Prof. Dr. Wolfgang J. Paul

Komplexitätstheorie

Teubner Verlag, 1978

Vorwort

Die Komplexitätstheorie ist ein Zweig der theoretischen Informatik bzw. der Angewandten Mathematik, der sich mit quantitativen Aspekten der Berechenbarkeit beschäftigt. Während man in der Rekursionstheorie danach fragt, welche Funktionen überhaupt durch Algorithmen, d.h. durch mechanische Rechenverfahren, berechenbar sind und wie man die nichtberechenbaren Funktionen klassifizieren kann, bemüht man sich in der Komplexitätstheorie darum, die berechenbaren Funktionen nach ihrer Komplexität, d.h. nach dem zu ihrer Berechnung notwendigen Aufwand, zu klassifizieren.

Will man zeigen, daß eine Funktion mit einem gewissen Rechenaufwand berechnet werden kann, so gibt man im allgemeinen ein Programm an, das diese Funktion berechnet, und analysiert die Komplexität dieses Programms. Man kennt viele Techniken zum Design effizienter Algorithmen [2,42]. Im Gegensatz dazu kennt man nur wenige Techniken, hauptsächlich Diagonalisierung, Reduzierbarkeit und Simulation, mit denen man zeigen kann, daß ein gewisser Rechenaufwand zur Berechnung gewisser Funktionen notwendig ist. Das ist einerseits traurig, andererseits reichen diese wenigen Techniken erstaunlich weit, und man erhält eine einheitliche Theorie.

Dieses Buch präsentiert den aktuellen Stand dieser Theorie. Es ist aus einer Vorlesung im Wintersemester 1976/77 an der Universität Bielefeld entstanden.

Um das Buch außer für Informatikstudenten auch für Mathematikstudenten zugänglich zu machen, werden nur grundlegende Mathematikkenntnisse, aber keine Kenntnisse in Programmierung oder Rekursionstheorie vorausgesetzt.

Jeder Abschnitt enthält Übungsaufgaben, sehr viele Abschnitte enthalten offene Probleme.

Ich danke Frau W. Blenski für das Schreiben des Manuskripts und Dany für das Korrekturlesen.

Bielefeld, im Februar 1978

W.J. P.

Inhalt

1.	Berechenbarkeit	8
1.1.	Intuitives zum Begriff der Berechenbarkeit	8
1.2.	Die μ-rekursiven Funktionen	12
1.3.	Turingmaschinen	25
2.	Nicht berechenbare Funktionen	56
2.1.	Halteproblem und Reduzierbarkeit	56
2.2.	Die Unvollständigkeit der Arithmetik	68
3.	Realistische Komplexitätsmaße	86
3.1.	Registermaschinen	86
3.2.	Gegenseitige Simulation von Turingmaschinen und Registermaschinen	94
3.3.	Schnelle Simulation von Turingmaschinen durch Registermaschinen	1o4
4.	Komplexität von Turingmaschinen	113
4.1.	Akzeptoren und Komplexitätsklassen	113
4.2.	Hierarchiesätze	116
4.3.	Weitere untere Schranken	132
5.	Nichtdeterminismus	142
5.1.	Nichtdeterministische und deterministische Komplexitätsklassen	142
5.2.	Vergleich zwischen Speicherplatz und Rechenzeit	151
5.3.	Grammatiken	17o
6.	Vollständige Probleme	181
6.1.	Effiziente Reduzierbarkeit	181
6.2.	NP-vollständige Probleme	186
6.3.	PTAPE-vollständige Probleme	199
6.4.	Ein nachweislich schweres Problem	214
7.	Abstrakte Komplexitätstheorie	22o
7.1.	Abstrakte Komplexitätsmaße	22o
7.2.	Speed-up und Gap-Theorem	23o
7.3.	Anhang: Physikalische Grenzen für die Geschwindigkeit von Schaltvorgängen	236

Symbolverzeichnis 238

Literaturverzeichnis 24o

Sachverzeichnis 246

1. Berechenbarkeit

1.1. Intuitives zum Begriff der Berechenbarkeit

Ziel dieses einleitenden Kapitels ist es, den intuitiven Begriff der Berechenbarkeit formal zu fassen, d.h. die im intuitiven Sinn berechenbaren Funktionen formal zu charakterisieren.

Intuitiv ist eine Funktion f berechenbar, wenn wir uns einen Algorithmus, d.h. ein "effektives Rechenverfahren" vorstellen können, das, gestartet mit Eingabe x aus dem Definitionsbereich von f, schließlich als Ausgabe f(x), den Funktionenwert von f an der Stelle x, liefert. Beispiele für Algorithmen sind etwa die Verfahren zur Addition, Subtraktion, Multiplikation und Division von Dezimalzahlen, die man in der Schule lernt.

Die Charakterisierung soll als adäquat gelten, wenn man formal eine Funktionenklasse K definieren kann, die aus lauter berechenbaren Funktionen besteht und die auch alle im intuitiven Sinn berechenbaren Funktionen umfaßt. Eine solche Charakterisierung könnte z.B. dadurch erreicht werden, daß man eine spezielle exakt definierte Klasse von Rechenverfahren angibt, mit denen man jede Funktion berechnen kann, die durch irgendein Rechenverfahren, gleich welcher Art, berechenbar ist. Eine solche formale Charakterisierung sämtlicher berechenbarer Funktionen braucht man nicht, um von einer Funktion nachzuweisen, daß sie berechenbar ist; hierzu genügt die Angabe irgendeines Verfahrens zur Berechnung der Funktion. Man benötigt aber eine ganz präzise Formulierung dessen, was man unter Berechenbarkeit verstehen will, wenn man von einer speziellen Funktion nachweisen möchte, daß sie nicht oder höchstens mit großem Aufwand berechenbar ist.

Daß eine spezielle formal definierte Klasse K von Funktionen gleich der Klasse aller im intuitiven Sinn berechenbaren Funktionen ist, kann man nicht beweisen, weil Berechenbarkeit im intuitiven Sinn kein formaler Begriff ist. Man kann eine solche Aussage höchstens widerlegen, nämlich durch Angabe einer Funktion f, eines Verfahrens zur Berechnung von f und eines Beweises, daß f nicht in der Klasse K liegt.

Wir werden also eine Charakterisierung der berechenbaren Funktionen nicht als adäquat beweisen, sondern höchstens, gestützt auf gewisse Evidenz und Erfahrung, als adäquat ansehen können. Ein

wichtiges Kriterium, das die Klasse K erfüllen sollte, ist, daß jede Funktion, die auf einer elektronischen Rechenanlage berechnet werden kann, in K liegt.

1.1.1. Effektive Rechenverfahren

Bevor wir versuchen, die berechenbaren Funktionen zu charakterisieren, führen wir einige Eigenschaften auf, die wir von jedem effektiven Rechenverfahren erwarten.

1) Die Rechenvorschrift soll aus einem endlichen Text bestehen.

2) Der Ablauf einer Rechnung soll schrittweise als Folge von elementaren Rechenschritten geschehen.

3) Das Verfahren soll determiniert sein, d.h. in jedem Stadium einer Rechnung soll vollständig und eindeutig bestimmt sein, welcher elementare Rechenschritt als nächster getan wird.
   Der Text

$$f(x) = \begin{cases} \text{vielleicht } x \\ \text{vielleicht auch } 2x \end{cases}$$

   ist deshalb nicht als Teil einer Rechenschrift zulässig.

4) Das Verfahren soll abgeschlossen sein, d.h. welcher Rechenschritt als nächster getan wird, soll ausschließlich von den Eingabewerten und den vorangegangenen Zwischenergebnissen der Rechnung abhängen. Der Text

$$f(x) = \begin{cases} x & \text{sonntags} \\ 2x & \text{sonst} \end{cases}$$

   ist deshalb ebenfalls nicht als Teil einer Rechenvorschrift zulässig.

5) Die Rechenverfahren sollen im Prinzip beliebig große Zahlen handhaben können.

Wir werden im folgenden zwischen Rechenverfahren einerseits und den durch sie berechneten Funktionen andererseits zu unterscheiden haben. Zum Beispiel sind

Verfahren 1: "Multipliziere die Eingabe x mit 6;
teile dann das Ergebnis durch 3"

und

Verfahren 2: "Multipliziere die Eingabe x mit 2"

beides Verfahren zur Berechnung der Funktion f(x)=2x. Im allgemeinen gibt es zu einer berechenbaren Funktion viele Verfahren, die sie berechnen. Ein weiteres Beispiel ist

$$f(x) = \begin{cases} 1 & \text{falls Fermat's Vermutung richtig ist} \\ 0 & \text{sonst} \end{cases}$$

Wir haben die Funktion f nicht durch ein Verfahren definiert. Um den Wert von f an irgendeiner Stelle x zu bestimmen, muß man ein berühmtes offenes Problem der Zahlentheorie lösen. Dennoch können wir schließen, daß f sogar eine berechenbare Funktion ist. Denn f ist entweder die konstante Funktion 0 oder die konstante Funktion 1 (und z.Zt. weiß niemand, welche von beiden), aber jede dieser beiden Funktionen ist berechenbar. In diesem Schluß haben wir das "Prinzip vom ausgeschlossenen Dritten" benutzt. Wir werden auch künftig auf nichtkonstruktive Beweismethoden zurückgreifen.

1.1.2. Die Existenz nichtberechenbarer Funktionen

Obwohl wir bisher nur einige Eigenschaften genannt haben, die wir von jedem Rechenverfahren verlangen, können wir jetzt schon ein Argument für die Existenz nichtberechenbarer Funktionen $f:\mathbb{N}\to\mathbb{N}$ geben. $\mathbb{N}=\{0,1,2,\ldots\}$ bezeichnet hier wie im folgenden immer die Menge der natürlichen Zahlen einschließlich der Null. Zuvor einige Definitionen und Notationen:
Einen endlichen Zeichenvorrat nennt man auch Alphabet. Ist A ein Alphabet, dann bezeichnet A^* die Menge aller endlichen Zeichenreihen, die man mit den Zeichen aus A bilden kann (dabei dürfen gleiche Zeichen aus A beliebig oft in einer Zeichenreihe vorkommen). Zeichenreihen nennt man manchmal auch Worte. Man betrachtet üblicherweise auch die eindeutige Zeichenreihe, die aus überhaupt keinem Zeichen besteht, als Element von A^* und nennt diese Zeichenreihe das leere Wort. Die Menge aller Zeichenreihen aus A^* ohne das leere Wort bezeichnet man mit A^+.

Zwei Mengen M,M' heißen genau dann gleich mächtig, wenn es eine bijektive Abbildung $b:M\to M'$ gibt. Sind M und $\mathbb{N}$ gleich mächtig, nennt man M abzählbar. Ist A ein endlicher Zeichenvorrat, dann ist A^* abzählbar.

Satz 1.1: Es gibt nichtberechenbare Funktionen $f:\mathbb{N}\to\{0,1\}$.

Beweis: Sei A ein endlicher Zeichenvorrat, so daß man jedes Rechenverfahren durch eine endliche Zeichenreihe, die nur aus Zeichen aus A besteht, beschreiben kann. Für A wähle man etwa die großen und kleinen Buchstaben des deutschen Alphabets, die Ziffern von 0 bis 9, die Interpunktionszeichen, die in mathematischen Texten benutzten Sonderzeichen und ein Zeichen für den Zwischenraum. Dann kann man sogar jedes mathematische Buch als Zeichenreihe mit Zeichen aus A auffassen.

Da die Beschreibungen von Rechenverfahren alle in A* liegen, gibt es höchstens abzählbar viele verschiedene Rechenverfahren. Wegen der Determiniertheit der Rechenverfahren berechnet jedes Verfahren nur eine Funktion. Folglich gibt es höchstens abzählbar viele berechenbare Funktionen. Die Menge aller Funktionen $f:\mathbb{N}\to\{0,1\}$ hat jedoch die Mächtigkeit des Kontinuums, ist also überzählbar. Hieraus folgt die Behauptung. □

In dem obigen Beweis haben wir den Cantor'schen Satz von der Überabzählbarkeit des Kontinuums benutzt. Wir wiederholen nun im wesentlichen den gleichen Beweis, aber ohne den Cantor'schen Satz zu benutzen, so daß das entscheidende Argument, das eben verborgen blieb, offenbar wird:
Ist A endlich, dann kann man $\mathbb{N}$ surjektiv auf A* abbilden. Da alle Beschreibungen von Rechenverfahren in A* liegen und jedes Rechenverfahren nur eine Funktion berechnet, kann man also $\mathbb{N}$ surjektiv auf die Menge der berechenbaren Funktionen abbilden. Würde diese Menge die Menge aller Funktionen $f:\mathbb{N}\to\{0,1\}$ umfassen, dann gäbe es eine surjektive Abbildung

$$s:\mathbb{N}\to\{f \mid f:\mathbb{N}\to\{0,1\}\},$$

also

$$\{f \mid f:\mathbb{N}\to\{0,1\}\} = \{s(i) \mid i\in\mathbb{N}\}.$$

Um einen Widerspruch herzuleiten, genügt es nun, eine Funktion $g:\mathbb{N}\to\{0,1\}$ anzugeben, die von jeder der Funktionen $s(i)$ verschieden ist. Wir setzen $f_i=s(i)$ und definieren g so, daß sich g von f_i gerade am Argument i unterscheidet:

$$g(i) = \begin{cases} 1 & \text{falls } f_i(i) = 0 \\ 0 & \text{falls } f_i(i) = 1. \end{cases}$$

Wäre g etwa gleich f_j, dann wäre $g(j)=f_j(j)$, aber nach Definition von g ist $g(j)\neq f_j(j)$. Die Funktion g ist also in der Tat von allen

f_j verschieden. □

Betrachtet man die unendliche Matrix, die in Zeile i und Spalte j den Wert $f_i(j)$ enthält, dann wurde g gerade durch die Werte $f_i(i)$ auf der Hauptdiagonalen dieser Matrix definiert. Die oben benutzte Methode, die Funktion g zu konstruieren, und viele ähnliche Methoden, nennt man deshalb Diagonalisierungsverfahren, obwohl die Tatsache, daß man g gerade durch die Werte auf der Diagonalen definiert hat, hierbei gar nicht wesentlich ist. Es genügt, g so zu definieren, daß sich g von jedem f_i an irgendeiner Stelle - nicht notwendig gerade an der Stelle i - unterscheidet. Wir werden im folgenden noch oft Diagonalisierungsverfahren verwenden.

Nachdem wir nun wissen, daß es nichtberechenbare Funktionen gibt, wäre es wünschenswert, von speziellen, interessanten Funktionen nachzuweisen, daß sie nicht berechenbar sind. Wir werden das tun, sobald wir zu einer adäquaten formalen Charakterisierung der im intuitiven Sinn berechenbaren Funktionen gelangt sind.

Übungen

1. a) Man gebe eine surjektive Funktion
$$f:\mathbb{N}\to\{1,2,\ldots,9\}^*$$
an.

   b) Man zeige: Ist A endlich, dann ist A^* abzählbar.

2. Man zeige: Ist $F=\{f_1,f_2,\ldots\}$ eine abzählbare Menge von Funktionen $f_i:\mathbb{N}\to\{0,1\}$, dann gibt es Funktionen $g,h:\mathbb{N}\to\{0,1\}$, so daß
   a) $g(n) = 1$ für alle geraden n und $g\notin F$
   b) $h(n) = 1$ für alle n, die keine Quadratzahlen sind, und $h\notin F$.

1.2. Die μ-rekursiven Funktionen

1.2.1. Die Klasse der primitiv rekursiven Funktionen

Der historisch erste Ansatz, den Begriff Berechenbarkeit zu formalisieren, beruhte auf folgender Idee:
Man gibt zunächst einige ganz einfache Grundfunktionen an, die im intuitiven Sinn berechenbar sind. Dann gibt man zwei Prozesse an, mit denen man aus bekannten Funktionen neue gewinnt. Die in diesem Sinn berechenbaren Funktionen sind diejenigen, die man durch end-

lich viele Anwendungen der Erzeugungsprozesse aus den Grundfunktionen gewinnen kann.

Definition 1.2: Die Klasse der primitiv rekursiven Funktionen ist die kleinste Klasse $\mathbb{P}$ (der Durchschnitt aller Klassen $\mathbb{P}$) von Funktionen, für die gilt:

1) für alle $r,s\in\mathbb{N}$ ist die konstante Funktion
$c_s^r:\mathbb{N}^r\to\mathbb{N}$, $c_s^r(x) = s$ in $\mathbb{P}$

2) Die Nachfolgerfunktion $N:\mathbb{N}\to\mathbb{N}$, $N(x) = x+1$ ist in $\mathbb{P}$

3) Für alle $r\in\mathbb{N}$ und $i\leq r$ ist die Projektion
$p_i^r: \mathbb{N}^r\to\mathbb{N}$, $p_i^r(x_1,\ldots,x_r) = x_i$ in $\mathbb{P}$

4) Sind $f:\mathbb{N}^r\to\mathbb{N}$ und $g_1,\ldots,g_r:\mathbb{N}^m\to\mathbb{N}$ in $\mathbb{P}$, dann ist auch $h:\mathbb{N}^m\to\mathbb{N}, h(x)=f(g_1(x),\ldots,g_r(x))$ in $\mathbb{P}$. Man sagt, h entsteht durch Einsetzung oder Substitution der g_i in f.

