

---

## 7. Registermaschinen

---

In diesem Abschnitt geben wir eine alternative Formalisierung der berechenbaren zahlentheoretischen Funktionen  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ . Hierzu entwerfen wir wiederum eine geeignete (Familie von) Basismaschine(n), gehen jedoch hier einen abstrakteren Weg als bei den Turingmaschinen. Insbesondere ignorieren wir die Frage der Zahldarstellung, d.h. die Frage, wie Zahlen intern im Speicher der Maschine repräsentiert werden und wie der Zugriff auf den Speicher erfolgt. Dass dies möglich ist, ergibt sich aus der Beobachtung, dass sich alle berechenbaren Funktionen auf  $\mathbb{N}$  auf die Grundoperationen des Hoch- und Herunterzählens (In- bzw. Dekrementieren) reduzieren lassen, wenn man zusätzlich noch feststellen kann, ob eine Zahl die Null ist.<sup>1</sup> Eine Registermaschine, wie unser neuer Maschinentyp heißt, besitzt als Speicher eine feste Anzahl von Speicherzellen, genannt Register, von denen jede eine natürliche Zahl aufnehmen kann. Grundoperationen sind das In- und Dekrementieren sowie der Nulltest für jedes Register. Die  $n$  Eingaben schreibt man in die ersten  $n$  Register, die Ausgabe entnimmt man dem  $(n + 1)$ -ten Register. Die übrigen Register (soweit vorhanden) dienen zum Speichern von Hilfsgrößen. „Leere“ Register enthalten die Null. Das Inkrementieren ist die Funktion  $f(n) = n + 1$ , die üblicherweise *Nachfolgerfunktion* genannt und mit  $S$  (successor) bezeichnet wird. Das Dekrementieren ist die duale Funktion  $f(n) = n \dot{-} 1$ , die den *Vorgänger* berechnet. Hierbei greift man auf die auf  $\mathbb{N}$  beschränkte Subtraktion  $\dot{-}$  zurück, bei der negative Ergebnisse durch die Null ersetzt werden:

$$n \dot{-} m := \begin{cases} n - m & \text{falls } n \geq m \\ 0 & \text{sonst.} \end{cases}$$

7.1 DEFINITION. Die *k-Register-Basismaschine* zur Berechnung  $n$ -stelliger (partieller) Funktionen  $\varphi : \mathbb{N}^n \rightarrow \mathbb{N}$  (mit  $n < k$ ) ist die Basismaschine

$$RB(k, n) = (I, O, S, \text{in}, \text{out}, \text{OPER}, \text{TEST})$$

wobei

- $I = \mathbb{N}^n$
- $O = \mathbb{N}$
- $S = \mathbb{N}^k$   
Für  $s = (n_1, \dots, n_k)$  heißt  $n_l$  der Wert des  $l$ -ten Register bei der Speicherbelegung  $s$ .
- $\text{in} : \mathbb{N}^n \rightarrow \mathbb{N}^k$  ist gegeben durch  $\text{in}(m_1, \dots, m_n) = (m_1, \dots, m_n, 0, \dots, 0)$ .
- $\text{out} : \mathbb{N}^k \rightarrow \mathbb{N}$  ist gegeben durch  $\text{out}(m_1, \dots, m_k) = m_{n+1}$ .

---

<sup>1</sup>Dies ist die bereits erwähnte Idee mathematische Strukturen als Datentyp zu beschreiben, wobei hier  $\mathbb{N}$  die Objektmenge ist und das In- und Dekrementieren sowie der Nulltest die Grundoperationen.

