

Inhaltsverzeichnis

1	Registermaschinen und μ-Rekursivität	1
1.1	Registermaschinen	1
1.1.1	Definition	3
1.1.2	Definition	4
1.1.3	Lemma	5
1.1.4	Definition	5
1.1.5	Lemma	6
1.1.6	Definition	8
1.1.7	Lemma	9
1.1.8	Lemma	10
1.1.9	Definition	12
1.1.10	Lemma	12
1.1.11	Lemma	13
1.1.12	Induktive Definition	14
1.1.13	Lemma	14
1.1.14	Satz	15
1.1.15	Aufgaben	15
1.2	Partielle Funktionen, Berechenbarkeit und μ -Rekursivität . .	16
1.2.1	Definition	16
1.2.2	Definition	17
1.2.3	Definition	17
1.2.4	Definition	18
1.2.5	Lemma	18
1.2.6	Definition der partiellen Gleichheit	20
1.2.7	Definition	20
1.2.8	Lemma	21
1.2.9	Definition	23
1.2.10	Lemma	25
1.2.11	Induktive Definition	26
1.2.12	Satz	26

1.2.13	Korollar	27
1.2.14	Definition	27
1.2.15	Lemma	28
1.2.16	Induktive Definition der μ -partiell-rekursiven Funktionen	28
1.2.17	Definition	29
1.2.18	Lemma	29
1.2.19	Satz	30
1.2.20	Aufgaben	31
1.3	Primitiv-rekursive Funktionen und Prädikate	31
1.3.1	Lemma	32
1.3.2	Lemma	32
1.3.3	Lemma	32
1.3.4	Beispiele	32
1.3.5	Lemma	33
1.3.6	Lemma	34
1.3.7	Definition des beschränkten μ -Operators	35
1.3.8	Lemma	35
1.3.9	Definition	36
1.3.10	Definition	36
1.3.11	Beispiele	36
1.3.12	Lemma	37
1.3.13	Lemma	37
1.3.14	Definition	38
1.3.15	Lemma	38
1.3.16	Bemerkungen	38
1.3.17	Definition der Tupel-Kodierung	39
1.3.18	Satz über Kodierung und Dekodierung	39
1.3.19	Lemma	40
1.3.20	Bemerkung	40
1.3.21	Definition	40
1.3.22	Lemma	41
1.3.23	Definition	41
1.3.24	Bemerkung	42
1.3.25	Lemma	42
1.3.26	Lemma	43
1.3.27	Definition	43
1.3.28	Lemma	44
1.3.29	Satz über Wertverlaufsrekursion	44
1.3.30	Definition	44
1.3.31	Aufgaben	45

2	Elementare Rekursionstheorie	47
2.1	Arithmetisierung und Kleene's Normalform-Theorem	47
2.1.1	Definition	48
2.1.2	Lemma	49
2.1.3	Definition	50
2.1.4	Definition	51
2.1.5	Lemma	51
2.1.6	Definition	51
2.1.7	Lemma	52
2.1.8	Definition	52
2.1.9	Lemma	52
2.1.10	Lemma	53
2.1.11	Kleene's Normalform-Theorem	54
2.1.12	Definition	54
2.1.13	Äquivalenz der Berechenbarkeitsbegriffe	55
2.1.14	Definition	55
2.1.15	Aufzählungstheorem für partiell-rekursive Funktionen.	55
2.1.16	Korollar	56
2.2	Rekursive und rekursiv aufzählbare Prädikate	56
2.2.1	Definition	57
2.2.2	Lemma	57
2.2.3	Korollar	57
2.2.4	Aufzählungs-Theorem von Kleene	58
2.2.5	Satz von Post	58
2.2.6	Lemma	59
2.2.7	Lemma	60
2.2.8	Satz	60
2.2.9	Definition	61
2.2.10	Satz	62
2.2.11	Definition	63
2.2.12	Satz	63
2.2.13	Satz	63
2.2.14	Lemma (Cantor)	64
2.2.15	Satz	64
2.2.16	Aufgabe	65
2.3	S-m-n- und Rekursions-Theorem und der Satz von Rice	65
2.3.1	Lemma	65
2.3.2	Korollar	66
2.3.3	Lemma	67
2.3.4	Lemma	67
2.3.5	S-m-n-Theorem	68

2.3.6	Korollar	69
2.3.7	Korollar	69
2.3.8	Korollar	69
2.3.9	Korollar	69
2.3.10	Korollar	69
2.3.11	Rekursions-Theorem	70
2.3.12	Korollar	71
2.3.13	Definition	72
2.3.14	Satz	72
2.3.15	Satz von Rice	73
2.3.16	Aufgaben	73
3	Einfache unlösbare Probleme	75
3.1	Halte-Probleme für Registermaschinen	75
3.1.1	Definition	75
3.1.2	Lemma	76
3.1.3	Satz: Unlösbarkeit des Halte-Problems	76
3.1.4	Lemma	77
3.1.5	Satz von Shepherdson und Sturgis	78
3.1.6	Korollar	80
3.1.7	Korollar	80
3.2	Wort-Probleme für Semi-Thue-Systeme und Halbgruppen . . .	81
3.2.1	Definition	81
3.2.2	Definition	82
3.2.3	Definition	82
3.2.4	Lemma	82
3.2.5	Lemma	83
3.2.6	Definition	83
3.2.7	Definition	84
3.2.8	Lemma	85
3.2.9	Lemma	85
3.2.10	Lemma	85
3.2.11	Definition	86
3.2.12	Lemma	87
3.2.13	Definition	87
3.2.14	Lemma	87
3.2.15	Lemma	88
3.2.16	Lemma	88
3.2.17	Definition	89
3.2.18	Definition	89
3.2.19	Definition	89

3.2.20	Definition	90
3.3	Entscheidungs-Probleme für mathematische Theorien	91
3.3.1	Definition	92
3.3.2	Definition	93
3.3.3	Definition	93
3.3.4	Lemma	94
3.3.5	Lemma	94
3.3.6	Lemma	94
3.3.7	Lemma	95
3.3.8	Lemma	95
4	Arithmetische Prädikate	99
4.1	Die arithmetische Hierarchie	99
4.1.1	Rekursive Definition der Klassen Σ_n und Π_n	100
4.1.2	Lemma	101
4.1.3	Lemma	101
4.1.4	Lemma	101
4.1.5	Lemma	102
4.1.6	Lemma	103
4.1.7	Arithmetisches Aufzählungs-Theorem.	103
4.1.8	Arithmetisches Hierarchie-Theorem (Kleene-Mostowski).	104
4.1.9	Definition	105
4.1.10	Korollar	105
4.1.11	Korollar	106
4.2	Entbehrlichkeit der primitiven Rekursion	106
4.2.1	Induktive Definition der streng μ -rekursiven Funktionen.	106
4.2.2	Lemma	107
4.2.3	Lemma	107
4.2.4	Lemma	108
4.2.5	Lemma	108
4.2.6	Lemma	108
4.2.7	Lemma	109
4.2.8	Definition	109
4.2.9	Lemma	109
4.2.10	Lemma	110
4.2.11	Chinesischer Restsatz	111
4.2.12	Definition der Gödel'schen β -Funktion.	111
4.2.13	Lemma	111
4.2.14	Satz (Gödel 1931)	112
4.2.15	Satz	112
4.3	$\Sigma(\mathcal{N})$ -Definitionen und Repräsentierbarkeit	114

4.3.1	Definition der Sprache $L(Z)$ der Zahlentheorie.	115
4.3.2	Induktive Definition der $\Sigma(\mathcal{N})$ -Formeln.	116
4.3.3	Definition	116
4.3.4	Lemma	116
4.3.5	Lemma	117
4.3.6	Satz	118
4.3.7	Korollar	119
4.3.8	Korollar	119
4.3.9	Korollar	119
4.3.10	Definition	120
4.3.11	Satz	120
4.3.12	Definition des Robinson-Formalismus ROB	121
4.3.13	Lemma	121
4.3.14	Lemma	122
4.3.15	Lemma	123
4.3.16	$\Sigma(\mathcal{N})$ - Vollständigkeit von ROB	123
4.3.17	Definition	124
4.3.18	Satz über schwache Repräsentierbarkeit.	124
4.3.19	Definition	125
4.3.20	Lemma	125
4.3.21	Satz über starke Repräsentierbarkeit.	126

**5 Unvollständigkeit und Widerspruchsfreiheit - Die Gödel-
schen Sätze** **129**

5.1	Arithmetisierung mathematischer Theorien	130
5.1.1	Definition	131
5.1.2	Lemma	131
5.1.3	Definition der Herleitungen in einer Theorie T	132
5.1.4	Definition	133
5.1.5	Lemma	134
5.1.6	Definition	134
5.1.7	Lemma	134
5.1.8	Definition	135
5.1.9	Lemma	135
5.1.10	Definition	136
5.1.11	Lemma	136
5.1.12	Lemma	136
5.1.13	Definition	137
5.1.14	Lemma	137
5.1.15	Definition	137
5.1.16	Lemma	138

5.1.17	Lemma	138
5.1.18	Definition	139
5.1.19	Lemma	139
5.1.20	Rekursive Definition der Arithmetisierung $\ulcorner \urcorner$ von Herleitungen H in T	140
5.1.21	Definition	140
5.1.22	Lemma	141
5.1.23	Satz	141
5.1.24	Definition	142
5.1.25	Lemma	142
5.1.26	Definition	143
5.1.27	Lemma	143
5.1.28	Satz	144
5.1.29	Definition	144
5.1.30	Lemma	144
5.2	Unentscheidbarkeit und Unvollständigkeit	145
5.2.1	Definition	146
5.2.2	Lemma	146
5.2.3	Definition	146
5.2.4	Lemma	146
5.2.5	Unentscheidbarkeitssatz (Church 1935).	147
5.2.6	Satz	148
5.2.7	1. Gödelscher Unvollständigkeitssatz (1931, in der Fas- sung von Rosser 1936)	148
5.3	Anhang: Die Loop-Hierarchie	149
5.3.1	Induktive Definition der n -Loop-Maschinen ($n - LM$)	150
5.3.2	Definition	150
5.3.3	Lemma	150
5.3.4	Lemma	151
5.3.5	Lemma	152
5.3.6	Lemma	152
5.3.7	Induktive Definition der n -rekursiven Funktionen	152
5.3.8	Definition der simultanen Iteration	153
5.3.9	Lemma	154
5.3.10	Definition der "kleinen" Ackermannfunktionen $B_n (n \in$ $\mathbb{N})$	154
5.3.11	Definition	157
5.3.12	Lemma	157
5.3.13	Definition	159

Kapitel 1

Registermaschinen und μ -Rekursivität

1.1 Registermaschinen

Man nennt eine Funktion f *berechenbar*, wenn es ein effektives Verfahren (einen Algorithmus) gibt, das zu gegebenem Argument x in endlicher Zeit den Funktionswert $f(x)$ liefert. Als erste Präzisierung dessen, was mit "effektiven Verfahren" gemeint ist, betrachten wir idealisierte Rechenmaschinen, die nach einer eindeutigen Vorschrift, dem Programm, das ihnen eingegebene Material, die Speicherinhalte, in einfachen Rechenschritten bearbeiten. Dabei kann der Inhalt eines Speichers zur Bestimmung des nächsten Rechenschrittes herangezogen werden.

Als erster entwickelte um 1936 der englische Mathematiker Turing ein Konzept einer solchen idealisierten Rechenmaschine. Die Turing-Maschinen arbeiten auf einem einzigen Band und können ihr Arbeitsfeld in einem Rechenschritt nur um ein Feld nach rechts oder links verlagern. Wir betrachten hier statt dessen die seit 1961 von Minsky, Rödding, Shepherdson und Sturgis entwickelten Registermaschinen. Sie haben mehr Ähnlichkeit mit realen Rechenmaschinen, und einfache Programme lassen sich für sie leichter schreiben. Aus diesen und auch aus tiefer liegenden Gründen sind sie in den letzten Jahren immer mehr in den Vordergrund getreten.

Wir beginnen mit einer losen Beschreibung, die noch keine strenge mathematische Definition darstellt.

Beschreibung einer Registermaschine. Eine Registermaschine besitzt endlich viele Register R_1, \dots, R_N , von denen jedes genau eine natürliche Zahl

enthält (speichert). Der Index i des i -ten Registers R_i wird die **Adresse** von R_i genannt. Eine Registermaschine kann für jede Adresse i folgende **Rechenschritte** ausführen:

1. **Addierschritt:** Sie kann zum Inhalt der i -ten Registers 1 hinzuzählen;
2. **Subtrahierschritt:** Sie kann vom Inhalt des i -ten Registers 1 abziehen, falls dieser positiv ist, und nichts tun, falls dieser 0 ist;
3. **Testschritt:** Sie kann prüfen (testen), ob der Inhalt des i -ten Registers positiv oder 0 ist.

Dies sind die elementaren Operationen, die eine Registermaschine ausführen kann. Die sämtlichen von einer Registermaschine ausführbaren Operationen erhält man, wenn die Registermaschine mehrere elementare Operationen nacheinander ausführt.

In welcher Reihenfolge und an welchen Registern die Registermaschine diese Rechenschritte ausführt, wird durch das **Programm** der Registermaschine geregelt: Das Programm besteht aus endlich vielen **Instruktionen** I_0, \dots, I_{s-1} , die enthalten

1. die Nummer der Instruktion,
2. den auszuführenden Rechenschritt und
3. die Nummer der nächsten zu bearbeitenden Instruktion (Folgeinstruktion), falls der Rechenschritt von der Art 1. oder 2. ist, bzw. die Nummern von zwei Folgeinstruktionen, falls der Rechenschritt ein Testschritt 3. ist; von diesen ist die erste zu bearbeiten, falls die Prüfung einen positiven Registerinhalt ergibt, sonst die zweite.

Es ist auch erlaubt, dass die aufgerufene Nummer gleich s , also nicht mehr Nummer einer Instruktion ist. In diesem Fall **stoppt** die Maschine, sie hört auf zu rechnen.

Eine Instruktion läßt sich hiernach durch 4 Zeichen mitteilen, etwa in der Gestalt

1. **Addier-Instruktion:** $ji + l$
2. **Subtrahier-Instruktion:** $ji - l$ oder
3. **Test-Instruktion:** $jikl$

Hierbei bezeichnet jeweils j die Nummer der Instruktion, i die Adresse des zu bearbeitenden Registers; $+$ und $-$ teilen mit, dass zum Inhalt dieses i -ten Registers 1 zu addieren bzw. von ihm (soweit möglich) 1 zu subtrahieren ist, und l ist die Nummer der im Fall 1. und 2. einzigen Folgeinstruktion; im Fall 3. ist als nächstes die Instruktion mit der Nummer k zu bearbeiten, sofern der Inhalt des i -ten Registers positiv war, sonst die mit der Nummer l .

Das Wesentliche an einer Registermaschine ist also ihr Programm. Die Register sind dagegen nur Stellen, an denen je eine natürliche Zahl steht. Die Adressen aller Register, die im Laufe der Rechnung eventuell angesprochen werden oder deren Inhalt durch eine Test-Instruktion 3. die weitere Rechnung beeinflussen kann, müssen im Programm auftreten, nämlich als zweites Zeichen in einer Instruktion. Als "relevante" Registerzahl einer Registermaschine kann man also das Maximum aller in ihrem Programm auftretenden Adressen bezeichnen. Dann kann man eine Registermaschine mit ihrem Programm identifizieren, und diese Identifikation liefert die einfachste Möglichkeit, Registermaschinen zu definieren.

1.1.1 Definition

Eine **Registermaschine** M , auch (Register-)Programm genannt, ist eine endliche Folge I_0, \dots, I_{s-1} ($s \geq 0$) von Quadrupeln, den **Instruktionen** von M , der Gestalt

$$I_j \equiv j \ i \ b \ l$$

mit $j < s$, $i \geq 1$, $b \in \{+, -, 0, \dots, s\}$ und $l \leq s$. Die Anzahl s der Instruktionen von M nennt man auch die **Länge** von M . M hat die **Registerzahl** N und ist eine **N -Registermaschine**, wenn N mindestens so groß wie jede von M angesprochene Adresse ist:

$$N \geq \max \{i \mid \text{es gibt } j, b, l : j \ i \ b \ l \text{ ist Instruktion von } M\}.$$

Wegen besserer Lesbarkeit schreiben wir die Instruktionen einer Registermaschine meistens untereinander.

Beispiele

Wir untersuchen die Wirkung einiger besonders einfacher Registermaschinen.

- 1 a) $\begin{array}{cccc} 0 & 1 & - & 1 \end{array}$ Diese Registermaschine subtrahiert, soweit möglich, 1 vom Inhalt von R_1 , und bleibt dann stehen.
- 1 b) $\begin{array}{cccc} 0 & 1 & - & 0 \end{array}$ Diese Registermaschine löscht den Inhalt von R_1 , ändert dann nichts mehr, stoppt aber nicht.
- 1 c) $\begin{array}{cccc} 0 & 1 & 1 & 2 \\ 1 & 1 & - & 0 \end{array}$ Diese Registermaschine löscht den Inhalt von R_1 und stoppt dann.
- 1 d) $\begin{array}{cccc} 0 & 1 & 2 & 1 \\ 1 & 1 & - & 0 \end{array}$ Diese Registermaschine stoppt sofort, wenn der Inhalt von R_1 positiv ist. Ist der Inhalt von R_1 gleich 0, so ändert sie nichts, stoppt aber auch nicht.
- 1 e) Die leere Registermaschine \emptyset besteht aus 0 Instruktionen, ist also die Folge der Länge 0 und ist die einzige 0-Registermaschine. Sie startet gar nicht erst; sie stoppt schon, wenn sie starten sollte.

Bevor wir weitere Programme für Registermaschinen und die von ihnen bewirkten Operationen auf ihren Registern angeben, wollen wir präzisieren, wie eine Rechnung auf einer Registermaschine im einzelnen aussieht.

Gegeben sei eine N -Registermaschine M . Sie bearbeitet einen - unabhängig von M gegebenen - **Speicherinhalt**. Das ist ein N -Tupel oder N -Vektor von natürlichen Zahlen, nämlich eine natürliche Zahl in jedem der N Register. Ist \mathbf{x} der Speicherinhalt nach einigen Rechenschritten, so muss man wissen, nach welcher Instruktion I_j aus M der Inhalt \mathbf{x} bearbeitet werden soll. Sind Instruktionsnummer j **und** Inhalt \mathbf{x} bekannt, so liegt durch M nicht nur der nächste Rechenschritt, sondern auch die Nummer der Folgeinstruktion und damit - durch Wiederholung des Arguments - die ganze Rechnung fest.

1.1.2 Definition

Gegeben sei eine N -Registermaschine M der Länge s . Eine M -**Konfiguration** ist dann ein $(N + 1)$ -Tupel (j, \mathbf{x}) , wobei $j \leq s$ und \mathbf{x} ein N -Tupel x_1, \dots, x_N von natürlichen Zahlen ist. Hierin heißt j **Instruktionsnummer** und \mathbf{x} **Speicherinhalt** der Konfiguration. Ist $j = s$, so heißt (j, \mathbf{x}) auch **M -Endkonfiguration**. Eine M -Konfiguration (j', \mathbf{x}') heißt **M -Folgekonfiguration** der M -Konfiguration (j, \mathbf{x}) in folgenden Fällen:

1. $j < s$ und $I_j = j_i + l$:

- $j' = l$ und $\mathbf{x}' = (x_1, \dots, x_i + 1, \dots, x_N)$
2. $j < s$ und $I_j = j \ i \ - \ 1$:
 $j' = l$ und $\mathbf{x}' = (x_1, \dots, x_i \div 1, \dots, x_N)$
3. $j < s$ und $I_j = j \ i \ k \ l$:
 $j' = k$, falls $x_i > 0$, sonst $j' = l$, und stets $\mathbf{x}' = \mathbf{x}$
- E. $j = s$:
 $j' = s = j$ und $\mathbf{x}' = \mathbf{x}$

In 2. ist

$$x \div 1 = \begin{cases} x - 1, & \text{falls } x > 0 \text{ ist,} \\ 0, & \text{falls } x = 0 \text{ ist.} \end{cases}$$

Die Determiniertheit der Registermaschinen basiert auf folgender einfacher Bemerkung:

1.1.3 Lemma

Jede M -Konfiguration besitzt eine und nur eine M -Folgekonfiguration.

Der **Beweis** ergibt sich durch Fallunterscheidung entsprechend der Definition 1.1.2. Ist (j, \mathbf{x}) gegeben, so liegt - im Fall $j < s$ abhängig vom dritten Symbol in I_j - genau einer der Fälle 1. - 3. bzw. E. vor, und in jedem dieser Fälle ist genau ein (j', \mathbf{x}') als M -Folgekonfiguration in Definition 1.1.2 angegeben.

Bemerkung. Wichtig ist uns die Eindeutigkeitsaussage von Lemma 1.1.3. Die Existenz einer M -Folgekonfiguration ist im Fall E. per Definition (willkürlich) dadurch gesichert, dass eine Endkonfiguration ihre eigene Folgekonfiguration ist.

Die vorige Definition diene der Präzisierung des intuitiven Begriffs der Rechnung auf einer Registermaschine, die sich jetzt als endliche Konfigurationenfolge festlegen läßt.

1.1.4 Definition

Eine M -**Rechnung** der **Dauer** T auf einer Registermaschine M ist eine endliche Folge

$$((j_0, \mathbf{x}_0), \dots, (j_T, \mathbf{x}_T))$$

von insgesamt $T + 1$ M -Konfigurationen, so dass

1. $j_0 = 0$ ist und
2. für $t < T$ stets $(j_{t+1}, \mathbf{x}_{t+1})$ M -Folgekonfiguration von (j_t, \mathbf{x}_t) ist.

\mathbf{x}_0 heißt dabei die **Eingabe** und $(0, \mathbf{x}_0)$ die **Anfangskonfiguration** der Rechnung. Die Rechnung ist **beendet** oder **abgeschlossen**, wenn $j_T = s$, also (j_T, \mathbf{x}_T) eine M -Endkonfiguration ist. Dann heißt \mathbf{x}_T auch das **Ergebnis** der M -Rechnung zur Eingabe \mathbf{x}_0 , und dafür schreibt man oft

$$M : \mathbf{x}_0 \Longrightarrow \mathbf{x}_T \quad \text{oder} \quad \begin{array}{c} \mathbf{x}_0 \\ \Downarrow M \\ \mathbf{x}_T \end{array} .$$

Aus Lemma 1.1.3 ergibt sich durch Induktion nach der Rechenzeit die Determiniertheit der Ergebnisse:

1.1.5 Lemma

Das Ergebnis einer M -Rechnung ist, sofern es existiert, durch die Eingabe bereits eindeutig bestimmt:

Zu jedem Speicherinhalt \mathbf{x} gibt es höchstens ein \mathbf{y} mit $M : \mathbf{x} \Rightarrow \mathbf{y}$.

Beweis. \mathbf{x}_0 sei die Eingabe in M . Durch Induktion nach T folgt, dass es auf M genau eine Rechnung der Dauer T mit der Anfangskonfiguration $(0, \mathbf{x}_0)$ gibt. Denn

1. für $T = 0$ ist die Rechnung $((0, \mathbf{x}_0))$ der Dauer 0 vorgegeben.
2. Wenn es nach Induktionsvoraussetzung genau eine Rechnung $((0, \mathbf{x}_0), \dots, (j_T, \mathbf{x}_T))$ der Dauer T auf M gibt, so gibt es zur M -Konfiguration (j_T, \mathbf{x}_T) nach Lemma 1.1.3 genau eine M -Folgekonfiguration $(j_{T+1}, \mathbf{x}_{T+1})$. Dann ist

$$((0, \mathbf{x}_0), \dots, (j_T, \mathbf{x}_T), (j_{T+1}, \mathbf{x}_{T+1}))$$

die eindeutig bestimmte Rechnung der Dauer $T + 1$ auf M .

Ist nun $j_T = s$ für ein T , so ist $(j_{T+1}, \mathbf{x}_{T+1}) = (j_T, \mathbf{x}_T)$ nach Definition 1.1.2, E. Wenn also einmal in einem Zeitpunkt T im Laufe einer Rechnung eine M -Endkonfiguration erreicht wird, stimmt diese mit allen späteren M -Konfigurationen überein. Der Speicherinhalt \mathbf{x}_T zu diesem Zeitpunkt T ist dann das eindeutig bestimmte Ergebnis der Rechnung.

Eine Rechnung braucht auch bei beliebiger Fortsetzung keine Endkonfiguration zu erreichen, sie braucht kein Ergebnis zu haben. Dies zeigt sich schon in den Beispielen 1 b) und 1 d) und ist in 1.1.5 berücksichtigt.

Abkürzend teilen wir Speicherinhalte auch in der Form $\mathbf{x}x_i\mathbf{y}$ bzw. $\dots x_i \dots$ oder $\mathbf{x}x_i\mathbf{y}x_k\mathbf{z}$ bzw. $\dots x_i \dots x_k \dots$ mit, wenn uns nur der Inhalt des i -ten bzw. des i -ten und k -ten Registers interessiert.

Beispiele. Wir suchen Kopiermaschinen, die den Inhalt des i -ten Registers ins k -te Register übertragen. Dies kann man in verschiedener Weise auffassen.

2 a) Für $1 \leq i, k$ sei K_{ik} die Registermaschine mit dem Programm

$$\begin{array}{l} 0 \quad i \quad 1 \quad 3 \\ 1 \quad i \quad - \quad 2 \\ 2 \quad k \quad + \quad 0. \end{array}$$

Ist $i \neq k$, so liefert K_{ik} für jede Eingabe ein Ergebnis, und zwar ist

$$K_{ik} : \dots x_i \dots x_k \dots \Rightarrow \dots 0 \dots x_k + x_i \dots, \text{ speziell } \dots x_i \dots 0 \dots \Rightarrow \dots 0 \dots x_i \dots .$$

2 b) Will man den Inhalt des i -ten Registers ins k -te Register kopieren, unabhängig davon, was vorher im k -ten Register stand, so wird man erst das k -te Register löschen und dann K_{ik} anwenden, etwa mit folgendem Programm:

$$\left. \begin{array}{l} 0 \quad k \quad 1 \quad 2 \\ 1 \quad k \quad - \quad 0 \end{array} \right\} : \dots x_i \dots x_k \dots \Rightarrow \dots x_i \dots 0 \dots$$

$$\left. \begin{array}{l} 2 \quad i \quad 3 \quad 5 \\ 3 \quad i \quad - \quad 4 \\ 4 \quad k \quad + \quad 2 \end{array} \right\} : \dots x_i \dots 0 \dots \Rightarrow \dots 0 \dots x_i \dots$$

2 c) Die Maschinen unter 2 a) und 2 b) löschen beide das i -te Register (für $i \neq k$). Will man das nicht, so kopiert man den Inhalt des i -ten Registers zweimal in die Register k und l ; anschließend kopiert man mit K_{li} wieder von l nach i zurück. Wir definieren also K_{ikl} durch

$$\begin{array}{l} 0 \quad i \quad 1 \quad 4 \\ 1 \quad i \quad - \quad 2 \\ 2 \quad k \quad + \quad 3 \\ 3 \quad l \quad + \quad 0. \end{array}$$

Dies hat die Wirkung, falls i, k, l paarweise verschieden sind,

$$K_{ikl} : \dots x_i \dots x_k \dots x_l \dots \Rightarrow \dots 0 \dots x_k + x_i \dots x_l + x_i \dots$$

Danach rechnen wir mit K_{li} weiter, d. h. wir setzen K_{ikl} fort durch

$$4 + K_{li} : \begin{array}{r} 4 \ l \ 5 \ 7 \\ 5 \ l \ - \ 6 \\ 6 \ i \ + \ 4 \end{array}$$

und erhalten insgesamt

$$\dots x_i \dots x_k \dots x_l \dots \Rightarrow \dots x_l + x_i \dots x_k + x_i \dots 0 \dots$$

Für $x_k = x_l = 0$ ist dies das Gewünschte. Sonst muss man wie unter 2 b) die Löschmodulare für die Register k und l vorschalten.

An diesen einfachen Beispielen fällt zweierlei auf:

Bemerkung 1. Die Maschine unter 2 c) verwendet hilfsweise ein Register R_l , das sowohl vor als auch nach der Rechnung 0 enthalten soll, das aber unterwegs gebraucht wird, und, wie man sich intuitiv klarmacht, auch unentbehrlich ist. Zu einer Maschine $M : \mathbf{x}0 \Rightarrow \mathbf{y}0$ braucht es also keineswegs eine Maschine $M' : \mathbf{x} \Rightarrow \mathbf{y}$ zu geben. Dagegen folgt $M : \mathbf{x}\mathbf{z} \Rightarrow \mathbf{y}\mathbf{z}$ mit beliebigem N' -Tupel \mathbf{z} aus $M : \mathbf{x} \Rightarrow \mathbf{y}$. Hier taucht die Frage auf, zu welchen Zwecken man wieviele Register tatsächlich braucht.

Bemerkung 2. Unter 2 b) und 2 c) ist davon die Rede, dass man ein Programm an ein anderes "anhängt" bzw. einem anderen vorschaltet, allgemeiner dass man mehrere Maschinen "nacheinander rechnen" läßt. Es ist also von der bisher noch nicht definierten **Verkettung** von Registermaschinen die Rede und damit vom Operieren mit Registermaschinen. Jede Registermaschine bewirkt nach Definition 1.1.4 eine Operation auf den Speicherinhalten, aber hier treten Operationen auf, die Registermaschinen wieder Registermaschinen zuordnen, gewissermaßen Operationen höheren Typs. Einige spezielle von diesen Operationen werden wir jetzt untersuchen.

1.1.6 Definition

Ist M' eine Registermaschine und $s \in \mathbb{N}$, so sei $s + M'$ die Folge von Instruktionen, die man aus M' erhält, indem man alle Instruktionsnummern um s vergrößert, indem man also jede Instruktion

$$\begin{array}{lll} j \ i + l & \text{durch} & (s + j) \ i \quad + \quad (s + l) \\ j \ i - l & \text{durch} & (s + j) \ i \quad - \quad (s + l) \\ j \ i \ k \ l & \text{durch} & (s + j) \ i \ (s + k) \ (s + l) \end{array}$$

ersetzt.

Ist nun M eine Registermaschine der Länge s , so heißt die Instruktionenfolge $\begin{matrix} M \\ s + M' \end{matrix}$ die **Verkettung** von M und M' , die mit $(M M')$ bezeichnet wird.

Ferner wird die **k -te Potenz** einer Registermaschine M rekursiv definiert durch

$$M^0 \equiv \emptyset \text{ und } M^{k+1} \equiv (M^k M).$$

Für $s > 0$ und $M' \neq \emptyset$ ist $s + M'$ keine Registermaschine; wohl aber gilt:

1.1.7 Lemma

Die Verkettung (MM') von zwei Registermaschinen M und M' ist stets wieder eine Registermaschine. Genauer:

Sind M und M' N -Registermaschinen der Längen s bzw. s' , so ist (MM') eine N -Registermaschine der Länge $s + s'$. Es ist stets

$$(M\emptyset) \equiv (\emptyset M) \equiv M \text{ und } ((M M')M'') \equiv (M(M' M'')).$$

Beweis. In der ersten Spalte von $(MM') \equiv \begin{matrix} M \\ s + M' \end{matrix}$ treten nacheinander die Zahlen $0, \dots, s - 1, s + 0 = s, \dots, s + (s' - 1)$ als Instruktionsnummern auf. Ebenso treten in der dritten und vierten Spalte nur Zahlen $\leq s$ (in M) bzw. $\leq s + s'$ (in $s + M'$) auf.

Damit ist (MM') eine Registermaschine der Länge $s + s'$. In der zweiten Spalte schließlich werden nur Adressen $\leq N$ aufgerufen.