5) Sind $g:\mathbb{N}^r\to\mathbb{N}$ und $h:\mathbb{N}^{r+2}\to\mathbb{N}$ in $\mathbb{P}$, dann ist auch jede Funktion $f:\mathbb{N}^{r+1}\to\mathbb{N}$ in $\mathbb{P}$, welche die folgenden Gleichungen erfüllt:

(5.1) $f(0,x) = g(x)$

(5.2) $f(n+1,x) = h(n,f(n,x),x)$ $n\in\mathbb{N}, x\in\mathbb{N}$.

Man sagt, f entsteht aus g und h durch primitive Rekursion.

Entsteht f durch primitive Rekursion aus g und h, dann ist f eindeutig bestimmt. Denn sind etwa f_1 und f_2 zwei Funktionen, die (5.1) und (5.2) erfüllen, dann folgt aus (5.1): $f_1(0,x)=g(x)=f_2(0,x)$ und aus (5.2) durch Induktion über n: $f_1(n+1,x)=h(n,f_1(n,x),x)=h(n,f_2(n,x),x)=f_2(n+1,x)$.

Eine Funktion f ist genau dann primitiv rekursiv, wenn es eine endliche Folge von Funktionen $f_1,\ldots,f_s$ gibt mit folgender Eigenschaft: $f=f_s$ und für alle $i\leq s$ ist f_i entweder wegen 1), 2) oder 3) in $\mathbb{P}$, oder f_i entsteht durch Einsetzung oder primitive Rekursion direkt aus einigen $f_j, j<i$. Sei nämlich D die Klasse aller Funktionen, für die eine solche Folge $f_1,\ldots,f_s$ existiert. D ist in jeder Klasse, die 1) bis 5) erfüllt, enthalten, und D erfüllt selbst 1) bis 5). Also ist D die kleinste Klasse, die 1) bis 5) erfüllt.

Die Folge $f_1,\ldots,f_s$ nennt man eine Ableitung von f.

Beispiel 1.3:
$$f_1(x) = p_1^1(x)$$
$$f_2(x) = N(x)$$
$$f_3(0,y) = f_1(y)$$
$$f_3(x+1,y) = f_2(f_3(x,y))$$

ist eine Ableitung für $f_3(x,y)=x+y$.

Die Funktionen aus $\mathbb{P}$ sind im intuitiven Sinn berechenbar. Sei etwa $f_1,\ldots,f_s$ eine Ableitung der Funktion f. Wir zeigen (informal) durch Induktion über i, daß man zu jeder der Funktionen f_i ein mechanisches Verfahren finden kann, das f_i auswertet.

Ist f_i in $\mathbb{P}$ wegen 1), 2) oder 3), so kann man leicht ein mechanisches Verfahren zur Auswertung von f_i angeben.

Ist $f_i(x)=f_{i_1}(f_{i_2}(x),\ldots,f_{i_r}(x)),i_1,\ldots,i_r<i$, dann wertet man $f_i(x)$ aus, indem man zuerst $f_{i_2}(x)$, dann $f_{i_3}(x)$ usw. .... bis $f_{i_r}(x)$ mittels der per Induktionsannahme bereits bekannten Verfahren für $f_{i_2},\ldots,f_{i_r}$ auswertet. Das liefert Ergebnisse $y_2,\ldots,y_r$. Schließlich wertet man $f_{i_1}(y_2,\ldots,y_r)$ mit dem per Induktionsannahme bekannten Verfahren für f_{i_1} aus. Damit hat man den Ausdruck

$$f_{i_1}(f_{i_2}(x),\ldots,f_{i_r}(x))$$

"von innen nach außen und von links nach rechts" ausgewertet. Entsteht $f=f_i$ durch primitive Rekursion aus $g=f_{i_1}$ und $h=f_{i_2},i_1,i_2<i$. Dann wertet man den Ausdruck f(n,x) wie folgt aus: Man ersetzt f(n,x) durch h(n-1,f(n-1,x),x). In diesem Ausdruck ersetzt man f(n-1,x) durch h(n-2,f(n-2,x),x) usw. Da das erste Argument von f bei jedem Schritt um eins kleiner wird, bricht die Folge dieser Ersetzungen nach n Schritten bei f(0,x) ab. Dies wird durch g(x) ersetzt. Man hat damit einen im allgemeinen ziemlich langen Ausdruck, in dem g und h, aber nicht mehr f vorkommen. Mittels der per Induktionsannahme bereits bekannten Ver-

fahren für g und h wertet man nun diesen Ausdruck von innen nach außen und von links nach rechts aus.

Dem folgenden Beispiel liegt die Ableitung aus Beispiel 1.3 zugrunde.

Beispiel 1.4:

$$f_3(2,4) = f_2(f_3(1,4)) = f_2(f_2(f_3(0,4)) = f_2(f_2(f_1(4)) =$$
$$f_2(f_2(p_1^1(4)) = f_2(f_2(4)) = f_2(N(4)) = f_2(5) = N(5) = 6.$$

Eine Behauptung wie "$\mathbb{P}$ ist gleich der Klasse der im intuitiven Sinn berechenbaren Funktionen" ist, wie schon früher erwähnt, höchstens widerlegbar. In der Tat gab Ackermann [1] eine Funktion an, die im intuitiven Sinn berechenbar ist, aber nicht in $\mathbb{P}$ liegt (Übung 7). Wir werden später einen weiteren Erzeugungsprozeß angeben, der zusammen mit 1) bis 5) eine hinreichend weite Klasse von berechenbaren Funktionen liefert. Andererseits zeigt sich, daß die Klasse $\mathbb{P}$ die Menge aller Funktionen, die mit vertretbarem Aufwand berechenbar sind, weit umfaßt, selbst wenn man einen Aufwand von 2^{2^n} Rechenschritten zum Auswerten einer Funktion an der Stelle n noch für vertretbar hielte.
Wir studieren nun die Klasse $\mathbb{P}$ näher.

1.2.2. Einige primitiv rekursive Funktionen

Wir haben im vorigen Abschnitt gesehen, daß x+y primitiv rekursiv ist.

$x \cdot y$ ist primitiv rekursiv,

denn

$$0 \cdot y = c_0^1(y),$$
$$(x+1)y = x + x \cdot y .$$

Wir schreiben in Zukunft statt $c_s^r(y)$ auch einfach s. Die modifizierte Differenz

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

ist primitiv rekursiv. Sei zunächst $u(x)=x \dot{-} 1$.

u ist in $\mathbb{P}$ wegen

$$u(0) = 0$$
$$u(x+1) = p_1^1(x).$$

Wir schreiben in Zukunft statt $p_1^1(x)$ auch einfach x.
Nun ist

$$x \dot{-} 0 = x$$
$$x \dot{-} (y+1) = u(x \dot{-} y).$$

In dieser Definition läuft die Rekursion über das zweite Argument von $\dot{-}$, was in der Definition des primitiven Rekursionsschemas nicht vorgesehen ist. Hier wie auch später wird die folgende Regel benutzt:
Ist $(i_1,\ldots,i_r)$ eine Permutation von $(1,\ldots,r)$ und ist $f:\mathbb{N}^r \to \mathbb{N}$ primitiv rekursiv, dann auch $f':\mathbb{N}^r \to \mathbb{N}$, definiert durch

$$f'(x_1,\ldots,x_r) = f(x_{i_1},\ldots,x_{i_r}), x_i \in \mathbb{N},$$

denn

$$f'(x) = f(p^r_{i_1}(x),\ldots,p^r_{i_r}(x)), x \in \mathbb{N}^r.$$

x^y ist primtiv rekursiv, denn

$$x^0 = 1$$
$$x^{y+1} = x^y \cdot x .$$

Die Vorzeichenfunktion

$$sg(x) = \begin{cases} 1 & \text{falls } x>0 \\ 0 & \text{sonst} \end{cases}$$

ist primitiv rekursiv, denn

$$sg(0) = 0$$
$$sg(x+1) = 1.$$

Eine andere Ableitung von sg ist:

$$\overline{sg}(x) = 1 \dot{-} x$$
$$sg(x) = \overline{sg}(\overline{sg}(x)).$$

1.2.3. Primitiv rekursive Prädikate

Ein Prädikat auf $\mathbb{N}^r$ ist eine Aussageform, in der r freie*) Variablen $x_1,\ldots,x_r$ vorkommen und die je nach Belegung der x_i mit Zahlen aus $\mathbb{N}$ zu einer wahren oder falschen Aussage wird.

Zum Beispiel sind

$$x \leq y,\ x|y \text{ ("x teilt y")},\ 2x<y$$

*) Man unterscheidet zwischen gebundenen und freien Variablen. Die gebundenen Variablen sind die, über die quantifiziert wird.

Prädikate über $\mathbb{N}^2$. $(\forall y)(y \geq x)$ ist ein Prädikat über $\mathbb{N}$ und ist wahr genau dann, wenn $x=0$.

Für "P(x) ist wahr" schreibt man auch "P(x) gilt" oder einfach "P(x)".

Man ordnet jedem Prädikat P(x) über $\mathbb{N}^r$ auf naheliegende Weise eine Funktion $C_P: \mathbb{N}^r \to \mathbb{N}$ zu durch

$$C_P(x) = \begin{cases} 1 & \text{falls } P(x) \\ 0 & \text{sonst} \end{cases}$$

und nennt P ein <u>primitiv rekursives Prädikat</u>, wenn C_P eine primitiv rekursive Funktion ist.

Das Prädikat $x<y$ ist primitiv rekursiv, denn

$$C_<(x,y) = sg(y \dot{-} x).$$

Ist P(x) ein primitiv rekursives Prädikat auf $\mathbb{N}^r$ und sind $g_1,\ldots,g_r$ $\mathbb{N}^m \to \mathbb{N}$ primitiv rekursiv, dann ist auch

$$Q(x) = P(g_1(x),\ldots,g_r(x))$$

ein primitiv rekursives Prädikat. Denn $C_Q(x)=C_P(g_1(x),\ldots,g_r(x))$. Also ist z.B. auch $2x<y$ ein primitiv rekursives Prädikat.

Aus Prädikaten P und Q kann man die neuen Prädikate $P \wedge Q$, $P \vee Q$ und $\sim_P$ bilden, wobei $P \wedge Q$ gilt, wenn P und Q beide gelten, $P \vee Q$ gilt, wenn entweder P oder Q oder beide gelten, und $\sim P$ gilt genau dann, wenn P falsch ist.

<u>Lemma 1.5:</u> Sind P und Q primitiv rekursive Prädikate, dann auch $P \wedge Q$, $P \vee Q$ und $\sim P$.

<u>Beweis:</u>

$$C_{P \wedge Q}(x) = C_P(x) \cdot C_Q(x),$$
$$C_{P \vee Q}(x) = sg(C_P(x)+C_Q(x)),$$
$$C_{\sim P}(x) = \overline{sg}(C_P(x)).$$

Wegen

$$x = y \iff \sim((x<y) \vee (y<x)) \quad \text{und}$$
$$x \leq y \iff (x<y) \vee (x=y)$$

sind deshalb auch die Prädikate "=" und "$\leq$" primitiv rekursiv. Prädikate kann man benutzen, um Funktionen durch Fallunterscheidung zu definieren.

<u>Satz 1.6:</u> Sind die Prädikate $P_1,\ldots,P_k$ auf $\mathbb{N}^r$ und die Funktionen $f_1,\ldots,f_k: \mathbb{N}^r \to \mathbb{N}$ primitiv rekursiv und ist für jedes $x \in \mathbb{N}^r$

höchstens ein $P_i(x)$ wahr, dann ist auch die folgende Funktion $g(x):\mathbb{N}^r \to \mathbb{N}$ primitiv rekursiv:

$$g(x) = \begin{cases} f_1(x) & \text{falls } P_1(x) \\ \vdots & \\ f_k(x) & \text{falls } P_k(x) \\ 0 & \text{sonst} \end{cases}$$

Beweis: $g(x) = C_{P_1}(x) \cdot f_1(x) + \ldots + C_{P_k}(x) f_k(x)$.

Als wichtige Folgerung ergibt sich

Korollar 1.7: Gilt für $f:\mathbb{N}^r \to \mathbb{N}$ nur an endlich vielen Stellen $f(x) \neq 0$, dann ist f primitiv rekursiv.

1.2.4. Der beschränkte μ-Operator

Abbildungen, die Funktionen in Funktionen abbilden, heißen Operatoren. Im folgenden erklären wir den Operator μ_b.

Definition 1.8: Für $f:\mathbb{N}^{r+1} \to \mathbb{N}$ wird die Funktion $\mu_b f:\mathbb{N}^{r+1} \to \mathbb{N}$ erklärt durch

$$\mu_b f(n,x) = \begin{cases} \min\{m \mid m \leq n \wedge f(m,x)=0\} & \text{falls dieses existiert} \\ 0 & \text{sonst;} \end{cases} \qquad n,m \in \mathbb{N}, x \in \mathbb{N}^r.$$

μ_b heißt beschränkter μ-Operator.

Satz 1.9: Ist $f:\mathbb{N}^{r+1} \to \mathbb{N}$ primitiv rekursiv, dann auch $\mu_b f$.

Beweis: $\mu_b f(0,x) = 0$.

$$\mu_b f(n+1,x) = \begin{cases} \mu_b f(n,x) & \text{falls } \mu_b f(n,x) \neq 0 \\ n+1 & \text{falls } \mu_b f(n,x)=0 \wedge f(0,x) \neq 0 \wedge f(n+1,x)=0 \\ 0 & \text{sonst} \end{cases}$$

Insbesondere folgt

Korollar 1.1o: Ist P ein primitiv rekursives Prädikat auf $\mathbb{N}^{r+1}$, dann ist die folgende Funktion $g:\mathbb{N}^{r+1} \to \mathbb{N}$ primitiv rekursiv:

$$g(n,x) = \begin{cases} \min\{m \mid m \leq n \wedge P(m,x)\} & \text{falls dieses existiert} \\ 0 & \text{sonst;} \end{cases} \qquad n,m \in \mathbb{N}, x \in \mathbb{N}^r.$$

Beweis: $g(n,x) = \mu_b C_{\sim P}(n,x)$.

Nun kann gezeigt werden, daß die folgenden Funktionen primitiv rekursiv sind:

i) x/y = ganzzahliger Anteil bei Division von x durch y

$$x/y = \begin{cases} \text{ganzzahliger Anteil bei Division von } x \text{ durch } y & \text{falls } y \neq 0 \\ 0 & \text{sonst} \end{cases}$$

denn

$$x/y = \begin{cases} \min\{m \mid m \leq x \wedge y(m+1) > x\} & \text{falls } y \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

ii) Rest (x,y) = Rest bei ganzzahliger Division von x durch y

denn

$$\text{Rest}\ (x,y) = x \dot{-} y \cdot (x/y).$$

iii) Das Prädikat $x|y$ (x teilt y)

denn

$$x|y \iff \text{Rest}\ (x,y) = 0.$$

1.2.5. Primitiv rekursive Bijektionen

Es ist oft nützlich, Tupel von natürlichen Zahlen als eine einzige natürliche Zahl kodieren und auch wieder dekodieren zu können. Das ist mit primitiv rekursiven Funktionen möglich.

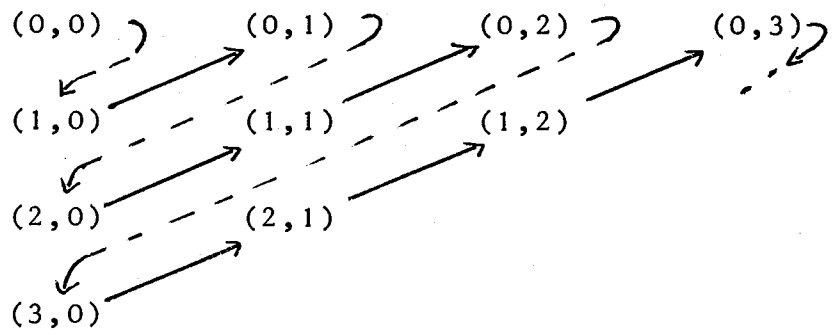
Satz 1.11: Für alle $k \in \mathbb{N}$ gibt es primitiv rekursive Funktionen $b^{(k)}: \mathbb{N}^k \to \mathbb{N}$ und $b_i^{(k)}: \mathbb{N} \to \mathbb{N}$, $(1 \leq i \leq k)$, so daß gilt:

i) b^k ist bijektiv

ii) $b_i^{(k)}(b^{(k)}(x_1,\ldots,x_k)) = x_i$.

Beweis durch Induktion über k.

Für k=2 bildet man $\mathbb{N}^2$ wie folgt auf $\mathbb{N}$ ab:



(0,0) wird auf 0 abgebildet, (1,0) auf 1, (0,1) auf 2, usw.

Allgemein:

$$b^{(2)}(n,m) = \sum_{i=1}^{n+m} i+m$$
$$= \frac{1}{2}(n+m)(n+m+1)+m$$

Für $s(x)=\max\{i \mid i\leq x, \frac{1}{2} i(i+1)\leq x\}$ gilt $s(b^{(2)}(n,m))=n+m$.

