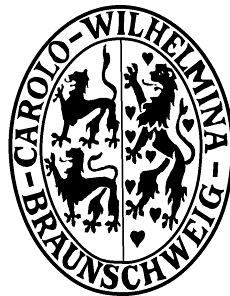
Theoretische Informatik

Vorlesungsskript

Dietmar Wätjen



Institut für Theoretische Informatik

Technische Universität Braunschweig

Oktober 2000

Inhaltsverzeichnis

Vorwort **3**

1 Endliche Automaten **5**
1.1 Einführung . . . 5
1.2 Mealy- und Moore-Automaten . . . 7
1.3 Reduktion von Automaten . . . 16

2 Turingmaschinen **23**
2.1 Definitionen . . . 23
2.2 Beispiele für Turingmaschinen und ihre Zusammensetzbarkeit . . . 25
2.3 Modifizierte Turingmaschinen . . . 31
2.4 Turing-Berechenbarkeit . . . 34
2.5 Gödelisierung . . . 40
2.6 Universelle Turingmaschinen . . . 43
2.7 Unentscheidbare Probleme . . . 44
2.8 Registermaschinen . . . 49

3 Rekursive Funktionen **61**
3.1 Primitiv-rekursive Funktionen . . . 61
3.2 Die Ackermann-Funktion . . . 66
3.3 Der μ-Operator und μ-rekursive Funktionen . . . 72

4 Sprachen, Grammatiken und erkennende Automaten **79**
4.1 Einführung . . . 79
4.2 Die Chomsky-Hierarchie . . . 81
4.3 Endliche erkennende Automaten und reguläre Sprachen . . . 85
4.4 Reguläre Ausdrücke . . . 92
4.5 Weitere Automatentypen und ihre zugehörigen Sprachen . . . 96

5 Fixpunkttheorie und kontextfreie Sprachen **101**
5.1 Partielle Ordnungen und Fixpunkte . . . 101
5.2 Fixpunkttheorie und kontextfreie Sprachen . . . 110

6 Deterministische Polynomialzeitalgorithmen **117**
6.1 Beispiele effizienter Algorithmen . . . 118
6.2 Die Komplexitätsklasse P . . . 129
6.3 Berechnungsprobleme und Reduzierbarkeit . . . 135
6.4 Die Robustheit der Klassen P und FP . . . 141
6.5 Effiziente geometrische Algorithmen . . . 145

7 Nichtdeterministische Polynomialzeitalgorithmen **153**
7.1 Die Komplexitätsklasse NP . . . 153
7.2 NP-Vollständigkeit . . . 157
7.3 Weitere NP-vollständige Probleme . . . 163

8 Komplexität von Optimierungsalgorithmen 171
8.1 Optimierungsprobleme 171
8.2 Approximation von Optimierungsproblemen 178

9 Raumkomplexität 185

10 Parallele Algorithmen 191
10.1 Das PRAM-Modell 191
10.2 PRAM-Algorithmen 192
10.3 Simulationen zwischen verschiedenen PRAM-Modellen 200
10.4 Die Komplexitätsklasse *NC* 203
10.5 Boolesche Schaltkreise und PRAMs 206
10.6 Netzwerkmodelle 214

Literaturverzeichnis 221

Index 223

Vorwort

Die Theoretische Informatik beschäftigt sich mit Abstraktionen, Modellbildungen, allgemein mit Grundlagenforschung, die mit der Struktur, Verarbeitung, Darstellung und Übertragung von Informationen im Zusammenhang steht. Es handelt sich vor allem um Probleme, die mit Computern und deren Anwendung zu tun haben. Die Theoretische Informatik ist also ein weites Feld, und sie hat naturgemäß in vielen Bereichen Berührungspunkte mit anderen Gebieten der Informatik.

In ihrer Methodik ist die Theoretische Informatik mathematisch, indem sie Definitionen, Sätze und Beweise benutzt. Sie erhält jedoch ihre Impulse in erster Linie aus den vorher bereits genannten praktischen Problemen. Es wird versucht, für solche Probleme theoretische Modelle zu finden, die eine genaue und verständliche Beschreibung ermöglichen. Damit können unter Umständen praktische Lösungsansätze erleichtert werden.

Ohne Anspruch auf Vollständigkeit zu erheben, nennen wir im folgenden einige Bereiche und Gebiete, die in der Theoretischen Informatik behandelt werden. Der zentrale Begriff der Informatik und damit auch der Theoretischen Informatik ist der des *Algorithmus*. Schon vor Erfindung der Computer hat man sich mit der Frage der *Berechenbarkeit* auseinandergesetzt, also mit der Frage, welche Probleme algorithmisch lösbar sind. Verschiedene formale Modelle für den Algorithmusbegriff wurden eingeführt, und sie werden auch heute noch weiter untersucht. Neben der prinzipiellen Frage der Berechenbarkeit ist es natürlich interessant zu wissen, mit welchem Aufwand eine solche Berechnung durchgeführt werden kann. Die *Komplexitätstheorie* untersucht solche Fragestellungen. Mit Hilfe der Berechnungsmodelle definiert man dabei verschiedene Komplexitätsklassen. Neben diesen allgemeinen Überlegungen werden jedoch auch konkrete Algorithmen betrachtet, z.B. Algorithmen zur *Mustererkennung*, *Such-* und *Sortieralgorithmen*, *Graphalgorithmen* und *Algorithmen in der Zahlentheorie*. Die letztgenannten sind wichtig in der *Kryptologie*, in der Lehre vom Ver- und Entschlüsseln von Nachrichten, die mit der zunehmenden Verbreitung von Computersystemen eine immer stärkere Bedeutung erhält. In all diesen Fällen ist die *Analyse der Algorithmen* wichtig, d.h. Untersuchungen über ihr Zeitverhalten und ihren Platzbedarf im schlechtesten Fall oder im Mittel. Eine Verringerung des Rechenaufwands kann mit *parallelen Algorithmen* erreicht werden. In diesem Zusammenhang werden auch *parallele Prozesse* und *parallele Rechnerarchitekturen* betrachtet.

Zur praktischen Berechnung werden Algorithmen als Programme in einer Programmiersprache kodiert. So beschäftigt sich die Theoretische Informatik auch mit den Grundlagen der *imperativen*, *funktionalen*, *logischen* und *objektorientierten Programmierung* und betrachtet in diesem Zusammenhang die *Syntax und Semantik von Programmiersprachen*, die *Logik für Programmiersprachen* und die *Programmverifikation*.

Programmiersprachen können als spezielle *formale Sprachen* aufgefaßt werden. Formale Sprachen werden durch *Grammatiken* und andere *Ersetzungssysteme* erzeugt oder können durch verschiedene Arten von *Automaten* und *Maschinen* erkannt werden. Grammatiken wurden erstmals in der Linguistik betrachtet, wo man versucht hat, mit ihrer Hilfe natürliche Sprachen zu beschreiben. Einige Ersetzungssysteme wie

z.B. *Lindenmayersysteme* wurden in der Biologie verwendet, um die Entwicklung biologischer Organismen darzustellen. Die Automaten- und Maschinentypen dienen außer zur Spracherkennung auch als mehr oder weniger realistische Modelle von Computersystemen.

Ein wichtiges praktisches Problem ist die Organisation von Daten in Speichern von Rechensystemen. Hierfür werden theoretische Untersuchungen über *Datenstrukturen* und *Datenbanken* durchgeführt und entsprechende semantische Fragen behandelt.

Aus dem eben skizzierten weiten Spektrum der Theoretischen Informatik kann in dieser Vorlesung nur ein kleiner Teil behandelt werden. Wir beschränken uns in der Vorlesung „Theoretische Informatik I“ im wesentlichen auf die Grundlagen der Berechenbarkeit. Diese sind in der Informatik von fundamentaler Bedeutung, da sie einen Begriff von den prinzipiellen Möglichkeiten von Rechensystemen und ihren Beschränkungen vermitteln. Die Kapitel 1 bis 3 beschreiben die Berechnung von Funktionen durch unterschiedlich leistungsfähige Berechnungsmodelle. Die kleinste Klasse von Funktionen wird durch *Schaltfunktionen* bestimmt, die Sie in den Vorlesungen der Technischen Informatik kennenlernen werden (siehe auch [23], Kapitel 1). *Endliche Automaten* berechnen eine umfangreichere Klasse und mit Hilfe von *Turingmaschinen* oder *Registermaschinen* (*random access Maschinen*) kann schließlich jede intuitiv berechenbare Funktion dargestellt werden. Äquivalent ist dies auch mit *μ-rekursiven Funktionen* möglich.

In den Kapiteln 4 und 5 werden *Sprachen*, *Grammatiken* sowie *endliche erkennende Automaten*, *Kellerautomaten*, *linear beschränkte Automaten* sowie *erkennende Turingmaschinen* betrachtet. Die Berechnungen der Automaten und Maschinen führen in diesen Fällen zur Erkennung von Wörtern von Sprachen. Auf die *Komplexität* solcher Berechnungen wird insbesondere in der Vorlesung „Theoretische Informatik II “ eingegangen.

In der Vorlesung „Theoretische Informatik II“ wird in den Kapiteln 6 bis 9 ausführlich auf die *Komplexität von Algorithmen* eingegangen, und es werden die entsprechenden Berechnungsmodelle und Komplexitätsklassen besprochen. In Kapitel 6 werden *effiziente Algorithmen* betrachtet, also solche, die in vernünftiger Zeit die gestellten Probleme lösen. Für die Algorithmen aus Kapitel 7 gilt das leider nicht. Dazu gehören sehr viele Algorithmen für sehr wichtige Probleme aus der Praxis. Damit man auch dafür wenigstens näherungsweise vernünftige Lösungen bekommt, werden in Kapitel 8 *approximierende Algorithmen* eingeführt. Kapitel 9 betrachtet den *Raumbedarf* von Algorithmen, während in Kapitel 10 *parallele Algorithmen* und parallele Berechnungsmodelle untersucht werden.

Ein großer Teil des Stoffes dieser Vorlesung befindet sich in meinem Buch [23]. Wichtig bei der Ausarbeitung der Vorlesung waren die Monographien von *Bobrow/Arbib* [3], *Hermes* [9], *Salomaa* [19], *Moll/Arbib/Kfoury* [15], *Hopcroft/Ullman* [11], *Bovet/Crescenzi* [4], *JáJá* [12] und *Vollmar/Worsch* [22]. Neben diesen Büchern sind im Literaturverzeichnis noch weitere genannt, die zur vertiefenden Beschäftigung mit dem hier behandelten Stoff geeignet sind.

Dietmar Wätjen — im Oktober 1998

1 Endliche Automaten

1.1 Einführung

Wenn Sie in den Vorlesungen der Technischen Informatik Schaltfunktionen und Schaltnetze kennenlernen werden, werden sie feststellen, daß dabei im wesentlichen nicht auf den Zeitbedarf eingegangen wird. Idealisierend können wir annehmen, daß nach dem Anlegen des n-Tupels von Eingabewerten das Ergebnis, d.h. der Funktionswert der Schaltfunktion $f : \{0,1\}^n \to \{0,1\}$, nach einem sehr kurzen Zeitintervall ausgegeben wird. Verallgemeinernd kann man bei m parallelen Schaltnetzen auch ein m-Tupel als Ausgabewert erhalten. Beim Anlegen eines weiteren n-Tupels von Eingabewerten erfolgt eine weitere Ausgabe, die nicht vom früheren Verhalten des Schaltnetzes beeinflußt wird. Durch Schaltnetze sind also Vorgänge, die vom früheren Verhalten abhängig sind und somit ein „Gedächtnis" erfordern, nicht realisierbar. Dies leisten jedoch, wenn auch nicht für alle Probleme, endliche Automaten. Mit ihnen kann somit eine umfassendere Klasse von Funktionen berechnet werden.

Beispiel 1.1.1 Anhand einiger einfacher Probleme und ihrer Lösungen wollen wir die Definition von endlichen Automaten motivieren, die dann im Abschnitt 1.2 formal angegeben wird.

(a) Wir betrachten die serielle Addition. Als Eingabefluß kommen nacheinander die Paare von Bits $\binom{i_0}{j_0}, \binom{i_1}{j_1}, \binom{i_2}{j_2}, \ldots, \binom{i_n}{j_n}$, $i_k, j_k \in \{0,1\}$, $k = 0, \ldots, n$, $n \in \mathbb{N}_0$, in dieser Reihenfolge an. Als Ausgabe soll die Dualsumme $i_n \ldots i_0 + j_n \ldots j_0$ geliefert werden, die bitweise, von hinten nach vorn, ausgegeben wird. Dazu ist eine Übertragsverarbeitung notwendig, die durch die Einführung von Zuständen möglich wird. Durch die folgende Tabelle wird die Arbeit eines solchen Additionsautomaten beschrieben:

Eingänge x_1 / Zustand x_2	x_1: 0, x_2: 0	x_1: 0, x_2: 1	x_1: 1, x_2: 0	x_1: 1, x_2: 1	
z_0	z_0	z_0	z_0	z_1	nächster Zustand
	0	1	1	0	Ausgabe
z_1	z_0	z_1	z_1	z_1	
	1	0	0	1	

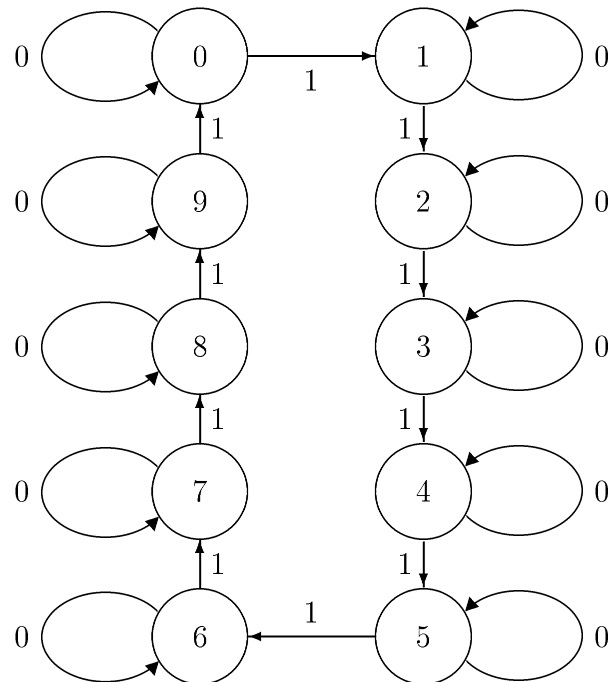
Die Arbeit beginnt im Zustand z_0. Dieser Zustand zeigt an, daß zuvor kein Übertrag stattgefunden hat. Befindet sich der Automat im Zustand z_0 und wird $\binom{0}{1}$ eingegeben, so wird der Wert 1 ausgegeben, und der Automat bleibt im Zustand z_0, da bei der Addition $0 + 1$ kein Übertrag stattgefunden hat. Wird jedoch $\binom{1}{1}$ im Zustand z_0 eingegeben, so wird 0 ausgegeben, und es findet ein Übergang in den Zustand z_1 statt. Mit Hilfe von z_1 wird sich also gemerkt, daß ein Übertrag stattgefunden hat.

(b) Es soll jetzt ein Modulo-m-Zähler konstruiert werden. Gegeben sei die Menge

$\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$, $m \geq 1$. Die Funktion $\sigma : \mathbb{Z}_m \to \mathbb{Z}_m$, die durch

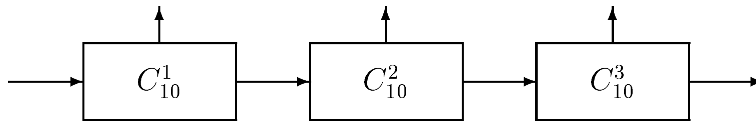
$$\sigma(n) = \begin{cases} n+1, & \text{falls } 0 \leq n < m-1 \\ 0, & \text{falls } n = m-1 \end{cases}$$

definiert ist, beschreibt das Zählen modulo m. σ wird für jeden Zählimpuls angewendet. Der Modulo-m-Zähler C_m ist ein Gerät mit m Zuständen $0, 1, ..., m-1$ und zwei möglichen Eingaben 0 und 1, das modulo m die 1-Eingaben zählt, und zwar mit Hilfe der Zustände. C_m ist durch einen gerichteten Graphen (auch Zustandsdiagramm oder Zustandsgraph genannt) beschreibbar. Speziell erhalten wir für C_{10} den folgenden Graphen:

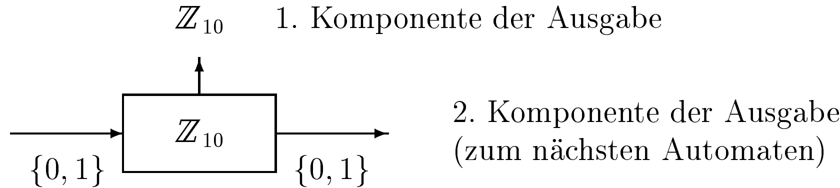


Die Knoten dieses Graphen sind mit den Zuständen markiert. Die Pfeile sind mit der Eingabe bezeichnet, wobei die Pfeilrichtung den zugehörigen Zustandsübergang angibt. Dieser Gesamtautomat „merkt“ sich, wieviele Einsen er erhalten hat, seitdem er zuletzt im Zustand 0 war. Er hat also ein beschränktes (endliches) Gedächtnis. Als Ausgabe können wir den jeweiligen aktuellen Zustand verwenden.

(c) Aus C_{10} soll jetzt ein Modulo-1000-Zähler aufgebaut werden. Dafür installieren wir je einen Zähler für die Einer-, Zehner- und Hunderterstelle. Der Eingang von außen führt nur zum „Einerautomaten“. Ein Übertrag vom Einerautomaten zum Zehnerautomaten erfolgt genau dann, wenn der Einerautomat im Zustand 9 ist und eine 1 eintrifft. Dann wird eine 1 als Ausgabe des Einerautomaten geliefert, die vom Zehnerautomaten als Eingabe verarbeitet wird. Dieser Aufbau wird durch



verdeutlicht. Im Detail wird ein einzelner Automat durch



dargestellt. Dabei ist er formal durch die folgenden fünf Größen spezifiziert:

(1) Die Menge Z der (internen) Zustände, hier speziell $Z = \mathbb{Z}_{10}$.
(2) Die Menge X der möglichen Eingabesymbole, hier $X = \{0,1\}$.
(3) Die Menge Y der möglichen Ausgabesymbole, hier $Y = \mathbb{Z}_{10} \times \{0,1\}$.
(4) Eine Funktion $\delta : Z \times X \to Z$, die in Abhängigkeit vom gegenwärtigen Zustand und der gegenwärtigen Eingabe den nächsten Zustand festlegt. Diese Zustandsüberführungsfunktion wird in unserem Beispiel durch

$$\delta : \mathbb{Z}_{10} \times \{0,1\} \to \mathbb{Z}_{10} \text{ mit } \delta(k,x) = \begin{cases} k, & \text{falls } x = 0 \\ \sigma(k), & \text{falls } x = 1 \end{cases}$$

definiert.
(5) Eine Funktion $\lambda : Z \times X \to Y$, die in Abhängigkeit vom gegenwärtigen Zustand und der gegenwärtigen Eingabe die Ausgabe festlegt. Diese Ausgabefunktion ist hier durch

$$\lambda : \mathbb{Z}_{10} \times \{0,1\} \to \mathbb{Z}_{10} \times \{0,1\}$$

bestimmt. Dabei ist

$$\lambda(k,x) = (k,y) \text{ mit } y = \begin{cases} 1, & \text{falls } k = 9 \text{ und } x = 1 \\ 0 & \text{sonst.} \end{cases}$$ □

Es gibt auch Probleme, die mit endlichen Automaten nicht gelöst werden können. Es sei z.B. eine Zeichenkette gegeben, die aus Symbolen a und b besteht. Wir wollen entscheiden, ob sie mit lauter a's beginnt, gefolgt von einem b und schließlich mit ebensovielen a's wie am Anfang endet. Wir können uns vorstellen, daß die Zeichenkette ein Symbol nach dem anderen dem Automaten als Eingabe angeboten wird. Diese Aufgabe ist jedoch durch einen endlichen Automaten nicht lösbar, wie wir in Beispiel 1.2.5 sehen werden.

1.2 Mealy- und Moore-Automaten

Definition 1.2.1 Ein 5-Tupel $M = (Z, X, Y, \delta, \lambda)$ heißt *(endlicher) Mealy-Automat* (kurz *(endlicher) Automat*), wenn Z, X und Y endliche nichtleere Mengen (*Zustands-*, *Eingabe-* bzw. *Ausgabemenge*) und $\delta : Z \times X \to Z$ und $\lambda : Z \times X \to Y$ Abbildungen sind (*Zustandsüberführungs-* bzw. *Ausgabefunktion*). □

Anschaulich können wir einen Mealy-Automaten so interpretieren, daß er, falls er sich im Zustand $z \in Z$ befindet und die Eingabe $x \in X$ erhält, in den Folgezustand $\delta(z,x)$ übergeht und dabei die Ausgabe $\lambda(z,x)$ liefert. Erhält er anschließend eine neue Eingabe x', so geht er in den Zustand $\delta(\delta(z,x),x')$ über usw. Diese verallgemeinerte Arbeitsweise werden wir in Definition 1.2.5 formalisieren.

Wir erkennen sofort, daß Mealy-Automaten *deterministisch* sind, daß also der Folgezustand und die Ausgabe eindeutig bestimmt sind. Das liegt daran, daß δ und λ Abbildungen sind und daher für jedes Paar $(z,x) \in Z \times X$ einen eindeutigen Wert $\delta(z,x)$ bzw. $\lambda(z,x)$ liefern. Nichtdeterministische Automaten sind dagegen unvollständig oder nicht eindeutig definiert, es können z.B. zwei mögliche Folgezustände zu einem Paar (z,x) existieren. Auf nichtdeterministische Automaten werden wir im Zusammenhang mit endlichen erkennenden Automaten in Kapitel 4 zurückkommen.

Gelegentlich wird in der Literatur auf die Forderung der Endlichkeit der Mengen Z, X und Y verzichtet. In diesem Buch sollen sie jedoch prinzipiell endlich sein.

Für jeden Mealy-Automaten kann implizit ein Taktgeber angenommen werden. Das bedeutet, daß der Automat nur zu diskreten Zeitpunkten betrachtet wird. Werden diese diskreten Zeitpunkte, wie üblich, mit der Menge $\mathbb{N}_0$ identifiziert, also mit der Menge der natürlichen Zahlen einschließlich der 0, so ist das Übergangsverhalten des Automaten durch

$$\delta(z(t),x(t)) = z(t+1) \quad \text{und} \quad \lambda(z(t),x(t)) = y(t)$$

für $t \in \mathbb{N}_0$ gegeben. Dabei bedeutet $z(t)$ den Zustand zur Zeit t, $x(t)$ die Eingabe und $y(t)$ die Ausgabe zur Zeit t. Die Ausgabefunktion wird gelegentlich auch als $\lambda(z(t),x(t)) = y(t+1)$ interpretiert. Wir wollen jedoch in diesem Buch die vorhergehende Form vorziehen.

Jede Schaltfunktion $f : \{0,1\}^n \to \{0,1\}$ kann offenbar durch einen Mealy-Automaten berechnet werden. Man wähle einfach $X = \{0,1\}^n$, $Y = \{0,1\}$ und $Z = \{z\}$. Weiter werde $\lambda(z,(x_1,\ldots,x_n)) = f(x_1,\ldots,x_n)$ und $\delta(z,(x_1,\ldots,x_n)) = z$ für jedes $(x_1,\ldots,x_n) \in \{0,1\}^n$ gesetzt. Die Berechnung der Ausgabe hängt somit nicht vom Zustand ab, da es nur einen gibt. Dies entspricht unserer früheren Bemerkung, daß die Berechnung einer Schaltfunktion unabhängig ist von dem vorhergehenden Verhalten des Schaltnetzes.

Ein Mealy-Automat kann durch Angabe seiner *Wertetabellen* für die Funktionen δ und λ der Form

δ	x_1	$\cdots$	x_j	$\cdots$	x_n
z_1					
$\vdots$			$\vdots$		
z_i		$\cdots$	$\delta(z_i,x_j)$	$\cdots$	
$\vdots$			$\vdots$		
z_m					

und

λ	x_1	$\cdots$	x_j	$\cdots$	x_n
z_1					
$\vdots$			$\vdots$		
z_i		$\cdots$	$\lambda(z_i,x_j)$	$\cdots$	
$\vdots$			$\vdots$		
z_m					

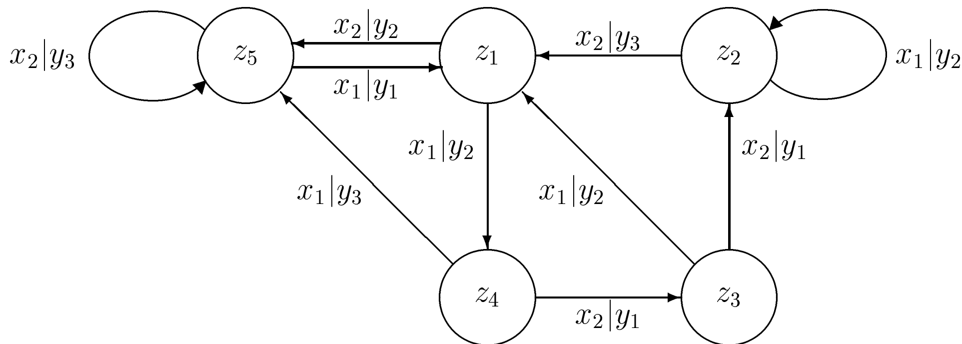
festgelegt werden. Dabei sind $X = \{x_1,\ldots,x_n\}$ und $Z = \{z_1,\ldots,z_m\}$ implizit durch die Tabellen gegeben. Y kann außer den in der rechten Tabelle vorkommenden Elementen $\lambda(z_i,x_j)$ noch weitere, aber niemals ausgegebene und somit überflüssige Ausgabesymbole enthalten. Eine ähnliche Beschreibung haben wir bereits in Beispiel 1.1.1(a) gewählt.

Optisch übersichtlicher ist eine Darstellung als Zustandsdiagramm oder Zustandsgraph wie in Beispiel 1.1.1(b). Wir erhalten

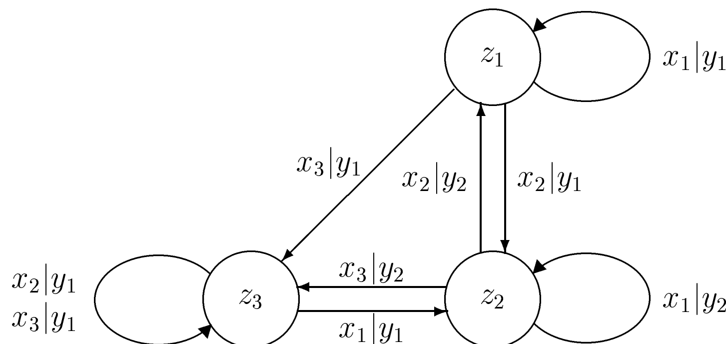
Definition 1.2.2 Es sei $M = (Z, X, Y, \delta, \lambda)$ ein Mealy-Automat. Ein *Zustandsdiagramm* oder *Zustandsgraph* von M ist ein markierter gerichteter Graph, durch den M wie folgt beschrieben wird. Die Zustände des Automaten und die Knoten des Graphen werden einander bijektiv zugeordnet. Die Knoten sind durch die Zustände bezeichnet. Zwei Knoten z und z' werden durch eine gerichtete Kante (Pfeil) von z nach z' verbunden, wenn ein $x \in X$ existiert mit $\delta(z, x) = z'$. Die Markierung einer solchen Kante ist $x \mid \lambda(z, x)$. □

Beispiel 1.2.1 Wir geben zunächst Zustandsdiagramme für drei Mealy-Automaten an. Für einen von ihnen notieren wir auch die Wertetabelle. Wir werden sehen, daß der Automat (b) eine eingeschränkte Form besitzt. Als Verallgemeinerung dieser Einschränkung führen wir anschließend in Definition 1.2.3 einen entsprechenden Automatentyp ein.

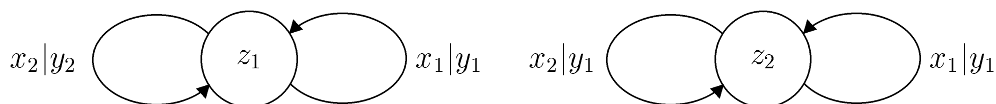
(a)



(b)



(c)



Für den Automaten aus (b) geben wir zusätzlich die beiden Wertetabellen an:

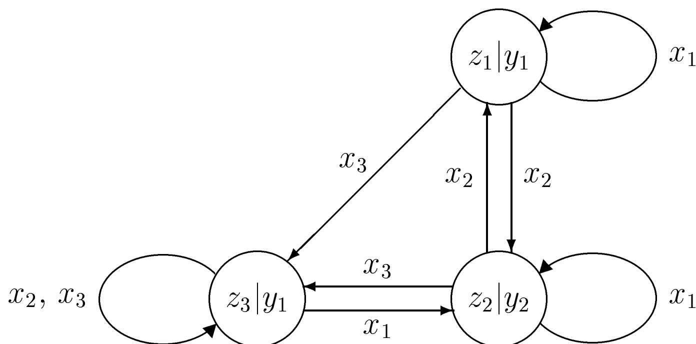
δ	x_1	x_2	x_3
z_1	z_1	z_2	z_3
z_2	z_2	z_1	z_3
z_3	z_2	z_3	z_3

λ	x_1	x_2	x_3
z_1	y_1	y_1	y_1
z_2	y_2	y_2	y_2
z_3	y_1	y_1	y_1

Er unterscheidet sich von denen aus (a) und (c) dadurch, daß seine Ausgabe nur vom Zustand abhängt. Statt der Abbildung λ würde eine Abbildung $\beta : Z \to Y$ mit $\beta(z_1) = y_1$, $\beta(z_2) = y_2$ und $\beta(z_3) = y_1$ genügen. λ kann formal aus β mit Hilfe der Projektionsabbildung $pr_1 : Z \times X \to Z$, die durch $pr_1(z, x) = z$ definiert ist, gewonnen werden. Es gilt dann nämlich

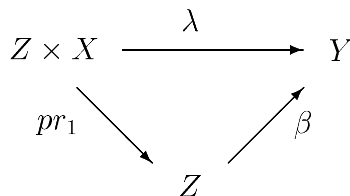
$$\lambda(z, x) = (\beta \circ pr_1)(z, x) = \beta(z) = y.$$

Aufgrund dieser Abbildung β kann (b) auch durch seinen *Zustand-Ausgabe-Graphen* dargestellt werden:



Im Zustand-Ausgabe-Graphen sind die Knoten durch $z|y$ bezeichnet, d.h. durch einen Zustand z wie im Zustandsdiagramm und die zugehörige Ausgabe. Entsprechend entfällt bei den Kanten die Markierung durch die Ausgaben. □

Definition 1.2.3 Es sei $M = (Z, X, Y, \delta, \lambda)$ ein Automat. M heißt *Moore-Automat (Zustand-Ausgabe-Automat)*, wenn eine Funktion $\beta : Z \to Y$ existiert, so daß das Diagramm



kommutativ ist, also $\lambda = \beta \circ pr_1$ gilt. Man schreibt auch $M = (Z, X, Y, \delta, \beta)$. □

Die Funktion β wird auch *Ausgabefunktion* des Moore-Automaten genannt. Die Taktung eines Moore-Automaten ergibt sich unter Beachtung des Diagramms von Definition 1.2.3 aus der eines Mealy-Automaten auf Seite 8, also

$$\delta(z(t), x(t)) = z(t + 1) \ , \ \beta(z(t)) = y(t).$$

Auf den ersten Blick ist ein Moore-Automat eingeschränkter als ein Mealy-Automat. Wir werden jedoch in Satz 1.2.2 zeigen, daß zu jedem Mealy- ein Moore-Automat existiert mit im allgemeinen größerer Zustandszahl, der auf die Eingabe einer Folge von Eingabesymbolen mit der gleichen Folge von Ausgabesymbolen wie der Mealy-Automat antwortet. Allerdings hinkt er gegenüber dem Mealy-Automaten um einen Takt hinterher, was wir durch die folgende Skizze andeuten:

Mealy-Automat:	$y(t)$,	$y(t+1)$,	$y(t+2)$,	$\dots$
	$\diagdown\!\!\diagdown$	$\diagdown\!\!\diagdown$	$\diagdown\!\!\diagdown$	
Moore-Automat:		$\hat{y}(t+1)$,	$\hat{y}(t+2)$,	$\hat{y}(t+3)$, $\dots$

Da Automaten Folgen von Eingabesymbolen, also sogenannte Wörter, verarbeiten sollen, wird zunächst dieser Begriff definiert.

Definition 1.2.4 (a) Es sei A eine endliche Menge. Dann heißt $A^* = \{x_1 \dots x_n \mid x_i \in A \ , \ 1 \le i \le n \ , \ n \in \mathbb{N}_0\}$ die *Menge der Wörter über* A. $w = x_1 \dots x_n$ heißt *Wort über* A, es hat die *Länge* $|w| = n$. Für $n = 0$ erhalten wir das *leere Wort* ε mit $|\varepsilon| = 0$.

(b) Es seien $w_1 = x_1 \dots x_n$ und $w_2 = y_1 \dots y_m$ zwei Wörter über A. Dann wird $w_1 w_2 = x_1 \dots x_n y_1 \dots y_m$ die *Konkatenation* von w_1 und w_2 genannt.

(c) Es werde $A^+ = A^* - \{\varepsilon\}$ gesetzt. $\square$

Aus der Algebra ist bekannt, daß eine Menge H mit einer Verknüpfung $\circ : H \times H \to H$ *Halbgruppe* heißt, wenn für alle $h_1, h_2, h_3 \in H$ das Assoziativgesetz

$$(h_1 \circ h_2) \circ h_3 = h_1 \circ (h_2 \circ h_3)$$

gilt. Eine Halbgruppe ist ein *Monoid*, wenn zusätzlich ein Einselement $e \in H$ existiert mit

$$e \circ h = h \circ e = h$$

für alle $h \in H$. Wir stellen fest, daß A^* ein Monoid ist, wenn wir als Verknüpfung die Konkatenation und als Einselement das leere Wort ε wählen. Wir nennen A^* auch *freies Monoid* über A. A^+ ist eine Halbgruppe, die *freie Halbgruppe* über A.

Oben wurde bemerkt, daß Automaten Wörter verarbeiten sollen. Zur Beschreibung dieser Arbeitsweise wird die Abbildung δ erweitert.

Definition 1.2.5 Es seien Z und X endliche Mengen, und $\delta : Z \times X \to Z$ sei eine Abbildung. Die *X^*-Erweiterung von δ* ist durch die Abbildung $\delta^* : Z \times X^* \to Z$ mit

$$\delta^*(z, \varepsilon) = z \quad \text{und} \quad \delta^*(z, wx) = \delta(\delta^*(z, w), x)$$

für alle $z \in Z, w \in X^*, x \in X$ gegeben. $\square$

Aus dieser Definition ergibt sich, daß jedes Wort von links nach rechts abgearbeitet wird. Als Folgerung der Definition erhalten wir

Satz 1.2.1 Für alle $z \in Z$, $x \in X$ und $w, w' \in X^*$ gilt

$$\delta^*(z,x) = \delta(z,x) \text{ und } \delta^*(z, w'w) = \delta^*(\delta^*(z,w'), w).$$

Beweis: Die erste Gleichung folgt aus

$$\delta^*(z,x) = \delta^*(z, \varepsilon x) = \delta(\delta^*(z,\varepsilon), x) = \delta(z,x).$$

Die zweite wird durch Induktion über die Länge von w bewiesen. Für $|w| = 0$ gilt $w = \varepsilon$, und aus Definition 1.2.5 folgt dann

$$\delta^*(z, w'\varepsilon) = \delta^*(z,w') = \delta^*(\delta^*(z,w'), \varepsilon)$$

für alle $z \in Z$, $w' \in X^*$. Die Behauptung sei nun für alle $z \in Z$, $w' \in X^*$ und alle Wörter $w_1 \in X^*$ einer festen Länge erfüllt. Dann gilt für $w = w_1 x$ mit $x \in X$

$$\begin{aligned} \delta^*(z, w'w_1x) &= \delta(\delta^*(z, w'w_1), x) \\ &= \delta(\delta^*(\delta^*(z,w'), w_1), x) \\ &= \delta^*(\delta^*(z,w'), w_1x). \end{aligned}$$

Das erste und das letzte Gleichheitszeichen gelten aufgrund Definition 1.2.5, das zweite wegen der Induktionsannahme. □

Der nächste Satz zeigt, daß Moore- und Mealy-Automaten im wesentlichen dasselbe leisten und in diesem Sinn als äquivalent aufgefaßt werden können.

Satz 1.2.2 Es sei $M = (Z, X, Y, \delta, \lambda)$ ein Mealy-Automat. Dann existiert ein Moore-Automat $\bar{M} = (\bar{Z}, X, Y, \bar{\delta}, \beta)$ und eine injektive Abbildung $k : Z \to \bar{Z}$ mit

$$\lambda(\delta^*(z,w), x) = \beta(\bar{\delta}^*(k(z), wx)) \qquad (1)$$

für alle $z \in Z, w \in X^*, x \in X$.

Beweis: Es sei ein Mealy-Automat $M = (Z, X, Y, \delta, \lambda)$ gegeben. Wir definieren $\bar{M} = (\bar{Z}, X, Y, \bar{\delta}, \beta)$ durch

$$\bar{Z} = Z \times Y,$$

$$\bar{\delta} : \bar{Z} \times X \to \bar{Z} \;\text{ mit }\; \bar{\delta}((z,y),x) = (\delta(z,x), \lambda(z,x)) \;\text{ und}$$

$$\beta : \bar{Z} \to Y \;\text{ mit }\; \beta(z,y) = y.$$

Außerdem setzen wir für ein beliebiges, aber festes $y' \in Y$

$$k : Z \to \bar{Z} \;\text{ mit }\; k(z) = (z, y').$$

Zu zeigen ist, daß (1) gilt. Dazu beweisen wir durch vollständige Induktion für alle $z \in Z$, $y \in Y$, $w \in X^*$, $x \in X$ die stärkere Aussage

$$\lambda(\delta^*(z,w), x) = \beta(\bar{\delta}^*((z,y), wx)) \qquad (2).$$

Dabei benutzen wir Definition 1.2.5 und Satz 1.2.1 sowie die Definitionen von β und $\bar{\delta}$. Für den Induktionsbeginn setzen wir $w = \varepsilon$. Dann gilt

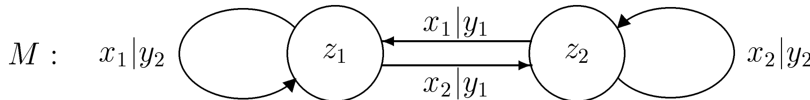
$$\lambda(\delta^*(z,\varepsilon), x) = \lambda(z,x) = \beta(\delta(z,x), \lambda(z,x))$$

$$= \beta(\bar{\delta}^*((z,y),x)) = \beta(\bar{\delta}^*((z,y),\varepsilon x)).$$

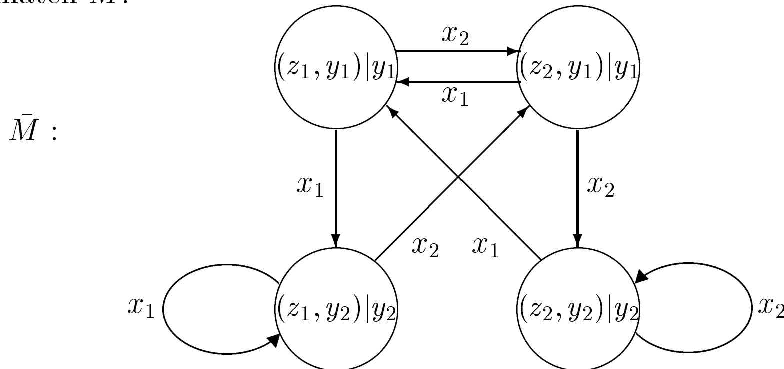
Für den Induktionsschluß nehmen wir an, daß (2) für $w = w_1$ und alle $z \in Z$, $y \in Y$ und $x \in X$ bewiesen ist. Zu zeigen ist, daß (2) auch für $w = x_1 w_1$ mit beliebigem $x_1 \in X$ gilt. Wir erhalten in der Tat

$$\begin{aligned}
\beta(\bar{\delta}^*((z,y),x_1w_1x)) &= \beta(\bar{\delta}^*(\bar{\delta}((z,y),x_1),w_1x)) \\
&= \beta(\bar{\delta}^*((\delta(z,x_1),\lambda(z,x_1)),w_1x)) \\
&= \lambda(\delta^*(\delta(z,x_1),w_1),x) \\
&= \lambda(\delta^*(z,x_1w_1),x). \quad \square
\end{aligned}$$

Beispiel 1.2.2 Wir geben zunächst einen Mealy-Automaten M an:



Zu diesem Automaten konstruieren wir gemäß dem Beweis von Satz 1.2.2 einen Moore-Automaten $\bar{M}$:



Wir wollen die Arbeitsweise der beiden Automaten bei Eingabe von $x_1x_2x_1x_2$ betrachten, wenn wir zur Zeit 0 bei M im Zustand z_1 und bei $\bar{M}$ im Zustand (z_1, y_1) starten. Der Mealy-Automat M liefert zu den Zeiten 0 bis 3 die Ausgaben y_2, y_1, y_1, y_1, der Moore-Automat $\bar{M}$ zu den Zeiten 0 bis 4 die Ausgaben y_1, y_2, y_1, y_1, y_1. Wir erkennen die Verzögerung um einen Zeittakt. $\square$

Definition 1.2.6 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat und $z \in Z$. Die Funktion $M_z : X^* \to Y$ mit $M_z(w) = \beta(\delta^*(z,w))$ für alle $w \in X^*$ heißt *Antwortfunktion* oder *Verhalten* von M beim Anfangszustand $z \in Z$.

Die Antwortfunktion M_z liefert, ausgehend vom Zustand z, das Ausgabesymbol, das sich nach Eingabe des Wortes w ergibt. Die zuvor bei der schrittweisen Abarbeitung der eingegebenen Symbole von w erhaltenen Ausgaben werden dabei nicht berücksichtigt. Man kann jedoch M_z zu einer Funktion $M'_z : X^* \to Y^*$ durch

$$M'_z(\varepsilon) = M_z(\varepsilon) = \beta(z) \quad \text{und} \quad M'_z(wx) = M'_z(w)M_z(wx)$$

für alle $w \in X^*$, $x \in X$ erweitern. Offenbar beschreibt M'_z das gesamte Verhalten von M und nicht nur die letzte Ausgabe. Für den Automaten $\bar{M}$ aus Beispiel 1.2.2 gilt so

$$\bar{M}_{(z_1,y_1)}(x_1x_2x_1x_2) = y_1 \quad \text{und} \quad \bar{M}'_{(z_1,y_1)}(x_1x_2x_1x_2) = y_1y_2y_1y_1y_1.$$

Definition 1.2.7 Eine beliebige Funktion $f : X^* \to Y$ heiße *Verhaltensfunktion*. Das *Realisierungsproblem* ist dann wie folgt definiert: Gegeben sei eine Verhaltensfunktion $f : X^* \to Y$. Gibt es einen Moore-Automaten M und einen Zustand z von M (Anfangszustand) mit $M_z = f$? Falls ein solches Paar (M, z) existiert, heißt es *Realisierung* von f. □

Wir werden sehen, daß die meisten Verhaltensfunktionen $f : X^* \to Y$ keine Realisierung durch einen Automaten mit endlicher Zustandsmenge besitzen. Im folgenden Satz wie auch später wird die Anzahl der Elemente einer Menge X mit $|X|$ bezeichnet.

Satz 1.2.3 Es seien X und Y endliche Mengen mit $|Y| \geq 2$. Dann gibt es eine Verhaltensfunktion $f : X^* \to Y$, für die *keine* Realisierung (M, z) von f existiert.

Beweis: Es genügt zu zeigen, daß mehr Funktionen $f : X^* \to Y$ existieren als Antwortfunktionen von Moore-Automaten mit Eingabemenge X und Ausgabemenge Y. Es sei $w_0, w_1, \ldots$ eine fest gewählte Aufzählung von X^* (Für $X = \{x_1, \ldots, x_n\}$ kann man z.B. die spezielle Aufzählung $\varepsilon \mapsto 0$, $x_{i_m} x_{i_{m-1}} \ldots x_{i_1} \mapsto \sum_{j=1}^{m} i_j n^{j-1}$ wählen. Anschaulich bedeutet das die Auflistung der Wörter aus X^* der Länge nach und bei gleicher Länge in lexikographischer Reihenfolge). Jede Funktion $f : X^* \to Y$ kann bezüglich dieser Aufzählung eindeutig durch die unendliche Folge

$$f(w_0) f(w_1) \ldots \quad \text{mit} \quad f(w_i) \in Y, i = 0, 1, \ldots,$$

dargestellt werden. Gilt $Y = \{y_0, \ldots, y_{N-1}\}$, so ist eine beliebige Folge $y_{i_1} y_{i_2} \ldots$ mit $i_1, i_2, \ldots \in \{0, \ldots, N-1\}$ als N-adische Zahl $0, i_1 i_2 \ldots$ auffaßbar. Da $|Y| \geq 2$ ist, heißt dies, daß jede reelle Zahl r mit $0 \leq r \leq 1$ eine Verhaltensfunktion festlegt (ggf. auch zwei Verhaltensfunktionen, z.B. $0,1 = 0,0999\ldots$ für $N = 10$). Somit ist die Mächtigkeit der Menge der Verhaltensfunktionen gleich der der Menge der reellen Zahlen zwischen 0 und 1. Wir wissen, daß diese Mächtigkeit überabzählbar ist.

Weiter wird gezeigt, daß nur abzählbar viele Funktionen M_z existieren. Es sei $M = (Z, X, Y, \delta, \beta)$ ein endlicher Moore-Automat. Dann hat die Bezeichnung der Zustände offenbar keinen Einfluß auf das Verhalten des Automaten. So können wir die Zustandsmenge für $|Z| = n$ stets mit $\{z_0, \ldots, z_{n-1}\}$ benennen. Dabei sei M_{z_0} immer das Verhalten von M. Da X und Y fest vorgegebene endliche Mengen sind, existieren bis auf die Isomorphie, die durch die verschiedenen Bezeichnungen gegeben ist, nur endlich viele Automaten mit genau n Zuständen. Es sei α_n die Anzahl dieser Automaten. Ihre Auflistung sei $M^{n1}, \ldots, M^{n\alpha_n}$. Wir erhalten dann bis auf Isomorphie die Aufzählung aller Moore-Automaten mit den endlichen Mengen X und Y als Ein- bzw. Ausgabemenge durch

$$\underbrace{M^{11}, \ldots, M^{1\alpha_1}}_{1-Zustandsautom.}, \underbrace{M^{21}, \ldots, M^{2\alpha_2}}_{2-Zustandsautom.}, \underbrace{M^{31}, \ldots, M^{3\alpha_3}}_{3-Zustandsautom.}, \ldots.$$

Damit existieren auch nur abzählbar viele Realisierungen (M, z). □

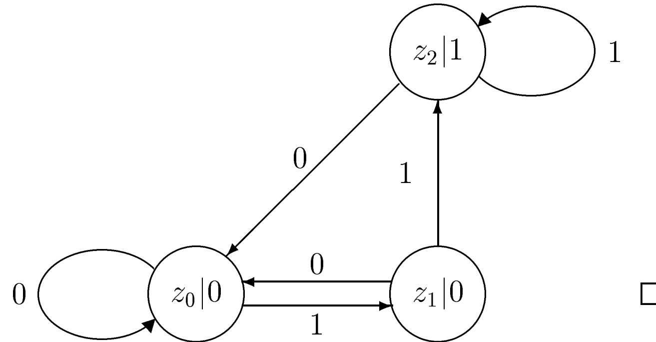
Der Beweis von Satz 1.2.3 zeigt, daß sogar überabzählbar viele Verhaltensfunktionen nicht durch einen endlichen Moore-Automaten realisiert werden können. Die

schwächere Aussage des Satzes 1.2.3 wird auch jeweils durch die folgenden Beispiele 1.2.4 und 1.2.5 bewiesen. Dabei handelt es sich um intuitiv berechenbare Funktionen, die nicht durch einen Moore-Automaten realisierbar sind. Wir sehen somit, daß auch Moore-Automaten noch nicht geeignet sind, alle berechenbaren Probleme zu lösen.

Beispiel 1.2.3 Gegeben sei die Funktion $f : \{0,1\}^* \to \{0,1\}$ mit

$$f(w) = \begin{cases} 1, & \text{falls } w = w'11, w' \in \{0,1\}^* \\ 0 & \text{sonst.} \end{cases}$$

Offenbar wird f durch den folgenden Automaten M mit Anfangszustand z_0 realisiert:



□

Beispiel 1.2.4 Es sei $X = \{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}$ und $Y = \{0,1\}$. Wir betrachten die Verhaltensfunktion $f : X^* \to Y$ mit

$$f(\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}) = \text{ Koeffizient von } 2^{n-1} \text{ im Produkt } (\sum_{j=1}^{n} x_j 2^{j-1})(\sum_{k=1}^{n} y_k 2^{k-1}),$$

wobei $x_1 \dots x_n$ die umgekehrte Binärkodierung von $\sum_{j=1}^{n} x_j 2^{j-1}$ ist. Wir stellen fest, daß $f(\binom{x_1}{y_1})$ den Koeffizienten von 2^0 im angegebenen Produkt liefert, da die Summanden für $j = 2, \dots, n$ bei diesem Koeffizienten nicht berücksichtigt werden. Allgemein wird durch $f(\binom{x_1}{y_1} \cdots \binom{x_i}{y_i})$, $i = 1, \dots, n$, der Koeffizient von 2^{i-1} im Produkt bestimmt. Schließlich wird durch $f(\binom{x_1}{y_1} \cdots \binom{x_n}{y_n} \binom{0}{0}^r)$ der Koeffizient von 2^{n+r-1} berechnet. Wir erkennen somit, daß f die Produktbildung der Dualzahlen $x_n \dots x_1$ und $y_n \dots y_1$ darstellt.

Wir nehmen nun an, daß f das Verhalten eines Moore-Automaten M ist, M also das Produkt von Dualzahlen berechnen kann. M muß X als Eingabemenge und Y als Ausgabemenge besitzen. Ohne Beschränkung der Allgemeinheit sei z_0 ein Zustand mit $M_{z_0} = f$. Der Automat M besitze r Zustände. Es sei $r' > r$, und wir betrachten $2^{r'} 2^{r'} = 2^{2r'}$. Dann muß

$$M_{z_0}(\begin{pmatrix}0\\0\end{pmatrix}^{r'} \begin{pmatrix}1\\1\end{pmatrix} \begin{pmatrix}0\\0\end{pmatrix}^{j}) = \begin{cases} 1, & \text{falls } j = r' \\ 0 & \text{sonst} \end{cases}$$

gelten, da genau für $j = r'$ der Koeffizient von $2^{2r'}$ im Produkt bestimmt wird, der in diesem Fall gleich 1 ist. Wir setzen $z'_j = \delta^*(z_0, \binom{0}{0}^{r'} \binom{1}{1} \binom{0}{0}^j)$. Für $j < r'$ ergibt sich dann

$$M_{z'_j}(\begin{pmatrix}0\\0\end{pmatrix}^{k}) = \beta(\delta^*(z'_j, \begin{pmatrix}0\\0\end{pmatrix}^{k})) = \beta(\delta^*(\delta^*(z_0, \begin{pmatrix}0\\0\end{pmatrix}^{r'} \begin{pmatrix}1\\1\end{pmatrix} \begin{pmatrix}0\\0\end{pmatrix}^{j}), \begin{pmatrix}0\\0\end{pmatrix}^{k}))$$

$$= M_{z_0}(\binom{0}{0}^{r'}\binom{1}{1}\binom{0}{0}^{j+k}) = \begin{cases} 1, & \text{falls } k = r' - j \\ 0 & \text{sonst.} \end{cases}$$

Daraus folgt, daß für alle $j = 0, 1, \ldots, r' - 1$ die Verhaltensfunktionen $M_{z'_j}$ paarweise verschieden sind. Somit sind auch die Zustände $z'_0, \ldots, z'_{r'-1}$ paarweise verschieden, d.h., M besitzt mindestens $r' > r$ Zustände. Dies steht im Widerspruch zur Annahme über die Anzahl der Zustände von M. Das Produkt von zwei beliebigen Zahlen kann also von keinem Moore-Automaten berechnet werden. □

Beispiel 1.2.5 Gegeben sei die Funktion $f : \{a, b\}^* \to \{0, 1\}$ mit

$$f(w) = \begin{cases} 1, & \text{falls } w = a^n b a^n, n \in \mathbb{N} \\ 0 & \text{sonst.} \end{cases}$$

Wir nehmen an, daß es einen Moore-Automaten $M = (Z, X, Y, \delta, \beta)$ gibt mit $|Z| < \infty$ und $M_{z_0} = f$. Ohne Beschränkung der Allgemeinheit gelte $Z = \{z_0, \ldots, z_r\}$, $r \in \mathbb{N}$. Wir nehmen weiter an, daß $i, j \in \mathbb{N}$, $i \neq j$, existieren mit $\delta^*(z_0, a^i) = \delta^*(z_0, a^j)$. Damit erhalten wir

$$\delta^*(z_0, a^i b a^i) = \delta^*(\delta^*(z_0, a^i), b a^i) = \delta^*(\delta^*(z_0, a^j), b a^i) = \delta^*(z_0, a^j b a^i).$$

Nach der Definition von $f = M_{z_0}$ folgt dann jedoch

$$1 = M_{z_0}(a^i b a^i) = \beta(\delta^*(z_0, a^i b a^i)) = \beta(\delta^*(z_0, a^j b a^i)) = M_{z_0}(a^j b a^i) = 0,$$

ein Widerspruch. Es muß also $\delta^*(z_0, a^i) \neq \delta^*(z_0, a^j)$ für alle Paare $i, j \in \mathbb{N}$, $i \neq j$, gelten. Für $|Z| < \infty$ ist diese Bedingung nach dem Schubfachprinzip jedoch nicht erfüllbar. Folglich ist f durch keinen endlichen Moore-Automaten realisierbar. □

1.3 Reduktion von Automaten

Wie bei Schaltnetzen ist man auch bei Moore-Automaten daran interessiert, Verhaltensfunktionen mit möglichst geringem Aufwand zu realisieren. Als Maß der Komplexität kann die Anzahl der Zustände verwendet werden.