Per Definition ist $(M \emptyset) \equiv (\emptyset M) \equiv M$, weil \emptyset die Länge 0 hat. Schließlich ist

$$((M M')M'') \equiv \begin{matrix} M \\ s + M' \\ (s + s') + M'' \end{matrix} \equiv \begin{matrix} M \\ s + M' \\ s + (s' + M'') \end{matrix} \equiv (M(M' M'')),$$

weil es auf dasselbe hinauskommt, ob man die Instruktionsnummern von M'' um $s + s'$ erhöht oder ob man sie erst nur um s' und dann nochmals um s erhöht.

Bemerkung. Nach Lemma 1.1.7 bilden die Registermaschinen bezüglich

der Verkettung eine Halbgruppe mit neutralem Element \emptyset . Wegen der Assoziativität der Verkettung können wir die runden Klammern fortlassen. Wir schreiben also MM' für $(M M')$. Die Gleichheitsrelation \equiv zwischen Registermaschinen bezeichnet hier die buchstäbliche Übereinstimmung (syntaktische Identität) ihrer Programme. Das ist eine engere Relation als die Übereinstimmung der Wirkungen von Registermaschinen, ihre sog. extensionale Gleichheit, für die zwei Registermaschinen bei gleichen Eingaben gleiche Ergebnisse liefern müssen. \equiv ist echt enger als die extensionale Gleichheit, weil man in jedes Programm $\neq \emptyset$ Instruktionen einbauen kann, die nie aufgerufen werden. Z. B. sind

$$0\ 1 + 1 \text{ und } \begin{array}{l} 0\ 1 + 2 \\ 1\ 1\ 1\ 1 \end{array}$$

zwei extensional gleiche, aber nicht identische Registermaschinen. Lemma 1.1.7 ist formal in dem Sinne, dass es sich nur auf die Gestalt der Programme bezieht und von dem zentralen Begriff der Rechnung auf einer Registermaschine unabhängig ist. Nun war die Verkettung von Registermaschinen gerade so eingeführt, dass die Rechnungen auf MM' durch Hintereinanderschalten von Rechnungen auf M und Rechnungen auf M' entstehen:

1.1.8 Lemma

M, M' seien N -Registermaschinen. Dann ist $MM' : \mathbf{x} \Rightarrow \mathbf{z}$ gleichwertig damit, dass es ein N -Tupel \mathbf{y} gibt mit

$$M : \mathbf{x} \Rightarrow \mathbf{y} \text{ und } M' : \mathbf{y} \Rightarrow \mathbf{z}.$$

Die Maschine M' kommt also erst zum Zuge, wenn die M -Rechnung mit Eingabe \mathbf{x} abgeschlossen ist.

Eine kompakte und suggestive Notation hierfür ist auch

$$\mathbf{x} \xRightarrow{M} \mathbf{y} \xRightarrow{M'} \mathbf{z}.$$

Beweis. Ist M oder M' leer, so ist nichts zu beweisen. Seien M und M' nicht leer. Aus

$$M : \mathbf{x} \Rightarrow \mathbf{y} \text{ und } M' : \mathbf{y} \Rightarrow \mathbf{z}$$

folgt, dass es eine M -Rechnung $((0, \mathbf{x}), \dots, (s, \mathbf{y}))$ und eine M' -Rechnung $((0, \mathbf{y}), \dots, (s', \mathbf{z}))$ gibt, wobei die Instruktionsnummern s bzw. s' nur in der letzten Konfiguration der M - bzw. M' -Rechnung auftreten. Dann ist zunächst $((0, \mathbf{x}), \dots, (s, \mathbf{y}))$ eine MM' -Rechnung ohne Ergebnis und ferner

$$(0, \mathbf{x}), \dots, (s, \mathbf{y}), \dots, (s + s', \mathbf{z})$$

eine MM' -Rechnung mit Ergebnis \mathbf{z} , weil die Instruktionen aus $s + M'$ keine Instruktionsnummern $< s$ enthalten, also keine Instruktion aus M als Folgeinstruktion haben können.

Gilt umgekehrt $MM' : \mathbf{x} \Rightarrow \mathbf{z}$, so gibt es eine MM' -Rechnung

$$((0, \mathbf{x}), \dots, (s + s', \mathbf{z})).$$

Da M, M' nicht leer sind, ist $0 < s < s + s'$. Also muss es in dieser Rechnung eine erste Konfiguration (j, \mathbf{y}_1) mit Folgekonfiguration (j', \mathbf{y}_2) geben, so dass $j < s \leq j'$ ist. Dann ist $j' = s$, weil in der Instruktion I_j aus M keine größeren Instruktionsnummern als s auftreten.

Also gilt $M : \mathbf{x} \Rightarrow \mathbf{y}$ für $\mathbf{y} = \mathbf{y}_2$ und $M' : \mathbf{y} \Rightarrow \mathbf{z}$, weil in der gegebenen MM' -Rechnung nach $(j', \mathbf{y}_2) = (s, \mathbf{y})$ nur noch Instruktionen aus $s + M'$ herangezogen werden.

Warnung. Die angeführten M, M' - und MM' -**Rechnungen** haben in der Regel **nicht** die Dauer s, s' bzw. $s + s'$. s und s' sind nur die Längen der **Programme** M und M' . Wenn etwa die M -Rechnung die Dauer T hat und für $t \leq T$ die t -te Konfiguration der M -Rechnung die Instruktionsnummer j_t hat, so wird nur ausgenutzt, dass $j_T = s$ und $j_t < s$ ist für $t < T$.

Die Multiplikation wird bereits auf der Grundschule als Iteration der Addition eingeführt:

$$x \cdot y = \underbrace{x + x + \dots + x}_y \text{ mal}$$

Um $x \cdot y$ auf einer Registermaschine zu berechnen, prüft man deshalb zuerst, ob $y > 0$ ist; ist dies der Fall, so subtrahiert man 1 von y , addiert x zum Register, in dem der Wert erscheinen soll, und beginnt von vorn; ist $y = 0$, so hört man auf. Man iteriert also den ersten Teil des Programms so lange, bis $y = 0$ ist.

Setzen wir $s_3 := 0 \ 3 - 1$, so ist zunächst $s_3 K_{214} K_{42}$ eine 4-Registermaschine mit der Wirkung

$$z \ x \ y \ 0 \Rightarrow z + x \ x \ y \div 1 \ 0$$

und dem Programm

M

$$\left. \begin{array}{l} 0\ 3\ -\ 1 \\ 1\ 2\ 2\ 5 \\ 2\ 2\ -\ 3 \\ 3\ 1\ +\ 4 \\ 4\ 4\ +\ 1 \\ 5\ 4\ 6\ 8 \\ 6\ 4\ -\ 7 \\ 7\ 2\ +\ 5 \end{array} \right\} \begin{array}{l} s_3 \\ 1 + K_{214} \\ 5 + K_{42} \end{array}$$

Das neue Programm ist dann

$(M)_3$

$$\left. \begin{array}{l} 0\ 3\ 1\ 9 \\ 1\ 3\ -\ 2 \\ 2\ 2\ 3\ 6 \\ 3\ 2\ -\ 4 \\ 4\ 1\ +\ 5 \\ 5\ 4\ +\ 2 \\ 6\ 4\ 7\ 0 \\ 7\ 4\ -\ 8 \\ 8\ 2\ +\ 6 \end{array} \right\} \begin{array}{l} 1 + s_3 \\ 2 + K_{214} \\ 6 + K_{42}(\text{mod } 9) \end{array}$$

und bewirkt

$$(M)_3 : 0\ x\ y\ 0 \Rightarrow x \cdot y\ x\ 0\ 0.$$

Zur Konstruktion von $(M)_3$ verkettet man also erst s_3 mit K_{214} und K_{42} und iteriert diese neue Registermaschine $M := s_3 K_{214} K_{42}$ nach dem 3. Register. Das erreicht man dadurch, dass man eine Testinstruktion $0\ 3\ 1\ 9$ vorschaltet, dann die acht Instruktionen von $1 + M$ folgen läßt, dabei aber in $1 + M$ jedes Auftreten der (Stop-)Instruktionsnummer 9 durch 0 ersetzt, wodurch der Eingangstest $0\ 3\ 1\ 9$ wieder aufgerufen wird, wenn eine Rechnung aus $1 + M$ herausführt. Zum Stop kommt $(M)_3$ also nur, wenn dieser Test ein leeres drittes Register vorfindet. Diesen Prozess definieren wir allgemein.

1.1.9 Definition

Gegeben sei eine Registermaschine M der Länge s und eine Adresse i . Ersetzt man in $1 + M$ die Instruktionsnummer $s + 1$ überall durch 0, so erhält man eine Instruktionenfolge, die wir mit $1 + M(\text{mod } s + 1)$ bezeichnen. Die **Iteration von M nach dem i -ten Register** ist dann

$$(M)_i := \begin{array}{l} 0\ i\ 1\ (s + 1) \\ 1 + M(\text{mod } s + 1). \end{array}$$

1.1.10 Lemma

Die Iteration einer Registermaschine nach einem Register ist stets wieder eine Registermaschine. Genauer: Ist M eine N -Registermaschine der Länge s , so ist $(M)_i$ eine $\max(N, i)$ -Registermaschine der Länge $s + 1$.

Dieses Lemma entspricht Lemma 1.1.7 und wird entsprechend bewiesen. Für die Wirkungsweise der Iteration gilt:

1.1.11 Lemma

M sei eine N -Registermaschine $\neq \emptyset$; o. B. d. A. sei $i \leq N$. Dann ist

$$(M)_i : \mathbf{x} \Rightarrow \mathbf{y}$$

gleichwertig damit, dass zugleich

1. $y_i = 0$ ist und
2. $M^k : \mathbf{x} \Rightarrow \mathbf{y}$ für ein k gilt, für das aus $l < k$ stets folgt:
 M^l hat, angesetzt auf \mathbf{x} , ein Ergebnis \mathbf{z} mit $z_i \neq 0$.

Beweis. M sei nicht leer. Gelten 1. und 2. und definieren wir induktiv

$$\mathbf{y}^0 = \mathbf{x} \text{ und für } l < k :$$

$$\mathbf{y}^{l+1} \text{ ist das Ergebnis von } M, \text{ angesetzt auf } \mathbf{y}^l,$$

so haben wir

$$\mathbf{x} = \mathbf{y}^0 \xRightarrow{M} \mathbf{y}^1 \xRightarrow{M} \dots \xRightarrow{M} \mathbf{y}^k = \mathbf{y}.$$

Durch Induktion nach k sieht man, dass $(M)_i$ bei \mathbf{y}^k stoppt, aber nicht vorher, weil für $l < k$ der Eingangstest 0 i 1 $s + 1$, angesetzt auf \mathbf{y}^l , wegen 2. ein positives Ergebnis hat, so dass der Programmteil 1 + M aufgerufen wird. Dieser liefert das Ergebnis \mathbf{y}^{l+1} , das nun wieder dem Eingangstest unterworfen wird.

Umgekehrt gelte $(M)_i : \mathbf{x} \Rightarrow \mathbf{y}$. Es gibt also eine $(M)_i$ -Rechnung $((0, \mathbf{x}), \dots, (s + 1, \mathbf{y}))$ mit Eingabe \mathbf{x} , Ergebnis \mathbf{y} und Dauer T (also $j_T = s + 1$). Wir zeigen 1. und 2. durch Induktion nach T . Offenbar ist $T > 0$.

Ist $T = 1$, so ist $\mathbf{x} = \mathbf{y}$, der Eingangstest ergibt 0 im i -ten Register, es ist also $y_i = x_i = 0$, und 2. gilt für $k = 0$.

Ist $T > 1$, so ist $x_i > 0$, M kommt zum Einsatz und liefert ein Ergebnis \mathbf{y}^1 , auf das $(M)_i$ wieder angesetzt wird. Die vorausgesetzte $(M)_i$ -Rechnung zerfällt also in zwei Teile wie folgt:

$$\mathbf{x} \xRightarrow{\text{Test}} \mathbf{x} \xRightarrow{M} \mathbf{y}^1 \xRightarrow{(M)_i} \mathbf{y}$$

Die Dauer T' der $(M)_i$ -Rechnung, angesetzt auf \mathbf{y}^1 , ist kürzer als T . Nach Induktionsvoraussetzung gilt

1'. $y_i = 0$ und

2'. es gibt ein k' mit $M^{k'} : \mathbf{y}^1 \Rightarrow \mathbf{y}$ und aus $l < k'$ folgt stets $M^l : \mathbf{y}^1 \Rightarrow \mathbf{z}$ für ein \mathbf{z} mit $z_i \neq 0$.

1'. ist 1., und setzt man $k := k' + 1$, so folgt 2.

Warnung. Die M -Rechnungen mit den Eingaben \mathbf{y}^l , wie sie oben betrachtet wurden, sind für verschiedene l in der Regel **nicht** von gleicher Dauer, z. B. kann die M -Rechnung $M : \mathbf{x} \Rightarrow \mathbf{y}^1$ allein länger dauern als $M^{k-1} : \mathbf{y}^1 \Rightarrow \mathbf{y}^k$ für große k .

Die Operationen der Verkettung und der Iteration von Registermaschinen geben uns die Möglichkeit, eine übersichtliche Klasse von Registermaschinen auszuzeichnen, deren Programme wir nicht mehr in einer Matrix hinzuschreiben brauchen, sondern linear ohne explizite Angabe der Instruktionsnummern. Das erleichtert den praktischen Umgang mit diesen speziellen Registermaschinen erheblich.

1.1.12 Induktive Definition

der normierten Registermaschinen.

1. Jede Maschine

$$a_i \equiv 0 \ i + 1 \quad \text{und} \quad s_i \equiv 0 \ i - 1$$

ist eine normierte Registermaschine.

2. Sind M_1 und M_2 normierte Registermaschinen, so ist auch $(M_1 M_2)$ eine normierte Registermaschine.

3. Ist M eine normierte Registermaschine und $i \geq 1$, so ist $(M)_i$ eine normierte Registermaschine.

Beispiele. Die in Beispiel 2 eingeführten Lösch- und Kopiermaschinen sind so, wie sie dort aufgeschrieben wurden, normierte Registermaschinen. Denn es ist

$$\begin{aligned} L_i &\equiv (s_i)_i \\ K_{ik} &\equiv (s_i a_k)_i \quad \text{und} \\ K_{ikl} &\equiv (s_i a_k a_l)_i. \end{aligned}$$

1.1.13 Lemma

L_i, K_{ik}, K_{ikl} sind normierte Registermaschinen.

Lemma 1.1.7 und 1.1.10 ergeben sofort:

1.1.14 Satz

Jede normierte Registermaschine ist eine Registermaschine.

Der Beweis durch Induktion nach der Definition der normierten Registermaschinen ist trivial.

Die Umkehrung gilt natürlich nicht: Bei normierten Registermaschinen kann eine Testinstruktion $j \ i \ k \ l$ nur am Anfang eines Iterationsprogramms auftreten, und dann ist $k = j + 1$. Dies kann bei beliebigen Registermaschinen verletzt sein. Bei normierten Registermaschinen kann man auch nicht aus dem Inneren eines Iterationsteils herausspringen, was bei anderen Registermaschinen selbstverständlich möglich ist. Normierte Registermaschinen werden im großen und ganzen von oben nach unten Instruktion für Instruktion abgearbeitet; nur beim Beginn eines Iterationsteils wird getestet, ob dieser Teil en bloc übersprungen werden soll, und am Ende des Iterationsteils springt man zu seinem Beginn, dem Eingangstest, zurück. Trotzdem werden wir feststellen, dass die normierten Registermaschinen dasselbe leisten wie die Registermaschinen insgesamt, wenn auch nicht immer mit derselben Registerzahl.

1.1.15 Aufgaben

1. Man definiere eine Addiermaschine Add mit 4 Registern mit der Wirkung

$$Add : 0 \ x \ y \ 0 \Rightarrow x + y \ x \ y \ 0.$$

Das 4. Register wird also nur hilfsweise gebraucht (vgl. Bem. 1). Man kann Add leicht aus den Kopierprogrammen aus Beispiel 2 a) bis 2 c) zusammensetzen.

2. Was leistet K_{ikk} für $i \neq k$, was für $i = k$?
3. Man zeige $M^{k+l} \equiv M^k M^l$.
4. Die arithmetische Differenz \div ist die 2-stellige Funktion

$$\div : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

mit

$$x \div y = \begin{cases} x - y, & \text{falls } y \leq x \text{ ist;} \\ 0, & \text{falls } y > x \text{ ist.} \end{cases}$$

Zu $1 \leq i, k, l, i \neq k \neq l \neq i$ definiere man einen $+/-$ -Kopierer M mit der Wirkung

$$M : \dots x_i \dots x_k \dots x_l \dots \Rightarrow \dots 0 \dots x_k + x_i \dots x_l \div x_i \dots$$

Mit diesem M und einigen Kopiermaschinen definiere man durch Verkettung eine Subtrahiermaschine Sub mit 4 Registern und der Wirkung

$$Sub : 0 x y 0 \Rightarrow x \dot{-} y x y 0.$$

5. Man wandle die Registermaschine $(s_3K_{214}K_{42})_3$, die vor 1.1.9 eingeführt wurde, ab zu einer 5-Registermaschine $Mult$ mit der Wirkung

$$Mult : 0 x y 0 0 \Rightarrow x \cdot y x y 0 0 .$$

6. Ausgehend von der Registermaschine $Mult$ konstruiere man eine Potenzier-Maschine Pot als 6-Registermaschine mit

$$Pot : 0 x y 0 0 0 \Rightarrow x^y x y 0 0 0 .$$

Dann schätze man die Dauer dieser Rechnungen in Abhängigkeit von x, y ab.

1.2 Partielle Funktionen, Berechenbarkeit und μ -Rekursivität

Wir haben in 1.1 den Begriff der Rechnung präzisiert, nicht aber den der berechenbaren Funktion. Dass eine n -stellige Funktion f durch eine Registermaschine M berechnet wird, soll natürlich heißen, dass M , angesetzt auf ein Argument \mathbf{x} von f , ein Ergebnis hat und $f(\mathbf{x})$ liefert. Da unsere Registermaschinen nur natürliche Zahlen speichern, muss das Argument \mathbf{x} jedenfalls ein n -Tupel natürlicher Zahlen und $f(\mathbf{x})$ eine natürliche Zahl sein, d. h. f muss eine zahlentheoretische Funktion sein. Da eine Registermaschine M nicht für jedes n -Tupel \mathbf{x} ein Ergebnis zu liefern braucht, liegt es nahe, nicht nur totale, überall auf \mathbb{N}^n definierte Funktionen zu betrachten, sondern auch partielle Funktionen, die auf Teilmengen von \mathbb{N}^n definiert sind.

1.2.1 Definition

Eine Funktion $f : D \rightarrow \mathbb{N}$ mit $D \subseteq \mathbb{N}^n$ heißt eine n -stellige **partielle** (zahlentheoretische) **Funktion**. Ist $D = \mathbb{N}^n$, so heißt f auch **total**. Den **Definitionsbereich** D von f bezeichnen wir mit $dom f$.

Partielle (reelle) Funktionen sind aus der Analysis geläufig: z. B. das Inverse $\frac{1}{ia}$ und der Tangens \tan sind reelle Funktionen, die auf echten Teilmengen von \mathbb{R} definiert sind.

Oft ist es nützlich, aus dem Ergebnis einer Rechnung die Eingabe wieder ablesen zu können. Deshalb definieren wir die Berechnung einer Funktion so, dass im Ergebnis Funktionswert und Argumente nebeneinander stehen.

1.2.2 Definition

Eine Registermaschine M **berechnet** die partielle n -stellige Funktion

$f : D \rightarrow \mathbb{N}$ ($D \subseteq \mathbb{N}^n$), wenn gilt:

$$\begin{array}{ll} \text{für } \mathbf{x} \in D & M : 0\mathbf{x}0 \dots 0 \Rightarrow f(\mathbf{x})\mathbf{x}0 \dots 0 \quad \text{und} \\ \text{für } \mathbf{x} \notin D & M : 0\mathbf{x}0 \dots 0 \quad \text{stoppt nicht.} \end{array}$$

Genau für die $(n + 1)$ -Tupel $0\mathbf{x}$ mit $\mathbf{x} \in D$ (ergänzt durch Nullen zu einem N -Tupel, wenn M eine N -Registermaschine ist) hat also M ein Ergebnis, und dieses ist $f(\mathbf{x})\mathbf{x}$ (wieder ergänzt durch Nullen). Die f berechnende Registermaschine M kann also mehr als $n + 1$ Register haben. Die zusätzlichen Register stehen dann für die Rechnungen auf M zur Verfügung, müssen aber beim Abschluß der Rechnung wieder gelöscht sein. In einzelnen Fällen kann M aber auch weniger als $n + 1$ Register haben, wenn nämlich f von seinen letzten Argumenten nicht abhängt und deshalb die letzten Adressen im Programm M nicht auftreten. Dieser Fall kommt in Lemma 1.2.5 vor. Auch dann gilt noch $M : 0\mathbf{x} \Rightarrow f(\mathbf{x})\mathbf{x}$, weil M nach Definition 1.1.1 auch eine $(n + 1)$ -Registermaschine ist.

1.2.3 Definition

Eine Funktion f heißt **partiell (Register-) berechenbar**, wenn f partiell ist und es eine Registermaschine gibt, die f berechnet. f heißt **partiell normiert berechenbar**, wenn f partiell ist und es eine normierte Registermaschine gibt, die f berechnet. Ist f hierbei total, so heißt f **berechenbar** bzw. **normiert berechenbar**.

Eine n -stellige berechenbare Funktion ist demnach eine totale Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$, die von einer Registermaschine berechnet wird. Nach 1.1.15, 1., 4. und 5. sind z. B. die Addition $+$, die arithmetische Differenz \div und die Multiplikation \cdot berechenbar. Wir führen einige noch einfachere berechenbare Funktionen ein.

1.2.4 Definition

Als **Nachfolgerfunktion** suc (vom englischen “successor”) bezeichnet man die Funktion

$$suc : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto x + 1.$$

Als **Projektions-** oder **Identitätsfunktionen** bezeichnet man alle Funktionen

$$U_i^n : \mathbb{N}^n \rightarrow \mathbb{N}, x_1, \dots, x_n \mapsto x_i \quad \text{mit } n \geq 1, 1 \leq i \leq n.$$

Als **konstante Funktionen** bezeichnet man alle Funktionen

$$C_k^n : \mathbb{N}^n \rightarrow \mathbb{N}, \mathbf{x} \mapsto k \quad \text{mit } n, k \in \mathbb{N}.$$

1.2.5 Lemma

Alle Funktionen suc , U_i^n und C_k^n sind normiert berechenbar.

Beweis. Eine 3-Registermaschine für suc ist z. B. $K_{213}K_{32}a_1$; eine $(n + 2)$ -Registermaschine für U_i^n ist

$$K_{i+11n+2}K_{n+2} \ i+1,$$

und eine 1-Registermaschine für C_k^n ist $(a_1)^k$. (Für C_0^n kann man etwa s_1 wählen.) Mit Lemma 1.1.13 folgt die Behauptung.

Diese Funktionen sind für sich noch nicht interessant, aber mit $+$ und \cdot kann man aus ihnen schon alle Polynome (in endlich vielen Variablen) zusammensetzen; z. B. erhält man

$$f : x, y \mapsto x \cdot y + 2y^2$$

so: Es sei

$$f_0 : x, y \mapsto x \cdot y \text{ die Multiplikation, ferner}$$

$$f_1 : x, y \mapsto y^2 = y \cdot y = U_2^2(x, y) \cdot U_2^2(x, y)$$

und in informeller Schreibweise

$$f_2 = C_2^2 \cdot f_1 = C_2^2 \cdot U_2^2 \cdot U_2^2.$$

Dann ist

$$f : x, y \mapsto x \cdot y + f_2(x, y),$$

also

$$f = f_0 + f_2 = f_0 + (C_2^2 \cdot U_2^2 \cdot U_2^2).$$

Das Zusammensetzen von Funktionen ist eine Operation auf Funktionen und entspricht der Verkettung von Registermaschinen. Wir wollen diese Operation für n -stellige partielle Funktionen definieren. Dabei stellt sich schon für einstellige g und h das Problem:

Für welche $x \in \mathbb{N}$ soll $h(g(x))$ als definiert gelten?

In der Berechenbarkeitstheorie folgt man dem Prinzip des **call by value** (auch **inside-first**-Prinzip genannt), nach dem zuerst der Wert von $g(x)$ bekannt sein muss, bevor man $h(g(x))$ ausrechnen kann – auch wenn h eine konstante Funktion ist und $h(g(x))$ deshalb von dem Wert von $g(x)$ gar nicht abhängt.

Für eine übersichtliche Schreibweise führen wir ein neues Zeichen $*$ als **Symbol der undefiniertheit** ein und setzen "abkürzend":

$$(1) \quad f(\mathbf{x}) = * \Leftrightarrow \mathbf{x} \notin \text{dom } f.$$

Insbesondere gehören Tupel, in denen $*$ auftritt, zu keinem Definitionsbereich, so dass stets gilt:

$$(2) \quad f(\mathbf{y}, *, \mathbf{z}) = *.$$

Diese Gleichung (2) kann man als kurze Fassung des **call-by-value**-Prinzips lesen.

Beispiele.

- 1 a) $\text{suc}^{-1}(x)$ sei das y , dessen Nachfolger $\text{suc } y = x$ ist. Dann ist $\text{suc}^{-1}(0) = *$ und auch

$$\text{suc}^{-1}(0) \cdot 0 = * \cdot 0 = *,$$

obwohl $x \cdot 0 = 0$ unabhängig von x gilt.

- 1 b) Betrachten wir die gewöhnliche Differenz $-$ als partielle Funktion mit $x - y = *$ für $x < y$, so ist zwar

$$\begin{aligned} \text{suc}(x - y) &= \text{suc } x - y && \text{für } x \geq y, \text{ aber} \\ \text{suc}(x - y) &= * \neq 0 = \text{suc } x - y && \text{für } y = x + 1. \end{aligned}$$

Die Ausdrücke $\text{suc}(x - y)$ und $\text{suc } x - y$ sind also nicht "stets" gleich, weil sie nicht "stets zugleich" definiert sind.

Wir definieren formal diese strenge Form der Gleichheit unter Einbeziehung des undefinierten Wertes $*$.

1.2.6 Definition der partiellen Gleichheit

Gegeben seien zwei Ausdrücke (Terme) s und t , die aus (Zeichen für) partielle Funktionen und höchstens aus den Variablen $x_1, \dots, x_n =: \mathbf{x}$ zusammengesetzt sind. s und t heißen **partiell gleich**, man schreibt

$$s \simeq t,$$

wenn für alle $\mathbf{x} \in \mathbb{N}^n$ die (in $\mathbb{N} \cup \{*\}$ liegenden) Werte von s und t übereinstimmen. Insbesondere muss dann $s = * \Leftrightarrow t = *$ sein.

Beispiele.

2 a) Es ist $x \cdot 0 \simeq 0$ trotz Beispiel 1 a).

2 b) Ebenso ist $U_i^n(\mathbf{x}) \simeq x_i$ und $C_k^n(\mathbf{x}) \simeq k$.

2 c) Wegen 1 b) gilt **nicht** $\text{suc}(x - y) \simeq \text{suc}x - y$.

Mit dem Begriff der partiellen Gleichheit kann man die Komposition partieller Funktionen knapp und übersichtlich definieren:

1.2.7 Definition

Sind g_1, \dots, g_r r n -stellige und ist h eine r -stellige partielle Funktion, so heißt die durch

$$f(\mathbf{x}) \simeq h(g_1(\mathbf{x}), \dots, g_r(\mathbf{x}))$$

definierte partielle Funktion f die **Komposition von h nach g_1, \dots, g_r** , bezeichnet mit $h \circ (g_1, \dots, g_r)$.

Man sagt auch: f entsteht aus h, g_1, \dots, g_r **durch simultane Einsetzung**.

Unter Berücksichtigung von (2) liegt hiermit auch der Definitionsbereich von f fest: Es ist

$$\mathbf{x} \in \text{dom } f \iff \mathbf{x} \in \text{dom } g_i \text{ für } i = 1, \dots, r \text{ und } (g_1(\mathbf{x}), \dots, g_r(\mathbf{x})) \in \text{dom } h.$$

Denn ist $g_i(\mathbf{x}) = *$ für ein i , so ist

$$f(\mathbf{x}) = h(\dots, *, \dots) = *$$

wegen (2). Ist andererseits $g_i(\mathbf{x}) \in \mathbb{N}$ für alle $i = 1, \dots, r$, so ist ohnehin

$$\mathbf{x} \in \text{dom } f \iff (g_1(\mathbf{x}), \dots, g_r(\mathbf{x})) \in \text{dom } h.$$

Bemerkung. Für $r = 1$ ist dies die übliche Komposition von Funktionen. Aber in jedem Fall $r \geq 0$ kann man $g = (g_1, \dots, g_r)$ als Funktion von $\bigcap \{dom\ g_i | 1 \leq i \leq r\}$ in \mathbb{N}^r auffassen gemäß

$$g(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_r(\mathbf{x})).$$

Dann ist $h \circ (g_1, \dots, g_r) = h \circ g$ wieder eine Komposition im üblichen Sinne. Dies und Lemma 1.1.8 machen die Analogie zwischen Verkettung von Registermaschinen und Komposition von Funktionen deutlich. Wegen der speziellen Art, in der Funktionen nach Definition 1.2.2 berechnet werden sollen, ist an folgendem Lemma aber doch noch etwas zu rechnen.

1.2.8 Lemma

Sind g_1, \dots, g_r n -stellige partiell normiert berechenbare Funktionen und ist h eine r -stellige partiell normiert berechenbare Funktion, so ist auch $f = h \circ (g_1, \dots, g_r)$ partiell normiert berechenbar:

Die Klasse der partiell normiert berechenbaren Funktionen ist abgeschlossen unter Komposition.

Folgerung: Alle Polynome sind normiert berechenbar.

Beweis. Nach Voraussetzung gibt es normierte N -Registermaschinen M_0, \dots, M_r , die h, g_1, \dots, g_r berechnen, d. h. für die $\mathbf{x} \in dom\ g_i$ gleichwertig ist mit $M_i : 0\mathbf{x}\bar{0} \Rightarrow g_i(\mathbf{x})\mathbf{x}\bar{0}$ für $1 \leq i \leq r$ und $\mathbf{x} \in dom\ h$ mit $M_0 : 0\mathbf{x}\bar{0} \Rightarrow h(\mathbf{x})\mathbf{x}\bar{0}$. Wir suchen dann normierte $(N + r + n)$ -Registermaschinen N_1 bis N_4 , die folgendes leisten:

$$\begin{aligned} N_1 : & \quad 0\mathbf{x}\bar{0}^r 0^n \Rightarrow 0\mathbf{x}\bar{0}g(\mathbf{x})0^n \\ N_2 : & \quad \Rightarrow 0g(\mathbf{x})\bar{0}^r \mathbf{x} \\ N_3 : & \quad \Rightarrow f(\mathbf{x})g(\mathbf{x})\bar{0}^r \mathbf{x} \\ N_4 : & \quad \Rightarrow f(\mathbf{x})\mathbf{x}\bar{0}^r 0^n. \end{aligned}$$

(0^k steht für $000\dots 0$. $\bar{0}$ steht für eine genügende Anzahl von Nullen.) Z. B. wird dies geleistet von

$$\begin{aligned} N_1 &= M_1 K_{1\ N+1} M_2 K_{1\ N+2} \dots M_r K_{1\ N+r}, \\ N_2 &= K_{2\ N+r+1} \dots K_{n+1\ N+r+n} K_{N+1\ 2} \dots K_{N+r\ r+1}, \\ N_3 &= M_0, \\ N_4 &= L_2 \dots L_{r+1} K_{N+r+1\ 2} \dots K_{N+r+n\ n+1}. \end{aligned}$$

$M = N_1 N_2 M_0 N_4$ berechnet dann $f = h \circ (g_1, \dots, g_r)$. M ist also eine Verkettung von M_0, M_1, \dots, M_r , etlichen Kopier- und einigen Löschmaschinen. M

liefert auch die richtige Definitionsmenge von f . Mit der Zahl der Register ist hier großzügig umgegangen worden; man kann sie im allgemeinen niedriger halten, was uns im Moment aber nicht interessiert.