- OPER =  $\{a_l : 1 \leq l \leq k\} \cup \{s_l : 1 \leq l \leq k\}$ , wobei  
 $a_l(m_1, \dots, m_k) = (m_1, \dots, m_{l-1}, m_l + 1, m_{l+1}, \dots, m_k)$   
 („Inkrementiere Register  $l$ “)  
 und  
 $s_l(m_1, \dots, m_k) = (m_1, \dots, m_{l-1}, m_l - 1, m_{l+1}, \dots, m_k)$   
 („Dekrementiere Register  $l$ “).
- TEST =  $\{t_l : 1 \leq l \leq k\}$ , wobei  $t_l(m_1, \dots, m_k) = 1$  g.d.w.  $m_l = 0$  („Teste, ob das Register  $l$  leer ist“).

7.2 BEISPIEL. Eine Registermaschine  $M = (B, P)$  zur Berechnung der Summe zweier natürlicher Zahlen benutzt  $RB(3, 2)$  als Basismaschine. Es werden die beiden Eingaberegister nacheinander auf Null heruntergezählt und gleichzeitig in jedem Schritt das Ausgaberegister entsprechend hochgezählt. Hierbei ist vor jedem Schritt zu testen, ob das zu dekrementierende Register nicht bereits mit 0 belegt, d.h. leer ist. Das Programm  $P$  besteht also aus den Instruktionen

(0,	$t_1$ ,	1,	3)	Teste, ob 1. Register leer
(1,	$s_1$ ,	2)		Dekrementiere 1. Register
(2,	$a_3$ ,	0)		Inkrementiere 3. Register
(3,	$t_2$ ,	4,	6)	Teste, ob 2. Register leer
(4,	$s_2$ ,	5)		Dekrementiere 2. Register
(5,	$a_3$ ,	3)		Inkrementiere 3. Register

Statt Registermaschinen betrachten wir im Folgenden die komfortableren Registeroperatoren, die die iterative Ausführung der Registermaschinenoperationen erlauben. Diese sind entsprechend den Turingoperatoren für das Registermaschinenmodell definiert. Registermaschinen und Registeroperatoren sind äquivalente Konzepte, was wir hier jedoch nicht beweisen werden. (Die Inklusionen  $F(RO) \subseteq F(RM) \subseteq F(TM)$  werden jedoch in den Übungen gezeigt, woraus sich die Äquivalenz später ergeben wird.)

Die  $n$ -Registeroperatoren  $P$  sind induktiv wie folgt definiert. Zusammen mit den Operatoren  $P$  geben wir jeweils die zugehörige partielle Speichertransformation  $P : S \rightarrow S$ , d.h.  $P : \mathbb{N}^n \rightarrow \mathbb{N}^n$ , an, die wir ebenfalls mit  $P$  bezeichnen.

7.3 DEFINITION.  $P$  ist ein  $n$ -Registeroperator ( $n$ -RO), falls einer der folgenden Fälle zutrifft:

1.  $P \in \{a_1, \dots, a_n, s_1, \dots, s_n\}$ , wobei

$$\begin{aligned} a_l(m_1, \dots, m_n) &= (m_1, \dots, m_{l-1}, m_l + 1, m_{l+1}, \dots, m_n) \\ s_l(m_1, \dots, m_n) &= (m_1, \dots, m_{l-1}, m_l - 1, m_{l+1}, \dots, m_n) \end{aligned}$$

(„inkrementiere bzw. dekrementiere Register  $l$ “)

2.  $P = P_1 P_2$  für  $n$ -ROs  $P_1$  und  $P_2$ , wobei

$$P_1 P_2(m_1, \dots, m_n) = P_2(P_1(m_1, \dots, m_n))$$

(„erst  $P_1$ , dann  $P_2$ “)

3.  $P = [P_1]_l$  für einen  $n$ -RO  $P_1$  und  $1 \leq l \leq n$ , wobei

$$[P_1]_l(m_1, \dots, m_n) = \text{Iter}_l(P_1)(m_1, \dots, m_n)$$

(„iteriere  $P_1$  solange, bis das Register  $l$  leer ist“)

