beweis Lemma)