Bisher ist die Iteration nur innerhalb der Kopier- und Löschmaschinen aufgetreten. Allein mit Kopiermaschinen ist auch die Addition normiert berechenbar. Denn eine Registermaschine

$$Add : 0xy0 \Rightarrow (x + y)xy0$$

ist zum Beispiel

$$Add = K_{214}K_{42}K_{314}K_{43}.$$

Für die Subtraktion (arithmetische Differenz) brauchen wir den $+/-$ -Kopierer

$$M_{ikl} : \dots x_i \dots x_k \dots x_l \dots \Rightarrow \dots 0 \dots x_k + x_i \dots x_l \div x_i \dots,$$

der wegen

$$M_{ikl} = (s_i a_k s_l)_i$$

auch eine normierte Registermaschine ist. Dann erhält man

$$Sub : 0xy0 \Rightarrow (x \div y)xy0$$

z. B. als

$$Sub = K_{214}K_{42}(s_3 a_4 s_1)_3 K_{43},$$

und das ist eine normierte Registermaschine.

Eine Registermaschine für die Multiplikation

$$Mult : 0xy00 \Rightarrow (x \cdot y) xy00$$

erhält man durch Iteration der Addition, z. B. als

$$Mult = K_{345}K_{43}(s_5 K_{214}K_{42})_5.$$

Die Registermaschine

$$(s_5 K_{214}K_{42})_5$$

addiert den Inhalt des 2. Registers so oft zum Inhalt des 1. Registers, wie im 5. Register angegeben ist, und löscht dabei das 5. Register. (Dies ist im wesentlichen die Arbeitsweise einer Handrechenmaschine mit 3 Registern, nämlich dem Addierwerk R_2 , dem Resultatwerk R_1 und dem Zählwerk R_5 , das die Umdrehungen des Addierwerkes zählt.) Diese Iteration nach R_5 erzeugt die

Multiplikation, weil diese durch folgende Rekursionsgleichungen mit Hilfe der Addition festgelegt ist:

$$x \cdot 0 = 0$$

$$x \cdot (y + 1) = x \cdot y + x.$$

Entsprechendes gilt für die arithmetische Differenz

$$x \dot{-} 0 = x$$

$$x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1$$

und für die Addition

$$x + 0 = x$$

$$x + (y + 1) = (x + y) + 1.$$

Hierdurch werden \cdot , $\dot{-}$ und $+$ auf die elementaren Operationen $+1$ und $\dot{-} 1$ zurückgeführt. Auch $\dot{-} 1$ läßt sich noch festlegen durch die Rekursionsgleichungen

$$0 \dot{-} 1 = 0$$

$$(y + 1) \dot{-} 1 = y.$$

Während zur Berechnung (Definition) von $x \cdot \text{sucy}$ auf $x \cdot y$, von $x + \text{sucy}$ auf $x + y$ und von $x \dot{-} \text{sucy}$ auf $x \dot{-} y$ zurückgegriffen wird, hängt $\text{sucy} \dot{-} 1$ von $y \dot{-} 1$ gar nicht ab, sondern nur von y .

Wir untersuchen jetzt, was die Iteration einer Registermaschine allgemein mit solchen Rekursionsgleichungen zu tun hat.

1.2.9 Definition

Gegeben seien eine n -stellige und eine $(n + 2)$ -stellige partielle Funktion $g : \text{dom } g \rightarrow \mathbb{N}$ und $h : \text{dom } h \rightarrow \mathbb{N}$. Die durch folgendes **Schema der primitiven Rekursion**

$$(PR) \quad \begin{aligned} f(\mathbf{x}, 0) &\simeq g(\mathbf{x}) \\ f(\mathbf{x}, y + 1) &\simeq h(\mathbf{x}, y, f(\mathbf{x}, y)) \end{aligned}$$

eindeutig festgelegte partielle $(n + 1)$ -stellige Funktion f heißt durch **primitive Rekursion aus g, h** definiert und wird mit Rgh bezeichnet.

Für die Definitionsmenge $\text{dom } f$ von f gilt wegen (2):

$$\begin{aligned} (\mathbf{x}, 0) \in \text{dom } f &\iff \mathbf{x} \in \text{dom } g \\ (\mathbf{x}, y + 1) \in \text{dom } f &\iff \mathbf{x}, y \in \text{dom } f \text{ und } (\mathbf{x}, y, f(\mathbf{x}, y)) \in \text{dom } h. \end{aligned}$$

Als Ausartungsfall der primitiven Rekursion wird oft die Definition durch **Fallunterscheidung** nach dem Schema

$$(PR') \quad \begin{aligned} f(\mathbf{x}, 0) &\simeq g(\mathbf{x}) \\ f(\mathbf{x}, y + 1) &\simeq h(\mathbf{x}, y) \end{aligned}$$

betrachtet. Das aus dem n -stelligen g und dem $(n + 1)$ -stelligen h durch Fallunterscheidung definierte $(n + 1)$ -stellige f wird mit $R'gh$ bezeichnet. Für $dom f$ gilt dann

$$\begin{aligned} (\mathbf{x}, 0) \in dom f &\iff \mathbf{x} \in dom g \\ (\mathbf{x}, y + 1) \in dom f &\iff (\mathbf{x}, y) \in dom h. \end{aligned}$$

Bemerkung 1. Ist $f = Rgh$ durch (PR) definiert und ist $(\mathbf{x}, y) \notin dom f$, so ist $(\mathbf{x}, z) \notin dom f$ für alle $z \geq y$. Dies gilt nicht für (PR') , weswegen wir (PR') überhaupt eingeführt haben.

Bemerkung 2. Ist $f = Rgh$ und sind g, h total, so ist auch f total. Das Umgekehrte gilt nicht: Sind f und g total, so braucht h im letzten Argument nicht total zu sein. Das ist bei (PR') einfacher, denn hier gilt

$$g \text{ und } h \text{ total} \iff f \text{ total} .$$

In diesem Fall, wenn g und h in (PR') total sind, kann man (PR') mit Komposition und Projektionsfunktionen auf (PR) zurückführen. Denn für

$$h^+ = h \circ (U_1^{n+2}, \dots, U_{n+1}^{n+2})$$

ist $f = R'gh = Rgh^+$. Wegen Bemerkung 1 ist die Lösung für beliebige partielle g und h komplizierter (vgl. 1.3.5).

Beispiele. Es ist

3 a)

$$\begin{aligned} x + 0 &= x = U_1^1(x) \\ x + (y + 1) &= suc(x + y) = suc \circ U_3^3(x, y, x + y), \\ \text{also } + &= R U_1^1(suc \circ U_3^3). \text{ Entsprechend ist} \end{aligned}$$

3 b)

$$\begin{aligned} x \dot{-} 0 &= x = U_1^1(x) \\ x \dot{-} (y + 1) &= (x \dot{-} y) \dot{-} 1 = \dot{-}1 \circ U_3^3(x, y, x \dot{-} y), \\ \text{also } \dot{-} &= R U_1^1(\dot{-}1 \circ U_3^3). \text{ Entsprechend ist} \end{aligned}$$

3 c)

$$x \cdot 0 = 0 = C_0^1(x)$$

$$x \cdot (y + 1) = x \cdot y + x = + \circ (U_3^3, U_1^3)(x, y, x \cdot y),$$

$$\text{also } \cdot = R C_0^1(+ \circ (U_3^3, U_1^3)).$$

Einfachste Beispiele totaler Funktionen, die durch Fallunterscheidung (PR') gegeben sind, aber auch sofort mit (PR) definiert werden können, sind die Vorzeichen- (*signum*-)Funktionen sg und \overline{sg} sowie der Vorgänger $\div 1$:

4 a)

$$sg(0) = 0 = C_0^0$$

$$sg(y + 1) = 1 = C_1^2(y, sg(y))$$

$$\text{also } sg = RC_0^0C_1^2$$

4 b)

$$\overline{sg}(0) = 1 = C_1^0$$

$$\overline{sg}(y + 1) = 0 = C_0^2(y, sg(y))$$

$$\text{also } \overline{sg} = RC_1^0C_0^2. \text{ Allerdings ist auch explizit}$$

$$\overline{sg}(y) = 1 \div y \text{ und daher } sg(y) = 1 \div \overline{sg}(y) = 1 \div (1 \div y).$$

Schließlich ist

4 c)

$$0 \div 1 = 0 = C_0^0$$

$$(y + 1) \div 1 = y = U_1^2(y, y \div 1),$$

$$\text{also } \div 1 = R C_0^0U_1^2 \text{ und daher } \div = R U_1^1((R C_0^0U_1^2) \circ U_3^3).$$

1.2.10 Lemma

Ist g eine n -stellige und h eine $(n + 2)$ -stellige partiell normiert berechenbare Funktion, so ist auch die durch (PR) definierte $(n + 1)$ -stellige partielle Funktion $f = Rgh$ partiell normiert berechenbar.

Beweis. Nach Voraussetzung gibt es normierte Registermaschinen M_g und M_h , die g und h berechnen. Ihre maximale Registerzahl sei $N - 1$. Dann suchen wir normierte N -Registermaschinen N_1 und N_2 mit folgender Wirkung:

$$N_1 : 0 \quad \overline{xy} \overline{0} \quad 0 \quad \Rightarrow f(\mathbf{x}, 0) \quad \mathbf{x} \quad 0 \quad \overline{0} \quad y$$

$$N_2 : f(\mathbf{x}, z) \quad \overline{\mathbf{x}z} \overline{0} \quad y - z \quad \Rightarrow f(\mathbf{x}, z + 1) \quad \mathbf{x} \quad z + 1 \quad \overline{0} \quad y - (z + 1) \text{ für } z < y.$$

Denn dann berechnet $M = N_1(N_2)_N$ die Funktion $f = Rgh$, wie man sich durch Induktion nach y , dem Inhalt des $(n+2)$ -ten Registers, klar macht. Wegen (PR) wird N_1 wesentlich mit M_g und N_2 wesentlich mit M_h konstruiert, etwa

$$\begin{aligned} N_1 &= K_{n+2} N M_g \text{ und} \\ N_2 &= K_{1 \ n+3} M_h L_{n+3} a_{n+2} s_N. \end{aligned}$$

Damit ist das Lemma bewiesen.

Es zeigt sich, daß sehr viele übliche zahlentheoretische Funktionen durch primitive Rekursion und simultane Einsetzung aus einfacheren Funktionen gewonnen werden können. Die aus einer gegebenen Menge Ψ von Funktionen mit diesen beiden Operationen konstruierbaren Funktionen nennt man die in Ψ primitiv-rekursiven Funktionen.

1.2.11 Induktive Definition

der in Ψ primitiv-rekursiven Funktionen. Es sei Ψ eine Menge von partiellen Funktionen.

1. *suc*, alle U_i^n , alle C_k^n und alle $f \in \Psi$ sind primitiv-rekursiv in Ψ .
2. Sind n -stellige g_1, \dots, g_r und ein r -stelliges h primitiv-rekursiv in Ψ , so ist auch $h \circ (g_1, \dots, g_r)$ primitiv-rekursiv in Ψ .
3. Sind ein n -stelliges g und ein $(n+2)$ -stelliges h primitiv-rekursiv in Ψ , so ist auch das durch (PR) definierte $f = Rgh$ primitiv-rekursiv in Ψ .

Ist f primitiv-rekursiv in der leeren Menge ($\Psi = \emptyset$) so heißt f (schlechthin) **primitiv-rekursiv** (abgekürzt **prim-rek**).

Beispiele. $+1, +, \div 1, \div, \cdot$, Potenz und Fakultät sind primitiv-rekursiv.

Bemerkung. Besteht Ψ nur aus totalen Funktionen (z. B. wenn Ψ leer ist), so ist auch jede in Ψ primitiv-rekursive Funktion total.

1.2.12 Satz

Es sei Ψ eine Menge von (partiell) normiert berechenbaren Funktionen. Dann ist jede in Ψ primitiv-rekursive Funktion (partiell) normiert berechenbar.

Beweis durch Induktion nach Definition 1.2.11.

1. Die Ausgangsfunktionen sind normiert berechenbar nach Lemma 1.2.5 und wegen der Voraussetzung über Ψ .

2. Die Komposition führt nach Lemma 1.2.8 von (partiell) normiert berechenbaren Funktionen stets wieder zu (partiell) normiert berechenbaren Funktionen.
3. Dasselbe gilt nach Lemma 1.2.10 für die primitive Rekursion.

Mit derselben Induktion sieht man auch die letzte Bemerkung ein.

1.2.13 Korollar

Jede primitiv-rekursive Funktion ist normiert berechenbar.

Dies ist der wesentlichste Spezialfall des Satzes.

Mit der primitiven Rekursion sind die Möglichkeiten, die die Iteration von (normierten) Registermaschinen bietet, nicht erschöpft. Ein trivialer Grund: Primitiv-rekursive Funktionen sind notwendig total, dagegen gibt es offenbar echt partielle, normiert berechenbare Funktionen. Man benutzt für den Übergang von g, h zu Rgh nur den Spezialfall $(s_i M)_i$, wobei M eine (normierte) Registermaschine ist, in deren Programm die Adresse i nicht auftritt. Der Inhalt des i -ten Registers, nach dem iteriert wird, wird daher in jeder Iterationsschleife um genau 1 verringert. Das Ende der Rechnung ist also abzusehen, sobald (vor der Iteration) eine Zahl in das i -te Register hineinkopiert ist. Wir untersuchen deshalb eine weitergehende Anwendung der Iteration, bei der die Registermaschine zwar nicht "zufällig" (wegen Lemma 1.1.5), aber doch "unvorhersehbar" zum Stop, d. h. zu einem Ergebnis kommt.

1.2.14 Definition

Es sei g eine $(n + 1)$ -stellige partielle Funktion, $\mathbf{x} \in \mathbb{N}^n$. Dann bezeichnet $\mu y(g(\mathbf{x}, y) = 0)$ die, sofern vorhanden, eindeutig bestimmte (kleinste) Zahl y mit $g(\mathbf{x}, y) = 0$ und $g(\mathbf{x}, z) > 0$ für alle $z < y$. Die durch die Gleichung

$$(\mu) \quad f(\mathbf{x}) \simeq \mu y(g(\mathbf{x}, y) = 0)$$

festgelegte partielle Funktion f heißt **mit dem μ -Operator zu g definiert** und wird durch μg bezeichnet. Die Gleichung (μ) heißt **Anwendung des μ -Operators auf g** .

$\text{dom } f$ besteht dann aus allen \mathbf{x} , zu denen es ein y gibt mit den Eigenschaften

1. $g(\mathbf{x}, y) = 0$
2. $(\mathbf{x}, z) \in \text{dom } g$ für alle $z \leq y$

Wenn es zu jedem \mathbf{x} ein solches y gibt, wenn also f total ist, so heißt (μ) eine Anwendung des μ -Operators **im Normalfall**.

1.2.15 Lemma

Ist g partiell normiert berechenbar, so auch $f = \mu g$. Liegt zusätzlich eine Anwendung des μ -Operators im Normalfall vor, so ist f normiert berechenbar.

Beweis. Wenn M g berechnet, so berechnet sowohl $a_1(L_1Ma_{n+2})_1K_{n+2}1s_1$ als auch $M(L_1a_{n+2}M)_1K_{n+2}1$ die Funktion $f = \mu g$. Der 2. Teil wiederholt nur, dass f (genau dann) total ist, wenn (μ) im Normalfall angewandt wird.

Bemerkung. Offenbar braucht f nicht total zu sein, wenn g es ist, und umgekehrt. Die Totalität von f hängt mehr an den Nullstellen von g als an der Totalität von g : g muss zu jedem \mathbf{x} "genügend früh" eine Nullstelle \mathbf{x}, y haben.

Der Übergang von g nach μg verwendet die Iteration einer Registermaschine in wesentlicher, nicht mehr eingeschränkter Form, im Gegensatz zur primitiven Rekursion. Zwar wird im Iterationsteil $(L_1Ma_{n+2})_1$ der Registermaschine, die μg berechnet, (vgl. Beweis von Lemma 1.2.15) zuerst das erste Register gelöscht, aber dann rechnet M einen Funktionswert von g in dieses Register hinein. Ob die Iterationsschleife noch einmal durchlaufen wird, hängt also davon ab, ob M einen positiven Funktionswert produziert. Auch wenn M für jede Eingabe ein Ergebnis liefert, wenn also g total ist, kann die Suche nach einer Nullstelle von g erfolglos bleiben, so dass das Programm $(L_1Ma_{n+2})_1$ niemals abbricht und niemals ein Ergebnis liefert. Darin äußert sich, dass μg nicht total zu sein braucht, auch wenn g es ist.

1.2.16 Induktive Definition der μ -partiell-rekursiven Funktionen

1. suc , alle U_i^n und alle C_k^n sind μ -partiell-rekursiv.
2. Mit g_1, \dots, g_r und h passender Stellenzahlen ist auch $h(g_1, \dots, g_r)$ μ -partiell-rekursiv. ○
3. Mit g, h passender Stellenzahlen ist auch $f = Rgh$ μ -partiell-rekursiv.
4. Mit g ist auch μg μ -partiell-rekursiv.

1.2.17 Definition

Die totalen μ -partiell-rekursiven Funktionen heißen μ -**rekursiv**.

Bemerkung. Wird in der Definition einer μ -partiell-rekursiven Funktion der μ -Operator nur im Normalfall angewandt, so ist sie μ -rekursiv. Die Umkehrung hiervon gilt hier nicht.

Die Definition der primitiv-rekursiven Funktionen wird in 1.2.16 nur um den 4. induktiven Schritt, die Anwendung des μ -Operators, erweitert. Da es sich um eine induktive Definition handelt, kann man die primitiv-rekursiven Operationen 2. und 3. wieder auf Funktionen anwenden, die mit dem μ -Operator definiert sind, um weitere μ -partiell-rekursive Funktionen zu erhalten. Folgendes Lemma liefert eine Alternative zu Definition 1.2.16, die die Definition 1.2.11 explizit verwendet.

1.2.18 Lemma

Die Klasse der μ -partiell-rekursiven Funktionen ist die kleinste Klasse Φ von partiellen Funktionen mit den folgenden beiden Eigenschaften:

1 Φ . Jede in Φ primitiv-rekursive Funktion gehört zu Φ .

4 Φ . Mit g gehört auch μg zu Φ .

Beweis. Die Klasse der μ -partiell rekursiven Funktionen hat nach 1., 2. und 3. in Definition 1.2.16 die Eigenschaft 1 Φ und nach 4. die Eigenschaft 4 Φ . Sie ist also eine Klasse der betrachteten Art und enthält deshalb die kleinste Klasse Φ mit den Eigenschaften 1 Φ und 4 Φ .

Umgekehrt enthält jede Klasse mit der Eigenschaft 1 Φ nach Definition 1.2.11 die Ausgangsfunktionen unter 1. in Definition 1.2.16, und sie ist abgeschlossen gegen Komposition und primitive Rekursion, erfüllt also 2. und 3. Durch Induktion nach Definition 1.2.16 folgt daher, dass jede, also auch die kleinste Klasse Φ mit 1 Φ und 4 Φ alle μ -partiell-rekursiven Funktionen enthält.

Die Aussage 1 Φ beinhaltet hiernach:

Korollar 1. Ist Ψ eine Menge von μ -partiell-rekursiven Funktionen, so ist jede in Ψ primitiv-rekursive Funktion μ -partiell-rekursiv.

Als Spezialfall für leeres Ψ erhält man hieraus:

Korollar 2. Die primitiv-rekursiven Funktionen sind μ -rekursiv.

Aus 1.2.12 und 1.2.15 ergibt sich eine Verbindung zwischen μ -Rekursivität und Berechenbarkeit.

1.2.19 Satz

Die μ -partiell-rekursiven Funktionen sind partiell normiert berechenbar. Die μ -rekursiven Funktionen sind demnach normiert berechenbar.

Beweis durch Induktion nach der induktiven Definition 1.2.16 der μ -partiell-rekursiven Funktionen.

1. Ist f eine Ausgangsfunktion suc, U_i^n oder C_k^n , so ist f nach Lemma 1.2.5 normiert berechenbar.
2. Ist $f = h \circ (g_1, \dots, g_r)$ nach 2. in Definition 1.2.16 eingeführt, so sind h, g_1, \dots, g_r μ -partiell-rekursive Funktionen, die vor f definiert worden sind. Dann sind nach Induktionsvoraussetzung h, g_1, \dots, g_r partiell normiert berechenbar, und nach Lemma 1.2.8 ist auch f partiell normiert berechenbar.
3. Ist $f = Rgh$ nach 3. in Definition 1.2.16 eingeführt, so sind g und h μ -partiell-rekursive Funktionen, die vor f definiert worden sind. Dann sind g und h nach Induktionsvoraussetzung partiell normiert berechenbar, und nach Lemma 1.2.10 gilt dies auch für f .
4. Ist $f = \mu g$ nach 4. in Definition 1.2.16 eingeführt, so ist g eine μ -partiell-rekursive Funktion, die vor f definiert worden ist. Dann ist g nach Induktionsvoraussetzung und deshalb nach Lemma 1.2.15 auch f partiell normiert berechenbar.

Damit ist die erste Aussage des Satzes bewiesen. Der zweite Teil ist nur die Einschränkung dieser ersten Aussage auf totale Funktionen.

Bemerkung. Die Teile 1. bis 3. dieses Beweises ergeben einen Beweis von Satz 1.2.12, wenn man “ μ -partiell-rekursiv” durch “primitiv-rekursiv in Ψ ” ersetzt. Deshalb ist der Beweis von Satz 1.2.12 auch nicht so ausführlich wie hier durchgeführt worden. Angesichts von Lemma 1.2.18 genügt es zu zeigen, dass die Klasse der partiell normiert berechenbaren Funktionen die Eigenschaften 1Φ und 4Φ hat, was nach Satz 1.2.12 und Lemma 1.2.15 der Fall ist.

Da jede Registermaschine einen Algorithmus, ein konstruktives Verfahren festlegt, sind insbesondere alle μ -rekursiven Funktionen im inhaltlichen Sinne berechenbar. Problematischer ist hiervon die Umkehrung, dass die μ -

rekursiven Funktionen bereits die sämtlichen im inhaltlichen Sinne berechenbaren (totalen) Funktionen sind und damit bereits alle denkbaren (totalen) Algorithmen liefern:

Church'sche These. Die μ -rekursiven Funktionen sind genau die im inhaltlichen Sinne berechenbaren Funktionen.

Diese These ist ihrer Art nach nicht zu beweisen, wohl aber zu erhärten. Wir haben dazu zumindest zu zeigen, dass alle (partiell) Register-berechenbaren Funktionen schon μ -partiell-rekursiv sind. Das beweisen wir in Kapitel 2 und erhalten damit eine Umkehrung der Sätze 1.1.14 und 1.2.19.

1.2.20 Aufgaben

1. Für welche Funktionen g, h gilt:

a) Rgh ist die Potenzfunktion $x, y \mapsto x^y$?

b) Rgh ist die Fakultät $x \mapsto x! = 1 \cdot 2 \cdot \dots \cdot (x - 1) \cdot x$?

Man schlieÙe mit Lemma 1.2.10, dass diese Funktionen normiert berechenbar sind.

2. Man definiere ohne μ -Operator die Funktionen

$$x \mapsto \mu y(x + y = 0)$$

$$x \mapsto \mu y(x \div y = 0)$$

$$x \mapsto \mu y(x \cdot y = 0).$$

1.3 Primitiv-rekursive Funktionen und Prädikate

Auch für die weitere Entwicklung der allgemeinen Theorie ist es jetzt notwendig, konkreter zu werden und wesentlich mehr Beispiele primitiv-rekursiver Funktionen kennen zu lernen. Zunächst einige einfache Struktureigenschaften.

1.3.1 Lemma

Hinzunahme, Vertauschung und Identifikation von Variablen sind primitiv-rekursive Operationen, d. h. ist

$$f(\mathbf{x}) \simeq g(\mathbf{x}')$$

und ist jedes x'_i ein x_{j_i} , so ist f primitiv-rekursiv in g .

Denn es ist $f = g \circ (U_{j_1}^n, \dots, U_{j_r}^n)$.

1.3.2 Lemma

Lokale Einsetzung ist eine primitiv-rekursive Operation, d. h. ist

$$f(\mathbf{x}) \simeq g(\mathbf{x}'_1, h(\mathbf{x}'_2), \mathbf{x}'_3)$$

und ist jedes x'_{ik} ein x_j , so ist f primitiv-rekursiv in g, h .

Denn nach Lemma 1.3.1 gibt es g', h' primitiv-rekursiv in g bzw. h mit

$$g'(\mathbf{x}, z) \simeq g(\mathbf{x}'_1, z, \mathbf{x}'_3) \text{ und } h'(\mathbf{x}) \simeq h(\mathbf{x}'_2),$$

und dann ist

$$f = g' \circ (U_1^n, \dots, U_n^n, h').$$

1.3.3 Lemma

Ist

$$f(\mathbf{x}, 0) \simeq g(\mathbf{x}') \text{ und } f(\mathbf{x}, y + 1) \simeq h(\mathbf{x}'', f(\mathbf{x}, y))$$

und ist jedes x'_i ein x_j , jedes x''_i ein x_k oder y , so ist f primitiv-rekursiv in g, h .

Denn nach Lemma 1.3.1 gibt es g', h' primitiv-rekursiv in g, h mit $f = Rg'h'$.

Hiernach können wir den Rekursionsgleichungen einer Funktion unmittelbar ansehen, ob sie primitiv-rekursiv ist.

1.3.4 Beispiele

$suc, +, \div 1, \div, \cdot, sg, \overline{sg}$ sind aus 1.2 als primitiv-rekursiv bekannt.

1. Die absolute Differenz $| - |$ mit

$$|x - y| = (x \div y) + (y \div x)$$

ist nach Lemma 1.3.2 primitiv-rekursiv.

2. Das Maximum max mit

$$max(x, y) = x, \text{ falls } y \leq x, \text{ } max(x, y) = y \text{ sonst}$$

ist nach 1.3.2 primitiv-rekursiv, weil $max(x, y) = x + (y \dot{-} x)$ ist.

3. Die Potenz und die Fakultät ! genügen

$$x^0 = 1 \text{ und } x^{y+1} = x^y \cdot x \text{ bzw. } 0! = 1 \text{ und } (y+1)! = y! \cdot (y+1)$$

und sind nach Lemma 1.3.3 primitiv-rekursiv.

4. Bezeichnet $res(x, y+1)$ den Rest bei Division von x durch $y+1$, so ist

$$res(0, y+1) = 0 \text{ und } res(x+1, y+1) = (res(x, y+1)+1) \cdot sg(y \dot{-} res(x, y+1)),$$

und das ist nach Lemma 1.3.3 primitiv-rekursiv.

5. Der ganzzahlige Anteil $\left[\frac{x}{y+1} \right]$ vom Quotienten $\frac{x}{y+1}$ ist primitiv-rekursiv, wie man sich überlegt.

Das Schema (PR') der Fallunterscheidung ist nicht nur für totale Funktionen aus (PR) ableitbar, sondern in jedem Fall:

1.3.5 Lemma

Die Fallunterscheidung ist eine primitiv-rekursive Operation: Gilt

$$(PR') \quad \begin{aligned} f(\mathbf{x}, 0) &\simeq g(\mathbf{x}) \\ f(\mathbf{x}, y+1) &\simeq h(\mathbf{x}, y), \end{aligned}$$

so ist f primitiv-rekursiv in g, h .

Beweis. Wir setzen

$$g'(\mathbf{x}, 0) = 0 \text{ und } g'(\mathbf{x}, z+1) \simeq U_1^2(g(\mathbf{x}), g'(\mathbf{x}, z)), \text{ ferner}$$

$$h'(\mathbf{x}, y, 0) = 0 \text{ und } h'(\mathbf{x}, y, z+1) \simeq U_1^2(h(\mathbf{x}, y), h'(\mathbf{x}, y, z)).$$

Nach Lemma 1.3.3 sind dann g', h' primitiv-rekursiv in g, h und es ist

$$g'(\mathbf{x}, 1) \simeq U_1^2(g(\mathbf{x}), 0) \simeq g(\mathbf{x}) \text{ und}$$

$$h'(\mathbf{x}, y, 1) \simeq U_1^2(h(\mathbf{x}, y), 0) \simeq h(\mathbf{x}, y).$$

(Mit Induktion nach z folgt allgemein

$$g'(\mathbf{x}, z + 1) \simeq g(\mathbf{x}) \text{ und } h'(\mathbf{x}, y, z + 1) \simeq h(\mathbf{x}, y),$$

aber das wird nicht gebraucht.) Nun sei

$$f(\mathbf{x}, y) \simeq g'(\mathbf{x}, \overline{sg}(y)) + h'(\mathbf{x}, y \div 1, sg(y)).$$

Nach Lemma 1.3.2 ist dann f primitiv-rekursiv in g', h' , also auch in g, h , und es folgt

$$f(\mathbf{x}, 0) \simeq g'(\mathbf{x}, 1) + h'(\mathbf{x}, 0, 0) \simeq g(\mathbf{x}) + 0 \simeq g(\mathbf{x})$$

$$f(\mathbf{x}, y + 1) \simeq g'(\mathbf{x}, 0) + h'(\mathbf{x}, y, 1) \simeq 0 + h(\mathbf{x}, y) \simeq h(\mathbf{x}, y).$$

Damit ist die Behauptung bewiesen.

Um f zu definieren, führt man also eine zusätzliche Rekursionsstelle ein, nämlich das letzte Argument z von g' und h' . Diese neuen Rekursionen braucht man nur bis zum Wert $z = 1$ laufen zu lassen. Dieser Definition von f entspricht auch die Konstruktion einer Registermaschine, die f berechnet, aus Registermaschinen, die g und h berechnen.

1.3.6 Lemma

Die übliche Fallunterscheidung ist eine primitiv-rekursive Operation, d. h. ist

$$\begin{aligned} f(\mathbf{x}) &\simeq g(\mathbf{x}), \text{ falls } k(\mathbf{x}) = 0, \\ f(\mathbf{x}) &\simeq h(\mathbf{x}), \text{ falls } k(\mathbf{x}) > 0, \\ &(\textit{f nicht definiert, falls } k \textit{ nicht definiert ist}), \end{aligned}$$

so ist f primitiv-rekursiv in g, h, k .

Beweis. $h'(\mathbf{x}, y) \simeq h(\mathbf{x})$ ist nach 1.3.1 primitiv-rekursiv in h ; dann ist $f' = R'gh'$ nach 1.3.5 primitiv-rekursiv in g und h , und es ist

$$\begin{aligned} f(\mathbf{x}) &\simeq f'(\mathbf{x}, k(\mathbf{x})) \quad \text{wegen} \\ f(\mathbf{x}) &\simeq f'(\mathbf{x}, 0) \simeq g(\mathbf{x}) \quad \text{für } k(\mathbf{x}) = 0 \quad \text{und} \\ f(\mathbf{x}) &\simeq f'(\mathbf{x}, y + 1) \simeq h'(\mathbf{x}, y) \simeq h(\mathbf{x}) \quad \text{für } k(\mathbf{x}) > 0. \end{aligned}$$

Nach 1.3.2 ist daher f primitiv-rekursiv in f' und k , also auch in g, h und k .

1.3.7 Definition des beschränkten μ -Operators

Das kleinste $i < y$, für das $g(\mathbf{x}, i) = 0$ ist, ist definiert durch (\vee wird **oder** gelesen)

$$(\mu i < y)(g(\mathbf{x}, i) = 0) = \mu i(g(\mathbf{x}, i) = 0 \vee i = y)$$

Der beschränkte μ -Operator ($\mu i < y$) sucht also nach Nullstellen nur unterhalb von y . Wenn die Suche bis dahin endlich und erfolglos war, wirft er als Wert die Schranke y aus: $(\mu i < y)(g(\mathbf{x}, i) = 0)$ ist genau dann $= y$, wenn $g(\mathbf{x}, i) > 0$ ist für alle $i < y$. Insbesondere ist stets $(\mu i < 0)(g(\mathbf{x}, i) = 0) = 0$.