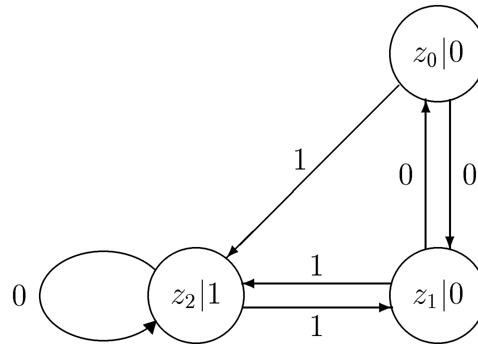
Definition 1.3.1 Es sei $S \neq \emptyset$ eine Menge, auf der eine Relation $\equiv$ erklärt ist. Das bedeutet, daß für je zwei Elemente $x, y \in S$ feststeht, ob $x \equiv y$ gilt oder nicht. Diese Relation heißt *Äquivalenzrelation*, wenn für alle $x, y, z \in S$ die folgenden Bedingungen erfüllt sind:
(a) $x \equiv x$ (Reflexivität).
(b) Aus $x \equiv y$ folgt $y \equiv x$ (Symmetrie).
(c) Aus $x \equiv y$ und $y \equiv z$ folgt $x \equiv z$ (Transitivität).
Dann heißt $[x]_\equiv = \{y \mid y \in S \wedge y \equiv x\}$ die *Äquivalenzklasse von x modulo* $\equiv$ und $S/{\equiv} = \{[x]_\equiv \mid x \in S\}$ die *Faktormenge von S*. Falls keine Verwechslung möglich ist, schreibt man auch $[x]$ statt $[x]_\equiv$. □

Um den Begriff der Äquivalenzklasse zu verdeutlichen, betrachten wir zunächst ein bekanntes Beispiel der Zahlentheorie.

Beispiel 1.3.1 Es sei $\mathbb{Z}$ die Menge der ganzen Zahlen. Für ein festes $k \in \mathbb{N}$ wollen wir eine Äquivalenzrelation $\equiv_k$ auf $\mathbb{Z}$ definieren. Für $a, b \in \mathbb{Z}$ setzen wir $a \equiv_k b$, falls ein $y \in \mathbb{Z}$ existiert mit $a - b = k \cdot y$. Offenbar gilt $a - a = k \cdot 0$, also $a \equiv_k a$. Für $a \equiv_k b$ gibt es nach Definition von $\equiv_k$ ein $y \in \mathbb{Z}$ mit $a - b = k \cdot y$. Dies ist äquivalent zu $b - a = k \cdot (-y)$, und es folgt $b \equiv_k a$. Gilt $a \equiv_k b$ und $b \equiv_k c$, so erhalten wir $a - b = k \cdot y_1$ und $b - c = k \cdot y_2$ für geeignete $y_1, y_2 \in \mathbb{Z}$. Durch Addition der beiden Gleichungen ergibt sich $a - c = k \cdot (y_1 + y_2)$. Nach Definition der Relation $\equiv_k$ bedeutet dies $a \equiv_k c$. Insgesamt ist also $\equiv_k$ eine Äquivalenzrelation auf $\mathbb{Z}$. Wir schreiben statt $a \equiv_k b$ auch $a \equiv b \bmod k$. Mit $\mathbb{Z}_k = \mathbb{Z}_{\equiv_k} = \{[0], [1], \ldots, [k-1]\}$ bezeichnen wir die Menge der Äquivalenzklassen modulo k. $\square$

Das folgende Beispiel dient der Vorbereitung des Reduktionsbegriffs von Moore-Automaten.

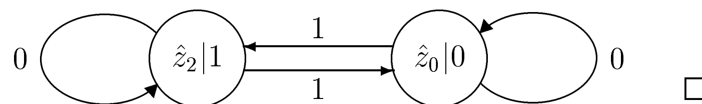
Beispiel 1.3.2 Wir geben einen Moore-Automaten M an, dessen Verhaltensfunktionen auch durch einen anderen Moore-Automaten $\hat{M}$ mit weniger Zuständen zu berechnen sind. Der Automat M soll also auf einen Automaten $\hat{M}$ mit demselben Verhalten reduziert werden. Wir betrachten M:



Es gilt $M_z = \beta \circ \delta^*(z, -)$ (siehe Definition 1.2.6). Wir erhalten

$$M_{z_0}(w) = \begin{cases} 1, & \text{falls } w \text{ eine ungerade Anzahl des Symbols 1 enthält} \\ 0 & \text{sonst} \end{cases}$$

sowie $M_{z_1} = M_{z_0}$ und $M_{z_2} = 1 - M_{z_0}$. Obwohl drei Zustände vorhanden sind, gibt es nur zwei verschiedene Verhaltensfunktionen. Daher ist es naheliegend, Zustände mit dem gleichen Verhalten zu verschmelzen. Dies sind die Zustände z_0 und z_1. Der so gewonnene reduzierte Automat $\hat{M}$ ist durch den folgenden Zustand-Ausgabe-Graphen bestimmt:



$\square$

Diese in Beispiel 1.3.2 durchgeführte Reduktion wollen wir allgemein untersuchen. Unser Ziel ist es, für jeden Automaten M einen reduzierten Automaten $\hat{M}$ zu konstruieren, der unter allen Automaten, die dieselben Verhaltensfunktionen wie M und $\hat{M}$ haben, eine minimale Anzahl von Zuständen besitzt.

Definition 1.3.2 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat, und es seien $z_1, z_2 \in Z$. z_1 heißt *äquivalent zu* z_2 ($z_1 \sim z_2$), wenn $M_{z_1} = M_{z_2}$ gilt. □

Es ist sofort einzusehen, daß „$\sim$" eine Äquivalenzrelation ist.

Satz 1.3.1 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat, und $\sim$ sei die Relation gemäß Definition 1.3.2. Falls $z_1 \sim z_2$ gilt, dann folgt $\delta(z_1, x) \sim \delta(z_2, x)$ und $\beta(z_1) = \beta(z_2)$ für alle $x \in X$.

Beweis: Die Relation $z_1 \sim z_2$ ist nach Definition 1.3.2 äquivalent zu $M_{z_1} = M_{z_2}$ und damit auch zu $M_{z_1}(w) = M_{z_2}(w)$ für alle $w \in X^*$. Zum Beweis von $\delta(z_1, x) \sim \delta(z_2, x)$ wählen wir ein beliebiges $x \in X$. Dann gilt $M_{z_1}(xw') = M_{z_2}(xw')$ für alle $w' \in X^*$. Mit Hilfe von Satz 1.2.1 ergibt sich damit

$$\begin{aligned} M_{\delta(z_1,x)}(w') &= \beta(\delta^*(\delta(z_1, x), w')) = \beta(\delta^*(z_1, xw')) \\ &= M_{z_1}(xw') = M_{z_2}(xw') = \beta(\delta^*(z_2, xw')) \\ &= \beta(\delta^*(\delta(z_2, x), w')) = M_{\delta(z_2,x)}(w') \end{aligned}$$

für alle $w' \in X^*$. Folglich ist $\delta(z_1, x) \sim \delta(z_2, x)$ erfüllt. Außerdem erhalten wir

$$\beta(z_1) = \beta(\delta^*(z_1, \varepsilon)) = M_{z_1}(\varepsilon) = M_{z_2}(\varepsilon) = \beta(\delta^*(z_2, \varepsilon)) = \beta(z_2). \quad \square$$

Dieser Satz erlaubt uns, die folgende Definition anzugeben.

Definition 1.3.3 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat und $\sim$ die Relation gemäß Definition 1.3.2. Dann heißt

$$\hat{M} = (\hat{Z}, X, Y, \hat{\delta}, \hat{\beta}) \text{ mit } \hat{Z} = Z/\sim$$

der *zu M reduzierte Moore-Automat*, wenn die Abbildungen $\hat{\delta} : \hat{Z} \times X \to \hat{Z}$ und $\hat{\beta} : \hat{Z} \to Y$ durch

$$\hat{\delta}([z], x) = [\delta(z, x)] \text{ und } \hat{\beta}([z]) = \beta(z)$$

für $z \in Z$, $x \in X$ gegeben sind. □

Durch $[z]$ wird die Äquivalenzklasse von z modulo $\sim$ bezeichnet. Aufgrund von Satz 1.3.1 gelten für $z_1 \neq z_2$ mit $[z_1] = [z_2]$ die Gleichungen $\beta(z_1) = \beta(z_2)$ und $[\delta(z_1, x)] = [\delta(z_2, x)]$ für alle $x \in X$, so daß die Angabe von $\hat{\delta}$ und $\hat{\beta}$ in Definition 1.3.3 wohldefiniert ist.

Satz 1.3.2 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat und $\hat{M} = (\hat{Z}, X, Y, \hat{\delta}, \hat{\beta})$ der zu M reduzierte Automat. Dann gilt $\hat{M}_{[z]} = M_z$ für alle $z \in Z$.

Beweis: Wir müssen zeigen, daß $\hat{M}_{[z]}(w) = M_z(w)$ für alle $w \in X^*$ gilt. Der Beweis erfolgt durch vollständige Induktion über $|w|$. Für $w = \varepsilon$ gilt

$$\hat{M}_{[z]}(\varepsilon) = \hat{\beta}([z]) = \beta(z) = M_z(\varepsilon).$$

Weiter sei $w' = xw$ mit $x \in X$, $w \in X^*$. Dann erhalten wir unter Benutzung von Satz 1.2.1 und der Induktionsvoraussetzung

$$\begin{aligned} \hat{M}_{[z]}(xw) &= \hat{\beta}(\hat{\delta}^*([z], xw)) = \hat{\beta}(\hat{\delta}^*(\hat{\delta}([z], x), w)) \\ &= \hat{\beta}(\hat{\delta}^*([\delta(z,x)], w)) = \hat{M}_{[\delta(z,x)]}(w) = M_{\delta(z,x)}(w) \\ &= \beta(\delta^*(\delta(z,x), w)) = \beta(\delta^*(z, xw)) = M_z(xw). \quad \square \end{aligned}$$

Dieser Satz sagt aus, daß ein Moore-Automat und der zu ihm reduzierte Automat dieselben Verhaltensfunktionen berechnen und somit dasselbe leisten. In diesem Sinn sind sie als äquivalent aufzufassen. Aus Satz 1.3.2 folgt sofort

Satz 1.3.3 : Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat und $\hat{M} = (\hat{Z}, X, Y, \hat{\delta}, \hat{\beta})$ der zu M reduzierte Automat. In $\hat{M}$ sind keine verschiedenen Zustände äquivalent.

Beweis: Aus $\hat{M}_{[z_1]} = \hat{M}_{[z_2]}$ folgt nach Satz 1.3.2 die Gleichung $M_{z_1} = M_{z_2}$. Nach Definition 1.3.2 ist also $z_1 \sim z_2$, und somit gilt $[z_1] = [z_2]$. $\square$

Satz 1.3.2 und Satz 1.3.3 gelten auch für Moore-Automaten mit unendlicher Zustandszahl.

Die Definition 1.3.3 ist nicht unmittelbar geeignet, den reduzierten Automaten $\hat{M}$ eines gegebenen Moore-Automaten M zu konstruieren. Beim Übergang von M zu $\hat{M}$ müssen die Zustände verschmolzen werden, die äquivalent sind, die also nach Definition 1.3.2 nicht unterscheidbar sind bezüglich beliebig langer Wörter. Um einen Algorithmus zur Reduktion von Moore-Automaten anzugeben, müssen wir diesen unendlichen Test auf ein endliches Problem zurückführen. Wir betrachten daher zunächst solche Zustände, die für Wörter der Länge $\leq k$ nicht unterscheidbar sind.

Definition 1.3.4 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat, $k \in \mathbb{N}_0$ und $z_1, z_2 \in Z$. z_1 und z_2 heißen *k-äquivalent* ($z_1 \sim_k z_2$), wenn $M_{z_1}(w) = M_{z_2}(w)$ für alle $w \in X^*$ mit $|w| \leq k$ gilt. $\square$

Aus der Definition ergibt sich als Folgerung

Satz 1.3.4 : Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat.

(a) Es gilt $z_1 \sim_0 z_2$ genau dann, wenn $\beta(z_1) = \beta(z_2)$ ist.

(b) Es seien $k, k' \in \mathbb{N}_0$, $k' \geq k$. Aus $z_1 \sim_{k'} z_2$ folgt dann $z_1 \sim_k z_2$.

(c) Es sei $k \in \mathbb{N}_0$ und $Z_k = Z/\sim_k$, wobei die Elemente von Z_k durch $[z]_k = \{z' \mid z' \sim_k z\}$ definiert sind. Für $k' \geq k$ läßt sich eine surjektive Abbildung $Z_{k'} \to Z_k$ durch $[z]_{k'} \mapsto [z]_k$ festlegen.

Beweis: (a) und (b) sind unmittelbar einsichtig. Nach (b) ist dann aber auch die Festlegung der Abbildung in (c) wohldefiniert und surjektiv. $\square$

Satz 1.3.5 Es gilt $z_1 \sim_{k+1} z_2$ genau dann, wenn $z_1 \sim_k z_2$ sowie $\delta(z_1, x) \sim_k \delta(z_2, x)$ für alle $x \in X$ gilt.

Beweis: $z_1 \sim_{k+1} z_2$ ist äquivalent mit $M_{z_1}(w) = M_{z_2}(w)$ für alle $w \in X^*$, $|w| \leq k+1$. Dies ist damit gleichwertig, daß $M_{z_1}(w') = M_{z_2}(w')$ und $M_{z_1}(xw') = M_{z_2}(xw')$ für alle $x \in X$, $w' \in X^*$, $|w'| \leq k$ gilt, was offenbar wegen

$$M_{z_i}(xw') = \beta(\delta^*(z_i, xw')) = \beta(\delta^*(\delta(z_i, x), w')) = M_{\delta(z_i,x)}(w'), i = 1, 2,$$

genau dann der Fall ist, wenn $z_1 \sim_k z_2$ sowie $\delta(z_1, x) \sim_k \delta(z_2, x)$ für alle $x \in X$ erfüllt ist. □

Satz 1.3.6 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat. Aus $Z_j = Z_{j+1}$ für ein $j \in \mathbb{N}_0$ folgt $Z_j = Z_k$ für alle $k \geq j$ und somit $Z_j = \hat{Z}$.

Beweis: Es reicht zu zeigen, daß aus $Z_j = Z_{j+1}$ die Gleichung $Z_{j+1} = Z_{j+2}$ folgt. Nach Satz 1.3.4 (b) wissen wir, daß sich $z_1 \sim_{j+1} z_2$ aus $z_1 \sim_{j+2} z_2$ ergibt. Umgekehrt sei nun $z_1 \sim_{j+1} z_2$. Nach Satz 1.3.5 erhalten wir $z_1 \sim_j z_2$ sowie $\delta(z_1, x) \sim_j \delta(z_2, x)$ für alle $x \in X$. Da $Z_j = Z_{j+1}$ gilt, können wir die Relation $\sim_j$ durch $\sim_{j+1}$ ersetzen und auf die Gültigkeit von $z_1 \sim_{j+1} z_2$ sowie $\delta(z_1, x) \sim_{j+1} \delta(z_2, x)$ für alle $x \in X$ schließen. Nach Satz 1.3.5 folgt dann $z_1 \sim_{j+2} z_2$. □

Satz 1.3.7 Es sei $M = (Z, X, Y, \delta, \beta)$ ein Moore-Automat und $\hat{M} = (\hat{Z}, X, Y, \hat{\delta}, \hat{\beta})$ der zu M reduzierte Moore-Automat. Es gelte $|\hat{Z}| \leq n$ und $n \geq 2$. Dann ist $Z_{n-2} = \hat{Z}$.

Beweis: Es sei zunächst $Z_1 = Z_0 = \hat{Z}$. Wegen $n - 2 \geq 0$ gilt dann $Z_{n-2} = Z_0 = \hat{Z}$. Im weiteren betrachten wir den Fall $Z_1 \neq Z_0$. Es sei j das größte $j \in \mathbb{N}$ mit $Z_j \neq Z_{j-1}$. Nach Satz 1.3.6 erhalten wir $Z_j = \hat{Z}$. Zu zeigen ist $j \leq n - 2$. Wegen $Z_1 \neq Z_0$ ist $|Z_1| \geq |Z_0| + 1$, da mindestens eine Äquivalenzklasse aufgespalten wird. Allgemein folgt dann

$$|Z_j| \geq |Z_{j-1}| + 1 \geq |Z_0| + j.$$

Aus $n \geq |\hat{Z}| = |Z_j|$ ergibt sich $n \geq |Z_0| + j$, also $j \leq n - |Z_0|$. Falls $|Z_0| = 1$ ist, haben alle Zustände die gleiche Ausgabe. Sie sind somit äquivalent, und es folgt $Z_0 = \hat{Z} = Z_1$, ein Widerspruch zu $Z_0 \neq Z_1$. Es muß also $|Z_0| \geq 2$ gelten. Dafür erhalten wir sofort $j \leq n - 2$. □

Die Sätze 1.3.5 bis 1.3.7 dienen als Grundlage des Reduktionsalgorithmus. Aufgrund von Satz 1.3.7 endet er spätestens mit der Konstruktion von Z_{n-2}.

Reduktionsalgorithmus: (endet mit Schritt $n - 2$, falls $|\hat{Z}| \leq n$)

Schritt 0:

Konstruktion von Z_0: Alle Zustände mit der gleichen Ausgabe kommen in jeweils dieselbe Klasse (die Elemente einer solchen Klasse sind 0-äquivalent). Falls $|Z_0| = |Z|$ ist, haben alle Zustände paarweise verschiedene Ausgaben. Dann gilt $Z = \hat{Z}$, und der Algorithmus stoppt. Anderenfalls folgt Schritt 1.

Schritt $r + 1$:

Konstruktion von Z_{r+1}: Es seien $S_1^{(r)}, \ldots, S_p^{(r)}$ die Elemente von Z_r, also Klassen von r-äquivalenten Elementen von Z. Es werden nach Satz 1.3.5 solche $S_j^{(r)}$ weiter unterteilt, die Zustände z_i, z_k enthalten mit $\delta(z_i, x) \in S_l^{(r)}$, $\delta(z_k, x) \in$

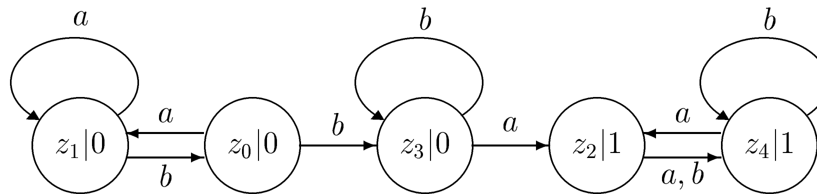
$S_m^{(r)}$, $l \neq m$, für ein $x \in X$. Die Unterteilung erfolgt so, daß diese Zustände in verschiedene neue Klassen kommen. Falls kein $S_j^{(r)}$ unterteilt wird, gilt nach Satz 1.3.6 $\hat{Z} = Z_r$, und der Algorithmus stoppt. Anderenfalls bilden die neuen Klassen $T_1^{(r+1)}, \ldots, T_{p'}^{(r+1)}$, deren Elemente $(r+1)$-äquivalent sind, ein neues Z_{r+1}, und es folgt Schritt $r+2$. □

Hat der zu reduzierende Automat n Zustände, so besitzt auch der reduzierte Automat höchstens n Zustände. Der Algorithmus benötigt somit höchstens $n-2+1 = n-1$ Schritte. Man kann zeigen, daß der reduzierte Automat $\hat{M}$ von M unter allen Automaten, die dieselbe Verhaltensfunktion wie M und $\hat{M}$ haben, eine minimale Anzahl von Zuständen besitzt.

Beispiel 1.3.3 Wir betrachten den Moore-Automaten

$$M = (\{z_0, z_1, z_2, z_3, z_4\}, \{a, b\}, \{0, 1\}, \delta, \beta),$$

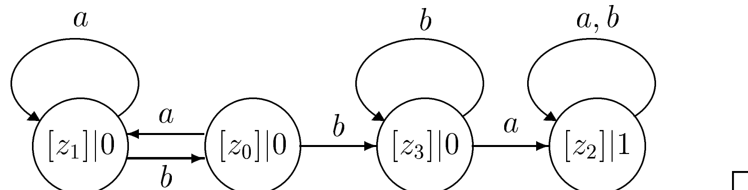
wobei sich δ und β aus dem folgenden Zustand-Ausgabe-Graphen ablesen lassen.



Der Reduktionsalgorithmus liefert

$$\begin{aligned} Z_0 &= \{\{z_0, z_1, z_3\}, \{z_2, z_4\}\}, \\ Z_1 &= \{\{z_0, z_1\}, \{z_3\}, \{z_2, z_4\}\}, \\ Z_2 &= \{\{z_0\}, \{z_1\}, \{z_3\}, \{z_2, z_4\}\}, \\ Z_3 &= Z_2. \end{aligned}$$

Für alle $k \geq 2$ folgt dann $Z_k = Z_2 = \hat{Z}$. Wir erkennen, daß die Zustände z_0, z_1 und z_3 von M jeweils zu keinem anderen Zustand äquivalent sind. Dies ergibt sich aber erst nach drei Schritten des Algorithmus. Die Zustände z_2 und z_4 sind dagegen äquivalent und werden zu einem neuen Zustand verschmolzen. Bei Verwendung der Repräsentantenschreibweisen $[z_0] = \{z_0\}$, $[z_1] = \{z_1\}$, $[z_3] = \{z_3\}$ und $[z_2] = \{z_2, z_4\}$ wird der reduzierte Automat $\hat{M}$ von M durch den folgenden Graphen dargestellt:



□

Zum Abschluß wollen wir noch einige Überlegungen zum Aufwand des obigen Reduktionsalgorithmus durchführen. Es sei $\mathbb{R}^+$ die Menge der reellen Zahlen ≥ 0.

Definition 1.3.5 Es seien $f, g : \mathbb{N}_0 \to \mathbb{R}^+$ Funktionen. Es gilt $f(n) = O(g(n))$ (kurz $f = O(g)$), wenn $c \in \mathbb{R}$ mit $c > 0$ und ein $n_0 \in \mathbb{N}$ existieren, so daß für alle $n \in \mathbb{N}$ mit $n \geq n_0$ die Ungleichung

$$f(n) \leq c \cdot g(n)$$

erfüllt ist. □

Wir nehmen an, daß n die Anzahl der Zustände des gegebenen Automaten ist. Zu jedem der Schritte des Algorithmus sind $p \leq n$ Klassen $S_j^{(r)}$ zu überprüfen. In jeder Klasse werden höchstens $|X| \cdot n^2$ Zustände verglichen. Es werden aber auch höchstens $n - 1$ Schritte des Algorithmus durchgeführt. Das bedeutet, daß bei festem X insgesamt $O(n^4)$ Vergleiche vorgenommen werden. Dies ist offensichtlich eine sehr großzügige „worst-case“-Abschätzung. Auf jeden Fall wird der Rechenaufwand durch ein Polynom in n nach oben abgeschätzt.

Mit diesen Überlegungen sind wir kurz auf ein Komplexitätsmaß für Algorithmen eingegangen, nämlich auf die Anzahl der Einzelschritte bei einer Berechnung. Dieses steht natürlich mit dem Komplexitätsmaß der Zeit in engem Zusammenhang. Ein anderes Komplexitätsmaß ist zum Beispiel der Speicherplatz, den man zur Ausführung eines Algorithmus benötigt. Auf Fragen der Komplexität werden wir in der Vorlesung „Theoretische Informatik II“ (ab Kapitel 6) noch ausführlich eingehen.

2 Turingmaschinen

Wir haben im vorangegangenen Kapitel gesehen, daß Mealy- und Moore-Automaten nur beschränkte Fähigkeiten haben. So gibt es z.B. keinen Automaten, der das Produkt zweier beliebiger Dualzahlen berechnen kann (Beispiel 1.2.4). Auch andere einfache Probleme können von Moore- oder Mealy-Automaten nicht bearbeitet werden. Um solche Probleme zu lösen, sind allgemeinere Rechnermodelle erforderlich. Wir werden in diesem Kapitel Turingmaschinen betrachten. Sie stellen eines der verschiedenen äquivalenten Modelle dar, die den Begriff der Berechenbarkeit formalisieren. Ein weiteres solches Modell sind die μ-rekursiven Funktionen, die wir in Kapitel 3 kennenlernen werden.

2.1 Definitionen

Definition 2.1.1 Ein 4-Tupel $T = (Z, X, \delta, z_0)$ heißt *Turingmaschine*, wenn

(a) Z eine endliche nichtleere Menge (*Zustandsmenge*),

(b) X eine endliche nichtleere Menge (*Bandalphabet, Eingabealphabet*),

(c) $\delta : Z \times \bar{X} \to Z \times (\bar{X} \cup \{l, r, s\})$ eine Abbildung (*lokale Überführungsfunktion, Überführungsabbildung*) mit $\bar{X} = X \cup \{b\}$ ist, wobei $b \notin X$ und $\bar{X} \cap \{l, r, s\} = \emptyset$ gelten ($b \neq \varepsilon$ heißt *Blankzeichen*, *Leerzeichen* oder *uneigentliches Symbol*, l, r oder s symbolisieren *Linksbewegung, Rechtsbewegung* oder *Stopp*), und

(d) $z_0 \in Z$ der *Startzustand* oder *Anfangszustand* ist. □

Wir sagen, daß die Turingmaschine aus Definition 2.1.1 *vollständig* ist, da zu jedem Zustand z und zu jedem Bandsymbol x ein Wert unter der Abbildung δ festgelegt und somit eine Überführung bestimmt ist. Da es auch nur genau einen Wert $\delta(z, x)$ gibt, ist sie auch *deterministisch*. Im Anschluß an Definition 2.1.2 wird die Arbeitsweise anschaulich beschrieben.

Definition 2.1.1 ist eine von vielen Möglichkeiten, eine Turingmaschine anzugeben. Es gibt Modifikationen, die nicht mehr vollständig oder deterministisch sind oder eine andere Anzahl von Bändern haben. Bezüglich ihrer prinzipiellen Fähigkeiten sind jedoch diese verschiedenen Modelle gleich. In Abschnitt 2.3 werden wir einige von ihnen betrachten.

Definition 2.1.2 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine. Die *Turingtafel* von T wird wie folgt gegeben:

(a) Es sei $X = \{a_1, \ldots, a_m\}$. Für alle $z \in Z$ bilde man eine vierspaltige, $(m+1)$-zeilige Matrix

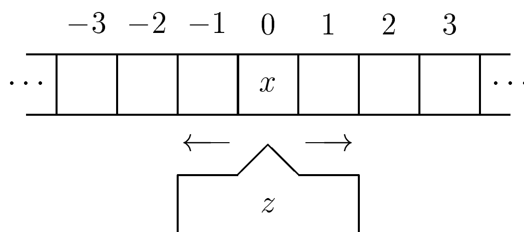
$$\begin{matrix} z & b & z_{i_0} & a'_0 \\ z & a_1 & z_{i_1} & a'_1 \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ z & a_m & z_{i_m} & a'_m \end{matrix} .$$

Dabei gelte $z_{i_j} \in Z$, $0 \le j \le m$, $a'_k \in \bar{X} \cup \{l, r, s\}$, $0 \le k \le m$, mit $\delta(z, b) = (z_{i_0}, a'_0)$ und $\delta(z, a_k) = (z_{i_k}, a'_k)$.

(b) Die Turingtafel von T besteht aus allen $|Z|$ untereinander geschriebenen Matrizen nach (a), wobei die erste die durch den Startzustand z_0 definierte Matrix ist. □

Offenbar ist die Beschreibung einer Turingmaschine durch das 4-Tupel $T = (Z, X, \delta, z_0)$ äquivalent zu der durch eine Turingtafel.

Man kann eine Turingmaschine interpretieren als ein Gerät, das einen Lese- und Schreibkopf hat sowie ein Band, das in numerierte Felder unterteilt und nach beiden Seiten unendlich ist. Der Lese- und Schreibkopf steht jeweils über einem Feld (Arbeitsfeld) des Bandes, das mit einem Symbol $x \in \bar{X}$ beschrieben ist.



Befindet sich die Turingmaschine im Zustand z, so kann ihre nächste Aktion aus der mit $z\ x$ beginnenden Zeile der Turingtafel entnommen werden. Die Zeile $z\ x\ z'\ a$ der Matrix bedeutet, daß beim Lesen von x im Zustand z die Turingmaschine in den Zustand z' übergeht und dabei

- für $a \in \bar{X}$ das Arbeitsfeld mit a beschreibt,
- für $a = l$ einen Schritt nach links geht,
- für $a = r$ einen Schritt nach rechts geht,
- für $a = s$ stoppt und die Arbeit beendet.

Die Arbeit der Turingmaschine beginnt im Startzustand z_0. Zur formalen Beschreibung der Arbeitsweise dienen die folgenden Definitionen. Zunächst charakterisieren wir die Beschriftungen der Felder.

Definition 2.1.3 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine. Eine Abbildung $\beta : \mathbb{Z} \to \bar{X}$ heißt eine *Bandfunktion* der Turingmaschine T. Dabei ist $\beta(x)$ der *Inhalt des Feldes* x und $\Gamma(\beta) = \{(k, \beta(k)) \mid k \in \mathbb{Z}\}$ die *Bandinschrift.* □

$\Gamma(\beta)$ ist der Graph der Bandfunktion β. Im folgenden werden wir nur solche Bandfunktionen betrachten, für die

$$|\{k \mid k \in \mathbb{Z} \wedge \beta(k) \neq b\}| < \infty$$

gilt. Das bedeutet, daß bei allen Bändern von Turingmaschinen nur endlich viele Felder mit Zeichen beschriftet sind, die verschieden von dem Blankzeichen b sind.

Definition 2.1.4 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine und β eine Bandfunktion von T.

(a) $K = (n, \beta, z)$ heißt *Konfiguration von* T, wenn $n \in \mathbb{Z}$ (Nummer des Arbeitsfeldes), β eine Bandfunktion von T und $z \in Z$ ist.

(b) $K = (n, \beta, z)$ heißt *Anfangskonfiguration*, wenn $z = z_0$ und $n = 0$ ist.
(c) $K = (n, \beta, z)$ heißt *Endkonfiguration*, wenn $z\ \beta(n)\ z'\ s$ eine Zeile der Turingtafel von T ist mit $z, z' \in Z$. □

Aus der Anfangskonfiguration ergibt sich die Beschriftung des Bandes und der Zustand z_0 zu Beginn der Arbeit der Turingmaschine. Man beachte, daß bei einer Endkonfiguration (n, β, z) noch ein weiterer Zustandsübergang nach z' erfolgt und danach die Arbeit beendet wird. Die Ausführung eines Schrittes der Turingmaschine, d.h. die einmalige Anwendung der Abbildung δ, bewirkt den Übergang von einer Konfiguration in die Folgekonfiguration:

Definition 2.1.5 Es sei T eine Turingmaschine und $K = (n, \beta, z)$ eine Konfiguration von T. $K' = (n', \beta', z')$ heißt *Folgekonfiguration von K*, wenn gilt:

(a)
$$n' = \begin{cases} n, & \text{falls } z\ \beta(n)\ z'\ d \text{ eine Zeile der Turingtafel ist} \\ & \quad \text{mit } d \neq r, d \neq l \\ n+1, & \text{falls } z\ \beta(n)\ z'\ r \text{ eine Zeile der Turingtafel ist} \\ n-1, & \text{falls } z\ \beta(n)\ z'\ l \text{ eine Zeile der Turingtafel ist.} \end{cases}$$

(b)
$$\begin{cases} \beta'(n) = v \text{ und } \beta'(m) = \beta(m) \text{ für } m \neq n, & \text{falls } z\ \beta(n)\ z'\ v \text{ Zeile der} \\ & \text{Turingtafel ist mit} \\ & v \notin \{l, r, s\} \\ \beta' = \beta & \text{sonst.} \end{cases}$$

(c) z' ergibt sich als dritte Komponente der einzigen mit $z\ \beta(n)$ beginnenden Zeile der Turingtafel von T. □

Eine Folgekonfiguration ist, da eine Turingmaschine nach Definition 2.1.1 vollständig und deterministisch ist, immer eindeutig bestimmt. Bei Angabe einer Anfangskonfiguration ist die Arbeitsweise einer Turingmaschine durch die sich daraus nacheinander ergebenden Folgekonfigurationen gekennzeichnet.

Nur in endlich vielen Feldern des Bandes einer Turingmaschine stehen zu jedem Zeitpunkt der Arbeit Zeichen, die keine Blankzeichen sind. Trotzdem können im Laufe der Zeit beliebig viele Felder mit Nichtblankzeichen beschrieben werden. Manchmal wird diese „potentielle Unendlichkeit“ des Bandes dadurch hervorgehoben, daß man annimmt, daß das Band eine endliche Länge hat und es nur bei Bedarf an den Enden um jeweils ein mit dem Blankzeichen beschriebenes Feld erweitert wird.

2.2 Beispiele für Turingmaschinen und ihre Zusammensetzbarkeit

Wir werden jetzt einige einfache Turingmaschinen angeben. Zum einen wollen wir dadurch die Arbeitsweise von Turingmaschinen besser kennenlernen und verstehen, zum anderen werden wir diese Turingmaschinen später bei komplexeren Problemen benötigen.

Beispiel 2.2.1 Die *Linksmaschine* $\mathbf{l} = (\{z_0, z_1\}, X, \delta, z_0)$ mit der Turingtafel

$$\begin{array}{cccc} z_0 & b & z_1 & l \\ z_0 & a_1 & z_1 & l \\ & \cdot & & \\ & \cdot & & \\ z_0 & a_n & z_1 & l \\ z_1 & b & z_1 & s \\ & \cdot & & \\ & \cdot & & \\ z_1 & a_n & z_1 & s \end{array}$$

bewegt den Kopf einen Schritt nach links, ändert dabei den Zustand und bleibt dann stehen. Die *Rechtsmaschine* $\mathbf{r} = (\{z_0, z_1\}, X, \delta, z_0)$ ist entsprechend definiert. Dabei wird in der Tafel die Linksbewegung l in der letzten Spalte durch die Rechtsbewegung r ersetzt. □

Beispiel 2.2.2 Es sei $X = \{a_1, \ldots, a_n\}$ und $a_0 = b$. Für jedes i, $i = 0, \ldots, n$, definieren wir die *Schreibmaschine* $\mathbf{a}_i = (\{z_0\}, X, \delta, z_0)$ durch die Turingtafel

$$\begin{array}{cccc} z_0 & b & z_0 & a_i \\ z_0 & a_1 & z_0 & a_i \\ & \cdot & & \\ & \cdot & & \\ z_0 & a_{i-1} & z_0 & a_i \\ z_0 & a_i & z_0 & s \\ z_0 & a_{i+1} & z_0 & a_i \\ & \cdot & & \\ & \cdot & & \\ z_0 & a_n & z_0 & a_i. \end{array}$$

Sie beschreibt offenbar das Arbeitsfeld mit a_i und bleibt dann stehen. Man beachte, daß es auch die Schreibmaschine **b** gibt. □

Turingmaschinen können sich gegenseitig simulieren. Dies wird in der folgenden Definition exakt ausgedrückt.

Definition 2.2.1 Es seien T_1 und T_2 Turingmaschinen. T_1 und T_2 heißen *äquivalent*, wenn folgendes gilt: Beginnen T_1 und T_2 mit der gleichen beliebigen Bandinschrift $\Gamma(\beta)$ und auf dem gleichen Anfangsfeld i, so erreicht T_1 genau dann eine Endkonfiguration $(j, \Gamma(\beta'), z')$, wenn T_2 eine Endkonfiguration $(j, \Gamma(\beta'), z'')$ erreicht. □

Beispiel 2.2.3 Die Turingmaschine T_1, die durch $\bar{X} = \{b, a_1\}$ und die Turingtafel

$$\begin{array}{cccc} z_0 & b & z_0 & a_1 \\ z_0 & a_1 & z_1 & r \\ z_1 & b & z_1 & s \\ z_1 & a_1 & z_1 & s \end{array}$$

gegeben ist, geht beim Lesen des Zeichens a_1 einen Schritt nach rechts und stoppt. Beim Lesen des Zeichens b beschreibt sie das Band mit a_1, geht anschließend einen Schritt nach rechts und stoppt dann. Diese Aktion wird auch dadurch bewirkt, daß man die Schreibmaschine $\mathbf{a}_1$ und die Rechtsmaschine $\mathbf{r}$ hintereinander ausführt. Schreibt man die Turingtafeln von $\mathbf{a}_1$ und $\mathbf{r}$ untereinander, also

$$\begin{array}{ll} \mathbf{a}_1 \left\{ \begin{array}{cccc} z'_0 & b & z'_0 & a_1 \\ z'_0 & a_1 & z'_0 & s \end{array} \right. & \longleftarrow \quad z'_0 \quad a_1 \quad z'_1 \quad a_1 \\ \mathbf{r} \left\{ \begin{array}{cccc} z'_1 & b & z'_2 & r \\ z'_1 & a_1 & z'_2 & r \\ z'_2 & b & z'_2 & s \\ z'_2 & a_1 & z'_2 & s, \end{array} \right. & \end{array}$$

und ersetzt dabei die Zeile $z'_0\ a_1\ z'_0\ s$ durch die Zeile $z'_0\ a_1\ z'_1\ a_1$, so entsteht die Turingtafel der gesuchten Maschine. Diese Turingmaschine notieren wir als $T_2 = \mathbf{a}_1\mathbf{r}$. Offenbar sind die Turingmaschinen T_1 und T_2 äquivalent.

Man beachte dabei, daß wir die Zustände von $\mathbf{a}_1$ und $\mathbf{r}$ aus Beispiel 2.2.1 und Beispiel 2.2.2 abgeändert haben. Die Umbenennung der Zustände einer Turingmaschine ist immer möglich, ohne daß sich dabei die Arbeitsweise der Maschine ändert. □

Die in Beispiel 2.2.3 durchgeführte Komposition zweier Turingmaschinen zu einer neuen kann allgemein durchgeführt werden.

Definition 2.2.2 : (Aufbau größerer Turingmaschinen) Es seien T_1, T_2 Turingmaschinen über $X = \{a_1, \ldots, a_n\}$. Man setze $a_0 = b$. Für $a_k \in \bar{X}$, $k = 0, \ldots, n$, erhält man eine neue Turingmaschine

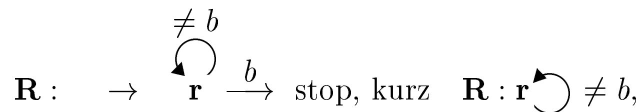
$$T_1 \stackrel{a_k}{\rightarrow} T_2$$

als Komposition der Turingmaschinen T_1 und T_2 durch die folgende Konstruktion ihrer Turingtafel: Die Tafeln von T_1 und T_2 werden untereinander geschrieben, wobei die Zeilen der Form $z\ a_k\ z'\ s$ aus der Tafel von T_1 ersetzt werden durch $z\ a_k\ z'_0\ a_k$ mit dem Anfangszustand z'_0 von T_2. Dabei müssen ggf. die Zustände von einer der beiden Turingmaschinen so umbezeichnet werden, daß die Zustandsmengen von T_1 und T_2 disjunkt sind. □

Würde die Maschine T_1 beim Lesen eines mit a_k beschrifteten Feldes stoppen, so wird bei der Maschine $T_1 \stackrel{a_k}{\rightarrow} T_2$ der Ablauf von T_1 an T_2 übergeben. Wir können den Pfeil zwischen T_1 und T_2 weglassen, wenn eine Übergabe für alle $x \in \bar{X}$ möglich ist. Um Mißverständnisse zu vermeiden, kann die Teil-Turingmaschine, mit der die Arbeit beginnt, durch einen auf sie weisenden unmarkierten Pfeil gekennzeichnet werden (s.u., Beispiele 2.2.4 und 2.2.5).

Eine Schleife $T_1 \circlearrowleft a_k$ ist ebenfalls erlaubt. Die zugeordnete Turingtafel ergibt sich aus der von T_1, indem Zeilen $z\ a_k\ z'\ s$ durch Zeilen $z\ a_k\ z_0\ a_k$ mit dem Anfangszustand z_0 von T_1 ersetzt werden.

Beispiel 2.2.4 Die *große Rechtsmaschine*



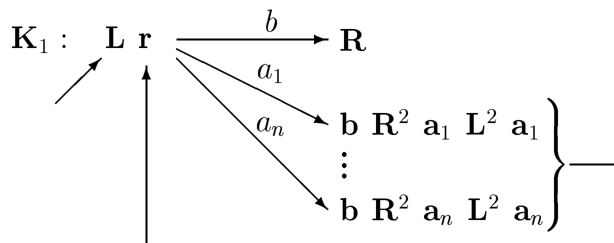
sucht, angesetzt auf ein beliebiges Feld, das erste rechts davon stehende Leerzeichen b. Analog arbeitet die *große Linksmaschine*

$$\mathbf{L} = \mathbf{l}\circlearrowleft \neq b.$$

Diese primitiven Maschinen spielen eine wichtige Rolle bei der Konstruktion komplexerer Maschinen. □

Wir geben im folgenden den Bandinhalt oder die Bandinschrift einer Turingmaschine mit Bandalphabet X häufig durch die Darstellung $\ldots bwb \ldots$ mit $w \in \bar{X}^*$ an. Das soll bedeuten, daß sich auf dem Band links und rechts von w nur noch Blankzeichen befinden. Die Nummer der jeweiligen Felder ist dadurch im Gegensatz zu Definition 2.1.3 nicht spezifiziert. Die Stellung des Kopfes kann zusätzlich durch Unterstreichen des Symbols, über dessen Feld der Kopf der Turingmaschine zu dem entsprechenden Zeitpunkt steht, markiert werden.

Beispiel 2.2.5 Die *Kopiermaschine*



überführt $\ldots bwb \ldots$ mit $w \in X^*$ in $\ldots bwbwb \ldots$, falls sie unmittelbar rechts von w oder auf dem letzten Feld von w startet. $\mathbf{R}^2$ ist dabei eine abkürzende Schreibweise für $\mathbf{RR}$. Entsprechendes gilt für $\mathbf{L}^2$. Die Wirkung der Kopiermaschine wollen wir an einem Beispiel verdeutlichen. Die Kopiermaschine starte mit dem Bandinhalt $\ldots ba_1a_2a_1b \ldots$. Das Startfeld werde durch Unterstreichung markiert. Die Kopiermaschine beginnt also mit $\ldots ba_1a_2a_1\underline{b} \ldots$ oder $\ldots ba_1a_2\underline{a_1}b \ldots$. Die weitere Arbeit wird durch die folgende Tabelle dargestellt. Dabei wird das Kopieren eines Symbols a_i durch die Folge der Turingmaschinen $\mathbf{rbR}^2\mathbf{a}_i\mathbf{L}^2\mathbf{a}_i$ bewirkt.

Wirkung von	liefert Bandinhalt
$\mathbf{L}$	$\ldots \underline{b} a_1 a_2 a_1 b \ldots$
$\mathbf{r}$	$\ldots b \underline{a_1} a_2 a_1 b \ldots$
$\mathbf{b}$	$\ldots b \underline{b} a_2 a_1 b \ldots$
$\mathbf{R}^2$	$\ldots b b a_2 a_1 b \underline{b} \ldots$
$\mathbf{a}_1$	$\ldots b b a_2 a_1 b \underline{a_1} b \ldots$
$\mathbf{L}^2$	$\ldots b \underline{b} a_2 a_1 b a_1 b \ldots$
$\mathbf{a}_1$	$\ldots b \underline{a_1} a_2 a_1 b a_1 b \ldots$
$\mathbf{r}$	$\ldots b a_1 \underline{a_2} a_1 b a_1 b \ldots$
$\mathbf{b}$	$\ldots b a_1 \underline{b} a_1 b a_1 b \ldots$
$\mathbf{R}^2$	$\ldots b a_1 b a_1 b a_1 \underline{b} \ldots$
$\mathbf{a}_2$	$\ldots b a_1 b a_1 b a_1 \underline{a_2} b \ldots$
$\mathbf{L}^2$	$\ldots b a_1 \underline{b} a_1 b a_1 a_2 b \ldots$
$\mathbf{a}_2$	$\ldots b a_1 \underline{a_2} a_1 b a_1 a_2 b \ldots$
$\mathbf{r}$	$\ldots b a_1 a_2 \underline{a_1} b a_1 a_2 b \ldots$
$\mathbf{b}$	$\ldots b a_1 a_2 \underline{b} b a_1 a_2 b \ldots$
$\mathbf{R}^2$	$\ldots b a_1 a_2 b b a_1 a_2 \underline{b} \ldots$
$\mathbf{a}_1$	$\ldots b a_1 a_2 b b a_1 a_2 \underline{a_1} b \ldots$
$\mathbf{L}^2$	$\ldots b a_1 a_2 \underline{b} b a_1 a_2 a_1 b \ldots$
$\mathbf{a}_1$	$\ldots b a_1 a_2 \underline{a_1} b a_1 a_2 a_1 b \ldots$
$\mathbf{r}$	$\ldots b a_1 a_2 a_1 \underline{b} a_1 a_2 a_1 b \ldots$
$\mathbf{R}$	$\ldots b a_1 a_2 a_1 b a_1 a_2 a_1 \underline{b} \ldots$

□

Beispiel 2.2.6 Die *Rechts-Suchmaschine*

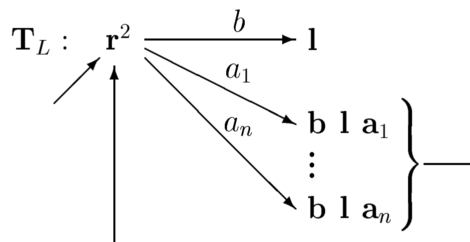
$$\mathbf{R}_s : \quad \mathbf{r} \circlearrowleft b$$

sucht nach rechts das erste beschriftete Feld. Analog sucht die *Links-Suchmaschine*

$$\mathbf{L}_s : \quad \mathbf{l} \circlearrowleft b$$

nach links das erste beschriftete Feld. Ähnlich könnten Suchmaschinen konstruiert werden, die nach links oder rechts ein Feld suchen, das ein ganz bestimmtes Element enthält. □

Beispiel 2.2.7 Die *linke Translationsmaschine*

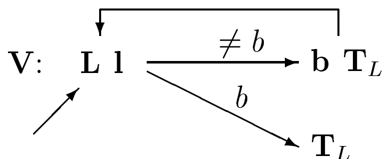


überführt $\ldots\underline{x}bwb\ldots$ für $x \in \bar{X}$ und $w \in X^+$ in $\ldots xw\underline{b}b\ldots$, d.h., sie löscht ein Zeichen b zwischen dem Arbeitsfeld und dem rechts davon stehenden Wort w. Sie schiebt also w von links an x heran. Die Wirkung der linken Translationsmaschine verdeutlichen wir am folgenden Beispiel:

$$\ldots\underline{x}ba_1a_2b\ldots \stackrel{\mathbf{r}^2}{\mapsto} \ldots xb\underline{a_1}a_2b\ldots \stackrel{\mathbf{bla}_1}{\longmapsto} \ldots x\underline{a_1}ba_2b\ldots \stackrel{\mathbf{r}^2}{\mapsto} \ldots xa_1b\underline{a_2}b\ldots$$

$$\stackrel{\mathbf{bla}_2}{\longmapsto} \ldots xa_1\underline{a_2}bb\ldots \stackrel{\mathbf{r}^2}{\mapsto} \ldots xa_1a_2b\underline{b}\ldots \stackrel{\mathbf{l}}{\mapsto} \ldots xa_1a_2\underline{b}b\ldots . \quad \square$$

Beispiel 2.2.8 Die *Verschiebemaschine*

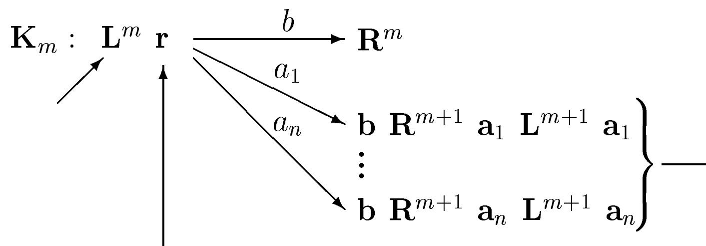


realisiert für $w_1, w_2 \in X^+$ die Überführung $\ldots bw_1bw_2\underline{b}\ldots \mapsto \ldots bw_2\underline{b}\ldots$, wobei w_2 nach links verschoben wird. Als Beispiel betrachten wir

$$\ldots ba_1a_2bw_2\underline{b}\ldots \stackrel{\mathbf{Ll}}{\mapsto} \ldots ba_1\underline{a_2}bw_2b\ldots \stackrel{\mathbf{bT}_L}{\longmapsto} \ldots ba_1bw_2\underline{b}\ldots \stackrel{\mathbf{Ll}}{\mapsto} \ldots b\underline{a_1}bw_2b\ldots$$

$$\stackrel{\mathbf{bT}_L}{\longmapsto} \ldots bbw_2\underline{b}\ldots \stackrel{\mathbf{Ll}}{\mapsto} \ldots \underline{b}bw_2b\ldots \stackrel{\mathbf{T}_L}{\mapsto} \ldots bw_2\underline{b}\ldots . \quad \square$$

Beispiel 2.2.9 Die *m-Kopiermaschine* $(m \geq 1)$



überführt für $w_1, \ldots, w_m \in X^+$ die Bandinschrift $\ldots bw_1bw_2b\ldots bw_m\underline{b}\ldots$ in $\ldots bw_1bw_2b\ldots bw_mbw_1\underline{b}\ldots$. Die Arbeitsweise ist ähnlich der aus Beispiel 2.2.5. Zum Kopieren von w_1 müssen hier jedoch die $m-1$ dazwischenliegenden Wörter $w_2, \ldots, w_m$ zusätzlich übersprungen werden, so daß statt der Maschinen $\mathbf{R}^2$ und $\mathbf{L}^2$ die Maschinen $\mathbf{R}^{m+1}$ und $\mathbf{L}^{m+1}$ verwendet werden. $\square$

Als abschließendes Beispiel konstruieren wir eine Turingmaschine, die die Arbeit eines Mealy-Automaten simuliert.

Beispiel 2.2.10 Es sei $M = (Z, X, Y, \delta, \lambda)$ ein Mealy-Automat. Für jeden Zustand $\bar{z} \in Z$ definieren wir eine Turingmaschine $T_{\bar{z}} = (Z \cup Z', X \cup Y, \delta', \bar{z})$ durch $Z' = \{(z, x) \mid z \in Z, x \in X\}$ und $\delta'(z, x) = ((z, x), \lambda(z, x))$ sowie $\delta'((z, x), y) = (\delta(z, x), r)$ für alle $z \in Z$, $x \in X$, $y \in Y$. Außerdem gelte $\delta'(z, b) = (z, s)$, und für alle weiteren Paare $(\tilde{z}, \tilde{x}) \in (Z \cup Z') \times (X \cup Y)$ werde $\delta'(\tilde{z}, \tilde{x}) = (\tilde{z}, s)$ gesetzt. Die Turingmaschine $\mathbf{Lr}T_{\bar{z}}$ überführt offenbar $\ldots bw\underline{b}\ldots$ für $w \in X^*$ in $\ldots bv\underline{b}\ldots$, wobei v die Folge der Ausgabesymbole ist, die der Mealy-Automat M, im Zustand $\bar{z}$ startend, bei Eingabe von w liefert. Ähnlich kann auch ein Moore-Automat simuliert werden. $\square$

2.3 Modifizierte Turingmaschinen

Es gibt eine Reihe anderer Definitionen von Turingmaschinen, die an ihren prinzipiellen Möglichkeiten nichts verändern, wohl aber zu anderen Komplexitäten, z.B. bezüglich des Zeitbedarfs, führen können. Wir geben hier eine Aufstellung einiger modifizierter Turingmaschinen an. Sie werden anschaulich charakterisiert und nur in (a), (b) und (d) vollständig angegeben.