Sei
$$b_1^{(2)}(x) = x - \frac{1}{2} s(x)(s(x)+1)$$
$$b_1^{(2)}(x) = s(x)-b_2^{(2)}(x).$$

Durch Einsetzen von $b(n,m)$ für x verifiziert man sofort, daß b_1 und b_2 das Gewünschte leisten.

Für $k\geq 3$ definiert man die Funktionen $b^{(k)}$ und $b_i^{(k)}$ aus den Funktionen $b^{(k-1)}$ und $b_i^{(k-1)}$ wie folgt:

$$b^{(k)}(x_1,\ldots,x_k) = b^{(2)}(x_1, b^{(k-1)}(x_2,\ldots,x_k))$$
$$b_1^{(k)}(x) = b_1^{(2)}(x)$$
$$b_i^{(k)}(x) = b_{i-1}^{(k-1)}(b_2^{(2)}(x)) \text{ für } 2\leq i\leq k.$$

1.2.6. Der μ-Operator

Wir zeigen, daß die Klasse der primitiv rekursiven Funktionen nicht alle berechenbaren Funktionen umfaßt.

Satz 1.12: Es gibt Funktionen, die im intuitiven Sinn berechenbar, aber nicht primitiv rekursiv sind.

Beweis: Wir beschreiben informal eine Funktion, die offenbar im intuitiven Sinn berechenbar ist, aber nicht primitiv rekursiv sein kann (siehe auch Übung 7).

Sei A ein Zeichenvorrat, so daß man alle Ableitungen primitiv rekursiver Funktionen in A* darstellen kann. Beispielsweise genügen für A die Zeichen c,p,N,f,=,(,),+,1,x, hochgestellte und tiefgestellte Ziffern von 0 bis 9 zum Indizieren von Funktionssymbolen c,p,f und von Variablen x sowie das Komma und der Zwischenraum. Von jeder Zeichenreihe $w\in A^*$ kann man effektiv, d.h. durch ein mechanisches Rechenverfahren entscheiden, ob w Ableitung einer primitiv rekursiven Funktion $f:\mathbb{N}\to\mathbb{N}$ ist. Man kann ebenfalls mechanisch alle Zeichenreihen aus A^+ aufzählen, indem man erst alle Zeichenreihen der Länge 1, dann alle Zeichenreihen der Länge 2,3 usw. generiert.

Sei G_i die i-te Ableitung, die in diesem Prozeß aufgezählt wird. Zu vorgegebenem i kann man G_i effektiv berechnen, indem man beginnt, die Zeichenreihen $w \in A^+$ aufzuzählen, zu jeder Zeichenreihe w entscheidet, ob w eine Ableitung ist, und dies so lange fortgesetzt, bis man die i-te Ableitung gefunden hat.

Sei g_i die durch G_i definierte primitiv rekursive Funktion. Dann ist $h:\mathbb{N}\to\mathbb{N}$, definiert durch $h(i)=g_i(i)+1$ eine berechenbare Funktion. Denn wir haben eben gesehen, wie man zu i die Ableitung G_i findet, und wir haben zu Beginn dieses Abschnittes beschrieben, wie man eine primitiv rekursive Funktion, die durch ihre Ableitung gegeben ist, auswertet.

Wäre h primitiv rekursiv, müßte es eine Ableitung H für h geben sowie einen Index j, so daß $H=G_j$. Das führt aber zu dem Widerspruch

$$g_j(j) = h(j) = g_j(j)+1.$$

Das eben beschriebene Verfahren, die Funktion h zu konstruieren, hat nur zwei Eigenschaften der primitiv rekursiven Funktionen benutzt, nämlich daß alle "Programme" (Ableitungen) dieser Funktionen effektiv generiert und auf einheitliche Weise ausgewertet werden können. Der Widerspruch: "$g_j(j)$ existiert und ist von $g_j(j)$ verschieden" konnte herbeigeführt werden, weil alle primitiv rekursiven Funktionen an allen Argumenten definiert sind.

Ist eine Funktion an allen Argumenten definiert, nennen wir sie total, andernfalls partiell.

Wir müssen nach dem eben Gesagten erwarten, daß man die Klasse aller totalen berechenbaren Funktionen nicht konstruktiv charakterisieren kann, d.h. man wird die Programme aller totalen berechenbaren Funktionen entweder nicht effektiv aufzählen oder nicht in einheitlicher Weise auswerten können. Andernfalls würde die Existenz einer totalen berechenbaren Diagonalfunktion h zu einem Widerspruch führen.

Es gelingt jedoch, die Klasse aller berechenbaren Funktionen konstruktiv zu charakterisieren, indem man auch partielle berechenbare Funktionen in diese Klasse aufnimmt. Diese Funktionen werden von Programmen berechnet, die bei einigen Eingaben in eine unendliche Schleife geraten, d.h. immer wieder die gleiche Folge von elementaren Rechenschritten durchlaufen und nie ein Ergebnis

liefern. Die Diagonalfunktion h ist dann auch partiell berechenbar, aber an den kritischen Stellen j nicht definiert.

Wir fügen nun den Erzeugungsprozessen für die primitiv rekursiven Funktionen einen weiteren hinzu, der auch partielle Funktionen erzeugt.

Definition 1.13: Für $f:\mathbb{N}^{r+1}\to\mathbb{N}$ ist $\mu f:\mathbb{N}^r\to\mathbb{N}$ definiert durch:

$$\mu f(x) = \begin{cases} \min\{n \mid f(n,x)=0 \text{ und für alle } m\leq n \text{ ist } f(m,x) \text{ definiert}\} \\ \qquad\qquad \text{falls ein solches } n \text{ existiert} \\ \text{nicht definiert} \qquad\qquad \text{sonst} \end{cases}$$

Ist f berechenbar (und eventuell partiell), dann kann man aus einem Verfahren M für f ein Verfahren für μf gewinnen, indem man bei Eingabe $x\in\mathbb{N}^r$ das Verfahren M erst auf (0,x), dann auf (1,x), (2,x) usw. ansetzt, bis man ein n mit f(n,x) gefunden hat. Gibt es kein solches n, oder gibt es ein m, so daß $m<\min\{n \mid f(n,x)=0\}$ und f(m,x) ist nicht definiert, so bricht das Verfahren M angesetzt auf (m,x) nicht ab, und damit bricht auch das gesamte Verfahren nicht ab.

Wir erklären nun die Klasse der μ-rekursiven Funktionen. Im Vergleich zur Definition der primitiv rekursiven Funktionen sind neben der Einführung des μ-Operators einige weitere Ergänzungen nötig, um festzulegen, an welchen Stellen die Funktionen definiert sind.

Definition 1.14: Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse R von Funktionen, für die gilt:

1) Für alle $r,s\in\mathbb{N}$ ist $c_s^r:\mathbb{N}^r\to\mathbb{N}, c_s^r(x)=s$ in R.

2) $N:\mathbb{N}\to\mathbb{N}, N(x)=x+1$ ist in R.

3) Für alle $r\in\mathbb{N}$ und $i\leq r$ ist $p_i^r:\mathbb{N}^r\to\mathbb{N}, p_i^r(x_1,\ldots,x_r)=x_i$ in R.

4) Sind $f:\mathbb{N}^r\to\mathbb{N}$ und $g_1,\ldots,g_r:\mathbb{N}^m\to\mathbb{N}$ in R, dann ist auch $h:\mathbb{N}^m\to\mathbb{N}, h(x)=f(g_1(x),\ldots,g_r(x))$ in R. Dabei ist h(x) genau dann definiert, wenn alle $g_i(x)$ und $f(g_1(x),\ldots,g_r(x))$ definiert sind.

5) Sind $g:\mathbb{N}^r\to\mathbb{N}$ und $h:\mathbb{N}^{r+2}\to\mathbb{N}$ in R, dann ist auch die Funktion $f:\mathbb{N}^{r+1}\to\mathbb{N}$ in R, welche die folgenden Gleichungen erfüllt:

$$f(0,x) = g(x)$$
$$f(n+1,x) = h(n,f(n,x),x), n\in\mathbb{N},\ x\in\mathbb{N}^r.$$

Dabei ist f(0,x) genau dann definiert, wenn g(x) definiert

ist. Für $n \geq 1$ ist $f(n+1,x)$ genau dann definiert, wenn $h(n,f(n,x),x)$ und für alle $i<n$ $f(i,x)$ definiert ist.

6) Ist $f:\mathbb{N}^{r+1} \to \mathbb{N}$ in R, dann ist auch μf in R.

Jede primitiv rekursive Funktion ist offenbar auch μ-rekursiv. μc_1^1, die überall undefinierte Funktion, ist μ-rekursiv und nicht primitiv rekursiv. Jede μ-rekursive Funktion ist eine im intuitiven Sinn berechenbare, eventuell partielle Funktion.

Wir werden später begründen, warum man die Klasse der μ-rekursiven Funktionen für eine gelungene Charakterisierung der Klasse alle berechenbaren Funktionen hält.

Übungen

1. Man zeige, daß die folgenden Funktionen primitiv rekursiv sind:
   a) $X!$
   b) $|X-Y|$.

2. Man beweise Korollar 2.4.

3. Man zeige: Sind $g_o, g_1 : \mathbb{N}^r \to \mathbb{N}$ und $h : \mathbb{N}^{r+2} \to \mathbb{N}$ primitiv rekursiv, dann ist auch $f : \mathbb{N}^{r+1} \to \mathbb{N}$ definiert durch

$$\begin{aligned} f(0,x) &= g_o(x) \\ f(1,x) &= g_1(x) \\ f(n+1,x) &= h(n,f(n,x),x) \text{ für } n \geq 1 \end{aligned}$$

   primitiv rekursiv.

4. Für $f:\mathbb{N}^{r+1} \to \mathbb{N}$ ist $(\max_b f) : \mathbb{N}^{r+1} \to \mathbb{N}$ definiert durch

$$(\max_b f)(a,x) = \begin{cases} \max\{m \mid m \leq a \text{ und } f(m,x)=0\} \text{ falls dieses existiert} \\ a+1 \qquad \text{sonst} \end{cases}$$

   Man zeige: Ist f primitiv rekursiv, dann auch $(\max_b f)$

5. Simultane Rekursion.
   Sind $g_o, g_1 : \mathbb{N}^r \to \mathbb{N}$ und $h_o, h_1 : \mathbb{N}^{r+3} \to \mathbb{N}$ primitiv rekursiv, dann sind auch

$$f_o, f_1 : \mathbb{N}^{r+1} \to \mathbb{N}$$

   definiert durch

$$\begin{aligned} f_i(0,x) &= g_i(x) \\ f_i(n+1,x) &= h_i(n,f_1(n,x),f_2(n,x),x) \qquad i \in \{1,2\}, n \in \mathbb{N}, x \in \mathbb{N}^r \end{aligned}$$

primitiv rekursiv.

Hinweis: Man verwende Satz 1.11.

6. Sind $g_o, g_1 : \mathbb{N}^r \to \mathbb{N}$ und $h : \mathbb{N}^{r+3} \to \mathbb{N}$ primitiv rekursiv, dann ist auch $f : \mathbb{N}^{r+1} \to \mathbb{N}$ definiert durch

$$\begin{aligned} f(0,x) &= g_o(x) \\ f(1,x) &= g_1(x) \\ f(n+1,x) &= h(n,f(n,x),f(n-1,x),x) \text{ für } n \geq 1 \end{aligned}$$

primitiv rekursiv.

7. $A : \mathbb{N}^2 \to \mathbb{N}$ sei definiert durch

$$\begin{aligned} A(0,y) &= y+1 \\ A(x+1,0) &= A(x,1) \\ A(x+1,y+1) &= A(x,A(x+1,y)) \quad \text{(Ackermannfunktion).} \end{aligned}$$

Man zeige:

a) Für $A_x(y) = A(x,y)$ gilt:

α) $A_1(y) = y+2$

β) $A_2(y) > 2y$

γ) $A_3(y) > 2^{y+1}$

b) α) $y < A(x,y)$

β) $A(x,y) < A(x+1,y)$

γ) $A(x,y) < A(x,y+1)$

δ) $A(x,y+1) \leq A(x+1,y)$

ε) $A(x,2y) < A(x+3,y)$

c) Für $z \in \mathbb{N}^r$ sei $|z| = \sum_{i=1}^{r} z_i$. Für $m \in \mathbb{N}$ sei

$B_m = \{f \mid f \text{ primitiv rekursiv und für alle } z \text{ gilt } f(|z|) < A(m,|z|)\}$

α) $c_s^r \in B_s$

β) $p_i^r \in B_o$

γ) $N \in B_1$

d) Sind $f,g_1,\ldots,g_r \in B_m$ und entsteht h durch Einsetzung der g_i in f, dann ist $h \in B_{m+\lceil \log_2 r \rceil+2}$.*)

e) Sind $g,h \in B_m$ und entsteht f aus g und h durch primitive Rekursion, dann ist $f \in B_{m+5}$.

f) Für jede primitiv rekursive Funktion f gibt es ein $m \in N$, so daß $f(|z|)<A(m,|z|)$ für alle z.

g) A ist nicht primitiv rekursiv.

1.3. Turingmaschinen

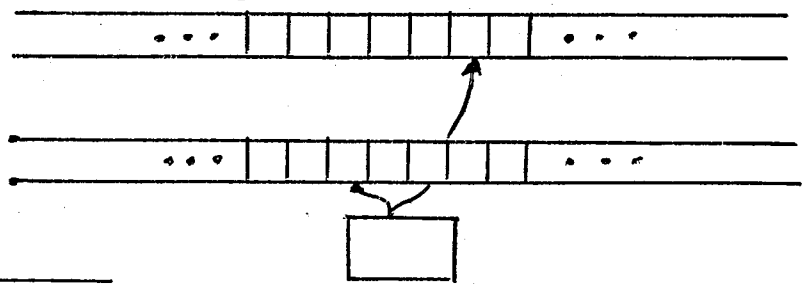
In diesem Abschnitt studieren wir einen auf den ersten Blick völlig verschiedenen Ansatz, die berechenbaren Funktionen zu charakterisieren. Es wird eine Klasse von Funktionen erklärt, die von gewissen ganz einfachen Maschinen (Turingmaschinen) berechnet werden.

1.3.1. Physikalische Beschreibung des Maschinenmodells

Eine k-Band-Turingmaschine besteht aus einer endlichen Steuereinheit und k linearen zweiseitig-unendlichen Bändern. Auf jedem Band operiert ein Lese-Schreib-Kopf, der mit der Steuereinheit verbunden ist.

Die Steuereinheit befindet sich zu jedem Zeitpunkt in einem von endlich vielen möglichen Zuständen. Jedes Band ist in Felder eingeteilt. Jedes Feld kann eines von endlich vielen Zeichen speichern. Die Menge dieser Zeichen heißt Bandalphabet. Jeder Lese-Schreib-Kopf steht zu jedem Zeitpunkt auf genau einem Feld seines Bandes. Er kann erkennen, welches Zeichen auf dem Feld steht, und es evtl. durch ein anderes Zeichen aus dem Bandalphabet überdrucken.

2-Band-Turingmaschine



*) $\lceil X \rceil$ ist die kleinste ganze Zahl größer oder gleich x,
$\lfloor X \rfloor$ ist die größte ganze Zahl kleiner oder gleich x.

Die Zeichen 0,1,# sowie ein spezielles Zeichen B sollen im Bandalphabet jeder Turingmaschine sein, und zu jedem Zeitpunkt soll der Inhalt von höchstens endlich vielen Feldern eines jeden Bandes von B verschieden sein. B dient als Leersymbol. Der von B verschiedene Anteil eines Bandes heißt Bandinschrift.

In einem Rechenschritt geschieht folgendes: In Abhängigkeit vom Zustand der Steuereinheit und der k Zeichen, welche die Köpfe lesen, überdruckt jeder der Köpfe sein Feld mit einem Zeichen des Bandalphabets, bewegt sich dann evtl. ein Feld nach links oder rechts, und die Steuereinheit geht in einen neuen Zustand über. Was in Abhängigkeit von Zustand und den k gelesenen Zeichen in einem Schritt getan werden soll, kann durch eine endliche Funktion beschrieben werden. Diese Funktion heißt Übergangsfunktion.

In der Zustandsmenge gibt es einen ausgezeichneten Anfangszustand z_o und eine ausgezeichnete Menge E von Endzuständen.
Schreibt man auf jedes Band eine Zeichenreihe und startet die Maschine im Anfangszustand, dann tut die Maschine einen Schritt nach dem anderen, bis sie in einen Endzustand kommt. Wenn und falls das geschieht, hält die Maschine an. Die Inschrift der Bänder zu diesem Zeitpunkt nennt man das Resultat der Rechnung.

Ist A das Bandalphabet, dann berechnet eine k-Band-Turingmaschine M also eine im allgemeinen partielle Funktion von $(A^*)^k$ nach $(A^*)^k$.

Jede durch eine Turingmaschine berechenbare Funktion ist im intuitiven Sinn berechenbar, denn Turingmaschinen sind ja selbst mechanische Rechenverfahren.

1.3.2. Definitionen und Beispiele

Die formale Definition von Turingmaschinen ist etwas mühsam, aber in ihrer Arbeitsweise nicht schwierig. Die Mühe lohnt sich jedoch, denn gerade diese bis ins letzte Detail formalisierte Definition erlaubt es, einige der schönsten Ergebnisse in der Rekursions- und Komplexitätstheorie herzuleiten. Nach den Definitionen folgen zur Erläuterung einige Beispiele.