1.3.8 Lemma

Beschränkte Summe, beschränktes Produkt und beschränkter μ -Operator sind primitiv-rekursive Operationen, d. h. ist

$$\begin{aligned} f(\mathbf{x}, y) &\simeq \sum_{i < y} g(\mathbf{x}, i), \\ f(\mathbf{x}, y) &\simeq \prod_{i < y} g(\mathbf{x}, i) \text{ bzw.} \\ (1) \quad f(\mathbf{x}, y) &\simeq (\mu i < y) (g(\mathbf{x}, i) = 0). \end{aligned}$$

so ist f primitiv-rekursiv in g .

Beweis. In den beiden ersten Fällen ist

$$\begin{aligned} f(\mathbf{x}, 0) = 0 \text{ und } f(\mathbf{x}, y + 1) &\simeq f(\mathbf{x}, y) + g(\mathbf{x}, y) \quad \text{bzw.} \\ f(\mathbf{x}, 0) = 1 \text{ und } f(\mathbf{x}, y + 1) &\simeq f(\mathbf{x}, y) \cdot g(\mathbf{x}, y), \end{aligned}$$

und $f(\mathbf{x}, y)$ ist nur definiert, wenn $g(\mathbf{x}, z)$ für alle $z < y$ definiert ist. Das ist beim beschränkten μ -Operator nicht notwendig. $f(\mathbf{x}, y)$ ist auch dann definiert, wenn für ein $z < y$ $\mu i(g(\mathbf{x}, i) = 0) < z$, aber $g(\mathbf{x}, z)$ nicht definiert ist.

Wir definieren f primitiv-rekursiv in g durch

$$f(\mathbf{x}, 0) = 0 \text{ und } f(\mathbf{x}, y + 1) \simeq f(\mathbf{x}, y) + sg(g(\mathbf{x}, f(\mathbf{x}, y)))$$

und zeigen (1) durch Induktion nach y .

1. $f(\mathbf{x}, 0) = 0 = (\mu i < 0)(g(\mathbf{x}, i) = 0)$.
2. Gelte (1) als Induktionsvoraussetzung. Wir unterscheiden drei Fälle.
 - 2.1 $f(\mathbf{x}, y) = z < y$. Nach Induktionsvoraussetzung ist dann $z = \mu i(g(\mathbf{x}, i) = 0)$ und daher $g(\mathbf{x}, z) = 0$. Also ist $f(\mathbf{x}, y + 1) = z + 0 = z = (\mu i < y + 1)(g(\mathbf{x}, i) = 0)$.

2.2 $f(\mathbf{x}, y) = y$. Dann ist

$$f(\mathbf{x}, y + 1) \simeq y + sg(g(\mathbf{x}, y)) = \begin{cases} y, & \text{falls } g(\mathbf{x}, y) = 0 \\ y + 1, & \text{falls } g(\mathbf{x}, y) > 0 \\ *, & \text{falls } g(\mathbf{x}, y) = *. \end{cases}$$

Weil nach Induktionsvoraussetzung $g(\mathbf{x}, i) > 0$ für alle $i < y$ ist, gilt stets die Induktionsbehauptung

$$f(\mathbf{x}, y + 1) \simeq (\mu i < y + 1)(g(\mathbf{x}, i) = 0)$$

2.3 $f(\mathbf{x}, y) = *$. Dann ist nach Induktionsvoraussetzung auch die rechte Seite von (1) nicht definiert.

In jedem Fall folgt die Induktionsbehauptung.

Mit Induktion folgt (1). Damit ist das Lemma bewiesen.

Oft möchte man nicht nur Funktionen, sondern auch Prädikate betrachten und Eigenschaften wie die (primitive) Rekursivität von Funktionen auf Prädikate übertragen.

1.3.9 Definition

Gegeben sei ein n -stelliges zahlentheoretisches Prädikat P , also $P \subseteq \mathbb{N}^n$. Die totale Funktion χ_P mit

$$\chi_P(\mathbf{x}) = 0 \leftrightarrow P(\mathbf{x}), \quad \chi_P(\mathbf{x}) = 1 \leftrightarrow \neg P(\mathbf{x})$$

heißt die **charakteristische Funktion** von P .

1.3.10 Definition

Ein Prädikat P heißt **primitiv-rekursiv** (in einer Menge M von Funktionen und Prädikaten), wenn χ_P primitiv-rekursiv (in $\{f \mid f \text{ Funktion aus } M\} \cup \{\chi_Q \mid Q \text{ Prädikat aus } M\}$) ist.

1.3.11 Beispiele

6. Die Gleichheit $=$ hat die charakteristische Funktion $\chi_= = sg \circ | - |$ wegen

$$sg(|x - y|) = 0 \leftrightarrow x = y, \quad sg(|x - y|) = 1 \leftrightarrow x \neq y.$$

Also ist $=$ primitiv-rekursiv.

7. Wegen $\overline{sg}(y \dot{-} x) = 0 \leftrightarrow x < y$, ist auch die Kleiner-Relation $<$ primitiv-rekursiv. Ebenso sind $\leq, >, \geq$ primitiv-rekursiv.
8. Wegen $res(x, y + 1) = 0 \leftrightarrow y + 1 | x$ ($y + 1$ teilt x) ist auch die Teilbarkeitsrelation $|$ primitiv-rekursiv. Hier stört, dass man res nur auf Zahlen $y + 1 > 0$ anwenden kann. Wie bezieht man den unwichtigen Fall der Division durch 0 ein? Wir untersuchen dazu einfache logische Zusammensetzungen von Prädikaten.

1.3.12 Lemma

Die Junktoren und die beschränkten Quantoren sind primitiv-rekursive Operationen, d. h. ist

$$\begin{array}{lll}
 P(\mathbf{x}) & \leftrightarrow & \neg Q(\mathbf{x}) \quad (\text{nicht } Q(\mathbf{x})), \\
 P(\mathbf{x}) & \leftrightarrow & Q(\mathbf{x}) \vee R(\mathbf{x}) \quad (Q(\mathbf{x}) \text{ oder } R(\mathbf{x})), \\
 P(\mathbf{x}) & \leftrightarrow & Q(\mathbf{x}) \wedge R(\mathbf{x}) \quad (Q(\mathbf{x}) \text{ und } R(\mathbf{x})), \\
 P(\mathbf{x}) & \leftrightarrow & (Q(\mathbf{x}) \rightarrow R(\mathbf{x})) \quad (\text{aus } Q(\mathbf{x}) \text{ folgt } R(\mathbf{x})), \\
 P(\mathbf{x}, y) & \leftrightarrow & \exists i < y Q(\mathbf{x}, i) \quad (\text{für ein } i < y : Q(\mathbf{x}, i)), \text{ bzw.} \\
 P(\mathbf{x}, y) & \leftrightarrow & \forall i < y Q(\mathbf{x}, i) \quad (\text{für alle } i < y : Q(\mathbf{x}, i)),
 \end{array}$$

so ist P primitiv-rekursiv in Q (und R).

Beweis. Es ist

$$\begin{array}{ll}
 \chi_{\neg Q} & = \overline{sg} \circ \chi_Q, \\
 \chi_{Q \vee R} & = \chi_Q \cdot \chi_R, \\
 \chi_{Q \wedge R} & = \max \circ (\chi_Q, \chi_R), \\
 \chi_{Q \rightarrow R} & = (\overline{sg} \circ \chi_Q) \cdot \chi_R,
 \end{array}$$

und in den letzten beiden Fällen

$$\begin{array}{ll}
 \chi_P(\mathbf{x}, y) & = \prod_{i < y} \chi_Q(\mathbf{x}, i) \text{ bzw.} \\
 \chi_P(\mathbf{x}, y) & = sg(\sum_{i < y} \chi_Q(\mathbf{x}, i)).
 \end{array}$$

Schreibweise. Im folgenden steht $\exists i \leq y Q(x, i)$ oft für $\exists i < y + 1 Q(x, i)$, ebenso $\forall i \leq y Q(x, i)$ für $\forall i < y + 1 Q(x, i)$.

1.3.13 Lemma

Die Teilbarkeitsrelation $|$ ist wegen

$$x | y \leftrightarrow \exists i \leq y \ x \cdot i = y$$

primitiv-rekursiv.

Beweis. $x|y$ ist definiert durch $\exists i \ x \cdot i = y$. Daraus folgt stets $\exists i \leq y \ x \cdot i = y$, auch für $y = 0$. Auf Grund dieser Darstellung ist $|$ primitiv-rekursiv wegen Beispiel 6 in 1.3.11 und Lemma 1.3.12.

1.3.14 Definition

Eine natürliche Zahl n heißt **Primzahl**, $Prim(x)$, wenn $x > 1$ ist und aus $i|x$ stets $i = 1$ oder $i = x$ folgt. (Es gibt unendlich viele Primzahlen.) p_i bezeichne die *i*-te **Primzahl**: $p_0 = 2$, und p_{i+1} ist die kleinste Primzahl $> p_i$. Es sei $(0)_i = 0$ für alle i . In der (eindeutigen) Primfaktorzerlegung von $x > 0$ bezeichne $(x)_i$ den Exponenten der *i*-ten Primzahl. Wir nennen $(x)_i$ die *i*-te **Komponente** von x .

1.3.15 Lemma

Die Primzahleigenschaft $Prim$, die Primzahlfunktion $p : i \mapsto p_i$ und die Komponentenfunktion $x, i \mapsto (x)_i$ sind primitiv-rekursiv.

Beweis. $Prim(x) \leftrightarrow x > 1 \wedge \forall i \leq x (i|x \rightarrow i = 1 \vee i = x)$. Also ist $Prim$ primitiv-rekursiv nach Lemma 1.3.12, und da $>, =$ und $|$ primitiv-rekursiv sind.

Da nach Euklid $p_{i+1} \leq p_i! + 1$ ist, ist $p_0 = 2$ und

$$p_{i+1} = (\mu x \leq p_i! + 1)(p_i < x \wedge Prim(x)).$$

Mit $!, suc, <$ und $Prim$ ist nach Lemma 1.3.2, 1.3.8, 1.3.12 auch die durch

$$h(i, z) = (\mu x \leq z! + 1)(z < x \wedge Prim(x))$$

definierte Funktion h primitiv-rekursiv. Also ist $p = R2h : i \mapsto p_i$ primitiv-rekursiv. Schließlich ist

$$(x)_i = (\mu y \leq x)(\neg p_i^{y+1} | x),$$

so dass nach Lemma 1.3.8, 1.3.12, 1.3.13 auch diese Funktion primitiv-rekursiv ist.

1.3.16 Bemerkungen

Für alle x und i gilt:

$$(0)_i = (1)_i = 0,$$

$$\begin{aligned}
x > 0 &\rightarrow ((x)_i > 0 \leftrightarrow p_i | x), \\
i \geq x &\rightarrow (x)_i = 0, \quad x > 0 \rightarrow (x)_i < x, \\
x > 0 &\leftrightarrow \exists n \ x = \langle (x)_0, \dots, (x)_{n-1} \rangle
\end{aligned}$$

(für $n = 0$ ist dies das einzige 0-Tupel $\langle \rangle = 1$).

Abkürzung. Statt $((x)_i)_j$ schreibt man $(x)_{i,j}$.

Beispiel. $(512)_0 = 9$, $(512)_{0,0} = 0$, $(512)_{0,1} = 2$, $(512)_{0,1,0} = 1$.

1.3.17 Definition der Tupel-Kodierung

Zu jedem $n \in \mathbb{N}$ sei $\langle \dots \rangle^n$ die n -stellige **Kodierfunktion**

$$\langle \dots \rangle : x_0, \dots, x_{n-1} \mapsto \langle x_0, \dots, x_{n-1} \rangle := \prod_{i < n} p_i^{x_i}.$$

Häufig lassen wir den oberen Index n fort, wenn klar ist, welche Stellenzahl gemeint ist. Wir nennen $\langle \mathbf{x} \rangle$ auch den **Code** von \mathbf{x} .

1.3.18 Satz über Kodierung und Dekodierung

Für jedes $n \in \mathbb{N}$ ist $\langle \dots \rangle^n$ eine primitiv-rekursive Injektion von \mathbb{N}^n in \mathbb{N} mit primitiv-rekursiven Inversen:

$$x = \langle x_0, \dots, x_{n-1} \rangle \implies (x)_i = x_i.$$

Beweis. Nach Lemma 1.3.8 und 1.3.15 ist

$$x_0, \dots, x_{n-1} \mapsto \prod_{i < n} p_i^{x_i}$$

primitiv-rekursiv. Wegen der Eindeutigkeit der Primfaktorzerlegung ist diese Abbildung injektiv, und ihre Inversen sind für $i = 0, \dots, n-1$ die Funktionen $x \mapsto (x)_i$, weil $((x_0, \dots, x_{n-1}))_i = \left(\prod_{i < n} p_i^{x_i} \right)_i = x_i$ ist. Und die sind nach Lemma 1.3.15 und 1.3.2 primitiv-rekursiv.

1.3.19 Lemma

Zu jeder n -stelligen partiellen Funktion f ist durch

$$\langle f \rangle(x) \simeq f((x)_0, \dots, (x)_{n-1})$$

eine 1-stellige partielle Funktion $\langle f \rangle$ primitiv-rekursiv in f definiert, für die

$$f(\mathbf{x}) \simeq \langle f \rangle(\langle \mathbf{x} \rangle)$$

ist, so dass auch f primitiv-rekursiv in $\langle f \rangle$ ist.

Beweis. Nach Lemma 1.3.15 ist $\langle f \rangle$ primitiv-rekursiv in f . Ferner ist nach Satz 1.3.18 für $\mathbf{x} = x_0, \dots, x_{n-1}$

$$f(\mathbf{x}) \simeq f(\langle \mathbf{x} \rangle_0, \dots, \langle \mathbf{x} \rangle_{n-1}) \simeq \langle f \rangle(\langle \mathbf{x} \rangle).$$

Damit ist f auch primitiv-rekursiv in $\langle f \rangle$.

Modulo Primitiv-rekursivem kann man sich also auf 1-stellige Funktionen beschränken.

1.3.20 Bemerkung

Für kein n ist $\langle \dots \rangle^n$ bijektiv auf \mathbb{N} , und es ist stets $\langle \mathbf{x}, 0 \rangle = \langle \mathbf{x} \rangle$, so dass die Vereinigung verschiedenstelliger $\langle \dots \rangle$ nicht mehr injektiv ist. Beides lässt sich leicht beheben. Z. B. ist die Funktion $\pi : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit

$$\pi(x, y) = \frac{1}{2} \cdot (x + y) \cdot (x + y + 1) + x$$

eine primitiv-rekursive Bijektion von \mathbb{N}^2 auf \mathbb{N} mit primitiv-rekursiven Inversen, wie man sich selbst überlegt. Diese Funktion π liegt offenbar dem 2. Cantorschen Diagonalverfahren, dem Beweis der Abzählbarkeit von $\mathbb{N} \times \mathbb{N}$, zu Grunde.

Andererseits ist die Vereinigung aller $\langle \dots \rangle^n$ "beinahe" injektiv, nämlich auf der Menge aller Tupel, deren letztes Glied nicht 0 ist. Dieses letzte Glied $\neq 0$ bestimmt die (relevante) Länge eines Tupels \mathbf{x} und auch seines Codes $\langle \mathbf{x} \rangle$:

1.3.21 Definition

Die **Länge** $lh(x)$ einer Zahl x sei das kleinste $n \in \mathbb{N}$, für das x im Bild von $\langle \dots \rangle^n$ liegt, also Code eines n -Tupels ist. Gibt es kein solches n , so sei $lh(x) = 0$. x heißt **Sequenzzahl**, $Seq(x)$, wenn x Code eines $lh(x)$ -Tupels aus lauter positiven Zahlen ist.

1.3.22 Lemma

Die Funktion lh und das Prädikat Seq sind primitiv-rekursiv wegen

$$lh(x) = (\mu i < x) \forall j < x ((x)_j > 0 \rightarrow j < i) \text{ und}$$

$$Seq(x) \leftrightarrow x > 0 \wedge \forall i < lh(x) (x)_i > 0$$

Beweis. Jedes $x > 0$ hat eine Primfaktorzerlegung und ist deshalb Code eines Tupels, und zwar eines $lh(x)$ -Tupels. Dann ist für alle x (auch für $x = 0$)

$$lh(x) = \mu n (\forall i ((x)_i > 0 \rightarrow i < n))$$

Da für $i \geq x$ ohnehin $(x)_i = 0$ ist, kann man hierin den Quantor $\forall i$ ersetzen durch den beschränkten Quantor $(\forall i < x)$. Nach Lemma 1.3.12 und 1.3.15 gibt es also eine primitiv-rekursive Funktion g mit

$$\begin{aligned} g(x, i) = 0 &\leftrightarrow (\forall j < x) ((x)_j > 0 \rightarrow j < i) \\ &\leftrightarrow \forall j ((x)_j > 0 \rightarrow j < i). \end{aligned}$$

Es ist stets $g(x, x) = 0$, da selbstverständlich $(\forall j < x) (\dots \rightarrow j < x)$ gilt, so dass

$$lh(x) = \mu i (g(x, i) = 0) = (\mu i < x) (g(x, i) = 0)$$

die behauptete primitiv-rekursive Darstellung von lh ist. Die Behauptung über Seq ist klar.

Zur späteren Verwendung führen wir folgende mit den Kodierfunktionen $\langle \dots \rangle$ zusammenhängende Funktionen ein.

1.3.23 Definition

$ggT(x, y)$ bezeichne den größten gemeinsamen Teiler von x und y . Die **bedingte Division** $:$ ist dann definiert durch

$$\begin{aligned} x : 0 &= x \\ x : y &= x : ggT(x, y) \text{ für } y > 0. \end{aligned}$$

Die **Ersetzung** der i -ten Komponente von x durch z ist definiert durch

$$\begin{aligned} 0(z/i) &= 0 \text{ und} \\ x(z/i) &= \langle (x)_0, \dots, z, \dots, (x)_{lh(x)-1} \rangle \text{ für } x > 0. \end{aligned}$$

Die **Verkettungsfunktion** $*$ ist definiert durch

$$\begin{aligned} 0 * y &= x * 0 = 0 \text{ und} \\ x * y &= \langle (x)_0, \dots, (x)_{lh(x)-1}, (y)_0, \dots, (y)_{lh(y)-1} \rangle \text{ für } x, y > 0. \end{aligned}$$

1.3.24 Bemerkung

Die Tupel-Kodierung verbindet die Arithmetik der positiven natürlichen Zahlen mit einer Arithmetik der Tupel. Grundlegend sind die Regeln (für $x, y > 0$):

$$\begin{aligned} \forall i (x \cdot y)_i &= (x)_i + (y)_i \text{ und} \\ x|y &\iff \forall i (x)_i \leq (y)_i, \end{aligned}$$

woraus man schließt auf

$$\forall i (ggT(x, y))_i = \min((x)_i, (y)_i)$$

und weiter auf

$$y|x \Rightarrow \forall i (x : y)_i = (x)_i - (y)_i = (x)_i \dot{-} (y)_i.$$

Bei der bedingten Division $x : y$ wird für $y > 0$ die Zahl x nur durch die Teiler von y dividiert, durch die x teilbar ist. Der zusätzliche (dritte) Punkt über $:$ bei $:$ entspricht dem zusätzlichen Punkt über $-$ bei $\dot{-}$. Man hat allgemein:

$$\forall i (x : y)_i = (x)_i \dot{-} (y)_i.$$

Kodieren x und y Zahlentupel \mathbf{x} und \mathbf{y} , ist also $x = \langle \mathbf{x} \rangle$ und $y = \langle \mathbf{y} \rangle$, so erhält man bei naheliegender Festsetzung der verwendeten Operationen auf Tupeln:

$$\begin{aligned} x \cdot y &= \langle \mathbf{x} + \mathbf{y} \rangle \\ x|y &\iff \mathbf{x} \leq \mathbf{y} \\ ggT(x, y) &= \langle \min(\mathbf{x}, \mathbf{y}) \rangle \\ x : y &= \langle \mathbf{x} \dot{-} \mathbf{y} \rangle \end{aligned}$$

und falls $lh(x)$ die Länge von \mathbf{x} ist,

$$x * y = \langle \mathbf{x}, \mathbf{y} \rangle$$

1.3.25 Lemma

Die bedingte Division $:$, die Ersetzung $(/)$ und die Verkettung $*$ sind primitivrekursiv wegen

$$\begin{aligned} x : y &= \langle (x)_0 \dot{-} (y)_0, \dots, (x)_x \dot{-} (y)_x \rangle \\ &= \prod_{i < lh(x)} p_i^{(x)_i \dot{-} (y)_i} \text{ für } x > 0, \\ x(z/i) &= (x : p_i^{(x)_i}) \cdot p_i^z \text{ und} \\ x * y &= x \cdot \prod_{i < lh(y)} p_{lh(x)+i}^{(y)_i} \text{ für } y > 0. \end{aligned}$$

Beweis. Die Behauptung über $\dot{}$ folgt aus der eindeutigen Primfaktorzerlegung und $x \dot{} y = x - \min(x, y)$. Wegen

$$p_i^{(x)i} = \langle 0, \dots, (x)_i, \dots, 0 \rangle$$

ist daher

$$x \dot{} p_i^{(x)i} = \langle (x)_0, \dots, 0, \dots, (x)_x \rangle.$$

woraus die zweite Behauptung folgt. Schließlich ist für $x, y > 0$

$$\begin{aligned} x \cdot \prod_{i < \text{lh}(y)} p_{\text{lh}(x)+i}^{(y)i} &= x \cdot \langle 0, \dots, 0, (y)_0, \dots, (y)_y \rangle \\ &= \langle (x)_0, \dots, (x)_{\text{lh}(x)-1}, (y)_0, \dots, (y)_{\text{lh}(y)-1} \rangle. \end{aligned}$$

Nach Lemma 1.3.8 und 1.3.15 sind damit die betrachteten Funktionen primitiv-rekursiv.

Zum Schluß dieses Paragraphen betrachten wir Verallgemeinerungen des Schemas (*PR*) der primitiven Rekursion, die ebenfalls durch die Kodierfunktionen \langle, \dots, \rangle nahegelegt werden.

1.3.26 Lemma

Die simultane Rekursion ist eine primitiv-rekursive Operation, d. h. ist für $i < n$

$$\begin{aligned} f_i(\mathbf{x}, 0) &\simeq g_i(\mathbf{x}) \quad \text{und} \\ f_i(\mathbf{x}, y+1) &\simeq h_i(\mathbf{x}, y, f_0(\mathbf{x}, y), \dots, f_{n-1}(\mathbf{x}, y)), \end{aligned}$$

so sind alle f_i primitiv-rekursiv in $\{g_0, h_0, \dots, g_{n-1}, h_{n-1}\}$, falls alle g_i und alle h_i dieselbe Definitionsmenge haben.

Beweis. Es sei

$$g(\mathbf{x}) \simeq \langle g_0(\mathbf{x}), \dots, g_{n-1}(\mathbf{x}) \rangle \quad \text{und}$$

$$h(\mathbf{x}, y, z) \simeq \langle h_0(\mathbf{x}, y, (z)_0, \dots, (z)_{n-1}), \dots, h_{n-1}(\mathbf{x}, y, (z)_0, \dots, (z)_{n-1}) \rangle.$$

Für $f = Rgh$ gilt dann $f_i(\mathbf{x}, y) \simeq (f(\mathbf{x}, y))_i$. Daraus folgt die Behauptung.

1.3.27 Definition

Zu jedem $(n+1)$ -stelligen f heißt \bar{f} mit

$$\bar{f}(\mathbf{x}, y) \simeq \langle f(\mathbf{x}, 0), f(\mathbf{x}, 1), \dots, f(\mathbf{x}, y-1) \rangle$$

der Wertverlauf von f .

1.3.28 Lemma

Der Wertverlauf von f ist primitiv-rekursiv in f .

Dem es ist

$$\bar{f}(\mathbf{x}, y) \simeq \prod_{i < y} p_i^{f(\mathbf{x}, i)},$$

und das ist nach 1.3.8 und 1.3.15 primitiv-rekursiv in f .

1.3.29 Satz über Wertverlaufsrekursion

Die Wertverlaufsrekursion ist eine primitiv-rekursive Operation, d. h. ist

$$f(\mathbf{x}, y) \simeq h(\mathbf{x}, y, \bar{f}(\mathbf{x}, y)),$$

so ist f primitiv-rekursiv in h .

Beweis. Es ist

$$\begin{aligned} \bar{f}(\mathbf{x}, 0) &= 1 \quad \text{und} \\ \bar{f}(\mathbf{x}, y + 1) &\simeq \bar{f}(\mathbf{x}, y) \cdot p_y^{h(\mathbf{x}, y, \bar{f}(\mathbf{x}, y))}. \end{aligned}$$

Also ist \bar{f} primitiv-rekursiv in h , und es ist

$$f(\mathbf{x}, y) \simeq (\bar{f}(\mathbf{x}, y + 1))_y,$$

q. e. d.

Gängige Operationen auf Funktionen sind hiernach sehr oft primitiv-rekursiv. Aber mit einfachsten Mitteln kann man auch totale, inhaltlich berechenbare Funktionen angeben, die nicht primitiv-rekursiv sind.

1.3.30 Definition

Die (Ackermann-)Pétersche Funktion f ist definiert durch

$$\begin{aligned} f(0, y) &= y + 1 \\ f(x + 1, 0) &= f(x, 1) \\ f(x + 1, y + 1) &= f(x, f(x + 1, y)). \end{aligned}$$

Satz (ohne Beweis). Die Pétersche Funktion ist nicht primitiv-rekursiv. Vielmehr gibt es zu jeder n -stelligen primitiv-rekursiven Funktion g eine Zahl x_0 , so dass stets

$$g(x_1, \dots, x_n) < f(x_0, x_1 + \dots + x_n).$$

Jeder "Zweig" der Péterschen Funktion, d. h. jede Funktion $f_x : y \mapsto f(x, y)$ ist offenbar primitiv-rekursiv; obwohl f durch ein der primitiven Rekursion ähnliches Schema definiert ist, steigt f (in beiden Argumenten) nach dem Satz zu stark, um noch primitiv-rekursiv zu sein. Es gibt also im inhaltlichen Sinne berechenbare Funktionen, die nicht primitiv-rekursiv sind. Wir werden im nächsten Kapitel Aufzählungsverfahren kennenlernen, die diese Tatsache fast unmittelbar, ohne längere Überlegung, einsichtig machen.

1.3.31 Aufgaben

1. Zeigen Sie: Das Minimum min , gegeben durch

$$min(x, y) = x, \text{ falls } x \leq y, \quad min(x, y) = y \text{ sonst}$$

ist primitiv-rekursiv.

2. Beweisen Sie die Behauptung von Beispiel 5 aus 1.3.4
3. Zeigen Sie: Ist

$$\begin{aligned} f(\mathbf{x}) &\simeq g_1(\mathbf{x}), & \text{falls } k_1(\mathbf{x}) = 0 \\ f(\mathbf{x}) &\simeq g_i(\mathbf{x}), & \text{falls } k_j(\mathbf{x}) > 0 \quad \text{für } j < i \text{ und } k_i(\mathbf{x}) = 0 \quad (i < r) \\ f(\mathbf{x}) &\simeq g_r(\mathbf{x}), & \text{falls } k_j(\mathbf{x}) > 0 \quad \text{für } j = 0, \dots, r-1, \end{aligned}$$

so ist f primitiv-rekursiv in $\{g_1, \dots, g_r, k_1, \dots, k_{r-1}\}$.

4. Beweisen Sie die Behauptungen aus 1.3.16
5. Zeigen Sie, dass die Funktion π aus 1.3.20 eine primitiv-rekursive Bijektion von $\mathbb{N} \times \mathbb{N}$ auf \mathbb{N} ist mit primitiv-rekursiven Inversen π_1 und π_2 , gegeben durch

$$\pi_1(\pi(x, y)) = x \quad \text{und} \quad \pi_2(\pi(x, y)) = y.$$

6. Zeigen Sie für $x, y > 0$:

$$x \cdot y = \langle (x)_0 + (y)_0, \dots, (x)_{\max(x,y)} + (y)_{\max(x,y)} \rangle \text{ und}$$

$$ggT(x, y) = \langle min((x)_0, (y)_0), \dots, min((x)_x, (y)_x) \rangle.$$

Kapitel 2

Elementare Rekursionstheorie

2.1 Arithmetisierung und Kleene's Normalform-Theorem

In 1.2 haben wir eine partielle Funktion f als **partiell berechenbar** definiert, wenn es eine Registermaschine gibt, die f berechnet. Anschließend haben wir zahlreiche Funktionen als (partiell) berechenbar nachgewiesen. Der Begriff diente also in erster Linie zur Demonstration der Leistungsfähigkeit und Reichhaltigkeit des Konzepts der Registermaschinen. In diesem Kapitel sind wir umgekehrt daran interessiert, Grenzen der Leistungsfähigkeit der Registermaschinen und damit Grenzen des Berechenbarkeitsbegriffs aufzuzeigen. Dazu benutzen wir die Definition aus 1.2, um festzustellen, welche Forderung an die Registermaschine M gestellt wird, damit M eine partielle Funktion berechnet:

Eine Registermaschine M **berechnet** eine n -stellige partielle Funktion genau dann, wenn jede (also insbesondere die kürzeste) M -Rechnung mit einer Eingabe $0\mathbf{x}\bar{0}$, die endet, ein Ergebnis $y_1\mathbf{x}\bar{0}$ hat.

Wegen der Determiniertheit von M ist dieses y_1 eindeutig bestimmt. Die durch M berechnete n -stellige partielle Funktion f hat dann die Definitionsmenge

$$\text{dom } f = \{ \mathbf{x} \mid M, \text{ angesetzt auf } 0\mathbf{x}\bar{0}, \text{ hat ein Ergebnis } \},$$

und es ist für alle $\mathbf{x} \in \text{dom } f$

$$(1) \quad f(\mathbf{x}) = \text{das } y_1, \text{ für das } M : 0\mathbf{x}\bar{0} \Rightarrow y_1\mathbf{x}\bar{0} \text{ gilt .}$$

Wir verdeutlichen den mathematischen Kern dieses Sachverhalts, indem wir zunächst ein inhaltliches T -Prädikat \mathcal{T}_n definieren:

$$\mathcal{T}_n(M, \mathbf{x}, R) : \iff \begin{array}{l} M \text{ ist eine Registermaschine und} \\ R \text{ ist eine abgeschlossene } M\text{-Rechnung} \\ \text{mit der Eingabe } 0\mathbf{x}\bar{0} \end{array}$$

Ferner definieren wir für solche Rechnungen R die inhaltliche Auswertungsfunktion \mathcal{U} durch

$$\mathcal{U}(R) := \text{der Inhalt des 1. Registers in der letzten Konfiguration von } R.$$

Wenn nun die Registermaschine M die n -stellige partielle Funktion f berechnet, können wir aus (1) folgern:

$$(2) \quad f(\mathbf{x}) \simeq \mathcal{U}(\text{das kürzeste } R \text{ mit } \mathcal{T}_n(M, \mathbf{x}, R))$$

Eine Registermaschine M ist eine endliche Folge von Instruktionen, und eine M -Rechnung ist eine endliche Folge von M -Konfigurationen. M -Instruktionen und M -Konfigurationen sind ihrerseits endliche Folgen von Grundzeichen. Diese Grundzeichen sind außer den natürlichen Zahlen nur noch die Zeichen $+$ und $-$. Wenn wir auch $+$ und $-$ "unverwechselbar" durch natürliche Zahlen bezeichnen, dann können wir nach Satz 1.3.18 auch endliche Folgen von Grundzeichen und dann auch endliche Folgen von solchen endlichen Folgen durch natürliche Zahlen kodieren oder arithmetisieren und die Folgenglieder wieder aus einem solchen Code ablesen.