(a) In diesem modifizierten Modell wird auf die Vollständigkeit aus Definition 2.1.1 verzichtet, außerdem soll in einem einzigen Schritt der Turingmaschine gleichzeitig Zustandsänderung, Bandbeschreibung und Bewegung des Kopfes stattfinden. Daher ersetzen wir die Abbildung δ aus Definition 2.1.1(c) durch eine *partielle Abbildung* $\delta : Z \times \bar{X} \to Z \times \bar{X} \times \{l, r, n\}$. Partielle Abbildung bedeutet, daß δ nicht mehr für alle Elemente aus $Z \times \bar{X}$ definiert sein muß, sondern nur auf einer Teilmenge von $Z \times \bar{X}$. Die zugehörige Turingtafel hat Zeilen des Typs

$$z\ x\ z'\ x'\ d.$$

Vom Zustand z geht die Turingmaschine beim Lesen von x in den Zustand z' über, beschreibt das Arbeitsfeld mit x', und ihr Kopf bewegt sich anschließend gemäß d. Dabei bedeutet $d = n$, daß sich der Kopf nicht bewegt. Diese Turingmaschine kann also das Beschreiben und eine Bewegung des Kopfes um ein Feld in einem Schritt durchführen, wozu die Maschine aus Definition 2.1.1 zwei Schritte benötigt. Ist $\delta(z, x)$ für ein Paar $(z, x) \in Z \times \bar{X}$ nicht definiert, so schreiben wir auch $\delta(z, x) = \emptyset$. Solche Paare (z, x) charakterisieren gerade eine Endkonfiguration. Es gibt dann in der zugehörigen Turingtafel keine Zeile, die mit $z\ x$ beginnt.
Wir wollen uns klarmachen, daß diese unvollständige Turingmaschine und die Turingmaschine aus Definition 2.1.1 sich gegenseitig simulieren, also ein vorgelegtes Band auf die gleiche Weise bearbeiten. Zu jeder Zeile der Turingtafel des einen Modells konstruieren wir die entsprechende Zeile oder Zeilen der Turingtafel des anderen Modells. Dabei müssen allerdings auch nicht vorhandene Zeilen der unvollständigen Turingmaschine, also der Fall $\delta(z, x) = \emptyset$, berücksichtigt werden. Wir gehen zunächst von einer Turingmaschine gemäß Definition 2.1.1 aus:

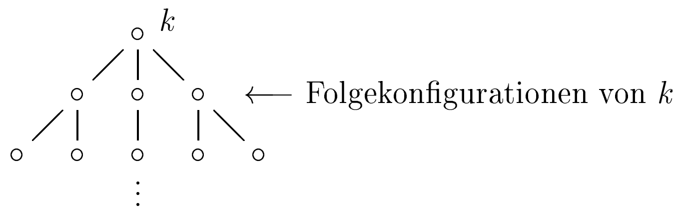
$$\begin{aligned} z\ x\ z'\ x' \ \text{ mit } \ x' \in \bar{X} \quad &\mapsto \quad z\ x\ z'\ x'\ n, \\ z\ x\ z'\ d \ \text{ mit } \ d \in \{r, l\} \quad &\mapsto \quad z\ x\ z'\ x\ d, \\ z\ x\ z'\ s \quad &\mapsto \quad \delta(z, x) = \emptyset. \end{aligned}$$

Das Band wird offenbar von beiden Turingmaschinen auf die gleiche Weise beschrieben. Die umgekehrte Simulation ergibt sich durch

$$\begin{aligned} z\ x\ z'\ x'\ n \quad &\mapsto \quad z\ x\ z'\ x', \\ z\ x\ z'\ x'\ d \ \text{ mit } \ d \in \{r, l\} \quad &\mapsto \quad z\ x\ z_1'\ x' \ \text{ und } \ z_1'\ x'\ z'\ d, \\ \delta(z, x) = \emptyset \quad &\mapsto \quad z\ x\ z\ s. \end{aligned}$$

Dabei wird für jede Zeile $z\ x\ z'\ x'\ d$ mit $d \in \{r, l\}$ ein neuer Zustand z_1' der zu konstruierenden Turingmaschine eingeführt.

(b) Eine *nichtdeterministische Turingmaschine* erhalten wir, wenn die Abbildung δ aus Definition 2.1.1(c) in eine Abbildung $\delta : Z \times \bar{X} \to \mathcal{P}(Z \times (\bar{X} \cup \{l, r, s\}))$ abgeändert wird. Dabei ist $\mathcal{P}(Z \times (\bar{X} \cup \{l, r, s\}))$ die Potenzmenge von $Z \times (\bar{X} \cup \{l, r, s\})$. Die nichtdeterministische Arbeitsweise dieser Turingmaschine ist durch einen Baum der Art



darstellbar. Zu einer Konfiguration dieser Maschine sind, im Gegensatz zum Fall einer vollständigen, deterministischen Turingmaschine, mehrere Folgekonfigurationen möglich. Eine Simulierung durch eine deterministische Turingmaschine erreicht man, indem man „Schicht für Schicht" alle Folgekonfigurationen verfolgt. Dabei werden auf dem Band der deterministischen Turingmaschine alle Konfigurationen der nichtdeterministischen Turingmaschine abgespeichert, die noch nicht weiterverfolgt wurden. Die Einzelheiten dieser Konstruktion sind sehr aufwendig und sollen hier nicht dargestellt werden. Eine deterministische Turingmaschine kann natürlich als spezielle nichtdeterministische Turingmaschine aufgefaßt werden.

(c) Eine *Turingmaschine mit einseitig begrenztem Band* hat ein Band der Form

0	1	2	3	4	5	6	⋯

Sie hält, wenn sie sich über den linken Rand hinaus bewegt. Sie kann durch eine Turingmaschine mit beidseitig unendlichem Band

⋯	#	0	1	2	3	4	5	⋯

simuliert werden, wobei die simulierende Turingmaschine hält, wenn sie links auf das Zeichen „#" trifft. Umgekehrt kann eine Turingmaschine gemäß Definition 2.1.1 durch eine Turingmaschine mit einseitig begrenztem Band simuliert werden. Die Zuordnung der Bänder erfolgt durch

⋯	−3	−2	−1	0	1	2	3	⋯

$$\Downarrow$$

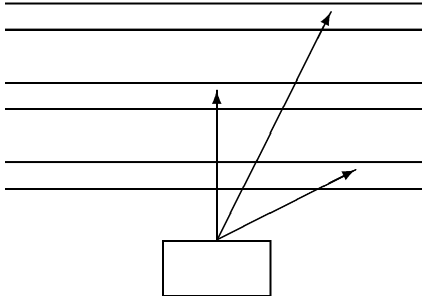
0	1	2	3	4	5	6	⋯
#	−1	−2	−3	−4	−5	−6	⋯

.

Diese Turingmaschine mit einseitig begrenztem Band besitzt ein Band, bei dem

ein Feld zwei Einträge hat. Jedes Feld kann dann durch ein Element eines kartesischen Produkts von zwei Mengen dargestellt werden. Die Turingmaschine hat also ein Band mit zwei Spuren. Durch eine entsprechende Zustandskomponente kann man sich merken, ob gerade die untere oder obere Spur bearbeitet wird.

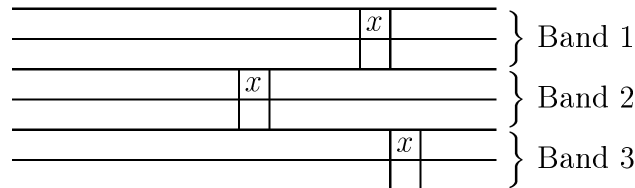
(d) Bei einer *Turingmaschine mit mehreren Bändern* bewegen sich alle Köpfe unabhängig voneinander, jedoch aufgrund der Gesamtinformation. Eine solche Turingmaschine mit drei Bändern kann durch das folgende Bild dargestellt werden.



Formal kann eine solche k-Band-Turingmaschine ($k \in \mathbb{N}$) durch ein Quadrupel $T = (Z, X, \delta, z_0)$ definiert werden, wobei im Unterschied zu Definition 2.1.1 die Überführungsfunktion durch

$$\delta : Z \times \bar{X}^k \to Z \times (\bar{X} \cup \{l, r, s\})^k$$

definiert wird. Eine Bandinschrift ist durch ein k-Tupel von Bandfunktionen $(\beta_1, \ldots, \beta_k)$ mit $\beta_i : \mathbb{Z} \to \bar{X}$, $i = 1, \ldots, k$, bestimmt. Eine Endkonfiguration $K = ((n_1, \ldots, n_k), (\beta_1, \ldots, \beta_k), z)$ mit $(n_1, \ldots, n_k) \in \mathbb{Z}^k$ und $z \in Z$ liegt vor, wenn $\delta(z, (\beta_1(n_1), \ldots, \beta(n_k))) = (z', (x_1, \ldots, x_k))$ gilt mit $z, z' \in Z$ und $x_\nu \in \bar{X} \cup \{l, r, s\}$, wobei für mindestens ein ν die Beziehung $x_\nu = s$ erfüllt ist. Die Turingmaschine stoppt also insgesamt, wenn auf mindestens einem Band eine Endkonfiguration vorliegt. Eine solche Turingmaschine wird durch eine Turingmaschine der Darstellung



simuliert, wobei die jeweils obere Spur der drei Bänder die Kopfstellung der zu simulierenden Turingmaschine durch ein Zeichen x markiert und die jeweils untere Spur zur Bearbeitung des ursprünglichen Bandes dient. Ein Feld ist also in 6 Komponenten aufgeteilt. Ein Zustand dieser Turingmaschine besteht aus einem Zustand der simulierten Turingmaschine , einem Kopfzähler und zusätzlichen Informationen. Die Maschine beginnt beim linken Kopf mit 1 als Wert des Kopfzählers. Sie läuft dann nach rechts, wobei sie das jeweilige Symbol unter x in den Zustand aufnimmt und den Kopfzähler um 1 erhöht. Beim rechten Kopf, der durch den Wert 3 des Kopfzählers identifizierbar ist, sind alle nötigen Informationen zur Simulation der oberen Turingmaschine gesammelt. Die Maschine läuft nun nach links, wobei die Operationen des jeweiligen Kopfes der oberen

Turingmaschine ausgeführt werden. Beim linken Kopf erfolgt schließlich zusätzlich eine Änderung des Zustandes gemäß der simulierten Turingmaschine sowie ein Zurücksetzen des Kopfzählers auf 1. Umgekehrt kann eine Turingmaschine gemäß Definition 2.1.1 als eine Turingmaschine mit n Bändern aufgefaßt werden. Auf einem Band erfolgt die Arbeit wie auf der gegebenen Maschine. Die anderen Bänder und Köpfe werden nicht verändert bzw. bewegt.

(e) Eine *Turingmaschine mit mehreren Köpfen auf einem Band* wird ähnlich wie die Turingmaschine aus (d) simuliert.

(f) Bei einer *Turingmaschine mit zweidimensionalem Band* erfolgt die Bewegung des Kopfes nach links, rechts, oben oder unten. Die folgenden Überlegungen können auch auf Maschinen mit n-dimensionalem Band übertragen werden. Wir betrachten das zweidimensionale Band

			b			
	b	b	x_1	b	x_2	
b	x_3	b	b	x_4	b	b
	b	x_5	x_6	b	x_7	
			b			

Es wird durch ein Band $\ldots b * bbx_1bx_2 * x_3bbx_4b * bx_5x_6bx_7 * b \ldots$ einer Turingmaschine mit zwei Bändern kodiert, wobei auf dem zweiten Band vermerkt werden kann, wieviele Symbole in einer Zeile stehen. Die Bewegungen der zweidimensionalen Turingmaschine können durch solche Turingmaschinen mit zwei Bändern simuliert werden. Wird dabei z.B. außerhalb des umrandeten Teils des Bandes ein neues Zeichen x geschrieben, so müssen in der Kodierung außer x auch Blankzeichen eingefügt werden.

2.4 Turing-Berechenbarkeit

Man sagt, daß eine Funktion Turing-berechenbar ist, wenn eine Turingmaschine existiert, die für jedes Argument, für das die Funktion definiert ist, nach endlich vielen Schritten einen Funktionswert liefert. Der Funktionswert befindet sich dann in geeignet kodierter Form auf dem Band. Außer ihm können auf dem Band noch weitere Informationen stehen, z.B. Zwischenwerte oder die Argumente der Funktion. Man kann annehmen, daß zu Beginn der Arbeit der Kopf der Turingmaschine über dem ersten Blankzeichen nach der Eingabe steht. Man könnte jedoch ebenso verabreden, daß er vorn über dem ersten Nichtblankzeichen der Eingabe steht. Dies hat keine prinzipielle Bedeutung für die berechneten Funktionen.

Definition 2.4.1 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine und $n \in \mathbb{N}_0$, $\Sigma \subset X$. T berechnet eine n-stellige partielle Funktion $f : (\Sigma^*)^n \to \Sigma^*$ wie folgt. Es sei $(w_1, \ldots, w_n) \in (\Sigma^*)^n$.

(a) T befindet sich zu Beginn im Zustand z_0. Der Bandinhalt ist $\ldots bw_1b \ldots bw_n\underline{b} \ldots$ für $n \geq 1$ bzw. $\ldots \underline{b} \ldots$ für $n = 0$.

(b) Hält T nach endlich vielen Schritten, so befindet sich unmittelbar links vom Kopf immer ein Wort $xw \in \bar{X}^*$ mit $w \in \Sigma^*, x \notin \Sigma$. Wir setzen $f(w_1, \ldots, w_n) = w$. Hält T nicht, so ist $f(w_1, \ldots, w_n)$ nicht definiert.

Die Funktion f heißt *Turing-berechenbar*. □

Die Turingmaschine T kann also, je nach Wahl von Σ und n, verschiedene partielle Funktionen berechnen. Für $n = 0$ wird eine 0-stellige Funktion, d.h. eine Konstante oder ggf. eine undefinierte Funktion, berechnet. Auch die Forderung $\Sigma \subset X$ könnte man fallenlassen, sofern nur $b \notin \Sigma$ gilt. Wenn nämlich ein Argument Symbole enthält, die nicht zu X gehören, dann gilt der zugehörige Wert der Funktion auf diesem Argument als undefiniert.

Wir vereinbaren, daß wir bei der Berechnung zahlentheoretischer Funktionen, also partieller Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$, wie oft üblich die Kodierung $\varphi : \mathbb{N}_0 \to \{|\}^+$ mit $\varphi(x) = |^{x+1}$ benutzen. Da man ohne Beschränkung der Allgemeinheit für das Bandalphabet $X = \{x_1, \ldots, x_n\}$ immer $x_1 = |$ annehmen kann, berechnet gemäß Definition 2.4.1 jede Turingmaschine bei Festlegung von $\Sigma = \{x_1\} = \{|\}$ eine solche Funktion. Allerdings könnte zusätzlich das leere Wort ε berechnet werden. Wir verabreden, daß in diesem Fall die zugehörige zahlentheoretische Funktion nicht definiert ist. In diesem Sinn sprechen wir von Turing-berechenbaren partiellen Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$.

Beispiel 2.4.1 Die Addition $x + y$ mit $x, y \in \mathbb{N}_0$ leistet die Maschine

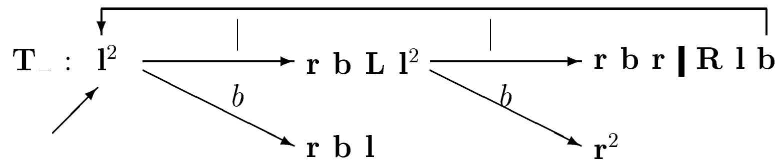
$$\mathbf{T}_+ : \quad \mathbf{L} \, \blacksquare \, \mathbf{R} \; \mathbf{l} \; \mathbf{b} \; \mathbf{l} \; \mathbf{b},$$

wobei ▌ die Schreibmaschine für $|$ ist. $\mathbf{T}_+$ überführt $\ldots b|^{x+1}b|^{y+1}\underline{b}\ldots$ in $\ldots b|^{x+y+1}\underline{b}\ldots$. Zum Beispiel ergibt sich aus $\ldots b|||b||\underline{b}\ldots$ mit **L** ▌ der Bandinhalt $\ldots b||||||b\ldots$. Durch **R** erhält man $\ldots b|||||\underline{b}\ldots$ und mit Hilfe von **l b l b** schließlich $\ldots b||||\underline{b}\ldots$. Dies entspricht der Rechnung $2 + 1 = 3$. □

Beispiel 2.4.2 Für $x, y \in \mathbb{N}_0$ definieren wir die Subtraktion (mit dem *Monus-Operator* $\dot{-}$)

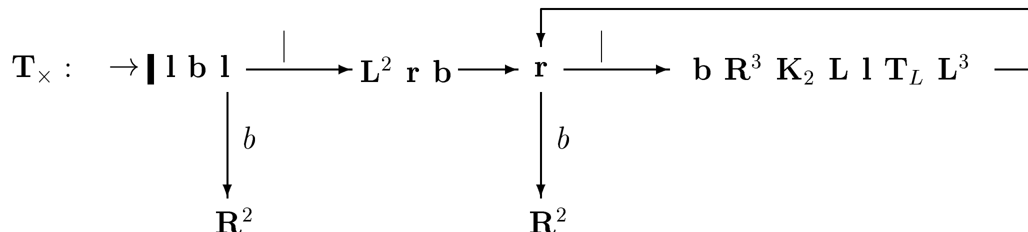
$$x \dot{-} y = \begin{cases} x - y, & \text{falls } x > y \\ 0 & \text{sonst.} \end{cases}$$

Sie wird durch



realisiert. $\mathbf{T}_-$ liefert, angesetzt auf $\ldots b|^{x+1}b|^{y+1}\underline{b}\ldots$, das Ergebnis, wobei in jeder Schleife in jeder der beiden $|$-Folgen ein $|$ entfernt wird. □

Beispiel 2.4.3 Die Multiplikation $x \cdot y$ mit $x, y \in \mathbb{N}_0$ wird durch die Turingmaschine



realisiert. $\mathbf{T}_\times$ wird auf $\ldots b|^{x+1}b|^{y+1}\underline{b}\ldots$ angesetzt und liefert als Ergebnis $\ldots bwb|^{xy+1}\underline{b}\ldots$ für ein $w \in \{b, |\}^*$. Die Turingmaschine führt folgende Arbeitsgänge

aus. Zunächst schreibt sie rechts von den Argumentwerten einen zusätzlichen Strich. Dann kürzt sie die Anzahl der Striche der beiden Argumentwerte jeweils um 1. Anschließend hängt sie x-mal jeweils y Striche an der rechten Seite an. Wir beschreiben nun die Arbeitsweise im einzelnen. Für $y = 0$ liefert die Turingmaschine $\mathbf{|lblR}^2$ die Bandinschrift $\ldots b|^{x+1}bb|\underline{b}\ldots$, d.h., $\mathbf{T}_\times$ hat 0 berechnet. Für $y \neq 0$ ist vor Anwendung der Rechtsmaschine $\mathbf{r}$, die sich in der Schleife befindet, also nach Beendigung der Arbeit von $\mathbf{|lblL}^2\mathbf{rb}$, die Bandinschrift durch $\ldots \underline{b}|^x b|^y b|b\ldots$ gegeben. Wir nehmen an, daß nach k Schleifendurchläufen, $x \geq k \geq 0$, die Bandinschrift $\ldots \underline{b}|^{x-k}b|^y b|^{ky+1}b\ldots$ lautet. Falls $x = k$ ist, führt die Anwendung von $\mathbf{rR}^2$ zu $\ldots b|^y b|^{xy+1}\underline{b}\ldots$, so daß die Multiplikation erfolgreich beendet wird. Anderenfalls liefert $\mathbf{rbR}^3\mathbf{K}_2$ die Bandinschrift $\ldots b|^{x-(k+1)}b|^y b|^{ky+1}b|^y\underline{b}\ldots$, mit $\mathbf{LlT}_L\mathbf{L}^3$ schließen wir einen Schleifendurchlauf ab und erhalten $\ldots \underline{b}|^{x-(k+1)}b|^y b|^{(k+1)y+1}b\ldots$. Mit jedem Durchlauf wird also der Wert von k um 1 erhöht, so daß wir insgesamt das gewünschte Ergebnis erhalten. □

Dies Beispiel zeigt zusammen mit Beispiel 2.2.10 und 1.2.4, daß Turingmaschinen leistungsfähiger als Moore- und damit auch als Mealy-Automaten sind.

In den angegebenen Beispielen bleiben nicht alle Argumentwerte erhalten. Das ist nach Definition 2.4.1 zulässig. Im folgenden werden wir eine standardisierte Form der Berechnung verwenden, bei der dies nicht erlaubt ist.

Definition 2.4.2 Es sei $n \in \mathbb{N}_0$ und $f : \mathbb{N}_0^n \to \mathbb{N}_0$ eine n-stellige partielle Funktion. f heißt *normiert Turing-berechenbar*, wenn eine Turingmaschine $T = (Z, X, \delta, z_0)$ mit $\{|\} \subset X$ existiert, für die das folgende gilt:

(a) T befindet sich zu Beginn im Zustand z_0, und bei Vorlage der Argumente $x_1, \ldots, x_n$ ist der Bandinhalt $\ldots b|^{x_1+1}b\ldots b|^{x_n+1}\underline{b}\ldots$ für $n \geq 1$ bzw. $\ldots \underline{b}\ldots$ für $n = 0$.

(b) Falls $f(x_1, \ldots, x_n)$ definiert ist, stoppt T nach endlich vielen Schritten mit dem Bandinhalt
$$\ldots b|^{x_1+1}b\ldots b|^{x_n+1}b|^{f(x_1,\ldots,x_n)+1}\underline{b}\ldots.$$
Die Kodierung der Argumentwerte steht in denselben Feldern wie zu Beginn der Berechnung. Die rechts vom Arbeitsfeld stehenden Felder sind leer. Bei der Berechnung geht T nicht über das mit b beschriftete Feld links vom Argument hinaus. Falls $f(x_1, \ldots, x_n)$ nicht definiert ist, hält T nicht. □

Die normierte Turing-Berechenbarkeit hat in Beweisen viele Vorteile, wie wir vor allem in Kapitel 3 sehen werden. Es ist einer Turingmaschine jedoch im allgemeinen nicht anzusehen, ob sie eine Funktion normiert Turing-berechnet. Wir wissen, daß nach Definition 2.4.1 jede beliebige Turingmaschine T bei Wahl von n und $\Sigma = \{|\}$ eine Funktion $f : \mathbb{N}_0^n \to \mathbb{N}_0$ Turing-berechnet. Der Beweis des nächsten Satzes zeigt, daß dann zu T eine Turingmaschine konstruiert werden kann, die f normiert Turing-berechnet.

Satz 2.4.1 Es sei $f : \mathbb{N}_0^n \to \mathbb{N}_0$, $n \in \mathbb{N}_0$, eine n-stellige partielle Abbildung. Genau dann ist f Turing-berechenbar, wenn f normiert Turing-berechenbar ist.

Beweis: Aus der normierten Turing-Berechenbarkeit von f folgt unmittelbar die Turing-Berechenbarkeit.

Es sei nun umgekehrt f Turing-berechenbar durch eine Turingmaschine $T = (Z, X, \delta, z_0)$. Es sei $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$ ein Argument von f. Zur Abkürzung setzen wir $w = |^{x_1+1}b \ldots b|^{x_n+1}$. Die Turingmaschine T beginnt im Zustand z_0 mit der Bandinschrift $\ldots bw\underline{b}\ldots$. Falls T hält, sei die Bandinschrift ihrer Endkonfiguration durch $\ldots bw_1w'\underline{y}w_2b\ldots$ mit $w' \in \{|\}^*$, $w_1, w_2 \in \bar{X}^*$ und $y \in \bar{X}$ gegeben. Dabei sei w' das berechnete Wort. Wir wollen annehmen, daß die Turingmaschine während der vorangegangenen Rechnung alle die Felder, die am Ende mit dem Wort w_1 bzw. w_2 beschrieben sind, betreten hat, nicht jedoch die Felder, die links bzw. rechts davon stehen. Falls $w_1 \neq \varepsilon$ ist, muß nach Definition 2.4.1 die Gleichung $w_1 = w_1'x$ mit $w_1' \in \bar{X}^*$, $x \neq |$, gelten.

Wir konstruieren als erstes eine Turingmaschine $T' = (Z', X', \delta', z_0)$, die T simuliert und schon einige der gewünschten Eigenschaften hat. Wir beschreiben zunächst ihre Arbeitsweise. T' besitzt zwei neue Bandsymbole λ und ρ, die als linke bzw. rechte Randmarkierung des Arbeitsbereiches des Bandes dienen. Die Turingmaschine T' startet im Zustand z_0 mit dem Band $\ldots b\lambda w\underline{b}\rho b\ldots$, das, abgesehen von den Randzeichen, dem initialen Band von T gleicht. Während der Arbeit bleibt das Feld mit der Beschriftung λ unverändert. Der Kopf von T' geht bei der Berechnung niemals nach links über dieses Feld hinaus. Am Ende stoppt die Turingmaschine mit dem Band $\ldots b\lambda w_1w'\underline{y}w_2\rho b\ldots$. Bis auf λ und ρ und eine Verschiebung des Bandes stimmen die Bandinschriften der Endkonfigurationen von T und T' einschließlich der Stellung des Kopfes überein.

Um diese Arbeitsweise zu erreichen, muß eine Linksbewegung von T, die bei der Simulierung durch T' zunächst auf das linke Randzeichen λ führt, durch eine Verschiebung des Arbeitsbereichs zwischen λ und ρ um ein Feld nach rechts simuliert werden. Anschließend kann dann unmittelbar rechts von λ über einem eingefügten Blankzeichen b die Simulation von T fortgesetzt werden. Außerdem muß bei Bedarf das rechte Randzeichen nach rechts verschoben und auch dort ein Zeichen b eingetragen werden. Wir zeigen im folgenden, daß wir eine solche Turingmaschine T' tatsächlich konstruieren können.

Zunächst werde $X' = X \cup \{\lambda, \rho\}$ und

$$Z' = Z \quad \cup \quad \{(z, \leftarrow), (z, x, S), (z, x, \rightarrow), (z, \rightarrow), (z, S), (z, e) \mid z \in Z, x \in \bar{X} \cup \{\lambda, \rho\}\}$$

gesetzt. Wir konstruieren die Turingtafel von T'. Sie besteht aus den Zeilen von T und wird durch weitere Zeilen ergänzt. Die angegebenen Zeilen liefern keine vollständige Turingmaschine, da für einige Zustände und Bandsymbole Instruktionen fehlen. Diese Zeilen können jedoch, da sie bei Berechnungen niemals auftreten, z.B. dadurch gegeben werden, daß bei ihnen die Turingmaschine ohne Änderung des Zustands stoppt. Insgesamt erhalten wir eine Turingmaschine wie in Definition 2.1.1.

Wir beginnen mit den Zeilen für die Rechtsverschiebung des Bandes. Für alle $z \in Z$ erhalten wir:

(1)	z	λ	(z, b, S)	r	
(2)	(z, x, S)	y	$(z, y, \rightarrow)$	x	für alle $x \in \bar{X}, y \in \bar{X} \cup \{\rho\}$
(3)	$(z, x, \rightarrow)$	y	(z, x, S)	r	für alle $x \in \bar{X} \cup \{\rho\}, y \in \bar{X}$
(4)	(z, ρ, S)	b	$(z, \leftarrow)$	ρ	
(5)	$(z, \leftarrow)$	x	$(z, \leftarrow)$	l	für alle $x \in \bar{X} \cup \{\rho\}$
(6)	$(z, \leftarrow)$	λ	z	r	.

Die zugehörige Arbeitsweise wird erläutert. Befindet sich nach einer Instruktion gemäß T die Turingmaschine im Zustand z mit dem Kopf über dem Feld mit der Beschriftung λ, so ergibt sich durch (1) ein Übergang in den Zustand (z, b, S). Damit wird sich der Zustand z sowie die Absicht gemerkt, rechts von λ ein Blankzeichen einzutragen (die dritte Komponente S bedeutet: Schreiben). Gleichzeitig wird eine Rechtsbewegung zu diesem Feld durchgeführt. Durch (2) erfolgt dieses Schreiben von b, wobei gleichzeitig der in diesem Feld vorgefundene Wert y im Zustand $(z, y, \rightarrow)$ notiert wird. Bevor dieser Wert y geschrieben werden kann, muß sich der Kopf von T' durch (3) nach rechts bewegen. Diese abwechselnde Anwendung der Instruktionen (2) und (3) wird fortgesetzt, bis man zum Abschluß dieser Rechtsbewegungen auf den rechten Rand stößt. Dabei wird ρ durch eine Instruktion gemäß (2) in den Zustand aufgenommen. Nach einer Rechtsbewegung mit Hilfe von (3) wird schließlich durch (4) das Randzeichen ρ rechts neben seinem ursprünglichen Feld geschrieben und in den Zustand $(z, \leftarrow)$ übergegangen. Das bedeutet die Beendigung des Verschiebens. Der Rücklauf nach links erfolgt durch (5), bis man am linken Rand auf λ trifft. Durch Anwendung von (6) geht man wieder in den Zustand z über. Der Kopf bewegt sich ein Feld nach rechts und steht über dem Blankzeichen, das im zweiten Schritt geschrieben wurde. Damit ist die Verschiebung abgeschlossen.

Befindet sich T' nach einer Instruktion gemäß T im Zustand z mit dem Kopf über ρ, so kann die Rechtserweiterung des Bandes für jedes $z \in Z$ durch die Folge von neuen Zeilen

$$\begin{array}{cccc} z & \rho & (z, \rightarrow) & b \\ (z, \rightarrow) & b & (z, S) & r \\ (z, S) & b & (z, e) & \rho \\ (z, e) & \rho & z & l \end{array}$$

erreicht werden. Dies liefert tatsächlich einen Übergang von $\ldots b\lambda w''\underline{\rho}b\ldots$ in $\ldots b\lambda w''\underline{b}\rho b\ldots$, wobei der Zustand z erhalten bleibt. Damit ist T' konstruiert. T' hält genau dann, wenn T hält und berechnet dasselbe Ergebnis.

Wir geben jetzt mit Hilfe der Turingmaschinen aus Abschnitt 2.2 die Turingmaschine an, die f normiert Turing-berechnet. Dies ist

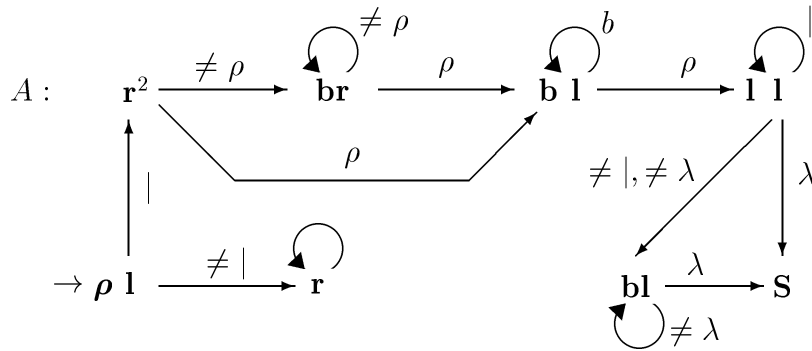
$$\mathbf{K}_n^n \mathbf{L}^n \boldsymbol{\lambda} \mathbf{R}^n \mathbf{r}\ \boldsymbol{\rho} \mathbf{l} T' A.$$

Dabei ist $\mathbf{K}_n$ die n-Kopiermaschine und A die Abschlußmaschine, die noch zu definieren ist. Falls T' hält, erhalten wir vor Berücksichtigung von A die Überführungen

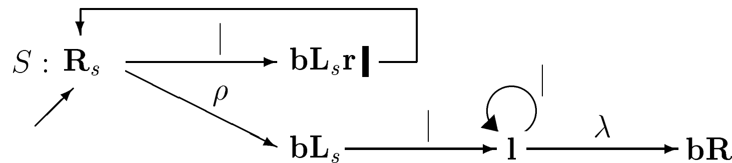
$$\ldots bw\underline{b}\ldots \xmapsto{\mathbf{K}_n^n} \ldots bwbw\underline{b}\ldots \xmapsto{\mathbf{L}^n\boldsymbol{\lambda}} \ldots bw\underline{\lambda}wb\ldots \xmapsto{\mathbf{R}^n\mathbf{r}\boldsymbol{\rho}\mathbf{l}} \ldots bw\lambda w\underline{b}\rho b\ldots$$

$$\xmapsto{T'} \ldots bw\lambda w_1 w' \underline{y} w_2 \rho b\ldots.$$

Die Abschlußmaschine A schiebt, falls nicht $w' = \varepsilon$ berechnet wurde, $w' = |^{f(x_1,\ldots,x_n)+1}$ an λ heran und ersetzt dabei überflüssige Zeichen durch das Blankzeichen. A ist gegeben durch



mit einer weiteren Turingmaschine S. Wir erklären die Arbeitsweise von A. Zunächst wird durch Anwendung von $\boldsymbol{\rho}\mathbf{l}$ das Symbol y durch ρ ersetzt und im Feld gleich links davon überprüft, ob $w' = \varepsilon$ gilt. In diesem Fall schließt sich eine unendliche Rechtsbewegung an, so daß $f(x_1, \ldots, x_n)$ undefiniert ist. Anderenfalls ist $w' = |^{f(x_1,\ldots,x_n)+1}$. Durch aufeinander folgende Anwendung von $\mathbf{r}^2$, einer 0- oder mehrfachen Iteration von $\mathbf{br}$, einer Anwendung von $\mathbf{b}$ sowie einer Iteration von $\mathbf{l}$ ergibt sich das Band $\ldots bw\lambda w_1|^{f(x_1,\ldots,x_n)+1}\underline{\rho}b\ldots$. Dann werden nach links alle Striche übersprungen, bis die Turingmaschine das erste Feld erreicht, das nicht $|$ enthält. Falls dies nicht bereits das linke Randzeichen ist, werden dieses und alle weiteren Felder bis zum linken Randzeichen mit dem Blankzeichen beschrieben. Vor Ausführung von S erhalten wir also $\ldots bw\underline{\lambda}v|^{f(x_1,\ldots,x_k)+1}\rho b\ldots$ mit einem $v \in \{b\}^*$. Die Turingmaschine S ist durch



gegeben. Dabei ist $\mathbf{R}_s$ die Rechts- und $\mathbf{L}_s$ die Links-Suchmaschine. Die Wirkung von S entspricht der linken Translationsmaschine, jedoch mit beliebiger Reichweite. Man beachte, daß auch für $v = \varepsilon$ die große Schleife durchlaufen wird, wobei jedoch keine echte Translation stattfindet. Am Ende wird in jedem Fall ρ gefunden und durch ein Blankzeichen ersetzt. Die folgenden Teil-Turingmaschinen liefern schließlich die Bandinschrift $\ldots bwb|^{f(x_1,\ldots,x_k)+1}\underline{b}\ldots$, wobei bwb in denselben Feldern wie am Anfang steht. Die Felder links davon wurden niemals benutzt, und rechts von $|^{f(x_1,\ldots,x_k)+1}$ befinden sich lauter Blankzeichen. Das bedeutet, daß f normiert Turing-berechenbar ist. $\square$

Wir geben nun noch ein spezielles Ergebnis an, das wir im Abschnitt 2.7 benötigen werden.

Satz 2.4.2 Es seien $f : \mathbb{N}_0 \to \mathbb{N}_0$, $g : \mathbb{N}_0 \to \mathbb{N}_0$ Turing-berechenbare partielle Funktionen. Dann ist auch ihre Komposition $f \circ g : \mathbb{N}_0 \to \mathbb{N}_0$ Turing-berechenbar.

Beweis: Wegen Satz 2.4.1 sind f und g normiert Turing-berechenbar. Es seien M_f und M_g die entsprechenden Turingmaschinen. Dann wird $f \circ g$ durch die Turingmaschine

$$M_g M_f \mathbf{V}$$

normiert Turing-berechnet, wobei **V** die Verschiebemaschine ist. In der Tat liefert sie für alle $x \in I\!N_0$, für die $g(x)$ und $f(g(x))$ definiert sind, die Überführung

$$\ldots b|^{x+1}\underline{b}\ldots \overset{M_g}{\mapsto} \ldots b|^{x+1}b|^{g(x)+1}\underline{b}\ldots \overset{M_f}{\mapsto} \ldots b|^{x+1}b|^{g(x)+1}b|^{f(g(x))+1}\underline{b}\ldots$$

$$\overset{\mathbf{V}}{\mapsto} \ldots b|^{x+1}b|^{f(g(x))+1}\underline{b}\ldots.$$

Nach Satz 2.4.1 ist dann $f \circ g$ auch Turing-berechenbar. $\square$

2.5 Gödelisierung

In diesem Abschnitt wollen wir vor allem eine Aufzählung aller Turingmaschinen angeben. Dazu wird eine Methode verwendet, die 1931 in einem ähnlichen Zusammenhang von *K. Gödel* eingeführt wurde.

Wir wählen eine andere Numerierung der Bandfelder der Turingmaschine als bisher, nämlich

	5	3	1	0	2	4	6	
...								...

Analog Definition 2.1.3 geben wir die *Bandfunktion* $\beta : I\!N_0 \to \bar{X}$ an. Die *Bandinschrift* wird durch $\Gamma(\beta) = \{(n, \beta(n)) \mid n \in I\!N_0\}$ gegeben, wobei $\beta(n)$ der *Inhalt des Feldes* n ist. Wir haben bereits in Abschnitt 2.1 verabredet, daß nur endlich viele Felder mit Nichtblankzeichen beschriftet sind. Entsprechend Definition 2.1.4 wird eine *Konfiguration* durch (n, β, z) mit $n \in I\!N_0$, Bandfunktion β und $z \in Z$ definiert. Anfangs- und Endkonfiguration sind dann wie in Definition 2.1.4 festgelegt. Die formale Definition einer Folgekonfiguration erfordert hier einige zusätzliche Fallunterscheidungen und wird nicht angegeben.

Definition 2.5.1 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine mit $X = \{a_1, \ldots, a_n\}$, und $\beta = I\!N_0 \to \bar{X}$ sei eine *Bandfunktion von* T. Dann heißt

$$G(\beta) = \prod_{i=0}^{\infty} p_{i+1}^{\mathrm{ind}(\beta(i))}$$

die *zur Bandfunktion gehörige Gödel-* oder *Bandnummer*, wobei p_j die j-te Primzahl ist und $\mathrm{ind} : \bar{X} \to \{0, \ldots, n\}$ die Abbildung mit $\mathrm{ind}(b) = 0$ und $\mathrm{ind}(a_j) = j$ für $1 \le j \le n$. $\square$

Es sei daran erinnert, daß $p_1 = 2$, $p_2 = 3$, $p_3 = 5$ usw. gilt. Obwohl in Definition 2.5.1 eine unendliche Produktbildung vorgenommen wird, erhalten wir immer einen endlichen Wert, da alle bis auf endlich viele Faktoren 1 sind. Für $\beta(i) = b$ gilt nämlich $p_{i+1}^{\mathrm{ind}(\beta(i))} = p_{i+1}^0 = 1$. Wenn alle Felder mit dem Blankzeichen beschrieben sind, ergibt sich $G(\beta) = 1$.

Beispiel 2.5.1 Es sei $X = \{a_1 = T, a_2 = S, a_3 = D, a_4 = A\}$ und $G(\beta) = 3.234.000$. Die Primfaktorzerlegung ergibt $2^4 \cdot 3^1 \cdot 5^3 \cdot 7^2 \cdot 11^1$. Daraus folgt, daß nur die Felder 0

bis 4 mit Zeichen $\neq b$ belegt sind.

Die Exponenten	4	1	3	2	1
liefern die in den Feldern	0	1	2	3	4
stehenden Zeichen	A	T	D	S	T ,

d.h., wir erhalten das Band

	5	3	1	0	2	4	6	
$\cdots$	b	S	T	A	D	T	b	$\cdots$

□

Nach der Gödelisierung der Bänder folgt jetzt die Gödelisierung beliebiger Turingmaschinen.

Definition 2.5.2 Es sei $T = (Z, X, \delta, z_0)$ eine Turingmaschine mit $Z = \{z_0, \ldots, z_m\}$ und $X = \{a_1, \ldots, a_n\}$. Weiter sei C die Matrix mit vier Spalten aus natürlichen Zahlen, die der Turingtafel von T bei der Zuordnung

$$\begin{array}{l} z_0 \mapsto 0, \ldots, z_m \mapsto m, \\ b \mapsto 0, a_1 \mapsto 1, \ldots, a_n \mapsto n, \\ l \mapsto n+1,\ r \mapsto n+2,\ s \mapsto n+3 \end{array}$$

der jeweiligen Elemente entspricht. Dann heißt

$$G(T) = p_1^n p_2^m \prod_{i=1}^{(m+1)(n+1)} \prod_{j=3}^{4} p_{\sigma_2(i,j)}^{c_{ij}}$$

die *Gödelnummer von T*. Dabei ist c_{ij} das Element der i-ten Zeile und j-ten Spalte der Matrix C, und $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ ist die Abbildung mit $\sigma_2(x, y) = 2^x(2y+1) - 1$. □

Man muß sich auf eine Numerierung der Zeilen von C festlegen. So kann man z.B. die Reihenfolge wählen, die durch die lexikographische Ordnung der beiden ersten Komponenten gegeben wird, also $(0,0), (0,1), \ldots, (0,n), (1,0), \ldots, (m,n)$. Dann wird zu jeder Turingmaschine genau eine Gödelnummer berechnet. Umgekehrt werden wir sehen, daß jede Zahl auch höchstens eine Turingmaschine darstellt. Dazu benötigen wir den folgenden Satz.

Satz 2.5.1 Die durch $\sigma_2(x, y) = 2^x(2y+1) - 1$ definierte Abbildung $\sigma_2 : \mathbb{N}_0^2 \to \mathbb{N}_0$ ist bijektiv.

Beweis: Wir zeigen zunächst, daß σ_2 injektiv ist. Es sei $(x_1, y_1) \neq (x_2, y_2)$. Wir nehmen an, daß

$$\sigma_2(x_1, y_1) = 2^{x_1}(2y_1+1) - 1 = 2^{x_2}(2y_2+1) - 1 = \sigma_2(x_2, y_2)$$

ist. Dann folgt

$$(*) \qquad 2^{x_1 - x_2}(2y_1+1) = 2y_2 + 1.$$

Für $x_1 \neq x_2$ sei ohne Beschränkung der Allgemeinheit $x_1 > x_2$. Dann ist die linke Seite der Gleichung $(*)$ gerade, die rechte ungerade, ein Widerspruch. Für $x_1 = x_2$ muß

wegen $(x_1, y_1) \neq (x_2, y_2)$ die Ungleichung $y_1 \neq y_2$ gelten. Dann kann die Gleichung $(*)$ nicht erfüllt sein. Folglich ist σ_2 injektiv.

Wir zeigen weiter, daß σ_2 surjektiv ist. Es sei $z \in \mathbb{N}_0$, und wir betrachten $z+1$. Wir teilen $z+1$ so oft wie möglich durch 2 (sagen wir x-mal), so daß das Ergebnis ganzzahlig bleibt. Dies liefert $\dfrac{z+1}{2^x} = 2y+1$, eine ungerade Zahl. Damit bestimmen wir $y = \dfrac{\frac{z+1}{2^x} - 1}{2}$. Es folgt $\sigma_2(x, y) = 2^x(2y+1) - 1 = z$. Somit ist σ_2 auch surjektiv. □

Die Umkehrabbildung σ_2^{-1} von σ_2 wird durch die Zuordnung $z \mapsto (\sigma_{21}(z) = x, \sigma_{22}(z) = y)$ gegeben, wobei die Werte x und y wie im Beweis von Satz 2.5.1 konstruiert werden. Diese Gödelisierung von Paaren läßt sich auf die Gödelisierung von n-Tupeln erweitern. Speziell für Tripel erhalten wir

$$\sigma_3 : \mathbb{N}_0^3 \to \mathbb{N}_0 \quad \text{mit} \quad \sigma_3(k, l, m) = \sigma_2(\sigma_2(k, l), m),$$

wobei die Umkehrabbildung σ_3^{-1} durch $\sigma_3^{-1}(z) = (\sigma_{31}(z), \sigma_{32}(z), \sigma_{33}(z))$ mit

$$\sigma_{31}(z) = \sigma_{21}(\sigma_{21}(z)), \ \sigma_{32}(z) = \sigma_{22}(\sigma_{21}(z)) \quad \text{und} \quad \sigma_{33}(z) = \sigma_{22}(z)$$

definiert ist.

Satz 2.5.2 Für ein beliebiges $x \in \mathbb{N}_0$ existiert ein Algorithmus, der entscheidet, ob x die Gödelnummer $G(T)$ einer Turingmaschine T ist. Wenn dies der Fall ist, kann T bestimmt werden.

Beweis: Wir betrachten die Darstellung der Gödelnummer einer Turingmaschine in Definition 2.5.2. Wegen $\sigma_2(i, j) \geq 6$ für $j \geq 3$ sind p_1 und p_2 verschieden von allen Primzahlen $p_{\sigma_2(i,j)}$. Da σ_2 nach Satz 2.5.1 bijektiv ist, sind insgesamt alle Primzahlen p_1, p_2 und $p_{\sigma_2(i,j)}$ paarweise verschieden. Wir bilden nun die Primfaktorzerlegung der vorgegebenen Zahl x. Diese Zerlegung ist bekanntlich eindeutig und erlaubt so die Bestimmung der Zahlen n, m, der $\sigma_2(i, j)$ und damit der Zahlen i und j sowie der $c_{i,j}$. Falls x die Gödelnummer einer Turingmaschine ist, müssen nach Definition 2.1.1 für die Exponenten von $p_1 = 2$ und $p_2 = 3$ die Relationen $n \geq 1$ bzw. $m \geq 0$ gelten. Weiter wird dann $i \leq (m+1)(n+1)$ verlangt, und j darf nur die Werte 3 oder 4 annehmen. Schließlich müssen die Werte von c_{i3} zwischen 0 und m und die von c_{i4} zwischen 0 und $n+3$ liegen. In diesem Fall sind alle Daten der Matrix C bestimmt, und die Turingmaschine mit der Gödelnummer x kann angegeben werden. Anderenfalls ist x nicht die Gödelnummer einer Turingmaschine. □

Die Gödelnummer einer Turingmaschine ist im allgemeinen sehr groß. Es ist kein Algorithmus bekannt, der für beliebige große Zahlen die Primfaktorzerlegung in vernünftiger Zeit liefert. Unter praktischen Gesichtspunkten ist also die Darstellung einer Turingmaschine als Gödelnummer nicht interessant, jedoch aus theoretischen Gründen. Sie liefert eine einheitliche Kodierung aller Turingmaschinen, die wir im restlichen Kapitel noch mehrmals benutzen werden. Bis auf Isomorphie existieren nur abzählbar viele Turingmaschinen. Wir können darüber hinaus sogar eine Aufzählung aller Turingmaschinen

$$T_0, T_1, T_2, \ldots, T_x, \ldots$$

entsprechend ihrer Gödelnummer x angeben. Wenn dabei x keine Gödelnummer ist, so wird für T_x speziell eine Turingmaschine gewählt, die niemals hält, z.B.

$$T = (\{z_0\}, \{x_1\}, \delta, z_0) \quad \text{mit} \quad \delta(z_0, x) = (z_0, r) \quad \text{für alle} \quad x \in \{b, x_1\}.$$

Diese Turingmaschine berechnet die leere Funktion $\bot\colon \mathbb{N}_0 \to \mathbb{N}_0$, die nirgends definiert ist. Das Symbol $\bot$ wird als „bottom" ausgesprochen.

2.6 Universelle Turingmaschinen

Ein Universalrechner soll ein beliebiges Programm ausführen können und dieselben Ergebnisse liefern wie ein Spezialrechner. Analog soll eine universelle Turingmaschine die Arbeit einer beliebigen Turingmaschine übernehmen können. Dabei erhält die universelle Turingmaschine U als Parameter eine Beschreibung der zu simulierenden Turingmaschine T und des Arguments x von T. Die Turingmaschine T kann durch ihre Gödelnummer als entsprechende Anzahl von Strichen dargestellt werden. U muß dann, angesetzt auf x, dasselbe Ergebnis liefern wie T. Eine universelle Turingmaschine ist also „programmierbar".

Zur weiteren Beschreibung der Simulation verwenden wir eine Kodierung des Ablaufs der Turingmaschinen-Berechnung von T, die eindeutig durch die Folge der Konfigurationen $K_0, K_1, \ldots, K_n, \ldots$ beschrieben wird, wobei eventuell mit einer Endkonfiguration K_e ein Abbruch stattfindet. Für jede Turingmaschine T definieren wir eine Abbildung ν von der Menge der Konfigurationen von T (siehe Definition 2.1.4, man beachte zusätzlich die geänderte Numerierung der Bandfelder) in die Menge $\mathbb{N}_0 \times \mathbb{N} \times \mathbb{N}_0$ durch

$$\nu(i, \beta, z_j) = (i, G(\beta), j).$$

Definition 2.6.1 Es sei U eine Turingmaschine. U heißt *universelle Turingmaschine*, wenn sie jede beliebige Turingmaschine T, die auf dem Feld i des Bandes mit der Bandfunktion β steht, wie folgt simuliert:

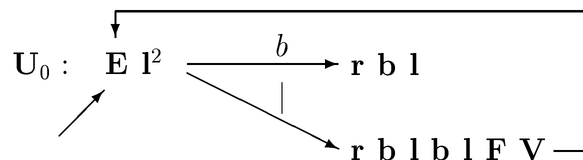
(a) Ist K_0 die Anfangskonfiguration von T, so startet U mit $\ldots b\,|^{G(T)+1}\, b\,|^{\sigma_3(\nu(K_0))+1}\, \underline{b} \ldots$.

(b) Ist für T die Konfiguration K_{m+1} die Folgekonfiguration von K_m, dann überführt U in $r \geq 1$ Schritten $\ldots b\,|^{G(T)+1}\, b\,|^{\sigma_3(\nu(K_m))+1}\, \underline{b} \ldots$ in $\ldots b\,|^{G(T)+1}\, b\,|^{\sigma_3(\nu(K_{m+1}))+1}\, \underline{b} \ldots$.

(c) Wenn T stoppt, dann stoppt auch U. □

Wenn U anhält, dann liefert sie als Ergebnis eine Kodierung der Endkonfiguration von T. Daraus kann dann das Ergebnis der Berechnung von T abgelesen werden. Im folgenden Satz ist das Grundprinzip der Konstruktion einer universellen Turingmaschine beschrieben.

Satz 2.6.1 Die Maschine



ist eine universelle Turingmaschine. Dabei berechnet **E** die Funktion e mit

$$e(G(T), \sigma_3(\nu(K_m))) = \begin{cases} ||, & \text{falls eine Folgekonfiguration } K_{m+1} \text{ zu } K_m \\ & \quad \text{existiert} \\ | & \text{sonst,} \end{cases}$$

und **F** überführt $\ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; \underline{b} \ldots$ in die Bandinschrift $\ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; b \; |^{\sigma_3(\nu(K_{m+1}))+1} \; \underline{b} \ldots$. □

Der Beweis, den wir hier nicht ausführen, verlangt die effektive Konstruktion der Turingmaschinen **E** und **F**. Offensichtlich muß man dabei auf die Struktur der zu simulierenden Turingmaschine T eingehen, die durch $G(T)$ kodiert ist. Der übrige Ablauf ist klar:

$$\ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; \underline{b} \ldots \overset{\mathbf{El}^2}{\longmapsto} \ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; b \underline{|} \; | \; b \ldots$$

(falls eine Nachfolgekonfiguration existiert)

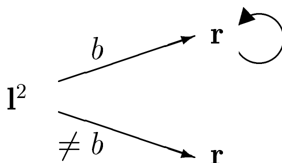
$$\overset{\mathbf{rblbl}}{\longmapsto} \ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; \underline{b} \ldots$$

$$\overset{\mathbf{F}}{\longmapsto} \ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_m))+1} \; b \; |^{\sigma_3(\nu(K_{m+1}))+1} \; \underline{b} \ldots$$

$$\overset{\mathbf{V}}{\longmapsto} \ldots b \; |^{G(T)+1} \; b \; |^{\sigma_3(\nu(K_{m+1}))+1} \; \underline{b} \ldots.$$

2.7 Unentscheidbare Probleme

Wir betrachten zunächst die Turingmaschine T, die durch



gegeben ist. Sie bewegt sich zunächst zwei Felder nach links und prüft, ob in diesem Feld ein Nichtblankzeichen steht. In diesem Fall macht sie eine Rechtsbewegung und stoppt. Anderenfalls bewegt sich der Kopf fortlaufend nach rechts. Es ist also in Abhängigkeit von der Eingabe und dem initialen Arbeitsfeld der Turingmaschine einfach zu entscheiden, ob sie hält oder nicht. Die Frage ist nun, ob es einen Algorithmus gibt, der für beliebige Turingmaschinen eine solche Entscheidung liefert.

Definition 2.7.1 Als *Halteproblem bei Turingmaschinen* bezeichnet man die Frage, ob ein Algorithmus existiert, mit dem man für eine beliebige Turingmaschine T und eine beliebige Anfangskonfiguration entscheiden kann, ob T nach endlich vielen Schritten hält oder nicht. □

Definition 2.7.2 Es sei X ein Alphabet, $M_1, M_2 \subset X^*$ und $M_1 \subset M_2$.

(a) M_1 heißt *entscheidbar relativ zu* M_2, wenn ein Algorithmus existiert, mit dem für jedes $x \in M_2$ effektiv festgestellt werden kann, ob $x \in M_1$ gilt oder nicht. Ein solcher Algorithmus heißt *Entscheidungsverfahren*.