Als Spezialfall der Registeroperatoren führen wir die primitiven Registeroperatoren ein. Bei diesen liegt im Falle einer Iteration die Anzahl der Iterationsschritte bereits zu Beginn der Ausführung der iterativen Anweisung fest. Man erhält hier also nur *for-Schleifen* oder, wie man auch sagt, *(for-)loops*, keine *while-* oder *repeat-until-Schleifen*.  
Eine *for-Schleife*

$$\text{for } i = 1, \dots, n \text{ do } \{f\}$$

(wobei  $f$  die Zählvariable  $i$  nicht verändern darf) lässt sich generell mit Hilfe einer *while-Schleife* durch

$$\begin{aligned} & i := 1; \\ & \text{while } i \leq n \text{ do } \{i := i + 1; f\} \end{aligned}$$

beschreiben. Entsprechend beschreibt in unserer Notation der RO  $[s_l P]_l$ , wobei  $P$  nicht auf Register  $l$  zugreift (d.h. die Befehle  $a_l$  und  $s_l$  nicht enthält), dass  $P$   $n$ -mal ausgeführt wird, wobei  $n$  die Belegung von Register  $l$  bei Schleifeneintritt ist. In jedem Schleifendurchlauf wird nämlich Register  $l$  durch Ausführung des Befehls  $s_l$  dekrementiert und danach durch  $P$  nicht mehr verändert.

7.4 DEFINITION. Ein *primitiver  $n$ -Registeroperator ( $n$ -PRO)*  $P$  ist wie ein  $n$ -RO definiert, wobei jedoch die 3. Klausel in der Definition wie folgt zu ersetzen ist.

3.  $P = [s_l P_1]_l$  für einen  $n$ -PRO  $P_1$ , in dem die Operatoren  $a_l$  und  $s_l$  nicht vorkommen, und  $1 \leq l \leq n$ .

Man beachte, dass im Gegensatz zu allgemeinen Registeroperatoren die primitiven Registeroperatoren stets total sind. Registeroperatoren beschreiben Speichertransformationen. Die von einem (P)RO berechnete Funktion erhält man durch Hinzunahme der Ein- und Ausgabefunktion wie bei Registermaschinen.

7.5 DEFINITION. Für  $n < k$  ist die von dem  $k$ -RO  $P$  berechnete *partielle  $n$ -stellige Funktion*  $\text{res}_P^n : \mathbb{N}^n \rightarrow \mathbb{N}$  die Funktion

$$\text{res}_P^n(\vec{m}) = \text{out}(P(\text{in}(\vec{m}))),$$

wobei  $\text{in}$  und  $\text{out}$  die Ein- bzw. Ausgabefunktion der Registerbasismaschine  $RB(k, n)$  sind. Gilt hierbei für alle  $\vec{m} \in \text{Db}(\text{res}_P^n)$

$$P(\vec{m}, 0, \dots, 0) = (\vec{m}, \text{res}_P^n(\vec{m}), 0, \dots, 0),$$

so berechnet  $P$   $\text{res}_P^n$  konservativ.

Mit  $F(\text{RO})$  und  $F_{\text{kon}}(\text{RO})$  bezeichnen wir die von ROs berechneten bzw. konservativ berechneten partiellen Funktionen.  $F(\text{PRO})$  und  $F_{\text{kon}}(\text{PRO})$  sind entsprechend für primitive Registeroperatoren definiert. Wie oben bereits bemerkt, sind primitive Registeroperatoren stets total, berechnen also nur totale Funktionen. Es gilt daher trivialerweise  $F(\text{PRO}) \subset F(\text{RO})$ , da z.B. die nirgends definierte Funktion  $\varphi : \mathbb{N} \rightarrow \mathbb{N}$  von dem nirgends terminierenden RO  $a_1[a_1]_1$  berechnet wird. Später werden wir zeigen, dass es auch *totale* Funktionen gibt, die RO-berechenbar, aber nicht PRO-berechenbar sind, d.h. dass  $F(\text{PRO}) \subset F_{\text{tot}}(\text{RO})$  gilt.