Das ist leicht möglich. Denn $+$ und $-$ treten nur in der 3. Spalte (des Programms) von M auf, wo sonst nur Instruktionsnummern $0, \dots, s$ ($s = \text{Anzahl der Instruktionen von } M$) auftreten. Zahlen $> s$ können wir in diesen Positionen also als Codes für $+$ und $-$ verwenden. Dementsprechend setzen wir:

2.1.1 Definition

der **Arithmetisierung** $\ulcorner \urcorner$ für Registermaschinen M der Länge s :

$$\begin{array}{ll} \ulcorner n \urcorner := n & \text{für } n \in \mathbb{N} \\ \ulcorner + \urcorner := s + 1 & \\ \ulcorner - \urcorner := s + 2 & \text{und weiter} \end{array}$$

$$\begin{array}{ll} \ulcorner (j, i, b, l) \urcorner & := \langle j, i, \ulcorner b \urcorner, l \rangle \\ \ulcorner (j, \mathbf{x}) \urcorner & := \langle j, \mathbf{x} \rangle \\ \ulcorner M \urcorner & := \langle \ulcorner I_0 \urcorner, \dots, \ulcorner I_{s-1} \urcorner \rangle \quad \text{für } M = (I_0, \dots, I_{s-1}) \text{ und} \end{array}$$

$\ulcorner R \urcorner := \langle \langle j_0, \mathbf{x}_0 \rangle, \dots, \langle j_T, \mathbf{x}_T \rangle \rangle$ für M -Rechnungen $R = ((j_0, \mathbf{x}_0), \dots, (j_T, \mathbf{x}_T))$

Die Codes $\ulcorner M \urcorner$ und $\ulcorner R \urcorner$ nennt man auch die **Gödel-Nummern** von M bzw. R .

Diese Codes sehen (als Rechenausdrücke) also aus wie die ursprünglichen Objekte, nur dass

- i. die runden Klammern durch spitze ersetzt sind und
- ii. $+$ durch $s + 1$, $-$ durch $s + 2$ ersetzt ist (für $s = lh(e)$).

2.1.2 Lemma

1. Ist e Gödel-Nummer einer Registermaschine M , so kann man M aus e eindeutig rekonstruieren.
2. Ist y Gödel-Nummer einer M -Rechnung R , so kann man R aus y eindeutig rekonstruieren.

Beweis.

1. Da für Instruktionen I_j stets $\ulcorner I_j \urcorner > 0$ ist, ist $lh(e) = s$ die Länge (Anzahl der Instruktionen) von M , und $(e)_j$ ist nach 1.3.18 für $j < s$ der Code $\langle j, i, \overline{b}, l \rangle$ der Instruktion I_j von M . Je nachdem $\overline{b} = s + 1, s + 2$ bzw. $\leq s$ ist, ist I_j die Addier-, Subtrahier- bzw. Test-Instruktion (j, i, b, l) .
2. folgt direkt aus 1.3.18. Aber nur wenn M , zumindest die Länge s von M bekannt ist, ist aus y zu entnehmen, ob R eine abgeschlossene M -Rechnung ist, nämlich genau im Fall $(y)_{lh(y)-1,0} = s$.

Schreibweise. Wegen dieser engen Entsprechung schreiben wir öfters ” e ist $\ulcorner Registermaschine \urcorner$ ” für ” e ist Code (Gödel-Nummer) einer Registermaschine” und ” y ist $\ulcorner e - Rechnung \urcorner$ ” für ” y ist Code einer M -Rechnung mit $\ulcorner M \urcorner = e$ ”, etc. Unsere Begriffe über Registermaschinen und über Rechnungen auf Registermaschinen werden durch die Arithmetisierung zu zahlentheoretischen Prädikaten. Insbesondere erhalten wir arithmetische Entsprechungen T_n und U zu den inhaltlichen T -Prädikaten \mathcal{T}_n und der Auswertungsfunktion \mathcal{U} :

2.1.3 Definition

Das **Kleene'sche T -Prädikat** ist (zu $n \in \mathbb{N}$) das $(n + 2)$ -stellige arithmetische Prädikat T_n , gegeben durch

$$T_n(e, \mathbf{x}, y) \iff y \text{ ist eine } \lceil \text{abgeschlossene Rechnung} \rceil \text{ auf einer} \\ \lceil \text{Registermaschine} \rceil \ e \text{ mit der Anfangskonfiguration } (0, 0\mathbf{x}\bar{0}).$$

Ferner liefert die 1-stellige arithmetische Auswertungsfunktion U zu einer $\lceil \text{abgeschlossenen } e\text{-Rechnung} \rceil \ y$ den Inhalt des 1. Registers der letzten $\lceil \text{Konfiguration} \rceil$ von y .

Ist e eine $\lceil \text{Registermaschine} \rceil$, die die partielle Funktion f berechnet, so ist nach (2) und Lemma 2.1.2

$$(3) \quad f(\mathbf{x}) \simeq U(\mu y T_n(e, \mathbf{x}, y)).$$

Wir wollen zeigen, dass man das Prädikat T_n und die Funktion U primitiv-rekursiv definieren kann. Dann ist die beliebige n -stellige partielle Registerberechenbare Funktion f durch (3) als spezielle μ -partiell-rekursive Funktion dargestellt. Das ist die Aussage von Kleene's Normalform-Theorem.

Kleene's Normalform-Theorem schließt die in Kapitel 1 gebliebene Lücke und zeigt die Gleichwertigkeit der drei Begriffe

- f ist partiell berechenbar,
- f ist partiell normiert berechenbar,
- f ist μ -partiell-rekursiv.

Es schließt damit die einleitenden Präzisierungen des intuitiven Berechenbarkeitsbegriffs ab.

Da (3) offenbar für **jedes** $e \in \mathbb{N}$ eine μ -partiell-rekursive Funktion f definiert - und nicht nur für die wenigen $\lceil \text{Registermaschinen} \rceil \ e$, die Funktionen berechnen -, liefert Kleene's Normalform-Theorem zusätzlich eine Aufzählung aller partiell berechenbaren Funktionen (mit Wiederholungen). Damit erhält man schlagartig einen besseren allgemeinen Überblick über diese Klasse und einen Zugang zur eigentlichen Rekursionstheorie.

Als Erstes arbeiten wir auf eine primitiv-rekursive Definition des T -Prädikats und der Auswertungsfunktion U und damit auf einen Beweis des Normalform-Theorems hin.

2.1.4 Definition

x ist 'Instruktion' in e :

$$Instr(x, e) : \iff lh(x) \leq 4 \wedge 1 \leq (x)_1 \wedge (x)_2 \leq lh(e) + 2 \wedge (x)_3 \leq lh(e)$$

e ist 'Registermaschine' :

$$RM(e) : \iff \forall j < lh(e) (Instr((e)_j, e) \wedge (e)_{j,o} = j) \wedge e > 0.$$

$RM(e)$ heißt nach 1.3.18 also: e ist ein Tupel $\langle (e)_0, \dots, (e)_{s-1} \rangle$ von $s = lh(e)$ 'Instruktionen', wobei die j -te 'Instruktion' $(e)_j$ ein Quadrupel $\langle j, i, \overline{b}, l \rangle$ mit $i \geq 1$, $\overline{b} \leq s$ oder $\overline{b} = s + 1$ (d. h. b ist $+$) oder $\overline{b} = s + 2$ (d. h. b ist $-$), und $l \leq s$ ist. e ist also der Code einer Registermaschine M :

2.1.5 Lemma

$$\begin{aligned} RM(e) &\iff e = \ulcorner M \urcorner \text{ für eine Registermaschine } M \\ &\iff e \text{ ist } \ulcorner \text{Registermaschine} \urcorner \end{aligned}$$

Das Prädikat RM , Gödel-Nummer einer Register-Maschine zu sein, ist ebenso wie das Prädikat $Instr$ primitiv-rekursiv.

Der zweite Teil folgt aus den Lemmata 1.3.12, 15 und 22.

Damit ist der Begriff der Registermaschine arithmetisiert und als primitiv-rekursiv erkannt. Wir wenden uns der Arithmetisierung der Begriffe "Rechnung" und "Ergebnis" zu.

2.1.6 Definition

x ist ' e - Konfiguration':

$$Konf(e, x) : \iff (x)_0 \leq lh(e) \wedge x > 0.$$

Wenn e Gödelnummer einer Registermaschine M der Länge $lh(e) = s$ ist, bedeutet $Konf(e, x)$ daher $x = \langle x_0, \dots, x_N \rangle$, wobei

1. $x_0 \leq s$, also x_0 eine (eventuell die Stop-) Instruktionsnummer von M ist und
2. $N \geq lh(x) \div 1$ eine mögliche Registerzahl von M ist. In den Registern vom $lh(x)$ -ten ab steht die Zahl 0.

Nach den Ergebnissen von 1.3 gilt also:

2.1.7 Lemma

Das Prädikat $Konf$ ist primitiv-rekursiv. Ist e Gödel-Nummer einer Registermaschine M , so ist $Konf(e, x)$ gleichwertig damit, dass $x = \langle j, \mathbf{x} \rangle$ für eine M -Konfiguration (j, \mathbf{x}) ist.

Für $RM(e)$ und $j = (x)_0 < lh(e)$ ist nach Lemma 2.1.5 $(e)_j = \langle j, i, b, l \rangle$ der Code von $I_j \equiv j \ i \ b \ l$. Die Instruktion I_j , die auf die e -Konfiguration x "anzuwenden" ist, ist daher aus e und x primitiv-rekursiv ablesbar. Damit können wir auch die e -Folgekonfiguration von x primitiv-rekursiv ausrechnen. Wir verwenden dazu die Funktionen aus Definition 1.3.23.

2.1.8 Definition

der e -Folgekonfiguration von x , $FK(e, x)$

Es sei $s = lh(e)$, $j = (x)_0$ und $(e)_j = \langle z, i, b, l, \mathbf{z} \rangle$.

$$\begin{aligned} FK(e, x) &= 0, \text{ falls } \neg Konf(e, x) \vee \neg RM(e); \\ &= x, \text{ falls } Konf(e, x) \wedge RM(e) \wedge j \geq s. \end{aligned}$$

In den übrigen Fällen sei stets $Konf(e, x) \wedge RM(e) \wedge j < s$. Dann ist $(e)_j = \langle j, i, b, l \rangle$, also $z = j$ und $\mathbf{z} = \bar{0}$.

$$\begin{aligned} FK(e, x) &= x(b/0), & \text{falls } b \leq s \wedge p_i | x, \\ &= x(l/0), & \text{falls } b \leq s \wedge \neg p_i | x, \\ &= x(l/0) \cdot p_i, & \text{falls } b = s + 1, \\ &= x(l/0) \dot{:} p_i, & \text{falls } b \geq s + 2. \end{aligned}$$

2.1.9 Lemma

Die Funktion FK ist primitiv-rekursiv. Ist e Gödel-Nummer der Registermaschine M und ist $x = \langle j, \mathbf{x} \rangle$ eine e -Konfiguration, so ist $FK(e, x) = \langle j', \mathbf{x}' \rangle$ derart, dass (j', \mathbf{x}') die M -Folgekonfiguration von (j, \mathbf{x}) ist. Insbesondere gilt

$$RM(e) \wedge Konf(e, x) \rightarrow Konf(e, FK(e, x)).$$

Beweis. FK ist nach den Lemmata 1.3.6, 12, 13, 15, 25 und 2.1.5 und 7 primitiv-rekursiv. Gelte $RM(e)$ und $Konf(e, x)$. Dann ist $x = \langle j, x_1, \dots, x_N \rangle$ mit $j = (x)_0 \leq lh(e)$, und N ist Registerzahl der Registermaschine M , deren Gödel-Nummer e ist.

Wir schreiben wieder $s := lh(e)$. Ist $j = s$, so ist (j, \mathbf{x}) M -Endkonfiguration, die ihre eigene M -Folgekonfiguration ist, und tatsächlich ist dann $FK(e, x) =$

x .

Ist $j < s$ und $b = (e)_{j,2} \leq s$, so ist I_j eine Testinstruktion, die \mathbf{x} unverändert läßt und die b bzw. l aufruft, je nachdem für $i = (e)_{j,1}$ $(x)_i = x_i > 0$ oder $= 0$ ist. Die M -Folgeinstruktion ist dann (b, \mathbf{x}) bzw. (l, \mathbf{x}) , und tatsächlich ist $FK(e, x) = \langle b, \mathbf{x} \rangle$ bzw. $= \langle l, \mathbf{x} \rangle$, je nachdem $(x)_i > 0$, d. h. $p_i|x$, oder $(x)_i = 0$, d. h. $\neg p_i|x$, gilt.

Ist $j < s$ und $b = (e)_{j,2} > s$, so ist I_j für $b = s + 1$ eine Addier- und für $b = s + 2$ eine Subtrahier-Instruktion. Dann ist die M -Folgeinstruktion $(l, \mathbf{x}') = (l, \dots, x_i + 1, \dots)$ bzw. $= (l, \dots, x_i \div 1, \dots)$, und tatsächlich ist $FK(e, x) = \langle l, \mathbf{x} \rangle \cdot p_i$ bzw. $= \langle l, \mathbf{x} \rangle : p_i$, also in beiden Fällen $= \langle l, \mathbf{x}' \rangle$ (vgl. Def. 1.3.23).

Mit Lemma 1.1.3 folgt die Behauptung.

Für $e = \lceil M \rceil$ entstehen $\lceil e \text{ - Rechnungen} \rceil$ offenbar durch iterierte Anwendung von $FK(e, \cdot)$ auf eine $\lceil \text{Anfangskonfiguration} \rceil$. Uns interessieren spezielle e -Rechnungen: Die Eingabe ist ein vorgegebenes n -Tupel \mathbf{x} auf den Registern 2 bis $n + 1$, ergänzt um Nullen in den Registern 1 und $n + 2$ bis N , und die Rechnung hat ein Ergebnis. Eine e -Rechnung y mit diesen Eigenschaften (bezüglich des gegebenen \mathbf{x}) erfüllt Kleene's T -Prädikat:

2.1.10 Lemma

Das Kleene'sche T -Prädikat läßt sich wie folgt charakterisieren:

$$T_n(e, \mathbf{x}, y) \Leftrightarrow RM(e) \wedge (y)_0 = \langle 0, 0, \mathbf{x} \rangle \wedge (\forall i < lh(y) \div 1)(y)_{i+1} = FK(e, (y)_i) \\ \wedge (y)_{lh(y) \div 1, 0} = lh(e)$$

Ferner kann man die Auswertungsfunktion U allgemein definieren durch

$$U(y) = (y)_{lh(y) \div 1, 1}.$$

Das Kleene'sche T -Prädikat und die Auswertungsfunktion sind demnach primitiv-rekursiv.

Beweis. $T_n(e, \mathbf{x}, y)$ besagt: e ist $\lceil \text{Registernmaschine} \rceil$, also $RM(e)$, und y ist $\lceil e \text{-Rechnung} \rceil \langle (y)_0, \dots, (y)_T \rangle$ ($T = lh(y) \div 1$) zur Eingabe $(0, \mathbf{x})$, d. h. $(y)_0 = \lceil (0, 0, \mathbf{x}) \rceil = \langle 0, 0, \mathbf{x} \rangle$ und für alle $i < T$ ist $(y)_{i+1}$ $\lceil e \text{-Folgekonfiguration} \rceil$ von $(y)_i$, also $(y)_{i+1} = FK(e, (y)_i)$, und die Rechnung endet, d. h. $(y)_{T,0} = lh(e)$.

Bei solchen $\lceil e \text{-Rechnungen} \rceil$ y steht $U(y)$ im 1. Register der letzten $\lceil e \text{-Konfiguration} \rceil$ $(y)_T$, also $U(y) = (y)_{T,1}$, und das ist allgemein eine

primitiv-rekursive Festlegung von U .

Nach 1.3.12, 15, 18, 22 und 2.1.5 und 9 sind T_n und U wegen dieser Charakterisierungen primitiv-rekursiv. Damit ist das Lemma bewiesen.

2.1.11 Kleene's Normalform-Theorem

Es gibt eine 1-stellige primitiv-rekursive Funktion U und zu jedem $n \in \mathbb{N}$ ein $(n + 2)$ -stelliges primitiv-rekursives Prädikat T_n , so dass es zu jeder n -stellig partiell berechenbaren Funktion f eine natürliche Zahl e (nämlich die Gödel-Nummer der f berechnenden Registermaschine) gibt, so dass gilt:

$$(3) \quad f(\mathbf{x}) \simeq U(\mu y T_n(e, \mathbf{x}, y))$$

Der **Beweis** ist durch die obige Konstruktion bereits im wesentlichen geführt. Sei e eine 'Registermaschine', die f berechnet. Dann erfüllt f wegen (3) die angegebene partielle Gleichung, und nach 2.1.10 sind T_n und U primitiv-rekursiv.

Dieser Satz ist unser bisher wichtigstes Ergebnis. Er erfordert keinen besonders schwierigen Beweis, dagegen aber die wesentliche neue Idee der Arithmetisierung, die von Gödel 1931 stammt. Im Rest des Kapitels werden wir uns weitgehend mit den Konsequenzen dieses Satzes beschäftigen.

2.1.12 Definition

Die e -te n -stellige μ -partiell-rekursive Funktion $\{e\}^n$ ist gegeben durch

$$\{e\}^n(\mathbf{x}) \simeq U(\mu y T_n(e, \mathbf{x}, y))$$

mit den primitiv-rekursiven T_n und U aus 2.1.10. Die Zahl $e \in \mathbb{N}$ heißt **Gödel-Nummer** oder **Index** der Funktion $\{e\}^n$.

Jede Funktion, die einen Index hat, ist μ -partiell-rekursiv, weil T_n und U primitiv-rekursiv sind. Umgekehrt kann man Kleene's Normalform-Theorem auch so formulieren:

Jede partiell berechenbare Funktion hat einen Index

Wir ziehen einige Konsequenzen aus dem Normalform-Theorem.

2.1.13 Äquivalenz der Berechenbarkeitsbegriffe

Folgende Eigenschaften partieller Funktionen f sind äquivalent:

1. f ist partiell berechenbar;
2. f ist partiell normiert berechenbar;
3. f ist μ -partiell-rekursiv;
4. es gibt $e \in \mathbb{N}$, so dass f durch (3) definiert ist.

Beweis. Da T_n und U prim-rek und damit μ -rekursiv sind, folgt 3. aus 4. Nach Satz 1.2.19 folgt 2. aus 3., und nach Satz 1.1.14 folgt 1. aus 2. Der wesentliche Schritt, dass 4. aus 1. folgt, ist gerade die Aussage des Normalform-Theorems 2.1.11.

Die verschiedenen, jetzt als äquivalent erkannten Berechenbarkeitsbegriffe brauchen wir hiernach nicht mehr zu unterscheiden.

2.1.14 Definition

Eine partielle Funktion f heißt **partiell-rekursiv**, wenn f partiell (normiert) berechenbar bzw. μ -partiell-rekursiv ist. Ist eine solche Funktion f total, so heißt f **rekursiv**.

2.1.15 Aufzählungstheorem für partiell-rekursive Funktionen.

Zu jedem $n \in \mathbb{N}$ gibt es eine $(n + 1)$ -stellige partiell-rekursive Funktion g , so dass eine n -stellige partielle Funktion f genau dann partiell-rekursiv ist, wenn für ein $e \in \mathbb{N}$ gilt

$$f(\mathbf{x}) \simeq g(e, \mathbf{x}).$$

Beweis. Dies ist eine etwas pauschalere, weniger präzise Fassung des Normalform-Theorems. Man definiert explizit

$$g(e, \mathbf{x}) \simeq U(\mu y T_n(e, \mathbf{x}, y)).$$

g ist dann partiell-rekursiv. Dann ist jedes f mit $f(\mathbf{x}) \simeq g(e, \mathbf{x})$ (für ein e) auch partiell-rekursiv, etwa wegen 4. \Rightarrow 3. in 2.1.13; umgekehrt ist jedes partiell-rekursive f nach dem Normalform-Theorem so darstellbar.

Dieses Korollar zeigt, dass es eine "universelle" partiell-rekursive Funktion g gibt, die alle n -stelligen partiell-rekursiven Funktionen "aufzählt".

2.1.16 Korollar

Jede μ -rekursive Funktion besitzt eine Definition (einen Aufbau), in der der μ -Operator nur einmal im Normalfall angewandt wird.

Denn gilt (3) und ist f total, so gibt es zu jedem \mathbf{x} ein y mit $T_n(e, \mathbf{x}, y)$. Also wird hier der μ -Operator auf die charakteristische Funktion $\chi_{T,e}$ von $T_n(e, \cdot, \cdot)$, definiert durch

$$\chi_{T,e}(\mathbf{x}, y) := \begin{cases} 0 & \iff T_n(e, \mathbf{x}, y) \\ 1 & \iff \neg T_n(e, \mathbf{x}, y) \end{cases}$$

im Normalfall angewandt.

Bemerkung zur "Stabilität" des Normalform-Theorems. Als Funktionen berechnende Registermaschinen haben wir nur solche Registermaschinen angesehen, die bei einer Eingabe $0\mathbf{x}\bar{0}$, wenn überhaupt, ein Ergebnis $y_1\mathbf{x}\bar{0}$ liefern. Das war für den Beweis von Satz 1.2.19 zweckmäßig, aber es ist für das Normalform-Theorem 2.1.11 nicht nötig. Wir könnten ebenso gut jede Registermaschine als Funktionen-berechnend ansehen, und im Fall $0\mathbf{x}\bar{0} \Rightarrow y_1\mathbf{z}$ die Zahl y_1 als Wert an der Stelle \mathbf{x} ansehen. In dieser allgemeineren, einfacheren Form haben wir auch das T -Prädikat in 2.1.3 definiert: Der Unterschied ist für das Normalform-Theorem offenbar belanglos. - Das Normalform-Theorem läßt sich im Prinzip genau wie hier auch für jeden einigermaßen ähnlich festgelegten Berechenbarkeitsbegriff beweisen.

2.2 Rekursive und rekursiv aufzählbare Prädikate

Dem intuitiven Berechenbarkeitsbegriff für Funktionen entspricht der intuitive Entscheidbarkeitsbegriff für Prädikate. Ein zahlentheoretisches Prädikat $P \subseteq \mathbb{N}^n$ ist im intuitiven Sinne entscheidbar, wenn es einen Algorithmus gibt, der von jedem n -Tupel \mathbf{x} entscheidet, ob P auf \mathbf{x} zutrifft oder nicht. Davon zu unterscheiden ist der intuitive Begriff der Aufzählbarkeit. Ein Prädikat P ist im intuitiven Sinne aufzählbar, wenn es einen Algorithmus gibt, der mit der Zeit alle n -Tupel \mathbf{x} produziert oder aufzählt, auf die P zutrifft. Wenn wir die Church'sche These zugrunde legen, können wir diese inhaltlichen Begriffe mathematisch wie folgt präzisieren:

2.2.1 Definition

Ein Prädikat P heißt **rekursiv** (kurz: **rek**) oder **entscheidbar**, wenn seine charakteristische Funktion χ_P rekursiv ist. P heißt **rekursiv aufzählbar** (kurz: **r.a.**), wenn $\{\mathbf{x} \mid P(\mathbf{x})\}$ die Definitionsmenge einer partiell-rekursiven Funktion ist. (1-stellige Prädikate heißen auch **Mengen**.)

Wir können P nicht als r.a. definieren, wenn χ_P partiell-rekursiv ist, weil charakteristische Funktionen hier als total definiert sind (vgl. Def. 1.3.9). Es gilt aber:

2.2.2 Lemma

P ist genau dann rekursiv aufzählbar, wenn

$$(1) \quad P(\mathbf{x}) \iff f(\mathbf{x}) = 0$$

für eine partiell-rekursive Funktion f gilt.

Beweis.

- a) Ist $P = \text{dom } f$, so ist $P = \text{dom}(0 \cdot f)$, also $P(\mathbf{x}) \iff 0 \cdot f(\mathbf{x}) = 0$.
- b) Ist f partiell-rekursiv, so auch $g : \mathbf{x} \mapsto \mu y (f(\mathbf{x}) = 0)$ (wobei y nicht im Tupel \mathbf{x} vorkommt). Dann ist aber $\text{dom } g$ die Nullstellenmenge von f .

Damit ist die Analogie optimal hergestellt:

- 1. P ist prim-**rek** \iff es gibt ein prim-**rek** f mit (1);
- 2. P ist **rekursiv** \iff es gibt ein **rekursives** f mit (1);
- 3. P ist r.a. \iff es gibt ein partiell-**rekursives** f mit (1).

2.2.3 Korollar

Jedes rekursive Prädikat ist r.a.

Dass die Umkehrung hiervon nicht gilt, werden wir aus folgender anderen Charakterisierung der r.a. Prädikate ableiten, die für Prädikate teilweise dem entspricht, was im Aufzählungstheorem 2.1.15 über partielle Funktionen ausgesagt ist.

2.2.4 Aufzählungs-Theorem von Kleene

Folgende Eigenschaften von n -stelligen Prädikaten P sind äquivalent:

1. P ist rekursiv-aufzählbar;
2. P ist Projektion eines rekursiven Prädikates, d. h. es gibt ein $(n + 1)$ -stelliges rekursives Prädikat Q mit

$$(2) \quad P(\mathbf{x}) \iff \exists y Q(\mathbf{x}, y);$$

3. P ist Projektion eines primitiv-rekursiven Prädikates, d. h. es gibt ein $(n + 1)$ -stelliges primitiv-rekursives Prädikat Q mit (2);
4. es gibt eine Zahl e mit

$$(3) \quad P(\mathbf{x}) \iff \exists y T_n(e, \mathbf{x}, y).$$

Beweis.

1. \Rightarrow 4. Es sei $P = \text{dom } f$, und e sei ein Index von f . Dann gilt (3).
4. \Rightarrow 3., da T_n prim-rek ist, und
3. \Rightarrow 2. sind klar.
2. \Rightarrow 1. Gilt 2., so ist $f : \mathbf{x} \mapsto \mu y Q(\mathbf{x}, y)$ partiell-rekursiv mit $\text{dom } f = P$.

Der wesentliche Schritt 1. \iff 4. dieses Beweises benutzt also das Normalform-Theorem. Durch (3) werden alle n -stelligen r.a. Prädikate durch alle $e \in \mathbb{N}$ "aufgezählt".

Wir verwenden diesen Satz zunächst zu einer Charakterisierung der rekursiven Prädikate innerhalb der r.a. Prädikate.

2.2.5 Satz von Post

Ein Prädikat P ist genau dann rekursiv, wenn sowohl P als auch $\neg P$ rekursiv aufzählbar sind.

Beweis.

- a) Sei P rekursiv. Dann ist nach Lemma 1.3.12 $\neg P$ prim-rek in P , also rekursiv. Nach Korollar 2.2.3 sind dann P und $\neg P$ r.a.

b) Seien P und $\neg P$ r.a. Nach Satz 2.2.4 gibt es rekursive Q und R mit (2) und

$$\neg P(\mathbf{x}) \iff \exists y R(\mathbf{x}, y).$$

Wegen $P(\mathbf{x}) \vee \neg P(\mathbf{x})$ gibt es zu jedem \mathbf{x} daher ein y mit

$$Q(\mathbf{x}, y) \vee R(\mathbf{x}, y),$$

und nach Lemma 1.3.12 ist das Prädikat $Q \vee R$ wieder rekursiv. Also können wir auf $Q \vee R$ den μ -Operator im Normalfall anwenden, und die Funktion

$$f : \mathbf{x} \mapsto \mu y (Q(\mathbf{x}, y) \vee R(\mathbf{x}, y))$$

ist (total) rekursiv. Also gilt stets $Q(\mathbf{x}, f(\mathbf{x})) \vee R(\mathbf{x}, f(\mathbf{x}))$, und da

$$Q(\mathbf{x}, f(\mathbf{x})) \Rightarrow \exists y Q(\mathbf{x}, y) \Rightarrow P(\mathbf{x}) \quad \text{und}$$

$$R(\mathbf{x}, f(\mathbf{x})) \Rightarrow \exists y R(\mathbf{x}, y) \Rightarrow \neg P(\mathbf{x}), \quad \text{folgt}$$

$$P(\mathbf{x}) \iff Q(\mathbf{x}, f(\mathbf{x})),$$

und deshalb ist P rekursiv nach Lemma 1.3.2.

Während das Aufzählungs-Theorem verschiedene Charakterisierungen der r.a. Prädikate angibt, liefert der Satz von Post eine Charakterisierung der rekursiven Prädikate mit Hilfe der r.a. Prädikate. Eine ebenso einfache (nämlich vom Normalform-Theorem unabhängige) wie grundsätzlich wichtige Bemerkung ist nun, dass sich auch die partiell-rekursiven Funktionen durch r.a. Prädikate charakterisieren lassen: Sie sind genau die Funktionen, deren Graph r.a. ist.

2.2.6 Lemma

Eine Funktion f ist genau dann partiell-rekursiv, wenn ihr Graph, definiert durch

$$(4) \quad \text{Graph}(f)(\mathbf{x}, y) \iff f(\mathbf{x}) = y,$$

rekursiv aufzählbar ist.

Beweis.

a) Ist $\text{Graph}(f)$ r.a., so gibt es nach 2.2.4 ein rekursives Prädikat Q mit

$$\text{Graph}(f)(\mathbf{x}, y) \iff \exists z Q(\mathbf{x}, y, z).$$

Zu jedem \mathbf{x} gibt es höchstens ein y mit $\text{Graph}(f)(\mathbf{x}, y)$. Also ist

$$f(\mathbf{x}) \simeq (\mu z Q(\mathbf{x}, (z)_0, (z)_1))_0,$$

und f ist partiell rekursiv.

b) Ist f partiell rekursiv, so ist die Funktion g mit

$$g(\mathbf{x}, y) \simeq |f(\mathbf{x}) - y|$$

primitiv-rekursiv in f und damit (nach Lemma 1.2.18) partiell-rekursiv. Dann ist wegen (4)

$$\text{Graph}(f)(\mathbf{x}, y) \iff g(\mathbf{x}, y) = |f(\mathbf{x}) - y| = 0,$$

und $\text{Graph}(f)$ ist nach Lemma 2.2.2 r.a.

Die Wahl der Bezeichnung "rekursiv aufzählbar" weckt die Vorstellung, dass ein r.a. Prädikat P durch eine (partiell-)rekursive Funktion f aufgezählt wird, dass also $P = \text{Im}(f)$ ist. Um das zu zeigen, schicken wir folgendes Lemma voraus:

2.2.7 Lemma

Ist Q r.a., so ist die durch (2) definierte Projektion P von Q ebenfalls r.a.

Beweis. o. E. sei Q nicht 0-stellig. Nach 2.2.4 gibt es ein rekursives R mit

$$Q(\mathbf{x}, y) \iff \exists z R(\mathbf{x}, y, z).$$

Dann ist auch $R(\mathbf{x}, (y)_0, (y)_1)$ rekursiv, so dass wegen

$$P(\mathbf{x}) \iff \exists y \exists z R(\mathbf{x}, y, z) \iff \exists y R(\mathbf{x}, (y)_0, (y)_1)$$

nach dem Aufzählungs-Theorem 2.2.4 auch P r.a. ist.

2.2.8 Satz

Folgende Eigenschaften von Mengen (1-stelligen Prädikaten) M sind äquivalent:

1. M ist rekursiv aufzählbar;
2. M ist leer oder Bild einer primitiv-rekursiven Funktion, d. h. es gibt ein primitiv-rekursives f mit

$$(5) \quad M = \text{Im}(f) := \{y | \exists x f(x) = y\};$$

3. M ist leer oder Bild einer rekursiven Funktion;
4. M ist Bild einer partiell-rekursiven Funktion.

Beweis.