Um eine Turingmaschine zu spezifizieren, genügt die Angabe der Zustandsmenge, des Bandalphabets, der Übergangsfunktion, des Anfangs- und des Endzustandes.

Definition 1.15: Eine k-Band-Turingmaschine M wird spezifiziert durch ein 5-Tupel

$$M = (Z, A, \delta, z_0, E),$$

wobei gilt:

Z ist eine endliche Menge (Zustandsmenge).
A ist eine endliche Menge (Bandalphabet), $Z \cap A = \emptyset$, $\{0, 1, \#, B\} \subseteq A$.
$\delta: Z \times A^k \to Z \times A^k \times \{R, L, N\}^k$ ist eine Funktion (Übergangsfunktion; R,L,N stehen für Kopfbewegung nach rechts, links und "neutral").
$z_0 \in Z$ (Anfangszustand).
$E = \{z_e \in Z \mid \delta(z_e, a)$ ist für kein $a \in A^k$ definiert$\}$ (Menge der Endzustände).
$\delta(z, a_1, \ldots, a_k) = (z', c_1, \ldots, c_k, s_1, \ldots, s_k)$ bedeutet:
Liest die Maschine im Zustand z für $1 \leq i \leq k$ auf Band i Zeichen a_i, so druckt die Maschine für jedes i auf Band i Zeichen c_i, macht die durch s_i spezifizierte Kopfbewegung und geht in Zustand z'.

Zu jedem Zeitpunkt ist die Maschine vollständig beschrieben durch den Zustand der Steuereinheit, die Bandinschriften und die Position der Köpfe relativ zu den Bandinschriften. Ein solcher "Schnappschuß" der gesamten Maschine heißt Konfiguration.

Formal beschreibt man die Arbeitsweise einer Turingmaschine, indem man für jede Konfiguration k diejenige Konfiguration k' angibt, die aus k durch einen Schritt der Maschine hervorgeht. Da wir die formale Definition später nur für 1-Band-Turingmaschinen benötigen, führen wir sie auch nur für diesen Fall durch.

Zuvor einige Notationen: Ist A ein Alphabet und $w \in A^*$ eine Zeichenreihe, so bezeichnet $|w|$ die Länge von w, d.h. die Anzahl der Zeichen in w. Für $i \in \{1, \ldots, |w|\}$ bezeichnet w_i das i-te Zeichen in w, also $w = w_1 \ldots w_{|w|}$.

Ist eine 1-Band-Turingmaschine im Zustand z, ist w ein endlicher Teil der Bandinschrift, so daß rechts und links von w nur Leerzeichen stehen, und steht der Kopf der Maschine auf w_i, so beschreiben wir dies durch die Zeichenreihe $w_i \ldots w_{i-1} z w_i \ldots w_{|w|}$.

Definition 1.16: Ist $M = (Z, A, \delta, z_0, E)$ eine 1-Band-Turingmaschine, so heißt eine Zeichenreihe k der Form uzv mit $z \in Z$ und $u, v \in A^*$ eine Konfiguration von M. k heißt Endkonfiguration, falls $z \in E$.

Wir führen auf der Menge der Konfigurationen von M eine Relation "$\vdash$" ein. $k \vdash k'$ soll bedeuten, daß k' diejenige Konfiguration ist,

die entsteht, wenn die Maschine, gestartet in Konfiguration k, einen Schritt macht. Wir sagen: k' ist die Nachfolgekonfiguration von k. Wir definieren die Relation $\vdash$ formal:

Definition 1.17: $M=(Z,A,\delta,z_o,z_e)$ sei eine 1-Band-Turingmaschine. Für $u,v\in A^*$, $a,b,c\in A$ und $z,z'\in Z$ gilt:

$$ubzav\vdash \begin{cases} uz'bcv & \text{falls } \delta(z,a)=(z',c,L)\\ ubz'cv & \text{falls } \delta(z,a)=(z',c,N)\\ ubcz'v & \text{falls } \delta(z,a)=(z',c,R)\end{cases}$$

$$zav\vdash \begin{cases} z'Bcv & \text{falls } \delta(z,a)=(z',c,L)\\ z'cv & \text{falls } \delta(z,a)=(z',c,N)\\ cz'v & \text{falls } \delta(z,a)=(z',c,R)\end{cases}$$

$$ubz\vdash \begin{cases} uz'bc & \text{falls } \delta(z,B)=(z',c,L)\\ ubz'c & \text{falls } \delta(z,B)=(z',c,N)\\ ubcz' & \text{falls } \delta(z,B)=(z',c,R)\end{cases}$$

$$z\vdash \begin{cases} z'Bc & \text{falls } \delta(z,B)=(z',c,L)\\ z'c & \text{falls } \delta(z,B)=(z',c,N)\\ cz' & \text{falls } \delta(z,B)=(z',c,R).\end{cases}$$

Glücklicherweise können wir uns in allen späteren Anwendungen auf die eine oder andere Teilmenge dieser Regeln beschränken.

In völlig analoger Weise kann man Konfigurationen und Nachfolgekonfigurationen für Mehrbandmaschinen definieren.

Wird eine 1-Bandmachine mit einer Eingabe w auf dem Band gestartet, so verabreden wir, daß der Kopf der Maschine anfangs auf dem ersten Zeichen der Eingabe steht.

Definition 1.18: $M=(Z,A,\delta,z_o,E)$ sei eine 1-Band-Turingmaschine, $w\in A^*$ sei eine Eingabe für M.

Eine endliche Folge von Konfigurationen $(k_o,\ldots k_T)$ ist eine endliche Rechnung von M angesetzt auf w, falls gilt:

i) k_o hat die Form $B\ldots Bz_owB\ldots B$

ii) $k_i\vdash k_{i+1}$ für alle $i\in\{0,\ldots,T-1\}$

iii) k_T ist Endfiguration.

Eine unendliche Folge $(k_o,k_1,\ldots)$, die i) und ii) erfüllt, ist eine unendliche Rechnung von M angesetzt auf w. Gibt es eine endliche Rechnung von M angesetzt auf w, sagen wir: M angesetzt auf w hält,

andernfalls sagen wir: M angesetzt auf w hält nicht.

Offenbar kann es nach dieser Definition mehrere Rechnungen von M angesetzt auf w geben. Diese unterscheiden sich jedoch in trivialer Weise und gehen durch Streichen bzw. Anhängen von B's an die Konfigurationen auseinander hervor.

Die Gründe, warum wir dies zulassen, sind rein technischer Natur. Die Rechnung, die mit $k_o = z_o w$ beginnt, nennen wir die kanonische Rechnung von M angesetzt auf w.

Beispiel 1.19: Wir geben eine 1-Band-Turingmaschine M an, die eine Bandinschrift $w \in \{0,1\}^*$ als Binärdarstellung*) einer natürlichen Zahl interpretiert, 1 hinzuaddiert und zum Anfang der Bandinschrift zurückläuft.

$$\left.\begin{aligned} \delta(z_o,0) &= (z_o,0,R)\\ \delta(z_o,1) &= (z_o,1,R)\\ \delta(z_o,B) &= (z_1,B,L) \end{aligned}\right\} \text{suche rechtes Ende der Bandinschrift}$$

$$\left.\begin{aligned} \delta(z_1,0) &= (z_2,1,L)\\ \delta(z_1,1) &= (z_1,0,L)\\ \delta(z_1,B) &= (z_e,1,N) \end{aligned}\right\} \text{Addiere 1}$$

$$\left.\begin{aligned} \delta(z_2,0) &= (z_2,0,L)\\ \delta(z_2,1) &= (z_2,1,L)\\ \delta(z_2,B) &= (z_e,B,R) \end{aligned}\right\} \text{suche linkes Ende der Bandinschrift}$$

Die kanonische Rechnung von M angesetzt auf 101 ist:

$(z_o 101,\ 1 z_o 01,\ 10 z_o 1,\ 101 z_o,\ 10 z_1 1,\ 1 z_1 00,\ z_2 110,\ z_2 B110,\ B z_e 110)$.

Beispiel 1.20: Für eine 1-Bandmaschine M mit Startzustand z_o und Alphabet A sei $\delta(z_o,à)=(z_o,a,R)$ für alle $a \in A$. Angesetzt auf eine beliebige Inschrift w tut die Maschine nichts weiter, als mit dem Kopf nach rechts abzuwandern. M angesetzt auf w hält für kein w.

Ist $(k_o,\ldots,k_T)$ eine Rechnung, so heißt T die Länge der Rechnung. Für kanonische Rechnungen heißt die Größe $\max\{|k_i|-1 \mid i \in \{0,\ldots,T\}\}$ der Bandverbrauch der Rechnung und gibt gerade die Anzahl der Felder an, die ursprünglich beschriftet waren oder während der

*) $b=b_1\ldots b_k$ mit $b_i \in \{0,1\}$ für $1 \le i \le k$ heißt Binärdarstellung von n genau dann, wenn

$$n = \sum_{i=1}^{k} b_i 2^{k-i}.$$

Rechnung vom Kopf besucht wurden. Wir bemerken, daß auch für unendliche Rechnungen die Größe $\max\{|k_i|-1 \mid i\in\mathbb{N}\}$ sinnvoll sein kann, nämlich dann, wenn eine Maschine auf einem endlichen Stück Band in eine unendliche Schleife gerät.

Alle bisherigen Definitionen lassen sich leicht auf Mehrbandmaschinen übertragen. Wir vereinbaren für Mehrbandmaschinen, daß beim Start einer Rechnung jeder Kopf auf dem ersten von B verschiedenen Zeichen der Inschrift seines Bandes steht (sofern das Band nicht leer ist). Der Bandverbrauch wird erklärt als die Anzahl der Felder auf allen Bändern, die ursprünglich beschriftet waren oder während der Rechnung besucht wurden. Ist M eine Mehrband-Turingmaschine und w eine einzelne Zeichenreihe, so vereinbaren wir, daß die Rechnung von M angesetzt auf w mit w als Inschrift auf Band 1 und mit ansonsten leeren Bändern beginnt.

Die folgende Definition ist dann für 1-Bandmaschinen wie für Mehrbandmaschinen sinnvoll:

Definition 1.21: Sei $t:\mathbb{N}\to\mathbb{N}$ eine eventuell partielle Funktion und M eine Turingmaschine. M heißt t(n)-zeitbeschränkt, wenn für alle Eingaben w die Länge der kanonischen Rechnung von M angesetzt auf w (und damit die Länge jeder Rechnung von M angesetzt auf w) höchstens $t(|w|)$ ist. M heißt t(n)-bandbeschränkt, wenn für alle Eingaben w der Bandverbrauch der kanonischen Rechnung von M angesetzt auf w höchstens t(n) ist.

Die Maschine aus Beispiel 1.19 ist (2n+2)-zeitbeschränkt und (n+2)-bandbeschränkt.

Beispiel 1.22: Zur Illustration der Arbeitsweise von Mehrbandmaschinen geben wir eine 2-Bandmaschine an, die die Inschrift von Band 1 auf Band 2 kopiert und dann beide Köpfe gleichzeitig zu ihrer anfänglichen Position zurückbewegt.

$$\delta(z_o,a,B) = (z_o,a,a,R,R) \qquad \text{für } a\neq B$$
$$\delta(z_o,B,B) = (z_1,B,B,L,L)$$
$$\delta(z_1,a,a) = (z_1,a,a,L,L) \qquad \text{für } a\neq B$$
$$\delta(z_1,B,B) = (z_e,B,B,R,R)\,.$$

Die Maschine ist (2n+2)-zeitbeschränkt und (2n+4)-bandbeschränkt.

Das nächste Beispiel zeigt, wie man die endliche Zustandsmenge einer Maschine dazu benutzen kann, im Laufe einer Rechnung vor-

übergehend beschränkt viel Information zu speichern.

Beispiel 1.23: Wir geben eine Maschine $M=(Z,A,\delta,z_o,\{z_e\})$ an. Angesetzt auf $w\in(A\setminus\{B\})^*$ verschiebt M die Inschrift w um 1 Feld nach rechts.

$Z=\{z_o,z_1,z_2,z_e\}\cup\{z^a \mid a\in A, a\neq B\}$.

$$\begin{aligned}
\delta(z_o,a) &= (z_o,a,R) && a\neq B\\
\delta(z_o,B) &= (z_1,B,L)\\
\delta(z_1,a) &= (z^a,B,R) && a\neq B\\
\delta(z^a,B) &= (z_o,a,L)\\
\delta(z_1,B) &= (z_2,B,R)\\
\delta(z_2,B) &= (z_e,B,R)\ .
\end{aligned}$$

Diese Maschine merkt sich in den Zuständen z^a, welches Zeichen sie gerade nach rechts verschiebt.

Seien $a,b\in A\setminus\{B\}$. Die kanonische Rechnung von M angesetzt auf $w=ab$ ist

$$(z_oab,\ az_ob,\ abz_o,\ az_1bB,\ aBz^bB,\ az_oBb,\ z_1aBb,\ Bz^aBb,\ z_oBab,\ z_1BBab,\ Bz_2Bab,\ BBz_eab).$$

M ist $(3n+3)$-zeitbeschränkt und $(n+2)$-bandbeschränkt.

1.3.3. Techniken zur Konstruktion von Turingmaschinen

Das Spezifizieren von Übergangsfunktionen für Turingmaschinen ist mühsam. Wir schaffen uns deshalb Techniken, mit denen man aus einfachen Maschinen kompliziertere Maschinen zusammenbauen kann.

i) Wir wollen n k-Bandmaschinen $M_i=(z_i,A_i,\delta_i,z_{oi},E_i)$ $(1\leq i\leq n)$ zusammenschalten. Da durch Umbenennung von Zuständen das Verhalten von Maschinen nicht geändert wird, kann OBdA $z_i\cap z_j=\emptyset$ für $i\neq j$ angenommen werden. Wir bilden zunächst formal eine Maschine mit Zustandsmenge $\cup Z_i$, Alphabet $\cup A_i$ und Übergangsfunktion $\delta(z,a)=\delta_i(z,a)$ für $z\in Z_i$. Für diese Maschine schreiben wir

$$M_1 \ldots M_k$$

Ist $q\in E_i$, so soll es erlaubt sein, q mit dem Anfangszustand z_{oj} irgendeiner Maschine M_j zu identifizieren, indem man für alle a

$$\delta(q,a) = \delta_j(z_{oj},a)$$

setzt.

Ist $q \in E_i$ mit z_{oj} identifiziert, notieren wir dies durch

$$M_i \overset{q}{\rightarrow} M_j .$$

Werden alle Endzustände von M_i mit dem Anfangszustand von M_j identifiziert, schreiben wir einfach

$$M_i \rightarrow M_j$$

Durch mehrfaches Identifizieren von Zuständen kann man somit schon ziemlich komplizierte Maschinen aufbauen.

Mit

$$\text{start} \rightarrow M_i$$

notieren wir, daß der Anfangszustand der neuen Maschine gleich z_{oi} sein soll.

Mit der Schreibweise

$$M_i \overset{q}{\rightarrow} \text{stop}$$

betonen wir, daß $q \in E_i$ ein Endzustand der neuen Maschine sein soll. Sind alle Endzustände von M_i Endzustände der neuen Maschine, schreiben wir einfach

$$M_i \rightarrow \text{stop} .$$

Beispiel 1.24: Sei M die Maschine aus Beispiel 1.19, die zu Binärzahlen eins hinzuaddiert.

Die Maschine

$$\text{start} \rightarrow M \rightarrow M \rightarrow M \rightarrow \text{stop}$$

addiert zu Binärzahlen drei hinzu.

Für die Maschine

$$\text{start} \rightarrow M_1 \rightarrow \ldots \rightarrow M_n \rightarrow \text{stop}$$

schreiben wir einfach "$M_1;\ldots;M_n$".

Beispiel 1.25: Sei wieder M wie in Beispiel 1.19. Angesetzt auf die Binärdarstellung von n erzeugt die Maschine

$$\text{start} \rightarrow M$$

die Binärdarstellungen von $n+1, n+2, \ldots$.

Leser, die schon programmiert haben, werden in diesen durch markierte Pfeile verbundenen Systemen von Maschinen sofort Flußdiagramme wiedererkennen können.

ii) Ist $M=(Z,A,\delta,z_o,E)$ eine k-Bandmaschine, ist $m>k$ und ist $\{i_1,\ldots,i_k\}$ eine der Größe nach geordnete Auswahl von k verschiedenen Zahlen aus $\{1,\ldots,m\}$, dann kann man aus M eine m-Bandmaschine $M'=(Z',A',\delta',z_o,E')$ konstruieren, die sich auf den Bändern $i_1,\ldots,i_k$ wie M verhält, nämlich

$$Z'=Z,\ A'=A,\ z_o'=z_o,\ E'=E,$$

$$\delta'(z,a)=(z',c,s)\quad (z,z'\in Z, a,c\in A^m, s\in\{L,R,N\}^m)$$

wobei gilt:

$$a_i=c_i \text{ und } s_i=N \text{ für } i\notin\{i_1,\ldots,i_k\},$$

$$(z',c_{i_1},\ldots,c_{i_k},s_{i_1},\ldots,s_{i_k})=\delta(z,a_{i_1},\ldots,a_{i_k}).$$

Wir kürzen die Maschine M' durch $M(i_1,\ldots,i_k)$ ab.