(b) M_1 heißt *entscheidbar*, wenn M_1 entscheidbar ist relativ zu X^*. □

Wir müssen uns über den Algorithmusbegriff Klarheit verschaffen. Jeder hat eine Vorstellung von einem effektiven Algorithmus. Er soll auf irgendeine vernünftige Art und Weise Probleme lösen. Man kann ihn intuitiv verstehen als eine präzise endliche Beschreibung eines allgemeinen Verfahrens, wobei die Beschreibung auf ausführbare elementare Einzelschritte zurückgeführt wird. Es ist ersichtlich, daß eine solche Begriffserklärung viele Fragen offen läßt und sicherlich nicht erlaubt, allgemeine Aussagen über Algorithmen in mathematisch zuverlässiger Form zu treffen. Man benötigt vielmehr ein mathematisches Modell für Algorithmen, das universell ist in dem Sinn, daß jeder vorstellbare Algorithmus mit ihm simuliert werden kann. Man hat viele Algorithmenmodelle angegeben, die alle zueinander äquivalent sind, mit deren Hilfe man also jeweils dieselbe Klasse von Funktionen berechnen kann. Ein solches Modell ist durch die in diesem Kapitel besprochenen Turingmaschinen gegeben, aber ebenso durch die Registermaschinen aus Abschnitt 2.8, durch μ-rekursive Funktionen (siehe Kapitel 3), Markoff-Algorithmen, **while**-Programme, durch den λ-Kalkül usw. Die Churchsche These, die zunächst für den λ-Kalkül formuliert wurde, behauptet, daß diese Modelle universell sind.

Churchsche These: Eine partielle Funktion ist genau dann intuitiv berechenbar, wenn sie durch eine Turingmaschine berechnet wird. □

Die Richtigkeit der Churchschen These ist nicht beweisbar, weil ja gerade der intuitive Algorithmusbegriff nicht zu formalisieren ist. Da jedoch so viele äquivalente Algorithmenmodelle gefunden wurden, aber noch kein umfassenderes Modell, kann man dies als einen Hinweis für die Gültigkeit der Churchschen These verstehen.

Aufgrund dieser Überlegungen ist das Entscheidungsverfahren nach Definition 2.7.2 mit Hilfe einer Turingmaschine durchführbar. Dabei kann die Entscheidung dadurch erfolgen, daß die Turingmaschine über einem Feld mit einem entsprechenden Entscheidungssymbol, z.B. „j“ oder „n“, stehenbleibt. Ebenso ist es aber auch möglich, daß ein bestimmter Wert Turing-berechnet wird, z.B. 1 im Falle der positiven und 0 im Falle der negativen Entscheidung. Dieses unterschiedliche Entscheidungsverhalten hat jedoch keine besondere Bedeutung, da eine gegenseitige Simulation der verschiedenen Verhaltensweisen immer durchgeführt werden kann. Wir werden im folgenden Satz die Entscheidung durch das Berechnen der Werte 1 oder 0 herbeiführen. Wir sagen kurz, die Turingmaschine stoppt mit 1 oder 0.

Satz 2.7.1 Das Halteproblem ist nicht entscheidbar, d.h., es existiert kein Algorithmus, der zu entscheiden erlaubt, ob eine beliebige Turingmaschine, angesetzt auf eine beliebige Bandinschrift, stoppt oder nicht.

Beweis: Wir betrachten zunächst ein spezielles Halteproblem und nehmen ohne Beschränkung der Allgemeinheit an, daß $X = \{|\}$ gilt. Mit $\overline{\overline{n}}$ bezeichnen wir die Bandinschrift $\ldots b\,|^{n+1}\,\underline{b} \ldots$ für alle $n \in \mathbb{N}_0$. Es sei E eine Eigenschaft für Turingmaschinen. Für eine Turingmaschine T bedeute $E(T)$, daß die Eigenschaft E auf T zutrifft. Wir betrachten die folgende Eigenschaft E:

$$E(T) \text{ gilt genau dann, wenn } T, \text{ angesetzt auf } \overline{\overline{G(T)}}, \text{ mit 0 stoppt.}$$

$\overline{\overline{G(T)}}$ ist eine Kodierung von T. Folglich entspricht E dem Halteproblem bei Selbstanwendung. Zu zeigen ist, daß die Eigenschaft E unentscheidbar ist. Wir nehmen das Gegenteil an. Es existiert dann eine Turingmaschine T_0, die $E(T)$ für beliebige T entscheidet. Die Entscheidung ergibt sich wie folgt:

(∗) $E(T)$ gilt genau dann, wenn T_0, angesetzt auf $\overline{\overline{G(T)}}$, mit 1 stoppt,
$E(T)$ gilt genau dann nicht, wenn T_0, angesetzt auf $\overline{\overline{G(T)}}$, mit 0 stoppt.

Durch Einsetzen von T_0 in (∗) sowie in die Definition von E erhalten wir:

$E(T_0)$ gilt genau dann, wenn T_0, angesetzt auf $\overline{\overline{G(T_0)}}$, mit 1 stoppt,
$E(T_0)$ gilt genau dann, wenn T_0, angesetzt auf $\overline{\overline{G(T_0)}}$, mit 0 stoppt.

Dies ist ein Widerspruch, folglich muß E unentscheidbar sein.

Wir nehmen nun an, daß das allgemeine Halteproblem entscheidbar ist und somit ein Entscheidungsverfahren V für das allgemeine Halteproblem existiert. Mit Hilfe von V können wir dann auch ein solches Verfahren für das spezielle Problem E konstruieren, das durch die Anwendung von V auf T und $\overline{\overline{G(T)}}$ definiert wird. Dies Verfahren liefert die Antwort auf die Frage, ob die Turingmaschine T, falls sie auf $\overline{\overline{G(T)}}$ angesetzt wird, stoppt oder nicht. Im Fall des Stoppens kann man überprüfen, ob T mit dem Ergebnis 0 stoppt oder nicht, d.h., ob $E(T)$ erfüllt ist oder nicht. Im anderen Fall ist $E(T)$ nicht erfüllt. Folglich entscheidet dieses Verfahren die Eigenschaft E, was im Widerspruch zu dem oben gewonnenen Ergebnis steht. □

Im folgenden wollen wir wieder Turing-berechenbare Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$ betrachten. Ohnehin kann jede berechenbare Funktion so kodiert werden. In Übereinstimmung mit Definition 2.7.2 (man wähle $X = \{|\}$) heißt nun eine Teilmenge $A \subset \mathbb{N}_0$ genau dann *entscheidbar*, wenn die *charakteristische Funktion* $\chi_A : \mathbb{N}_0 \to \mathbb{N}_0$ mit

$$\chi_A(x) = \begin{cases} 1, & \text{falls } x \in A \\ 0, & \text{falls } x \notin A \end{cases}$$

Turing-berechenbar ist. Mit A ist offenbar auch das Komplement $\mathbb{N}_0 - A$ entscheidbar.

Wie auf Seite 42 sei eine Aufzählung

$$T_0, T_1, T_2, T_3, \ldots, T_n, \ldots$$

aller Turingmaschinen gegeben. Zur Vereinfachung betrachten wir die von ihnen berechneten einstelligen Funktionen $\varphi_n : \mathbb{N}_0 \to \mathbb{N}_0$. Dabei sei n die Gödelnummer der entsprechenden Turingmaschine. Ist jedoch n keine Gödelnummer einer Turingmaschine, dann wird für T_n speziell eine Turingmaschine gewählt, die niemals hält und die überall undefinierte partielle Funktion $\perp: \mathbb{N}_0 \to \mathbb{N}_0$ berechnet. Durch die Aufzählung dieser Turingmaschinen ist auch eine Aufzählung

$$\varphi_0, \varphi_1, \varphi_2, \varphi_3, \ldots, \varphi_n, \ldots$$

aller partiellen berechenbaren Funktionen $\mathbb{N}_0 \to \mathbb{N}_0$ gegeben. Dabei nennen wir n den Index der Turingmaschine T_n oder der Funktion φ_n.

Definition 2.7.3 Eine Menge von Indizes $J \subset I\!N_0$ *respektiert Funktionen*, wenn für alle $i, j \in I\!N_0$ mit $i \in J$ und $\varphi_i = \varphi_j$ die Beziehung $j \in J$ folgt. □

Beispiel 2.7.1 Offenbar respektieren die folgenden Indexmengen Funktionen:
(a) $A = \{i \mid 7$ liegt im Wertebereich von $\varphi_i\}$,
(b) $B = \{i \mid \varphi_i$ ist total$\}$,
(c) $C = \{i \mid \varphi_i$ ist überall undefiniert$\}$,
(d) $D = \{i \mid \varphi_i = \xi\}$, wobei $\xi : I\!N_0 \to I\!N_0$ eine gegebene partielle Funktion ist. □

Die Menge $E = \{i \mid \varphi_i(i)$ ist definiert$\}$ respektiert Funktionen hingegen nicht. Man kann zeigen, daß ein Index e existiert, für den der Definitionsbereich von φ_e nur aus dem Element e besteht. Dies ist nicht einfach und soll hier nicht durchgeführt werden. Es gibt also einen Algorithmus, der genau dann terminiert, wenn er seine eigene Beschreibung als Eingabe erhält. Es sei e' ein Index mit $e' \neq e$ und $\varphi_{e'} = \varphi_e$. Ein solcher Index e' existiert. Dies sehen wir wie folgt ein. Es sei $T_e = (Z, X, \delta, z_0)$. Wir wählen ein $z' \notin Z$ und definieren eine neue Turingmaschine $T = (Z', X, \delta', z_0)$ durch $Z' = Z \cup \{z'\}$ und

$$\delta'(z, x) = \delta(z, x) \text{ und } \delta'(z', x) = (z', |) \text{ für } z \in Z, x \in \bar{X}.$$

Es sei e' der zugehörige Index. Offenbar wird der Zustand z' nie erreicht, so daß T_e und $T_{e'}$ dieselbe Funktion berechnen. Es gilt $e \in E$, $\varphi_e = \varphi_{e'}$ und $e' \notin E$.

Satz 2.7.2 (Satz von *Rice*) Es sei $J \subset I\!N_0$ eine Menge von Indizes, die Funktionen respektiert. Dann ist J genau dann entscheidbar, wenn $J = \emptyset$ oder$J = I\!N_0$ gilt.

Beweis: Der Beweis erfolgt durch Reduktion auf die Unentscheidbarkeit des Halteproblems. Für $J = \emptyset$ oder $J = I\!N_0$ ist J offensichtlich entscheidbar. Die umgekehrte Richtung beweisen wir durch Kontraposition. Es sei eine Indexmenge J mit $\emptyset \neq J \neq I\!N_0$ gegeben. Wir müssen zeigen, daß J nicht entscheidbar ist. Es sei $\perp$ die überall undefinierte Funktion. Da J Funktionen respektiert, liegen alle Indizes von $\perp$ entweder alle in $I\!N_0 - J$ oder alle in J. Im ersten Fall existiert eine berechenbare Funktion $\theta \neq \perp$, deren Indizes in J liegen. Im zweiten Fall liegen diese Indizes in $I\!N_0 - J$. Wir nehmen ohne Beschränkung der Allgemeinheit an, daß die Indizes von θ in J liegen. Es sei j ein solcher Index.

Für jeden Index $i \in I\!N_0$ betrachten wir die φ_i berechnende Turingmaschine T_i. Nach Satz 2.4.1 können wir dann eine Turingmaschine $T_{\tilde{i}}$ konstruieren, die φ_i normiert Turing-berechnet. Wir bilden damit die Turingmaschine

$$T_{f(i)} = (\mathbf{r}\,\mathbf{|}\,)^{i+1}\,\mathbf{r}\,T_{\tilde{i}} \longrightarrow \underset{\circlearrowleft |}{\mathbf{bl}} \xrightarrow{\;b\;} \underset{\circlearrowleft |}{\mathbf{bl}} \xrightarrow{\;b\;} T_j$$

Ihr Index $f(i)$ hängt bei festem j nur von i ab und kann effektiv bestimmt werden. Nach der Churchschen These existiert dann eine Turingmaschine, die diese „Indextransformation“ $f : I\!N_0 \to I\!N_0$ berechnet. Die Konstruktion einer entsprechenden Turingmaschine ist sehr aufwendig und soll hier entfallen. Für ein Argument $x \in I\!N_0$

berechnet $T_{f(i)}$ zunächst $\varphi_i(i)$. Falls $\varphi_i(i)$ definiert ist, entfernt $T_{f(i)}$ zunächst das Ergebnis $\varphi_i(i)$, dann das Argument i und berechnet anschließend $\theta(x)$. Es gilt also

$$\varphi_{f(i)} = \begin{cases} \theta, & \text{falls } \varphi_i(i) \text{ definiert ist} \\ \perp, & \text{falls } \varphi_i(i) \text{ undefiniert ist.} \end{cases}$$

Für alle $i \in \mathbb{N}_0$ zeigen wir die Behauptung

$$f(i) \in J \iff \varphi_i(i) \text{ ist definiert.}$$

Wenn $\varphi_i(i)$ definiert ist, dann gilt $\varphi_{f(i)} = \theta$. Da der Index j von θ in J liegt, folgt $f(i) \in J$ wegen Definition 2.7.3. Umgekehrt gelte $f(i) \in J$. Entweder gilt $\varphi_{f(i)} = \theta$ oder $\varphi_{f(i)} = \perp$. Da der Index von $\perp$ nicht in J liegt, folgt wegen $f(i) \in J$ die Aussage $\varphi_{f(i)} = \theta$. Also ist $\varphi_i(i)$ definiert.

Es sei χ_J die charakteristische Funktion von J. Dann gilt wegen der obigen Behauptung für die Funktion

$$\chi_J(f(i)) = \begin{cases} 1, & \text{falls } \varphi_i(i) \text{ definiert ist} \\ 0 & \text{sonst.} \end{cases}$$

Die Funktion $\chi_J \circ f : \mathbb{N}_0 \to \mathbb{N}_0$ beschreibt genau das spezielle Halteproblem (Halteproblem bei Selbstanwendung, siehe Beweis von Satz 2.7.1) und ist daher nicht berechenbar. Da jedoch $f : \mathbb{N}_0 \to \mathbb{N}_0$ berechenbar ist, kann $\chi_J : \mathbb{N}_0 \to \mathbb{N}_0$ nach Satz 2.4.2 nicht berechenbar sein. Das bedeutet, daß J nicht entscheidbar ist. □

Beispiel 2.7.2 Da die Indexmengen A, B, C und D aus Beispiel 2.7.1 Funktionen respektieren, sind die folgenden Probleme nicht entscheidbar.

(a) Gibt ein beliebiger Algorithmus für eine geeignete Eingabe die Zahl 7 aus?
(b) Hält ein beliebiger Algorithmus bei jeder Eingabe?
(c) Hält ein beliebiger Algorithmus bei keiner Eingabe?
(d) Berechnet ein beliebiger Algorithmus die gegebene partielle Funktion $\xi : \mathbb{N}_0 \to \mathbb{N}_0$? □

Definition 2.7.3 und Satz 2.7.2 können mit einigen Abänderungen auf n-stellige Indexmengen $J \subset \mathbb{N}_0^n$, $n \geq 2$, übertragen werden. Man erhält z.B., daß die Menge $M = \{(i,j) \mid \varphi_i = \varphi_j\}$ Funktionen respektiert. Betrachtet man nämlich ein zweites Paar (i',j') mit $\varphi_i = \varphi_{i'}$ und $\varphi_j = \varphi_{j'}$, so folgt wegen $\varphi_i = \varphi_j$ auch $\varphi_{i'} = \varphi_{j'}$ und damit $(i',j') \in M$. M ist daher nicht entscheidbar. Also ist das *Äquivalenzproblem* für Algorithmen nicht entscheidbar.

Die Probleme aus Beispiel 2.7.2 für Algorithmen können natürlich auch für Programme, z.B. in Fortran, Modula-2 oder Algol 68, formuliert werden. Selbst wenn vorausgesetzt wird, daß unbegrenzt Speicherplatz zur Verfügung steht und keine Zeitgrenze gegeben ist, so sind die entsprechenden Probleme nicht effektiv lösbar. Ein anderes unentscheidbares Problem ist auch das Postsche Korrespondenzproblem, das neben dem Halteproblem häufig als ein Problem verwendet wird, auf das man andere Unentscheidbarkeitsprobleme zurückführt.

Definition 2.7.4 Ein *Postsches Korrespondenzproblem* $P = (X, n, \alpha, \beta)$ besteht aus einer endlichen Menge X, einer Zahl $n \in \mathbb{N}$ sowie zwei n-Tupeln

$$\alpha = (\alpha_1, \ldots, \alpha_n), \quad \beta = (\beta_1, \ldots, \beta_n)$$

mit $\alpha_i, \beta_i \in X^+$, $i = 1, \ldots, n$. Eine *Lösung* von P ist eine nichtleere endliche Folge von Indizes $i_1, \ldots, i_k$, $i_\kappa \in \{1, \ldots, n\}$, $\kappa = 1, \ldots, k$, mit

$$\alpha_{i_1} \ldots \alpha_{i_k} = \beta_{i_1} \ldots \beta_{i_k}. \quad \square$$

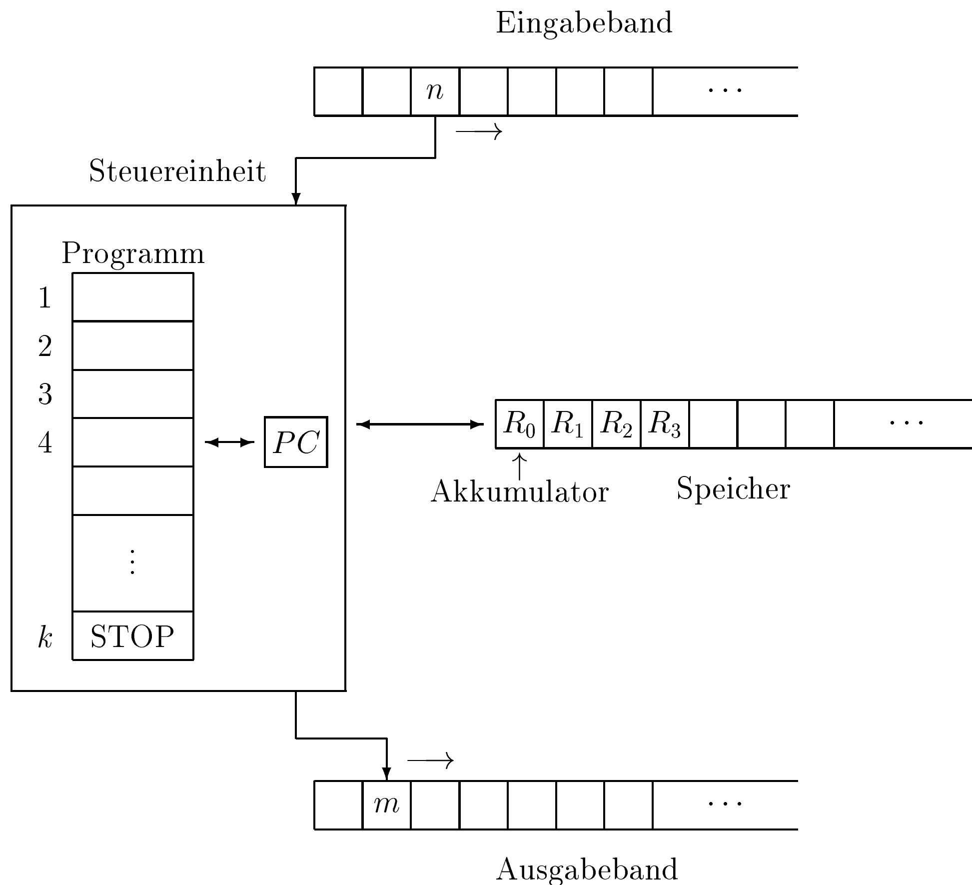
In [19] findet sich der Beweis des folgenden Satzes.

Satz 2.7.3 Es ist unentscheidbar, ob ein beliebiges Postsches Korrespondenzproblem eine Lösung besitzt. $\square$

2.8 Registermaschinen

Wir haben in den vorhergehenden Abschnitten gesehen, daß das Turingmaschinenmodell ausreichend ist, um Fragen der Berechenbarkeit zu untersuchen. Eine Turingmaschine hat jedoch mit einem modernen Computer wenig gemeinsam. Daher mag es sinnvoll sein, der Realität eher entsprechende Modelle zu betrachten. Das wichtigste Beispiel hierfür ist die Registermaschine mit wahlfreiem Speicherzugriff, die auch RAM (englisch: *random access machine*) genannt wird. Dieses Rechnermodell besitzt einen Speicher, der ähnlich dem eines modernen Mikroprozessors aufgebaut ist. Gleichzeitig stehen elementare arithmetische Operationen zur Verfügung. Der Speicher besteht aus einer unendlichen Folge von Registern, die Zahlen beliebiger Größe aufnehmen können, was natürlich eine Abstraktion der Wirklichkeit bedeutet. Ein Programm einer Registermaschine ist mit einem Programm vergleichbar, das in einer Assembler- oder höheren Programmiersprache geschrieben ist. Wir werden sehen, daß Turingmaschinen und Registermaschinen äquivalent sind.

Das folgende Bild stellt den schematischen Aufbau einer Registermaschine dar.



Wir geben nun die nötigen formalen Definitionen an.

Definition 2.8.1 Eine *Registermaschine mit wahlfreiem Speicherzugriff* (kurz RAM) besteht aus einer *Steuereinheit*, einem *Speicher* R, einem *Eingabe-* und einem *Ausgabeband*.

(a) Die Steuereinheit enthält ein gemäß Definition 2.8.2 definiertes *Programm*, also eine endliche Folge von durchnumerierten Befehlen, sowie den Befehlszähler PC. Der Befehlszähler PC beinhaltet eine Zahl $b \in \mathbb{N}$, die die Nummer des jeweils auszuführenden Befehls angibt.

(b) Der Speicher besteht aus einer durchnumerierten Menge von Registern R_i, $i = 0, 1, 2, \ldots$, die jeweils eine Zahl $r(i) \in \mathbb{N}_0$ enthalten. Dabei muß $\{r(i) \mid r(i) \neq 0\}$ zu jedem Zeitpunkt endlich sein. Das Register R_0 wird auch *Akkumulator* genannt.

(c) Das Eingabeband besteht aus unendlich vielen Feldern, die jeweils eine Zahl aus $\mathbb{N}_0$ enthalten. Ein Feld, das 0 enthält, wird als leer angenommen. Nur endlich viele Felder sind nicht leer. Auf dem Eingabeband steht ein Lesekopf, der nach dem Lesen einen Schritt nach rechts wandert. Das gelesene Symbol wird mit „ read“ bezeichnet.

(d) Das Ausgabeband besteht aus unendlich vielen Feldern, die jeweils eine Zahl aus $\mathbb{N}_0$ enthalten. Ein Feld, das 0 enthält, wird auch als leer angenommen. Nur endlich viele Felder sind nicht leer. Über dem Ausgabeband steht ein Schreibkopf, der nach dem Schreiben einen Schritt nach rechts wandert. □

Der Speicher aus (b) kann auch zu jedem Zeitpunkt als eine Funktion $r : \mathbb{N}_0 \to \mathbb{N}_0$ mit jeweils endlicher Menge $\{r(i) \mid r(i) \neq 0\}$ aufgefaßt werden. Die Register und auch die Felder des Eingabe- und des Ausgabefeldes sind potentiell unendlich groß, da sie beliebig große Zahlen enthalten können. Im folgenden fassen wir die Register R_i auch als Variablen für Zahlen aus $\mathbb{N}_0$ auf, für den Befehlszähler wird im üblichen Sinn b sowohl als Variable als auch als Wert aufgefaßt.

Definition 2.8.2 Ein *Programm* einer Registermaschine besteht aus einer ab 1 durchnumerierten Folge von Befehlen, wobei der letzte Befehl ein STOP-Befehl sein muß. Im folgenden sei zunächst $i \in \mathbb{N}_0$. Als Befehle stehen dann zur Verfügung:

Befehl	*Wirkung*
READ	$R_0 :=$ read, $b := b + 1$, der Lesekopf bewegt sich einen Schritt nach rechts.
WRITE	b:= b+1, der Schreibkopf schreibt $r(0)$ in das Ausgabefeld und bewegt sich einen Schritt nach rechts.
LOAD i	$R_0 := r(i)$ und $b := b + 1$.
STORE i	$R_i := r(0)$ und $b := b + 1$.
ADD i	$R_0 := r(0) + r(i)$ und $b := b + 1$.
SUB i	$R_0 := r(i) \dot{-} r(0)$ und $b := b + 1$.
GOTO m	$b := m$.
IF $R_i = 0$ GOTO m	Falls $r(i) = 0$ gilt, folgt $b := m$, anderenfalls $b := b + 1$.
IF $R_i > 0$ GOTO m	Falls $r(i) > 0$ gilt, folgt $b := m$, anderenfalls $b := b + 1$.
STOP	Die Registermaschine beendet die Ausführung des Programms.

In den vier Befehlen der mittleren Gruppe kann i auch durch $!i$ für eine *Konstante* $i \in \mathbb{N}_0$ (mit Ausnahme von STORE $!i$) oder $*i$ für *indirekte Adressierung* ersetzt werden. So hat z.B. LOAD $!i$ die Wirkung $R_0 := i$ und $b := b + 1$, wohingegen LOAD $*i$ die Wirkung $R_0 := r(r(i))$ und $b := b + 1$ besitzt. Unter SUB $!i$ wollen wir $R_0 := r(0) \dot{-} i$ verstehen.

Bei m in den Sprungbefehlen muß es sich um eine existierende Zeilennummer des Programms handeln. Ist bei jedem Sprungbefehl nur *eine* solche Nummer angegeben, so sprechen wir von einer *deterministischen* Registermaschine, sonst von einer *nichtdeterministischen.*

Die Registermaschine arbeitet taktweise. In jedem Takt wird der Befehl mit der Zahl b aus dem Befehlszähler PC ausgeführt. Zu Beginn ist dieser Wert $b = 1$. Am Anfang sind alle Register leer ($r(i) = 0$ für alle $i \in \mathbb{N}_0$), ebenso das Ausgabeband. Die Eingabe steht auf dem Eingabeband. Die Berechnung endet, wenn die Registermaschine auf den Befehl STOP trifft. Die Ausgabe ergibt sich dann aus der Beschreibung der Ausgabefelder. □

Neben den in Definition 2.8.2 angegebenen Befehlen können auch weitere arithmetische Operationen durch entsprechende Befehle aufgenommen werden. Sie lassen sich andererseits aber leicht als Unterprogramme definieren. Dies soll an einigen wenigen Beispielen deutlich gemacht werden.

Beispiel 2.8.1 Bei den folgenden Unterprogrammen wird der Akkumulator R_0 benutzt. Wenn sein Wert nach dem Unterprogramm noch zur Verfügung stehen soll, so muß er zunächst in ein anderes, sonst nicht benutztes Register, etwa R_j, durch STORE j weggespeichert werden, um nach Ausführung des Unterprogramms durch LOAD j zurückgeholt zu werden.

(a) Die Funktion SUCC i mit der Wirkung $R_i := r(i) + 1$ wird für $i > 0$ durch das folgende Unterprogramm definiert (für $i = 0$ durch ADD !1).

Befehl	*Wirkung*
STORE j	$R_j := r(0)$
LOAD !1	$R_0 := 1$
ADD i	$R_0 := r(i) + r(0) = r(i) + 1$
STORE i	$R_i := r(i) + 1$
LOAD j	$R_0 := r(j)$

(b) Die Funktion PRED i mit der Wirkung $R_i := r(i) \dot{-} 1$ für $i > 0$:

Befehl	*Wirkung*
STORE j	$R_j := r(0)$
LOAD !1	$R_0 := 1$
SUB i	$R_0 := r(i) \dot{-} r(0) = r(i) \dot{-} 1$
STORE i	$R_i := r(i) \dot{-} 1$
LOAD j	$R_0 := r(j)$

Die Funktion PRED 0:

Befehl	*Wirkung*
STORE j	$R_j := r(0)$
LOAD !1	$R_0 := 1$
SUB j	$R_0 := r(j) \dot{-} 1$

(c) IF $R_i > k$ GOTO m für Konstanten $k \in \mathbb{N}_0$ und $m \in \mathbb{N}$ mit der üblichen Wirkung für $i > 0$:

Befehl	*Wirkung*
STORE j	$R_j := r(0)$
LOAD !k	$R_0 := k$
SUB i	$R_0 := r(i) \dot{-} r(0) = r(i) \dot{-} k$
STORE s	$R_s := r(i) \dot{-} k$
LOAD j	$R_0 := r(j)$
IF $R_s > 0$ GOTO m	Sprung wie gewünscht

Man muss dafür Sorge tragen, daß auch das Register R_s sonst nicht benutzt wird. Für $i = 0$ kann man fast dasselbe Unterprogramm verwenden, wenn man den Befehl SUB i durch SUB j ersetzt. □

Da wir durch Registermaschinen, wie schon bei der normierten Turing-Berechenbarkeit, partielle Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$ berechnen wollen, müssen wir im Ein- und Ausgabeband die Argumentwerte bzw. den Funktionswert der Funktion um 1 erhöhen, da ein leeres Register durch 0 dargestellt wird. Wir erhalten

Definition 2.8.3 Es sei $n \in \mathbb{N}_0$ und $f : \mathbb{N}_0^n \to \mathbb{N}_0$ eine n-stellige partielle Funktion. Die Funktion f heißt RAM-berechenbar, wenn eine Registermaschine existiert, für die das folgende gilt:

(a) Bei Vorlage der Argumente $x_1, \ldots, x_n$ enthalten zu Beginn die ersten n Felder des Eingabebandes die Zahlen $x_1 + 1, \ldots, x_n + 1$, die anderen Felder enthalten 0.

(b) Falls $f(x_1, \ldots, x_n)$ definiert ist, dann trifft die Registermaschine nach endlich vielen Schritten auf den Befehl STOP, und für den Wert y des ersten Feldes des Ausgabebandes gilt dann $y > 0$ und $f(x_1, \ldots, x_n) = y - 1$. Falls $f(x_1, \ldots, x_n)$ nicht definiert ist, gilt $y = 0$, oder die Registermaschine hält nicht. □

Eine Registermaschine kann also, je nach Wahl von n, beliebigstellige Funktionen berechnen. Offensichtlich können wir bei Berücksichtigung weiterer Ausgabefelder auch partielle Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0^m$ berechnen. Wir erkennen, daß wir auf das Ein- und das Ausgabeband einer RAM verzichten können, wenn wir die Argumente und auch den Funktionswert im Speicher ab dem Register R_1 in geeigneter Weise ablegen. Der Vorteil der Benutzung des Eingabe- und des Ausgabebandes ist es jedoch, daß dann im Programm die Berücksichtigung der Stelligkeit im allgemeinen weniger Schwierigkeiten macht.

Wir zeigen die RAM-Berechenbarkeit der Addition und der Multiplikation.

Beispiel 2.8.2 (a) Die Funktion $f(n, m) = n + m$ wird von einer Registermaschine mit dem folgenden Programm berechnet:

Befehl	*Wirkung*
READ	$R_0 := n + 1$
PRED 0	
STORE 1	$R_1 := r(0) = n$
READ	$R_0 := m + 1$
PRED 0	
ADD 1	$R_0 := r(1) + r(0) = n + m$
SUCC 0	
WRITE	Ausgabe von $n + m + 1$

(b) Die Funktion $f(n, m) = n \cdot m$ kann durch n-malige Addition von m berechnet werden. Man beachte im folgenden Programm, daß zu Beginn im Register R_3 der Wert 0 steht. Die Duchnumerierung der Befehle entspricht nicht derjenigen, die sich bei Zurückgehen auf die Befehle von Definition 2.8.2 ergeben würde.

	Befehl	*Wirkung*
1.	READ	$R_0 := n + 1$
2.	PRED 0	
3.	STORE 1	$R_1 := r(0) = n$
4.	READ	$R_0 := m + 1$
5.	PRED 0	
6.	STORE 2	$R_2 := r(0) = m$

	Befehl	*Wirkung*
7.	IF $R_1 = 0$ GOTO 13	n-malige Addition von m beendet
8.	PRED 1	$R_1 := r(1) - 1$
9.	LOAD 2	$R_0 := r(2) = m$
10.	ADD 3	$R_0 = r(3) + r(0) = r(3) + m$
11.	STORE 3	$R_3 := r(0) = r(3) + m$
12.	GOTO 7	Ende des Schleifenkörpers
13.	LOAD 3	$R_0 := r(3)$ (Resultat)
14.	SUCC 0	
15.	WRITE	Ausgabe von $n \cdot m + 1$ □

Obwohl Turingmaschinen weniger der Realität entsprechen als Registermaschinen, können sie trotzdem dieselben Funktionen berechnen. Darüberhinaus haben Turingmaschinen den Vorteil, daß sie einfacher aufgebaut sind und damit für theoretische Überlegungen besser geeignet sind.

Satz 2.8.1 Es sei $f : \mathbb{N}_0^n \to \mathbb{N}_0$, $n \in \mathbb{N}_0$, eine n-stellige partielle Funktion. Genau dann ist f Turing-berechenbar, wenn f RAM-berechenbar ist.

Beweis: Es sei $f : \mathbb{N}_0^n \to \mathbb{N}_0$, $n \in \mathbb{N}_0$, eine Turing-berechenbare Funktion. Nach Satz 2.4.1 existiert dann eine Turingmaschine $T = (Z, X, \delta, z_0)$ mit $\{|\} \subset X$, die f normiert Turing-berechnet. Das bedeutet nach Definition 2.4.2, daß bei Vorlage der Argumente $y_1, \ldots, y_n$ der anfängliche Bandinhalt

$$\ldots b|^{y_1+1}b \ldots b|^{y_n+1}\underline{b} \ldots$$

ist und T während seiner Arbeit niemals über das mit b beschriftete Feld links vom Argument hinausgeht. Es sei $Z = \{z_0, z_1, \ldots, z_r\}$, $\bar{X} = \{b = x_0, x_1 = |, x_2, \ldots, x_s\}$ für gewisse $r, s \in \mathbb{N}$. Zustände und Eingaben können durch ihren Index in den Registern des Speichers dargestellt werden. Das Eingabeband der Registermaschine ist mit

$y_1 + 1$	$y_2 + 1$	$\cdots$	$y_n + 1$	0	0	$\cdots$

beschrieben. Für die Simulierung von T werden in den Registern $R_4, R_5 \ldots$ die Felder des Bandes von T dargestellt, wobei R_4 für das Feld steht, in dem sich anfangs das Zeichen b links vom Argument von T befindet. R_2 gibt die Kopfposition von T in bezug auf diese Register an. Dies gilt in allen folgenden Teilprogrammen.

Zunächst wird im Speicher das Band der Turingmaschine initialisiert. Dabei werden in das Register R_1 der Reihe nach die Argumente des Eingabebandes übernommen. Entsprechend viele Einsen als Darstellung des Striches $|$ werden dann jeweils in die passenden Register eingetragen, für das erste Argument z. B. in die Register R_4 bis R_{4+y_1}. Die Initialisierung erfolgt durch das folgende Programmstück.

	Befehl	*Wirkung*
1.	LOAD !5	$R_0 := 5$, hier beginnt der Eintrag der Einsen, wobei eine 1 einem | des Bandes entspricht, in R_4 steht 0 für das Blankzeichen b
2.	STORE 2	$R_2 := r(0) = 5$, Kopfposition für das erste Zeichen $\neq b$
3.	READ	$R_0 := r(0) = y_k + 1$ (das k-te Argument, Beginn der k-ten Iteration)
4.	STORE 1	$R_1 := r(0) = y_k + 1$
5.	IF $R_1 = 0$ GOTO 14	alle Argumente sind gelesen worden, der Kopfzähler zeigt zum Register mit dem Blankzeichen rechts vom Argument
6.	IF $R_1 = 0$ GOTO 12	das Argument der k-ten Iteration ist in Einsen umgewandelt
7.	LOAD !1	$R_0 := 1$
8.	STORE $*2$	$R_{r(2)} := 1$, trage die 1 in das dem Feld zugehörige Register ein
9.	SUCC 2	verschiebe den Kopf um eine Position nach rechts
10.	PRED 1	$R_1 := r(1) \dot{-} 1$, es sind noch $r(1) \dot{-} 1$ viele Einsen einzutragen
11.	GOTO 6	die nächste 1 soll eingetragen werden
12.	SUCC 2	verschiebe den Kopf um eine Position nach rechts, d.h., das Register nach den Registern mit den Einsen behält die ursprüngliche 0 für b
13.	GOTO 3	zum nächsten Argument

Der Kopf, der durch das Register R_2 charakterisiert wird, steht jetzt auf dem Blankzeichen hinter der Eingabe.

Als nächstes wird die Arbeit der Turingmaschine simuliert. Das Register R_1 dient nun zur Darstellung der Zustände und R_3 für die Bandsymbole, diese werden zunächst mit 0 für z_0 und b initialisiert. R_2 bleibt das Register für die Kopfposition, wobei der zuvor bestimmte Wert übernommen wird. Die folgenden Zeilen des Programms werden zum Teil gar nicht oder nur symbolisch numeriert.

	Befehl	*Wirkung*
14.	LOAD !0	$R_0 := 0$
15.	STORE 1	$R_1 := r(0) = 0$, R_1 enthält den Index des Anfangzustands, R_3 hat noch den Wert 0 für b
16.	IF $R_1 > 0$ GOTO q_1	der Zustand ist nicht z_0
17.	IF $R_3 > 0$ GOTO $p_{1,1}$	das Bandsymbol ist nicht b
	UP(0,0)	Unterprogramm zur Berechnung von $\delta(z_0, b)$ und Entscheidung über den nächsten Schritt

	Befehl	*Wirkung*
$p_{1,1}$	IF $R_3 > 1$ GOTO $p_{1,2}$	das Bandsymbol ist weder b noch $\vert$
	UP(0,1)	$\delta(z_0, \vert)$
$\vdots$	$\vdots$	$\vdots$
$p_{1,s}$	UP(0,s)	$\delta(z_0, x_s)$
q_1	IF $R_1 > 1$ GOTO q_2	der Zustand ist weder z_0 noch z_1
	IF $R_3 > 0$ GOTO $p_{2,1}$	
$\vdots$	$\vdots$	$\vdots$
$p_{r,s}$	UP(r,s)	$\delta(z_r, x_s)$

Insgesamt sind $(r+1)\cdot(s+1)$ Unterprogramme UP(i,j) aufzuführen. Dabei sind vier Typen zu unterscheiden, und zwar:

(a) Für $\delta(z_i, a_j) = (z_{i'}, a_{j'})$:

Befehl	*Wirkung*
LOAD !i'	$R_0 := i'$
STORE 1	$R_1 := r(0) = i'$, der Index des Folgezustands
LOAD !j'	$R_0 := j'$
STORE 3	$R_3 := r(0) = j'$, der Index des neuen Bandsymbols
STORE *2	$R_{r(2)} := r(0) = j'$, speichere diesen Index an der entsprechenden Bandposition
GOTO 16	gehe zum nächsten Schritt von T über

(b) Für $\delta(z_i, a_j) = (z_{i'}, l)$ (Linksbewegung):

Befehl	*Wirkung*
LOAD !i'	$R_0 := i'$
STORE 1	$R_1 := r(0) = i'$, der Index des Folgezustands
PRED 2	Linksbewegung
LOAD *2	$R_0 := r(r(2))$
STORE 3	R_3 enthält den Wert des neu gelesenen Bandsymbols
GOTO 16	gehe zum nächsten Schritt von T über

(c) Für $\delta(z_i, a_j) = (z_{i'}, r)$ (Rechtsbewegung):

wie (b), jedoch Austausch von PRED 2 durch SUCC 2.

(d) Für $\delta(z_i, a_j) = (z_{i'}, s)$ (Ende der Arbeit von T):

Befehl	*Wirkung*
LOAD !i'	$R_0 := i'$
STORE 1	$R_1 := r(0) = i'$, der Index des Folgezustands
GOTO Schluß	gehe zur Ausgabe über

Wenn die Turingmaschine T hält, die Registermaschine also die Befehle gemäß (d) ausgeführt hat, liefert die RAM durch das folgende Programmteil schließlich das Ergebnis.

	Befehl	*Wirkung*
Schluß	LOAD !0	$R_0 := 0$
	STORE 1	$R_1 := r(0) = 0$, Zähler für die Zahl der Striche des Resultats
	PRED 2	T steht auf dem letzten Strich $\vert$ des Resultats
P	SUCC 1	der Strich wird gezählt
	PRED 2	der Kopf wandert nach links
	LOAD *2	
	STORE 3	falls ein Strich vorhanden, hat R_3 den Wert 1
	IF $R_3 > 0$ GOTO P	ein weiterer Strich soll gezählt werden
	LOAD 1	Übertragung des Ergebnisses in den Akkumulator
	WRITE	schreibe Ergebnis in die Ausgabe, dies ist $f(y_1, \ldots, y_n) + 1$
	STOP	

Wir kommen nun zur umgekehrten Beweisrichtung. Wir werden hierfür nur eine Beweisskizze angeben, da die Einzelheiten des Beweises sehr aufwendig sind, es andererseits aber einsichtig ist, daß die Details mit entsprechendem Arbeitsaufwand ausgefüllt werden können. Es sei also eine Registermaschine vorgegeben, die f berechnet. Wir müssen eine Turingmaschine angeben, die f Turing-berechnet. Aufgrund der Überlegungen von Abschnitt 2.3(d) geben wir eine Turingmaschine T mit 6 Bändern und Köpfen an, die das Argument von f auf einem Eingabeband hält und das Ergebnis schließlich auf das Ausgabeband schreibt. Die simulierende Turingmaschine nach Abschnitt 2.3(d) besitzt dann ein Band, das in 12 Spuren aufgeteilt ist. Sie kann durch Einführung einiger initialer und finaler Schritte leicht so abgeändert werden, daß sie zunächst mit nur einer Spur, dem Eingabeband, beginnt, das Band dann in 12 Spuren aufteilt und schließlich mit nur einer Spur, dem Ausgabeband, endet. Auf diese Weise ist dann f auch gemäß Definition 2.4.1 Turing-berechenbar.

Unsere Turingmaschine T hat die folgenden Bänder, wobei in der Abbildung eine Bandinschrift zu einem späteren Zeitpunkt eingetragen ist.

Band 1 (Eingabe)	b	$\vert$	$\vert$	$\vert$	b	$\vert$	$\vert$	$\vert$	$\vert$	b	b	$\cdots$				
Band 2 (Speicher)	b	b	$\vert$	$\vert$	b	$\vert$	b	b	$\vert$	b	$\vert$	b	b	b	b	$\cdots$
Band 3 (Akkumulator)	b	$\vert$	$\vert$	$\vert$	$\vert$	b	$\cdots$									
Band 4 (Befehlszähler)	b	$\vert$	$\vert$	$\vert$	b	$\cdots$										
Band 5 (Hilfsband)	b	$\vert$	b	b	b	$\cdots$										
Band 6 (Ausgabe)	b	b	b	b	b	$\cdots$										

Auf Band 1 wird die Eingabe dargestellt, im Beispiel für ein 2-stelliges Argument $(y_1, y_2) = (2, 3)$ der Funktion f. Auf Band 2 wird der Speicher simuliert. Es werden

nur die bis dahin angesprochenen Register ohne Akkumulator in der Form

$$bb|^i b|^n bb|^j b|^m bbbb\ldots$$

abgespeichert. Diese Bandinschrift bedeutet, daß das Register R_i den Wert $r(i) = n$ und das Register R_j den Wert $r(j) = m$ beinhaltet. Im Falle des Wertes $m = 0$ gibt es drei aufeinanderfolgende Vorkommen von b. Vier solche aufeinanderfolgende Vorkommen bedeuten, daß keine weiteren Register mehr rechts davon vorkommen. Im Beispiel finden wir die Register R_2 und R_1, jeweils mit dem Wert 1. Auf dem Speicherband werden die Registerwerte niemals überschrieben, sondern ein zu ändernder Wert wird hinter die bisherigen Register angehängt. Um also den Inhalt eines Registers zu lesen, muß die Turingmaschine auf Band 2 ganz nach hinten gehen, also nach $bbbb$, um dann nach links laufend hinter einem Vorkommen von bb die passende Registernummer zu suchen. Wegen der Möglichkeit der indirekten Adressierung kann von vornherein nicht eine feste Obergrenze für eine Registernummer angegeben werden. Das Suchen der Nummer erfolgt i. allg. durch Vergleich mit einer auf dem Hilfsregister abgelegten Nummer. Das erste so gefundene Register liefert dann in den rechts daneben liegenden Feldern den aktuellen Wert. Wird ein Register nicht gefunden, so ist sein Wert 0. Der Wert 0 eines Registers muß daher auch nicht eingetragen werden, sofern nicht zuvor ein anderer Wert gesetzt war.

Der Wert $a \in \mathbb{N}_0$ des Akkumulators wird auf Band 3 durch $b|^a b\ldots$ dargestellt. Im Beispiel hat der Akkumulator den Wert 4. Auf Band 4 befindet sich der Befehlszähler, dessen Werte auf dieselbe Weise notiert werden. Band 5 dient als Hilfsband, wo Zwischenrechnungen abgelegt werden können. Am Ende wird auf Band 6 die Ausgabe $f(y_1, \ldots, y_n)$ in der Form

$$b|^{f(y_1,\ldots,y_n)+1}b$$

geschrieben.

Initial ist nur das Band 1 mit der Eingabe beschrieben, alle anderen Bänder sind mit lauter Blankzeichen belegt. Zu Beginn beschreibt die Turingmaschine das Band 4 mit $b|b\ldots$, also mit der Nummer 1 des ersten auszuführenden Befehls der Registermaschine. Allgemein beginnt die Simulation der Ausführung eines Befehls der Registermaschine damit, daß auf Band 4 die Nummer des nächsten Befehls gelesen wird. Da es nur endlich viele Befehle gibt, kann sich diese Zahl in einer geeigneten Zustandskomponente gemerkt werden. Entsprechend diesem Zustand durchläuft die Turingmaschine eine Reihe von Aktionen, die die Simulation des Befehls bewirken. Für jeden Befehl des Programms der RAM müssen also entsprechende Zustände und Überführungen der Turingmaschine definiert werden. Wir wollen für einige Befehle die Arbeitsweise der Turingmaschine beschreiben.

Beim Befehl READ nehmen wir an, das bereits $k-1$, $k \geq 1$, Argumente gelesen worden sind und nun der Kopf 1 auf dem ersten Blankzeichen von $b|^{y_k+1}b$ steht. Zunächst löscht der Kopf 3 den Akkumulator. Dann wird durch paralleles Zusammenwirken von Kopf 1 und Kopf 3 $b|^{y_k+1}b\ldots$ auf Band 3 geschrieben. Anschließend wird der Befehlszähler durch Kopf 4 um 1 erhöht.

Bei STORE i sucht Kopf 2 das Ende des Speicherbandes, also $bbbb$. Auf dem dritten dieser vier Blankzeichen beginnt dann nach rechts der Eintrag von $|^i b|^m bb\ldots$. Da i eine feste Zahl des Programms ist, kann durch entsprechende Zustandsänderungen festgestellt werden, wann i Striche durch Kopf 2 geschrieben worden sind. Der Wert

von m wird dagegen während des Schreibens durch Kopf 2 mit Hilfe von Kopf 3 aus dem Akkumulator geholt. Anschließend wird der Befehlszähler durch Kopf 4 um 1 erhöht.

Die indirekte Adressierung STORE $*i$ ist komplizierter. Die Zahl m des Akkumulators soll in das Register R_j geschrieben werden, wobei der Wert von j im Register R_i gespeichert ist. Folglich sucht für $i \neq 0$ Kopf 2 von rechts kommend das erste Vorkommen von $bb|^i b|^j$ auf Band 2. Da i eine feste Zahl des Programms ist, kann die Überprüfung der Zahl i wieder durch entsprechende Zustandänderungen erfolgen. Für $i = 0$ wird die Zahl j dem Band 3 entnommen. Ist ein solches Vorkommen gefunden, dann wird zunächst das Hilfsband 5 gelöscht und anschließend j durch gleichzeitige Arbeit von Kopf 2 und Kopf 5 in der Form $b|^j$ auf Band 5 geschrieben. Wird i nicht gefunden, so ist $j = 0$ anzunehmen. Für $j \neq 0$ wird danach unter Benutzung des Hilfsbandes und des Akkumulators $bb|^j b|^m$ hinter den letzten Strich des Speicherbandes geschrieben und dann der Befehlszähler um 1 erhöht. Für $j = 0$ bleibt wegen $r(r(i)) = r(0) := r(0)$ der Wert des Akkumulators unverändert, es wird nur der Befehlszähler erhöht.

Es dürfte keine Schwierigkeiten machen, auch die anderen Befehle aus Definition 2.8.2 in ähnlicher Weise zu beschreiben. Bei den Sprüngen muß natürlich das Band 4 gelöscht werden, bevor der neue Stand des Befehlszählers eingetragen werden kann.

Ein WRITE-Befehl wird simuliert, indem mit Hilfe von Kopf 3 und Kopf 6 der Inhalt des Akkumulators auf das Ausgabeband 6 geschrieben wird. Beim ersten Auftreten von WRITE ist dies $b|^{f(y_1,\ldots,y_n)+1}b\ldots$, also das gewünschte Ergebnis der Berechnung von T. Die Registermaschine hält, wenn sie auf den Befehl STOP trifft. Dann stoppt auch unsere Turingmaschine T. Hält die Registermaschine nicht, dann auch nicht T. Wird nach dem ersten WRITE-Befehl die Zahl 0 in das Ausgabeband eingetragen, bleibt das Ausgabeband der Registermaschine also leer, dann geht die Turingmaschine sofort in eine unendliche Schleife über, da auch in diesem Fall nach Definition 2.8.3 der Wert von $f(y_1, \ldots, y_n)$ als nicht definiert gilt. Die Turingmaschine berechnet also dieselbe Funktion wie die Registermaschine. $\square$

Damit haben wir auf zwei verschiedene, aber äquivalente Arten die Berechenbarkeit von partiellen Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$ charakterisiert. Im folgenden Kapitel werden wir eine weitere äquivalente Methode zur Definition von berechenbaren partiellen Funktionen kennenlernen.

3 Rekursive Funktionen

In den bisherigen Betrachtungen haben wir uns bei der Berechnung von Funktionen immer auf Automaten und Maschinen bezogen, d.h. auf Systeme, die entweder in einem oder in endlich vielen Schritten ein Resultat liefern, sofern sie überhaupt eines liefern. Diese Vorgehensweise ist für die Informatik typisch, nicht jedoch für die Mathematik. Wir wollen die algorithmischen Möglichkeiten jetzt anders charakterisieren, nämlich über die Klasse der Funktionen, die man erhält, wenn man von Grundfunktionen mit gewissen Operationen ausgeht.

3.1 Primitiv-rekursive Funktionen

Im folgenden wollen wir uns wie in Kapitel 2 mit zahlentheoretischen partiellen Funktionen $f : \mathbb{N}_0^n \to \mathbb{N}_0$, $n \in \mathbb{N}_0$, beschäftigen, also mit Funktionen, die auf einer Teilmenge von $\mathbb{N}_0^n$ definiert sind. Speziell werden auch *totale* Funktionen betrachtet, die für alle Elemente aus $\mathbb{N}_0^n$ definiert sind. Eine besonders einfache Klasse von Funktionen, die offensichtlich im intuitiven Sinn berechenbar sind, wird durch Definition 3.1.1 gegeben.

Definition 3.1.1 Die *primitiv-rekursiven Grundfunktionen* werden durch die folgenden totalen Funktionen gegeben:

(a) $C_0^{(0)} = 0$ (*nullstellige Nullfunktion*),
(b) $S : \mathbb{N}_0 \to \mathbb{N}_0$ mit $S(x_1) = x_1 + 1$ (*Nachfolgerfunktion*) und
(c) $U_i^{(n)} : \mathbb{N}_0^n \to \mathbb{N}_0$ mit $U_i^{(n)}(x_1, \ldots, x_n) = x_i$ für $i = 1, \ldots, n$ (*Projektionsfunktionen*). □

Bei $C_0^{(0)}$ deutet (0) auf die Stelligkeit der Funktion und der untere Index 0 auf ihren errechneten konstanten Wert hin.

Auf der Basis der primitiv-rekursiven Grundfunktionen können wir mit Hilfe der Konstruktionen der beiden nächsten Definitionen eine schon recht umfangreiche Klasse von Funktionen gewinnen.

Definition 3.1.2 Es seien $n, r \in \mathbb{N}_0$, $r \geq 1$, und $h_i : \mathbb{N}_0^n \to \mathbb{N}_0$ mit $i = 1, \ldots, r$ und $g : \mathbb{N}_0^r \to \mathbb{N}_0$ seien partielle Funktionen. Eine partielle Funktion $f : \mathbb{N}_0^n \to \mathbb{N}_0$ *geht aus g durch Einsetzung von* $h_1, \ldots, h_r$ *hervor*, wenn

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_r(x_1, \ldots, x_n))$$

für beliebige $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$ gilt. □

Die rechte und damit auch die linke Seite der Gleichung sind offenbar genau dann definiert, wenn sowohl die Werte $h_1(x_1, \ldots, x_n), \ldots, h_r(x_1, \ldots, x_n)$ definiert sind als auch $g(h_1(x_1, \ldots, x_n), \ldots, h_r(x_1, \ldots, x_n))$.