1. \Rightarrow 2. Sei M r.a. und etwa $a \in M$, also $M \neq \emptyset$. Nach Satz 2.2.4 gibt es ein 2-stelliges prim-rek Q mit

$$x \in M \iff \exists y Q(x, y).$$

Wir definieren eine einstellige Funktion f durch

$$f(z) = a, \text{ falls } \neg Q((z)_0, (z)_1), \quad f(z) = (z)_0, \text{ falls } Q((z)_0, (z)_1).$$

Nach 1.3.6 ist f prim-rek in Q , also prim-rek, und es ist

$$x \in \text{Im}(f) \iff x = a \vee \exists z Q(x, (z)_1) \iff x \in M.$$

2. \Rightarrow 3. und
3. \Rightarrow 4. sind klar. \emptyset ist Bild der leeren Funktion, die partiell-rekursiv ist.
4. \Rightarrow 1. Ist f partiell-rekursiv, so ist $\text{Graph}(f)$ nach Lemma 2.2.6 r.a. Dann ist auch

$$\text{Im}(f) = \{y \mid \exists x \text{Graph}(f)(x, y)\}$$

r.a. nach Lemma 2.2.7.

Damit ist der Satz bewiesen.

Damit haben wir einige Charakterisierungen der rekursiv aufzählbaren (und der rekursiven) Prädikate kennengelernt. Wir zeigen jetzt, dass nicht alle r.a. Prädikate rekursiv sind.

Wir kehren zum Aufzählungs-Theorem 2.2.4 zurück. Genau die r.a. Mengen werden mit Hilfe des Prädikates T_1 aufgezählt. Folgende Bezeichnungen sind üblich:

2.2.9 Definition

Das T -Prädikat T_1 bezeichnet man mit T . Die e -te r.a. Menge bezeichnet man mit W_e , also

$$x \in W_e \iff \exists y T(e, x, y).$$

Ferner sei

$$K = \{x \mid x \in W_x\}.$$

Bemerkung. W_e ist per Definition der Definitionsbereich der e -ten 1-stelligen partiell-rekursiven Funktion $\{e\}^1$. Ferner ist

$$x \in K \iff \exists y T(x, x, y),$$

so dass nach 2.2.4 K auch r.a. ist. Schon der Ausdruck $x \in W_x$ in der Definition von K , in dem x sowohl als schlichte Zahl als auch als Gödel-Nummer einer r.a. Menge auftritt, suggeriert aber, dass man auf K den klassischen Cantorschen Diagonalschluss anwenden kann.

2.2.10 Satz

$\mathbb{N} - K$ ist nicht rekursiv aufzählbar; K ist rekursiv aufzählbar, aber nicht rekursiv.

Beweis. Angenommen, $\mathbb{N} - K$ wäre r.a. Dann gäbe es nach dem Aufzählungs-Theorem ein e , so dass $\mathbb{N} - K = W_e$, also für alle x

$$x \notin W_x \iff x \notin K \iff x \in W_e$$

wäre. Das ergibt für $x = e$ den Widerspruch $e \notin W_e \iff e \in W_e$.

K selbst ist, wie oben bemerkt, r.a. K ist aber nicht rekursiv, weil sonst nach Lemma 1.3.12 sein Komplement $\mathbb{N} - K$ rekursiv sein müßte.

Es gibt also "angeborene" r.a. Mengen, die nicht rekursiv sind, z. B. K . Wenn man die Church'sche These zugrunde legt, bedeutet dies: Zwar gibt es einen Algorithmus, der nach und nach jedes Element von K liefert oder aufzählt, es gibt aber keinen Algorithmus, der von einer vorgelegten Zahl entscheidet, ob diese Zahl zu K gehört oder nicht. Falls $x \in K$ ist, wird x irgendwann von dem aufzählenden Algorithmus als Element von K nachgewiesen; falls $x \notin K$ ist, geschieht dies nie, aber zu keinem Zeitpunkt kann man sicher sein, ob x nicht später doch noch als Element von K aufgezählt wird.

Nach Satz 2.2.5 kann keine r.a. Menge W_e das Komplement von K ausschöpfen. Wenn $W_e \cap K = \emptyset$ ist, gibt es also ein x_e außerhalb $W_e \cup K$. In diesem Fall gilt dies sogar für $x_e = e$. Denn da nach Definition von K

$$e \in K \iff e \in W_e$$

gilt, ist

$$e \in W_e \cup K \iff e \in W_e \cap K.$$

Aus $W_e \cap K = \emptyset$ folgt also $e \notin W_e \cup K$. Solche Mengen wie K heißen kreativ.

2.2.11 Definition

Eine r.a. Menge M heißt **kreativ**, wenn es eine rekursive Funktion f gibt, so dass aus $W_e \cap M = \emptyset$ stets $f(e) \notin W_e \cup M$ folgt. f heißt dann **produktive Funktion** von M .

Wir haben bewiesen:

2.2.12 Satz

K ist kreativ, mit der Identität U_1^1 als produktiver Funktion.

Bemerkung. Kreative Mengen sind stets r.a. und nicht rekursiv. Insofern ist Satz 2.2.12 eine Verschärfung von Satz 2.2.10.

Die Menge K ist ein sehr wichtiges Beispiel einer "krummen" Menge. Ihre Definition $K = \{x \mid \exists y T(x, x, y)\}$ schließt sich unmittelbar an die Definition des T -Prädikats an, und Satz 2.2.10 ist im Grunde eine leichte Konsequenz des Normalform-Theorems. Die Eigenschaften von K werden wir im nächsten Kapitel noch ausnutzen. Hier wollen wir zeigen, dass K auch Definitionsmenge einer partiell-rekursiven Funktion ist, die nicht Beschränkung einer total rekursiven Funktion ist.

2.2.13 Satz

Die 1-stellige partiell-rekursive Funktion f mit

$$f(x) \simeq U(\mu y T(x, x, y))$$

läßt sich nicht zu einer total-rekursiven Funktion fortsetzen.

Beweis. Ist f^+ 1-stellig und rekursiv, dann ist auch $suc \circ f^+$ rekursiv. Nach dem Normalform-Theorem 2.1.11 gibt es dann eine Zahl e mit

$$(suc \circ f^+)(x) = U(\mu y T(e, x, y))$$

für alle x . Für $x = e$ ist dann

$$f^+(e) + 1 = (suc \circ f^+)(e) = U(\mu y T(e, e, y)) = f(e),$$

so dass $e \in dom f$ ist, und f^+ ist keine Fortsetzung von f .

Bemerkung. Man kann als f auch die Funktion mit

$$f(x) \simeq \mu y T(x, x, y)$$

wählen. Ist dann e eine Gödel-Nummer eines totalen f^+ , so ist

$$f^+(e) = U(\mu y T(e, e, y)) = U(f(e)) < f(e),$$

und f^+ ist keine Fortsetzung von f . In beiden Fällen ist $K = \text{dom } f$. Wichtig ist die Voraussetzung, dass f^+ total ist; sonst kommt kein Widerspruch zustande, weil dann nur die Gödel-Nummer von f^+ nicht zu $\text{dom } f^+$ gehört.

Mit demselben Diagonalschluss, aber ohne Normalform-Theorem und seine Konsequenzen, zeigt man, dass sich die (total-)rekursiven Funktionen und Prädikate nicht von einer rekursiven Funktion aufzählen lassen, im Gegensatz zur Lage bei den partiell-rekursiven Funktionen (vgl. 2.1.15) und den r.a. Prädikaten (vgl. 2.2.4).

2.2.14 Lemma (Cantor)

Gilt für alle $x, y \in \mathbb{N}$

$$(6) \quad g_x(y) = g(x, y),$$

so ist die Funktion $f : y \mapsto g(y, y) + 1$ von allen Funktionen g_x verschieden.

Da der Übergang von g zu diesem f eine primitiv-rekursive Operation ist, folgt:

Korollar 1. Es gibt kein primitiv-rekursives g , so dass jede 1-stellige primitiv-rekursive Funktion eine Funktion g_x mit (6) ist.

Korollar 2. Es gibt kein rekursives g , so dass jede 1-stellige rekursive Funktion eine Funktion g_x mit (6) ist.

Diese an sich triviale Bemerkung, die auch mit der Rekursionstheorie nichts zu tun hat, läßt sich wegen Satz 2.1.11 verschärfen zu:

2.2.15 Satz

Die Menge der Gödel-Nummern rekursiver Funktionen

$$R = \{e \mid \{e\}^1 \text{ ist total}\} = \{e \mid \forall x \exists y T(e, x, y)\}$$

ist nicht rekursiv aufzählbar.

Beweis. Angenommen, R wäre r.a. Dann gäbe es nach Satz 2.2.8 eine (primitiv-)rekursive Funktion f , deren Bild R wäre. Für jedes z wäre dann die Funktion

$$\{f(z)\}^1 : x \mapsto U(\mu y T(f(z), x, y))$$

total rekursiv, also wäre auch die partiell-rekursive Funktion

$$x \mapsto U(\mu y T(f(x), x, y)) + 1$$

total rekursiv, so dass die Gödel-Nummer e dieser Funktion im Bild von f läge, etwa $e = f(z_0)$. Für alle $x \in \mathbb{N}$ wäre also

$$U(\mu y T(f(x), x, y)) + 1 = U(\mu y T(f(z_0), x, y)).$$

Für $x = z_0$ ergibt dies einen Widerspruch.

2.2.16 Aufgabe

Zeigen Sie: Die Menge der Gödel-Nummern rekursiver Mengen

$$Rek = \{e \mid W_e \text{ ist rekursiv} \}$$

ist nicht r. a.

2.3 S-m-n- und Rekursions-Theorem und der Satz von Rice

In 2.1 haben wir jeder Registermaschine M eine natürliche Zahl, ihre Gödel-Nummer $\ulcorner M \urcorner$, zugeordnet, und in 1.1 haben wir die Verkettung und die Iteration von Registermaschinen untersucht. Wie wirkt sich z. B. die Verkettung von zwei Registermaschinen auf ihre Gödel-Nummer aus? Das ist eine erste Frage zu einer "Arithmetik der Gödel-Nummern", die uns weiter in die Rekursionstheorie hineinführt.

2.3.1 Lemma

Es gibt eine primitiv-rekursive Funktion $kett$, die die Verkettung von Registermaschinen beschreibt: Sind M_1, M_2 zwei Registermaschinen, so ist

$$\ulcorner M_1 M_2 \urcorner = kett(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner).$$

Beweis. Es sei $\ulcorner M_1 \urcorner = c$, $\ulcorner M_2 \urcorner = d$ und $\ulcorner M_1 M_2 \urcorner = e$. Dann sind $s = lh(c)$ und $s' = lh(d)$ die Längen von M_1 bzw. M_2 . Wir wollen zeigen, dass e primitiv-rekursiv von c und d abhängt, was nach der Definition der Verkettung in 1.1 auch ohne Beweis einleuchtet. Leider ist nicht $e = c * d$ (vgl. Def. 1.3.23), weil $M_1 M_2$ nicht einfach die Instruktionenfolge

$$\begin{array}{l} M_1 \\ M_2 \end{array}, \text{ sondern } \begin{array}{l} M_1 \\ s + M_2 \end{array}$$

ist, wobei die Befehle $+$ und $-$ einheitlich durch $s + s' + 1$ bzw. $s + s' + 2$ wiedergegeben werden.

Wir untersuchen, in welcher Form die Gödel-Nummer x einer Instruktion $I_j \equiv j \ i \ b \ l$ aus M_1 bzw. M_2 in $e = \lceil M_1 M_2 \rceil$ eingeht. Ist I_j aus M_1 , so tritt I_j unverändert in $M_1 M_2$ auf, aber die Gödel-Nummer $x = \langle j, i, b, l \rangle$ bleibt nur für $b \leq s$ unverändert. Die Befehle $+$ und $-$ dagegen werden nicht mehr durch $b = (x)_2 = s + 1$ bzw. $= s + 2$ wiedergegeben, sondern durch um s' vergrößerte Zahlen, wie oben angegeben. Diese Verschiebung von x wird also durch folgende primitiv-rekursive Funktion v ausgedrückt:

$$v(x, s, s') = \begin{cases} x, & \text{falls } (x)_2 \leq s, \\ x \cdot 5^{s'}, & \text{falls } (x)_2 > s. \end{cases}$$

Ist I_j aus M_2 , so werden alle Instruktionsnummern (einschließlich der Nummern für die Befehle $+$ und $-$) um s erhöht. x geht über in $\langle j+s, i, b+s, l+s \rangle$, also primitiv-rekursiv gemäß

$$x \mapsto x \cdot 2^s \cdot 5^s \cdot 7^s = x \cdot 70^s$$

Die Verschiebung v ist auf alle $(c)_j$ mit $j < s$, die Verschiebung $x \mapsto x \cdot 70^s$ auf alle $(d)_j$ mit $j < s'$ anzuwenden, und diese so "korrigierten" Folgen c und d sind dann mit der Verkettungsfunktion $*$ aus Def. 1.3.23 zu verknüpfen. Mit $s = lh(c)$ und $s' = lh(d)$ ist also

$$\begin{aligned} e = kett(c, d) &= \langle v((c)_0, s, s'), \dots, v((c)_{s-1}, s, s') \rangle * \langle (d)_0 \cdot 70^s, \dots, (d)_{s'-1} \cdot 70^s \rangle \\ &= \prod_{j < lh(c)} p_j^{v((c)_j, lh(c), lh(d))} \cdot \prod_{j < lh(d)} p_{lh(c)+j}^{(d)_j \cdot 70^{lh(c)}}. \end{aligned}$$

Diese Funktion $kett$ ist also primitiv rekursiv und erfüllt die Behauptung des Lemmas. Die Darstellung von $kett$ durch die mittlere Zeile sollte auch ebenso verständlich sein wie der Begriff der Verkettung von Registermaschinen in Def. 1.1.6

Abkürzung. Statt $kett(c, d)$ schreiben wir $c \infty d$.

2.3.2 Korollar

Sind e_1, e_2, e_3 Gödel-Nummern von Registermaschinen, so ist

$$(e_1 \infty e_2) \infty e_3 = e_1 \infty (e_2 \infty e_3).$$

Dies folgt unmittelbar aus Lemma 2.3.1 und 1.1.7.

2.3.3 Lemma

Es gibt eine primitiv-rekursive Funktion $druck$, so dass für alle y

$$druck(y) = \ulcorner a_1^y \urcorner$$

die Gödel-Nummer einer 1-Registermaschine ist, die ins leere 1. Register die Zahl y druckt.

Beweis. Es ist $\ulcorner a_1 \urcorner = \langle \langle 0, 1, 2, 1 \rangle \rangle = 2^{525}$. Die durch

$$\begin{aligned} druck(0) &= \ulcorner \emptyset \urcorner = 1 \\ druck(y+1) &= druck(y) \circ \ulcorner a_1 \urcorner \end{aligned}$$

definierte Funktion $druck$ ist nach Lemma 1.3.3 primitiv-rekursiv und erfüllt die Behauptung, wie man durch Induktion nach y sieht.

Diese einfachen Ergebnisse wenden wir nun auf partiell-rekursive Funktionen an.

Wenn man eine $(m+n)$ -stellige partiell-rekursive Funktion nur in Abhängigkeit ihrer letzten n Argumente betrachtet und die ersten m Argumente festlegt, erhält man selbstverständlich stets wieder eine n -stellige partiell-rekursive Funktion, die nach dem Normalform-Theorem auch wieder einen Index hat: Zu jedem e und m -Tupel \mathbf{y} gibt es einen Index e' , so dass

$$\{e'\}^n(\mathbf{x}) \simeq \{e\}^{m+n}(\mathbf{y}, \mathbf{x}).$$

ist. Jedenfalls mit dem Auswahlaxiom folgt dann, dass es eine $(m+1)$ -stellige Funktion S_n^m gibt, die jedem e und \mathbf{y} ein solches e' zuordnet. Dann ist

$$\{S_n^m(e, \mathbf{y})\}^n(\mathbf{x}) \simeq \{e\}^{m+n}(\mathbf{y}, \mathbf{x}).$$

Die Arithmetik der Gödel-Nummern der Registermaschinen liefert aber das wesentlich schärfere Ergebnis, dass man eine solche Funktion S_n^m auch primitiv-rekursiv angeben kann. Wir beweisen das zunächst für $m=1$.

2.3.4 Lemma

Es gibt zu jedem $n \in \mathbb{N}$ eine 2-stellige primitiv-rekursive Funktion S_n , die für alle e und y die partielle Gleichung

$$\{S_n(e, y)\}^n(\mathbf{x}) \simeq \{e\}^{1+n}(y, \mathbf{x})$$

erfüllt.

Beweis. Sei g die $(n + 2)$ -stellige partielle Funktion mit

$$g(\mathbf{x}, y, e) \simeq U(\mu z T_{n+1}(e, y, \mathbf{x}, z)) \simeq \{e\}^{n+1}(y, \mathbf{x}).$$

g ist partiell rekursiv; es gibt also eine Registermaschine M , die g berechnet. Dann gilt

$$\begin{array}{lll} a_1^y K_{1\ n+2} a_1^e K_{1\ n+3} & : & 0\mathbf{x}00\bar{0} \Rightarrow 0\mathbf{x}y e\bar{0} \\ M & : & \Rightarrow \{e\}(y, \mathbf{x})\mathbf{x}y e\bar{0} \\ L_{n+2} L_{n+3} & : & \Rightarrow \{e\}(y, \mathbf{x})\mathbf{x}00\bar{0} \end{array}$$

Die Registermaschine $a_1^y K_{1\ n+2} a_1^e K_{1\ n+3} M L_{n+2} L_{n+3}$ berechnet insgesamt zu festem e und y die Funktion $\{S_n(e, y)\}^n$ und hat die Gödel-Nummer

$$S_n(e, y) := \text{druck}(y) \ulcorner K_{1\ n+2} \urcorner \infty \text{druck}(e) \ulcorner K_{1\ n+3} M L_{n+2} L_{n+3} \urcorner,$$

und das ist nach 2.3.1 und 2.3.3 eine primitiv-rekursive Definition von S_n . Damit ist das Lemma bewiesen. Wir verallgemeinern dieses Ergebnis auf Argumente-Tupel \mathbf{y} .

2.3.5 S-m-n-Theorem

Zu jedem m und n gibt es eine $(m + 1)$ -stellige primitiv-rekursive Funktion S_n^m , die für alle e und m -Tupel \mathbf{y} die partielle Gleichung

$$\{S_n^m(e, \mathbf{y})\}^n(\mathbf{x}) \simeq \{e\}^{m+n}(\mathbf{y}, \mathbf{x})$$

erfüllt.

Beweis. Der Satz ergibt sich durch Iteration von Lemma 2.3.4. Wir induzieren also nach m .

1. $S_n^0 = U_1^1$ ist primitiv-rekursiv.
2. S_n^{m+1} sei definiert durch

$$S_n^{m+1}(e, \mathbf{y}, z) = S_n(S_{n+1}^m(e, \mathbf{y}), z).$$

Dann ist mit S_{n+1}^m auch S_n^{m+1} primitiv-rekursiv, und für alle e, \mathbf{y}, z gilt nach Lemma 2.3.4 und Induktionsvoraussetzung

$$\begin{aligned} \{S_n^{m+1}(e, \mathbf{y}, z)\}^n(\mathbf{x}) &\simeq \{S_n(S_{n+1}^m(e, \mathbf{y}), z)\}^n(\mathbf{x}) \\ &\simeq \{S_{n+1}^m(e, \mathbf{y})\}^{1+n}(z, \mathbf{x}) \\ &\simeq \{e\}^{m+1+n}(\mathbf{y}, z, \mathbf{x}). \end{aligned}$$

Damit ist der Satz bewiesen. Er hat zahlreiche Anwendungen.

2.3.6 Korollar

Ist f $(m + n)$ -stellig partiell-rekursiv, so gibt es ein m -stelliges primitiv-rekursives g , so dass für alle m -Tupel \mathbf{y} gilt

$$f(\mathbf{y}, \mathbf{x}) \simeq \{g(\mathbf{y})\}^n(\mathbf{x}).$$

Denn ist $f = \{e\}^{m+n}$, so ist $g(\mathbf{y}) = S_n^m(e, \mathbf{y})$.

2.3.7 Korollar

$$\exists z T_{m+n}(e, \mathbf{y}, \mathbf{x}, z) \leftrightarrow \exists z T_n(S_n^m(e, \mathbf{y}), \mathbf{x}, z).$$

Denn für jedes \mathbf{y} haben wegen des S-m-n-Theorems die Funktionen $\{S_n^m(e, \mathbf{y})\}$ und $\mathbf{x} \mapsto \{e\}(\mathbf{y}, \mathbf{x})$ denselben Definitionsbereich, und das ist die Behauptung.

2.3.8 Korollar

Ist R $(m + n)$ -stellig r.a., so gibt es ein m -stelliges primitiv-rekursives g , so dass für alle \mathbf{y} gilt

$$R(\mathbf{y}, \mathbf{x}) \leftrightarrow \exists z T_n(g(\mathbf{y}), \mathbf{x}, z)$$

Denn nach dem Aufzählungs-Theorem 2.2.4 gibt es ein e mit

$$R(\mathbf{y}, \mathbf{x}) \leftrightarrow \exists z T_{m+n}(e, \mathbf{y}, \mathbf{x}, z),$$

woraus mit 2.3.7 die Behauptung folgt.

2.3.9 Korollar

Ist R 2-stellig r.a., so gibt es ein primitiv-rekursives g , so dass für alle y gilt

$$\{x | R(y, x)\} = W_{g(y)}.$$

Dies ist der Spezialfall $m = n = 1$ von 2.3.8.

Eine weniger unmittelbare Folgerung aus dem S-m-n-Theorem ist folgendes:

2.3.10 Korollar

Es gibt primitiv-rekursive Funktionen un und int , für die stets gilt

$$\begin{aligned} W_d \cup W_e &= W_{un(d,e)} & \text{und} \\ W_d \cap W_e &= W_{int(d,e)}. \end{aligned}$$

Beweis. $x \in W_d \cup W_e$

$$\begin{aligned}
&\iff \exists y T(d, x, y) \vee \exists y T(e, x, y) \\
&\iff \exists y (T(d, x, y) \vee T(e, x, y)) \\
&\iff R(d, e, x) \text{ f\u00fcr ein r.a. } R \text{ nach Lemma 1.3.12, Satz 2.2.4} \\
&\iff \exists y T(un(d, e), x, y) \text{ f\u00fcr eine primitiv-rekursive Funktion } un \\
&\quad \text{nach Korollar 2.3.8 mit } m = 2, n = 1. \text{ Ebenso ist} \\
&\quad x \in W_d \cap W_e \\
&\iff \exists y T(d, x, y) \wedge \exists z T(e, x, z) \\
&\iff \exists y \exists z (T(d, x, y) \wedge T(e, x, z)) \\
&\iff R(d, e, x) \text{ f\u00fcr ein r.a. } R \text{ nach Lemma 1.3.12, 2.2.7, Satz 2.2.4} \\
&\iff \exists y T(int(d, e), x, y) \text{ f\u00fcr eine primitiv-rekursive Funktion } int \\
&\quad \text{nach Korollar 2.3.8 mit } m = 2, n = 1.
\end{aligned}$$

Wir kehren zu Korollar 2.3.6. zur\u00fcck. Im Fall $m = 1$ sagt es, dass es zu jeder $(n + 1)$ -stelligen partiell-rekursiven Funktion f ein primitiv-rekursives g gibt mit $f(y, \mathbf{x}) \simeq \{g(y)\}(\mathbf{x})$. Wir wollen nun zeigen, dass diese Funktion g einen Fixpunkt besitzt: F\u00fcr eine passende Zahl e ist $g(e) = e$, also $f(e, \mathbf{x}) \simeq \{e\}(\mathbf{x})$.

2.3.11 Rekursions-Theorem

Zu jeder $(n + 1)$ -stelligen partiell-rekursiven Funktion f gibt es eine Zahl e , so dass

$$f(e, \mathbf{x}) \simeq \{e\}^n(\mathbf{x}).$$

Beweis. Nach dem Normalform-Theorem 2.1.11 gibt es eine Zahl d mit

$$\{d\}(u, \mathbf{x}) \simeq f(S_n(u, u), \mathbf{x}).$$

Man setzt $e = S_n(d, d)$ und rechnet aus

$$\{e\}(\mathbf{x}) \simeq \{S_n(d, d)\}(\mathbf{x}) \simeq \{d\}^{n+1}(d, \mathbf{x})$$

nach dem S-m-n-Theorem bzw. Lemma 2.3.4

$$\simeq f(S_n(d, d), \mathbf{x}) \simeq f(e, \mathbf{x})$$

nach Wahl von d und e .

Man kann das Rekursions-Theorem so auffassen, dass man zu jeder partiell-rekursiven Funktion f eine partiell-rekursive Funktion $\{e\}$ definieren kann unter R\u00fcckgriff auf f und eine G\u00f6del-Nummer e von $\{e\}$ selbst. Das impliziert insbesondere, dass jede durch eine Gleichung syntaktisch definierte partielle Funktion partiell-rekursiv ist.

2.3.12 Korollar

$F(f, \mathbf{x})$ sei ein Ausdruck (Term), in dem neben dem Funktionszeichen f und den Variablen \mathbf{x} beliebige partiell-rekursive Funktionen auftreten. Dann gibt es ein partiell-rekursives f , das der Gleichung genügt:

$$f(\mathbf{x}) \simeq F(f, \mathbf{x}).$$

Beweis. Wir können ein partiell-rekursives g durch

$$g(e, \mathbf{x}) \simeq F(\{e\}, \mathbf{x})$$

explizit definieren, wenn g in F nicht auftritt. Dabei steht rechts $\{e\}(t_1, \dots, t_n)$ jeweils für $U(\mu y T_n(e, t_1, \dots, t_n, y))$, was von der Zahl e abhängt und nicht von einer Funktion als einem Objekt höheren Typs. Nach dem Rekursions-Theorem gibt es eine Zahl e mit

$$\{e\}(\mathbf{x}) \simeq g(e, \mathbf{x}) \simeq F(\{e\}, \mathbf{x}).$$

Für $f = \{e\}$ erhalten wir die Behauptung.

Beispiel. Die Péter'sche Funktion f ist rekursiv (vgl. Def. 1.3.30). Man definiert eine Hilfsfunktion h durch Fallunterscheidung:

$$h(x, 0) \simeq f(x, 1) \text{ und } h(x, y + 1) \simeq f(x, f(x + 1, y)),$$

also $h = R'f \circ (U_1^1, C_1^1)f \circ (U_1^2, f \circ (suc \circ U_1^2, U_2^2))$ prim-rek in f . Wie man nachrechnet, sind dann die drei definierenden Gleichungen für f äquivalent zu

$$f(x, y) \simeq (R' suc h \circ (U_2^2, U_1^2))(y, x).$$

Nach dem Korollar zum Rekursions-Theorem besitzt diese Gleichung eine partiell-rekursive Lösung f . Durch Hauptinduktion nach x und Nebeninduktion nach y sieht man, dass diese Lösung total ist. Also ist f rekursiv.

Schon in 1.2 wurde klar, dass es unendlich viele Registermaschinen gibt, die dieselbe partiell-rekursive Funktion berechnen. Kann man wenigstens entscheiden, ob zwei Registermaschinen dieselbe Funktion berechnen, ob etwa im 1-stelligen Fall für ihre Gödel-Nummern e und d die Funktionen $\{e\}^1$ und $\{d\}^1$ extensional gleich sind, ob also

$$(1) \quad \forall x \{e\}^1(x) \simeq \{d\}^1(x)$$

ist oder nicht? Wir können diese Frage mit dem Begriff der extensionalen Menge allgemeiner fassen und unter Rückgriff auf das Rekursions-Theorem negativ beantworten.

2.3.13 Definition

Eine Menge $X \subseteq \mathbb{N}$ ist **extensional**, wenn aus $e \in X$ und (1) stets $d \in X$ folgt.

Beispiele für extensionale Mengen sind \emptyset und \mathbb{N} (trivialerweise), aber auch für jede Menge Ψ von 1-stelligen partiell-rekursiven Funktionen der Menge

$$\{e \in \mathbb{N} \mid \{e\}^1 \in \Psi\}.$$

Jede extensionale Menge läßt sich so darstellen.

2.3.14 Satz

A, B seien zwei extensionale, nicht leere Mengen. Dann gibt es keine rekursive Menge C , die A enthält und zu B disjunkt ist:

Extensionale nicht-leere Mengen lassen sich nicht rekursiv trennen.

Beweis. Sei $a \in A$ und $b \in B$. Angenommen, C sei eine rekursive Menge, die A und B trennt, d. h. $A \subseteq C$ und $B \cap C = \emptyset$. Dann gibt es eine partiell-rekursive Funktion f , gegeben durch

$$f(y, x) \simeq \begin{cases} \{a\}(x), & \text{falls } y \notin C, \\ \{b\}(x), & \text{falls } y \in C. \end{cases}$$

Nach dem Rekursions-Theorem gibt es eine Zahl e , für die stets

$$f(e, x) \simeq \{e\}(x)$$

ist. Wäre nun $e \in C$, so wäre

$$\{e\}(x) \simeq f(e, x) \simeq \{b\}(x), \text{ d. h. } \{e\}^1 = \{b\}^1,$$

also $e \in B$, weil B extensional ist, und $B \cap C = \emptyset$ ergäbe $e \notin C$.

Wäre $e \notin C$, so wäre

$$\{e\}(x) \simeq f(e, x) \simeq \{a\}(x), \text{ d. h. } \{e\}^1 = \{a\}^1,$$

also $e \in A$, weil A extensional ist, und $A \subseteq C$ ergäbe $e \in C$.

Danach ist weder $e \in C$ noch $e \notin C$, und das kann nicht sein. Also war die Annahme falsch, und es gibt kein rekursives C , das A und B trennt.

Als Korollar hiervon erhält man:

2.3.15 Satz von Rice

Extensionale Mengen, die weder leer noch $= \mathbb{N}$ sind, sind nicht rekursiv.

Beweis. Sei A extensional, nicht leer und $\neq \mathbb{N}$. Dann ist auch $\mathbb{N} - A$ extensional und nicht leer. Nach Satz 2.3.14 lassen sich A und $\mathbb{N} - A$ nicht rekursiv trennen. Offenbar ist aber A die einzige Menge, die A und $\mathbb{N} - A$ trennt. Also ist A nicht rekursiv. Damit ist der Satz von Rice bewiesen.

Ist f irgendeine 1-stellige partiell-rekursive Funktion, so ist hiernach insbesondere

$$\{e | \{e\}^1 = f\} = \{e | e \text{ ist Index von } f\}$$

nicht rekursiv. Ist Ψ eine Menge, die einige, aber nicht alle 1-stelligen partiell-rekursiven Funktionen enthält, so ist ebenso

$$\{e | \{e\}^1 \in \Psi\}$$

nicht rekursiv.

2.3.16 Aufgaben

1. Man kann jede Registermaschine M so zu einer Registermaschine M' ergänzen, daß

$$M : 0\mathbf{x}\bar{0} \Rightarrow y_1\mathbf{x}\bar{0}$$

äquivalent zu

$$M' : 0\mathbf{x}\bar{0}\bar{0} \Rightarrow y_1\mathbf{x}\bar{0}\bar{0}$$

ist und M' sonst kein Ergebnis hat. M' berechnet also eine Funktion, deren Werte auch in Ergebnissen von M erscheinen. Die Einschränkung auf Funktionen-berechnende Registermaschine ist daher nicht wesentlich.