Beispiel 1.26: Ist K die Maschine aus Beispiel 1.22, die die Inschrift von Band 1 auf Band 2 kopiert, so kopiert K(i,j) die Inschrift von Band i auf Band j.

Wir kürzen Maschinen, die wir später als Bausteine für andere Maschinen brauchen, mit Namen ab, die man sich leicht merken kann.

"Lösche Band" sei eine 1-Bandmaschine mit Übergangsfunktion

$$\delta(z_o,a)=(z_o,B,R) \qquad \text{für} \quad a\neq B$$

$$\delta(z_o,B)=(z_e,B,N)\ .$$

Diese Maschine löscht die gesamte Bandinschrift, falls diese nicht durch B's unterbrochen ist.

Die Maschine "Lösche Band (j); K(i,j)" wird mit "Band j: = Band i" abgekürzt. Auf ähnliche Art konstruiert man eine Maschine "Band j: = Band j# Band i", die die Inschrift von Band i, durch ein #-Zeichen getrennt, hinter die Inschrift von Band j schreibt.

Sei M die Maschine aus Beispiel 1. M(i) wird mit "Band i: = Band i+1" abgekürzt. Schließlich bezeichne "Band i: = Band i-1" eine Maschine, die von einer Binärzahl auf Band i eins abzieht und führende Nullen streicht.

iii) Setzt man die folgende 1-Bandmaschine auf eine Bandinschrift w an, die nicht von B's unterbrochen ist, so geht die Maschine in den Zustand "ja" falls w=0 und sonst in den Zustand "nein".

$Z=\{z_o, z_1, ja, nein\}$.

$\delta(z_o, a) = (nein, N)$ für $a \neq 0$

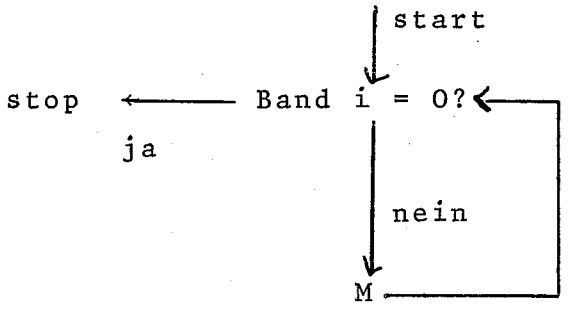
$\delta(z_o, 0) = (z_1, R)$

$\delta(z_1, a) = (nein, L)$ für $a \neq B$

$\delta(z_1, B) = (ja, L)$

Wir bezeichnen diese Maschine mit "Band = 0?" und die Maschine "Band = 0?" (i) mit "Band i = 0?".

Ist M eine k-Bandmaschine, so bezeichnen wir die Maschine



mit "while Band i≠0 do M".

Diese Maschine testet erst, ob die Inschrift von Band i=0 ist. Wenn nicht, verhält sich diese Maschine wie M.

Die Rechnung von M verändert im allgemeinen die Inschrift von Band i. Nach Beendigung der Rechnung von M wird wieder Band i=0? getestet. Dies wird so lange fortgesetzt, bis die Inschrift von Band i=0 ist. Wenn und falls dies geschieht, hält die gesamte Maschine.

Beispiel 1.27: Startet man die Maschine "while Band 2≠0 do

"Band 1: = Band 1+1;
Band 2: = Band 2-1""

mit Binärzahlen x und y auf Band 1 und 2, so steht nach dem i-ten Durchlauf der while-Schleife auf Band 1 x+i und auf Band 2 y-i. Die Maschine hält mit x+y auf Band 1.

1.3.4. Turingberechenbare Funktionen

Für $n \in \mathbb{N}$ bezeichnet im folgenden immer bin(n) die Binärdarstellung von n ohne führende Nullen. $bin: \mathbb{N} \to \{0,1\}^+$ ist eine injektive Funktion, und $bin^{-1}: \{0,1\}^+ \to \mathbb{N}$ mit

$$bin^{-1}(w_1 \ldots w_s) = \sum_{i=1}^{s} w_i 2^{s-i}$$

bezeichnet die Umkehrung von bin.

Jede Turingmaschine M mit Alphabet A definiert in natürlicher Weise eine eventuell partielle Funktion $f_M: A^* \to A^*$, nämlich

$f_M(w)$ = die Inschrift von Band 1 am Ende der Rechnung von M angesetzt auf w.

In (beinahe) ebenso natürlicher Weise definiert M zu jedem $r \in \mathbb{N} \setminus \{0\}$ eine Funktion $f_M^r : \mathbb{N}^r \to \mathbb{N}$: Hält M angesetzt auf $bin(x_1)\#\ldots\#bin(x_r)$ mit einer Inschrift $y \in \{0,1\}^+$ auf Band 1, so ist $f_M^r(x_1,\ldots,x_r) = bin^{-1}(y)$, andernfalls ist $f_M^r(x_1,\ldots,x_r)$ nicht definiert.

Definition 1.28: Eine Funktion $f: A^* \to A^*$ heißt turingberechenbar, wenn es eine Turingmaschine M mit $f = f_M$ gibt. Eine Funktion $f: \mathbb{N}^r \to \mathbb{N}$ heißt turingberechenbar, wenn es eine Turingmaschine M mit $f = f_M^r$ gibt.

Welche Funktionen sind turingberechenbar? Eine erste Antwort auf diese Frage gibt

Satz 1.29: Jede μ-rekursive Funktion ist turingberechenbar.

Beweis: Eine Maschine heißt regulär, wenn am Ende jeder Rechnung jeder Kopf auf dem ersten von B verschiedenen Zeichen seines Bandes steht. Alle bisher konstruierten Maschinen sind regulär. Zum Beweis von Satz 1.29 genügt es offenbar, die folgenden technischen Lemmas zu beweisen:

Lemma 1.3o: Die Nachfolgerfunktion N, die konstanten Funktionen c_s^r und die Projektionen p_i^r können durch reguläre 1-Band-Turingmaschinen berechnet werden.

Lemma 1.31: Sind $f: \mathbb{N}^r \to \mathbb{N}$ und $g_i: \mathbb{N}^m \to \mathbb{N}$ $(1 \leq i \leq r)$ Funktionen, entsteht h durch Einsetzung der g_i in f und sind $F, G_1, \ldots, G_r$ reguläre k-Band-Turingmaschinen, die $f, g_1, \ldots, g_r$ berechnen, dann gibt es eine reguläre k+2-Band-Turingmaschine M, die h berechent.

Lemma 1.32: Sind $g: \mathbb{N}^r \to \mathbb{N}$ und $h: \mathbb{N}^{r+2} \to \mathbb{N}$ Funktionen, geht f durch primitive Rekursionen aus g und h hervor und sind G und H reguläre k-Band-Turingmaschinen, die g und h berechnen, dann gibt es eine reguläre k+4-Band-Turingmaschine M, die f berechnet.

Lemma 1.33: Ist $f: \mathbb{N}^r \to \mathbb{N}$ eine Funktion und ist F eine reguläre

k-Band-Turingmaschine, die f berechnet, dann gibt es eine reguläre k+3-Band-Turingmaschine M, die μf berechnet.

Beweis von Lemma 1.3o: Die Maschine aus Beispiel 1.19 ist regulär und berechnet N.

Für jede Zeichenreihe $w\in\{0,1\}^+$ geben wir eine reguläre 1-Band-Turingmaschine (Z,A,δ,z_o,z_e) an, die w auf das leere Band schreibt.

Sei $w=w_1\ldots w_s, w_i\{0,1\}$.

Wir setzen $Z=\{z_o,z_1,\ldots,z_s\}$, $z_s=z_e$,

$$\delta(z_i,B)=(z_{i+1},w_{s-i},L) \quad \text{für} \quad 0\leq i\leq s-2,$$

$$\delta(z_{s-1},B)=(z_s,w_1,N).$$

Diese Maschine, die wir mit "Drucke w" abkürzen, ist gleichzeitig ein Beispiel dafür, wie man durch Konstruktion der Übergangsfunktion eine Maschine im voraus mit endlich viel Wissen (hier: der Kenntnis der Zeichenreihe w) ausstatten kann. Man sagt auch, daß die obige Maschine die Zeichenreihe w "im endlichen Gedächtnis gespeichert hat".
Die Maschine "Lösche Band; Drucke bin(s)" ist regulär und berechnet c_s^r für jedes r.

Man kann leicht reguläre 1-Bandmaschinen H bzw. T angeben, die, angesetzt auf eine Zeichenreihe $w^1\#w^2\#\ldots\#w^r$, $w^i\in\{0,1\}^+$ schließlich mit Bandinschrift w^1 bzw. $w^2\#\ldots\#w^r$ halten. Die Maschine "T;...;T((i-1)-mal);H" ist regulär und berechnet P_i^r. □

Wir bezeichnen die Maschinen "Lösche Band; Drucke w" (i) bzw. H(i) bzw. T(i) mit "Band i:=w" bzw. "Band i:=Head (Band i)" bzw. "Band i:=Tail (Band i)".

Beweis von Lemma 1.31: Für $i\in\{1,\ldots,r\}$ definieren wir die Maschine M_i durch:

M_i = "Band 3: = Band 1;

$G_i(3,\ldots,k+2)$;

Band 2: = { Band 3 für i=1; Band 2 # Band 3 für i>1;

Lösche Band 3;...; Lösche Band k+2".

Dann ist die Maschine M definiert durch:

M = "M_1;...;M_r;
F(2,...,k+1);
Band 1: = Band 2". □

Beweis von Lemma 1.32: Wir geben eine Maschine M mit einer while-Schleife an, die folgende Eigenschaft hat: Wird M mit Eingabe bin(n)#bin(x_1)#...#bin(x_r) gestartet, $n,x_j \in \mathbb{N}$, dann steht nach dem i-ten Durchlaufen der Schleife auf Band 1 bin(n-i), auf Band 2 bin(x_1)#...#bin(x_r), auf Band 3 bin(f(i,x_1,...,x_r)) und auf Band 4 bin(i). Die Maschine ist

```
M = "Band 2: = Band 1;
     Band 1: = Head(Band 1);
     Band 2: = Tail(Band 2);
     Band 3: = Band 2;
     G(3,...,k+2);
     Band 4: = 0;
     Lösche Band (5);...;Lösche Band (k+2);
     while Band 1≠0 do
          "Band 5: = Band 4;
           Band 5: = Band 5#Band 3;
           Band 5: = Band 5#Band 2;
           H(5,...,k+4);
           Band 3: = Band 5;
           Lösche Band (5);...;Lösche Band (k+4);
           Band 1: = Band 1-1;
           Band 4: = Band 4+1";
     Band 1: = Band 3". □
```

Beweis von Lemma 1.33: Wir geben eine Maschine M mit einer while-Schleife an, die folgende Eigenschaft hat: Wird M mit Eingabe bin(x_1)#...#bin(x_r) gestartet, dann steht nach dem i-ten Durchlaufen der Schleife auf Band 1 bin(f(i-1,x_1,...,x_r)), auf Band 2 bin(x_1)#...#bin(x_r) und auf Band 3 bin(i). Die Maschine ist gegeben durch:

```
M = "Band 2: = Band 1;
     Band 1: = 1;
     Band 3: = 0;
     while Band 1≠0 do
          "Band 4: = Band 3;
           Band 4: = Band 4#Band 2;
           F(4,...,k+3);
           Band 1: = Band 4;
           Lösche Band (4);...;Lösche Band (k+3);
           Band 3: = Band 3+1";
     Band 3: = 3-1;
     Band 1: = Band 3". □
```

1.3.5. Zwei einfache Simulationen

Sind M und M' Maschinen und ist für alle Eingaben w für M das Ergebnis der Rechnung von M' angesetzt auf w gleich dem Ergebnis der Rechnung von M angesetzt auf w, so sagen wir: M' simuliert M. Wir zeigen, daß jede Turingmaschine von einer Turingmaschine mit einseitig unendlichen Bändern simuliert werden kann und daß jede Mehrbandmaschine von einer 1-Bandmaschine simuliert werden kann.

Wir verabreden einige Notationen. Sind $f,g:\mathbb{N}\to\mathbb{N}$ Funktionen, so schreiben wir:

$f=O(g)$, falls es eine Zahl $c\in\mathbb{N}$ gibt, so daß $f(n)\leq c\cdot g(n)$ für alle $n\in\mathbb{N}$ gilt.

$f=o(g)$ falls $\lim_{n\to\infty} f(n)/g(n)=0$.

$f=\Omega(g)$ falls $g=O(f)$.

$f=\omega(g)$ falls $g=o(f)$.

Anschaulich bedeutet $f=O(g)$ bzw. $f=\Omega(g)$: f wächst bis auf einen konstanten Faktor höchstens bzw. mindestens so schnell wie g. $f=o(g)$ bzw. $f=\omega(g)$ bedeutet: f wächst asymptotisch langsamer bzw. schneller als g.

Satz 1.34: Jede k-Band-Turingmaschine M kann von einer k-Band-Turingmaschine M' simuliert werden, die nie Felder links von der ursprünglichen Position der Köpfe besucht. Ist M s(n)-bandbeschränkt und t(n)-zeitbeschränkt und $t(n)\geq n$, so ist M' O(s(n))-

bandbeschränkt und $O(t(n)+s^2(n))$-zeitbeschränkt falls k=1; M' ist $O(t(n))$-zeitbeschränkt falls $k\geq 2$.

Beweis: Wir zeigen zuerst einen Spezialfall:

Lemma 1.35: Sei $t(n)\geq n$ und M sei eine s(n)-bandbeschränkte und t(n)-zeitbeschränkte 1-Band-Turingmaschine, deren Ausgabe keine Felder links von der ursprünglichen Position der Köpfe belegt. Dann kann M von einer (s(n)+1)-bandbeschränkten und $O(t(n))$-zeitbeschränkten 1-Bandmaschine M' simuliert werden, die nie Felder links von der ursprünglichen Position des Kopfs benutzt.

Beweis: Sei $M=(Z,A,\delta,z_o,E)$. Wir stellen uns die Felder des Bandes von M mit den ganzen Zahlen $\mathbb{Z}=\{\ldots -2,-1,0,1,2,\ldots\}$ durchnumeriert vor. Der Kopf von M stehe anfangs auf Feld 0.

Sei $x\notin A$ ein neues Zeichen. M' ist eine Maschine mit Alphabet $A'=A\cup A\times(A\cup\{x\})$ und einer Zustandsmenge der Form $Z'=Z\times\{1,2\}\cup Z''$. M' kann also auf einem Feld ein Zeichen aus A oder ein Paar [a,b] von Zeichen mit $a\in A, b\in A\cup\{x\}$ speichern.

Am besten stellt man sich das Band von M' in zwei Spuren ähnlich den Spuren eines Stereotonbandgeräts eingeteilt vor. Ein Zeichen $a\in A$ auf einem Feld belegt beide Spuren, von einem Paar [a,b] belegt a die obere Spur und b die untere Spur. Das Zeichen x wird später zum Markieren des linken Endes der Bandinschrift von M' benutzt.

M' arbeitet nur rechts von Feld -1, und zwar so, daß für alle i gilt:

(*) Ist nach dem i-ten Schritt von M angesetzt auf eine Eingabe w die Maschine M im Zustand z, steht der Kopf von M auf Feld j und ist das Band von M gleich

	... -1		0		1 ...			
... B	u_s	...	u_1	v_o	v_1	...	v_s	B ...

mit $u_s\neq B$ oder $v_s\neq B$, so ist das Band von M' nach Simulation dieses Schritts gleich

0	1	...			
v_o	v_1	...	v_s	B ...	Spur 1
x	u_1	...	u_s		Spur 2,

der Kopf von M' steht auf Feld j, und der Zustand von M' ist

$$\begin{cases} (z,1) & \text{falls } j>0 \\ (z,2) & \text{falls } j<0 \\ (z,1) \text{ oder } (z,2) & \text{falls } j=0. \end{cases}$$

Die Bandinschrift von M wird also am Feld 0 gefaltet, und die Maschine M' merkt sich in der zweiten Komponente des Zustands, welche Spur sie gerade bearbeitet.

M' ist aus drei Maschinen zusammengesetzt:

$$M' = "M_1;M_2;M_3".$$

Angesetzt auf Eingabe $w=w_1 \ldots w_s$ erzeugt M_1 die Bandinschrift

w_1	w_2	...	w_s	B ...
x	B	...	B	

und bewegt den Kopf zum Anfang der Bandinschrift zurück. Damit gilt (*) für i=0.

Wir definieren $M_2=(A',Z\times\{1,2\},\delta_2,[z_o,1],E\times\{1,2\})$.