Definition 3.1.3 Es sei $n \in I\!N_0$, und $g : I\!N_0^n \to I\!N_0$ und $h : I\!N_0^{n+2} \to I\!N_0$ seien partielle Funktionen. Eine partielle Funktion $f : I\!N_0^{n+1} \to I\!N_0$ heißt *induktiv-rekursiv durch g und h definiert*, wenn

$$\begin{aligned} f(x_1, \ldots, x_n, 0) &= g(x_1, \ldots, x_n) \text{ und} \\ f(x_1, \ldots, x_n, y+1) &= h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{aligned}$$

für beliebige $(x_1, \ldots, x_n) \in I\!N_0^n$ und $y \in I\!N_0$ gilt. □

Man spricht vom *Induktions-Rekursionsschema* oder auch vom *primitiven Rekursionsschema*. Mit Hilfe des Induktions-Rekursionsschemas kann man bei gegebenen Funktionen g und h die Funktion f bestimmen. Speziell für $n = 1$ gilt zunächst $f(x, 0) = g(x)$. Damit erhält man

$$\begin{aligned} f(x,1) &= f(x, 0+1) = h(x, 0, f(x,0)), \\ f(x,2) &= f(x, 1+1) = h(x, 1, f(x,1)) \text{ usw.} \end{aligned}$$

Für alle $x, n \in I\!N_0$ sind somit die Werte $f(x, n)$ eindeutig konstruierbar.

Definition 3.1.4 Es sei $f : I\!N_0^n \to I\!N_0$ eine Funktion. f heißt *primitiv-rekursiv*, wenn f eine primitiv-rekursive Grundfunktion ist oder sich aus primitiv-rekursiven Grundfunktionen durch endlich viele Anwendungen der Einsetzung oder des Induktions-Rekursionsschemas gewinnen läßt. □

Die primitiv-rekursiven Grundfunktionen sind totale Funktionen. Sind weiter $h_1, \ldots, h_r$ und g total, dann ist auch $g \circ (h_1, \ldots, h_r)$ total. Sind g und h total, dann ist die induktiv-rekursiv durch g und h definierte Funktion f total. Somit sind alle primitiv-rekursiven Funktionen totale Funktionen.

Alle „üblichen" Funktionen sind primitiv-rekursiv, aber keineswegs alle Funktionen. Wir zeigen zunächst für einige bekannte Funktionen, daß sie primitiv-rekursiv sind.

Beispiel 3.1.1 (a) Die Addition. Es seien $x, y \in I\!N_0$. Dann gilt

$$\begin{aligned} +(x,0) &= U_1^{(1)}(x) \text{ und} \\ +(x, y+1) &= S(U_3^{(3)}(x, y, +(x,y))). \end{aligned}$$

Die Addition ist also durch das Induktions-Rekursionsschema definiert, wobei die Entsprechungen $f \triangleq +$, $g \triangleq U_1^{(1)}$ und $h \triangleq S \circ U_3^{(3)}$ gelten und h aus S durch Einsetzung von $U_3^{(3)}$ hervorgeht.

(b) Für $n, k \in I\!N_0$ sind die konstanten Funktionen $C_k^{(n)} : I\!N_0^n \to I\!N_0$ mit $C_k^{(n)}(x_1, \ldots, x_n) = k$ sukzessive wie folgt gegeben:
für $n = 0$ durch die Grundfunktion $C_0^{(0)}$ sowie durch die Einsetzung $C_{k+1}^{(0)} = S(C_k^{(0)})$, für $n \geq 1$ durch das Induktions-Rekursionsschema, und zwar für $n = 1$ durch

$$\begin{aligned} C_k^{(1)}(0) &= C_k^{(0)} \text{ und} \\ C_k^{(1)}(y+1) &= C_k^{(1)}(y) \quad (g \triangleq C_k^{(0)} \text{ und } h \triangleq U_2^{(2)}), \end{aligned}$$

für $n = 2$ durch

$$\begin{aligned} C_k^{(2)}(x,0) &= C_k^{(1)}(x) \quad \text{und} \\ C_k^{(2)}(x,y+1) &= C_k^{(2)}(x,y) \quad (g \triangleq C_k^{(1)} \quad \text{und} \quad h \triangleq U_3^{(3)}) \end{aligned}$$

usw.

(c) Die Multiplikation. Es seien $x, y \in I\!N_0$. Dann gilt

$$\begin{aligned} *(x,0) &= C_0^{(1)}(x) \quad \text{und} \\ *(x,y+1) &= (+(U_1^{(3)}, U_3^{(3)}))(x,y,*(x,y)), \end{aligned}$$

wobei $+(U_1^{(3)}, U_3^{(3)})$ durch Einsetzung von $U_1^{(3)}$ und $U_3^{(3)}$ aus $+$ hervorgeht.

(d) Die Fakultät. Es sei $x \in I\!N_0$. Dann gilt

$$\begin{aligned} \text{fak}(0) &= C_1^{(0)} \quad \text{und} \\ \text{fak}(x+1) &= (*(S \circ U_1^{(2)}, U_2^{(2)}))(x, \text{fak}(x)). \end{aligned}$$

(e) Die Potenz. Es seien $x, y \in I\!N_0$. Dann gilt

$$\begin{aligned} x^0 &= C_1^{(1)} \quad \text{und} \\ x^{y+1} &= (*(U_1^{(3)}, U_3^{(3)}))(x,y,x^y). \end{aligned}$$

(f) Die Vorgängerfunktion $V : I\!N_0 \to I\!N_0$. Es sei $y \in I\!N_0$. Dann gilt

$$\begin{aligned} V(0) &= C_0^{(0)} \quad (= 0) \quad \text{und} \\ V(y+1) &= U_1^{(2)}(y, V(y)) \quad (= y). \end{aligned}$$

(g) Die modifizierte Differenz (die Monus-Operation, siehe auch Beispiel 2.4.2). Es seien $x, y \in I\!N_0$. Dann soll gelten

$$x \dot{-} y = \begin{cases} 0 & \text{für } x < y \\ x - y & \text{für } x \geq y. \end{cases}$$

Dies wird durch

$$\begin{aligned} x \dot{-} 0 &= U_1^{(1)}(x) \quad (= x) \quad \text{und} \\ x \dot{-} (y+1) &= V(U_3^{(3)}(x,y,x \dot{-} y)) \quad (= V(x \dot{-} y)) \end{aligned}$$

erreicht.

(h) Der Abstand. Es seien $x, y \in I\!N_0$. Dann gilt offenbar

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

(i) Die Signum-Funktion. Es sei $x \in I\!N_0$. Dann ist die Signum-Funktion durch

$$\text{sign}(x) = \begin{cases} 0 & \text{für } x = 0 \\ 1 & \text{für } x > 0 \end{cases}$$

definiert. Dies wird durch

$$\begin{aligned} \text{sign}(0) &= C_0^{(0)} \quad (= 0) \quad \text{und} \\ \text{sign}(x+1) &= C_1^{(2)}(x, \text{sign}(x)) \end{aligned}$$

erreicht. Entsprechend ist auch die durch $\overline{\text{sign}}(x) = 1 - \text{sign}(x)$ gegebene $\overline{\text{sign}}$-Funktion eine primitiv-rekursive Funktion. □

Die Grundfunktionen sind offensichtlich im intuitiven Sinn berechenbar. Unmittelbar ist klar, daß der Prozeß der Einsetzung und der Gewinnung von Funktionen über das Induktions-Rekursionsschema intuitiv wieder zu berechenbaren Funktionen führt. Aufgrund der Churchschen These ist somit jede primitiv-rekursive Funktion Turing-berechenbar. Wir leiten dieses Ergebnis formal her.

Satz 3.1.1 Primitiv-rekursive Funktionen sind normiert Turing-berechenbar.

Beweis: Der Beweis erfolgt durch Induktion über die Struktur der Definition der primitiv-rekursiven Funktionen. Zunächst wird gezeigt, daß die Grundfunktionen normiert Turing-berechenbar sind. Dann muß bewiesen werden, daß alle Funktionen, die sich durch die Einsetzung oder durch das Induktions-Rekursionsschema aus normiert Turing-berechenbaren Funktionen ergeben, wieder normiert Turing-berechenbar sind. Man beachte dabei, daß sich nach Definition 2.4.2 bei der normierten Turing-Berechenbarkeit am Ende der Rechnung neben dem Funktionswert auch die Argumentwerte auf dem Band der Turingmaschine befinden.

Wir betrachten die Grundfunktionen. Die Nullfunktion $C_0^{(0)}$ wird durch die Turingmaschine $\mathbf{|}\,\mathbf{r}$ berechnet. Offenbar wird die Bandinschrift $\ldots\underline{b}\ldots$, die einem 0-stelligen Argument (keinem Argument) entspricht, durch $\mathbf{|}\,\mathbf{r}$ in $\ldots b|\underline{b}\ldots$ überführt. Die Nachfolgerfunktion S wird durch $\mathbf{K}_1\mathbf{|}\,\mathbf{r}$ berechnet. Für $k \in I\!N_0$ gilt nämlich

$$\ldots b|^{k+1}\underline{b}\ldots \overset{\mathbf{K}_1}{\mapsto} \ldots b|^{k+1}b|^{k+1}\underline{b}\ldots \overset{\mathbf{|r}}{\mapsto} \ldots b|^{k+1}b|^{k+2}\underline{b}\ldots.$$

Die Kopiermaschinen $\mathbf{K}_{n+1-i}$ realisieren die Projektionsabbildungen $U_i^{(n)}$, $i = 1, \ldots, n$, denn für $(x_1, \ldots, x_n) \in I\!N_0^n$ erhalten wir

$$\ldots b|^{x_1+1}b|^{x_2+1}b\ldots b|^{x_n+1}\underline{b}\ldots \overset{\mathbf{K}_{n+1-i}}{\longmapsto} \ldots b|^{x_1+1}b\ldots b|^{x_n+1}b|^{x_i+1}\underline{b}\ldots.$$

Wir kommen jetzt zur Einsetzung. Die Funktionen $g, h_1, \ldots, h_r$ seien normiert Turing-berechenbar durch die Turingmaschinen $T, T_1, \ldots, T_r$. Dann ist $f = g \circ (h_1, \ldots, h_r)$ normiert Turing-berechenbar durch

$$T_1(\mathbf{K}_{n-1+2}^n T_2 \mathbf{V}^n)\ldots(\mathbf{K}_{n-1+r}^n T_r \mathbf{V}^n) T \mathbf{V}^r,$$

wobei $\mathbf{V}$ die Verschiebemaschine aus Beispiel 2.2.8 ist. Speziell für $r = 1$ ist die Turingmaschine $T_1 T \mathbf{V}$ zu betrachten. Es sei $(x_1, \ldots, x_n) \in I\!N_0^n$ das Argument von f. Dann erhalten wir die Überführungen

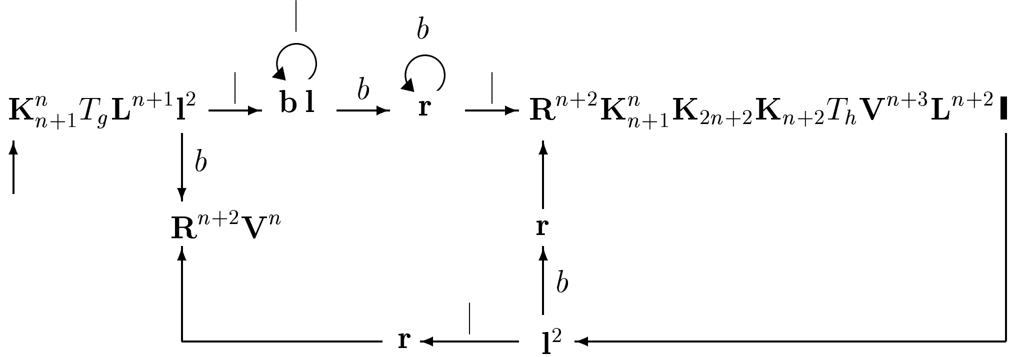
$$\begin{aligned}
&\ldots b\underbrace{|^{x_1+1}b|^{x_2+1}b\ldots b|^{x_n+1}}_{w_1}\underline{b}\ldots \overset{T_1}{\mapsto} \ldots bw_1b|^{h_1(x_1,\ldots,x_n)+1}\underline{b}\ldots \\
\overset{\mathbf{K}_{n-1+2}^n}{\longmapsto}\ &\ldots b\underbrace{w_1b|^{h_1(x_1,\ldots,x_n)+1}bw_1}_{w_2}\underline{b}\ldots \overset{T_2}{\mapsto} \ldots bw_2b|^{h_2(x_1,\ldots,x_n)+1}\underline{b}\ldots \\
\overset{\mathbf{V}^n}{\mapsto}\ &\ldots bw_1b|^{h_1(x_1,\ldots,x_n)+1}b|^{h_2(x_1,\ldots,x_n)+1}\underline{b}\ldots \mapsto \ldots\ldots \\
\mapsto\ &\ldots bw_1b|^{h_1(x_1,\ldots,x_n)+1}b\ldots b|^{h_{r-1}(x_1,\ldots,x_n)+1}\underline{b}\ldots \\
\overset{\mathbf{K}_{n-1+r}^n T_r \mathbf{V}^n}{\longmapsto}\ &\ldots bw_1b\underbrace{|^{h_1(x_1,\ldots,x_n)+1}b\ldots b|^{h_r(x_1,\ldots,x_n)+1}}_{w_3}\underline{b}\ldots
\end{aligned}$$

$$\overset{T}{\mapsto} \ldots bw_1bw_3b|^{g(h_1(x_1,\ldots,x_n),\ldots,h_r(x_1,\ldots,x_n))+1}\underline{b}\ldots$$

$$\overset{\mathbf{V}^r}{\mapsto} \ldots bw_1b|^{g(h_1(x_1,\ldots,x_n),\ldots,h_r(x_1,\ldots,x_n))+1}\underline{b}\ldots.$$

Wir sehen, daß $g(h_1(x_1,\ldots,x_n),\ldots,h_r(x_1,\ldots,x_n))$ der errechnete Funktionswert ist.

Schließlich seien g und h normiert Turing-berechenbar durch T_g und T_h. Dann wird die durch g und h induktiv-rekursiv definierte Funktion f normiert Turing-berechnet durch die Turingmaschine



Dies erkennen wir wie folgt. Für $(x_1,\ldots,x_n)\in I\!N_0^n$ setzen wir zur Abkürzung

$$\mathbf{x}=|^{x_1+1}b|^{x_2+1}b\ldots b|^{x_n+1},\ y=|^{y+1}\quad\text{und}\quad f_i=|^{f(x_1,\ldots,x_n,i)+1}.$$

Insbesondere gilt $f_0=|^{g(x_1,\ldots,x_n)+1}$. Die Berechnung von $f(x_1,\ldots,x_n,y)$ mit $y\geq 0$ wird mit Hilfe der folgenden Schritte durchgeführt:

$$\ldots b\mathbf{x}by\underline{b}\ldots \overset{\mathbf{K}^n_{n+1}}{\longmapsto} \ldots b\mathbf{x}byb\mathbf{x}\underline{b}\ldots \overset{T_g}{\mapsto} \ldots b\mathbf{x}byb\mathbf{x}bf_0\underline{b}\ldots \overset{\mathbf{L}^{n+1}}{\longmapsto} \ldots b\mathbf{x}by\underline{b}\mathbf{x}bf_0b\ldots.$$

Bei der folgenden Anwendung von $\mathbf{l}^2$ sind jetzt zwei Fälle zu unterscheiden. Für $y=0$ ist der gewünschte Wert bereits durch f_0 bestimmt, und die Arbeit der Turingmaschine wird durch

$$\ldots b\mathbf{x}b|\underline{b}\mathbf{x}bf_0b\ldots \overset{\mathbf{l}^2}{\mapsto} \ldots b\mathbf{x}\underline{b}|b\mathbf{x}bf_0b\ldots \overset{\mathbf{R}^{n+2}}{\longmapsto} \ldots b\mathbf{x}byb\mathbf{x}bf_0\underline{b}\ldots \overset{\mathbf{V}^n}{\mapsto} \ldots b\mathbf{x}bybf_0\underline{b}\ldots$$

abgeschlossen. Damit ist $f(x_1,\ldots,x_n,0)=g(x_1,\ldots,x_n)$ berechnet.

Für $y\geq 1$ erhalten wir

$$\ldots b\mathbf{x}b|^{y+1}\underline{b}\mathbf{x}bf_0b\ldots \overset{\mathbf{l}^2}{\mapsto} \ldots b\mathbf{x}b|^{y-1}\underline{|}\,|b\mathbf{x}bf_0b\ldots \overset{(\mathbf{bl})^y}{\longmapsto} \ldots b\mathbf{x}\underline{b}b^y|b\mathbf{x}bf_0b\ldots$$

$$\overset{\mathbf{r}^{y+1}}{\longmapsto} \ldots b\mathbf{x}bb^y\underline{|}b\mathbf{x}bf_0b\ldots \overset{\mathbf{R}^{n+2}}{\longmapsto} \ldots b\mathbf{x}bb^y|b\mathbf{x}bf_0\underline{b}\ldots.$$

Allgemein nehmen wir an, daß nach i Schleifendurchläufen, $i=0,\ldots,y-1$, und nach Anwendung von $\mathbf{R}^{n+2}$ die Bandinschrift

$$\ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_i\underline{b}\ldots$$

vorliegt. Wie wir eben gesehen haben, ist dies für $i=0$ richtig. Es gilt dann

$$\ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_i\underline{b}\ldots \overset{\mathbf{K}^n_{n+1}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_ib\mathbf{x}\underline{b}\ldots$$

$$\overset{\mathbf{K}_{2n+2}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_ib\mathbf{x}b|^{i+1}\underline{b}\ldots \overset{\mathbf{K}_{n+2}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_ib\mathbf{x}b|^{i+1}bf_i\underline{b}\ldots$$

$$\overset{T_h}{\longmapsto} \ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_ib\mathbf{x}b|^{i+1}bf_ibf_{i+1}\underline{b}\ldots .$$

An dieser Stelle ist der Wert

$$f(x_1,\ldots,x_n,i+1) = h(x_1,\ldots,x_n,i,f(x_1,\ldots,x_n,i))$$

berechnet. Weiter erhalten wir dann

$$\ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_ib\mathbf{x}b|^{i+1}bf_ibf_{i+1}\underline{b}\ldots \overset{\mathbf{V}^{n+3}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i}|^{i+1}b\mathbf{x}bf_{i+1}\underline{b}\ldots$$

$$\overset{\mathbf{L}^{n+2}\mathbf{l}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i-1}\underline{|}|^{i+1}b\mathbf{x}bf_{i+1}b\ldots .$$

Bei der folgenden Anwendung von $\mathbf{l}^2$ sind wieder zwei Fälle möglich. Für $y = i+1$ ist der gewünschte Wert bereits durch f_{i+1} bestimmt, und die Arbeit wird durch

$$\ldots b\mathbf{x}b\underline{|}|^{i+1}b\mathbf{x}bf_{i+1}b\ldots \overset{\mathbf{l}^2}{\mapsto} \ldots b\underline{\mathbf{x}}b||^{i+1}b\mathbf{x}bf_{i+1}b\ldots \overset{\mathbf{r}\ \mathbf{R}^{n+2}\mathbf{V}^n}{\longmapsto} \ldots b\mathbf{x}bybf_{i+1}\underline{b}\ldots$$

beendet. Dabei bedeutet $\underline{\mathbf{x}}$, daß die Turingmaschine über dem letzten Strich | von $\mathbf{x}$ steht. Insgesamt ist somit das Ergebnis $f(x_1,\ldots,x_n,y) = f(x_1,\ldots,x_n,i{+}1)$ berechnet.

Für $y > i+1$ erhalten wir

$$\ldots b\mathbf{x}bb^{y-i-1}\underline{|}|^{i+1}b\mathbf{x}bf_{i+1}b\ldots \overset{\mathbf{l}^2}{\mapsto} \ldots b\mathbf{x}b^{y-i-2}\underline{b}b|^{i+2}b\mathbf{x}bf_{i+1}b\ldots$$

$$\overset{\mathbf{r}\mathbf{R}^{n+2}}{\longmapsto} \ldots b\mathbf{x}bb^{y-i-1}|^{i+2}b\mathbf{x}bf_{i+1}\underline{b}\ldots .$$

Wir sehen also, daß unsere Annahme über die Gestalt der Bandinschrift nach i Schleifendurchläufen und nach Anwendung von $\mathbf{R}^{n+2}$ richtig ist. Insgesamt folgt, daß für $y \geq 1$ während des y-ten Schleifendurchlaufs der Funktionswert $f(x_1,\ldots,x_n,y)$ berechnet wird. Damit ist der Satz bewiesen. □

3.2 Die Ackermann-Funktion

Die Frage ist, ob die Klasse der primitiv-rekursiven Funktionen gleich der Klasse aller Turing-berechenbaren Funktionen ist. In diesem Fall wäre sie nach der Churchschen These auch gleich der Klasse aller intuitiv berechenbaren Funktionen. Das kann allerdings schon deswegen nicht gelten, weil es Turing-berechenbare nicht-totale Funktionen gibt (z.B. die leere Funktion), die als nicht-totale Funktionen nicht primitiv-rekursiv sein können. Am Beispiel der Ackermann-Funktion wird gezeigt, daß sogar totale Turing-berechenbare Funktionen existieren, die nicht primitiv-rekursiv sind.

Zunächst wollen wir einige Überlegungen durchführen, die schließlich zur Definition der Ackermann-Funktion führen werden. Die Grundidee dabei ist, eine Funktion zu finden, die in gewisser Weise schneller wächst als jede primitiv-rekursive Funktion. Dazu betrachten wir die Folge der immer stärker wachsenden Funktionen Summe, Produkt, Potenz. Da die Potenz aus dem Produkt auf ähnliche Weise entsteht wie das Produkt aus der Summe, läßt sich diese Folge verlängern: Für $n = 1, 2, 3$ werde mit

$f_n(x,y)$ die Summe, das Produkt bzw. die Potenz bezeichnet. Wir fügen die Funktion f_0 mit $f_0(x,y) = S(x)$ hinzu. Dann erhalten wir

$$\begin{aligned} f_1(x,0) &= x \\ f_1(x,y+1) &= f_0(f_1(x,y),x), \\[1em] f_2(x,0) &= 0 \\ f_2(x,y+1) &= f_1(f_2(x,y),x), \\[1em] f_3(x,0) &= 1 \\ f_3(x,y+1) &= f_2(f_3(x,y),x). \end{aligned}$$

Wir erkennen sofort, daß diese Definitionen, abgesehen von f_0, dem Gleichungssystem

$$\begin{aligned} f_{n+1}(x,0) &= g_{n+1}(x) \\ f_{n+1}(x,y+1) &= f_n(f_{n+1}(x,y),x) \end{aligned}$$

genügen. Wählen wir für g_{n+1} geeignete primitiv-rekursive Funktionen, so erhalten wir eine Folge f_n von Funktionen ($n = 0,1,2,\ldots$), welche die Anfangsfolge S, Summe, Produkt, Potenz fortsetzt. Wir stellen fest, daß jedes f_n primitiv-rekursiv ist, denn f_{n+1} ist induktiv-rekursiv durch g_{n+1} und h_n mit $h_n(x_1,x_2,x_3) = f_n(x_3,x_1)$ definiert. Dabei geht h_n aus f_n durch Einsetzung von $U_3^{(3)}$ und $U_1^{(3)}$ hervor.

Nun wird die unendliche Folge f_n von Funktionen von zwei Argumenten durch eine einzige Funktion f von drei Argumenten mit

$$f(n,x,y) = f_n(x,y)$$

ersetzt. Dabei kommt der Index n als erstes Argument von f vor. Die Funktion f ist offensichtlich im intuitiven Sinn berechenbar. Sie erfüllt aufgrund der Definition von f_n die Funktionalgleichung

$$f(n+1,x,y+1) = f(n,f(n+1,x,y),x)$$

für alle $n,x,y \in I\!N_0$. Damit erhalten wir eine Art induktiver Definition von f, die jedoch für den Fall, daß das erste oder dritte Argument 0 ist, noch geeignet ergänzt werden muß. Diese Definition ist allgemeiner als die Definition, die beim Induktions-Rekursionsschema vorkommt. Man kann zeigen, daß f nicht primitiv-rekursiv ist. Wir beweisen ein entsprechendes Ergebnis für eine etwas einfachere Funktion, die wir jetzt aus f ableiten.

In der obigen Gleichung für f spielt x, das überall unverändert als Parameter auftaucht, eine geringere Rolle als die Variablen n und y, für die auch die Nachfolger vorkommen. Wenn wir x weglassen und anschließend den Buchstaben x für n verwenden, dann erhalten wir die dritte Gleichung der folgenden Definition. Die beiden anderen Gleichungen sind so gewählt, daß die weiteren Überlegungen möglich werden.

Definition 3.2.1 Die *Ackermann-Funktion* $f : I\!N_0^2 \to I\!N_0$ ist definiert durch
(a) $f(0,y) = y+1$,
(b) $f(x+1,0) = f(x,1)$ und
(c) $f(x+1,y+1) = f(x,f(x+1,y))$. $\square$

Wir wollen uns durch Induktion über x und dann über y überzeugen, daß durch (a) bis (c) die Funktion f für jedes x und y eindeutig festgelegt ist. Für $x = 0$ gilt nach (a) die Gleichung $f(0, y) = y + 1$ für alle $y \in \mathbb{N}_0$, und anders kann $f(0, y)$ nicht bestimmt werden. Die erste Induktionsannahme für die Induktion über x ist, daß $f(x, y)$ für ein festes $x \in \mathbb{N}_0$ und alle $y \in \mathbb{N}_0$ eindeutig definiert ist. Wir müssen zeigen, daß dann $f(x + 1, y)$ für alle $y \in \mathbb{N}_0$ ebenfalls eindeutig definiert ist. Dies beweisen wir durch Induktion über y. Für $y = 0$ können wir nur (b) verwenden und erhalten $f(x + 1, 0) = f(x, 1)$. Wegen der ersten Induktionsannahme ist $f(x, 1)$ und damit $f(x+1, 0)$ eindeutig bestimmt. Die zweite Induktionsannahme für die Induktion über y ist, daß $f(x+1, y)$ eindeutig definiert ist. $f(x+1, y+1)$ kann nur gemäß (c) berechnet werden und liefert aufgrund der zweiten und dann der ersten Induktionsannahme das eindeutig bestimmte Element $f(x, f(x + 1, y))$.

Die Rechnungen gemäß (a) und (b) können leicht durchgeführt werden. Durch den Induktionsbeweis ist dann klar, wie für andere Argumente die Berechnungen zu erfolgen haben. Die Ackermann-Funktion ist also berechenbar und natürlich auch total. Schon für kleine Argumentwerte ist die Berechnung jedoch sehr aufwendig. So ist z.B.

$$f(2, 1) = f(1, f(2, 0)) = f(1, f(1, 1)).$$

Wegen

$$f(1, 1) = f(0, f(1, 0)) = f(1, 0) + 1 = f(0, 1) + 1 = (1 + 1) + 1 = 3$$

folgt dann weiter

$$\begin{aligned} f(2, 1) &= f(1, 3) = f(0, f(1, 2)) = f(1, 2) + 1 = f(0, f(1, 1)) + 1 \\ &= f(0, 3) + 1 = (3 + 1) + 1 = 5. \end{aligned}$$

Satz 3.2.1 Für die Ackermann-Funktion $f : \mathbb{N}_0^2 \to \mathbb{N}_0$ gelten für alle $x, y \in \mathbb{N}_0$ die folgenden Aussagen:

(a) $y < f(x, y)$.
(b) $f(x, y) < f(x, y + 1)$.
(c) $f(x, y + 1) \leq f(x + 1, y)$.
(d) $f(x, y) < f(x + 1, y)$.
(e) $f(1, y) = y + 2$.
(f) $f(2, y) = 2y + 3$.
(g) Es seien $c_1, \ldots, c_n \in \mathbb{N}_0$. Dann existiert ein $c \in \mathbb{N}_0$, so daß für alle $x \in \mathbb{N}_0$ die Ungleichung
$$\sum_{i=1}^{n} f(c_i, x) < f(c, x)$$
gilt.

Beweis: Man beachte, daß in dem Beweis die Gleichungen aus Definition 3.2.1 ohne entsprechenden Hinweis verwendet werden.

(a) Durch Induktion über x zeigen wir $y < f(x, y)$ für alle $y \in \mathbb{N}_0$. Für $x = 0$ gilt $y < y + 1 = f(0, y)$. Die erste Induktionsvoraussetzung lautet, daß (a) für ein x und jedes y gilt. Zu zeigen ist, daß (a) für $x + 1$ und jedes y richtig ist. Diesen Beweis innerhalb des Induktionsbeweises über x führen wir durch Induktion über

y. Für $y = 0$ gilt nach der ersten Induktionsvoraussetzung $0 < 1 < f(x,1) = f(x+1,0)$. Die zweite Induktionsvoraussetzung ist die Gültigkeit von (a) für $x+1$ und y. Zu zeigen bleibt, daß (a) für $x+1$ und $y+1$ erfüllt ist. Aufgrund der beiden Induktionsvoraussetzungen erhalten wir

$$y < f(x+1,y) < f(x, f(x+1,y)) = f(x+1,y+1).$$

Daraus folgt $y+1 < f(x+1,y+1)$.

(b) Für $x = 0$ gilt

$$f(0,y) = y+1 < y+2 = f(0,y+1).$$

Mit Hilfe von (a) erhalten wir weiter für alle $x \in \mathbb{N}_0$

$$f(x+1,y) < f(x, f(x+1,y)) = f(x+1,y+1).$$

(c) Wir führen eine Induktion über y durch. Für $y = 0$ gilt

$$f(x,1) = f(x+1,0).$$

Nach (a) ergibt sich $y+1 < f(x,y+1)$. Mit Hilfe der Induktionsvoraussetzung folgt daraus $y+2 \leq f(x,y+1) \leq f(x+1,y)$. Unter Zuhilfenahme von (b) können wir

$$f(x,y+2) \leq f(x, f(x+1,y)) = f(x+1,y+1)$$

schließen.

(d) Wegen (b) und (c) gilt $f(x,y) < f(x,y+1) \leq f(x+1,y)$.

(e) Der Beweis erfolgt durch Induktion über y. Wir erhalten

$$f(1,0) = f(0,1) = 1+1 = 2 \quad \text{und}$$

$$f(1,y+1) = f(0, f(1,y)) = f(0,y+2) = y+3.$$

(f) Unter Verwendung von (e) führen wir eine Induktion über y durch. Es gilt

$$f(2,0) = f(1,1) = 3 \quad \text{und}$$

$$f(2,y+1) = f(1, f(2,y)) = f(1,2y+3) = 2y+5 = 2(y+1)+3.$$

(g) Es reicht aus, den Fall $n = 2$ zu zeigen. Es sei $d = \max\{c_1, c_2\}$ und $c = d+4$. Mit Hilfe von (b), (c), (d) und (f) erhalten wir

$$f(c_1,x) + f(c_2,x) \leq f(d,x) + f(d,x) < 2f(d,x) + 3$$

$$= f(2, f(d,x)) < f(2, f(d+3,x)) \leq f(d+2, f(d+3,x))$$

$$= f(d+3,x+1) \leq f(d+4,x) = f(c,x). \quad \square$$

Satz 3.2.2 Es sei $g : \mathbb{N}_0^n \to \mathbb{N}_0$ primitiv-rekursiv. Dann gibt es ein $c \in \mathbb{N}_0$, so daß für alle $x_1, \ldots, x_n \in \mathbb{N}_0$ die Ungleichung

$$(*) \qquad g(x_1, \ldots, x_n) < f(c, x_1 + \ldots + x_n)$$

gilt (für $n = 0$: $g < f(c,0)$).

Beweis: Der Beweis erfolgt wieder durch Induktion über die Struktur der Definition der primitiv-rekursiven Funktionen. Wir beweisen zunächst die Ungleichung (*) für die primitiv-rekursiven Grundfunktionen. Dann wird gezeigt, daß alle Funktionen, die sich durch den Einsetzungsprozeß oder durch das Induktions-Rekursionsschema aus Funktionen ergeben, für die jeweils (*) gilt, ebenfalls (*) erfüllen. Die Hinweise (a), (b) usw. beziehen sich im folgenden auf die Teilaussagen von Satz 3.2.1.

Wir betrachten die Grundfunktionen. Wegen $f(0,0) = 1$ gilt

$$C_0^{(0)} < f(0,0).$$

Weiter erhalten wir für die Nachfolgerfunktion

$$S(x) < f(1,x),$$

denn nach (d) ist $S(x) = f(0,x) < f(1,x)$. Für die Projektionsfunktionen ergeben sich für $i = 1, \ldots, n$ die Ungleichungen

$$U_i^{(n)}(x_1, \ldots, x_n) < f(0, x_1 + \ldots + x_n),$$

da $U_i^{(n)}(x_1, \ldots, x_n) = x_i < (x_1 + \ldots + x_n) + 1 = f(0, x_1 + \ldots + x_n)$ ist.

Wir kommen jetzt zum Einsetzungsprozeß. Es werde angenommen, daß es zu den Funktionen $g, g_1, \ldots, g_n$ Zahlen $c, c_1, \ldots, c_n \in \mathbb{N}_0$ gibt mit

$$\begin{aligned} g(x_1, \ldots, x_n) &< f(c, x_1 + \ldots + x_n) \quad \text{und} \\ g_j(y_1, \ldots, y_r) &< f(c_j, y_1 + \ldots + y_r) \end{aligned}$$

für $j = 1, \ldots, n$ und alle $x_1, \ldots, x_n \in \mathbb{N}_0$, $y_1, \ldots, y_r \in \mathbb{N}_0$. Weiter gehe die Funktion h aus g durch Einsetzung von $g_1, \ldots, g_n$ hervor, es gelte also

$$h(y_1, \ldots, y_r) = g(g_1(y_1, \ldots, y_r), \ldots, g_n(y_1, \ldots, y_r))$$

für alle $y_1, \ldots, y_r \in \mathbb{N}_0$. Dann erhalten wir

$$\begin{aligned} & h(y_1, \ldots, y_r) \\ & \quad = g(g_1(y_1, \ldots, y_r), \ldots, g_n(y_1, \ldots, y_r)) \\ & \quad < f(c, g_1(y_1, \ldots, y_r) + \ldots + g_n(y_1, \ldots, y_r)) \\ & \quad < f(c, f(c_1, y_1 + \ldots + y_r) + \ldots + f(c_n, y_1 + \ldots + y_r)) \quad \text{(nach (b))} \\ & \quad < f(c, f(c^*, y_1 + \ldots + y_r)) \quad \text{(mit geeignetem } c^* \text{ nach (g))} \\ & \quad < f(c + c^*, f(c + c^* + 1, y_1 + \ldots + y_r)) \quad \text{(nach (b), (d))} \\ & \quad = f(c + c^* + 1, y_1 + \ldots + y_r + 1) \\ & \quad \leq f(c + c^* + 2, y_1 + \ldots + y_r) \quad \text{(nach (c))} \end{aligned}$$

für alle $y_1 \ldots, y_r \in \mathbb{N}_0$.

Es muß noch der Induktions-Rekursionsprozeß behandelt werden. Zu den Funktionen g_1, g_2 gebe es Zahlen $c_1, c_2 \in \mathbb{N}_0$ mit

$$g_1(x_1, \ldots, x_n) < f(c_1, x_1 + \ldots + x_n)$$

für alle $x_1, \ldots, x_n \in \mathbb{N}_0$ und

$$g_2(x_1, \ldots, x_n, y, z) < f(c_2, x_1 + \ldots + x_n + y + z)$$

für alle $x_1, \ldots, x_n, y, z \in \mathbb{N}_0$. Die Funktion h sei induktiv-rekursiv gegeben durch

$$h(x_1, \ldots, x_n, 0) = g_1(x_1, \ldots, x_n) \quad \text{und}$$

$$h(x_1, \ldots, x_n, y+1) = g_2(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)).$$

Wir müssen zeigen, daß es eine Konstante $c \in \mathbb{N}_0$ gibt mit

$$h(x_1, \ldots, x_n, y) < f(c, x_1 + \ldots + x_n + y)$$

für alle $x_1, \ldots, x_n, y \in \mathbb{N}_0$. Statt dessen beweisen wir die stärkere Aussage, daß ein $c \in \mathbb{N}_0$ existiert mit

$$(1) \qquad h(x_1, \ldots, x_n, y) + x_1 + \ldots + x_n + y < f(c, x_1 + \ldots + x_n + y)$$

für beliebige $x_1, \ldots, x_n, y \in \mathbb{N}_0$. Dafür zeigen wir zunächst, daß es ein $c_1^* \in \mathbb{N}_0$ gibt mit

$$(2) \qquad g_1(x_1, \ldots, x_n) + x_1 + \ldots + x_n < f(c_1^*, x_1 + \ldots + x_n)$$

für alle $x_1, \ldots, x_n \in \mathbb{N}_0$. Mit Hilfe von $U_i^{(n)}(x_1, \ldots, x_n) < f(0, x_1 + \ldots + x_n)$ und (g) folgt dies aus

$$\begin{aligned} & g_1(x_1, \ldots, x_n) + x_1 + \ldots + x_n \\ & \quad = g_1(x_1, \ldots, x_n) + U_1^{(n)}(x_1, \ldots, x_n) + \ldots + U_n^{(n)}(x_1, \ldots, x_n) \\ & \quad < f(c_1, x_1 + \ldots + x_n) + f(0, x_1 + \ldots + x_n) + \ldots + f(0, x_1 + \ldots + x_n) \\ & \quad < f(c_1^*, x_1 + \ldots + x_n) \ \text{(mit geeignetem } c_1^* \text{ nach (g)).} \end{aligned}$$

In der gleichen Weise wird gezeigt, daß es eine Konstante $c_2^* \in \mathbb{N}_0$ gibt mit

$$(3) \qquad \begin{aligned} & g_2(x_1, \ldots, x_n, y, z) + x_1 + \ldots + x_n + y + z \\ & \qquad < f(c_2^*, x_1 + \ldots + x_n + y + z) \end{aligned}$$

für alle $x_1, \ldots, x_n, y, z \in \mathbb{N}_0$. Durch Induktion über y wird nun die Ungleichung (1) bewiesen, wobei wir

$$c = \max\{c_1^*, c_2^*\} + 1$$

mit den zuvor bestimmten c_1^*, c_2^* setzen. Unter Berücksichtigung von (d) und (2) erhalten wir

$$\begin{aligned} h(x_1, \ldots, x_n, 0) + x_1 + \ldots + x_n &= g_1(x_1, \ldots, x_n) + x_1 + \ldots + x_n \\ &< f(c_1^*, x_1 + \ldots + x_n) < f(c, x_1 + \ldots + x_n). \end{aligned}$$

Wir nehmen an, daß die Induktionsvoraussetzung (1) gilt und betrachten den Fall $y+1$. Unter Verwendung von (3) ergibt sich

$$\begin{aligned} & h(x_1, \ldots, x_n, y+1) + x_1 + \ldots + x_n + y + 1 \\ & \quad = g_2(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)) + x_1 + \ldots + x_n + y + 1 \\ & \quad < f(c_2^*, x_1 + \ldots + x_n + y + h(x_1, \ldots, x_n, y)) + 1 \\ & \quad < f(c_2^*, f(c, x_1 + \ldots + x_n + y)) + 1. \end{aligned}$$

Der letzte Term ist wegen der Definition von c und der Aussage (d)

$$\begin{aligned} & \le f(c-1, f(c, x_1 + \ldots + x_n + y)) + 1 \\ & = f(c, x_1 + \ldots + x_n + y + 1) + 1. \end{aligned}$$

Zweimal wurde in der Abschätzung das Kleinerzeichen verwendet, so daß insgesamt

$$h(x_1, \ldots, x_n, y+1) + x_1 + \ldots + x_n + y + 1 < f(c, x_1 + \ldots + x_n + y + 1)$$

folgt. Damit ist der Induktionsbeweis für die Ungleichung (1) abgeschlossen und der Satz bewiesen. □

Satz 3.2.3 Die Ackermann-Funktion ist nicht primitiv-rekursiv.

Beweis: Wir nehmen an, daß die Ackermann-Funktion f primitiv-rekursiv ist. Dann ist es auch die durch $g(x) = f(x, x)$ für alle $x \in \mathbb{N}_0$ definierte Funktion $g : \mathbb{N}_0 \to \mathbb{N}_0$. g entsteht nämlich aus f durch Einsetzung von h_1, h_2 mit $h_1 = h_2 = U_1^{(1)}$ (Identitätsfunktion). Daraus folgt nach Satz 3.2.2, daß ein $c \in \mathbb{N}_0$ existiert mit $g(x) < f(c, x)$ für alle $x \in \mathbb{N}_0$. Dies gilt insbesondere für $x = c$, woraus sich $g(c) < f(c, c) = g(c)$ ergibt. Das ist ein offensichtlicher Widerspruch. □

3.3 Der μ-Operator und μ-rekursive Funktionen

Es seien $x_1, \ldots, x_n \in \mathbb{N}_0$ mit $n \in \mathbb{N}$. Ein *n-stelliges Prädikat* ρ ist eine Aussage ρ über n natürliche Zahlen, die auf gewisse n-Tupel von Zahlen zutrifft. Wir schreiben $\rho(x_1, \ldots, x_n)$ oder sagen, daß $\rho(x_1, \ldots, x_n)$ wahr ist, wenn ρ auf $(x_1, \ldots, x_n)$ zutrifft. Zum Beispiel ist das Primzahlprädikat ein einstelliges Prädikat, das auf 2,3,5,... zutrifft und auf 0,1,4,6,... nicht.

Definition 3.3.1 Es sei $n \in \mathbb{N}$, ρ ein n-stelliges Prädikat und χ_ρ eine n-stellige Funktion. χ_ρ heißt *charakteristische Funktion von* ρ, wenn χ_ρ gegeben wird durch

$$\chi_\rho(x_1, \ldots, x_n) = \begin{cases} 1, & \text{falls } \rho(x_1, \ldots, x_n) \\ 0 & \text{sonst.} \end{cases} \qquad \square$$

Für ein Prädikat ρ und seine charakteristische Funktion χ_ρ gilt also

$$\rho(x_1, \ldots, x_n) \iff \chi_\rho(x_1, \ldots, x_n) = 1.$$

Definition 3.3.2 Es sei ρ ein n-stelliges Prädikat mit $n \in \mathbb{N}$. ρ heißt *primitiv-rekursiv*, wenn eine primitiv-rekursive Funktion f existiert mit

$$\rho(x_1, \ldots, x_n) \iff f(x_1, \ldots, x_n) = 0$$

für alle $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$. □

Jedes primitiv-rekursive Prädikat ist entscheidbar, da die primitiv-rekursive Funktion f im intuitiven Sinn berechenbar ist und somit effektiv überprüft werden kann, ob $\rho(x_1, \ldots, x_n)$ zutrifft. Die Zugehörigkeit eines n-Tupels $(x_1, \ldots, x_n)$ zu der Menge $E = \{(x_1, \ldots, x_n) \mid \rho(x_1, \ldots, x_n)\}$ kann also effektiv festgestellt werden. Nach Definition 2.7.2 (siehe Seite 44) ist folglich die Menge E entscheidbar.

Satz 3.3.1 Es sei $n \in \mathbb{N}$ und ρ ein n-stelliges Prädikat. ρ ist genau dann primitiv-rekursiv, wenn seine zugehörige charakteristische Funktion primitiv-rekursiv ist.

Beweis: Es sei f die nach Definition 3.3.2 zu ρ gehörende primitiv-rekursive Funktion. Es gilt offenbar $\chi_\rho = \overline{\text{sign}} \circ f$. Da f und $\overline{\text{sign}}$ primitiv-rekursiv sind, ist es auch χ_ρ.

Ist umgekehrt χ_ρ primitiv rekursiv, so definieren wir die primitiv-rekursive Funktion $f = 1 \dot{-} \chi_\rho$. Damit erhalten wir

$$f(x_1, \ldots, x_n) = 0 \iff \chi_\rho(x_1, \ldots, x_n) = 1 \iff \rho(x_1, \ldots, x_n)$$

für alle $x_1, \ldots, x_n \in \mathbb{N}_0$. Folglich ist ρ nach Definition 3.3.2 primitiv-rekursiv. □

Wir zeigen jetzt, daß Funktionen, die durch eine einfache Fallunterscheidung aus primitiv-rekursiven Funktionen entstehen, selbst primitiv-rekursiv sind.

Satz 3.3.2 Es sei $n \in \mathbb{N}$, ρ sei ein n-stelliges primitiv-rekursives Prädikat und $f, g : \mathbb{N}_0^n \to \mathbb{N}_0$ seien primitiv-rekursive Funktionen. Dann ist die durch

$$r(x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n), & \text{falls } \rho(x_1, \ldots, x_n) \\ g(x_1, \ldots, x_n) & \text{sonst} \end{cases}$$

für alle $x_1, \ldots, x_n \in \mathbb{N}_0$ definierte Funktion $r : \mathbb{N}_0^n \to \mathbb{N}_0$ primitiv-rekursiv.

Beweis: Wir erkennen sofort, daß

$$\begin{aligned} r(x_1, \ldots, x_n) = \chi_\rho(x_1, \ldots, x_n) \cdot f(x_1, \ldots, x_n) \\ + (1 \dot{-} \chi_\rho(x_1, \ldots, x_n)) \cdot g(x_1, \ldots, x_n) \end{aligned}$$

gilt. Die Funktion r wird also durch mehrfache Anwendung des Einsetzungsprozesses aus den primitiv-rekursiven Funktionen f, g, χ_ρ, $*$, $+$, $\dot{-}$ sowie $C_1^{(n)}$ gewonnen und ist somit selbst primitiv-rekursiv. □

Definition 3.3.3 Es sei $n \in \mathbb{N}_0$ und ρ ein $(n+1)$-stelliges Prädikat.

(a) Falls es zu $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$ ein $y \in \mathbb{N}_0$ mit $\rho(x_1, \ldots, x_n, y)$ gibt, so gibt es ein eindeutig bestimmtes kleinstes y mit $\rho(x_1, \ldots, x_n, y)$. Dieses y bezeichnen wir mit $\mu y \rho(x_1, \ldots, x_n, y)$. Existiert dagegen zu $(x_1, \ldots, x_n)$ kein y mit $\rho(x_1, \ldots, x_n, y)$, so ist $\mu y \rho(x_1, \ldots, x_n, y)$ undefiniert. μ heißt *unbeschränkter μ-Operator.*

(b) Falls es zu $(x_1, \ldots, x_n, y) \in \mathbb{N}_0^{n+1}$ ein kleinstes $j \in \mathbb{N}_0$ mit $0 \le j \le y$ und $\rho(x_1, \ldots, x_n, j)$ gibt, so wird dieses j mit $\mu(i \le y)\rho(x_1, \ldots, x_n, i)$ bezeichnet. Anderenfalls setzen wir $\mu(i \le y)\rho(x_1, \ldots, x_n, i) = 0$. $\mu(i \le y)$ heißt *beschränkter μ-Operator.* □

Der griechische Buchstabe μ in μ-Operator steht für $\mu\iota\kappa\rho o\varsigma$ (klein). Mit Hilfe des μ-Operators werden nun Funktionen definiert. Im allgemeinen müssen dies keine berechenbaren Funktionen sein.

Definition 3.3.4 Es sei $n \in \mathbb{N}_0$.

(a) Es sei $f : \mathbb{N}_0^n \to \mathbb{N}_0$ eine partielle Funktion. f ist *durch den unbeschränkten μ-Operator definiert,* wenn ein $(n+1)$-stelliges Prädikat ρ existiert mit

$$f(x_1, \ldots, x_n) = \mu y \rho(x_1, \ldots, x_n, y)$$

für alle $x_1, \ldots, x_n \in \mathbb{N}_0$.

(b) Es sei $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ eine totale Funktion. f ist *durch den beschränkten μ-Operator definiert*, wenn ein $(n+1)$-stelliges Prädikat ρ existiert mit

$$f(x_1, \ldots, x_n, y) = \mu(i \leq y)\rho(x_1, \ldots, x_n, i)$$

für alle $x_1, \ldots, x_n, y \in \mathbb{N}_0$. □

Die beiden vorangegangenen Definitionen gelten für beliebige Prädikate. Zur Bildung von berechenbaren Funktionen sind entscheidbare Prädikate von besonderem Interesse. Diese können durch primitiv-rekursive Funktionen wie in Definition 3.3.2 gegeben sein, aber auch durch jede andere (intuitiv) berechenbare totale Funktion wie z.B. die Ackermann-Funktion. Wir wollen verdeutlichen, wie wir im Falle eines entscheidbaren Prädikats ρ beim unbeschränkten μ-Operator die Funktion f berechnen können. $f(x_1, \ldots, x_n)$ ist sicher dann definiert, wenn es zu $(x_1, \ldots, x_n)$ ein $y \in \mathbb{N}_0$ gibt mit $\rho(x_1, \ldots, x_n, y)$. Man entscheide der Reihe nach, ob $\rho(x_1, \ldots, x_n, 0)$, $\rho(x_1, \ldots, x_n, 1)$ usw. gilt, bis zum ersten Mal $\rho(x_1, \ldots, x_n, y)$ erfüllt ist. Dieses y ist nach Definition 3.3.3(a) und Definition 3.3.4(a) gleich $f(x_1, \ldots, x_n)$. Wenn es dagegen kein y gibt, für das $\rho(x_1, \ldots, x_n, y)$ gilt, so bricht das Verfahren nicht ab, und $f(x_1, \ldots, x_n)$ ist nicht definiert. f ist also im allgemeinen nur eine partielle Funktion. Dies sehen wir auch im folgenden Beispiel 3.3.1. Für den beschränkten μ-Operator dagegen wird das Verfahren spätestens mit $i = y$ abgebrochen. Falls für $i = 0, \ldots, y$ die Beziehung $\rho(x_1, \ldots, x_n, i)$ nicht gilt, wird der Wert 0 als Resultat geliefert. Hier erhalten wir für ein entscheidbares Prädikat also eine totale Funktion. Ist das Prädikat jedoch nicht entscheidbar, so scheitern die Verfahren beim ersten Auftreten eines n-Tupels $(x_1, \ldots, x_n, y)$, für das $\rho(x_1, \ldots, x_n, y)$ nicht bestimmt werden kann, obwohl doch $\rho(x_1, \ldots, x_n, y)$ entweder wahr oder falsch sein muß. In diesem Fall ist die Funktion f nicht berechenbar.

Beispiel 3.3.1 Wir definieren das Prädikat ρ durch

$$\rho(x, y) \iff x = 3y.$$

Damit kann eine partielle Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ durch

$$f(x) = \mu y \rho(x, y) = \begin{cases} \frac{x}{3}, & \text{falls } 3 \mid x \\ \text{undefiniert} & \text{sonst} \end{cases}$$

angegeben werden ($3 \mid x$ bedeutet: 3 teilt x). Das Prädikat ρ ist nach Definition 3.3.2 primitiv-rekursiv, da es durch die primitiv-rekursive Funktion $g(x, y) = (3y \dot{-} x) + (x \dot{-} 3y)$ bestimmt ist. Die Funktion f ist jedoch nicht total und somit auch nicht primitiv-rekursiv. □

Der beschränkte μ-Operator führt im Gegensatz zum unbeschränkten μ-Operator immer zu primitiv-rekursiven Funktionen, falls das Prädikat ρ primitiv-rekursiv ist.

Satz 3.3.3 Es sei $n \in \mathbb{N}_0$ und ρ ein $(n+1)$-stelliges primitiv-rekursives Prädikat. Dann ist die durch

$$f(x_1, \ldots, x_n, y) = \mu(i \leq y)\rho(x_1, \ldots, x_n, i)$$

für alle $x_1, \ldots, x_n, y \in \mathbb{N}_0$ definierte Funktion $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ primitiv-rekursiv.