2. Man zeige: Zu jeder $(n + 1)$ -stelligen rekursiven Funktion g gibt es eine $(n + 1)$ -stellige primitiv-rekursive Funktion f mit

$$\bigcap \{W_{g(\mathbf{x},y)} | y < z\} = W_{f(\mathbf{x},z)}.$$

3. Man zeige: Zu jeder $(n + 1)$ -stelligen rekursiven Funktion g gibt es eine n -stellige primitiv-rekursive Funktion f mit

$$\bigcup \{W_{g(\mathbf{x},y)} | y \in \mathbb{N}\} = W_{f(\mathbf{x})}.$$

4. Die 3-stellige Ackermann-Funktion f genügt den Gleichungen

$$\begin{aligned} f(0, x, y) &= y + 1 \\ f(1, x, 0) &= x \\ f(2, x, 0) &= 0 \\ f(n, x, 0) &= 1 \text{ für } n > 2 \\ f(n + 1, x, y + 1) &= f(n, x, f(n + 1, x, y)). \end{aligned}$$

- (a) Man zeige: Jede Funktion $f_n : x, y \mapsto f(n, x, y)$ ist primitiv-rekursiv.
- (b) Welche bekannten Funktionen werden durch f_1, f_2, f_3, f_4 bezeichnet?
- (c) f ist rekursiv.
5. Jeder primitiv-rekursiven Funktion f sei rekursiv nach dem Aufbau von f ein **PR-Index** $\ulcorner f \urcorner$ wie folgt zugeordnet:

$$\begin{aligned} \ulcorner suc \urcorner &= \langle 0, 1 \rangle, \ulcorner U_i^n \urcorner = \langle 1, n, i \rangle, \ulcorner C_k^n \urcorner = \langle 2, n, k \rangle, \\ \ulcorner h \circ (g_1, \dots, g_r) \urcorner &= \langle 3, n, \ulcorner g_1 \urcorner, \dots, \ulcorner g_r \urcorner, \ulcorner h \urcorner \rangle \text{ für } n\text{-stellige } g_i \\ \ulcorner Rgh \urcorner &= \langle 4, n + 1, \ulcorner g \urcorner, \ulcorner h \urcorner \rangle \text{ für } n\text{-stelliges } h. \end{aligned}$$

Wir definieren: $PRInd(y) \iff y$ ist PR-Index einer prim-rek Funktion.

- (a) Zeigen Sie: Das Prädikat $PRInd$ ist primitiv-rekursiv.
- (b) Zeigen Sie: Aus $PRInd(y)$ folgt: Es gibt genau eine primitiv-rekursive Funktion f mit $\ulcorner f \urcorner = y$. Dieses f ist $(y)_1$ -stellig.

Wir definieren eine Aufzählungsfunktion a durch

$$\begin{aligned} a(y, x) &= 0, \text{ falls } \neg PRInd(y) \\ a(y, x) &= f((x)_0, \dots, (x)_{n-1}), \text{ falls } \ulcorner f \urcorner = y, \text{ also } n = (y)_1. \end{aligned}$$

- (c) Man zeige: Ist f prim-rek, so gilt stets

$$f(\mathbf{x}) = a(\ulcorner f \urcorner, \langle \mathbf{x} \rangle).$$

- (d) Man zeige: Für jedes $y \in \mathbb{N}$ ist $a_y : x \mapsto a(y, x)$ prim-rek.
- (e) Man zeige: a selbst ist rekursiv, aber nicht primitiv-rekursiv.

Kapitel 3

Einfache unlösbare Probleme

In diesem Kapitel wenden wir die Ergebnisse von 2.2 auf Registermaschinen und andere kombinatorische Systeme wie Thue-Systeme und Halbgruppen, aber auch auf die klassische Prädikatenlogik und mathematische Theorien an. Wir zeigen, dass gewisse Entscheidungsprobleme unlösbar sind, die sich in natürlicher Weise im Zusammenhang mit diesen kombinatorischen Systemen ergeben.

3.1 Halte-Probleme für Registermaschinen

Kann man entscheiden, ob eine gegebene Registermaschine zu einer bestimmten Eingabe ein Ergebnis liefert und dann stoppt, oder ob sie dies nicht tut? Gibt es zu einer gegebenen Registermaschine einen Algorithmus, der diese Entscheidung für jede Eingabe fällt? Nach der Church'schen These können wir diese Frage wie folgt präzisieren:

3.1.1 Definition

Das **Halte-** oder **Stop-Problem** für eine N -Registermaschine ist die Frage: Ist die Menge

$$\{\mathbf{x} \mid M : \mathbf{x} \text{ stoppt}\}$$

rekursiv? Das Halte-Problem für M heißt **lösbar**, wenn diese Menge rekursiv ist, sonst **unlösbar**.

Oft hat man nicht den Überblick über alle Eingaben \mathbf{x} , sondern nur über spezielle Eingaben, die durch Projektion, Konstantsetzen etc, jedenfalls durch rekursive Operationen erzeugt werden. Das genügt aber für unsere Zwecke:

3.1.2 Lemma

Ist das Halte-Problem für M lösbar und sind f_1, \dots, f_N rekursive k -stellige Funktionen, so ist auch $\{\mathbf{y} | M : f_1(\mathbf{y}) \dots f_N(\mathbf{y}) \text{ stoppt}\}$ rekursiv.

Beweis. Sei das Halte-Problem für M lösbar. Dann ist die charakteristische Funktion χ_H der Menge $H = \{\mathbf{x} | M : \mathbf{x} \text{ stoppt}\}$ rekursiv. Dann ist aber auch $\chi_F = \chi_H \circ (f_1, \dots, f_N)$ rekursiv, und es ist

$$\chi_F(\mathbf{y}) = \chi_H(f_1(\mathbf{y}), \dots, f_N(\mathbf{y})) = 0 \iff M : f_1(\mathbf{y}) \dots f_N(\mathbf{y}) \text{ stoppt}$$

Damit ist auch dieses "eingeschränkte Halte-Problem" lösbar, und das war zu zeigen.

3.1.3 Satz: Unlösbarkeit des Halte-Problems

Es gibt normierte Registermaschinen M_K mit

$$\begin{aligned} M_K : 0x\bar{0} &\Rightarrow 0x\bar{0} \quad \text{für } x \in K \text{ (vgl. Def. 2.2.9)} \\ M_K : 0x\bar{0} &\text{ stoppt sonst nicht.} \end{aligned}$$

Das Halte-Problem für jede solche Registermaschine M_K ist unlösbar.

Beweis. Die Menge K ist nach Satz 2.2.10 r.a. Also ist K nach Def. 2.2.1 und 2 Definitions- und zugleich Nullstellenmenge einer partiell-rekursiven Funktion f , die nur den Wert 0 annimmt. Dann gibt es normierte Registermaschinen M_K , die dieses f berechnen. Wäre das Halte-Problem für ein solches M_K (ob normiert oder nicht) lösbar, so wäre $\{\mathbf{x} | M_K : \mathbf{x} \text{ stoppt}\}$ rekursiv. Dann wäre nach 3.1.2 (wegen $0x\bar{0} = C_0^1(x)U_1^1(x)C_0^1(x) \dots C_0^1(x)$) erst recht $K = \{x | M_K : 0x\bar{0} \text{ stoppt}\}$ rekursiv, im Widerspruch zu Satz 2.2.10.

Damit sind (normierte) Registermaschinen M_K mit unlösbarem Halte-Problem angegeben.

Diese Registermaschinen M_K können viele Register verwenden. Wir fragen deshalb, ob wir alle Registermaschinen auch auf kleinen Registermaschinen mit wenigen, genauer: mit 2 Registern simulieren können. Weil alle partiell-rekursiven Funktionen partiell normiert berechenbar sind, können wir die Frage auf normierte Registermaschinen beschränken; zur Simulation werden wir aber auch nicht-normierte 2-Registermaschinen verwenden.

Gegeben sei eine Registermaschine M mit Speicherinhalt \mathbf{x} . Dieser Inhalt werde durch eine Elementar-Operation $i+, i-$ bzw. $i?$ (Test, ob das i -te Register leer ist) abgewandelt in \mathbf{x}' . Welche Unterprogramme mit höchstens

2 Registern muss eine 2-Registermaschine \overline{M} aufrufen, damit auf ihr der kodierte Speicherinhalt $\langle \mathbf{x} \rangle_0$ übergeht in $\langle \mathbf{x}' \rangle_0$? Offenbar haben die Unterprogramme von \overline{M} folgende Aufgaben zu erfüllen:

1. Für $i+$ muss der Inhalt des 1. Registers mit p_{i-1} multipliziert werden;
2. für $i-$ muss er durch p_{i-1} dividiert werden, sofern dies möglich ist, und sonst unverändert bleiben;
3. für $i?$ ist zu testen, ob er durch p_{i-1} teilbar ist.

Solche 2-Registermaschine gibt es tatsächlich, und zwar nicht nur für Primzahlen p_{i-1} , sondern für alle $k > 0$.

3.1.4 Lemma

Zu jedem $k > 0$ gibt es 2-Registermaschinen $Mult(k), Div(k), Test(k)$ mit den Wirkungen:

$$\begin{aligned} Mult(k) & : x0 \Rightarrow (x \cdot k)0, \\ Div(k) & : (x \cdot k)0 \Rightarrow x0 \end{aligned}$$

$$Test(k) : x0 \Rightarrow \begin{cases} x1, & \text{falls } k|x, \\ x0 & \text{sonst.} \end{cases}$$

Beweis. Wir definieren

$$\begin{aligned} Mult(k) & := (s_1 a_2^k)_1 K_{21} \text{ und} \\ Div(k) & := (s_1^k a_2)_1 K_{21}. \end{aligned}$$

Dies sind sogar normierte 2-Registermaschinen mit der geforderten Wirkung. Das Test-Programm wird folgendermaßen konstruiert:

$Test(k)$ kopiert das 1. Register R_1 schrittweise mittels $s_1 a_2$ in das 2. Register und prüft vor dem Übertragen jeder 1, ob R_1 leer ist. Solange R_1 nicht leer ist, wird dieser Programmteil nach jeweils $k \cdot s_1 a_2$ -Schritten wieder von vorn durchlaufen. Ist R_1 leer, nachdem dieser Programmteil vollständig durchlaufen ist, nachdem also ein Vielfaches von k kopiert ist, so wird $K_{21} a_2$ angeschlossen; andernfalls wird nur K_{21} angeschlossen. Unter Ausnutzung der Definitionen 1.1.6 und 1.1.9 definieren wir zuerst den kritischen Programmteil

$U(k)$ aus k Schritten s_1a_2 und $k - 1$ Tests und dann die Maschine $Test(k)$.

$$\begin{array}{rcccl}
 & s_1a_2 & & & (U(k))_1 \\
 & 2 & 1 & 3 & s & 3k + K_{21} \\
 U(k) := & 3 & +s_1a_2 & & & Test(k) := s \ 2 + s' \\
 & 5 & 1 & 6 & s & (s + 1) + K_{21} \\
 & & \vdots & & & \\
 & 3k - 4 & 1 & 3k - 3 & s & \\
 & 3k - 3 & +s_1a_2 & & &
 \end{array}$$

$U(k)$ hat $2k + (k - 1) = 3k - 1$ Instruktionen. $(U(k))_1$ hat also $3k$ Instruktionen. $K_{21} = (s_2a_1)_2$ hat 3 Instruktionen. Also ist $s = 3k + 3$, und $Test(k)$ hat $s' = 3k + 7$ Instruktionen. Die nicht-normierten Stellen in $Test(k)$ sind genau die ausgeschriebenen Instruktionen, die in $U(k)$ auf s enden und in $Test(k)$ auf s' . Bei den in $U(k)$ auf s endenden Test-Instruktionen springt das Programm $Test(k)$ aus dem Inneren des Iterationsteils $(U(k))_1$ heraus zur $(s + 1)$ -ten Instruktion, also zum letzten Programmteil K_{21} . Bei der s -ten Instruktion $s \ 2 + s'$ überspringt $Test(k)$ gerade diesen letzten Programmteil und stoppt. Auch wenn $U(k)$ für sich keine Registermaschine ist, gibt es $U(k)$ -Rechnungen

$$\begin{array}{ll}
 ((0, x, 0), \dots, (s, 0, x)), & \text{falls } 0 < x < k \text{ ist, und} \\
 ((0, x, y), \dots, (3k - 1, x - k, y + k)), & \text{falls } x \geq k \text{ ist.}
 \end{array}$$

Durch Iteration erhält man hieraus $(U(k))_1$ -Rechnungen

$$\begin{array}{ll}
 ((0, x, 0), \dots, (3k, 0, x)), & \text{falls } k|x, \text{ und} \\
 ((0, x, 0), \dots, (s + 1, 0, x)) & \text{sonst.}
 \end{array}$$

Dann ist, wie geschildert, $Test(k)$ eine 2-Registermaschine, die das Verlangte leistet.

3.1.5 Satz von Shepherdson und Sturgis

Zu jeder normierten N -Registermaschine M gibt es eine (nicht-normierte) 2-Registermaschine \overline{M} mit

$$\begin{array}{l}
 \overline{M} : \langle \mathbf{x} \rangle 0 \Rightarrow \langle \mathbf{y} \rangle 0 \iff M : \mathbf{x} \Rightarrow \mathbf{y}. \\
 \overline{M} : \langle \mathbf{x} \rangle 0 \text{ stoppt sonst nicht.}
 \end{array}$$

Beweis. Die normierten Registermaschinen sind nach Definition 1.1.12 induktiv aus a_i, s_i mit Verkettung und Iteration definiert. Rekursiv nach die-

ser Definition ordnen wir jeder normierten Registermaschine M eine 2-Registermaschine \overline{M} wie folgt zu:

$$\begin{aligned}\overline{a_i} &:= Mult(p_{i-1}) \\ \overline{s_i} &:= Test(p_{i-1})(s_2 Div(p_{i-1}))_2 \\ \overline{M_1 M_2} &:= \overline{M_1} \overline{M_2} \\ \overline{(M)_i} &:= Test(p_{i-1})(s_2 \overline{M} Test(p_{i-1}))_2.\end{aligned}$$

Wir induzieren nach dem Aufbau von M und verwenden Lemma 3.1.4. Zunächst ist M stets eine 2-Registermaschine.

1. Für a_i, s_i ist der Satz nach den Vorbemerkungen klar.
2. Für $M = M_1 M_2$ folgt der Satz aus der Induktionsvoraussetzung unmittelbar aus Lemma 1.1.8.
3. Wir betrachten den Fall der Iteration $(M)_i$ von M . Die Behauptung des Satzes für M ist dann unsere Induktionsvoraussetzung.

3.1 Es sei $x_i = 0$. Dann ist $\neg p_{i-1} | \langle \mathbf{x} \rangle$ und daher

$$\overline{(M)_i} : \langle \mathbf{x} \rangle 0 \Rightarrow \langle \mathbf{x} \rangle 0 \iff (M)_i : \mathbf{x} \Rightarrow \mathbf{x}.$$

3.2 Es sei $x_i > 0$, also auch $p_{i-1} | \langle \mathbf{x} \rangle$. Nach Lemma 1.1.11 hat dann $(M)_i$ dieselbe Wirkung wie $M(M)_i$ und $\overline{(M)_i}$ dieselbe Wirkung wie

$$Test(p_{i-1}) s_2 \overline{M} Test(p_{i-1}) (s_2 \overline{M} Test(p_{i-1}))_2, \text{ und das ist } Test(p_{i-1}) s_2 \overline{M} \overline{(M)_i}.$$

Nach Induktionsvoraussetzung ist $M : \mathbf{x} \Rightarrow \mathbf{y}$ gleichwertig mit

$$\begin{array}{ll} Test(p_{i-1}) & : \langle \mathbf{x} \rangle 0 \Rightarrow \langle \mathbf{x} \rangle 1 \\ s_2 \overline{M} & : \quad \quad \Rightarrow \langle \mathbf{y} \rangle 0. \end{array}$$

Wir machen Nebeninduktion nach der Rechendauer. Die Nebeninduktionsvoraussetzung ist dann gemäß Lemma 1.1.11

$$\overline{(M)_i} : \langle \mathbf{y} \rangle 0 \Rightarrow \langle \mathbf{z} \rangle 0 \iff (M)_i : \mathbf{y} \Rightarrow \mathbf{z}$$

Beide Induktionsvoraussetzungen zusammen ergeben wegen Lemma 1.1.8

$$\overline{(M)_i} : \langle \mathbf{x} \rangle 0 \Rightarrow \langle \mathbf{z} \rangle 0 \iff (M)_i : \mathbf{x} \Rightarrow \mathbf{z}.$$

Damit ist der Satz bewiesen. Nach diesem Satz treten an 2-Registermaschinen in kodierter Form bereits alle Phänomene auf, die unkodiert an beliebigen (normierten) Registermaschinen auftreten. Insbesondere:

3.1.6 Korollar

Zu jeder einstelligen partiell-rekursiven Funktion f gibt es eine 2-Registermaschine M mit

$$\begin{aligned} M : 2^x 0 &\Rightarrow 2^{f(x)} 0, \text{ falls } x \in \text{dom} f, \text{ und} \\ M : 2^x 0 &\text{ stoppt sonst nicht.} \end{aligned}$$

Denn nach Satz 1.2.19 gibt es eine normierte Registermaschine M' , die f berechnet. Dann gilt für $M'' = K_{12} M' L_2$:

$$\begin{aligned} M'' : x \bar{0} &\Rightarrow f(x) \bar{0}, \text{ falls } x \in \text{dom} f, \text{ und} \\ M'' : x \bar{0} &\text{ stoppt sonst nicht.} \end{aligned}$$

Konstruieren wir $M = \overline{M''}$ nach Satz 3.1.5, so erfüllt M die Behauptung.

3.1.7 Korollar

Es gibt eine 2-Registermaschine, deren Halte-Problem unlösbar ist.

Denn nach Satz 3.1.3 gibt es normierte N -Registermaschinen M , für die $\{\mathbf{x} | M : \mathbf{x} \text{ stoppt}\}$ nicht rekursiv ist. Dann ist

$$\{\langle \mathbf{x} \rangle | M : \mathbf{x} \text{ stoppt} \} = \{x | \overline{M} : x 0 \text{ stoppt}, x > 0 \text{ und } lh(x) \leq N\}$$

auch nicht rekursiv, wenn \overline{M} zu M nach Satz 3.1.5 konstruiert ist. Nach Lemma 3.1.2 ist dann das Halte-Problem für \overline{M} nicht lösbar.

3.2 Wort-Probleme für Semi-Thué-Systeme und Halbgruppen

Wir betrachten eine endlich erzeugte Gruppe G mit den endlich vielen Erzeugenden a_1, \dots, a_N . Jedes Element $x \in G$ ist dann gleich einem Produkt von diesen Erzeugenden und ihren Inversen. Z. B. ist die additive Gruppe $(\mathbb{Z}, +)$ der ganzen Zahlen von der 1 erzeugt, wie überhaupt die zyklischen Gruppen genau die von einem Element erzeugten Gruppen sind. Die Gruppe $(\mathbb{Q}, +)$ ist dagegen nicht endlich erzeugt.

Jeder Term, der aus den Erzeugenden von G und ihren Inversen mit der Gruppenoperation gebildet ist, heisst ein **Wort in den Erzeugenden**. Verschiedene Wörter w, w' können durchaus dasselbe Element von G bezeichnen, $w = w'$ in G . Z. B. bezeichnen in kommutativen Gruppen alle Permutationen eines Wortes dasselbe Element. Das **Wortproblem für G** ist die Frage, ob es einen Algorithmus gibt, der für jedes Paar von Wörtern w, w' in den Erzeugenden von G entscheidet, ob $w = w'$ in G ist oder nicht. Es heisst **lösbar**, wenn es einen solchen Algorithmus gibt, sonst **unlösbar**. Die **Unlösbarkeit des Wortproblems der Gruppentheorie** ist der berühmte und tiefliegende Satz von Boone und Novikov: Es gibt eine endlich erzeugte Gruppe, deren Wortproblem unlösbar ist.

Der Beweis dieses Satzes sprengt den Rahmen einer einführenden Vorlesung. Dagegen ist der entsprechende, schwächere Satz für Halbgruppen mit unseren Mitteln zu beweisen, wenn man die Church'sche These zugrunde legt. Wir beschäftigen uns deshalb mit kombinatorischen Systemen, für die das Wortproblem einfach zu behandeln ist.

3.2.1 Definition

Gegeben sei eine nicht-leere endliche Menge A von Zeichen, das **Alphabet**. Jede endliche Folge von Zeichen aus A (auch die leere Folge, und auch Folgen mit Wiederholungen) heisst ein **Wort in A** oder **A -Wort**. Die Menge aller Wörter in A bezeichnen wir mit $W(A)$. Zu A -Wörtern w und w' heisst das A -Wort ww' , das durch Nebeneinanderschreiben von w und w' entsteht, die **Verkettung** von w und w' .

Bemerkung. $W(A)$ ist gegen die Operation der Verkettung abgeschlossen, und diese Operation ist assoziativ.

Beispiel. Jedes Wort unserer Sprache ist ein Wort in unserem Alphabet.

Nimmt man Satzzeichen und die Lücke zwischen Wörtern unserer Sprache zu unserem Alphabet hinzu, so ist jeder Text unserer Sprache ein Wort in diesem erweiterten Alphabet.

3.2.2 Definition

Ein **Semi-Thué-System** S ist ein Paar (A, \rightarrow) , bestehend aus einem Alphabet A und einer endlichen 2-stelligen Relation \rightarrow auf $W(A)$, d. h. $g \rightarrow g'$ gilt nur für endlich viele Paare (g, g') von Wörtern in A . \rightarrow heißt die **erzeugende Relation** von S .

3.2.3 Definition

$S = (A, \rightarrow)$ sei ein **Semi-Thué-System**, und w, w' seien Wörter in A . $w \geq_1 w'$ heißt ein **Reduktionsschritt in S** , wenn es A -Wörter u, v, g, g' gibt, so dass $g \rightarrow g'$ gilt und $w \equiv ugv$ und $w' \equiv ug'v$ ist. $w \geq w'$ heißt eine **Reduktion in S** , w **reduziert in S zu w'** , wenn es A -Wörter w_0, \dots, w_n gibt ($n \geq 0$), so dass $w \equiv w_0, w_n \equiv w'$ ist und für alle $i < n$ $w_i \geq_1 w_{i+1}$ gilt.

Ein Reduktionsschritt $w \geq_1 w'$ liegt also vor, wenn w' aus w hervorgeht, indem man ein Teilwort g in w durch ein g' ersetzt, für das $g \rightarrow g'$ gilt. $w \geq w'$ heißt dann: Von w führt eine Kette von solchen Reduktionsschritten zu w' . Man ersetzt sukzessive linke Hälften von Paaren $g \rightarrow g'$ aus der erzeugenden Relation to von S durch deren rechte Hälften. Dadurch können natürlich neue Ersetzungsmöglichkeiten entstehen. Einfachste Eigenschaften von \geq sind:

3.2.4 Lemma

Für jedes **Semi-Thué-System** $S = (A, \rightarrow)$ ist die von \rightarrow erzeugte Reduktionsrelation \geq reflexiv und transitiv und mit der Verkettung verträglich. Es gilt also

- (1) $w \geq w$
- (2) Aus $u \geq v$ und $v \geq w$ folgt $u \geq w$
- (3) Aus $u \geq v$ folgt $uw \geq vw$ und $wu \geq wv$

Beweis. Es ist $w \geq w$ in 0 Reduktionsschritten. Gilt $u \geq v$ in k und $v \geq w$ in 1 Reduktionsschritten, so gilt offenbar $u \geq w$ in $k+1$ Reduktionsschritten. Gilt $u \geq_1 v$, so gilt $uw \geq_1 vw$ und $wu \geq_1 wv$ auf Grund desselben erzeugenden Paares $g \rightarrow g'$. Durch Induktion nach der Länge der Reduktionsschritte folgt (3).

Da die erzeugende Relation \rightarrow selbst entscheidbar, und da jedes Wort nur

endlich viele Teilwörter besitzt, ist auch \geq_1 entscheidbar. Und die Reduktionsrelation \geq ist zumindest aufzählbar: Die Relation \rightarrow bestimmt einen Algorithmus, der zu jedem Wort w alle Wörter w' aufzählt, auf die w in S reduziert: Man bildet zunächst alle (nur endlich viele) w' mit $w \geq_1 w'$, dann zu jedem w' alle w'' mit $w' \geq_1 w''$, und so fort.

Alle erwähnten Begriffe lassen sich leicht arithmetisieren. Man kann o. B. d. A. annehmen, dass die Buchstaben a_1, \dots, a_N des Alphabets A von S die Zahlen $1, \dots, N$ sind. Zumindest kann man jeden Buchstaben a_i durch seinen Index i bezeichnen. Dann ist jedes Wort

$$a_{i_1} \dots a_{i_r} \in W(A)$$

durch die Zahl $\langle i_1, \dots, I - r \rangle$ eindeutig kodiert, und die Relationen \rightarrow, \geq_1 und \geq werden zu zweistelligen zahlentheoretischen Relationen. Indem man ihre Definition genau hinschreibt, erkennt man sofort:

3.2.5 Lemma

In einem Semi-Thué-System $S = (A, \rightarrow)$ sind die arithmetisierten Relationen \rightarrow und \geq_1 *prim - rek*, und die Reduktionsrelationen \geq ist r. a.

Aufgabe. Man führe den Beweis durch.

3.2.6 Definition

Das **Wortproblem** für das Semi-Thué-System $S = (A, \rightarrow)$ ist die Frage, ob die von \rightarrow erzeugte (zahlentheoretische) Reduktionsrelation \geq rekursiv ist. Das Wortproblem für S heißt **lösbar**, wenn dies der Fall ist, andernfalls unlösbar.

Das Wortproblem für $S = (A, \rightarrow)$ ist also die Frage nach der Existenz eines rekursiven Verfahrens, das für beliebige vorgelegte A -Wörter w und w' entscheidet, ob w in S zu w' reduziert oder nicht.

Die Unlösbarkeit des Wortproblems für geeignete Semi-Thué-Systeme läßt sich nun dadurch auf die Unlösbarkeit des Halte-Problems von 2 – *Registermaschine* zurückführen, dass man jeder 2 – *Registermaschine* M ein Semi-Thué-System derart zuordnet, dass die Rechnungen auf M zu Reduktionsketten in $S(M)$ werden. Dazu fassen wir jede M -Konfiguration (j, x, y) als ein Wort $(\bar{x}j\bar{y})$ auf, wobei $(\bar{x}$ und \bar{y} Wörter aus x bzw. y Strichen

| sind. Dadurch braucht man nur endlich viele Buchstaben, und es ist ziemlich klar, wie man die Instruktionen von M in erzeugende Paare von $S(M)$ übersetzt.

3.2.7 Definition

Es sei M eine 2-Registermaschine. Dann sei $S(M)$ das folgende Semi-Thué-System (A, \rightarrow) :

1. Das Alphabet A besteht aus $(,), |$ und den Instruktionsnummern von M (einschließlich der Stopnummer).
2. Die erzeugende Relation \rightarrow enthält genau folgende Paare:

Für jede Instruktion	gilt
$j\ 1\ +\ l$	$j \rightarrow l$
$j\ 2\ +\ l$	$j \rightarrow l $
$j\ 1\ -\ l$	$ j \rightarrow l$ und $(j \rightarrow (l$
$j\ 2\ -\ l$	$j \rightarrow l$ und $)j \rightarrow l)$
$j\ 1\ k\ l$	$ j \rightarrow k$ und $(j \rightarrow (l$
$j\ 2\ k\ l$	$j \rightarrow k $ und $)j \rightarrow l)$

Ist s die Länge (also die Stopnummer) von M , so gilt noch

$$|s \rightarrow s \text{ und } (s| \rightarrow (s.$$

Das Wort aus x Strichen

$$\underbrace{|\dots|}_{x\text{-mal}}$$

bezeichnen wir mit \bar{x} . Jedes A -Wort einer Gestalt $(\bar{x}j\bar{y})$ bezeichnen wir auch als M -Wort. Die M -Wörter $(\bar{x}s\bar{y})$ heißen auch **Endwörter**. Ein M -Wort $(\bar{x}j\bar{y})$ **entspricht** der M -Konfiguration $(j, \bar{x}\bar{y})$.

Bemerkung. Die M -Wörter sind genau die A -Wörter, die den M -Konfigurationen entsprechen. Die Endwörter entsprechen dabei den M -Endkonfigurationen. Die Definition 3.2.7 läuft darauf hinaus, dass man die 2-Registermaschine M insgesamt als Semi-Thué-System auffassen kann. Dabei geht die Folgekonfigurations-Relation über in \geq_1 , und die zweistellige Relation, erste und letzte M -Konfiguration einer M -Rechnung zu sein, geht über in \geq .

3.2.8 Lemma

w sei ein M -Wort, aber kein Endwort. Dann gilt $w \geq_1 w'$ genau dann, wenn w' ein M -Wort ist und die M -Folgekonfiguration der w entsprechenden M -Konfiguration entspricht w' .

Beweis. In w tritt genau eine Instruktionsnummer j auf. Zur Erzeugung von $w \geq_1 w'$ kann also nur ein Paar $g \rightarrow g'$ verwendet worden sein bei dem j in g auftritt. Von diesen gibt es höchstens zwei, die nach Definition von \rightarrow dieselbe Wirkung auf w haben wie die j -te Instruktion auf die w entsprechende M -Konfiguration.

3.2.9 Lemma

Ist w ein Endwort $\neq (s)$, so gilt $w \geq_1 w'$ genau dann, wenn w' ein Endwort ist, das genau ein Zeichen $|$ weniger enthält als w , und zwar fehlt dieses Zeichen $|$ links von s , solange w nicht mit $(s$ anfängt. Es gibt kein Wort w' mit $(s) \geq_1 w'$.

Dies ist nur die Übertragung der Definition 3.2.7 für Endwörter auf die Relation \geq_1 .

3.2.10 Lemma

Zu jedem M -Wort w gibt es höchstens ein Wort w' mit $w \geq_1 w'$, und dieses Wort ist ein M -Wort.

Dies folgt aus Lemma 3.2.8 und Lemma 1.1.3, solange w kein Endwort ist, und aus Lemma 3.2.9 für Endwörter w .

Durch iterierte Anwendung der Lemmata 3.2.8 und 3.2.9 erhält man

Satz 3.4. Es sei M eine 2 – Registermaschine mit s Instruktionen und $S(M)$ das zu definierte Semi-Thué-System. Dann gilt:

$$M : x y \text{ stoppt} \iff \text{In } S(M) \text{ gilt } (\bar{x}0\bar{y}) \geq (s).$$

Beweis. Genau dann, wenn $M : x y$ stoppt, gibt es eine M -Rechnung

$$((0, x, y), (j_1, x_1, y_1), \dots, (j_T, x_T, y_T)) \text{ mit } j_T = s, j_t < s \text{ für } t < T.$$

Dies ist nach Lemma 3.2.8 gleichwertig mit

$$(1) \quad (\bar{x}0\bar{y}) \geq_1 (\bar{x}j_1\bar{y}_1) \geq_1 \dots \geq_1 (\bar{x}_T j_T \bar{y}_T) \text{ mit } J_T = s, j_t < s \text{ für } t < T.$$

Hieraus folgt $(\bar{x}0\bar{y}) \geq (\bar{x}_T j_T \bar{y}_T)$ mit $j_T = s$ und $(\bar{x}_T s \bar{y}_T) \geq (s)$ durch $(x_T + y_T)$ -fache Anwendung von Lemma 3.2.9. Mit Lemma 3.2.4 folgt $(\bar{x}0\bar{y}) \geq (s)$.

Gilt umgekehrt $(\bar{x}0\bar{y}) \geq (s)$, so gibt es nach Lemma 3.2.10 eine eindeutig bestimmte Reduktionskette von $(\bar{x}0\bar{y})$ nach (m) und in dieser Kette ein erstes Endwort, auf das dann nach Lemma 3.2.9 nur noch Endwörter folgen. Also gibt es eine Reduktionskette (1), was nach obigem genau dann gilt, wenn $M : x y$ stoppt. Damit ist der Satz bewiesen.

Bemerkung. $S(M)$ arbeitet hiernach genau wie eine 2-Registermaschine, aber nicht genau wie M , sondern wie M mit angehängten Löschprogrammen, d. h. wie $M L_1 L_2$. Das Stopverhalten von M ist allerdings dasselbe wie das von $S(M)$, was sich hier als Reduktion auf das Wort (s) äußert.