Ist $\delta(z,a)=(z',c,s)$, so gilt mit den Bezeichnungen $\overline{R}=L, \overline{L}=R$ und $\overline{N}=N$:

$$\delta_2([z,1],[a,b])=([z',1],[c,b],s) \qquad b\in A,$$

$$\delta_2([z,2],[b,a])=([z',2],[b,c],\overline{s}) \qquad b\in A,$$

$$\delta_2([z,i],[a,x]) = \begin{cases} ([z',2],[c,x],R) & \text{falls } s=L \\ ([z',i],[c,x],N) & \text{falls } s=N \\ ([z',1],[c,x],L) & \text{falls } s=R \end{cases} \qquad i=1,2.$$

Ist $\delta(z,B)=(z',c,s)$, so gilt überdies

$$\delta_2([z,1],B)=([z',1],[c,B],s)$$

$$\delta_2([z,2],B)=([z',2],[B,c],\overline{s}).$$

Für alle i simuliert der i-te Schritt von M_2 den i-ten Schritt von M. Offenbar bleibt hierdurch (*) für i erfüllt, falls (*) für i-1 erfüllt war. Per Induktion folgt, daß die Bandinschrift von M' am Ende der Rechnung von M_2 gerade die am Feld 0 gefaltete Bandinschrift von M am Ende der Rechnung von M ist.

M_3 schließlich, gestartet mit dem Kopf auf irgendeinem von B verschiedenen Feld von

v_o	v_1	...	v_s	B ...
x	u_1	...	u_s	

,

erzeugt hieraus

v_o	....	v_s	B

.

Da nach Voraussetzung das Ergebnis der Rechnung von M auf den Feldern rechts von Feld -1 steht, hält M_3 mit dem Ergebnis der Rechnung von M auf dem Band. M' simuliert also M.

M' ist offenbar (s(n)+1)-bandbeschränkt. Es ist leicht, die Maschinen M_1 und M_3 so zu konstruieren, daß sie O(n)-zeitbeschränkt sind. Ist w eine Eingabe und $|w|=n$, so macht M_1 O(n) Schritte. M_2 macht t(n) Schritte und erzeugt dabei eine Inschrift der Länge höchstens t(n). M_3 macht daher höchstens O(t(n)) Schritte. M' angesetzt auf w macht also höchstens $O(n)+O(t(n))=O(t(n))$ Schritte. □

Da die gleiche Simulation gleichzeitig auf mehreren Bändern durchgeführt werden kann, gilt Lemma 1.35 auch für k-Bandmaschinen.

Sei (Z,A,δ,z_o,E) eine beliebige 1-Bandmaschine und $M'=(Z',A',\delta',z_o',E')$ sei eine 1-Bandmaschine, deren Band k Spuren hat, d.h. $A'\supset A_1\times\ldots\times A_k$ und es gelte $A_i=A$. Wir sagen M' <u>verhält sich auf Spur</u> i <u>wie</u> M falls gilt:

$\delta'(z,[a_1,\ldots,a_i,\ldots,a_k])=(z',[a_1,\ldots,c,\ldots,a_k],s)$, falls $\delta(z,a_i)=(z',c,s)$

$\delta'(z,B)=(z',[B,\ldots,c,\ldots,B],s)$ falls $\delta(z,B)=(z',c,s)$.

Eine ähnliche Konstruktion ist für Mehrbandmaschinen möglich. Zum Beweis von Satz 1.34 genügt es nun, das folgende Lemma zu zeigen:

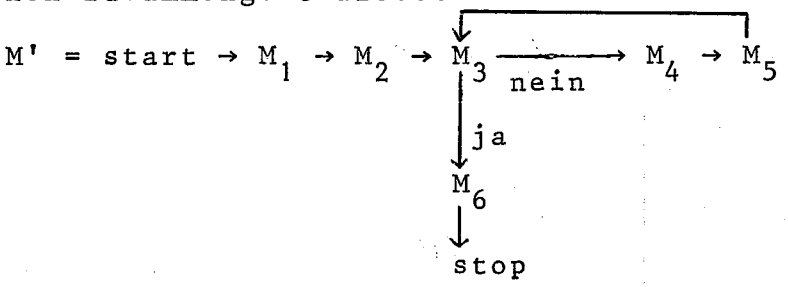
<u>Lemma 1.36:</u> Sei $t(n)\geq n$ und sei M eine s(n)-bandbeschränkte und t(n)-zeitbeschränkte k-Band-Turingmaschine. Dann kann M von einer k-Band-Turingmaschine M' simuliert werden, deren Ausgabe keine Felder links von der ursprünglichen Position der Köpfe belegt.

M' ist $O(s(n))$-bandbeschränkt und $O(t(n)+s^2(n))$-zeitbeschränkt, falls k=1; M' ist $O(t(n))$-zeitbeschränkt, falls $k \geq 2$.

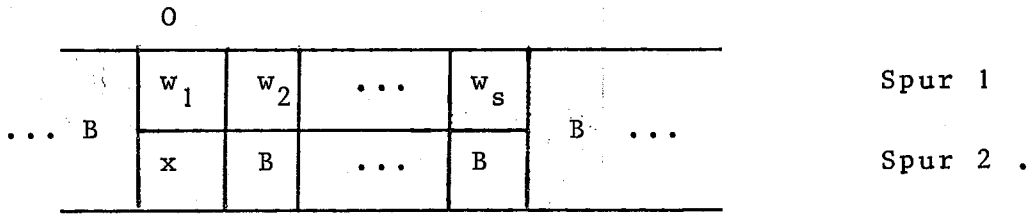
Beweis: OBdA druckt M nie das Leerzeichen B. (Andernfalls führt man ein neues Zeichen B' ein und modifiziert M so, daß M das Zeichen B' wie B interpretiert, aber statt B immer B' druckt; am Ende der gesamten Simulation ersetzt man dann im Ergebnis alle Zeichen B' wieder durch B).

Wir beweisen zuerst den Fall k=1:

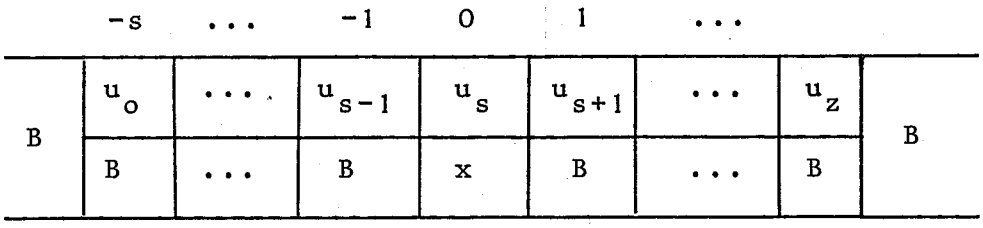
A sei das Bandalphabet von M, und $x \notin A$ sei wieder ein Sonderzeichen, das die ursprüngliche Kopfposition markiert. M' ist aus 6 Maschinen zusammengeschaltet:



M_1 ist identisch mit M_1 aus dem Beweis des vorigen Lemmas, d.h. angesetzt auf Eingabe w erzeugt M_1



M_2 verhält sich auf Spur 1 wie M und erzeugt schließlich eine Inschrift der Form



Falls M nicht regulär ist, muß M_2 am Ende der Simulation noch den Kopf zum linken Ende der Bandinschrift bewegen.

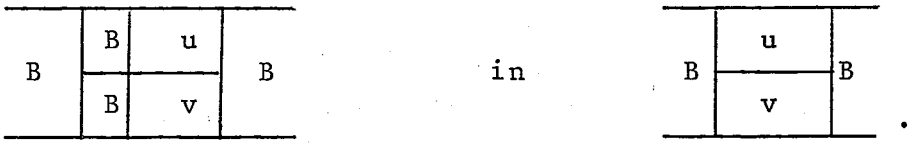
M_3 ist spezifiziert durch

$$\delta(z_o,[a,x]) = (ja,[a,x],N)$$

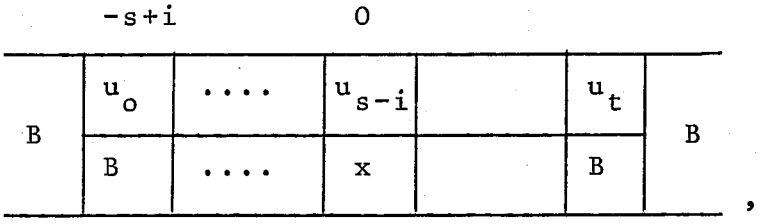
$$\delta(z_o,[a,b]) = (nein,[a,b],N) \qquad b \in A .$$

M_4 verhält sich auf Spur 1 wie die Maschine aus Beispiel 1.23. (Da M nie das Leerzeichen druckt, wird hierdurch tatsächlich die gesamte Inschrift von Spur 1 verschoben).

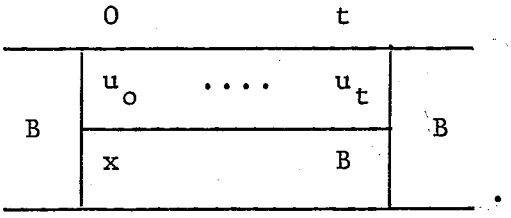
M_5 überführt eine Inschrift der Form



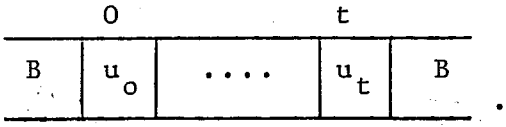
Nach dem i-ten Durchlauf der Schleife $M_3 \to M_4 \to M_5 \to M_3$ erhält man damit die Inschrift



d.h. für i=s:



M_6 erzeugt hieraus



M' simuliert offenbar M und ist (2s(n)+2)-bandbeschränkt. M_4 ist O(n)-zeitbeschränkt, M_3 und M_5 sind O(1)-zeitbeschränkt (d.h. sie machen nur eine konstante Anzahl von Schritten), und es ist leicht, M_1 und M_6 so zu konstruieren, daß sie O(n)-zeitbeschränkt sind. ist w eine Eingabe und ist |w|=n, so macht M_1 O(n) Schritte, M_2 macht O(t(n)) Schritte und erzeugt eine Inschrift der Länge höchstens s(n). Einmaliges Durchlaufen der Schleife $M_3 \to M_4 \to M_5 \to M_3$ erfordert deshalb O(s(n)) Schritte und die Schleife muß höchstens 2(s(n)) mal durchlaufen werden. M_6 findet eine Inschrift der Länge höchstens s(n)) vor und macht O(s(n)) Schritte. Damit ergibt sich als Gesamtschrittzahl

$$O(n)+O(t(n))+O(s^2(n))+O(s(n))=O(t(n)+s^2(n)).$$

Der Beweis für den Fall $k \geq 2$ ist einfacher:

$M' = "M_{-1}; M_o; \ldots; M_{2k}"$.

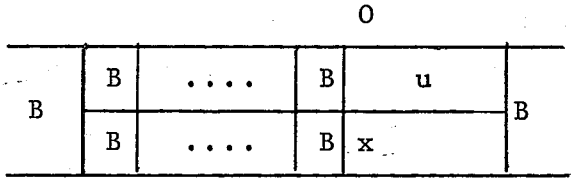
M_{-1} kopiert die Eingabe auf Spur 1 von Band 1 und markiert Spur 2 von Feld 0 eines jeden Bandes mit x; M_o verhält sich auf Spur 1 eines jeden Bandes wie M.

Für $2 \leq i \leq k$ tut M_i das folgende:

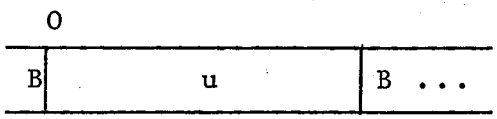
"Kopiere Spur 1 von Band i auf die Felder 1,2,... (d.h. rechts von x) von Spur 2 von Band 1, lösche gleichzeitig Spur 1 von Band i; kopiere die Felder 1,2,... von Spur 2 von Band 1 auf die Felder 1,2,... von Spur 1 von Band i, lösche gleichzeitig die Felder 1,2,... von Spur 2 von Band 1."

M_1 erhält man aus M_2, indem man die Rollen von Band 1 und Band 2 vertauscht.

Nach dem Ende der Rechnung von M_k hat für alle i die Inschrift von Band i die Form



M_{k+i} erzeugt hieraus



M' ist offenbar $O(s(n))$-bandbeschränkt, und man überzeugt sich leicht, daß man die Maschinen $M_{-1}, M_o, \ldots, M_{2k}$ so konstruieren kann, daß M' $O(t(n))$-zeitbeschränkt ist. □

Satz 1.37: Ist M eine s(n)-bandbeschränkte und t(n)-zeitbeschränkte k-Band-Turingmaschine, so kann M von einer regulären $O(s(n))$-bandbeschränkten und $O(s(n) \cdot t(n))$-zeitbeschränkten 1-Band-Turingmaschine M' simuliert werden.

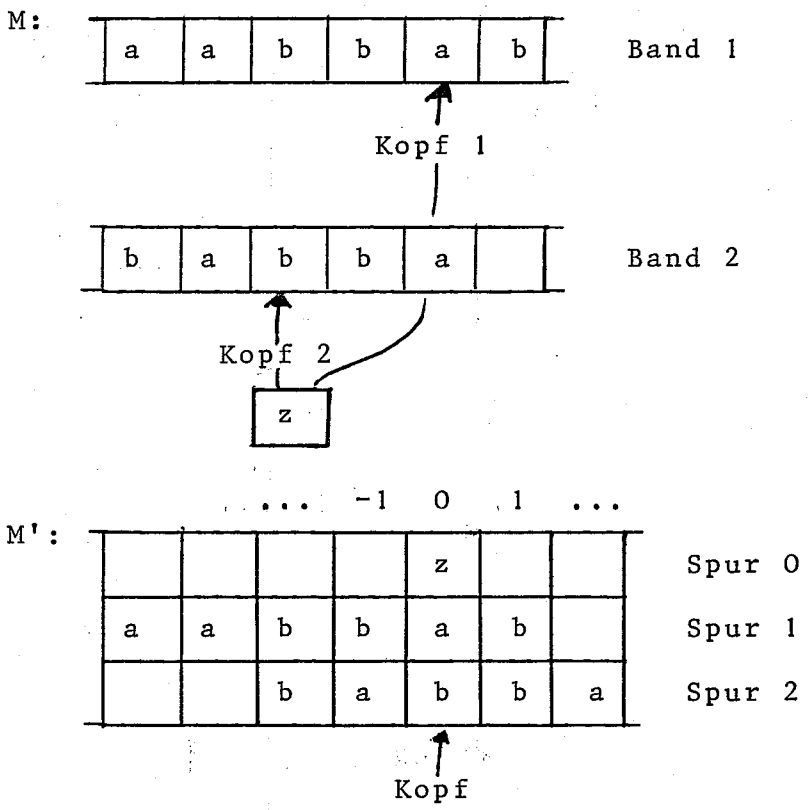
Beweis: Sei $M = (Z, A, \delta, z_o, E)$. OBdA druckt M nie das Leerzeichen. M' hat k+1 Spuren $0, \ldots, k$ und simuliert M so, daß für alle i und j gilt:

(*) Ist M angesetzt auf w nach i Schritten im Zustand z, ist $uav \in A^*$ mit $a \in A$ die Inschrift von Band j von M und steht der Kopf

von Band j auf Zeichen a, so gilt nach Simulation dieses Schritts durch M':

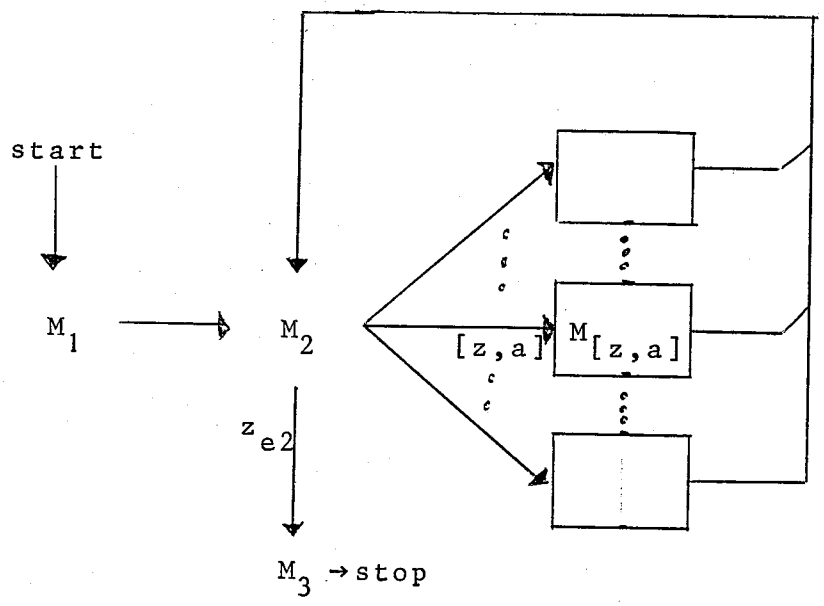
M' ist im Zustand z, die Inschrift von Spur j des Bandes von M' ist uav, wobei a auf Feld 0 steht, auf Feld 0 von Spur 0 steht z, Spur 0 ist ansonsten leer, und der Kopf von M' steht auf Feld 0.

Beispiel 1.38:



M' wird also Kopfbewegungen von M auf Band i durch Verschieben der Inschrift von Spur i in der entgegengesetzten Richtung simulieren müssen.

M' ist zusammengesetzt aus drei Maschinen: M_1, M_2, M_3 sowie je einer Maschine $M_{[z,a]}$ für jedes Tupel [z,a] mit $z \in Z \setminus E$, $a=(a_1,\ldots,a_k) \in A^k$. Jede Maschine $M_{[z,a]}$ simuliert die Kopfbewegungen, die M ausführt, wenn M im Zustand z auf Band 1 a_1, auf Band 2 a_2,... und auf Band k a_k liest.