## 7.6 BEISPIEL.

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 vorangehenden Beispiel beschriebene Registermaschine vorgeht.  $Sum$  ist nicht konservativ, da die Eingaben durch die Rechnung zerstört werden.

2. Der Registeroperator
- $[s_l]_l$
- , der das Register
- $l$
- leert, ist primitiv. Der iterierte Operator
- $P$
- ist hier leer, was formal nicht zulässig ist. Wir können jedoch
- $P = a_j s_j$
- für ein Register
- $j \neq l$
- setzen, ohne die Wirkung von
- $[s_l]_l$
- zu verändern, d.h.
- $[s_l]_l = [s_l a_j s_j]$
- . Wir werden dies im Folgenden stillschweigend verwenden.

3. Der primitive Registeroperator

$$T_{i \rightarrow j, k} = [s_j]_j [s_k]_k [s_i a_j a_k]_i [s_k a_i]_k$$

beschreibt den *Registertransfer* von Register  $i$  in Register  $j$  unter Verwendung von Register  $k$  als Hilfsregister ( $i, j, k$  paarweise verschieden): Nach Ausführung von  $T_{i \rightarrow j, k}$  steht der ursprüngliche Inhalt von Register  $i$  nun sowohl in Register  $i$  wie auch in Register  $j$ . Das Register  $k$  ist leer. (Hierzu werden zunächst die Register  $j$  und  $k$  geleert, dann Register  $i$  sowohl in Register  $j$  wie in Register  $k$  kopiert, wobei Register  $i$  allerdings gelöscht wird. Zum Ende muss also Register  $k$  in Register  $i$  zurückkopiert werden, wobei nun Register  $k$  gelöscht wird.)

Muss der alte Registerinhalt nicht erhalten bleiben, so verwendet man den Operator

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

der Register  $i$  in Register  $j$  kopiert und dabei Register  $i$  löscht. Hierzu wird kein Hilfsregister benötigt.

Registertransfers sind ein wichtiges Hilfsmittel bei (konservativen) RO-Berechnungen.

4. Mit Hilfe von Registertransfers kann man den PRO
- $Sum$
- in einen PRO zu konservativen Berechnung der Addition umwandeln, indem man Kopien der Summanden in zwei Hilfsregistern zwischenspeichert:

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

Das letzte Beispiel lässt sich leicht verallgemeinern: Zu jedem (P)RO  $P$ , der eine  $n$ -stellige partielle Funktion  $\varphi$  berechnet, können wir einen (P)RO  $P'$  angeben, der  $\varphi$  konservativ berechnet.  $P'$  benutzt  $n$  zusätzliche Hilfsregister, in die vor Ausführung von  $P$  die Eingaben kopiert werden. Nach Ausführung von  $P$  werden alle von  $P$  benutzten Register mit Ausnahme des Ausgaberegisters gelöscht, und schließlich die Eingaben aus den  $P'$ -Hilfsregistern in die Eingaberegister zurücktransferiert.

7.7 LEMMA.  $F((P)RO) = F_{kon}((P)RO)$ .

Bei dem Nachweis der Äquivalenz von Turingmaschinen und Registermaschinen beschränken wir uns wie bereits bemerkt auf die entsprechenden Operatoren. Hier zeigen wir zunächst, wie man Registeroperatoren durch Turingoperatoren simulieren kann. Auf die Umkehrung werden wir später zurückkommen.

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

Kern des Beweises ist das folgende Lemma.