Korollar. Es gibt Semi-Thué-Systeme, deren Wortproblem unlösbar ist.

Beweis. Nach Satz 3.3, Korollar 2, gibt es 2-Registermaschine M , deren Halte-Problem unlösbar ist, für die also nach Satz 3.4 die Menge

$$(2) \quad \{x, y | M : x y \text{ stoppt} \} = \{x, y | \text{In } S(M) \text{ gilt } (\bar{x}0\bar{y}) \geq (s)\}$$

nicht rekursiv ist. Da die Menge (der Nummern) aller M -Wörter primitivrekursiv ist, ist dann auch die (arithmetisierte) Reduktions-Relation \geq nicht rekursiv; denn wäre \geq rekursiv, so wäre nach Lemma 1.3.12 auch die Menge

$$\{w | w \geq (s) \text{ und } w \text{ ist } M\text{-Wort}\}$$

und damit auch die Menge (2) rekursiv.

Aufgabe. Man zeige: Die Menge der Nummern aller M -Wörter ist primitivrekursiv.

In Semi-Thué-Systemen braucht die Reduktions-Relation \geq nicht symmetrisch zu sein. In den Systemen $S(M)$ ist sie stets unsymmetrisch. Wir interessieren uns für die von \geq erzeugte Äquivalenzrelation und betrachten dementsprechend Thué-Systeme.

3.2.11 Definition

Ein Semi-Thué-System $T = (A, \rightarrow)$ ist ein **Thué-System**, wenn die erzeugende Relation \rightarrow symmetrisch ist, wenn also aus $g \rightarrow g'$ stets $g' \rightarrow g$ folgt.

3.2.12 Lemma

In einem Thué-System T ist der Reduktionsschritt \geq_1 symmetrisch und die Reduktions-Relation \geq eine Äquivalenzrelation.

Beweis. Gilt $w \geq_1 w'$, so ist $w \equiv ugv$ und $w' \equiv ug'v$ mit $g \rightarrow g'$. Da T ein Thué-System ist, gilt dann auch $g \rightarrow g'$, also $w' \equiv ug'v \geq_1 ugv \equiv w$. Die Relation \geq ist nach Lemma 3.2.4 schon reflexiv und transitiv. Da \geq von \geq_1 erzeugt ist, ist mit \geq_1 auch \geq symmetrisch.

Jedes Thué-System ist nach Definition ein Semi-Thué-System. Umgekehrt bestimmt jedes Semi-Thué-System eindeutig ein Thué-System nach folgendem Verfahren.

3.2.13 Definition

Ist S ein Semi-Thué-System (A, \rightarrow) , so sei $T(S)$ das Thué-System $(A, \rightarrow \cup \rightarrow^{-1})$, d. h.

1. Das Alphabet von $T(S)$ ist das Alphabet von S ;
2. In $T(S)$ gilt $g \rightarrow g'$ genau dann, wenn in S $g \rightarrow g'$ oder $g' \rightarrow g$ gilt.

Bemerkung. Die erzeugende Relation von $T(S)$ ist also die symmetrische Hülle der erzeugenden Relation von S . Diese symmetrische Hülle ist immer noch endlich, so dass $T(S)$ tatsächlich ein Thué-System ist. Für Thué-Systeme S ist $\rightarrow \cup \rightarrow^{-1} = \rightarrow$, also $T(S) = S$. Insbesondere ist stets $T(T(S)) = T(S)$.

3.2.14 Lemma

In $T(S)$ gilt $w \geq_1 w'$ genau dann, wenn $w \geq_1 w'$ oder $w' \geq_1 w$ in S gilt.

Denn $w \geq_1 w'$ in $T(S)$ bedeutet: es gibt $g \rightarrow g'$ in $T(S)$, also $g \rightarrow g'$ in S oder $g' \rightarrow g$ in S , so dass $w \equiv ugv$ und $w' \equiv ug'v$. Das heißt aber: $w \geq_1 w'$ in S oder $w' \geq_1 w$ in S .

Das Wortproblem für $T(S)$ braucht nicht ebenso kompliziert zu sein wie das von S . Die Reduktions-Relation in $T(S)$ ist im Gegensatz zur Situation bei \rightarrow und \geq_1 i. a. nicht einfach die Vereinigung von \geq und \leq in S , sondern sie ist die transitive Hülle dieser Vereinigung. Es ist daher möglich, dass S ein unlösbares Wortproblem hat, während in $T(S)$ stets $w \geq w'$ gilt.

Einfacher ist die Situation für die speziellen Thué-Systeme $T(S(M))$, die über

das Semi-Thué-System $S(M)$ zu einer 2 – *Registermaschine* M definiert sind. Für diese Systeme gilt das Eindeutigkeitslemma 3.2.10 nicht, weil es verschiedene M -Konfigurationen geben kann, die dieselbe Folgekonfiguration haben. Es gilt aber auch folgende Kürzungsregel:

3.2.15 Lemma

M sei eine 2 – *Registermaschine* und w ein M -Wort. Gilt $w' \geq_1 w$ und $w' \geq_1 w''$, so ist auch w' ein M -Wort und $w'' \equiv w$.

Beweis. In $S(M)$ gilt $g' \rightarrow g$ mit $w' \equiv ug'v$ und $w \equiv ugv \equiv (\bar{x}j\bar{y})$. g' unterscheidet sich von g , das j enthält, allenfalls in dieser Instruktionsnummer und um einen Strich |. Also unterscheidet sich auch w' von w höchstens in diesen beiden Punkten und ist wieder ein M -Wort. Nach Lemma 3.2.10 ist dann $w'' \equiv w$. - Diese Kürzungsregel genügt schon für folgenden entscheidenden Schritt:

3.2.16 Lemma

M sei eine 2 – *Registermaschine* und w ein M -Wort. Gilt $w \equiv (s)$ in $T(S(M))$, so gilt dies bereits in $S(M)$.

Beweis. Nach Voraussetzung gibt es Reduktionsketten von w nach (s) , und diese bestehen nach Lemma 3.2.10 und 3.2.14 nur aus M -Wörtern. Unter allen diesen sei

$$(3) \quad w \equiv w_o \geq_1 w_1 \geq_1 \dots \geq_1 w_T \equiv (s)$$

eine kürzeste. Ist dies keine Reduktionskette in $S(M)$, so gibt es einen letzten Reduktionsschritt $w_t \geq_1 w_{t+1}$ in der Kette, der in $T(S(M))$, aber nicht in $S(M)$ gilt. Dann gilt $w_{t+1} \geq_1 w_t$ in $S(M)$ nach Lemma 3.2.14, so dass w_{t+1} wegen Lemma 3.2.9 nicht das Wort (s) ist. Also gibt es noch w_{t+2} in der Kette, und $w_{t+1} \geq_1 w_{t+2}$ gilt in $S(M)$ nach Wahl von t . Nach Lemma 3.2.15 ist dann $w_t \equiv w_{t+2}$, so dass die Kette (3) doch verkürzbar ist im Gegensatz zur Annahme. Also gilt $w \geq (s)$ bereits in $S(M)$.

Satz 3.5 Es sei M eine 2 – *Registermaschine* mit s Instruktionen. Dann gilt:

$$M : xy \text{ stoppt} \iff \text{In } T(S(M)) \text{ gilt } (\bar{x}0\bar{y}) \geq (s).$$

Dies folgt unmittelbar aus Satz 3.4 und Lemma 3.2.16

Korollar. Es gibt Thué-Systeme, deren Wortproblem unlösbar ist.

Dies folgt ebenso auf Satz 3.5 wie das vorige Korollar aus Satz 3.4.

Thué-Systeme kann man als Halbgruppen ansehen, wenn man in ihnen aufeinander reduzierbare Wörter identifiziert.

3.2.17 Definition

Eine **Halbgruppe** (ein **Monoid**) ist eine Struktur (G, \circ) , mit einer nicht-leeren Menge G und einer 2-stelligen Funktion \circ , die assoziativ ist, d. h. für die

$$x \circ (y \circ z) = (x \circ y) \circ z$$

für alle x, y, z aus G gilt.

Beispiele. $(\mathbb{Z}, +)$, (\mathbb{Z}, \cdot) sind Halbgruppen; allgemeiner: jede Gruppe ist eine Halbgruppe; jeder Ring ist bezüglich der Multiplikation eine Halbgruppe.

3.2.18 Definition

Ist $A \subseteq G$, so heißt die Halbgruppe (G, \circ) **von A erzeugt**, wenn es zu jedem $x \in G$ $a_1, \dots, a_n \in A$ gibt mit $x = a_1 \circ \dots \circ a_n$. (Ein neutrales Element von G darf dabei durch das leere Produkt aus 0 Faktoren dargestellt werden.) Ist eine Halbgruppe von einer endlichen Menge erzeugt, so heißt sie **endlich erzeugt**.

Beispiel. Ist $A \neq \emptyset$, so ist $W(A)$, mit der Operation der Verkettung, die von A erzeugte **freie** Halbgruppe.

Wir können die Elemente jeder von A erzeugten Halbgruppe G durch die Wörter aus $W(A)$ darstellen. G ist genau dann frei, wenn verschiedene Wörter aus $W(A)$ stets verschiedene Elemente von G bezeichnen. I. A. werden aber verschiedene Wörter dasselbe Element von G bezeichnen können.

3.2.19 Definition

Es sei G eine von einer endlichen Menge A erzeugte Halbgruppe. Das Wortproblem für G ist die Frage, ob die (arithmetisierte) Relation

$$\{(w, w' | w, w' \in W(A), w = w' \text{ in } G)\}$$

rekursiv entscheidbar ist. Ist dies der Fall, so heißt das Wortproblem für G **lösbar**, andernfalls **unlösbar**.

Beispiel. Das Wortproblem für die von einer nicht-leeren Menge A erzeugte freie Halbgruppe $W(A)$ ist selbstverständlich lösbar. Denn es ist $w = w'$ in $W(A)$ nur dann, wenn die Wörter w und w' Buchstaben für Buchstaben übereinstimmen.

Interessante Halbgruppen erhält man, wenn man von einem Thué-System (A, \rightarrow) ausgeht und $W(A)$ nach der Äquivalenzrelation \geq faktorisiert, die nach Lemma 3.2.4 mit der Verkettung verträglich ist.

3.2.20 Definition

Es sei $T = (A, \rightarrow)$ ein Thué-System mit der Reduktions-Relation \geq . Für jedes $w \in W(A)$ bezeichnen wir die Äquivalenzklasse $\{w' \mid w \geq w'\}$ mit $[w]$, und wir definieren

$$[v] \circ [w] := [vw].$$

Die Operation \circ ist wohldefiniert. Wir setzen

$$W(A)/\geq := (\{[w] \mid w \in W(A)\}, \circ).$$

Die Gleichungen $[g] = [g']$, für die $g \rightarrow g'$ in T gilt, heißen die definierenden Gleichungen von $W(A)/\geq$.

Bemerkung. Gilt $u \geq v$ in T , so auch $uw \geq vw$. Da \geq eine Äquivalenzrelation auf $W(A)$ ist, ist dann

$$[u] \circ [w] = [v] \circ [w] \text{ und ebenso } [w] \circ [u] = [w] \circ [v].$$

Also ist \circ unabhängig vom Repräsentanten und damit wohldefiniert.

Satz 3.6. Ist $T = (A, \rightarrow)$ ein Thué-System mit der Reduktions-Relation \geq , so ist $W(A)/\geq$ eine endlich erzeugte Halbgruppe mit endlich vielen definierenden Gleichungen, in der $[w] = [w']$ gleichwertig ist mit $w \geq w'$ in T .

Beweis. Jedes $w \in W(A)$ ist ein Wort $a_1 \dots a_n$ mit $a_i \in A$. Also ist jedes $[w] \in W(A)/\geq$ gleich einem Produkt $[a_1] \circ \dots \circ [a_n]$ mit $a_i \in A$. $W(A)/\geq$ ist also von der endlichen Menge $\{[a] \mid a \in A\}$ der Äquivalenzklassen der Elemente von A erzeugt. Da \circ wohldefiniert ist, ist \circ eine zweistellige Operation auf $W(A)/\geq$. \circ ist assoziativ wegen

$$([u] \circ [v]) \circ [w] = [uvw] = [u] \circ ([v] \circ [w]).$$

Also ist $W(A)/\geq$ eine Halbgruppe. Sie hat nur endlich viele definierende Gleichungen, weil \rightarrow eine endliche Relation ist. In ihr steht $[w] = [w']$ per

Definition für $w \geq w'' \iff w' \geq w''$ f. a. w'' , und das ist gleichwertig mit $w \geq w'$, weil \geq eine Äquivalenzrelation ist.

Bemerkung Jede endlich erzeugte Halbgruppe G ist isomorph zu einer Halbgruppe $W(A)/\equiv$, der freien Halbgruppe $W(A)$ über einem endlichen Erzeugendensystem A von G , faktorisiert nach der Gleichheitsrelation in G . Aber diese Gleichheitsrelation braucht nicht von endlich vielen definierenden Gleichungen erzeugt zu werden.

Korollar. Es gibt endlich erzeugte Halbgruppen mit endlich vielen definierenden Gleichungen, deren Wortproblem unlösbar ist.

Beweis. Es sei $T = (A, \rightarrow)$ ein Thué-System. Die Halbgruppe $W(A)/\geq$ wird erzeugt von ihren Elementen $[a]$ mit $a \in A$. Ordnen wir dem Halbguppenelement $[a]$ stets dieselbe Nummer i wie dem Buchstaben a zu, so erhalten das Element $[w] = [a_1] \circ \dots \circ [a_n]$ und das $w = a_1 \dots a_n$ dieselbe Tupel-Nummer $\langle i_1, \dots, i_n \rangle$. Dann stimmen nach Satz 3.6 die arithmetisierten Relationen $\{[w], [w'] \mid [w] = [w'] \text{ in } W(A)/\geq\}$ und $\{w, w' \mid w \geq w' \text{ in } T\}$ völlig überein. Das Wortproblem für $W(A)/\geq$ ist also genau dann lösbar, wenn das Wortproblem für T lösbar ist. Mit dem Korollar zu Satz 3.5 folgt die Behauptung.

Die Halbgruppen, deren Wortproblem hier als unlösbar nachgewiesen ist, sind mit Sicherheit keine Gruppen. Es erfordert einen beträchtlichen zusätzlichen Aufwand, endlich erzeugte Gruppen mit unlösbarem Wortproblem nachzuweisen.

Die Ergebnisse in diesem Paragraphen lassen sich dahin verschärfen, dass die Alphabete der betrachteten Semi-Thué-Systeme nur zwei Buchstaben enthalten und dass entsprechend die Halbgruppen von nur zwei Elementen erzeugt werden.

3.3 Entscheidungs-Probleme für mathematische Theorien

In den zuletzt betrachteten Halbgruppen $G = W(A)/\geq$ folgten alle wahren Gleichungen durch einfache kombinatorische Schlüsse aus endlich vielen sog. definierenden Gleichungen. Diese einfachen Schlüsse werden in der elementaren Theorie der Gleichheit formuliert. Man wird deshalb fragen, ob jeder solchen Halbgruppe G eine mathematische Theorie $T(G)$ entspricht, in der genau die in G wahren Gleichungen herleitbar sind. Hat G ein un-

lösbares Wortproblem, so ist in $T(G)$ schon die Herleitbarkeit geschlossener Gleichungen nicht entscheidbar. Hieraus folgt aber die Unentscheidbarkeit der klassischen Prädikatenlogik, und zwar schon für eine recht einfach überschaubare Formelklasse.

Wir setzen hier erstmals elementare Kenntnisse aus der klassischen Prädikatenlogik voraus. Insbesondere verwenden wir folgende Begriffe: Eine **mathematische Theorie** T (der 1. Stufe) besteht aus einer **Sprache** $L(T)$ der 1. Stufe, in der die nichtlogischen Zeichen von T aufgeführt werden, und einer Formelmengende $Ax(T)$, der Menge der **Axiome** von T . Die nicht-logischen Zeichen einer Sprache sind die in ihr auftretenden **Prädikats-** und **Funktionszeichen**, darunter die **Konstanten** als 0-stellige Funktionszeichen. Das **Gleichheitszeichen** = gilt als logisches Zeichen. Wenn die Menge $Ax(T)$ leer ist, nennen wir T eine **logische Theorie**.

Eine **Struktur** \mathfrak{a} zu $L(T)$ beinhaltet eine **Interpretation** der nicht-logischen Zeichen von T auf einer Menge $|\mathfrak{a}|$, dem **Universum** oder **Individuenbereich** von \mathfrak{a} . Die **Modelle** von T sind die Strukturen zu $L(T)$, in denen alle Axiome von T **gelten**. Eine Formel B **gilt** in einer Theorie T , wenn B in allen Modellen von T gilt. B ist **herleitbar** in T , $T \vdash B$, wenn es eine **Herleitung** von B in T gibt, die außer Axiomen von T nur **logische Axiome und Grundschlüsse** verwendet. Nach dem Korrektheitsatz ist jede in T herleitbare Formel gültig in T , und nach dem Vollständigkeitssatz gilt auch die Umkehrung.

3.3.1 Definition

Das **Entscheidungsproblem** für eine mathematische Theorie T ist die Frage, ob die Menge der (arithmetisierten) Formeln, die in T herleitbar sind, rekursiv ist. Ist dies der Fall, so heißt T **entscheidbar** und das Entscheidungsproblem für T **lösbar**; andernfalls heißt T **unentscheidbar** und das Entscheidungsproblem für T **unlösbar**.

Die Terminologie weicht hier von der in den vorigen Paragraphen ab. In Analogie zu Definition 3.1.1, 3.2.6 und 3.2.19 würde man hier vom "Herleitbarkeits-Problem" für eine Theorie T sprechen. Dieses Problem ist allerdings das historisch erste von den hier behandelten Entscheidungsfragen. Es wird seit den zwanziger Jahren für verschiedene, immer feiner eingeteilte Formelklassen behandelt; die erste unentscheidbare Theorie wurde 1935 von A. Church angegeben.

Wir ordnen jetzt jeder endlich erzeugten Halbgruppe G mit endlich vielen er-

zeugenden Gleichungen, wie sie in 3.2 auftreten, eine mathematische Theorie $T(G)$ zu, in der genau die variablen-freien Gleichungen herleitbar sind, die in G wahr sind.

3.3.2 Definition

$T = (A, \rightarrow)$ sei ein Thué-System mit der Reduktions-Relation \geq ; (G, \circ) sei die Halbgruppe $W(A)/\geq$.

1. Die Sprache $L(G)$ enthalte als nicht-logische Zeichen:
das zweistellige Funktionszeichen \cdot ,
die Konstante e für die Äquivalenzklasse $[\]$ des leeren Wortes und
die Erzeugenden $[a]$ von (G, \circ) mit $a \in A$ als Konstanten.
2. Die Axiomenmenge $Ax(G)$ bestehe aus dem Assoziativgesetz

$$\forall x \forall y \forall z (x \cdot y) \cdot z = x \cdot (y \cdot z),$$

dem Gesetz vom neutralen Element $\forall x (x \cdot e = x \wedge e \cdot x = x)$, und den endlich vielen definierenden Gleichungen von (G, \circ) nach Ersetzung der Operation \circ durch das Zeichen \cdot .

Die Theorie $(L(G), Ax(G))$ heißt die **Theorie zur Halbgruppe** (G, \circ) und wird mit $T(G)$ bezeichnet.

In der Theorie $T(G)$ gelten wie in jeder mathematischen Theorie u. a. folgende logische Gesetze:

$$\begin{array}{ll} (refl) & \vdash \quad t = t \\ (sym) & \vdash \quad s = t \quad \rightarrow \quad t = s \\ (trans) & \vdash \quad r = s \quad \rightarrow \quad s = t \quad \rightarrow \quad r = t \\ (ers) & \vdash \quad s = s' \quad \rightarrow \quad t = t' \quad \rightarrow \quad s \cdot t = s' \cdot t' \\ (mp) & \text{Aus } \vdash B \quad \text{und } \vdash B \quad \rightarrow \quad C \text{ folgt } \vdash C. \end{array}$$

3.3.3 Definition

Ist $w \equiv a_1 \dots a_n (n \geq 0)$ ein Wort aus $W(A)$, so bezeichnen wir den Term

$$(((\dots([a_1] \cdot [a_2])\dots) \cdot [a_n])$$

aus $L(G)$ auch mit $[w]$.

3.3.4 Lemma

$T(G) \vdash [v] \cdot [w] = [vw]$.

Beweis durch Induktion nach der Länge von w .

1. w sei leer, also $[w] \equiv e$ nach Definition 3.3.3 und $vw \equiv v$. Dann gilt die Behauptung wegen des Gesetzes vom neutralen Element.
2. w sei ua . Dann ist $[w] \equiv ([u] \cdot [a])$, und es gilt

$$\begin{array}{ll} \vdash [v] \cdot [w] = ([v] \cdot [u]) \cdot [a] & \text{nach dem Assoziativgesetz,} \\ \vdash [v] \cdot [u] = [vu] & \text{nach Induktionsvoraussetzung, also folgt} \\ \vdash ([v] \cdot [u]) \cdot [a] = [vu] \cdot [a] & \text{mit (ers), (refl) und (mp),} \\ \text{also} & \\ \vdash [v] \cdot [w] = [vu] \cdot [a] & \text{mit (trans) und (mp),} \end{array}$$

und das ist wegen $[vu] \cdot [a] \equiv [vua] \equiv [vw]$ die Behauptung.

3.3.5 Lemma

Zu jedem variablenfreien Term t aus $l(G)$ gibt es ein Wort $w \in W(A)$, so dass $T(G) \vdash t = [w]$ ist.

Beweis durch Induktion nach der Länge von t .

1. Für $t = e$ wählen wir w leer, für $t = [a]$ wählen wir $w \equiv a$. In beiden Fällen ist $\vdash t = [w]$ wegen *(refl)*.
2. Ist $t = r \cdot s$, so gibt es nach Induktionsvoraussetzung u, v mit $\vdash r = [u]$ und $\vdash s = [v]$. Es folgt $\vdash t = [u] \cdot [v]$ mit *(ers)* und *(mp)*.

Daraus folgt nach Lemma 3.3.4 mit *(trans)* und *(mp)* die Behauptung für $w \equiv uv$.

3.3.6 Lemma

Aus $g \rightarrow g'$ folgt $T(G) \vdash [g] = [g']$.

Denn dann ist $[g] = [g']$ (mit \circ geschrieben) eine definierende Gleichung von (G, \circ) , so dass $[g] = [g']$ ein Axiom von $T(G)$ ist.

3.3.7 Lemma

Aus $w \geq_1 w'$ folgt $T(G) \vdash [w] = [w']$.

Beweis. Es gibt dann u, v und $g \rightarrow g'$ mit $w \equiv ugv$ und $w' \equiv ug'v$. Wegen (*ers*) gilt

$\vdash [u] = [u] \rightarrow [g] = [g'] \rightarrow [u] \cdot [g] = [u] \cdot [g']$, also
 $\vdash [u] \cdot [g] = [u] \cdot [g']$ mit (*mp*), (*refl*) und Lemma 3.3.6. Es folgt
 $\vdash [ug] = [ug']$ nach Lemma 3.3.4 mit (*trans*), (*sym*) und (*mp*).

Durch teilweise Wiederholung des Arguments, angewandt auf v , erhält man die Behauptung.

3.3.8 Lemma

Aus $w \geq W'$ folgt $T(G) \vdash [w] = [w']$.

Denn dann gilt $w \equiv w_o \geq_1 w_1 \geq_1 \dots \geq_1 w_n \equiv w'$ für geeignete Wörter w_i , also $\vdash [w_i] = [w_{i+1}]$ für $i < n$ nach Lemma 3.3.7. Durch Induktion nach n folgt mit (*refl*), (*trans*) und (*mp*) die Behauptung.

Bemerkung. Die Verwendung von (*sym*) im Beweis von Lemma 3.3.7 läßt sich vermeiden, wenn man vorher auch die symmetrische Fassung von Lemma 3.3.4, $\vdash [vw] = [v] \cdot [w]$, beweist.

Satz 3.7. $T = (A, \rightarrow)$ sei ein Thué-System mit der Reduktions-Relation $\geq; (G, \circ)$ sei die Halbgruppe $W(A)/\geq$. Für variablenfreie Terme s, t ist

$$s = t \text{ gültig in } (G, \circ) \iff T(G) \vdash s = t.$$

Beweis. Da die Struktur (G, \circ) mit ihren Erzeugenden zu $L(G)$ passen soll, haben wir das Zeichen \cdot durch die Operation \circ und jede Konstante $[a]$ durch sich selbst, e durch $[]$ zu interpretieren. Nach Wahl von $Ax(G)$ ist dann (G, \circ) ein Modell von $T(G)$.

a) Zu s und t gibt es nach Lemma 3.3.5 Wörter w und w' , so dass

$$\vdash s = [w] \text{ und } \vdash t = [w']$$

ist. Wegen der Modelleigenschaft von (G, \circ) gelten dann $s = [w]$ und $t = [w']$ in (G, \circ) , also auch $[w] = [w']$. Nach Satz 3.6 folgt hieraus $w \geq w'$ in T , also $T(G) \vdash [w] = [w']$ nach Lemma 3.3.8. Mit (*trans*), (*sym*) und (*mp*) ergeben unsere Voraussetzungen dann

$$T(G) \vdash s = t.$$

- b) Umgekehrt sei $T(G) \vdash s = t$. Da (G, \circ) ein Modell von $T(G)$ ist, ergibt der Korrektheitssatz: $s = t$ ist wahr in (G, \circ) .

Damit ist der Satz bewiesen.

Dieser Satz liefert eine Charakterisierung der Lösbarkeit des Wortproblems von (G, \circ) .

Korollar 1. Das Wortproblem für (G, \circ) ist genau dann lösbar, wenn die (arithmetisierte) Relation

$$(1) \{ (s, t) \mid T(G) \vdash s = t \text{ und } s, t \text{ sind variablenfreie Terme} \}$$

rekursiv ist.

Denn wegen Satz 3.7 ist (1) die Relation

$$\{ (s, t) \mid \text{In } (G, \circ) \text{ ist } s = t \text{ wahr} \}.$$

Geht s_G aus s hervor, wenn man in Term s das Zeichen \cdot durch die Operation \circ ersetzt, so ist $s = t$ wahr in (G, \circ) genau dann, wenn $s_G = t_G$ ist. Also ist (1) gerade die Identität auf (G, \circ) , die wegen Definition 3.2.19 genau dann rekursiv ist, wenn das Wortproblem für (G, \circ) lösbar ist.

Schon die Herleitbarkeit variablenfreier Gleichungen in $T(G)$ ist hiernach nur dann entscheidbar, wenn das Wortproblem von (G, \circ) lösbar ist. Da es nach dem Korollar zu Satz 3.6 Halbgruppen mit unlösbarem Wortproblem gibt, folgt hieraus:

Korollar 2. Es gibt endlich axiomatisierte Theorien mit nur einem zweistelligen Funktionszeichen und endlich vielen Konstanten, in denen schon die Herleitbarkeit variablenfreier Gleichungen nicht entscheidbar ist.

Wegen des Deduktions-Theorems der Prädikatenlogik kann man diese endlich vielen Axiome als Prämissen von Implikationen auffassen und erhält dann:

Korollar 3. Es gibt logische Theorien mit nur einem zweistelligen Funktionszeichen \cdot und endlich vielen Konstanten, in denen schon die Herleitbarkeit von geschlossenen Formeln der Gestalt

$$\begin{aligned} & \forall x \forall y \forall z ((x \cdot y) \cdot z = x \cdot (y \cdot z) \wedge x \cdot e = x \wedge e \cdot x = x) \rightarrow \\ & \rightarrow s_1 = t_1 \rightarrow s_2 = t_2 \rightarrow \dots \rightarrow s_n = t_n \rightarrow s = t \end{aligned}$$

nicht entscheidbar ist.

Korollar 4 (Church 1935). Es gibt unentscheidbare logische Theorien: Die klassische Prädikatenlogik (mit Identität und Funktionszeichen) ist unentscheidbar.

Unser Verfahren zur Konstruktion unentscheidbarer Theorien ist offenbar gar nicht auf Sparsamkeit in den verwendeten Mitteln angelegt. Es liegt daher nahe, dass sich etwa Korollar 3 noch wesentlich verschärfen läßt. Allerdings braucht man dann auch feinere Methoden.

Zusammenfassung. Ausgehend von der Menge $K = \{x \mid x \in W_x\}$ zeigten wir am Anfang des Kapitels, dass die *Registermaschinen* M_K , die die auf K beschränkte Nullfunktion $C_o^1 \upharpoonright K$ berechnet, ein unlösbares Halte-Problem hat. M_K simulierten wir durch eine 2-*Registermaschine* \overline{M}_K , deren Halte-Problem deshalb ebenfalls unlösbar ist. Diese 2-*Registermaschine* \overline{M}_K kann man auffassen als ein Semi-Thué-System $S(\overline{M}_K)$, dessen Wort-Problem dann unlösbar ist. Symmetrisiert man $S(\overline{M}_K)$, so erhält man das Thué-System $T(S(\overline{M}_K))$ mit ebenfalls unlösbarem Wort-Problem, d. h. unentscheidbarer Reduktions-Relation \geq . Das Alphabet A von $T(S(\overline{M}_K))$ ebenso wie von $S(\overline{M}_K)$ besteht dabei aus $(;), |$ und den endlich vielen Instruktionsnummern von \overline{M}_K . Faktorisiert man die freie Halbgruppe $W(A)$ über A nach \geq , so erhält man die Halbgruppe

$$W(A) / \geq = (G, \circ) = G(T(S(\overline{M}_K)))$$

mit unlösbarem Wort-Problem, die schließlich zu der unentscheidbaren Theorie

$$T(G) = T(G(T(S(\overline{M}_K))))$$

führt. Die ganze Entwicklung in diesem Kapitel beruht also wesentlich auf zwei Ergebnissen:

1. (Satz 2.5) K ist *r.a.*, aber nicht rekursiv;
2. (Satz 3.3) Jede *Registermaschine* läßt sich auf einer 2-*Registermaschine* simulieren.

Die Abfolge in diesem Kapitel ist natürlich nicht ganz so gleichförmig wie eben skizziert. Zur Konstruktion einer 2-*Registermaschine* mit unlösbarem Halte-Problem genügt wegen Satz 3.3 irgendeine *Registermaschine* mit unlösbarem Halte-Problem. Jede solche 2-*Registermaschine* M liefert dann ein Semi-Thué-System $T(S(M))$ mit unlösbarem Wort-Problem (Sätze 3.4 und 3.5). Dann genügt wieder irgendein Thué-System $T = (A, \rightarrow)$ mit unlösbarem Wort-Problem zur Konstruktion einer Halbgruppe $W(A) / \geq =$

$(G, \circ) = G(T)$ mit unlösbarem Wort-Problem und einer unentscheidbaren Theorie $T(G) = T(G(T))$ (Sätze 3.6 und 3.7). Die 2-Registermaschine und die Thué-Systeme nehmen also eine gewisse Schlüsselstellung ein, wobei sich nach unserer Auffassung die 2-Registermaschine "fast unmittelbar" als Thué-Systeme auffassen lassen. Dadurch stehen die 2-Registermaschine im Mittelpunkt der Untersuchungen dieses Kapitels.

Anhang (Aufgaben usw.)

Aufgabe. Zu jeder Registermaschine M gibt es eine normierte Registermaschine M' mit eventuell mehr Registern als M und

$$M' : \bar{x}0 \Rightarrow \bar{y}0 \iff M : x \Rightarrow y,$$

$$M' : \bar{x}0 \text{ stoppt sonst nicht.}$$

Nach Satz 3.2 kann man alle Registermaschinen auf einer einzigen normierten Registermaschine U_1 simulieren.

Aufgabe. Man schreibe Test (2) und Test (3) explizit auf.