Angesetzt auf Eingabe w kopiert M_1 die Eingabe auf Spur 1 und markiert Feld 0 von Spur 0 mit z_o. Damit gilt (*) für i=0.

M_2 hat Zustandsmenge $\{z_{o2}, z_{e2}\} \cup \{[z,a] \mid z \in Z \setminus E, a \in A^k\}$. M_2 simuliert das Verändern von Bandinschriften und den Wechsel des Zustands von M, und übergibt dann die Arbeit zur Simulation der Kopfbewegungen an eine passende Maschine $M_{[z,a]}$. Die Übergangsfunktion von M_2 ist

$$\delta_2(z_{o2},[z,a]) = \begin{cases} ([z,a],[z',c],N) \text{ falls } \delta(z,a)=(z',c,s) \\ (z_{e2},[z,a],N) \text{ falls } z \in E \; a,c \in A^k, s \in \{R,L,N\}^k \end{cases}$$

Für alle [z,a] mit $z \notin E$ ist der Endzustand [z,a] von M_2 mit dem Startzustand von $M_{[z,a]}$ identifiziert.

Jede Maschine $M_{[z,a]}$ besteht aus k Maschinen

$$M_{[z,a]} = "M^1_{[z,a]}; \ldots ; M^k_{[z,a]}" ,$$

wobei $M^i_{[z,a]}$ die Kopfbewegung von M auf Band i simuliert.

Wir bezeichnen mit "R-shift" die Maschine aus Beispiel 1.23, die die Bandinschrift um 1 Feld nach rechts verschiebt, und mit "L-shift" eine ähnliche Maschine, die die Bandinschrift um 1 Feld nach links verschiebt.

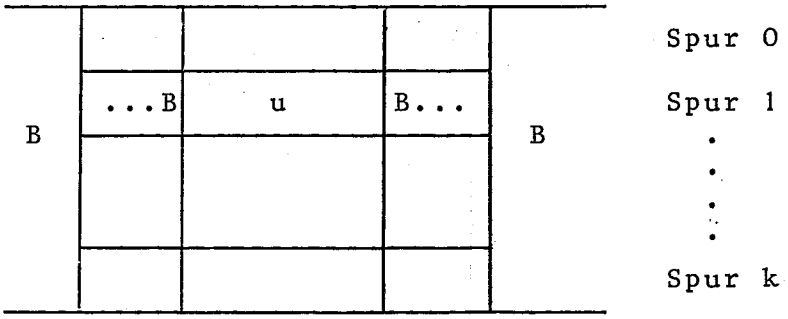
Ist $\delta(z,a)=(z'c,s)$ mit $s \in \{L,R,N\}^k$ und $s_i \neq N$, so bewegt $M^i_{[z,a]}$ den Kopf zum linken Ende der Inschrift von Spur i, verhält sich dann auf Spur i wie die Maschine "R-shift" falls $s_i=L$ bzw. wie "L-shift" falls $s_i=R$, und bewegt schließlich den Kopf zu Feld 0

zurück (die Maschine erkennt Feld 0 an dem von B verschiedenen Inhalt von Spur 0). Ist s_i=N, tut $M^i_{[z,a]}$ überhaupt nichts.

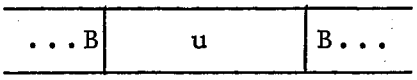
Der Endzustand jeder Maschine $M_{[z,a]}$ ist mit dem Startzustand von M_2 identifiziert. Das Durchlaufen eines Zyklus $M_2 \to M_{[z,a]} \to M_2$ simuliert einen Schritt von M; (*) bleibt offenbar für i erfüllt, falls (*) für i-1 erfüllt war.

Durch Induktion über i folgt, daß insbesondere nach der letzten Rechnung von M_2, d.h. wenn M_2 in den Zustand z_{e2} geht, Spur 1 des Bandes von M' die Inschrift von Band 1 von M am Ende der Rechnung von M enthält.

M_3 erzeugt aus Inschriften der Form



eine Inschrift



und hält mit dem Kopf auf dem ersten Zeichen von u.

M' simuliert offenbar M, ist O(s(n))-bandbeschränkt und regulär. Die Maschinen $M_{[z,a]}$ sind alle O(n)-zeitbeschränkt, die Maschinen M_1 und M_3 können leicht O(n)-zeitbeschränkt konstruiert werden. M_2 ist O(1)-zeitbeschränkt. Angesetzt auf Eingabe w mit |w|=n macht M_1 O(n) Schritte. M' erzeugt nie Inschriften der Länge > O(s(n)). t(n) Zyklen $M_2 \to M_{[z,a]}$ dauern deshalb höchstens O(t(n)·s(n)) Schritte. Angesetzt auf eine Inschrift der Länge O(s(n)) macht M_3 O(s(n)) Schritte. Man erhält als Gesamtschrittzahl

$$O(n)+O(t(n)s(n))+O(s(n)) = O(t(n)s(n)). \quad \square$$

Zusammen mit Satz 1.34 folgt:

Korollar 1.39: Für $t(n) \geq n$ kann jede s(n)-bandbeschränkte und t(n)-zeitbeschränkte k-Band-Turingmaschine von einer O(s(n))-bandbeschränkten und $O(t^2(n))$-zeitbeschränkten 1-Band-Turing-

maschine mit einseitig unbegrenztem Band simuliert werden.

Beweis: Simulation von k Bändern durch eines führt zu einer O(s(n))-bandbeschränkten und O(t(n)s(n))-zeitbeschränkten Maschine. Simulation dieser Maschine durch eine Maschine mit einseitig unbegrenztem Band liefert eine $O(t(n)s(n)+s^2(n))$-zeitbeschränkte Maschine. Da eine Maschine in t(n) Schritten nicht mehr als O(t(n)) Felder besuchen kann, folgt $s(n) \leq O(t(n))$ und damit die Behauptung. □

Korollar 1.4o: Jede turingberechenbare Funktion, insbesondere jede μ-rekursive Funktion, kann schon von einer regulären 1-Band-Turingmaschine berechnet werden.

Wir bemerken, daß in allen oben beschriebenen Simulationen die simulierende Maschine die simulierte Maschine Schritt-für-Schritt simuliert hat. Wie wir später sehen werden, ist dies nicht die einzige Art zu simulieren.

Bisher haben wir Maschinen meistens explizit angegeben und dabei eine ganze Reihe von Methoden zur Konstruktion solcher Maschinen sowie zur Analyse ihrer Laufzeit vorgestellt. Später werden wir nur noch die wesentlichen Merkmale der Arbeitsweise von Maschinen beschreiben. Mit den bisher entwickelten Techniken sollten interessierte Leser mit mehr oder weniger Mühe alle Details einfüllen können.

1.3.6. Turingberechenbarkeit und μ-Rekursivität

Wir zeigen die Umkehrung von Satz 1:

Satz 1.41: Ist $f:\mathbb{N}^r \to \mathbb{N}$ turingberechenbar, so ist f auch μ-rekursiv.

Beweis: Wir geben zuerst eine Beweisskizze. Ist f turingberechenbar und $M=(Z,A,\delta,z_o,E)$ eine Turingmaschine, die f berechnet, dann muß man die Arbeitsweise von M durch μ-rekursive Funktionen nachahmen. Nach Korollar 1.4o kann vorausgesetzt werden, daß M eine reguläre 1-Bandmaschine ist.

Turingmaschinen operieren auf Zeichenreihen; μ-rekursive Funktionen operieren auf Zahlen. Man kodiert deshalb Zeichenreihen in Zahlen, d.h. man gibt eine injektive Funktion $\psi:\{ZUA\}^+ \to \mathbb{N}$ an. Wir wählen eine Kodierung mit der Eigenschaft, daß man einfache Operatoren auf Zeichenreihen durch einfache primitiv rekursive

Funktionen nachahmen kann.

Für Konfigurationen k von M bezeichne $\Delta(k)$ die Nachfolgekonfiguration von k, d.h. die eindeutige Konfiguration k', so daß $k \vdash k'$ gilt.
Kernstück des Beweises von Satz 1.41 ist

Lemma 1.42: Es gibt primitiv rekursive Funktionen $\tilde{\Delta}:\mathbb{N}\to\mathbb{N}$ und $END:\mathbb{N}\to\mathbb{N}$, so daß für alle Konfigurationen k von M, die mit je zwei B's beginnen und enden, gilt: $\tilde{\Delta}(\psi(k))=\psi(B\Delta(k)B)$ und

$$END(\psi(k)) = \begin{cases} 0 & \text{falls } k \text{ eine Endkonfiguration ist} \\ 1 & \text{sonst.} \end{cases}$$

Die Bedingung, daß die Konfiguration k mit 2 B's beginnen und enden soll, dient nur zur Ersparnis von Schreibarbeit, denn jede solche Konfiguration k ist von der Form ubzav mit $u,v\in A^+$, $b,a\in A$ und $z\in Z$. Die Konfiguration $\Delta(k)$ geht aus k deshalb durch Anwendung der ersten Regelgruppe von Definition 1.17 hervor, und man kann die übrigen Fälle ignorieren.

Da $\tilde{\Delta}$ primitiv rekursiv ist, ist auch die Funktion $D:\mathbb{N}^2\to\mathbb{N}$ definiert durch

$$D(0,x) = x$$
$$D(n+1,x) = \tilde{\Delta}(D(n,x))$$

primitiv rekursiv. D heißt Iteration von $\tilde{\Delta}$. Durch Induktion über n folgt sofort, daß $D(n,\psi(k))$ die Kodierung einer Konfiguration $B...Bk'B...B$ ist, wobei k' aus k durch n Schritte der Maschine M entsteht.

Die Funktion $T:\mathbb{N}\to\mathbb{N}$

$$T(x) = \begin{cases} \min\{i \mid D(i,x) \text{ ist Kodierung einer Endkonfiguration}\} & \\ \qquad\qquad \text{falls dieses existiert} & \\ \text{nicht definiert} \qquad\qquad \text{sonst} & \end{cases}$$

ist μ-rekursiv, denn

$$T=\mu g \text{ für } g(i,x)=END(D(i,x)).$$

$T(\psi(k))$ ist die Anzahl der Schritte, die M gestartet in Konfiguration k macht und $D(T(\psi(k)),\psi(k))$ ist die Kodierung einer Konfiguration $B...Bk_eB...B$, wobei k_e die Endfiguration ist, die M hierbei schließlich erreicht. Nun zeigt man

Lemma 1.43: Es gibt primitiv rekursive Funktionen $E:\mathbb{N}^r\to\mathbb{N}$ und

$f:\mathbb{N}\to\mathbb{N}$, so daß für alle $x_i, x\in\mathbb{N}$ und alle $z\in Z$ gilt:

$$E(x_1,\ldots,x_r)=\psi(BBz_o\,bin(x_1)\#\ldots\#bin(x_r)BB),$$

$$F(\psi(B\ldots Bzbin(x)B\ldots B))=x.$$

Die Behauptung des Satzes folgt, denn

$$f(x_1,\ldots,x_r)=F(D(T(E(x_1,\ldots,x_r)),E(x_1,\ldots,x_r))).$$

Um den Beweis vollständig zu machen, müssen wir noch die Kodierung ψ angeben und dann Lemma 1.42 und Lemma 1.43 beweisen.

Dezimalzahlen sind Zeichenreihen aus $\{0,\ldots,9\}^+$, die natürliche Zahlen darstellen. Umgekehrt interpretiert man eine Zeichenreihe $v=v_1\ldots v_s$, $v_i\in\{0,\ldots,9\}$ als die natürliche Zahl $\sum_{i=1}^{s} v_i 10^{s-i}$. Da v führende Nullen haben kann, ist diese Zuordnung von Zahlen zu Zeichenreihen nicht injektiv. Die Zuordnung ist jedoch injektiv, wenn man sich auf $v\in\{1,\ldots,9\}^+$ beschränkt.

Wir wollen einfache Operationen auf Zeichenreihen durch einfache Operationen auf den zugehörigen Zahlen nachahmen. Eine einfache Operation auf Zeichenreihen ist die Konkatenation, d.h. das Hintereinanderschreiben von Zeichenreihen. Konkateniert man etwa die Zeichenreihen 169 und 225, dann sind auch die zugehörigen Zahlen auf einfache Weise verknüpft, nämlich $169225=169\cdot 10^3+225$. Diese Betrachtungen motivieren die folgende Wahl der Kodierung.

Für $AUZ=\{a_1,\ldots,a_p\}$ wird $\psi:(aUZ)^+\to\mathbb{N}$ erklärt durch

$$\psi(a_i) = i, a_i\in AUZ$$

$$\psi(v_1\ldots v_s) = \sum_{i=1}^{s}\psi(v_i)(p+1)^{s-i},\ v_i\in AUZ.$$

$v_1\ldots v_s$ wird also interpretiert als Darstellung einer Zahl zur Basis p+1. Das Bild von $(AUZ)^+$ unter ψ sind gerade die Zahlen, in deren Darstellung zur Basis p+1 keine Null vorkommt.

Beispiel 1.44: $AUZ=\{0,1,B,\#,z_o,z_e\}$, $\psi(0)=1$, $\psi(1)=2$, $\psi(B)=3$, $\psi(\#)=4$, $\psi(z_o)=5$, $\psi(z_e)=6$. $\psi(0z_o1) = 1\cdot 7^2+5\cdot 7+2 = 86$.

Wir bilden zunächst einige einfache Operationen auf Zeichenreihen durch primitiv rekursive Funktionen nach.

Lemma 1.45: Es gibt primitiv rekursive Funktionen:

L, CONCAT, PREFIX, SUFFIX, FIRST, LAST, SELECT,

so daß für alle $b,c\in(AUZ)^+$ und $i\in\{1,\ldots,|b|\}$ gilt:

1) $L(\psi(b)) = |b|$
2) $CONCAT(\psi(b),\psi(c)) = \psi(bc)$
3) $PREFIX(\psi(b),i) = \psi(b_1 \ldots b_i)$
4) $SUFFIX(\psi(b),i) = \psi(b_i \ldots b_{|b|})$
5) $FIRST(\psi(b)) = \psi(b_1)$
6) $LAST(\psi(b)) = \psi(b_{|b|})$
7) $SELECT(\psi(b),i) = \psi(b_i)$

Beweis: Um die primitiv rekursiven Definitionen dieser Funktionen zu verstehen, stellt man sich am besten $AUZ=\{1,\ldots,9\}$ vor, weil dann $\psi(b)$ gerade die Zahl ist, deren Dezimaldarstellung b ist.

1) $L(x) = \min\{m \mid m \leq x \wedge (p+1)^m > x\}$
2) $CONCAT(x,y) = x(p+1)^{L(y)}+y$
3) $PREFIX(x,i) = x/(p+1)^{L(x)-i}$
4) $SUFFIX(x,i) = Rest(x,(p+1)^{L(x)-i+1})$
5) $FIRST(x) = PREFIX(x,1)$
6) $LAST(x) = SUFFIX(x,L(x))$
7) $SELECT(x,i) = FIRST(SUFFIX(x,i))$.

Zur Vereinfachung der Schreibweise kürzen wir SELECT(x,i) durch $x_{(i)}$, CONCAT(x,y) durch [x,y] und [[x,y],z] durch [x,y,z] ab.

Beweis von Lemma 1.42: Sei $\psi(Z)=\{\psi(z) \mid z \in Z\}$. Das Prädikat $x \in \psi(Z)$ ist nur für endlich viele x wahr und ist deshalb nach Korollar 1.7 primitiv rekursiv.

Sei $z(x)=\min\{i \mid i<x \wedge x_{(i)} \in \psi(Z)\}$. Für $x=\psi(uzv)$ mit $u,v \in A^*$ und $z \in Z$ ist $z(x)=|u|+1$.

Sei

$$u(x) = PREFIX(x,z(x) \dot{-} 2)$$
$$v(x) = SUFFIX(x,z(x)+2)$$
$$w(x) = [x_{z(x) \dot{-} 1}, x_{(z(x))}, x_{z(x)+1}].$$

Die Funktionen z,u,v und w sind alle primitiv rekursiv, und für $x=\psi(ubzav)$ mit $u,v \in A^+$, $b,a \in A$ und $z \in Z$ ist $u(x)=\psi(u)$, $v(x)=\psi(v)$ und $w(x)=\psi(bza)$.

Die nun folgende Definition der Funktion $\tilde{\delta}:\mathbb{N} \to \mathbb{N}$ ist genau der Definition der Relation "$\vdash$" nachgebildet:

Ist $x=\psi(bza)$ mit $b,a \in A$ und $z \in Z$, so ist

$$\tilde{\delta}(x) = \begin{cases} \psi(bcz') & \text{falls } \delta(z,a)=(z',c,R) \\ \psi(bz'c) & \text{falls } \delta(z,a)=(z',c,N) \\ \psi(z'bc) & \text{falls } \delta(z,a)=(z',c,L). \end{cases}$$

In allen anderen Fällen ist $\tilde{\delta}(x) = 0$.