Beweis: Offenbar gilt $f(x_1, \ldots, x_n, 0) = 0$ für $x_1, \ldots, x_n \in \mathbb{N}_0$, und für $y \in \mathbb{N}_0$ erhalten wir

$$f(x_1, \ldots, x_n, y+1) = \begin{cases} f(x_1, \ldots, x_n, y), & \text{falls ein } z \in \mathbb{N}_0 \text{ existiert,} \\ & \quad 0 \leq z \leq y, \text{mit } \rho(x_1, \ldots, x_n, z) \\ y+1, & \text{falls dies nicht erfüllt ist,} \\ & \quad \text{jedoch } \rho(x_1, \ldots, x_n, y+1) \text{ gilt} \\ 0 & \text{sonst.} \end{cases}$$

Wir sehen, daß dies eine rekursive Definition von f ist, die wir auf das Induktions-Rekursionsschema zurückführen müssen. Dafür definieren wir als Zwischenschritt die Funktion $h : \mathbb{N}_0^{n+2} \to \mathbb{N}_0$ mit

$$h(x_1, \ldots, x_n, y, x) = \begin{cases} x, & \text{falls ein } z \in \mathbb{N}_0 \text{ existiert, } 0 \leq z \leq y, \\ & \quad \text{mit } \rho(x_1, \ldots, x_n, z) \\ y+1, & \text{falls dies nicht erfüllt ist,} \\ & \quad \text{jedoch } \rho(x_1, \ldots, x_n, y+1) \text{ gilt} \\ 0 & \text{sonst,} \end{cases}$$

in der f auf der rechten Seite entfernt wurde. Wir zeigen zunächst, daß h primitiv-rekursiv ist. Die Bedingung der oberen Alternative der Definition von h kann offenbar durch ein $(n+1)$-stelliges Prädikat $\overline{\rho}$ mit

$$\chi_{\overline{\rho}}(x_1, \ldots, x_n, y) = 1 \iff \mathrm{sign}(\sum_{z=0}^{y} \chi_\rho(x_1, \ldots, x_n, z)) = 1$$

ausgedrückt werden. Wir definieren zunächst eine primitiv-rekursive Funktion $g : \mathbb{N}_0^n \to \mathbb{N}_0$ durch

$$\begin{aligned} g(x_1, \ldots, x_n) &= \chi_\rho(U_1^{(n)}(x_1, \ldots, x_n), \ldots, U_n^{(n)}(x_1, \ldots, x_n), C_0^{(n)}(x_1, \ldots, x_n)) \\ &= \chi_\rho(x_1, \ldots, x_n, 0) \end{aligned}$$

Dann erhalten wir $\chi_{\overline{\rho}}$ induktiv-rekursiv durch

$$\begin{aligned} \chi_{\overline{\rho}}(x_1, \ldots, x_n, 0) &= g(x_1, \ldots, x_n) \\ \chi_{\overline{\rho}}(x_1, \ldots, x_n, y+1) &= h_1(x_1, \ldots, x_n, y, \chi_{\overline{\rho}}(x_1, \ldots, x_n, y)) \end{aligned}$$

mit $h_1 = \mathrm{sign} \circ (+) \circ (\chi_\rho \circ (U_1^{(n+2)}, \ldots, U_n^{(n+2)}, S \circ U_{n+1}^{(n+2)}), U_{n+2}^{(n+2)})$, also

$$\begin{aligned} & h_1(x_1, \ldots, x_n, y, \chi_{\overline{\rho}}(x_1, \ldots, x_n, y)) \\ & \qquad = \mathrm{sign}(\chi_\rho(x_1, \ldots, x_n, y+1) + \chi_{\overline{\rho}}(x_1, \ldots, x_n, y)). \end{aligned}$$

Das Prädikat $\overline{\rho}$ ist daher nach Satz 3.3.1 primitiv-rekursiv. Nach Satz 3.3.2 ist weiter

$$h_2(x_1, \ldots, x_n, y, x) = \begin{cases} y+1, & \text{falls } \rho(x_1, \ldots, x_n, y+1) \\ 0 & \text{sonst} \end{cases}$$

primitiv-rekursiv und damit auch

$$h(x_1, \ldots, x_n, y, x) = \begin{cases} x, & \text{falls } \overline{\rho}(x_1, \ldots, x_n, y) \\ h_2(x_1, \ldots, x_n, y, x) & \text{sonst.} \end{cases}$$

Wir beachten dabei, daß alle auf den rechten Seiten von h_2 und h vorkommenden Funktionen und Prädikate als $(n+2)$-stellige primitiv-rekursive Funktionen und Prädikate des Arguments $(x_1, \ldots, x_n, y, x)$ geschrieben werden können. Unter Verwendung der weiter oben gegebenen Definition von h sehen wir, daß sich f mit Hilfe dieser Funktion h durch das Induktions-Rekursionsschema

$$\begin{aligned} f(x_1, \ldots, x_n, 0) &= 0 \\ f(x_1, \ldots, x_n, y+1) &= h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{aligned}$$

ausdrücken läßt. Folglich ist f primitiv-rekursiv. □

Definition 3.3.5 Es sei $n \in \mathbb{N}_0$ und ρ ein $(n+1)$-stelliges Prädikat, das durch die Funktion $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ mit

$$\rho(x_1, \ldots, x_{n+1}) \Longleftrightarrow f(x_1, \ldots, x_{n+1}) = 0$$

definiert ist.

(a) Wenn man das so definierte Prädikat beim unbeschränkten μ-Operator anwendet, dann erhält man zu jeder Funktion $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ eine Funktion $\mu(f) : \mathbb{N}_0^n \to \mathbb{N}_0$ mit $\mu(f)(x_1, \ldots, x_n) = \mu y \rho(x_1, \ldots, x_n, y)$, die *Minimalisierung von f*.

(b) Wenn man das so definierte Prädikat beim beschränkten μ-Operator anwendet, dann erhält man zu jeder Funktion $f : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ eine Funktion $\mu(\leq)(f) : \mathbb{N}_0^{n+1} \to \mathbb{N}_0$ mit $\mu(\leq)(f)(x_1, \ldots, x_{n+1}) = \mu(i \leq x_{n+1})\rho(x_1, \ldots, x_n, i)$, die *beschränkte Minimalisierung von f*. □

Satz 3.3.3 zeigt, daß sich bei Anwendung der beschränkten Minimalisierung auf eine primitiv-rekursive Funktion f wieder eine primitiv-rekursive Funktion ergibt. Dazu betrachten wir das folgende Beispiel.

Beispiel 3.3.2 Wir wollen zeigen, daß die Funktion $g : \mathbb{N}_0^2 \to \mathbb{N}_0$ mit

$$g(x,y) = \begin{cases} \left[\frac{x}{y}\right] & \text{für } y > 0 \\ 0 & \text{für } y = 0 \end{cases}$$

primitiv-rekursiv ist. Dabei sei $\left[\frac{x}{y}\right]$ die größte Zahl $n \in \mathbb{N}_0$ mit $n \leq \frac{x}{y}$. Zunächst definieren wir eine Funktion $f : \mathbb{N}_0^3 \to \mathbb{N}_0$ durch

$$f(x,y,z) = \overline{\text{sign}}(z \cdot y \dot{-} x).$$

Da wir $f(x,y,z) = (\overline{\text{sign}} \circ (\dot{-}))((*(U_2^{(3)}, U_3^{(3)}))(x,y,z), U_1^{(3)}(x,y,z))$ schreiben können, ist f primitiv-rekursiv. Für $y > 0$ gilt

$$\begin{aligned} g(x,y) = \left[\frac{x}{y}\right] &= \max\{k \mid k \cdot y \leq x\} \\ &= \min\{k \mid k \cdot y > x\} \dot{-} 1 \\ &= \min\{k \mid \overline{\text{sign}}(k \cdot y \dot{-} x) = 0\} \dot{-} 1 \\ &= \min\{k \mid f(x,y,k) = 0\} \dot{-} 1 \\ &= \mu(\leq)(f)(x,y,x) \dot{-} 1. \end{aligned}$$

Auch für $y = 0$ ist wegen $g(x,0) = 0$ diese Gleichung erfüllt. Nach Satz 3.3.3 ist $\mu(\leq)(f)$ primitiv-rekursiv. Da sich g in der Form

$$g(x,y) \ = \dot{-} \ ((\mu(\leq)(f) \circ (U_1^{(2)}, U_2^{(2)}, U_1^{(2)}))(x,y), C_1^{(2)}(x,y))$$

notieren läßt, entsteht g aus primitiv-rekursiven Funktionen durch den Einsetzungsprozeß und ist selbst primitiv-rekursiv. □

Im allgemeinen ist eine Funktion, die durch die unbeschränkte Minimalisierung aus einer primitiv-rekursiven Funktion entsteht, nicht unbedingt primitiv-rekursiv. Das zeigte bereits Beispiel 3.3.1, wo $f = \mu(g)$ mit $g(x,y) = (3y \dot{-} x) + (x \dot{-} 3y)$ gilt und f nicht total ist. Man sagt, für ein Prädikat ρ liegt der *Normalfall* vor, wenn es zu jedem $(x_1, \ldots, x_n) \in \mathbb{N}_0^n$ ein $y \in \mathbb{N}_0$ gibt mit $\rho(x_1, \ldots, x_n, y)$. In diesem Fall führt das im Anschluß an Definition 3.3.4 beschriebene Verfahren für alle $(x_1, \ldots, x_n)$ zur Berechnung eines Wertes $f(x_1, \ldots, x_n)$, d.h., f ist eine totale Funktion. Man kann zeigen, daß sich die Ackermann-Funktion aus primitiv-rekursiven Prädikaten durch Anwendung des μ-Operators im Normalfall ergibt und somit durch unbeschränkte Minimalisierung im Normalfall entsteht. Die Ackermann-Funktion ist jedoch nach Satz 3.2.3 nicht primitiv-rekursiv. Sie ist ein Beispiel dafür, daß die Anwendung des μ-Operators im Normalfall auf ein primitiv-rekursives Prädikat nicht immer zu einer primitiv-rekursiven Funktion führen muß.

Eine umfangreichere Klasse als die der primitiv-rekursiven Funktionen erhalten wir durch die folgende Definition.

Definition 3.3.6 Es sei f eine partielle Funktion. f heißt *μ-rekursiv*, wenn f

- eine primitiv-rekursive Grundfunktion ist,
- durch Einsetzung aus μ-rekursiven Funktionen hervorgeht,
- induktiv-rekursiv durch μ-rekursive Funktionen definiert ist
- oder durch Anwendung der Minimalisierung auf eine totale μ-rekursive Funktion entsteht. □

Wir sehen sofort, daß die nicht primitiv-rekursive Funktion aus Beispiel 3.3.1 μ-rekursiv ist. Den Ausführungen, die der Definition 3.3.6 vorangehen, entnehmen wir weiter, daß auch die Ackermann-Funktion μ-rekursiv ist. Definition 3.3.6 ist eine induktive Definition, die auf den offensichtlich berechenbaren Grundfunktionen basiert. Der Einsetzungsprozeß und das Induktions-Rekursionsschema liefern aus berechenbaren Funktionen wieder solche Funktionen. Die Anwendung der Minimalisierung auf eine totale berechenbare μ-rekursive Funktion liefert nach den Darlegungen im Anschluß an Definition 3.3.4 ebenfalls eine berechenbare Funktion. Daher stellt sich die Frage, ob alle im intuitiven Sinn berechenbaren Funktionen nicht genau durch die μ-rekursiven Funktionen beschrieben werden können. Nach den Überlegungen in Kapitel 2 wissen wir, daß dies prinzipiell nicht bewiesen werden kann. Mit einigem Aufwand kann jedoch der nächste Satz bewiesen werden.

Satz 3.3.4 Eine Funktion f ist genau dann μ-rekursiv, wenn sie Turing-berechenbar ist. □

Satz 3.3.4 ist ein Argument für die Richtigkeit der Churchschen These. Zwei sehr unterschiedliche Modelle der Berechenbarkeit führen zur selben Klasse von Funktionen. Dasselbe gilt für das Modell der Registermaschinen und auch für weitere Berechenbarkeitsmodelle, die wir in diesem Buch jedoch nicht betrachten. Welche Modelle man bisher auch immer betrachtet hat, man ist nicht über die Klasse der Turing-berechenbaren Funktionen hinausgekommen.

Man kann zeigen, daß man sich bei der Definition der μ-rekursiven Funktionen auf eine einmalige Anwendung der Minimalisierung auf primitiv-rekursive Funktionen beschränken kann.

4 Sprachen, Grammatiken und erkennende Automaten

4.1 Einführung

In Beispiel 1.2.3 wurde die Funktion $f : \{0,1\}^* \to \{0,1\}$ mit

$$f(w) = \begin{cases} 1, & \text{falls } w = w'11\ ,\ w' \in \{0,1\}^* \\ 0 & \text{sonst} \end{cases}$$

betrachtet und durch einen endlichen Moore-Automaten realisiert. Sie kann als charakteristische Funktion der Menge

$$M = \{w'11 \mid w' \in \{0,1\}^*\} \subset \{0,1\}^*$$

aufgefaßt werden. In diesem Sinn wird ein Moore-Automat, der die Ausgabemenge $\{0,1\}$ hat, auch als Erkennungsgerät für Mengen von Wörtern betrachtet. Bei Vorlage von w erkennen wir durch den errechneten Wert, ob w zur Menge M gehört oder nicht. Solche Mengen werden auch Sprachen genannt.

Definition 4.1.1 Eine endliche Menge V heißt *Alphabet*. Eine Teilmenge $L \subset V^*$ heißt *(formale) Sprache über* V. Es seien L_1 und L_2 beliebige Sprachen. Dann ist ihre *Konkatenation* durch

$$L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$$

definiert. Für eine Sprache L und $n \in \mathbb{N}_0$ wird die n-fache Konkatenation von L mit sich selbst durch L^n bezeichnet, wobei $L^0 = \{\varepsilon\}$ gilt. Dann heißt

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

die *Iteration* von L. □

Beispiel 4.1.1 Es sei V = {A, Aachen, Aal, …, Zytode, z.Z., ., …, :} (Einträge im Duden). Dann ist die deutsche Sprache eine Sprache $L \subset V^*$. Eine endliche Sprache über V ist z.B. L = {ich habe Hunger, ich habe Durst, gute Nacht}, die nur aus drei „Wörtern" über V besteht. □

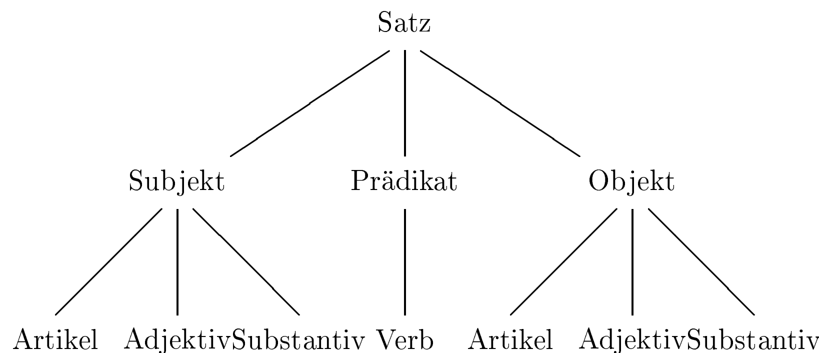
Wenn eine Sprache L endlich viele Elemente hat, dann ist sie durch Aufzählung ihrer Elemente darstellbar. Anderenfalls ist eine Formulierung nötig, die eine endliche Beschreibung der Sprache ermöglicht. Die oben gegebene Menge M konnte durch einen endlichen Moore-Automaten erkannt werden. Wir werden in den Abschnitten 4.3 und 4.5 verschiedene Typen von Automaten (Akzeptoren) einführen, die unterschiedliche Klassen von Sprachen mit Hilfe von Endzuständen erkennen. Neben Erkennungs- sind aber auch Generierungsverfahren von Bedeutung, die nach gewissen Regeln Wörter der

Sprache erzeugen. Besonders wichtig sind dabei Grammatiken, die wir in Abschnitt 4.2 ausführlich behandeln. Für verschiedene Programmiersprachen kann die Syntax durch eine Grammatik beschrieben werden. In Kapitel 8 von [23] wird dies für eine spezielle Programmiersprache durchgeführt. Zunächst geben wir einen Versuch an, die deutsche Sprache durch eine Grammatik zu erzeugen.

Beispiel 4.1.2 Ein Ausschnitt einer möglichen Grammatik für die deutsche Sprache ist

Satz $\to$ Subjekt Prädikat,
Satz $\to$ Subjekt Prädikat Objekt,
Subjekt $\to$ Substantiv,
Subjekt $\to$ Artikel Substantiv,
Subjekt $\to$ Artikel Adjektiv Substantiv,
$\vdots$
Objekt $\to$ Artikel Adjektiv Substantiv,
$\vdots$
Prädikat $\to$ Verb,
Prädikat $\to$ Hilfsverb Verb.

Diese Regeln müssen noch erweitert werden, da Artikel, Substantiv, Adjektiv, Verb und Hilfsverb durch konkrete derartige Wörter ersetzt werden müssen, z.B. durch eine Regel „Verb $\to$ bauen“. Eine Ableitung nach dieser Grammatik kann durch einen *Strukturbaum* dargestellt werden, etwa



Durch Einsetzen geeigneter Wörter, d.h. durch zusätzliche Verwendung von Regeln, die die Einträge des Duden liefern, erhält man etwa den Satz „Die fleißigen Maurer bauen ein schönes Haus“. Das Problem ist dabei jedoch, daß auch unsinnige Sätze gebildet werden können. Eine natürliche Sprache kann also durch eine formale Grammatik nicht befriedigend dargestellt werden. □

Wir stellen nun verschiedene Möglichkeiten zur Festlegung von Sprachen $L \subset V^*$ zusammen:

(a) Auflistung der Elemente: Dies ist nur für endliche Sprachen möglich.
(b) Erzeugungs- oder Generierungsverfahren: Es werden Regeln angegeben, nach denen gewisse Elemente aus V^* erzeugt werden können. Es wird festgelegt, daß L genau die Menge der Wörter ist, die durch diese Regeln erzeugt werden.
(c) Erkennungs- oder Rekognitionsverfahren: Es wird ein Algorithmus angegeben, der bei Anwendung auf ein $w \in V^*$ feststellt, ob $w \in L$ gilt oder nicht. Es kann

sich aber auch um ein Verfahren handeln, das bei Anwendung auf $w \in L$ nach endlich vielen Schritten abbricht mit der Aussage, daß $w \in L$ ist, jedoch für $w \notin L$ nicht abbricht.

(d) Konstruktionsverfahren: Es wird ein Ausdruck angegeben, der L darstellt.

(e) Lösung einer Gleichung: Es wird eine Gleichung über V^* angegeben. Die Lösungen dieser Gleichungen bestimmen eine (oder mehrere) formale Sprachen.

Wir werden in Abschnitt 4.2 Sprachen gemäß (b), in Abschnitt 4.3 und 4.5 gemäß (c) und in Abschnitt 5.2 gemäß (e) bestimmen. Ein Konstruktionsverfahren kann darin bestehen, einen regulären Ausdruck anzugeben. Das werden wir in Abschnitt 4.4 betrachten.

4.2 Die Chomsky-Hierarchie

Wir beginnen mit dem Begriff des kombinatorischen Systems, der etwas allgemeiner als der einer Grammatik ist.

Definition 4.2.1 $KS = (V, F)$ heißt *kombinatorisches System*, wenn
(a) V ein Alphabet ist und
(b) $F \subset \{(P, Q) \mid P, Q \in V^*\}$ gilt mit $|F| < \infty$.
$(P, Q) \in F$ heißt *Ersetzungsregel* oder *Produktion*. Man schreibt auch $P \to Q$. □

Definition 4.2.2 Es sei $KS = (V, F)$ ein kombinatorisches System.
(a) $P \in V^*$ *erzeugt direkt* $Q \in V^*$ (kurz $P \Longrightarrow_{KS} Q$ oder $P \Longrightarrow Q$), wenn $P', P_1, P'', Q_1 \in V^*$ existieren mit $P = P'P_1P''$, $Q = P'Q_1P''$ und $(P_1, Q_1) \in F$.
(b) $P \in V^*$ *erzeugt* $Q \in V^*$ (kurz $P \Longrightarrow^*_{KS} Q$ oder $P \Longrightarrow^* Q$), wenn $P_0, P_1, \ldots, P_k \in V^*$ existieren mit $k \in \mathbb{N}_0$, $P_0 = P$, $P_k = Q$ und $P_i \Longrightarrow P_{i+1}$ für $0 \leq i \leq k-1$.
Die Folge $P_0, P_1, \ldots, P_k$ heißt *Ableitung von Q aus P in KS*. Man schreibt auch $P_0 \Longrightarrow P_1 \Longrightarrow \ldots \Longrightarrow P_k$, wobei k als *Länge der Ableitung* bezeichnet wird. □

Durch Auszeichnung von gewissen Teilmengen von V^* erhält man Erzeugungssysteme.

Definition 4.2.3 Es sei $KS = (V, F)$ ein kombinatorisches System und $AX \subset V^*$ (*Axiomenmenge* genannt). Dann heißt

$$L_g(KS, AX) = \{Q \mid P \Longrightarrow^* Q,\ P \in AX\}$$

die *von (KS, AX) erzeugte Sprache*. □

Im folgenden wird V unterteilt in ein Terminalalphabet V_T und ein Nichtterminalalphabet V_N. Zur Sprache sollen nur noch die Wörter aus V_T^* gehören. Dies verdeutlichen wir zunächst an einem Beispiel.

Beispiel 4.2.1 Es sei $KS = (\{a, b, X\}, \{X \to \varepsilon, X \to aXb\})$. Wir wählen $V_N = \{X\}$ und $V_T = \{a, b\}$. Dann ist $L_g(KS, \{X\}) \cap \{a, b\}^* = \{a^i b^i \mid i \in \mathbb{N}_0\} = \{\varepsilon, ab, a^2b^2, a^3b^3, \ldots\}$. □

Allgemein erhalten wir

Definition 4.2.4 $G = (V_N, V_T, X_0, F)$ heißt *(erzeugende) Grammatik*, wenn

(a) V_N und V_T Alphabete sind mit $V_N \cap V_T = \emptyset$ (V_T heißt Menge der *Endsymbole* oder *Terminalzeichen* und V_N Menge der *Nichtendsymbole* oder *Nichtterminalzeichen*),

(b) $X_0 \in V_N$ ist (X_0 heißt *Anfangssymbol*) und

(c) F eine endliche Menge von Paaren (P, Q) ist mit $Q \in V^*$, $V = V_N \cup V_T$, und $P \in V^* V_N V^*$ ($(P, Q) \in F$ heißt *Ersetzungsregel* oder *Produktion*).

Falls nichts anderes gesagt wird, vereinbaren wir $V = V_N \cup V_T$. Offenbar bestimmt eine Grammatik G das durch G induzierte kombinatorische System (V, F). Folglich gilt Definition 4.2.2 auch für Grammatiken. In G bedeuten also „erzeugt (direkt)“, „ Ableitung“, „$\Longrightarrow$“ und „$\Longrightarrow^*$“ dasselbe wie im induzierten kombinatorischen System.

Definition 4.2.5 Es sei $G = (V_N, V_T, X_0, F)$ eine Grammatik. Dann heißt

$$L(G) = \{w \mid w \in V_T^*, X_0 \Longrightarrow^* w\}$$

die *von G erzeugte Sprache.*

Beispiel 4.2.2 Im allgemeinen ist es schwierig nachzuweisen, daß eine Grammatik eine bestimmte Wortmenge *und nur diese* erzeugt. Man betrachte

$$G = (\{X_0, X_1, X_2\}, \{a, b, c\}, X_0, \{X_0 \to abc,\ X_0 \to aX_1bc,\ X_1b \to bX_1,$$

$$X_1c \to X_2bcc,\ bX_2 \to X_2b,\ aX_2 \to aaX_1,\ aX_2 \to aa\}).$$

Wir behaupten $L(G) = \{a^i b^i c^i \mid i \in \mathbb{N}\}$. Die Produktion $X_0 \to abc$ liefert das Wort $a^1 b^1 c^1 \in L(G)$. Mit $X_0 \to aX_1bc$ gelangt man zu $a^1 X_1 b^1 c^1$. Allgemein muß man für ein Wort $a^i X_1 b^i c^i, i \in \mathbb{N}$, zunächst genau i-mal die dritte Produktion, einmal die vierte Produktion und i-mal die fünfte Produktion anwenden:

$$a^i X_1 b^i c^i \Longrightarrow^i a^i b^i X_1 c^i \Longrightarrow a^i b^i X_2 b c^{i+1} \Longrightarrow^i a^i X_2 b^{i+1} c^{i+1}.$$

Dabei haben wir die Anzahl der direkten Ableitungsschritte als oberen Index an $\Longrightarrow$ herangeschrieben. Andere Ableitungen sind für $a^i X_1 b^i c^i$ nicht möglich. Dann hat man die Wahl zwischen der sechsten und siebten Produktion, also

$$a^i X_2 b^{i+1} c^{i+1} \Longrightarrow a^{i+1} X_1 b^{i+1} c^{i+1} \quad \text{oder}$$

$$a^i X_2 b^{i+1} c^{i+1} \Longrightarrow a^{i+1} b^{i+1} c^{i+1}.$$

Folglich erhalten wir $L(G)$ wie angegeben. □

Im folgenden teilen wir Grammatiken in verschiedene Klassen ein.

Definition 4.2.6 Es sei $G = (V_N, V_T, X_0, F)$ eine Grammatik. G ist *vom Typ* i, $i = 0, 1, 2, 3$, wenn die Einschränkung (i) für F erfüllt ist:
(0) Keine Einschränkung.
(1) Jede Produktion in F ist von der Form $Q_1XQ_2 \to Q_1PQ_2$ mit $Q_1, Q_2 \in V^*$, $X \in V_N$ und $P \in V^+$ mit der eventuellen Ausnahme von $X_0 \to \varepsilon$. Im Ausnahmefall darf X_0 nicht auf der rechten Seite einer Produktion von F vorkommen.
(2) Jede Produktion in F ist von der Form $X \to P$ mit $X \in V_N$ und $P \in V^*$.
(3) Jede Produktion in F ist von einer der Formen $X \to Yw$ oder $X \to w$ mit $X, Y \in V_N$ und $w \in V_T^*$.
Es sei $i \in \{0, 1, 2, 3\}$ und L eine Sprache. L ist *vom Typ* i, wenn $L = L(G)$ gilt und G vom Typ i ist. $\mathcal{L}_i = \{L \mid L = L(G),\ G \text{ vom Typ } i\}$ heißt *Familie der Sprachen vom Typ* i. □

Definition 4.2.7 Es seien G_1 und G_2 Grammatiken. G_1 und G_2 heißen *äquivalent*, wenn $L(G_1) = L(G_2)$ gilt. □

Eine Grammatik vom Typ 1 heißt *kontextsensitiv*, da das Symbol X aus der Definition nur ersetzt werden kann, wenn es im Kontext $Q_1 \ldots Q_2$ steht. Grammatiken vom Typ 2 heißen auch *kontextfrei*, da das Zeichen X unabhängig vom Kontext ersetzt wird. Eine Grammatik vom Typ 3 wird *regulär* genannt. Die erzeugten Sprachen werden entsprechend als *kontextsensitive*, *kontextfreie* bzw. *reguläre* Sprachen bezeichnet.

Definition 4.2.8 Es sei G eine Grammatik. G heißt *monoton*, wenn für alle Produktionen $(P, Q) \in F$ die Ungleichung $|P| \le |Q|$ gilt mit der eventuellen Ausnahme von (X_0, ε). Dann darf X_0 jedoch nicht auf der rechten Seite einer Produktion vorkommen. □

Offenbar ist jede Grammatik vom Typ 1 auch monoton. Umgekehrt kann man zeigen, daß jede Sprache, die von einer monotonen Grammatik erzeugt wird, vom Typ 1 ist. Die Grammatik aus dem Beispiel 4.2.2 ist vom Typ 0 und monoton. Die erzeugte Sprache ist also vom Typ 1. Wir geben nun noch Beispiele für eine Typ-3- und eine Typ-2-Grammatik an.

Beispiel 4.2.3 Gegeben sei die Typ-3-Grammatik

$$G = (\{X_0, X_1\}, \{a, b\}, X_0, \{X_0 \to X_0b,\ X_0 \to X_1b,\ X_1 \to X_1a,\ X_1 \to a\}).$$

Die von G erzeugte Sprache ist

$$L(G) = \{a^ib^k \mid i, k \in \mathbb{N}\} = \{a\}^+\{b\}^+,$$

denn es sind nur Ableitungen der Form

$$X_0 \Longrightarrow^{k-1} X_0b^{k-1} \Longrightarrow X_1b^k \Longrightarrow^{i-1} X_1a^{i-1}b^k \Longrightarrow a^ib^k$$

möglich. Für ein Alphabet V_T erhalten wir die Sprache $L(G') = V_T^*$ durch die Typ-3-Grammatik $G' = (\{X_0\}, V_T, X_0, \{X_0 \to X_0a, X_0 \to \varepsilon \mid a \in V_T\})$. □

Beispiel 4.2.4 Die Typ-2-Grammatik

$$G = (\{X_0\}, \{a, b\}, X_0, \{X_0 \to aX_0b,\ X_0 \to \varepsilon\})$$

erzeugt die Sprache

$$L(G) = \{a^n b^n \mid n \in \mathbb{N}_0\}.$$

Man kann zeigen, daß diese Sprache nicht vom Typ 3 ist. Ähnlich wird die durch die charakteristische Funktion f in Beispiel 1.2.5 gegebene Sprache $\{a^n ba^n \mid n \in \mathbb{N}\}$ durch die Typ-2-Grammatik $G = (\{X_0\}, \{a, b\}, X_0, \{X_0 \to aX_0a,\ X_0 \to aba\})$ erzeugt. Wir wissen, daß die Funktion f nicht durch einen endlichen Moore-Automaten realisierbar ist. □

Definition 4.2.9 Es sei V ein Alphabet.

(a) $L \subset V^*$ heißt *rekursiv-aufzählbar*, wenn ein Algorithmus existiert, der nur die Wörter von L, gegebenenfalls mit Wiederholungen, auflistet. Mit $\mathcal{L}(ra)$ bezeichnen wir die *Familie der rekursiv-aufzählbaren Sprachen.*

(b) $L \subset V^*$ heißt *rekursiv*, wenn ein Algorithmus existiert, der für ein beliebiges Wort w entscheidet, ob $w \in L$ oder $w \notin L$ gilt. Mit $\mathcal{L}(rek)$ bezeichnen wir die *Familie der rekursiven Sprachen.* □

Die Definition einer rekursiven Sprache stimmt mit der einer entscheidbaren Menge (siehe Definition 2.7.2) überein. Daher nennen wir sie auch *entscheidbare Sprache.*

Satz 4.2.1 (a) Jede Typ-1-Sprache ist rekursiv.

(b) Jede Typ-0-Sprache ist rekursiv-aufzählbar.

(c) Jede rekursive Sprache ist rekursiv-aufzählbar.

Beweis: (a) Es sei $L = L(G)$, wobei G eine Typ-1-Grammatik ist. $\varepsilon \in L$ gilt genau dann, wenn (X_0, ε) eine Produktion in F ist. Das kann sofort überprüft werden. Sei nun $w \in V_T^+$. Wir betrachten die endlich vielen Folgen der Form

$$(*) \quad X_0 = P_0, P_1, \ldots, P_n = w \ \text{ mit } \ n \geq 0, P_i \in (V_N \cup V_T)^*,$$

$$P_i \neq P_j \ \text{ für } \ i \neq j \ \text{ und } \ |P_i| \leq |P_{i+1}|, 0 \leq i \leq n-1.$$

Wegen der Monotonie von G erhalten wir $w \in L(G)$ genau dann, wenn für eine derartige Folge

$$X_0 = P_0 \Longrightarrow P_1 \Longrightarrow \ldots \Longrightarrow P_n = w$$

gilt. Es ist immer möglich zu entscheiden, ob eine Folge diese Eigenschaft erfüllt.

(b) Es sei G eine Typ-0-Grammatik. Für jedes $k \in \mathbb{N}_0$ gibt es nur endlich viele Ableitungen in G, die mit X_0 beginnen und die die Länge k haben. Daher ist es möglich, alle Ableitungen der Länge k mit $k = 0, 1, 2, \ldots$ der Reihe nach zu überprüfen. Stammt das letzte Wort einer solchen Ableitung aus V_T^*, dann ist es in die Liste der Wörter von $L(G)$ aufzunehmen.

(c) Es sei $L \subset V^*$ eine rekursive Sprache, die durch einen Algorithmus A bestimmt ist. Wir konstruieren einen neuen Algorithmus B, der A als Teilalgorithmus enthält. B bearbeitet die Wörter von V^* in einer festgelegten Reihenfolge (z.B. der Länge nach und bei gleicher Länge in lexikographischer Reihenfolge). Für

jedes Wort $w \in V^*$ wendet er den Algorithmus A an, der die Entscheidung liefert, ob w zu L gehört. In diesem Fall wird w in eine Liste der Wörter von L aufgenommen. □

Der Beweis von Satz 4.2.1(a) gilt nicht für Typ-0-Sprachen, da es wegen der fehlenden Monotonie nicht nur endlich viele Folgen (∗) geben muß. Unter Annahme der Gültigkeit der Churchschen These kann der Aufzählungsalgorithmus aus Definition 4.2.9(a) durch eine Turingmaschine verwirklicht werden. Dann kann man zeigen, daß $\mathcal{L}_0$ und $\mathcal{L}(ra)$ übereinstimmen.

Die Menge $\{i \mid i \in \mathbb{N}_0,\ \varphi_i \text{ ist total}\}$ aus Beispiel 2.7.1 ist nach Satz 2.7.2 nicht entscheidbar, d.h. nicht rekursiv. Folglich ist auch die Menge $\{a^i \mid i \in \mathbb{N}_0,\ \varphi_i \text{ ist total}\}$ nicht rekursiv. Man kann zeigen, daß diese und die anderen Mengen aus Beispiel 2.7.1 nicht rekursiv-aufzählbar sind. Wegen der Unentscheidbarkeit des speziellen Halteproblems aus dem Beweis von Satz 2.7.1 ist auch $\{a^i \mid i \in \mathbb{N}_0,\ \varphi_i(i) \text{ ist definiert}\}$ nicht rekursiv. Es läßt sich jedoch zeigen, daß diese Sprache rekursiv-aufzählbar ist.

Mit $\mathcal{L}(fin)$ werde die Familie der endlichen Sprachen bezeichnet. Die Sprachfamilien der *Chomsky-Hierarchie* und ihre Inklusionsbeziehungen werden durch

$$\mathcal{L}(fin) \subsetneqq \mathcal{L}_3 \subsetneqq \mathcal{L}_2 \subsetneqq \mathcal{L}_1 \subsetneqq \mathcal{L}(rek) \subsetneqq \mathcal{L}(ra) = \mathcal{L}_0 \subsetneqq \bigcup_{X \text{ Alphabet}} \mathcal{P}(X^*)$$

dargestellt. Es gilt $\mathcal{L}(fin) \subset \mathcal{L}_3$, da jede endliche Sprache $L = \{w_1, \ldots, w_n\} \subset V_T^*$ durch eine Typ-3-Grammatik $G = (\{X\}, V_T, X, \{X \to w_1, \ldots, X \to w_n\})$ erzeugt werden kann. Wegen Definition 4.2.6, Satz 4.2.1 und $\mathcal{L}_0 = \mathcal{L}(ra)$ folgt sofort, daß alle einfachen Inklusionen bis auf $\mathcal{L}_2 \subset \mathcal{L}_1$ erfüllt sind. Für $\mathcal{L}_2 \subset \mathcal{L}_1$ verweisen wir auf das Buch von Salomaa [19]. Man kann zeigen, daß alle Inklusionen echt sind. Die echte Inklusion der Familie der endlichen Sprachen in $\mathcal{L}_3$ wird durch Beispiel 4.2.3 gegeben. In Abschnitt 4.3 werden wir sehen, daß $\{a^n b a^n \mid n \in \mathbb{N}\} \in \mathcal{L}_2 - \mathcal{L}_3$ gilt. Ohne Beweis notieren wir $\{a^n b^n c^n \mid n \in \mathbb{N}_0\} \in \mathcal{L}_1 - \mathcal{L}_2$. Oben wurde bereits $\{a^i \mid i \in \mathbb{N}_0,\ \varphi_i(i) \text{ ist definiert}\} \in \mathcal{L}_0 - \mathcal{L}(rek)$ erwähnt.

4.3 Endliche erkennende Automaten und reguläre Sprachen

Wir wollen jetzt bei Mealy- oder Moore-Automaten auf die Ausgabe verzichten, dafür aber Endzustände einführen. Solche Automaten können Sprachen erkennen.

Definition 4.3.1 $E = (Z, X, \delta, z_0, F)$ heißt *endlicher erkennender Automat*, wenn

(a) Z und X endliche nichtleere Mengen sind,
(b) $\delta : Z \times X \to Z$ eine Abbildung (*Zustandsüberführungsfunktion*) ist,
(c) $z_0 \in Z$ (*Anfangszustand*) und
(d) $F \subset Z$ (*Menge der Endzustände*) gilt. □

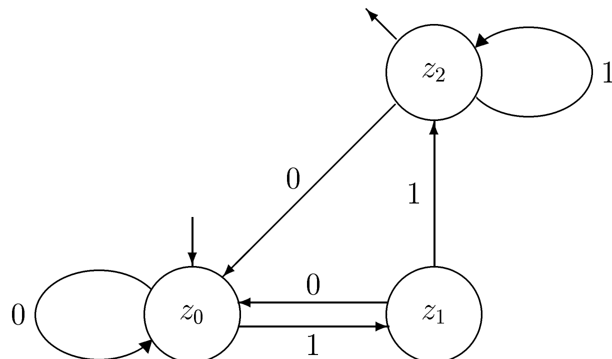
Definition 4.3.2 Es sei E ein endlicher erkennender Automat. Dann heißt

$$L(E) = \{w \in X^* \mid \delta^*(z_0, w) \in F\}$$

die *von E erkannte* oder *akzeptierte Sprache.* □

Definition 4.3.3 Es seien E_1 und E_2 endliche erkennende Automaten. E_1 und E_2 heißen *äquivalent*, wenn $L(E_1) = L(E_2)$ gilt. □

Beispiel 4.3.1 Ein endlicher erkennender Automat kann durch einen *Zustandsüberführungsgraphen* dargestellt werden. Wir betrachten den Automaten



Dabei wird der Anfangszustand z_0 mit einem unmarkierten eingehenden Pfeil und der Endzustand z_2 durch einen ausgehenden unmarkierten Pfeil gekennzeichnet. Die von diesem endlichen erkennenden Automaten erkannte Sprache ist $L(E) = \{w11 \mid w \in \{0,1\}^*\}$. □

Wir stellen fest, daß in diesem Beispiel die Sprache erkannt wird, deren charakteristische Funktion durch den Moore-Automaten aus Beispiel 1.2.3 realisiert wird. Ein Vergleich des Zustandsüberführungsgraphen aus Beispiel 4.3.1 mit dem Zustand-Ausgabe-Graphen aus Beispiel 1.2.3 zeigt, daß beide gleich sind, falls ein Endzustand des Automaten aus Beispiel 4.3.1 mit einem Zustand des Moore-Automaten mit der Ausgabe 1 identifiziert wird. Allgemein bedeutet dies, daß Moore-Automaten mit der Ausgabemenge $\{0,1\}$ sowie festem Anfangszustand äquivalent zu endlichen erkennenden Automaten sind. Dies wird genauer durch den folgenden Satz gegeben.

Satz 4.3.1 Es seien Z und X Alphabete. Eine bijektive Zuordnung zwischen endlichen erkennenden Automaten $E = (Z, X, \delta, z_0, F)$ und Moore-Automaten $M = (Z, X, \{0,1\}, \delta, \beta)$ mit ausgezeichnetem Zustand $z_0 \in Z$ sei durch

$$\beta(z) = 1 \iff z \in F \text{ für alle } z \in Z$$

gegeben. Dann ist $M_{z_0} : X^* \to \{0,1\}$ die charakteristische Funktion von $L(E)$.

Beweis: Es gilt

$$\begin{aligned} \{w \in X^* \mid M_{z_0}(w) = 1\} &= \{w \in X^* \mid \beta(\delta^*(z_0, w)) = 1\} \\ &= \{w \in X^* \mid \delta^*(z_0, w) \in F\} = L(E). \quad \square \end{aligned}$$

Aufgrund der durch diesen Satz gegebenen Äquivalenz können die Ergebnisse aus Kapitel 1 auf endliche erkennende Automaten übertragen werden. Speziell liefert der Beweis von Satz 1.2.3, daß es für eine feste endliche Menge X mit $|X| \geq 2$ überabzählbar viele charakteristische Funktionen $f : X^* \to \{0,1\}$ gibt, die keine Realisierung

(M, z_0) besitzen. Nach Satz 4.3.1 ist dies äquivalent dazu, daß es überabzählbar viele Sprachen $L \subset X^*$ gibt, die nicht durch einen endlichen erkennenden Automaten akzeptiert werden. Insbesondere zeigt Beispiel 1.2.5, daß die Sprache

$$\{a^n b a^n \mid n \in \mathbb{N}\}$$

nicht durch einen endlichen erkennenden Automaten akzeptiert wird. Zusammen mit Beispiel 4.2.4 und dem weiter unten angeführten Satz 4.3.5 ergibt sich daraus, daß die Inklusion $\mathcal{L}_3 \subset \mathcal{L}_2$ echt ist.

Auch die Reduktion von endlichen erkennenden Automaten kann auf die Reduktion von Moore-Automaten zurückgeführt werden. Es sei E ein endlicher erkennender Automat und M ein Moore-Automat, die gemäß Satz 4.3.1 einander zugeordnet sind. Nach Definition 1.3.2 gilt für Zustände $z_1, z_2 \in Z$

$$z_1 \sim z_2 \iff M_{z_1} = M_{z_2}.$$

Bezeichnen wir nun für ein beliebiges $z \in Z$ mit E_z den Automaten, der wie E, jedoch mit Anfangszustand z, definiert ist, so gilt (siehe Beweis von Satz 4.3.1)

$$M_z(w) = 1 \iff w \in L(E_z)$$

und damit

$$M_{z_1} = M_{z_2} \iff L(E_{z_1}) = L(E_{z_2}).$$

Die Äquivalenz zweier Zustände eines endlichen erkennenden Automaten kann also durch

$$z_1 \sim z_2 \iff L(E_{z_1}) = L(E_{z_2})$$

definiert werden. Bevor wir reduzierte endliche erkennende Automaten definieren können, müssen wir noch vereinfachte endliche erkennende Automaten betrachten.

Definition 4.3.4 Es sei $E = (Z, X, \delta, z_0, F)$ ein endlicher erkennender Automat. Ein Zustand $z \in Z$ heißt *erreichbar*, wenn ein Wort $w \in X^*$ mit $\delta^*(z_0, w) = z$ existiert. Der Automat E heißt *vereinfacht*, wenn alle seine Zustände erreichbar sind. □

Satz 4.3.2 Es sei $E = (Z, X, \delta, z_0, F)$ ein endlicher erkennender Automat. Dann existiert ein vereinfachter endlicher erkennender Automat E' mit $L(E) = L(E')$, der in endlich vielen Schritten konstruiert werden kann.

Beweis: Es sei n die Anzahl der Zustände aus Z. Wir zeigen zunächst, daß es für jeden erreichbaren Zustand $z \in Z$ ein $w \in X^*$, $|w| < n$, gibt mit $\delta^*(z_0, w) = z$. Es sei nämlich

$$\delta^*(z_0, w') = z, \;\; w' = a_1 \ldots a_m, \;\; m \geq n.$$

Dann wird bei der Abarbeitung von w' die Zustandsfolge

$$z_0, z_1 = \delta(z_0, a_1), z_2 = \delta(z_1, a_2), \ldots, z_m = z = \delta(z_{m-1}, a_m)$$

durchlaufen. Wegen $m \geq n$ tritt wenigstens ein Zustand in dieser Folge mindestens zweimal auf, etwa $z_i = z_j$, $0 \leq i < j \leq m$. Folglich wird bei Abarbeitung des Wortes $w'' = a_1 \ldots a_i a_{j+1} \ldots a_m$ die Zustandsfolge

$$z_0, \ldots, z_i = z_j, \ldots, z_m = z$$

durchlaufen. Dabei gilt $\delta^*(z_0, w'') = z$ und $|w''| < |w'|$. Dieser Prozeß wird (endlich oft) wiederholt, bis sich ein Wort $w \in X^*$ der gewünschten Gestalt ergibt.

Die Menge $\{z \mid z = \delta^*(z_0, w), w \in X^*, |w| < n\}$ besteht also aus genau allen erreichbaren Zuständen und ist in endlich vielen Schritten konstruierbar. Da die nicht erreichbaren Zustände überflüssig sind, liefert ihre Entfernung aus Z und F sowie eine entsprechende Änderung der Überführungsfunktion δ einen vereinfachten endlichen erkennenden Automaten mit den oben angegebenen Eigenschaften. □

Der zu einem endlichen erkennenden Automaten E' reduzierte endliche erkennende Automat $\hat{E}$ ergibt sich, indem zunächst der vereinfachte Automat $E = (Z, X, \delta, z_0, F)$ von E' konstruiert wird. Man betrachte dann den Moore-Automaten M, der nach Satz 4.3.1 E bijektiv zugeordnet ist. Weiter sei nach Definition 1.3.3 $\hat{M} = (\hat{Z}, X, \{0, 1\}, \hat{\delta}, \hat{\beta})$ der reduzierte Automat von M. Der entsprechende endliche erkennende Automat

$$\hat{E} = (\hat{Z}, X, \hat{\delta}, [z_0], \hat{F}) \text{ mit}$$

$$\hat{Z} = Z/\sim, \ \hat{F} = \{[z] \mid z \in F\} \ \text{ und } \ \hat{\delta} : \hat{Z} \times X \to \hat{Z} \ \text{ mit } \ \hat{\delta}([z], x) = [\delta(z, x)]$$

ist der reduzierte Automat von E'. Dabei ergibt sich die Definition von $\hat{F}$ unter Benutzung von Satz 4.3.1 wegen

$$[z] \in \hat{F} \iff \hat{\beta}([z]) = \beta(z) = 1 \iff z \in F.$$

Nach Satz 1.3.2 gilt speziell $M_{z_0} = \hat{M}_{[z_0]}$ und damit nach Satz 4.3.1

$$L(E') = L(E) = L(\hat{E}).$$

Auch der Algorithmus zur Konstruktion eines reduzierten endlichen erkennenden Automaten kann aus Kapitel 1 entsprechend übernommen werden. Man kann zeigen, daß der reduzierte Automat eines gegebenen endlichen erkennenden Automaten E' unter allen Automaten E mit $L(E) = L(E')$ eine minimale Anzahl von Zuständen besitzt. Damit diese Minimalitätseigenschaft gilt, ist die oben angegebene Vereinfachung erforderlich.

Endliche erkennende Automaten sind *deterministisch*. Zu jedem Zustand z und jeder Eingabe x gibt es genau einen Folgezustand $\delta(z, x)$. Wir wollen die Definition der Automaten so erweitern, daß ein solches Paar (z, x) keinen oder auch mehrere Folgezustände besitzen kann.

Definition 4.3.5 $E = (Z, X, \delta, z_0, F)$ heißt *nichtdeterministischer endlicher erkennender Automat*, wenn E wie in Definition 4.3.1 definiert ist, wobei jedoch die Abbildung δ durch $\delta : Z \times X \to \mathcal{P}(Z)$ ersetzt wird. Die *X^*-Erweiterung von δ* ist durch die Abbildung $\delta^* : Z \times X^* \to \mathcal{P}(Z)$ mit

$$\delta^*(z, \varepsilon) = \{z\}, \quad \delta^*(z, wx) = \bigcup_{z_1 \in \delta^*(z, w)} \delta(z_1, x)$$

für alle $z \in Z$, $x \in X$ und $w \in X^*$ gegeben. Dann ist die von einem nichtdeterministischen endlichen erkennenden Automaten E erkannte Sprache $L(E) = \{w \in X^* \mid \delta^*(z_0, w) \cap F \neq \emptyset\}$. □

Auch für Moore- und Mealy-Automaten können entsprechende nichtdeterministische Versionen definiert werden. Aufgrund der Definition möchte man annehmen, daß nichtdeterministische endliche erkennende Automaten leistungsfähiger sind als deterministische, also eine umfangreichere Familie von Sprachen akzeptieren. Wir zeigen, daß diese Annahme falsch ist.

Satz 4.3.3 Es sei L eine Sprache. L wird genau dann von einem endlichen erkennenden Automaten erkannt, wenn L von einem nichtdeterministischen endlichen erkennenden Automaten erkannt wird.

Beweis: Es sei L erkennbar von einem endlichen erkennenden Automaten

$$E_1 = (Z, X, \delta, z_0, F).$$

Dieser Automat ist auffaßbar als ein nichtdeterministischer endlicher erkennender Automat $E_1' = (Z, X, \delta', z_0, F)$, wobei $\delta'(z, x) = \{\delta(z, x)\}$ gilt. Es ist dann $\delta'^*(z, w) = \{\delta^*(z, w)\}$ für alle $z \in Z$ und $w \in X^*$, was wir durch Induktion über die Länge von w beweisen. Für $w = \varepsilon$ gilt $\delta'^*(z, \varepsilon) = \{z\} = \{\delta^*(z, \varepsilon)\}$. Es sei weiter die Aussage für w bewiesen. Dann folgt

$$\begin{aligned} \delta'^*(z, wx) &= \bigcup_{z_1 \in \delta'^*(z,w)} \delta'(z_1, x) = \bigcup_{z_1 \in \{\delta^*(z,w)\}} \{\delta(z_1, x)\} \\ &= \{\delta(\delta^*(z, w), x)\} = \{\delta^*(z, wx)\}, \end{aligned}$$

da $z_1 = \delta^*(z, w)$ gilt. Damit erhalten wir weiter

$$\begin{aligned} L(E_1) &= \{w \in X^* \mid \delta^*(z_0, w) \in F\} \\ &= \{w \in X^* \mid \delta'^*(z_0, w) \cap F \neq \emptyset\} = L(E_1'). \end{aligned}$$

Umgekehrt sei L erkennbar von einem nichtdeterministischen Automaten $E = (Z, X, \delta, z_0, F)$. Dann ist $E' = (\mathcal{P}(Z), X, \delta', \{z_0\}, F')$ mit

$$F' = \{Z' \mid Z' \subset Z, Z' \cap F \neq \emptyset\}, \quad \delta'(Z', x) = \bigcup_{z \in Z'} \delta(z, x)$$

ein deterministischer Automat. Wir zeigen zunächst für alle $Z' \subset Z$ und $w \in X^*$ die Gültigkeit von

$$\delta'^*(Z', w) = \bigcup_{z \in Z'} \delta^*(z, w),$$

also die Erweiterung der Definition von δ' auf Wörter. Dies geschieht durch Induktion über die Länge von w. Für $w = \varepsilon$ gilt

$$\delta'^*(Z', \varepsilon) = Z' = \bigcup_{z \in Z'} \delta^*(z, \varepsilon).$$

Für $w = w_1 x$ erhalten wir

$$\begin{aligned} \delta'^*(Z', w_1 x) &= \delta'(\delta'^*(Z', w_1), x) = \delta'(\bigcup_{z \in Z'} \delta^*(z, w_1), x) \\ &= \bigcup_{z_1 \in \bigcup_{z \in Z'} \delta^*(z, w_1)} \delta(z_1, x) \\ &= \bigcup_{z \in Z'} (\bigcup_{z_1 \in \delta^*(z, w_1)} \delta(z_1, x)) = \bigcup_{z \in Z'} \delta^*(z, w_1 x). \end{aligned}$$

Wir können jetzt die folgende Äquivalenz beweisen:

$$\begin{aligned} w \in L(E) &\iff \delta^*(z_0, w) \cap F \neq \emptyset \\ &\iff \delta^*(z_0, w) \in F' \\ &\iff \delta'^*(\{z_0\}, w) = \bigcup_{z \in \{z_0\}} \delta^*(z, w) = \delta^*(z_0, w) \in F' \\ &\iff w \in L(E'). \quad \square \end{aligned}$$

Um zu zeigen, daß $\mathcal{L}_3$ mit der Familie von Sprachen übereinstimmt, die von endlichen Automaten erkannt werden, benötigen wir als Hilfe den folgenden Satz. Seine Aussage ist auch von eigenem Interesse, da er eine gewisse „Normalform“ für Typ-3-Grammatiken liefert.

Satz 4.3.4 Es sei $G = (V_N, V_T, X_0, F)$ eine Typ-3-Grammatik. Dann existiert eine äquivalente Typ-3-Grammatik $G' = (V_N', V_T, X_0, F')$, die nur Produktionen der Form

$$(*) \qquad X \to Ya \ \text{ oder } \ X \to \varepsilon$$

mit $X, Y \in V_N'$ und $a \in V_T$ besitzt.

Beweis: Die Grammatik G kann außer eventuellen Produktionen der Form $(*)$ auch Produktionen der Form

$$X \to Ya_1 \dots a_k, \ X \to b_1 \dots b_l \ \text{ und } \ X \to Y$$

mit $X, Y \in V_N$, $a_i, b_j \in V_T$, $i = 1, \dots, k$, $j = 1, \dots, l$, $k \geq 2$ und $l \geq 1$ haben. Als Zwischenschritt konstruieren wir eine Grammatik $G'' = (V_N'', V_T, X_0, F'')$, die außer Produktionen des Typs $(*)$ noch Produktionen des Typs $X \to Y$ enthalten kann. Jede Produktion $X \to Ya_1 \dots a_k$ aus F mit $k \geq 2$ wird entfernt und durch Produktionen

$$X \to Y_k a_k, \ Y_k \to Y_{k-1} a_{k-1}, \dots, Y_2 \to Ya_1$$

aus F'' ersetzt, wobei $Y_k, \dots, Y_2$ neue Zeichen aus V_N'' sind. Alle neuen Nichtterminalzeichen sind paarweise verschieden und unterscheiden sich von allen Zeichen aus V_N und allen zuvor eingeführten Zeichen. Diese Folge von Produktionen aus G'' simuliert genau die ersetzte Produktion aus G. Ähnlich wird jede Produktion $X \to b_1 \dots b_l$, $l \geq 1$, ersetzt durch

$$X \to Y_l' b_l, \ Y_l' \to Y_{l-1}' b_{l-1}, \dots, \ Y_2' \to Y_1' b_1, \ Y_1' \to \varepsilon$$

mit neuen Zeichen $Y_l', \dots, Y_1'$ aus V_N''. Offensichtlich gilt $L(G) = L(G'')$, und G'' hat die oben angegebenen Eigenschaften. Zur Konstruktion von $G' = (V_N'', V_T, X_0, F')$ definieren wir für jedes $Y \in V_N''$

$$U(Y) = \{Y' \mid Y' \in V_N'', Y' \Longrightarrow^* Y\}.$$

Da die Wörter, die in einer Ableitung $Y' \Longrightarrow^* Y$ vorkommen, die Länge 1 haben und Nichtterminalzeichen sind, ist $U(Y)$ endlich und effektiv zu bestimmen. Für jede Produktion $Y \to Xa$ aus F'' werden zusätzlich die Produktionen

$$Y' \to Xa, \ Y' \in U(Y),$$

in F' aufgenommen, für $Y \to \varepsilon$ noch zusätzlich

$$Y' \to \varepsilon,\ Y' \in U(Y).$$

Damit können keine neuen Terminalwörter erzeugt werden, jedoch sind jetzt die Produktionen $X \to Y$ überflüssig und werden entfernt. Es gilt $L(G) = L(G')$, und alle Produktionen von G' sind von der Form $(*)$. □

Satz 4.3.5 Es sei L eine Sprache. L ist genau dann von einem endlichen erkennenden Automaten erkennbar, wenn L vom Typ 3 ist.