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

$$\begin{aligned} P^l(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots) &= \dots \underbrace{bP(\vec{n})b}_{\uparrow} \dots, & \text{falls } \vec{n} \in \text{Db}(P) \\ \text{und} \\ P^l(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots) &\uparrow & \text{sonst.} \end{aligned}$$

BEWEIS VON SATZ 7.8 (MIT LEMMA 7.9). Sei  $\varphi \in F(\text{RO})$   $m$ -stellig. Nach Lemma 7.7 gibt es einen  $k$ -RO  $P$ , der  $\varphi$  konservativ berechnet ( $k > m$  geeignet). Sei  $P^l$  der TO, der  $P$  gemäß Lemma 7.9 simuliert. Dann berechnet der TO  $P^l = P_+ P^l P_-$  die Funktion  $\varphi$  konservativ, wobei  $P_+$  ( $k - m$ )-mal die  $\underline{0}$  ( $= 1$ ) an das Bandende schreibt und  $P_-$  ( $k - m - 1$ )-mal die  $\underline{0}$  am Bandende löscht. Für eine Eingabe  $\vec{n} = (n_1, \dots, n_m) \in \text{Db}(\varphi)$  und für  $\vec{0} = (0, \dots, 0) \in \mathbb{N}^{k-m-1}$  gilt dann nämlich

$$\begin{aligned} P^l(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots) &= P_+ P^l P_-(\dots \underbrace{b\vec{n}b}_{\uparrow} \dots) \\ &= P^l P_-(\dots \underbrace{b(\vec{n}, \vec{0}, 0)}_{\uparrow} b \dots) \\ &= P_-(\dots \underbrace{b(\vec{n}, \varphi(\vec{n}), \vec{0})}_{\uparrow} b \dots) \\ &= \dots \underbrace{b(\vec{n}, \varphi(\vec{n}))}_{\uparrow} b \dots \end{aligned}$$

und  $P^l$  ist sonst undefiniert. Die Turingoperatoren  $P_+$  und  $P_-$  können mit Hilfe der in Kapitel 5 eingeführten Hilfsoperatoren wie folgt beschrieben werden:

$$P_+ = R_{bb}(R1R)^{k-m} LL_{bb} \quad \text{und} \quad P_- = R_{bb}(LbL)^{k-m-1} LL_{bb}.$$

□

BEWEIS VON LEMMA 7.9. Wir definieren den TO  $P^l$  durch Induktion nach dem Aufbau des  $k$ -RO  $P$ .

1.  $P = a_i$ . Der Turingoperator  $P^l$  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:

$$P^l = (K)^i R_{bb} 1 (L_b L)^k R (K)^{k-i} (E_i)^k R_1 L$$

2.  $P = s_i$ . Man verfährt analog. Nach Kopieren der  $i$ -ten Zahl an das Bandende prüft man nun, ob diese von Null verschieden ist und dekrementiert in diesem Fall, wobei man in jedem Fall nach dieser Operation vor der Kopie der  $i$ -ten Zahl stehen bleibt.

$$P^l = (K)^i R_{bb} LL [RbLL_b]_b (L_b L)^{k-1} R (K)^{k-i} (E_i)^k R_1 L$$

3.  $P = P_1 P_2$ . Dann ist  $P^l = P_1^l P_2^l$  für die nach Induktionsvoraussetzung gegebenen TOs  $P_1^l$  und  $P_2^l$ .

4.  $P = [P_1]_i$ . Hier iteriert  $P^i$  den nach I.V. existierenden Operator  $P_1^i$  so lange, bis die  $i$ -te Zahl erstmals Null ist:

$$P^i = (RR_b)^i LL[L_{bb}P_1^i(RR_b)^i LL]_b L_{bb}.$$

□

Die bisher gezeigten Beziehungen zwischen Turingmaschinen, Turingoperatoren und Registeroperatoren bzgl. der – evtl. konservativen – Berechnung von Funktionen auf den natürlichen Zahlen können wir wie folgt zusammenfassen:

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

(In den Übungen wird zusätzlich

$$F_{kon}(\text{RO}) \subseteq F_{kon}(\text{RM}) \subseteq F(\text{RM}) \subseteq F(\text{TM})$$

gezeigt.) Bevor wir die Äquivalenz von all diesen Konzepten zeigen, betrachten wir noch ein weiteres formales Berechnungskonzept.