Beweis: Es gelte $L = L(E)$ mit einem endlichen erkennenden Automaten $E = (Z, X, \delta, z_0, Z')$. Zu E und jedem Endzustand $z \in Z'$ konstruieren wir den endlichen erkennenden Automaten $E_z = (Z, X, \delta, z_0, \{z\})$, der sich von E nur durch die Menge der Endzustände unterscheidet. Dann definieren wir eine Typ-3-Grammatik $G_z = (Z, X, z, F)$ mit

$$F = \{z_2 \to z_1 x \mid z_1, z_2 \in Z, x \in X, \delta(z_1, x) = z_2\} \cup \{z_0 \to \varepsilon\}.$$

Wir betrachten ein Wort $w \in X^*$. Im Fall $w = \varepsilon$ erhalten wir aufgrund der Definitionen von E_z und G_z die Äquivalenz

$$\varepsilon \in L(E_z) \iff z = z_0 \iff \varepsilon \in L(G_z).$$

Anderenfalls sei $w = a_1 \ldots a_n$ mit $a_i \in X,\ i = 1, \ldots, n,\ n \in \mathbb{N}$. Es gilt $w \in L(E_z)$ genau dann, wenn Zustände $z_0, z_1, \ldots, z_n = z \in Z$ existieren mit

$$\delta(z_0, a_1) = z_1, \delta(z_1, a_2) = z_2, \ldots, \delta(z_{n-1}, a_n) = z_n = z.$$

Nach Definition von G_z ist dies gleichwertig zur Existenz von Produktionen

$$z_1 \to z_0 a_1,\ z_2 \to z_1 a_2,\ \ldots,\ z = z_n \to z_{n-1} a_n$$

in G_z. Wegen der Produktion $z_0 \to \varepsilon$ erhalten wir äquivalent eine Ableitung

$$z \Longrightarrow z_{n-1} a_n \Longrightarrow \ldots \Longrightarrow z_1 a_2 \ldots a_n \Longrightarrow z_0 a_1 \ldots a_n \Longrightarrow w$$

in G_z, die $w \in L(G_z)$ erzeugt. Es gilt also $L(E_z) = L(G_z)$ und damit

$$L(E) = \bigcup_{z \in Z'} L(E_z) = \bigcup_{z \in Z'} L(G_z).$$

Sind nun

$$G_{z_1} = (Z, X, z_1, F) \text{ und } G_{z_2} = (Z, X, z_2, F)$$

zwei solche Typ-3-Grammatiken, so ist mit einem neuen Symbol Y_0

$$(Z \cup \{Y_0\}, X, Y_0, F \cup \{Y_0 \to z_1, Y_0 \to z_2\})$$

eine Typ-3-Grammatik, die offenbar $L(G_{z_1}) \cup L(G_{z_2})$ erzeugt. Damit folgt, daß $L(E)$ vom Typ 3 ist.

Es sei umgekehrt eine Typ-3-Grammatik $G = (V_N, V_T, X_0, F)$ mit $L = L(G)$ gegeben. Nach Satz 4.3.4 kann ohne Beschränkung der Allgemeinheit angenommen werden, daß G höchstens Produktionen der Form

$$X \to Ya \quad \text{oder} \quad X \to \varepsilon$$

mit $X, Y \in V_N$ und $a \in V_T$ besitzt. Wir definieren einen deterministischen endlichen erkennenden Automaten $E = (\mathcal{P}(V_N), V_T, \delta, Z_0, V_F)$ mit

$$Z_0 = \{Y \mid Y \in V_N, (Y, \varepsilon) \in F\}, \; V_F = \{Z \mid Z \subset V_N, X_0 \in Z\} \;\text{ sowie}$$

$$\delta(Z, a) = \{Y \mid (Y, Xa) \in F, X \in Z\} \text{ für } Z \in \mathcal{P}(V_N) \text{ und } a \in V_T.$$

Für $w = \varepsilon$ gilt die Äquivalenz

$$\varepsilon \in L(G) \iff (X_0, \varepsilon) \in F \iff X_0 \in Z_0 \iff Z_0 \in V_F \iff \varepsilon \in L(E).$$

Für $w = a_1 \ldots a_n$, $a_i \in V_T$, $i = 1, \ldots, n$, $n \in I\!N$, folgt aus $w \in L(G)$ die Existenz von Produktionen

$$X_0 \to Y_n a_n, Y_n \to Y_{n-1} a_{n-1}, \ldots, Y_2 \to Y_1 a_1, Y_1 \to \varepsilon$$

in G mit geeigneten Zeichen $Y_1, \ldots, Y_n \in V_N$. Im Fall $n = 1$ wird $X_0 = Y_2$ gesetzt. Wegen $(Y_1, \varepsilon) \in F$ und der Definition von E gilt $Y_1 \in Z_0$. Mit dem Wort w und dem Zustand Z_0 des Automaten E betrachten wir dann die Folge von Zustandsübergängen

$$\delta(Z_0, a_1) = Z_1, \ldots, \delta(Z_{n-1}, a_n) = Z_n.$$

Wegen $Y_1 \in Z_0$, $(Y_2, Y_1 a_1) \in F$ und der Definition von δ erhalten wir $Y_2 \in Z_1$. Falls $n \geq 2$ ist, ergibt sich wegen $(Y_3, Y_2 a_2) \in F$ weiter $Y_3 \in Z_2$ usw., bis schließlich $X_0 \in Z_n$ folgt wegen $Y_n \in Z_{n-1}$, $(X_0, Y_n a_n) \in F$. Die Definition von V_F liefert $Z_n \in V_F$ und damit $w \in L(E)$.

Für $w \in L(E)$ gibt es umgekehrt eine Folge von Zustandsüberführungen

$$\delta(Z_0, a_1) = Z_1, \ldots, \delta(Z_{n-1}, a_n) = Z_n \quad \text{mit} \quad Z_n \in V_F.$$

Dann ist $X_0 \in Z_n$. Nach Definition von δ existiert $Y_n \in Z_{n-1}$ mit $X_0 \to Y_n a_n$ gemäß G. Falls $n \geq 2$ ist, gibt es zu Y_n ein Symbol $Y_{n-1} \in Z_{n-2}$ mit einer Produktion $Y_n \to Y_{n-1} a_{n-1}$ usw. In jedem Fall erhalten wir am Ende $Y_1 \in Z_0$ mit einer Produktion $Y_2 \to Y_1 a_1$. Aus $Y_1 \in Z_0$ folgt, daß auch $Y_1 \to \varepsilon$ eine Produktion in G ist. Somit ist $w \in L(G)$. Insgesamt haben wir damit $L(E) = L(G)$ bewiesen. □

4.4 Reguläre Ausdrücke

Wie wir auf Seite 81 bemerkt haben, besteht die Möglichkeit, eine Sprache durch ein Konstruktionsverfahren zu beschreiben. Dies soll hier für reguläre Sprachen durchgeführt werden. Zunächst beweisen wir drei einfache Abschlußeigenschaften, die auch für andere Typ-i-Sprachfamilien erfüllt sind (siehe z.B. [19]).

Satz 4.4.1 Es seien L und L' Sprachen vom Typ 3. Dann folgt

$$L \cup L' \in \mathcal{L}_3,\ LL' \in \mathcal{L}_3 \text{ und } L^* \in \mathcal{L}_3.$$

Beweis: Es seien $G = (V_N, V_T, X_0, F)$ und $G' = (V_N', V_T', X_0', F')$ Grammatiken vom Typ 3 mit $L(G) = L$ und $L(G') = L'$. Ohne Beschränkung der Allgemeinheit gelte $V_N \cap V_N' = \emptyset$. Die Vereinigung $L \cup L'$ wird von der Grammatik

$$G_1 = (V_N \cup V_N' \cup \{Y_0\}, V_T \cup V_T', Y_0, F \cup F' \cup \{Y_0 \to X_0, Y_0 \to X_0'\})$$

erzeugt, wobei Y_0 ein neues Nichtterminalzeichen ist.

Für die Konkatenation betrachten wir die Produktionen

$$X \to w,\ X \in V_N',\ w \in V_T'*$$

aus F'. Bei einer Ableitung gemäß G' können sie nur im letzten Ableitungsschritt angewendet werden. Wir ersetzen diese Produktionen daher durch die Produktionen $X \to X_0 w$ und erhalten so eine neue Produktionenmenge $\bar{F}$. Dann ist

$$G_2 = (V_N \cup V_N', V_T \cup V_T', X_0', F \cup \bar{F})$$

eine Typ-3-Grammatik mit $L(G_2) = LL'$.

Schließlich betrachten wir den Fall der Iteration. Wir ersetzen jede Produktion

$$X \to w,\ X \in V_N, w \in V_T^*,$$

in F durch $X \to Y_0 w$ mit einem neuen Symbol Y_0. Dadurch erhalten wir eine Produktionenmenge $\hat{F}$ und damit die Grammatik

$$G_3 = (V_N \cup \{Y_0\}, V_T, Y_0, \{Y_0 \to \varepsilon, Y_0 \to X_0\} \cup \hat{F}),$$

für die $L(G_3) = L^*$ gilt. □

Definition 4.4.1 Es seien V und $V' = \{\cup, {}^*, \emptyset, (,)\}$ disjunkte Alphabete. Es heißt $P \in (V \cup V')^*$ *regulärer Ausdruck über* V, wenn die folgenden Aussagen erfüllt sind:

(a) $P \in V \ \vee\ P = \emptyset$ oder
(b) $P = (Q \cup R) \ \vee\ P = (QR) \ \vee\ P = Q^*$
mit regulären Ausdrücken Q und R über V. □

Definition 4.4.2 Es sei P ein regulärer Ausdruck über V. Dann *bezeichnet* P *die Sprache* $< P >$ über V, wobei $< P >$ wie folgt gegeben ist:

(a) Die durch $\emptyset$ bezeichnete Sprache ist die leere Sprache.
(b) Die durch $a \in V$ bezeichnete Sprache besteht aus dem Wort a, d.h. $< a >= \{a\}$.
(c) Für reguläre Ausdrücke P und Q über V gilt

$$< (P \cup Q) >=< P > \cup < Q >,\ < (PQ) >=< P >< Q >,\ < P^* >=< P >^* .$$

□

Man beachte, daß formal das Zeichen $\cup$ in einem regulären Ausdruck nicht das Vereinigungszeichen zweier Mengen ist. In $< (P \cup Q) >=< P > \cup < Q >$ bedeutet z.B. nur das zweite $\cup$-Zeichen die Mengenvereinigung, entsprechendes gilt für das *-Zeichen und auch die „Konkatenation". Der Zusammenhang zwischen einem regulären Ausdruck und der durch ihn bezeichneten Sprache kann so gesehen werden, daß der reguläre Ausdruck der *syntaktische* Begriff ist, dessen *Semantik* durch die bezeichnete Sprache gegeben wird.

Wir vereinbaren, daß bei regulären Ausdrücken „unnötige" Klammern weggelassen werden können. Die Stärke der Bindung ist durch die Reihenfolge „Iteration", „Konkatenation", „Vereinigung" gegeben.

Beispiel 4.4.1 Wir wählen $V = \{a\}$ und erhalten damit z.B.

$$\begin{aligned} < (aa)^* > &= < (aa)^* \cup (\emptyset a) > = < (aa \cup aaaa)^* > = < (aa)^*(aa)^* > \\ &= \{w \mid w = a^{2n}, n \in I\!N_0\}, \\ < \emptyset^* > &= < \emptyset >^* = \emptyset^* = \{\varepsilon\}. \quad \square \end{aligned}$$

Satz 4.4.2 Es sei L eine Sprache. L ist genau dann durch einen regulären Ausdruck bezeichnet, wenn $L \in \mathcal{L}_3$ gilt.

Beweis: Wir gehen zunächst davon aus, daß L durch einen regulären Ausdruck über $\{a_1, \ldots, a_r\}$ bezeichnet ist, d.h., L ergibt sich aus den endlichen Sprachen $\emptyset$, $\{a_1\}, \ldots, \{a_r\}$ durch endlich viele Anwendungen der regulären Operationen Vereinigung, Konkatenation und Iteration. Nach Satz 4.4.1 gilt dann $L \in \mathcal{L}_3$.

Umgekehrt sei L eine Typ-3-Sprache. Dann wird L nach Satz 4.3.5 von einem deterministischen endlichen erkennenden Automaten $E = (Z, X, \delta, z_1, F)$ akzeptiert. Dabei gelte $Z = \{z_1, \ldots, z_n\}$ mit einem geeigneten $n \in I\!N$, wobei z_1 der Anfangszustand ist, und weiter sei $F = \{z'_1, \ldots, z'_k\}$ mit $k \in I\!N_0$ und $k \leq n$. Dann definieren wir für $i = 1, \ldots, k$ endliche erkennende Automaten $E^{z'_i}$ dadurch, daß sie im wesentlichen wie E definiert sind, nur die Endzustandsmenge F von E wird durch $\{z'_i\}$ ersetzt. Offenbar gilt

$$L(E) = \bigcup_{i=1}^{k} L(E^{z'_i}).$$

Ist nun $k = 0$, also $F = \emptyset$, so wird $L(E)$ nach Definition 4.4.2(a) durch den regulären Ausdruck $\emptyset$ bezeichnet. Für $k > 0$ müssen wir nach Definition 4.4.2(c) zeigen, daß jede Sprache $L(E^{z'_i})$ durch einen regulären Ausdruck bezeichnet ist. Wir beweisen allgemein, daß dies für jede Sprache $L(E^{z_j})$, $j = 1, \ldots, n$, gilt.

Wir definieren

$$L_{ij}^k, \ i, j, k \in I\!N_0, \ 0 \leq k \leq n, \ 1 \leq i, j \leq n,$$

als die Sprache, die aus allen Wörtern $w = x_1 \ldots x_t$, $x_i \in X$, $i = 1, \ldots, t$, $t \in I\!N_0$, mit $\delta^*(z_i, w) = z_j$ besteht, für die

$$t = 0 \text{ oder } t = 1$$

oder aber für $t > 1$

$$\delta(z_i, x_1) = z_{i_1}, \ \delta(z_{i_1}, x_2) = z_{i_2}, \ \ldots, \ \delta(z_{i_{t-1}}, x_t) = z_j$$

mit $i_\nu \le k, \nu = 1, \ldots, t-1$, gilt. Anschaulich ausgedrückt, ist L_{ij}^k die Menge der Wörter w, die den endlichen erkennenden Automaten E vom Zustand z_i in den Zustand z_j überführen und dabei nur Zustände $z_{k'}$ mit $k' \le k$ durchlaufen. Man beachte, daß $i, j > k$ gelten darf. Wir zeigen durch Induktion über k, daß L_{ij}^k durch einen regulären Ausdruck bezeichnet werden kann.

Nach Definition von L_{ij}^k gilt $\varepsilon \in L_{ij}^k$ genau dann, wenn $i = j$ ist. Wir betrachten L_{ij}^0. Für $i \ne j$ besteht L_{ij}^0 aus den Wörtern, die in einem Schritt den Zustand z_i in z_j überführen, so daß $L_{ij}^0 \subset V_T$ gilt. Für $i = j$ gilt noch zusätzlich $\varepsilon \in L_{ij}^0$ und damit $\varepsilon \in L_{ij}^0 \subset V_T \cup \{\varepsilon\}$. Da nach Definition 4.4.2(a) und (c) die Sprache $\{\varepsilon\}$ durch den regulären Ausdruck $\emptyset^*$ und jede Teilmenge $\{a_1, \ldots, a_l\} \subset V_T$ nach Definition 4.4.2(b) und (c) durch den regulären Ausdruck $(a_1 \cup \ldots \cup a_l)$ bezeichnet ist, ist auch L_{ij}^0 für alle i, j mit $1 \le i, j \le n$ durch einen regulären Ausdruck bezeichnet.

Wir nehmen weiter an, daß für ein festes k, $0 \le k \le n-1$, jede Sprache L_{ij}^k durch einen regulären Ausdruck bezeichnet ist. Dann gilt offenbar

$$L_{ij}^{k+1} = L_{ij}^k \cup L_{i(k+1)}^k (L_{(k+1)(k+1)}^k)^* L_{(k+1)j}^k.$$

Nach Definition 4.4.2(c) ist dann auch L_{ij}^{k+1} durch einen regulären Ausdruck bezeichnet. Damit ist der Induktionsbeweis beendet.

Speziell erkennen wir, daß $L(E^{z_j}) = L_{1j}^n$ für alle $j = 1, \ldots, n$ durch einen regulären Ausdruck bezeichnet ist. □

Definition 4.4.3 Es sei $w = a_1 a_2 \ldots a_n$, $a_i \in V_T$, $i = 1, \ldots, n$, $n \in \mathbb{N}_0$. Durch $sp(w) = a_n \ldots a_2 a_1$ wird die *Spiegeloperation* $sp : V_T^* \to V_T^*$ definiert. Speziell gilt $sp(\varepsilon) = \varepsilon$. Damit wird $sp(L) = \{w' \mid w' = sp(w), w \in L\}$ für jede Sprache $L \subset V_T^*$ gegeben.

Definition 4.4.4 Es seinen V und V' Alphabete. Eine Abbildung $h : V^* \to V'^*$ mit

$$h(\varepsilon) = \varepsilon, \bigwedge_{w, w' \in V^*} h(ww') = h(w)h(w')$$

heißt *Homomorphismus* von V^* in V'^*. Für eine Sprache $L \subset V^*$ werde $h(L) = \{h(v) \mid v \in L\}$ gesetzt. □

Offensichtlich ist ein solcher Homomorphismus h eindeutig bestimmt, wenn für jedes $a \in V$ der Bildwert $h(a)$ festgelegt ist.

Satz 4.4.3 Die Familie $\mathcal{L}_3$ ist abgeschlossen unter beliebigem Homomorphismus und der *sp*-Operation.

Beweis: Es sei $L \subset V_T^*$, $L \in \mathcal{L}_3$. Wegen Satz 4.4.2 wird L durch einen regulären Ausdruck P über V_T bezeichnet. Wir beweisen zunächst den Abschluß unter der *sp*-Operation. P' sei der reguläre Ausdruck, der sich aus P durch Vertauschung der Faktoren in allen Konkatenationen ergibt. Dann ist offenbar $sp(L)$ durch P' bezeichnet, und es folgt $sp(L) \in \mathcal{L}_3$. Weiter sei h ein Homomorphismus. Somit wird jedes $a \in V_T$ in den Wörtern von L durch die einelementige Menge $\{h(a)\} \in \mathcal{L}_3$ ersetzt. $\{h(a)\}$ ist jedoch durch einen regulären Ausdruck Q_a bezeichnet. Wird jedes $a \in V_T$ in P durch Q_a ersetzt, so erhalten wir nach Definition 4.4.1 einen regulären Ausdruck P'', der die Sprache $h(L)$ bezeichnet, d.h. $h(L) \in \mathcal{L}_3$. □

4.5 Weitere Automatentypen und ihre zugehörigen Sprachen

Es stellt sich die Frage, ob auch andere Sprachfamilien der Chomsky-Hierarchie durch geeignete Typen von Automaten charakterisiert werden können. Im folgenden wollen wir für $\mathcal{L}_0$, $\mathcal{L}_1$ und $\mathcal{L}_2$ entsprechende Akzeptoren definieren. Ohne Beweis geben wir dabei die Äquivalenz zwischen den Sprachfamilien und den Akzeptoren an. Im Buch von *Hopcroft* und *Ullman* [11] können diese Beweise nachgelesen werden. Wir betrachten zunächst die in Kapitel 2 eingeführten Turingmaschinen und versehen sie zusätzlich mit einer Menge von Endzuständen. Gleichzeitig lassen wir für sie Nichtdeterminismus zu.

Definition 4.5.1 $T = (Z, X, \delta, z_0, F)$ heißt *(akzeptierende) nichtdeterministische Turingmaschine*, wenn

(a) Z, X und z_0 wie in Definition 2.1.1 definiert sind,

(b) $\delta : Z \times \bar{X} \to \mathcal{P}'(Z \times (\bar{X} \cup \{l, r, s\}))$ eine Abbildung ist (*lokale Überführungsfunktion*), wobei $\mathcal{P}'(Z \times (\bar{X} \cup \{l, r, s\}))$ die Menge der nichtleeren Teilmengen von $Z \times (\bar{X} \cup \{l, r, s\})$ ist, und

(c) $F \subset Z$ gilt (*Endzustandsmenge*).

T heißt *(akzeptierende) deterministische Turingmaschine*, wenn $|\delta(z, x)| = 1$ für alle $(z, x) \in Z \times \bar{X}$ gilt. □

Das hier eingeführte Modell einer nichtdeterministischen Turingmaschine hat für jeden Zustand z und jedes Bandsymbol x mindestens eine Instruktion zur Verfügung. Diese Art der Definition ist zur Vereinfachung einiger Beweise in der Vorlesung „Theoretische Informatik II“ nützlich. Man kann aber auch hier für die Abbildung δ die leere Menge als Bild zulassen. Dies bedeutet jedoch keine Änderung der prinzipiellen Fähigkeiten, da $\delta(z, x) = \emptyset$ immer durch $\delta(z, x) = \{(z, s)\}$ simuliert werden kann.

Entsprechend Definition 2.1.2 können nichtdeterministische Turingmaschinen durch Turingtafeln äquivalent gekennzeichnet werden. Die Turingtafel für eine nichtdeterministische Turingmaschine kann jedoch für ein Paar $(z, x) \in Z \times \bar{X}$ mehrere Zeilen enthalten, die mit $z\ x$ beginnen. Bandfunktion, Bandinschrift, Konfiguration, Anfangs-, End- und Folgekonfiguration können wie in Definition 2.1.3, 2.1.4 bzw. 2.1.5 angegeben werden. Man beachte, daß aufgrund des Nichtdeterminismus Folgekonfigurationen nicht mehr eindeutig bestimmt sein müssen.

Definition 4.5.2 Es sei $T = (Z, X, \delta, z_0, F)$ eine nichtdeterministische Turingmaschine und $w \in X^*$. Eine Rechnung von T an w heißt *haltend*, wenn folgendes gilt:

(a) T befindet sich zu Beginn im Zustand z_0, und für $w \neq \varepsilon$ ist der Bandinhalt $\ldots bwb \ldots$, wobei der Kopf über dem ersten Symbol von w steht. Für $w = \varepsilon$ ist der Bandinhalt $\ldots \underline{b} \ldots$.

(b) T befindet sich nach endlich vielen (etwa $k - 1, k \geq 1$) Schritten in einer Endkonfiguration $(n, \Gamma(\beta), z)$. k wird als *Länge der Rechnung* bezeichnet.

Diese Rechnung heißt *akzeptierend*, wenn für den Zustand z der Endkonfiguration $z\ \beta(n)\ z'\ s$ mit $z' \in F$ eine Zeile der Turingtafel von T ist. □

Definition 4.5.3 Es sei T eine nichtdeterministische Turingmaschine.

$$L(T) = \{w \in X^* \mid \text{es existiert eine akzeptierende Rechnung an } w\}$$

heißt die *von T akzeptierte Sprache.* □

Mit diesen Definitionen läßt sich zeigen:

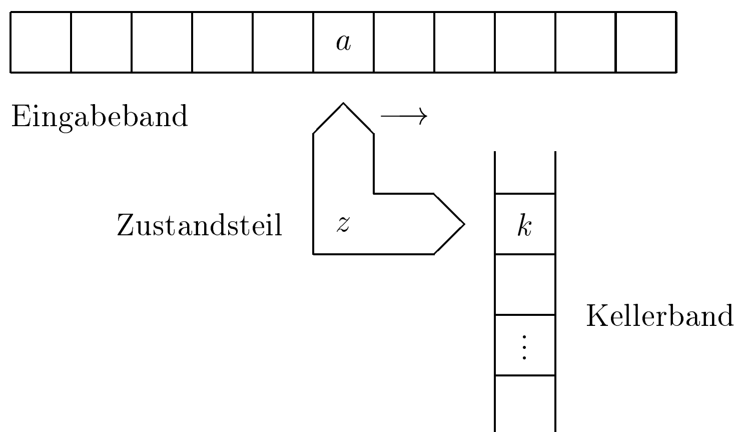
Satz 4.5.1 $\mathcal{L}_0$ ist gleich der Familie der Sprachen, die durch deterministische (oder nichtdeterministische) Turingmaschinen akzeptiert werden. □

Wir sehen also, daß die Familie der Typ-0-Sprachen sowohl durch deterministische als auch durch nichtdeterministische Turingmaschinen charakterisiert wird. Ebenso führen deterministische und nichtdeterministische endliche erkennende Automaten, wie wir in Satz 4.3.3 und Satz 4.3.5 gezeigt haben, zur gleichen Sprachfamilie $\mathcal{L}_3$.

Wir wollen abschließend noch Akzeptoren für die Sprachfamilien $\mathcal{L}_2$ und $\mathcal{L}_1$ angeben.

Definition 4.5.4 $S = (Z, X, K, \delta, z_0, k_0, F)$ heißt *Kellerautomat*, wenn
- (a) Z, X, K endliche nichtleere Mengen sind (Menge der *Zustände*, *Eingaben*, *Kellersymbole*),
- (b) $\delta : Z \times (X \cup \{\varepsilon\}) \times K \to \mathcal{P}_f(Z \times K^*)$ eine Abbildung ist (*Überführungsfunktion*), wobei $\mathcal{P}_f(Z \times K^*)$ die Menge der endlichen Teilmengen von $Z \times K^*$ ist,
- (c) $z_0 \in Z$ (*Anfangszustand*),
- (d) $k_0 \in K$ (*Startsymbol des Kellerbandes*) und
- (e) $F \subset Z$ (*Menge der Endzustände*) gilt. □

Einen Kellerautomaten können wir uns durch das folgende Bild veranschaulichen:



Ein Kellerautomat besteht aus einem Eingabeband, einem Kellerband, einem Zustandsteil, einem Lesekopf für das Eingabeband sowie einem Lese- und Schreibkopf für das Kellerband. Das Eingabeband enthält das jeweilige Eingabewort w, das von links nach rechts gelesen wird. Befindet sich der Kellerautomat im Zustand $z \in Z$, ist

$k \in K$ das oberste Kellerelement und steht unter seinem Lesekopf das Symbol $a \in X$, so kann er entweder a lesen oder auch ignorieren. Im ersten Fall geht der Automat nichtdeterministisch gemäß $(z', P) \in \delta(z, a, k)$ in einen Zustand z' über und ersetzt das oberste Kellerelement k durch P, wobei auch $P = \varepsilon$ gelten darf. Ist $P = k_1 \dots k_r$, $r \in \mathbb{N}$, dann wird k_1 das oberste Kellerelement. Der Kopf über dem Eingabeband geht anschließend zum nächsten Eingabezeichen weiter. Im zweiten Fall wählt der Kellerautomat eine Instruktion $(z', P) \in \delta(z, \varepsilon, k)$. Die Zustandsänderung und das Beschreiben des Kellerbandes erfolgt wie im ersten Fall, jedoch bleibt der Kopf an seinem Platz. Wir sehen, daß der Kellerautomat nur auf das jeweils oberste Element des Kellerbandes zugreifen kann.

Ein Wort w wird von einem Kellerautomaten akzeptiert, wenn er, mit dem Anfangszustand z_0 und dem Startsymbol k_0 beginnend, nach Abarbeitung von w in einen Endzustand übergegangen ist. Dabei beginnt die Bearbeitung von w mit dem ersten Symbol von w. Sie ist beendet, wenn der Lesekopf rechts von w steht, also das letzte Symbol von w abgearbeitet wurde. Eine Sprache L wird von einem Kellerautomaten S erkannt, wenn L aus genau den Wörtern besteht, die von S akzeptiert werden. Es gilt der folgende Satz.

Satz 4.5.2 $\mathcal{L}_2$ ist gleich der Familie der durch nichtdeterministische Kellerautomaten akzeptierten Sprachen. □

Als Spezialfall kann man deterministische Kellerautomaten betrachten. Bei ihnen muß in jeder Situation der nächste auszuführende Schritt eindeutig bestimmt sein. Dies wird durch die folgende Definition erreicht.

Definition 4.5.5 Es sei $S = (Z, X, K, \delta, z_0, k_0, F)$ ein Kellerautomat. S heißt *deterministischer Kellerautomat*, wenn folgende Bedingungen erfüllt sind:

(a) Falls $\delta(z, \varepsilon, k) \neq \emptyset$ für einen Zustand $z \in Z$ und ein Kellersymbol $k \in K$ gilt, so folgt $\delta(z, a, k) = \emptyset$ für alle $a \in X$.

(b) Für alle $z \in Z$, $k \in K$ und $a \in X \cup \{\varepsilon\}$ gilt $|\delta(z, a, k)| \leq 1$. □

Möglich ist, daß ein deterministischer Kellerautomat wegen fehlender Instruktion stehenbleibt. Man kann zeigen, daß für die Familie $\mathcal{DK}$ der von deterministischen Kellerautomaten erkannten Sprachen die Beziehung

$$\mathcal{L}_3 \subsetneq \mathcal{DK} \subsetneq \mathcal{L}_2$$

gilt.

Beispiel 4.5.1 Wir betrachten den deterministischen Kellerautomaten

$$S = (\{z_0, z_1, z_2, z_3\}, \{a, b\}, \{k_0, a, a'\}, \delta, z_0, k_0, \{z_3\}),$$

wobei die Überführungsfunktion gegeben ist durch

$$\begin{array}{rr} \delta(z_0, a, k_0) = (z_1, a'k_0) & \delta(z_1, b, a') = (z_2, a') \\ \delta(z_1, a, a) = (z_1, aa) & \delta(z_2, a, a) = (z_2, \varepsilon) \\ \delta(z_1, a, a') = (z_1, aa') & \delta(z_2, a, a') = (z_3, \varepsilon) \\ \delta(z_1, b, a) = (z_2, a). & \end{array}$$

Zur Vereinfachung sind dabei die einelementigen Mengen durch das jeweilige Element ausgedrückt. Dieser Kellerautomat erkennt die Sprache

$$\{a^n b a^n \mid n \in \mathbb{N}\}.$$

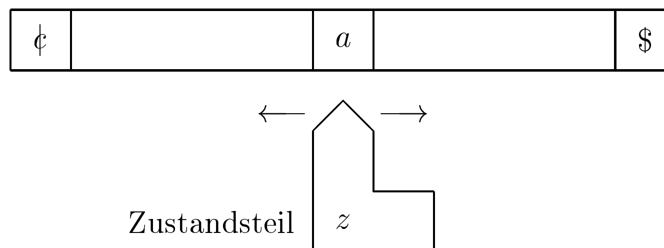
Der Lesekopf wird mit jedem Schritt bewegt. Das erste gelesene a wird als a' in den Keller geschrieben. Falls zuerst ein b gelesen wird, bleibt S wegen fehlender Instruktion stehen. Weitere gelesene Symbole a werden als a oben auf dem bisherigen Kellerinhalt gestapelt. Wird dann erstmals ein b gelesen, so erfolgt ein Übergang in den Zustand z_2, der die Überprüfung der restlichen Symbole vornimmt. Wird anschließend a gelesen und das oberste Kellerelement ist a, so wird a gelöscht. Wird a gelesen und das oberste Kellerelement ist a', so erfolgt ein Übergang in den Endzustand z_3. In den anderen Fällen bleibt S stehen, bevor der Automat das ganze Wort w abgearbeitet hat. Dazu gehört auch der Fall, daß in einem Endzustand ein weiteres Element gelesen wird, was z.B. bei $a^n b a^{n+1}$ oder $a^n b a^n b$ eintreten kann. $\square$

Zum Abschluß gehen wir noch auf das Automatenmodell ein, das die Sprachfamilie $\mathcal{L}_1$ charakterisiert.

Definition 4.5.6 $B = (Z, X, ¢, \$, \delta, z_0, F)$ heißt *linear beschränkter Automat*, wenn $T = (Z, X, \delta, z_0, F)$ eine nichtdeterministische Turingmaschine ist und $¢, \$ \in X$ gilt, so daß folgende Eigenschaften erfüllt sind:

(a) Gilt $(z', x') \in \delta(z, ¢)$ für $z, z' \in Z$, $x' \in \bar{X} \cup \{l, r, s\}$, so folgt $x' \in \{¢, r, s\}$.
(b) Gilt $(z', x') \in \delta(z, \$)$ für $z, z' \in Z$, $x' \in \bar{X} \cup \{l, r, s\}$, so folgt $x' \in \{\$, l, s\}$.
(c) Gilt $(z', x') \in \delta(z, x)$ für $z, z' \in Z$, $x' \in \bar{X} \cup \{l, r, s\}$, $x \in \bar{X} - \{¢, \$\}$, so folgt $x' \neq ¢$ und $x' \neq \$$. $\square$

Wir können einen linear beschränkten Automaten durch das folgende Bild veranschaulichen:



Ein linear beschränkter Automat besitzt ein beschränktes Lese- und Schreibband sowie einen Zustandsteil. Zwischen den beiden Randsymbolen wird das zu bearbeitende Wort w eingetragen, das weder ¢ noch \$ enthält. Der Automat geht in Abhängigkeit von seinem Zustand z und dem gerade gelesenen Symbol a in einen anderen Zustand über und ändert entweder das gelesene Zeichen oder bewegt seinen Schreib- und Lesekopf einen Schritt nach links oder rechts. Die Randsymbole dürfen nicht überschrieben werden. Eine Bewegung über die Ränder hinaus ist nicht möglich, so daß das Blankzeichen überflüssig ist und in der Definition entfernt werden könnte.

Ein Wort $w \in (X - \{¢, \$\})^*$ wird von einem linear beschränkten Automaten akzeptiert, wenn er, ausgehend vom Anfangszustand und angesetzt auf das erste Zeichen

von w (im Falle $w = \varepsilon$ auf das Zeichen $\cent$), irgendwann in einen Endzustand gelangt und hält. Eine Sprache $L \subset \Sigma^*$, $\Sigma \subset X$, wird von einem linear beschränkten Automaten B erkannt, wenn L aus genau den Wörtern $w \in \Sigma^*$ besteht, die von B akzeptiert werden.

Satz 4.5.3 $\mathscr{L}_1$ ist gleich der Familie der durch nichtdeterministische linear beschränkte Automaten akzeptierten Sprachen. □

Beispiel 4.5.2 Ein linear beschränkter Automat, der die Sprache

$$\{a^n b^n c^n \mid n \in \mathbb{N}\}$$

erkennt, arbeitet nach folgendem Prinzip: Durch Hin- und Herlaufen und Markieren von je einem a, b und c pro Durchgang wird geprüft, ob gleich viele a's, b's und c's vorhanden sind. Wir verzichten auf die explizite Angabe der Abbildung δ, die etwas aufwendig ist. □

Für linear beschränkte Automaten können wir, wie schon bei den zugrundeliegenden Turingmaschinen, ein deterministisches Modell definieren. Ungelöst ist hier die Frage, ob deterministische und nichtdeterministische Automaten zur gleichen Sprachfamilie führen, wie es bei endlichen erkennenden Automaten und Turingmaschinen der Fall ist. Dieses offene Problem wird *lbA-Problem* genannt.

5 Fixpunkttheorie und kontext-freie Sprachen

Jeder hat vielleicht schon einmal beim Umrühren einer Tasse Kaffee beobachtet, daß ein „Punkt“ auf der Oberfläche des Kaffees seinen ursprünglichen Platz dabei nicht verändert hat. Diesen „Tropfen“ Kaffee nennt man Fixpunkt. Wahrscheinlich kennt man auch bereits aus der Schule Fixpunkte in der Geometrie. So sind z.B. bei der Achsenspiegelung alle Punkte der Achse Fixpunkte. In der Informatik werden Fixpunkte in der Semantik von Programmiersprachen oder Datenbanken benutzt. Fixpunktmethoden dienen unter anderem dazu, die Semantik eines Programms zu definieren (siehe Kapitel 8 in [23]). Wir werden in diesem Kapitel sehen, daß Fixpunktmethoden auch in der Theorie der formalen Sprachen nützlich sind.

Die grundlegende Idee soll durch das folgende Beispiel veranschaulicht werden. Es sei $G = (V_N, V_T, X_0, F)$ eine kontextfreie Grammatik mit

$$V_N = \{X\},\ V_T = \{(,)\},\ X_0 = X \quad \text{und} \quad F = \{X \to XX, X \to (X), X \to ()\}.$$

Statt durch die Grammatik kann $L(G)$ auch durch das folgende Verfahren beschrieben werden. Man betrachte zunächst die Gleichung

$$\begin{aligned} g(L) &= LL + (L) + () \\ &= \{w_1w_2 \mid w_1, w_2 \in L\} \cup \{(w) \mid w \in L\} \cup \{()\}. \end{aligned}$$

Wenn wir $\emptyset$ an die Stelle von L einsetzen, erhalten wir

$$g(\emptyset) = \emptyset\emptyset + (\emptyset) + () = \{()\}.$$

Weiter ergibt sich

$$g(g(\emptyset)) = g(\{()\}) = \{()(), (()), ()\},$$

und es gilt $g(\emptyset), g(g(\emptyset)) \subset L(G)$ sowie $\emptyset = g^0(\emptyset) \subset g(\emptyset) \subset g(g(\emptyset))$. Später wird man sehen, daß für alle $n \in \mathbb{N}_0$

$$g^n(\emptyset) \subset g^{n+1}(\emptyset), \quad g^n(\emptyset) \subset L(G)$$

sowie

$$\bigcup_{n=0}^{\infty} g^n(\emptyset) = L(G) \quad \text{und} \quad L(G) = g(L(G))$$

gilt. Die letzte Gleichung zeigt, daß $L(G)$ ein Fixpunkt von g ist. Wir werden beweisen, daß $L(G)$ der kleinste Fixpunkt von g ist.

5.1 Partielle Ordnungen und Fixpunkte

Um Fixpunktmethoden in der Theorie der formalen Sprachen anwenden zu können, benötigen wir den *Kleeneschen* Fixpunktsatz. Als Voraussetzung werden zunächst partiell-geordnete Mengen und speziell vollständige partiell-geordnete Mengen eingeführt.

Definition 5.1.1 $(P, \leq)$ heißt *partiell-geordnete Menge* (englisch: *partially ordered set*, kurz: poset), wenn P eine Menge mit einer Ordnungsrelation $\leq$ ist, die für alle $x, y, z \in P$ die folgenden Beziehungen erfüllt:

$$\begin{array}{llll} x \leq x & & & \text{(Reflexivität)} \\ x \leq y, \;\; y \leq z & \Longrightarrow & x \leq z & \text{(Transitivität)} \\ x \leq y, \;\; y \leq x & \Longrightarrow & x = y & \text{(Antisymmetrie).} \end{array} \quad \square$$

Beispiel 5.1.1 Es sei $P = \mathbb{R}$, und $x \leq y$ habe die übliche Bedeutung „x kleiner oder gleich y“. Dann ist $(\mathbb{R}, \leq)$ eine partiell-geordnete Menge. □

Definition 5.1.2 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $A \subset P$. Die *Restriktion* $\leq_A$ von $\leq$ auf A wird durch

$$a \leq_A b \iff a \leq b$$

für alle $a, b \in A$ definiert. Wenn keine Mißverständnisse auftreten können, wird auch $\leq$ anstelle von $\leq_A$ geschrieben. □

Satz 5.1.1 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $A \subset P$. Dann ist $(A, \leq)$ eine partiell-geordnete Menge.

Beweis: Der Beweis folgt unmittelbar aus den Definitionen. □

Man erkennt also sofort, daß jede Menge von reellen, rationalen, ganzen oder natürlichen Zahlen partiell-geordnet ist. Zur Vereinfachung vereinbaren wir die folgenden Bezeichnungen.

Bezeichnungen: Es sei $(P, \leq)$ eine partiell-geordnete Menge. Wir setzen:

$$\begin{array}{lcl} x < y & \iff & x \leq y, \; x \neq y \\ x > y & \iff & y < x \\ x \geq y & \iff & y \leq x \quad . \end{array} \quad \square$$

In $(\mathbb{R}, \leq)$ gilt für alle $x, y \in \mathbb{R}$ die Ungleichung $x \leq y$ oder $y \leq x$. Eine derartige Aussage ist nicht für alle partiell-geordnete Mengen erfüllt, wie das folgende Beispiel zeigt.

Beispiel 5.1.2 Es sei A eine Menge, $P = \mathcal{P}(A)$ die Potenzmenge von A, und $\subset$ sei die Mengeninklusion. Dann ist $(P, \subset)$ eine partiell-geordnete Menge, denn für alle $X, Y, Z \subset A$ gilt offenbar:

$$\begin{array}{lll} X \subset X & & \\ X \subset Y, \; Y \subset Z & \Longrightarrow & X \subset Z \\ X \subset Y, \; Y \subset X & \Longrightarrow & X = Y. \end{array}$$

Man beachte, daß es für $A \neq \emptyset$, $a \in A$ und $A \neq \{a\}$ immer unvergleichbare Elemente bezüglich $\subset$ gibt. Es sei etwa $\{a, b\} \subset A$, $a \neq b$. Dann sind $\{a\}$ und $\{b\}$ unvergleichbar. □

Eine partiell-geordnete Menge kann also Elemente enthalten, die nicht miteinander zu vergleichen sind. Deswegen ist auch der Name *partielle* Ordnung aus Definition 5.1.1 gerechtfertigt.

Beispiel 5.1.3 Es sei $A \neq \emptyset$ eine beliebige nichtleere Menge. Für alle $a, b \in A$ definieren wir eine partielle Ordnung durch

$$a \leq b \iff a = b.$$

Dann sind alle verschiedenen Elemente unvergleichbar. Man sagt, $(A, \leq)$ besitzt die *diskrete* Ordnung. □

Definition 5.1.3 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $A \subset P$. Ein Element a heißt *kleinstes Element* von A, wenn

$$a \in A \text{ und } a \leq b \text{ für alle } b \in A$$

gilt. □

Satz 5.1.2 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $A \subset P$. Dann besitzt A genau ein oder kein kleinstes Element.

Beweis: Es seien a und b kleinste Elemente von A. Nach Definition 5.1.3 gilt $a \leq b$ und $b \leq a$. Aufgrund der Antisymmetrie folgt $a = b$. □

Beispiel 5.1.4 Das kleinste Element von $\mathbb{N}$ ist 1. $\mathbb{Z}$ besitzt kein kleinstes Element. Für jede Menge A ist $\emptyset$ das kleinste Element der Potenzmenge $\mathcal{P}(A)$ von A. □

Definition 5.1.4 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $S \subset P$. x heißt *obere Schranke* (englisch: *upper bound*) von S, wenn

$$x \in P \text{ und } s \leq x \text{ für alle } s \in S$$

gilt. Mit $UB(S)$ bezeichnen wir die *Menge aller oberen Schranken von* S. Dann wird die *kleinste obere Schranke* $\text{lub}(S)$ von S (englisch: *least upper bound*) als kleinstes Element von $UB(S)$ definiert. □

Da es ein kleinstes Element nicht geben muß, muß auch $\text{lub}(S)$ nicht in jedem Fall existieren. Es gilt jedoch der folgende Satz.

Satz 5.1.3 Es sei $(P, \leq)$ eine partiell-geordnete Menge, und es gelte $S \subset P$. Die kleinste obere Schranke $\text{lub}(S)$ von S existiere. Dann ist $\text{lub}(S)$ das eindeutige Element $x \in P$ mit

$$s \leq x \text{ für alle } s \in S \text{ und } x \leq y \text{ für alle } y \in UB(S).$$

Beweis: Da $x = \text{lub}(S)$ existiert, erfüllt es als obere Schranke nach Definition 5.1.4 die Relation $s \leq x$ für alle $s \in S$. Nach Definition 5.1.3 liefert x als kleinstes Element von $UB(S)$ die Gültigkeit von $x \leq y$ für alle $y \in UB(S)$. Die Eindeutigkeitsaussage des Satzes folgt aus Satz 5.1.2. □

Für die Potenzmenge $\mathcal{P}(A)$ von A wollen wir die kleinste obere Schranke einer Menge $S \subset \mathcal{P}(A)$ bestimmen.

Satz 5.1.4 Es sei A eine Menge und $S \subset \mathcal{P}(A)$. Dann gilt

$$\text{lub}(S) = \bigcup_{X \in S} X.$$

Beweis: Wir müssen die Eigenschaften einer kleinsten oberen Schranke aus Definition 5.1.4 für $\bigcup_{X \in S} X$ nachprüfen. Da S eine Menge von Teilmengen von A ist, folgt $\bigcup_{X \in S} X \in \mathcal{P}(A)$. Es sei weiter X' eine beliebige Menge aus S. Dann gilt offensichtlich $X' \subset \bigcup_{X \in S} X$, und wir erhalten, daß $\bigcup_{X \in S} X$ eine obere Schranke von S ist. Wir nehmen schließlich im Gegensatz zur Aussage des Satzes an, daß $\bigcup_{X \in S} X$ keine kleinste obere Schranke ist. Dann existiert eine obere Schranke $Y \in UB(S)$ mit $\bigcup_{X \in S} X \not\subset Y$. Es gibt also ein Element $x \in \bigcup_{X \in S} X$ mit $x \notin Y$. Wegen $x \in \bigcup_{X \in S} X$ folgt jedoch $x \in Z$ für ein geeignetes $Z \in S$. Damit erhalten wir $Z \not\subset Y$, einen Widerspruch zu $Y \in UB(S)$. □

Bezeichnungen: Es sei $(P, \leq)$ eine partiell-geordnete Menge und $(x_n \mid n = 0, 1, \ldots) = (x_n \mid n \geq 0)$ eine Folge von Elementen aus P. Dann wird $\text{lub}\{x_n \mid n = 0, 1, \ldots\}$ mit $\text{lub}(x_n)$, $\bigvee_{n \geq 0} x_n$ oder auch, wenn keine Mißverständnisse auftreten können, mit $\bigvee x_n$ bezeichnet. Statt von der kleinsten oberen Schranke spricht man auch von dem *Supremum* der Folge. Gelegentlich schreiben wir (x_n) für $(x_n \mid n \geq 0)$. □

Definition 5.1.5 Es sei $(P, \leq)$ eine partiell-geordnete Menge.
(a) (x_n) heißt (*aufsteigende*) *Kette* in $(P, \leq)$, wenn (x_n) eine Folge von Elementen aus P ist mit $x_n \leq x_{n+1}$ für alle $n \in \mathbb{N}_0$.
(b) $(P, \leq)$ heißt *vollständige partiell-geordnete Menge* (englisch: *complete partially ordered set*, kurz: cpo), wenn $(P, \leq)$ ein kleinstes Element $\perp$ (englisch: bottom) besitzt und, sofern (x_n) eine Kette in $(P, \leq)$ ist, $\text{lub}(x_n)$ in P existiert. □

Beispiel 5.1.5 Wir betrachten die partiell-geordnete Menge $(\mathbb{N}, \leq)$. Für jede endliche Kette

$$(x_0, x_1, \ldots, x_n)$$

oder jede konstant werdende Kette

$$x_0 \leq x_1 \leq x_2 \leq \ldots \leq x_k = x_{k+1} = \ldots$$

ist das maximale Element die kleinste obere Schranke. Dagegen besitzt die Folge

$$(x_k \mid k \geq 0) \quad \text{mit} \quad x_k = 2^k$$

keine obere Schranke. Um auch in solchen Fällen formal eine kleinste obere Schranke zu erhalten, erweitern wir $(\mathbb{N}, \leq)$ zu $(\mathbb{N} \cup \{\infty\}, \leq)$, wobei $x \leq \infty$ definiert wird für alle $x \in \mathbb{N}$. Dann ist $(\mathbb{N} \cup \{\infty\}, \leq)$ eine vollständige partiell-geordnete Menge (cpo), denn 1 ist das kleinste Element und $\text{lub}(x_n) = \infty$, falls (x_n) kein maximales Element besitzt. □

Beispiel 5.1.6 Wir betrachten die partiell-geordnete Menge $(\mathbb{R}, \leq)$. Sie besitzt kein kleinstes Element. Die Teilmenge $\{x \mid 0 \leq x\}$ hat dagegen ein kleinstes Element 0, aber keine obere Schranke.

Die reellen Zahlen erfüllen jedoch die Vollständigkeitseigenschaft: Ist (x_n) eine Kette in $(\mathbb{R}, \leq)$ mit $UB(x_n) \neq \emptyset$, dann existiert $\mathrm{lub}(x_n)$. Diese Eigenschaft muß nicht in jeder partiell-geordneten Menge gelten. Es sei zum Beispiel $P = \{x \mid x < 0\} \cup \{x \mid 0 < x\} \subset \mathbb{R}$. Ersichtlich ist $(P, \leq)$ partiell-geordnet. Wir definieren die Folge (x_n) durch

$$x_0 = -1, x_1 = -\frac{1}{2}, \ldots, x_n = -\frac{1}{n+1}, \ldots.$$

Dann gilt $UB(x_n) = \{x \mid 0 < x\} \neq \emptyset$. Da die Folge von links gegen $0 \notin P$ konvergiert, existiert allerdings $\mathrm{lub}(x_n)$ nicht. □

Beispiel 5.1.7 Es sei X ein Alphabet. Dann ist die *Teilwort-Ordnung* $\leq_{\mathrm{sub}}$ für $x, y \in X^*$ durch

$$x \leq_{\mathrm{sub}} y \iff \text{es existieren } u, v \in X^* \text{ mit } uxv = y$$

definiert. Durch Überprüfung von Definition 5.1.1 stellt man sofort fest, daß $(X^*, \leq_{\mathrm{sub}})$ eine partiell-geordnete Menge ist. Das kleinste Element ist ε. Bei einer nicht konstant werdenden Kette werden die Wörter immer länger, so daß es keine kleinste obere Schranke gibt und somit $(X^*, \leq_{\mathrm{sub}})$ keine cpo ist. □

Das nächste Beispiel ist für die Anwendung der Fixpunkttheorie von großer Bedeutung.

Beispiel 5.1.8 Es sei $(P, \subset)$ mit $P = \mathcal{P}(A)$ die partiell-geordnete Menge aus Beispiel 5.1.2. $(P, \subset)$ ist eine cpo, denn $\emptyset$ ist ihr kleinstes Element und für jede Kette (X_n) von Teilmengen von A existiert nach Satz 5.1.4 die kleinste obere Schranke $\mathrm{lub}(X_n) = \bigcup X_n$ als ein Element von P.

Als Verallgemeinerung dieser cpo betrachten wir für ein festes $k \in \mathbb{N}$

$$((\mathcal{P}(A))^k, \leq).$$

Ist $pr_i : (\mathcal{P}(A))^k \to \mathcal{P}(A)$ die i-te Projektion, die die i-te Komponente eines k-Tupels aus $(\mathcal{P}(A))^k$ liefert, so ist für $Y, Z \in (\mathcal{P}(A))^k$ die Relation $Y \leq Z$ definiert durch $pr_i Y \subset pr_i Z$ für alle $i = 1, \ldots, k$. Das kleinste Element von $((\mathcal{P}(A))^k, \leq)$ ist offenbar $(\emptyset, \ldots, \emptyset)$ (k-mal). Wenn wir im Beweis von Satz 5.1.4 für S die Kette (X_n) einsetzen und die Inklusion durch die Relation $\leq$ ersetzen, so läßt sich der dortige Beweis vollständig übertragen. Wir erhalten also, daß jede Kette (X_n) von k-Tupeln in $(\mathcal{P}(A))^k$ das k-Tupel

$$\mathrm{lub}(X_n) = (\bigcup(pr_1 X_n), \ldots, \bigcup(pr_k X_n)) \in (\mathcal{P}(A))^k$$

als kleinste obere Schranke besitzt. Folglich ist $((\mathcal{P}(A))^k, \leq)$ eine cpo. □

Satz 5.1.5 Es sei X ein Alphabet und $A = X^*$. (X_n) und (Y_n) seien Ketten in $(\mathcal{P}(A), \subset)$. Dann sind $(X_n Y_n)$ und $(X_n \cup Y_n)$ Ketten in $(\mathcal{P}(A), \subset)$, und es gilt

$$\mathrm{lub}(X_n Y_n) = \mathrm{lub}(X_n)\mathrm{lub}(Y_n) \text{ und } \mathrm{lub}(X_n \cup Y_n) = \mathrm{lub}(X_n) \cup \mathrm{lub}(Y_n).$$

Beweis: Da für Mengen X', X'', Y' und Y'' mit $X' \subset X''$ und $Y' \subset Y''$ die Inklusionen

$$X' \cup Y' \subset X'' \cup Y'' \text{ sowie } X'Y' = \{uv \mid u \in X' \subset X'', v \in Y' \subset Y''\} \subset X''Y''$$

gelten, sind $(X_n \cup Y_n)$ und $(X_n Y_n)$ Ketten.

Wir nehmen an, daß $S = \text{lub}(X_n)\text{lub}(Y_n)$ nicht die kleinste obere Schranke ist. Dann existiert ein $Z \in UB(X_n Y_n)$ mit $S = (\bigcup X_n)(\bigcup Y_n) \not\subset Z$. Folglich gibt es ein $x \in S$ mit $x \notin Z$. Wegen $x \in (\bigcup X_n)(\bigcup Y_n)$ erhält man $x \in X_l Y_r$ für geeignete r und l. Es sei ohne Beschränkung der Allgemeinheit $l \geq r$. Dann gilt wegen der Ketteneigenschaft $x \in X_l Y_l$ und damit $X_l Y_l \not\subset Z$. Das ist ein Widerspruch zu $Z \in UB(X_n Y_n)$.

Analog wird der Beweis für $\text{lub}(X_n \cup Y_n)$ geführt. Dabei ist allerdings die Ketteneigenschaft nicht nötig. □

Die Aussage dieses Satzes in bezug auf die Vereinigungsbildung gilt auch, falls A eine beliebige Menge ist.

Definition 5.1.6 Es seien $(A, \leq_A)$ und $(B, \leq_B)$ partiell-geordnete Mengen und $f : A \to B$ eine Abbildung. $f : (A, \leq_A) \to (B, \leq_B)$ heißt *monoton*, wenn aus $x \leq_A y$ mit $x, y \in A$ die Relation $f(x) \leq_B f(y)$ folgt. □

Wenn keine Mißverständnisse möglich sind, verwenden wir im folgenden für $\leq_A$ und $\leq_B$ dasselbe Zeichen $\leq$. Zur Vereinfachung sagen wir oft auch, daß $f : A \to B$ monoton ist, und wir sprechen von den partiell-geordneten Mengen A und B.

Beispiel 5.1.9 (a) $f : (\mathbb{R}, \leq) \to (\mathbb{R}, \leq)$ mit $f(x) = x + 1$ ist eine monotone Funktion.

(b) Es seien $A, B \neq \emptyset$, und es gelte $A \subset B$ und $b \in B - A$. Dann ist die durch $f(C) = C \cup \{b\}$ definierte Funktion $f : \mathcal{P}(A) \to \mathcal{P}(B)$ monoton.

(c) Es sei X ein Alphabet und $A = X^*$. Wir betrachten die partiell-geordnete Menge $(\mathcal{P}(A), \subset)$. Es sei weiter $L \subset X^*$. Dann definieren wir

$$\text{Sub}(L) = \{x \in X^* \mid \text{es existiert } y \in L \text{ mit } x \leq_{\text{sub}} y\}.$$

Die durch $L \mapsto \text{Sub}(L)$ gegebene Abbildung ist monoton auf der partiell-geordneten Menge $(\mathcal{P}(A), \subset)$. □

Die nächste Definition verbindet Überlegungen der diskreten Mathematik mit Konzepten der Analysis.

Definition 5.1.7 Es seien $(D, \leq)$ und $(D', \leq)$ vollständige partiell-geordnete Mengen, und $f : D \to D'$ sei eine Abbildung. $f : (D, \leq) \to (D', \leq)$ heißt *stetig*, wenn für jede Kette

$$x_0 \leq x_1 \leq x_2 \leq \ldots \quad \text{in } D$$

die kleinste obere Schranke $\bigvee f(x_n)$ der Folge $(f(x_n) \mid n \geq 0)$ in D' existiert und

$$\bigvee f(x_n) = f(\bigvee x_n)$$

erfüllt. □

Man erkennt die Analogie mit der Definition der Stetigkeit in der Analysis, wenn man $\bigvee$ durch lim ersetzt. Es ist ja $f : I\!\!R \to I\!\!R$ stetig, falls für eine Folge (x_n) von reellen Zahlen die Limites $\lim\limits_{n\to\infty} x_n$ und $\lim\limits_{n\to\infty} f(x_n)$ existieren und

$$\lim_{n\to\infty} f(x_n) = f(\lim_{n\to\infty} x_n)$$

gilt.

Satz 5.1.6 Es seien $(D, \leq)$ und $(D', \leq)$ vollständige partiell-geordnete Mengen, und $f : D \to D'$ sei eine Abbildung. Ist f stetig, dann ist f auch monoton.

Beweis: Es sei $f : (D, \leq) \to (D', \leq)$ stetig. Für $a, b \in D$ gelte $a \leq b$. Wir betrachten die Kette

$$a \leq b \leq b \leq b \leq \ldots .$$

Ihre kleinste obere Schranke ist b. Nach Definition 5.1.7 ist $\text{lub}\{f(a), f(b)\} = f(b)$. Es folgt $f(a) \leq f(b)$. $\square$

Wir können aus diesem Satz schließen, daß für eine stetige Funktion die Folge $(f(x_n) \mid n \geq 0)$ aus Definition 5.1.7 ebenfalls eine Kette ist.

Satz 5.1.7 Es sei A eine Menge und $k \in I\!\!N$.

(a) Die Projektionsabbildungen $pr_j : ((\mathcal{P}(A))^k, \leq) \to (\mathcal{P}(A), \subset)$, $1 \leq j \leq k$, sind stetig.

(b) Es sei A' eine beliebige Teilmenge von A. Dann ist die durch die Zuordnung $g(V_1, \ldots, V_k) = A'$ für $V_1, \ldots, V_k \in \mathcal{P}(A)$ gegebene konstante Abbildung $g : ((\mathcal{P}(A))^k, \leq) \to (\mathcal{P}(A), \subset)$ stetig.

Beweis: Es sei (X_n) eine Kette in $(\mathcal{P}(A))^k$.

(a) Nach Beispiel 5.1.8 ist $(pr_j X_n)$ eine Kette in der cpo $\mathcal{P}(A)$, die eine kleinste obere Schranke besitzt. Unter Verwendung von Beispiel 5.1.8 erhalten wir

$$\begin{aligned} \bigvee (pr_j X_n) &= \bigcup (pr_j X_n) \\ &= pr_j(\bigcup (pr_1 X_n), \ldots, \bigcup (pr_k X_n)) \ = \ pr_j(\bigvee X_n). \end{aligned}$$

(b) Wegen $g(X_n) = A'$ folgt $\bigvee g(X_n) = \bigvee A' = A' = g(\bigvee X_n)$. $\square$

Satz 5.1.8 Es sei X ein Alphabet und $A = X^*$, und $h_1, h_2 : ((\mathcal{P}(A))^k, \leq) \to (\mathcal{P}(A), \subset)$ seien stetige Abbildungen. Damit werden Abbildungen $h_1 \cdot h_2$, $h_1 + h_2 : ((\mathcal{P}(A))^k, \leq) \to (\mathcal{P}(A), \subset)$ durch

$$\begin{aligned} (h_1 \cdot h_2)(V_1, \ldots, V_k) &= h_1(V_1, \ldots, V_k) \cdot h_2(V_1, \ldots, V_k) \text{ und} \\ (h_1 + h_2)(V_1, \ldots, V_k) &= h_1(V_1, \ldots, V_k) \cup h_2(V_1, \ldots, V_k) \end{aligned}$$

definiert. Dann sind $h_1 \cdot h_2$ sowie $h_1 + h_2$ stetig.

Beweis: Es sei (X_n) eine Kette in $(\mathcal{P}(A))^k$. Mit Satz 5.1.5 und der Stetigkeit von h_1 und h_2 folgt

$$\begin{aligned}\bigvee(h_1 \cdot h_2)(X_n) &= \bigvee(h_1(X_n) \cdot h_2(X_n)) = \bigvee h_1(X_n) \cdot \bigvee h_2(X_n) \\ &= h_1(\bigvee X_n) \cdot h_2(\bigvee X_n) = (h_1 \cdot h_2)(\bigvee X_n)\end{aligned}$$

und

$$\begin{aligned}\bigvee(h_1 + h_2)(X_n) &= \bigvee(h_1(X_n) \cup h_2(X_n)) = \bigvee h_1(X_n) \cup \bigvee h_2(X_n) \\ &= h_1(\bigvee X_n) \cup h_2(\bigvee X_n) = (h_1 + h_2)(\bigvee X_n). \quad \square\end{aligned}$$

Die Aussage dieses Satzes in bezug auf die Vereinigungsbildung gilt auch, falls A eine beliebige Menge ist.

Satz 5.1.9 Es seien $f : (A, \leq) \to (B, \leq)$ und $g : (B, \leq) \to (C, \leq)$ stetige Abbildungen. Dann ist auch $g \circ f : (A, \leq) \to (C, \leq)$ stetig.

Beweis: Es sei (x_n) eine Kette in A. Dann ist nach den Bemerkungen im Anschluß an den Beweis von Satz 5.1.6 $(f(x_n))$ eine Kette in B. Somit gilt

$$\begin{aligned}\bigvee((g \circ f)(x_n)) &= \bigvee(g(f(x_n))) = g(\bigvee f(x_n)) \\ &= g(f(\bigvee x_n)) = (g \circ f)(\bigvee x_n). \quad \square\end{aligned}$$

Satz 5.1.10 Eine Abbildung $g : ((\mathcal{P}(A))^k, \leq) \to ((\mathcal{P}(A))^k, \leq)$ ist stetig genau dann, wenn $pr_j \circ g = g_j : ((\mathcal{P}(A))^k, \leq) \to (\mathcal{P}(A), \subset)$ für alle $j = 1, \ldots, k$ stetig ist.

Beweis: Ist g stetig, so folgt mit Hilfe von Satz 5.1.7(a) und Satz 5.1.9, daß g_j stetig ist. Für die andere Beweisrichtung stellen wir zunächst fest, daß für Teilmengen $V_1, \ldots, V_k$ von A offenbar

$$g(V_1, \ldots, V_k) = (g_1(V_1, \ldots, V_k), \ldots, g_k(V_1, \ldots, V_k))$$

gelten muß. Es sei nun (X_n) eine Kette in $(\mathcal{P}(A))^k$. Wegen der Stetigkeit der g_j sind die $g_j(X_n)$ Ketten in $\mathcal{P}(A)$. Dann gilt

$$\begin{aligned}\bigvee g(X_n) &= \bigvee(g_1(X_n), \ldots, g_k(X_n)) \\ &= (\bigvee g_1(X_n), \ldots, \bigvee g_k(X_n)) \quad \text{(nach Beispiel 5.1.8)} \\ &= (g_1(\bigvee X_n), \ldots, g_k(\bigvee X_n)) \quad \text{(wegen der Stetigkeit der } g_j\text{)} \\ &= g(\bigvee X_n). \quad \square\end{aligned}$$

Definition 5.1.8 Es sei $(A, \leq)$ eine partiell-geordnete Menge und $f : A \to A$ eine Abbildung. $a \in A$ heißt *Fixpunkt* von f, wenn $f(a) = a$ gilt. $a \in A$ heißt *kleinster Fixpunkt* von f, wenn a ein Fixpunkt von f ist und für alle Fixpunkte b von f die Relation $a \leq b$ gilt. $\square$

Diese Begriffe wollen wir an einem einfachen Beispiel verdeutlichen.

Beispiel 5.1.10 Fixpunkte können existieren oder auch nicht. Für die identische Abbildung $id : \mathbb{N} \to \mathbb{N}$ sind alle $n \in \mathbb{N}$ Fixpunkte. Dabei ist 1 der kleinste Fixpunkt. Die Abbildung $s : \mathbb{N} \to \mathbb{N}$ mit $s(n) = n + 1$ besitzt dagegen keinen Fixpunkt.

Für die Abbildung $id : \mathbb{Z} \to \mathbb{Z}$ sind alle $z \in \mathbb{Z}$ Fixpunkte. Es existiert jedoch kein kleinster Fixpunkt. □

Der folgende Satz ist zum Beweis des *Kleeneschen* Fixpunktsatzes nötig.

Satz 5.1.11 Es sei $(P, \leq)$ eine vollständige partiell-geordnete Menge, und $x_1 \leq x_2 \leq x_3 \leq \ldots$ und $y_1 \leq y_2 \leq y_3 \leq \ldots$ seien Ketten in P, die die folgenden Eigenschaften erfüllen:

(a) Für alle x_i gibt es ein y_j mit $x_i \leq y_j$.
(b) Für alle y_i gibt es ein x_j mit $y_i \leq x_j$.
(Man sagt, (x_i) und (y_j) sind *kofinal ineinander enthalten.*)

Dann folgt

$$\bigvee x_i = \bigvee y_j.$$

Beweis: Es sei $a = \bigvee x_i$ und $b = \bigvee y_j$. Zu zeigen ist $a \leq b$ und $b \leq a$. Wir zeigen $b \leq a$ (analog wird $a \leq b$ bewiesen). Es werde ein beliebiges y_j gewählt. Nach (b) existiert ein k mit $y_j \leq x_k$. Dann gilt $y_j \leq x_k \leq \bigvee x_i = a$. Aufgrund der Transitivität von $\leq$ folgt $y_j \leq a$. Da y_j beliebig ist, erhalten wir $a \in UB(y_j)$ und damit $b = \mathrm{lub}(y_j) \leq a$. □

Satz 5.1.12 (*Kleenescher* Fixpunktsatz) Es sei $(D, \leq)$ eine vollständige partiell-geordnete Menge und $f : (D, \leq) \to (D, \leq)$ eine stetige Abbildung. $\perp$ sei das kleinste Element von $(D, \leq)$. Es sei $f^n = f \circ \cdots \circ f$ die n-fache Komposition von f mit sich selbst. Dann besitzt f einen kleinsten Fixpunkt, und zwar

$$\bigvee_{n \in \mathbb{N}_0} f^n(\perp).$$

Beweis: Man setze $a_0 = \perp$ und $a_n = f^n(\perp)$ für $n \in \mathbb{N}$. Offenbar gilt $a_0 = \perp \leq a_1$. Da f als stetige Funktion auch monoton ist, folgt daraus $f(a_0) \leq f(a_1)$, d.h. $a_1 \leq a_2$. Wir gehen nun weiter induktiv vor und nehmen an, daß $f^n(\perp) \leq f^{n+1}(\perp)$ für ein $n \in \mathbb{N}$ gilt. Wegen der Monotonie von f erhalten wir $f^{n+1}(\perp) \leq f^{n+2}(\perp)$. Wir sehen also, daß $(a_i \mid i \geq 0)$ eine (aufsteigende) Kette ist. Da $(D, \leq)$ eine cpo ist, existiert somit die kleinste obere Schranke $c = \bigvee a_i$. Aus der Stetigkeit von f schließen wir

$$f(c) = f(\bigvee a_i) = \bigvee f(a_i) = \bigvee a_{i+1}.$$

Die Ketten $a_0 \leq a_1 \leq a_2 \leq \ldots$ mit $\bigvee a_i = c$ und $a_1 \leq a_2 \leq a_3 \leq \ldots$ mit $\bigvee a_{i+1} = f(c)$ sind offensichtlich kofinal ineinander enthalten. Nach Satz 5.1.11 folgt $c = f(c)$, d.h., c ist ein Fixpunkt von f.

Wir nehmen nun an, daß d ein beliebiger Fixpunkt von f ist, also $f(d) = d$. Es gilt $a_0 = \perp \leq d$. Aus der Monotonie von f folgt $a_1 = f(\perp) \leq f(d) = d$ und daraus $a_2 = ff(\perp) \leq ff(d) = d$ usw. Man erhält also $d \in UB(a_i \mid i \geq 0)$. Da c die kleinste obere Schranke von $(a_i \mid i \geq 0)$ ist, ergibt sich $c \leq d$. Somit ist c der kleinste Fixpunkt von f. □

5.2 Fixpunkttheorie und kontextfreie Sprachen

Wir kommen auf das zu Beginn dieses Kapitels betrachtete Beispiel zurück.

Beispiel 5.2.1 Man betrachte die kontextfreie Grammatik $G = (V_N, V_T, X, F)$ mit $V_N = \{X\}$, $V_T = \{(,)\}$ und

$$F = \{X \to XX, X \to (X), X \to ()\}.$$

Wir fassen X als Mengenvariable auf und definieren eine Abbildung $g : \mathcal{P}(V_T^*) \to \mathcal{P}(V_T^*)$ durch

$$g(X) = XX + (X) + (),$$

die sich durch „Addition" der Produktionen von G ergibt. Dabei wird + als Vereinigungsbildung interpretiert. Ist X eine beliebige Menge, dann ist

$$\begin{aligned} XX &= \{uv \mid u, v \in X\} \quad \text{(Konkatenation von } X \text{ mit sich selbst)} \\ (X) &= \{(\}X\{)\} = \{(u) \mid u \in X\} \\ () &= \{()\}. \end{aligned}$$

Später werden wir sehen, daß g stetig ist. Dann existiert nach Satz 5.1.12 die kleinste obere Schranke $\bigvee g^n(\perp)$, die auch der kleinste Fixpunkt von g ist. Man erhält

$$\begin{aligned} g(\emptyset) &= \{()\} \\ gg(\emptyset) &= g(\{()\}) = \{()(), (()), ()\} \\ &\text{usw.} \end{aligned}$$

Man überlegt sich, daß die Vereinigung dieser Folgen die Sprache der „passenden" Klammerstrukturen ergibt. □

Beispiel 5.2.2 Es sei $G = (V_N, V_T, X_1, F)$ eine kontextfreie Grammatik. Dabei sei $V_N = \{X_1, X_2, X_3\}$, $V_T = \{a, b\}$ und

$$F = \{X_1 \to X_2, X_1 \to X_3, X_2 \to aX_2b, X_2 \to ab, X_3 \to bX_3a, X_3 \to ba\}.$$

Offenbar gilt

$$L(G) = \{a^n b^n \mid n \geq 1\} \cup \{b^n a^n \mid n \geq 1\}.$$

Wie in Beispiel 5.2.1 betrachten wir X_1, X_2 und X_3 als Variablen, die ihre Werte in $\mathcal{P}(V_T^*)$ annehmen. Die rechten Seiten der Produktionen mit jeweils gleicher linker Seite werden aufaddiert und der entsprechenden Komponente einer Funktion $g : (\mathcal{P}(V_T^*))^3 \to (\mathcal{P}(V_T^*))^3$ zugeordnet. Wir erhalten so

$$g(X_1, X_2, X_3) = (X_2 + X_3, aX_2b + ab, bX_3a + ba).$$

Dabei ist z.B. $aX_2b + ab$ eine Abkürzung für $\{a\}X_2\{b\} \cup \{ab\}$. Nach Beispiel 5.1.8 ist $((\mathcal{P}(V_T^*))^3, \leq)$ eine cpo mit kleinstem Element $\perp = (\emptyset, \emptyset, \emptyset)$, und die Relation $\leq$ ist durch

$$(Y_1, Y_2, Y_3) \leq (Z_1, Z_2, Z_3) \iff Y_1 \subset Z_1, Y_2 \subset Z_2, Y_3 \subset Z_3$$

gegeben. Wenn (X_1^i, X_2^i, X_3^i) eine Kette in $(\mathcal{P}(V_T^*))^3$ ist, dann gilt

$$\text{lub}(X_1^i, X_2^i, X_3^i) = (\bigcup_i X_1^i, \bigcup_i X_2^i, \bigcup_i X_3^i) \in (\mathcal{P}(V_T^*))^3,$$

die kleinste obere Schranke wird also komponentenweise gebildet. Wir werden später sehen, daß g stetig ist. Dann wird nach Satz 5.1.12 der kleinste Fixpunkt von g durch

$$\begin{array}{lcl} g^0(\perp) & = & (\emptyset, \emptyset, \emptyset) \\ g^1(\perp) & = & (\emptyset, \{ab\}, \{ba\}) \\ g^2(\perp) & = & g(\emptyset, \{ab\}, \{ba\}) = (\{ab, ba\}, \{a^2b^2, ab\}, \{b^2a^2, ba\}) \\ \text{usw.} & & \end{array}$$

bestimmt. Durch Induktion über m erhält man

$$g^m(\emptyset, \emptyset, \emptyset)$$
$$= (\{a^kb^k, b^ka^k \mid 0 < k < m\}, \{a^kb^k \mid 0 < k \leq m\}, \{b^ka^k \mid 0 < k \leq m\}).$$

Dann folgt

$$\bigvee_{m \in \mathbb{N}_0} g^m(\perp) = (L(G), \{a^jb^j \mid j \geq 1\}, \{b^ja^j \mid j \geq 1\}).$$

Wir sehen also, daß $L(G)$ die erste Komponente des kleinsten Fixpunktes von g ist. □

Allgemein werden wir in Satz 5.2.1 feststellen, daß jede kontextfreie Grammatik mit k Nichtterminalzeichen eine stetige Abbildung $g : (\mathcal{P}(V_T^*))^k \to (\mathcal{P}(V_T^*))^k$ bestimmt. Jede Komponente ist mit einem anderen Nichtterminalzeichen assoziiert. Ist $(L_1, \ldots, L_k)$ der kleinste Fixpunkt und gehört die erste Komponente zum Anfangssymbol der Grammatik, dann werden wir in Satz 5.2.2 zeigen, daß $L_1 = L(G)$ gilt.

Definition 5.2.1 Es sei $G = (V_N, V_T, X_1, F)$ eine kontextfreie Grammatik. Dabei gelte $V_N = \{X_1, \ldots, X_k\}$ mit $k \in \mathbb{N}$. Die Produktionen aus F mit X_j, $j = 1, \ldots, k$, auf der linken Seite seien durch

$$X_j \to P_{j1}, \ldots, X_j \to P_{jk_j}$$

mit $k_j \in \mathbb{N}_0$ gegeben. Diese k_j Produktionen können für jedes j, $j = 1, \ldots, k$, jeweils als eine Abbildung $g_j : (\mathcal{P}(V_T^*))^k \to \mathcal{P}(V_T^*)$ aufgefaßt werden, die in der Form

$$g_j(X_1, \ldots, X_k) = P_{j1} + \cdots + P_{jk_j}$$

geschrieben werden kann, mit den Nichtterminalzeichen $X_1, \ldots, X_k$ als Variablen über $\mathcal{P}(V_T^*)$. Für $V_i \in \mathcal{P}(V_T^*)$, $i = 1, \ldots, k$, ergibt sich somit $g_j(V_1, \ldots, V_k)$, indem man in $P_{j1} + \cdots + P_{jk_j}$ jedes Vorkommens von X_i durch V_i ersetzt (Interpretation von + und Konkatenation wie in Beispiel 5.2.1). Dann ist die *Abbildung* $g : (\mathcal{P}(V_T^*))^k \to (\mathcal{P}(V_T^*))^k$ *der Grammatik* G durch

$$g(V_1, \ldots, V_k) = (g_1(V_1, \ldots, V_k), \ldots, g_k(V_1, \ldots, V_k))$$

definiert. □

Man beachte, daß speziell für $k_j = 0$ die Abbildung g_j in der Form $g_j(X_1, \ldots, X_k) = \emptyset$ geschrieben wird. Außerdem folgt aus der Definition von g die Gültigkeit von $g_j = pr_j \circ g$, $j = 1, \ldots, k$. Diese Gleichung werden wir in den Beweisen der nächsten beiden Sätze verwenden.

Satz 5.2.1 Die Abbildung $g : (\mathcal{P}(V_T^*))^k \to (\mathcal{P}(V_T^*))^k$ einer kontextfreien Grammatik ist stetig auf $((\mathcal{P}(V_T^*))^k, \leq)$.

Beweis: Wir stellen zunächst fest, daß für jedes $P \in (V_N \cup V_T)^*$ die durch $h(X_1, \ldots, X_k) = \{P\}$ gegebene Funktion $h : (\mathcal{P}(V_T^*))^k \to \mathcal{P}(V_T^*)$ wegen Satz 5.1.7 sowie Satz 5.1.8 (Multiplikation) stetig ist. Dies wollen wir an dem Beispiel $P = abX_jc$ mit $a, b, c \in V_T$ verdeutlichen. Die durch $f_1(X_1, \ldots, X_k) = \{ab\}$ und $f_2(X_1, \ldots, X_k) = \{c\}$ gegebenen konstanten Funktionen sind nach Satz 5.1.7(b) stetig, und die Projektionsfunktion $pr_j(X_1, \ldots, X_k) = X_j$ ist nach Satz 5.1.7(a) stetig. Aus Satz 5.1.8 (Multiplikation) folgt dann, daß $h = f_1 \cdot pr_j \cdot f_2$ stetig ist. Weiter folgt aufgrund der Definition der g_j (siehe Definition 5.2.1) und Satz 5.1.8 (Addition) die Stetigkeit der g_j, $j = 1, \ldots, k$. Wegen $g_j = pr_j \circ g$ schließen wir aus Satz 5.1.10, daß g stetig ist. □

Nach Satz 5.1.12 besitzt die stetige Abbildung g den kleinsten Fixpunkt $\bigvee_{n \geq 0} g^n(\bot)$ mit $\bot = (\emptyset, \ldots, \emptyset)$ (k-mal).

Im Beweis des folgenden Satzes werden Ableitungsbäume kontextfreier Grammatiken benötigt. Wir wollen sie hier nicht formal einführen. Durch das folgende Beispiel wird dieser Begriff deutlich gemacht. Der Strukturbaum aus Beispiel 4.1.2 stellt schon einen Ableitungsbaum dar.

Beispiel 5.2.3 Es sei $G = (\{X\}, \{a, b\}, X, \{X \to a, X \to b, X \to XX\})$ eine kontextfreie Grammatik. Es gilt offenbar $L(G) = \{a, b\}^+$. Zur Erzeugung des Wortes $aba \in L(G)$ gibt es unter anderem folgende Möglichkeiten:

$$\begin{array}{llllllll}
(1) & X & \Longrightarrow & XX & \Longrightarrow & XXX & \Longrightarrow & aXX \Longrightarrow abX \Longrightarrow aba \\
(2) & X & \Longrightarrow & XX & \Longrightarrow & Xa & \Longrightarrow & XXa \Longrightarrow aXa \Longrightarrow aba \\
(3) & X & \Longrightarrow & XX & \Longrightarrow & aX & \Longrightarrow & aXX \Longrightarrow abX \Longrightarrow aba
\end{array}$$

Diese Ableitungen lassen sich durch zwei Ableitungsbäume darstellen:

Ableitungsbaum für (1) und (2)

Ableitungsbaum für (1) und (3)

Die Knoten der Bäume sind mit Zeichen aus $V_N = \{X\}$ oder $V_T = \{a, b\}$ bezeichnet. Die Wurzel ist der einzige Knoten der Höhe 0. Die unmittelbar darüber liegenden Knoten haben die Höhe 1 usw. Beide Bäume haben die Höhe 3. Die Blätter sind mit Terminalsymbolen oder ggf., jedoch nicht in unserem Beispiel, mit ε bezeichnet. Verschiedene Ableitungen können, wie wir auch an diesem Beispiel sehen, denselben Ableitungsbaum besitzen. □

Satz 5.2.2 Es sei G eine kontextfreie Grammatik und g die Abbildung von G. Dann ist die erste Komponente des kleinsten Fixpunktes von g gleich $L(G)$.

Beweis: Es sei $G = (V_N, V_T, X_1, F)$ mit $V_N = \{X_1, \ldots, X_k\}$. Durch Induktion beweisen wir die folgende Behauptung:

> Für $w \in V_T^*$ existiert genau dann ein Ableitungsbaum mit der durch X_j bezeichneten Wurzel und der Höhe $\leq n$, wenn $w \in pr_j(g^n(\perp))$ gilt.

Wir erinnern zunächst daran, daß $\perp = (\emptyset, \ldots, \emptyset)$ (k-mal) gilt.

Für $n = 1$ wird ein Ableitungsbaum für w mit der durch X_j bezeichneten Wurzel durch eine Produktion $X_j \to w$ gegeben. Nach Definition 5.2.1 ist das genau dann der Fall, wenn w als Term in der Definition von g_j erscheint, also $g_j(X_1, \ldots, X_k) = \cdots + w + \cdots$ gilt. Wegen $w \in V_T^*$ ist dies äquivalent zu $w \in g_j(\perp) = pr_j(g(\perp))$.

Wir nehmen nun an, daß die Behauptung für alle $m \leq n$ erfüllt ist.

Für $w \in V_T^*$ existiere ein Ableitungsbaum der Höhe $\leq n + 1$, dessen Wurzel mit X_j bezeichnet ist. Wir betrachten die Knoten der Höhe 1 und nehmen an, daß ihre Bezeichnungen, von links nach rechts konkateniert, das Wort

$$\begin{aligned} \alpha \;&=\; v_1 X_{\alpha_1} v_2 \ldots X_{\alpha_l} v_{l+1} \in (V_N \cup V_T)^*, \\ &\qquad v_1, \ldots, v_{l+1} \in V_T^*,\ X_{\alpha_1}, \ldots, X_{\alpha_l} \in V_N,\ l \in \mathbb{N}_0, \end{aligned}$$

liefern. Zu diesem Wort gehört die Produktion $X_j \to \alpha$, es ist also $g_j(X_1, \ldots, X_k) = \cdots + \alpha + \cdots$. Jedes $X_{\alpha_\lambda} \in V_N$, $\lambda = 1, \ldots, l$, bezeichnet die Wurzel eines Baums der Höhe $\leq n$, der ein Teilwort w_{α_λ} von w ableitet. Nach Induktionsannahme gilt $w_{\alpha_\lambda} \in pr_{\alpha_\lambda}(g^n(\perp))$. Wir betrachten nun $pr_j(g^{n+1}(\perp)) = g_j(g^n(\perp))$. Ein spezielles Wort dieser Menge erhalten wir, wenn wir im Term α von g_j die Variablen X_{α_λ} durch $w_{\alpha_\lambda} \in pr_{\alpha_\lambda}(g^n(\perp))$ ersetzen. Damit ist $v_1 w_{\alpha_1} v_2 \ldots w_{\alpha_l} v_{l+1} = w \in g_j(g^n(\perp))$.

Diese Argumente sind umkehrbar. Es sei $w \in pr_j(g^{n+1}(\perp)) = g_j(g^n(\perp))$. Dann ist w mit Hilfe eines Terms aus g_j gewonnen worden. Wir nehmen an, daß er einer Produktion

$$X_j \to \alpha = v_1 X_{\alpha_1} v_2 \ldots X_{\alpha_l} v_{l+1}$$

wie oben entspricht. Dann hat w die Darstellung $w = v_1 w_{\alpha_1} v_2 \ldots w_{\alpha_l} v_{l+1}$ mit $w_{\alpha_\lambda} \in pr_{\alpha_\lambda}(g^n(\perp))$. Nach Induktionsannahme existiert für $w_{\alpha_\lambda} \in V_T^*$ ein Ableitungsbaum der Höhe $\leq n$, dessen Wurzel mit X_{α_λ} bezeichnet ist. Das Einsetzen dieser Ableitungsbäume an der Stelle der jeweiligen X_{α_λ} in der Produktion $X_j \to \alpha$ liefert einen Ableitungsbaum für $w \in V_T^*$ mit der durch X_j bezeichneten Wurzel und der Höhe $\leq n + 1$. Damit ist der Induktionsbeweis beendet.

Aus der eben bewiesenen Behauptung ergibt sich, daß $X_1 \Longrightarrow^* w$ mit $w \in V_T^*$, also $w \in L(G)$, genau dann erfüllt ist, wenn ein $n \geq 1$ existiert mit $w \in pr_1(g^n(\perp))$. Das bedeutet, daß

$$w \in \bigcup_{n\geq 0} pr_1(g^n(\perp))$$

gilt. Wegen der Stetigkeit von pr_1 erhalten wir äquivalent

$$w \in pr_1(\bigvee_{n\geq 0} g^n(\perp)).$$

Somit ist $L(G)$ nach Satz 5.1.12 die erste Komponente des kleinsten Fixpunktes $\bigvee_{n\geq 0} g^n(\perp)$ von g. □

Die Aussage von Satz 5.2.2 kann man so interpretieren, daß $g(X_1, \ldots, X_k)$ als ein System von Gleichungen aufgefaßt und nach $X_1, \ldots, X_k$ aufgelöst wird. Für den kleinsten Fixpunkt $(V_1, \ldots, V_k)$ von g gilt ja $g(V_1, \ldots, V_k) = (V_1, \ldots, V_k)$. So möchten wir in Beispiel 5.2.1 die Gleichung

$$X = XX + (X) + ()$$

auflösen nach der Mengenvariablen X und in Beispiel 5.2.2 das Gleichungssystem

$$\begin{aligned} X_1 &= X_2 + X_3 \\ X_2 &= aX_2b + ab \\ X_3 &= bX_3a + ba \end{aligned}$$

nach den Mengenvariablen X_1, X_2 und X_3. Diese Betrachtungsweise der Abbildung g einer kontextfreien Grammatik G wird auch im folgenden Satz angenommen. Die erhaltene Aussage ist aber allgemeiner gültig.

Satz 5.2.3 Es seien $A, B \subset V^*$ Sprachen über dem Alphabet V. Die kleinste Lösung der Gleichung

$$X = AX + B$$

ist A^*B, wobei $A^* = \bigcup_{n\in \mathbb{N}_0} A^n$ gilt (siehe Definition 4.1.1).

Beweis: Es ist $(\mathcal{P}(V^*), \subset)$ nach Beispiel 5.1.8 eine vollständige partiell-geordnete Menge mit $\perp = \emptyset$. Wir definieren eine Abbildung $g : \mathcal{P}(V^*) \to \mathcal{P}(V^*)$ durch

$$g(X) = AX + B.$$

Nach den Sätzen 5.1.7 und 5.1.8 ist g stetig. Es gilt $g(\emptyset) = B$, $g^2(\emptyset) = AB \cup B$ und allgemein $g^{n+1}(\emptyset) = A^nB \cup A^{n-1}B \cup \cdots \cup AB \cup B$ für $n \in \mathbb{N}_0$. Nach Satz 5.1.12 folgt, daß $\emptyset \cup B \cup AB \cup A^2B \cup \cdots = A^*B$ der kleinste Fixpunkt von g ist. □

Man kann zeigen, daß die Sprache A^*B vom Typ 3 ist, wenn A und B vom Typ 3 sind. Die Gleichung $X = AX + B$ muß, wie wir sehen, nicht unbedingt aus den Produktionen einer Grammatik herrühren. Wir erhalten in Satz 5.2.3 direkt eine geschlossene Lösung A^*B. Es ist nicht nötig, wie in Satz 5.2.2 die Folge $\perp, g(\perp), g^2(\perp)$ usw. zu berechnen.

Analog zu Satz 5.2.3 kann der folgende Satz bewiesen werden.

Satz 5.2.4 Es sei V ein Alphabet und $A, B \subset V^*$ Sprachen über V. Die kleinste Lösung der Gleichung

$$X = XA + B$$

ist BA^*. □

Beispiel 5.2.4 Wir betrachten das Gleichungssystem

(1) $$X_1 = aX_1 + bX_2 + a$$

(2) $$X_2 = aX_1 + aX_2 + b \quad .$$

Dabei sind X_1 und X_2 Variablen über $\mathcal{P}(V^*)$, wobei $V = \{a, b\}$ gilt. Mit Hilfe von Satz 5.2.3 erhalten wir aus (1)

(3) $$X_1 = a^*(bX_2 + a),$$

wobei a^* als abkürzende Schreibweise für $\{a\}^*$ verwendet wird. Einsetzen von (3) in (2) und anschließende Umordnung liefert

(4) $$X_2 = (aa^*b + a)X_2 + aa^*a + b.$$

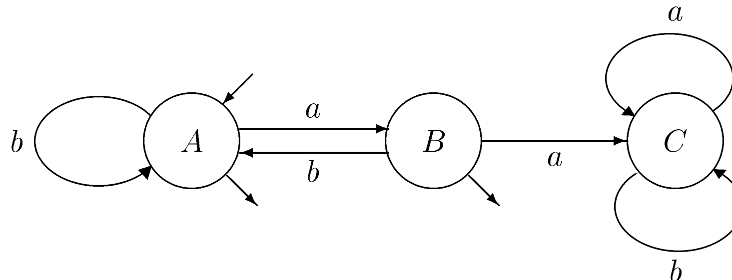
Erneute Anwendung von Satz 5.2.3 ergibt

(5) $$X_2 = (aa^*b + a)^*(aa^*a + b).$$

Durch Einsetzen von (5) in (3) finden wir dann die kleinste Lösung für X_1, nämlich

$$a^*(b(aa^*b + a)^*(aa^*a + b) + a). \quad □$$

Beispiel 5.2.5 Wir betrachten den folgenden endlichen erkennenden Automaten E:



Entsprechend dem Beweis von Satz 4.3.5 konstruieren wir zwei Typ-3-Grammatiken G_A und G_B, die beide dieselbe Produktionenmenge F haben. Dabei ist F durch die Produktionen

$$A \to Ab, \quad A \to Bb, \quad A \to \varepsilon,$$
$$B \to Aa,$$
$$C \to Ba, \quad C \to Ca, \quad C \to Cb$$

gegeben. Die Anfangssymbole sind A bzw. B. In den Grammatiken ist C von A oder B aus nicht erreichbar, so daß die Produktionen, die C enthalten, entfallen können. Nach dem Beweis von Satz 4.3.5 ist $L(E) = L(G_A) \cup L(G_B)$. Diese Vereinigung wird durch die Grammatik G erzeugt, die wir durch Hinzunahme der Produktionen

$$Y \to A, \; Y \to B$$

erhalten, wobei das neue Symbol Y das Anfangssymbol ist. $L(E) = L(G)$ ist nun die zu Y gehörende Komponente des kleinsten Fixpunktes der Abbildung der Grammatik

G, d.h. die kleinste Lösung für Y im Gleichungssystem

$$\begin{aligned} A &= Ab + Bb + \varepsilon \\ B &= Aa \\ Y &= A + B. \end{aligned}$$

Einsetzen von B in der ersten Gleichung liefert

$$A = Ab + Aab + \varepsilon = A(b + ab) + \varepsilon.$$

Nach Satz 5.2.4 ist

$$A = \varepsilon(b + ab)^* = (b + ab)^*$$

die kleinste Lösung für A. Somit ist

$$Y = A + B = A + Aa = A(\varepsilon + a) = (b + ab)^*(\varepsilon + a)$$

die kleinste Lösung für Y. Es folgt

$$L(E) = \{b, ab\}^*\{\varepsilon, a\}. \quad \square$$

Literaturverzeichnis

[1] *K. Alber, W. Struckmann:* Einführung in die Semantik von Programmiersprachen. Bibliographisches Institut, Wissenschaftsverlag, Mannheim 1988.

[2] *S. Baase:* Computer Algorithms. Adison-Wesley, Reading 1978.

[3] *L. S. Bobrow, M. A. Arbib:* Discrete Mathematics. Saunders, Philadelphia 1974.

[4] *D. P. Bovet, P. Crescenzi:* Introduction to the Theory of Complexity. Prentice Hall, New York 1994.

[5] *A. Brandstädt:* Graphen und Algorithmen. Teubner, Stuttgart 1994.

[6] *W. Brauer:* Automatentheorie. Teubner, Stuttgart 1984.

[7] *M. R. Garey, D. S. Johnson:* Computers and Intractability. Freeman, San Francisco 1979.

[8] *M. Gössel:* Automatentheorie für Ingenieure. Akademie-Verlag, Berlin 1991.

[9] *H. Hermes:* Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit, 3. Auflage. Springer, Berlin 1978.

[10] *B. Hohlfeld, W. Struckmann:* Einführung in die Programmverifikation. Bibliographisches Institut, Wissenschaftsverlag, Mannheim 1992.

[11] *J. E. Hopcroft, J. D. Ullman:* Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. Addison-Wesley, Bonn 1990.

[12] *J. JáJá:* An Introduction to Parallel Algorithms. Addison-Wesley, Reading 1992.

[13] *D. E. Knuth:* The Art of Computer Programming. Volume 2 (Seminumerical Algorithms), second edition. Addison Wesley, Reading 1981.

[14] *M. Minsky:* Computation: Finite and Infinite Machines. Prentice-Hall, London 1972.

[15] *R. N. Moll, M. A. Arbib, A. J. Kfoury:* An Introduction to Formal Language Theory. Springer, New York 1988.

[16] *H. Noltemeier:* Informatik 1. Einführung in Algorithmen und Berechenbarkeit. Hanser, München 1981.

[17] *H. Noltemeier, R. Laue:* Informatik 2. Einführung in Rechnerstrukturen und Programmierung. Hanser, München 1984.

[18] *C. Posthoff, K. Schultz:* Grundkurs Theoretische Informatik. Teubner, Stuttgart 1992.

[19] *A. Salomaa:* Formale Sprachen. Springer, Berlin 1978.

[20] *P. Sander, W. Stucky, R. Herschel:* Automaten, Sprachen, Berechenbarkeit. Teubner, Stuttgart 1992.

[21] *P. H. Starke:* Abstrakte Automaten. VEB Deutscher Verlag der Wissenschaften, Berlin 1969.

[22] *R. Vollmar, Th. Worsch:* Modelle der Parallelverarbeitung. Teubner, Stuttgart 1994.

[23] *D. Wätjen:* Theoretische Informatik. Eine Einführung. Oldenbourg, München 1994.

Index

Abbildung einer kontextfreien Grammatik, 111
Ableitung, 81
Ableitungsbaum einer kontextfreien Grammatik, 112
Abschluß unter Sprachoperationen
  Homomorphismus, 95
  Spiegeloperation, 95
  Substitution, 95
Ackermann-Funktion, 67–72
Adjazenzmatrix, 121
Äquivalenz
  von endlichen erkennenden Automaten, 86
  von Grammatiken, 83
  von Turingmaschinen, 26
  von Typ-3-Grammatiken und regulären Ausdrücken, 94
  von Zuständen, 18, 87
  zwischen Moore- und endlichen erkennenden Automaten, 86
  zwischen Turingmaschinen und Registermaschinen, 54
Äquivalenzrelation, 16
  Äquivalenzklasse, 16
  Faktormenge, 16
Alphabet, 79
Anfangssymbol, 82
Antwortfunktion eines Moore-Automaten, 13
Arbeitsweise einer Turingmaschine, 24–25
Aufzählung
  von berechenbaren Funktionen, 46
  von Turingmaschinen, 42
Ausdruck, logischer, 132
  Belegung der Variablen, 133
  erfüllbar, 133
  quantifizierte Boolesche Formel (QBF), 188
  SAT, 134
  Standardkodierung, 133
Automat, *siehe*
abstrakter Automat bei Programmiersprachen
endlicher erkennender Automat
Kellerautomat
linear beschränkter Automat
Mealy-Automat
Moore-Automat
Turingmaschine

Bandfunktion, 24, 40
Bandinhalt, 28
Bandinschrift, 24, 28, 40
Belegung von logischen Variablen, 133
beschränkte Minimalisierung, 76
beschränkter μ-Operator, 73
Blankzeichen, 23

charakteristische Funktion, 46, 72, 79
Chomsky-Hierarchie, 85
Churchsche These, 45, 64, 78
CLIQUE, 165
COMMON CRCW PRAM, 192
cpo, *siehe* vollständige partiell-geordnete Menge
CRCW PRAM, 192
CREW PRAM, 192

3-FÄRBBARKEIT, 128
dualer Graph, 137

EA-Turingmaschine, 211
Einsetzung, 61
endlicher erkennender Automat, 85
  Äquivalenz, 86
  Äquivalenz von Zuständen, 87
  erreichbarer Zustand, 87
  nichtdeterministischer, 88
  reduzierter, 88
  vereinfachter, 87
  Zustandsüberführungsgraph, 86
Entscheidbarkeit, 44
Entscheidungsproblem, 117
varepsilon-approximierender Algorithmus, 179

erfüllbarer Ausdruck, 133
Erfüllbarbeitsproblem, 203
Erfüllbarkeitsproblem, 134
Ersetzungsregel, 81, 82

Familie von Sprachen, 83, 85
Fixpunkt, 108
Fixpunktsatz von *Kleene*, 109
FNC, Komplexitätsklasse, 205
FP, Komplexitätsklasse, 135
freie Halbgruppe, 11
freies Monoid, 11

Gödelisierung, 40, 41
Grammatik, 82
  Abbildung einer kontextfreien Grammatik, 111
  Ableitungsbaum einer kontextfreien Grammatik, 112
  Äquivalenz von Grammatiken, 83
  erzeugte Sprache, 82
  kontextfrei, 83
  kontextsensitiv, 83
  monoton, 83
  regulär, 83
  Typ i, 83
Graph, 121
  Adjazenzlisten, 121
  Adjazenzmatrix, 121
  benachbarte Knoten, 126
  Dreieck, 126
  3-FÄRBBARKEIT, 128
  duale, 137
  k-FÄRBBARKEIT, 169
  Multigraph, 121
  schlichter, 121
  Schlinge, 126
  starke Komponente, 135
  topologische Ordnung, 123
  Weg, 123
  2-FÄRBBARKEIT, 126–135
  2-FÄRBBARKEIT, 128
  zyklenfrei, 123
  Zyklus, 123

Halbgruppe, 11
Halteproblem, 44, 45, 48
HAMILTONSCHER KREIS, 169
Handlungsreisender, 155
Heiratsproblem, 174
Homomorphismus, 95
Hyperwürfel, 217

Induktions-Rekursionsschema, 62
INNERE EINES POLYGONS, 147
Iteration, 79

k-Äquivalenz von Zuständen, 19
k-Band-Turingmaschine, 33, 142
k-FÄRBBARKEIT, 169
KANTEN-2-FÄRBBARKEIT, 138
Kellerautomat, 97–98
  deterministischer, 98
Kette, 104
Klausel vom Grad k, 133
Kleenescher Fixpunktsatz, 109
kleinste obere Schranke, 103, 104
kleinster Fixpunkt, 108
  bei Grammatiken, 113
KNAPSACK-PROBLEM, 155
KNOTENÜBERDECKUNG eines Graphen, 170
kombinatorisches System, 81
Komplexität, 16, 22
Komplexitätsklasse
  FNC, 205
  FP, 135
  NC, 204
  NP, 154
  NPO, 172
  P, 117, 131
  PO, 173
  PSPACE, 185
Komposition von Turingmaschinen, 27
Konfiguration, 24
Konkatenation
  von Sprachen, 79
  von Wörtern, 11
KONVEXE HÜLLE, 148, 198

lba-Problem, 100
leeres Wort ε, 11
linear beschränkter Automat, 99
Linksmaschine, 26
Literal, 133

Matching, 173

Matrizenmultiplikation, 216, 219
MAX, 192
MAXIMALE CLIQUE, 172
MAXIMALFLUSSPROBLEM, 176, 206
Maximierungsproblem, 172
Mealy-Automat, 7, 30
  Äquivalenz mit Moore-Automaten, 12
  Taktung, 8
  Wertetabelle, 8
  Zustandsdiagramm, 9
  Zustandsgraph, 9
Menge von Wörtern, 11
MINIMALE FÄRBBARKEIT, 171
MINIMALE KNOTENÜBERDECKUNG, 178
MINIMALES TSP, 172
Minimalisierung, 76
Minimierungsproblem, 171
Modulo-m-Zähler, 5
Monoid, 11
monotone Abbildung, 106
Moore-Automat, 10
  Äquivalenz mit Mealy-Automaten, 12
  Äquivalenz mit reduziertem Automaten, 18
  Äquivalenz von Zuständen , 18
  Anfangszustand, 13
  Antwortfunktion, 13
  Arbeitsweise, 11
  k-Äquivalenz von Zuständen, 19
  Realisierungsproblem, 14
  Reduktion, 17
  Reduktionsalgorithmus, 20
  reduzierter, 18
  Taktung, 10
  Verhaltensfunktion, 13
  Zustand-Ausgabe-Graph, 10
μ-Operator, 73
μ-rekursiv, 77
Multigraph, 121

NC, Komplexitätsklasse, 204
NC-reduzierbar, 205
Netzwerk, 214
  Hyperwürfel, 217
  Ring, 214
  zweidimensionales Gitter, 215
nichtdeterministische Turingmaschine, 32, 96
nichtdeterministischer endlicher erkennender Automat, 88
normiert Turing-berechenbar, 36, 64
NP, Komplexitätsklasse, 117, 154
NP-hart, 178
NP-vollständig, 157
NPO, Komplexitätsklasse, 172

O-Notation, 22
O-Notation, 119
obere Schranke, 103
Ω-Notation, 119
Optimierungsproblem, 171

P, Komplexitätsklasse, 117, 131
P-vollständig, 205
Palindrome, 130, 208
parallele Berechenbarkeitsthese, 213
partiell-geordnete Menge, 102
  Fixpunkt, 108
  Kette, 104
  kleinste obere Schranke, 103, 104
  kleinstes Element, 103
  monotone Abbildung, 106
  obere Schranke, 103
  Supremum, 104
  vollständige partiell-geordnete Menge, *siehe* vollständige partiell-geordnete Menge
partielle Abbildung, 31
PARTITION, 170
PO, Komplexitätsklasse, 173
Polygon, 145
polynomial reduzierbar, 137
poset, *siehe* partiell-geordnete Menge
Postsches Korrespondenzproblem, 49
Prädikat, 72
  charakteristische Funktion, 72
Präfixsumme, 199
PRAM, 191
  Scheduling-Darstellung, 194
  WT-Darstellung, 194
primitiv-rekursive Funktion, 61, 62

Beispiele, 62–63
Grundfunktionen, 61
Nachfolgerfunktion, 61
Nullfunktion, 61
Projektionsfunktion, 61
primitives Rekursionsschema, 62
PRIORITY CRCW PRAM, 192
Produktion, 81, 82
Projektionsfunktion, 10, 61, 105, 107
PSPACE, Komplexitätsklasse, 185

QBF, 188, 189
quantifizierte Boolesche Formel (QBF), 188

RAM, 50, 143
Äquivalenz mit Turingmaschine, 54
Arbeitsweise, 51
Bestandteile, 50
deterministische, 51
nichtdeterministische, 51
Programm, 51
RAM-Berechenbarkeit, 52
RAM-berechenbar, 52
Raumkomplexität, 185
Realisierungsproblem, 14
Rechtsmaschine, 26
Reduktion
NC-, 205
polynomiale, 137
Reduktion von Moore-Automaten, 17
Reduktionsalgorithmus für Moore-Automaten, 20
reduzierter endlicher erkennender Automat, 88
reduzierter Moore-Automat, 18
Registermaschine, *siehe* RAM, 143
logarithmische Zeitkomplexität, 143
uniforme Zeitkomplexität, 143
regulärer Ausdruck, 93
universell, 134
rekursiv, 84
rekursiv-aufzählbar, 84
Ring, 214

SAT, 134, 154, 203
SAT(2), 134, 140
SAT(3), 163
Schaltfunktion, 8
Schaltkreis, 207
Schaltkreisfamilie, 208
uniforme, 212
Schaltkreisfunktion, 207
Schaltnetz, 8
Scheduling Independent Tasks, 180
Schreibmaschine, 26
serielle Addition, 5
SIT(k), 180
Sortieren, 196–198
Spiegeloperation, 95
Sprache, 79
kontextfrei, 83
kontextsensitiv, 83
Palindrome, 130
regulär, 83, 94
rekursiv, 84
rekursiv-aufzählbar, 84
Typ i, 83
von endlichem Automaten erkannt, 85
von nichtdeterministischem endlichen Automaten erkannt, 88
Sprachfamilie, 83, 85
starke Komponente, 135, 140
stetige Abbildung, 106
Strukturbaum, 80
systolischer Algorithmus, 216

Taktung von Mealy-Automaten, 8
Taktung von Moore-Automaten, 10
topologische Ordnung, 123
topologisches Sortieren, 123–126
totale Funktion, 61
TRAVELING-SALESMAN-PROBLEM, 155
TSP, 155, 169
Turing-berechenbar, 34, 36
Turingmaschine, 23
Äquivalenz mit Registermaschine, 54
Äquivalenz von Turingmaschinen, 26
akzeptierende, 96
akzeptierende Rechnung, 96

akzeptierte Sprache, 97
Arbeitsweise, 24–25
Aufzählung, 42
Bandfunktion, 24, 40
Bandinhalt, 28
Bandinschrift, 24, 28, 40
Blankzeichen, 23
deterministische
  Zeitkomplexität, 129
Endzustände, 96
Gödelisierung, 40, 41
große Linksmaschine, 28
große Rechtsmaschine, 28
haltende Rechnung, 96
Halteproblem, 44, 45
Interpretation, 24
Komposition, 27
Konfiguration, 24
Kopiermaschine, 28
Länge der Rechnung, 96
Leerzeichen, 23
linke Translationsmaschine, 29
Links-Suchmaschine, 29
Linksmaschine, 26
m-Kopiermaschine, 30
mit Ein- und Ausgabeband, 211
mit k Bändern, 33, 142
modifizierte Definitionen, 31–34
nichtdeterministische, 32, 96
  Zeitkomplexität, 153
normiert Turing-berechenbar, 36
Raumkomplexität, 185
Rechts-Suchmaschine, 29
Rechtsmaschine, 26
Schreibmaschine, 26
Turing-berechenbar, 34, 36
Turingtafel, 23–24
  universelle, 43
  unvollständige, 31
  Verschiebemaschine, 30
Turingtafel, 23–24
Turnieralgorithmus, 193
Typ-i-Grammatik, 83
Typ-i-Sprache, 83

unbeschränkter μ-Operator, 73
Unentscheidbarkeit, 44
uniforme Schaltkreisfamilie, 212
universelle Turingmaschine, 43
unvollständige Turingmaschine, 31

Verhalten eines Moore-Automaten, 13
Verhaltensfunktion, 14
vollständige partiell-geordnete Menge, 104
  Fixpunktsatz, 109
  stetige Abbildung, 106

Wertetabelle von Mealy-Automaten, 8
Wort, 11

X^*-Erweiterung, 11, 88

ZERLEGBARKEIT, 153
Zertifikatensprache, 162
Zuordnung, 173
Zustand-Ausgabe-Automat, *siehe* Moore-Automat
Zustand-Ausgabe-Graph, 10
Zustandsdiagramm, 6, 9
Zustandsgraph, 6, 9
Zustandsüberführungsgraph, 86
2-FÄRBBARKEIT, 126–135
2-FÄRBBARKEIT, 128
zweidimensionales Gitter, 215
Zyklus, 123