$\tilde{\delta}(x)$ ist nur für endlich viele x von Null verschieden. Also ist nach Korollar 1.7 die Funktion $\tilde{\delta}$ primitiv rekursiv. Die gesuchten Funktionen sind

$$\tilde{\Delta}(x)=[B,[u(x),\tilde{\delta}(w(x)),v(x)],B]$$

und

$$\mathrm{END}(x) = \begin{cases} 0 & \text{falls } Z(x)\in\{\psi(e)\mid e\in E\} \\ 1 & \text{sonst.} \end{cases}$$ □

Beweis von Lemma 1.43: $E':\mathbb{N}^2\to\mathbb{N}$ wird definiert durch

$$E'(n,x) = (\mathrm{Rest}(x,2^n))/2^{n-1}.$$

E'(n,x) ist die n-te Stelle von hinten in der Binärdarstellung von x.

Ist $G:\mathbb{N}^{r+1}\to\mathbb{N}$ primitiv rekursiv, dann auch $H:\mathbb{N}^{r+1}\to\mathbb{N}$,

$$H(n,x) = \sum_{i=0}^{n} G(i,x), \quad x\in\mathbb{N}, x\in\mathbb{N}^r.$$

Denn $H(0,x)=G(0,x)$, $H(n+1,x)=G(n+1,x)+H(n,x)$.

Folglich ist

$$E''(n,x) = \sum_{i=1}^{n} \psi(E'(i,x))\cdot(p+1)^{i-1}$$

primitiv rekursiv. E''(L(x),x) ist aber gerade $\psi(\mathrm{bin}(x))$. Hieraus erhält man die Funktion E durch endlich viele Anwendungen der Funktion CONCAT.

Sei $F_1(x)=\mathrm{SUFFIX}(x,z(x)+1)$,
$F_2(x)=\min\{i\mid i\le L(x)\wedge x_{(i)}=\psi(B)\}$,
$F_3(x)=\mathrm{PREFIX}(x,F_2(x)\dot{-}1)$.

Ist $x=\psi(B\ldots Bz\ \mathrm{bin}(y)B\ldots B)$ mit $z\in Z$ und $y\in\mathbb{N}$, so ist

$$F_3(F_1(x)) = \psi(\mathrm{bin}(y)).$$

Ist $F_4(x)$ eine primitiv rekursive Funktion, so daß für alle $y\in\mathbb{N}$ $F_4(\psi(\mathrm{bin}(y))=y$ gilt, so leistet $F(x)=F_4(F_3(F_1(x))$ das Gewünschte. OBdA sei $\psi(0)=1$ und $\psi(1)=2$. Dann ist

$$F_4(x) = \sum_{i=1}^{L(x)} ((\mathrm{Rest}(x,(p+1)^i)/(p+1)^{i-1})-1)2^{i-1}.$$

Damit sind Lemma 1.42 und Satz 1.41 vollständig bewiesen. □□

Einige wichtige Folgerungen sind

Korollar 1.46: Eine Funktion $f:\mathbb{N}^r \to \mathbb{N}$ ist genau dann μ-rekursiv, wenn f turingberechenbar ist.

Korollar 1.47 (Kleene'sche Normalform): Ist $f:\mathbb{N}^r \to \mathbb{N}$ μ-rekursiv, dann gibt es primitiv rekursive Funktionen $p:\mathbb{N}^r \to \mathbb{N}$ und $q:\mathbb{N}^{r+1} \to \mathbb{N}$, so daß $f(x)=q(\mu p(x),x)$.

Beweis: Ist f μ-rekursiv, so ist f turingberechenbar. Aus der Turingmaschine, die f berechnet, gewinnt man nach dem Beweis von Satz 1.43 eine Ableitung der Funktion f, in der der μ-Operator nur einmal, nämlich bei der Definition von T benutzt wird.

Korollar 1.48: Eine Funktion $f:\mathbb{N}^r \to \mathbb{N}$ ist genau dann primitiv rekursiv, wenn es eine primitiv rekursive Funktion $t:\mathbb{N} \to \mathbb{N}$ und eine t(n)-zeitbeschränkte Turingmaschine M gibt, die f berechnet.

Der Beweis von Korollar 1.48 ist eine sehr lehrreiche Übung. □

1.3.7. Die Church'sche These

Die Klasse der μ-rekursiven Funktionen und die Klasse der turingberechenbaren Funktionen wurden auf ganz verschiedene Art definiert, und doch haben sich beide Klassen als identisch erwiesen. In den dreißiger Jahren und auch später wurden zahlreiche weitere Versuche unternommen, die Klasse der im intuitiven Sinn berechenbaren Funktionen formal zu charakterisieren. In jedem Fall konnte mit Techniken ähnlich denen der Beweise von Satz 1.29 und Satz 1.43 gezeigt werden, daß die gleiche Klasse von Funktionen definiert wurde. In diesem Sinn erwiesen sich also alle diese Formalisierungen als äquivalent.

Wir werden später zeigen, daß die Klasse der turingberechenbaren Funktionen auch mit der Klasse der auf (passend idealisierten) modernen Rechenanlagen berechenbaren Funktionen übereinstimmt.

Church postulierte schon 1936, daß die hierdurch charakterisierte Klasse von Funktionen mit der Klasse aller im intuitiven Sinn berechenbaren Funktionen identisch sei. Diese - wie schon mehrmals betont - nicht beweisbare Behauptung ist als die Church'sche These bekannt.

Ebenfalls in den dreißiger Jahren wurden zum ersten Mal für spezi-

fische Funktionen f Sätze der Form "f ist nicht μ-rekursiv" bewiesen. Formal besagt das nur, daß die Funktion f in einem gewissen willkürlichen Formalismus nicht ausgedrückt werden kann. Durch die Äquivalenzbeweise folgt, daß f auch in vielen anderen Formalismen nicht ausgedrückt werden kann.

Akzeptiert man die Church'sche These, so folgt die viel gewichtigere Aussage "f ist überhaupt nicht berechenbar".

Die Church'sche These wird heute allgemein akzeptiert.

Übungen

1. Man gebe eine reguläre 1-Band-Turingmaschine an, die angesetzt auf bin(n) mit bin(n$\dot{-}$1) auf dem Band hält.

2. Man gebe eine reguläre 1-Band-Turingmaschine an, die eine Bandinschrift $w\in\{0,1\}^+$ um 1 Feld nach links verschiebt.

3. Man gebe eine 1-Bandmaschine mit folgender Eigenschaft an: Wenn M mit dem Kopf auf einem beliebigen Feld des Bandes gestartet wird, hält M genau dann, wenn es auf dem Band ein von B verschiedenes Zeichen gibt.

4. a) Die Funktionen $f_1, f_2: \mathbb{N}^2 \to \mathbb{N}$, $f_1(x,y)=x\cdot y$, $f_2(x,y)=x^y$ sind turingberechenbar.

   b) Man gebe eine $O(n^2)$-zeitbeschränkte Mehrband-Turingmaschine an, die f_1 berechnet.

5. a) Man gebe eine $O(n^2)$-zeitbeschränkte 1-Bandmaschine M mit zwei Endzuständen z_+, z_- an, so daß für alle $w\in\{0,1\}^+$ gilt: Angesetzt auf w hält M im Zustand z_+ falls $w_1 \cdots w_{|w|} = w_{|w|} \cdots w_1$; andernfalls hält M im Zustand z_-.

   b) Man gebe eine $O(n)$-zeitbeschränkte 2-Bandmaschine an, die das gleiche tut.

6. a) Man gebe eine $O(n^2)$-zeitbeschränkte 1-Bandmaschine M mit zwei Endzuständen z_+, z_- an, so daß für alle $x,y\in\{0,1\}^+$ gilt: Angesetzt auf x#y hält M im Zustand z_+ falls $|y|\geq|x|$ und falls es einen Index $i\in\{1,\ldots,|y|-|x|\}$ gibt, so daß $x=y_i \cdots y_{i+|x|}$; andernfalls hält M im Zustand z_-.

   b) Man gebe eine $O(n)$-zeitbeschränkte 2-Bandmaschine an, die das gleiche tut.

7. Jede t(n)-zeitbeschränkte 1-Band-Turingmaschine M kann von einer t(n)-zeitbeschränkten 1-Band-Turingmaschine M', die in jedem Schritt den Kopf bewegt, simuliert werden.

8. Jede Turingmaschine M kann von einer regulären Turingmaschine M'="M_1;M_2" simuliert werden, wobei M_1 nie das Leerzeichen druckt und M_2 (2n+2)-zeitbeschränkt ist.

9. Man gebe die Funktionen p und q aus Korollar 1.47 an.

1o. Man beweise Korollar 1.48.

2. Nicht berechenbare Funktionen

Wir haben im ersten Kapitel, ausgehend von der Klasse der μ-rekursiven bzw. turingberechenbaren Funktionen, Evidenz dafür geliefert, daß diese Klasse alle berechenbaren Funktionen umfaßt. In diesem Kapitel weisen wir von einer Reihe von Problemen nach, daß sie nicht durch Algorithmen lösbar sind. Im folgenden meinen wir mit berechenbaren Funktionen immer turingberechenbare Funktionen, und für Maschinen M mit Alphabet A bezeichnet wieder $f_M: A^* \to A^*$ die von Maschine M berechnete Funktion im Sinne von Definition 1.28.

2.1. Halteproblem und Reduzierbarkeit

2.1.1. Entscheidbarkeit

Wir interessieren uns im folgenden häufig für Entscheidungsprobleme, d.h. wir geben eine Menge L an und fragen nach Algorithmen, die bei Eingabe x entscheiden, ob $x \in L$ oder $x \notin L$ gilt. Einen solchen Algorithmus nennt man auch ein Entscheidungsverfahren für L.

Definition 2.1: Ist $L \subset A^*$, so heißt

$$\chi_L : A^* \to \{0,1\},$$

$$\chi_L(x) = \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{sonst} \end{cases}$$

die charakteristische Funktion von L. L heißt entscheidbar, wenn χ_L eine totale berechenbare Funktion ist.

In analoger Weise kann man die Entscheidbarkeit von Mengen $L \subset \mathbb{N}$ bzw. $L \subset \mathbb{N}^r$ erklären. Beispielsweise sind die Menge aller geraden Zahlen und die Menge aller Primzahlen entscheidbar.

Ist $L \subseteq A^*$, so heißt $\overline{L} := \{w \mid w \in A^* \setminus L\}$ das Komplement von L. Es ist klar, daß L genau dann entscheidbar ist, wenn $\overline{L}$ entscheidbar ist.

Da es nur abzählbar viele Entscheidungsverfahren, aber überzählbar viele Teilmengen $L \subset A^*$ bzw. $L \subset \mathbb{N}$ gibt, muß es nichtentscheidbare Mengen geben(vgl. Satz 1.1). Wir interessieren uns nun dafür, wie man die Nichtentscheidbarkeit von speziellen Mengen nachweist.

2.1.2. Gödelisierung von Turingmaschinen

In allen bekannten Beweisen von der Nichtentscheidbarkeit spezieller Mengen spielen Mengen, die durch Eigenschaften von Turingmaschinen definiert sind, eine zentrale Rolle. Man kodiert Turingmaschinen in $\{0,1\}^*$ und kann damit die Beschreibung von Turingmaschinen selbst zum Objekt von Berechnungen machen. Wie man speziell kodiert, ist weitgehend willkürlich, solange man nur aus der Kodierung in einfacher Weise auf die Arbeitsweise der kodierten Maschine schließen kann. Wir geben eine spezielle solche Kodierung für 1-Band-Maschinen an.

Sei $M=\{Z,A,\delta,q,E\}$ eine 1-Bandmaschine. Wir numerieren die Zustände und Alphabetzeichen von M von Null an durch, also $Z=\{z_o,\ldots,z_t\}$ und $A=\{a_o,\ldots,a_r\}$. Wir werden später das i-te Alphabetzeichen bzw. den i-ten Zustand durch bin(i) kodieren; damit wir dabei nicht die Information verlieren, welches der Anfangszustand q und welches die ausgezeichneten Alphabetzeichen 0,1,#,B sind, vereinbaren wir, daß immer $z_o=q$, $a_o=0$, $a_1=1$, $a_2=\#$ und $a_3=B$ gelten soll.

Nun ordnet man jeder Regel $\delta(z_i,a_j)=(z_k,a_\ell,s)$ das Wort

$$w_{ijk\ell m} = \#\#\text{bin}(i)\#\text{bin}(j)\#\text{bin}(k)\#\text{bin}(\ell)\#\text{bin}(m)$$

zu mit:

$$m = \begin{cases} 0 & \text{falls } s=N \\ 1 & \text{falls } s=R \\ 2 & \text{falls } s=L. \end{cases}$$

Schreibt man alle diese Worte in beliebiger Reihenfolge hintereinander, so erhält man eine Kodierung von M in $\{0,1,\#\}^*$.

Beispiel 2.2: Sei $M=(Z,A,\delta,z_o,E)$, $Z=\{z_o,z_1\}$, $A=\{B,0,1,\#\}$,

(1) $\delta(z_o,0) = (z_o,1,R)$,
(2) $\delta(z_o,1) = (z_o,0,R)$,
(3) $\delta(z_o,\#) = (z_o,\#,R)$,
(4) $\delta(z_o,B) = (z_1,\#,N)$.

Man kodiert

(1) als ##0#0#0#1#1
(2) als ##0#1#0#0#1
(3) als ##0#10#0#10#1
(4) als ##0#11#1#10#0.

Wir definieren eine Abbildung $h:\{0,1,\#\}^*\to\{0,1\}^*$ durch $h(0)=00$,

$h(1)=01$, $h(\#)=11$ und

(*) $h(w_1 \ldots w_s) = h(w_1) \ldots h(w_s), w_i \in \{0,1,\#\}$;

also beispielsweise $h(0\ 1)=001101$. Eine Abbildung, für die (*) gilt, nennt man auch (Monoid-)Homomorphismus.

Ist u' eine Kodierung einer Maschine M in $\{0,1,\#\}^*$, so ist $u=h(u')$ eine Kodierung der Maschine M in $\{0,1\}^*$ und heißt eine Gödelisierung von M. Die natürliche Zahl mit Binärdarstellung u kodiert ebenfalls die Maschine M und heißt eine Gödelnummer von M. In ähnlicher Weise kann man Mehrbandmaschinen gödelisieren.

Je nachdem, wie man die vom Anfangszustand verschiedenen Zustände und die von 0,1,#,B verschiedenen Alphabetzeichen einer Maschine numeriert und in welcher Reihenfolge man die Worte $w_{ijk\ell m}$ hintereinanderschreibt, erhält man verschiedene Gödelisierungen der gleichen Maschine. Es können auch verschiedene Maschinen gemeinsame Gödelisierungen haben; solche Maschinen gehen aber durch Umbenennung der von Startzustand verschiedenen Zustände und der von 0,1,#,B verschiedenen Alphabetzeichen auseinander hervor. Deshalb gilt:

Satz 2.3: Haben die Maschinen M und M' eine gemeinsame Gödelisierung und ist $v \in \{0,1,\#\}^*$, so hält M angesetzt auf v genau dann, wenn M' angesetzt auf v hält. Ist $f_M(v) \in \{0,1,\#\}^*$, so ist $f_M(v)=f_{M'}(v)$.

Weiter gilt offenbar

Lemma 2.4: Eine Zeichenreihe $u \in \{0,1\}^*$ ist genau dann Gödelisierung einer 1-Band-Turingmaschine, falls $u=h(u')$ gilt für eine Zeichenreihe $u' \in \{0,1,\#\}^*$, die i) und ii) erfüllt:

i) u' ist aus lauter Worten $w_{ijk\ell m}$ zusammengesetzt.

ii) Kommen $w_{ijk\ell m}$ und $w_{i'j'k'\ell'm'}$ in u' vor, so ist $(i,j) \neq (i',j')$.

Wir geben nun eine Art Umkehrung der Gödelisierung an, indem wir jeder Zeichenreihe u aus $\{0,1\}^*$ eine Maschine M_u zuordnen: Ist u nicht Gödelisierung einer 1-Band-Turingmaschine, so sei M_u eine Maschine, die für keine Eingabe hält, andernfalls sei $u=h(u')$, und u' erfülle die Bedingungen i) und ii) aus Lemma 2.4.
Wir setzen $M_u=(Z,A,\delta,z_o,E)$, $Z=\{z_o,\ldots,z_t\}$, $A=\{a_o,\ldots,a_r\}$ mit

$t = \max\{i,k \mid \exists j,\ell,m, \text{ so daß } w_{ijk\ell m} \text{ in u' vorkommt}\}$,

$v = \max\{j,\ell \mid \exists i,k,m, \text{ so daß } w_{ijk\ell m} \text{ in u' vorkommt}\}$,

$$\delta(z_i,a_j) = \begin{cases} (z_k,a_\ell,N) & \text{falls } w_{ijk\ell 0} \text{ in u' vorkommt} \\ (z_k,a_\ell,R) & \text{falls } w_{ijk\ell 1} \text{ in u' vorkommt} \\ (z_k,a_\ell,L) & \text{falls } w_{ijk\ell 2} \text{ in u' vorkommt} \end{cases}$$

$E = \{i \mid \sim\exists jk\ell m, \text{ so daß } w_{ijk\ell m} \text{ in u' vorkommt}\}$.

Ist u Gödelisierung irgendeiner 1-Band-Turingmaschine, so ist u offenbar auch eine Gödelisierung von M_u. Aus Satz 2.3 folgt

Korollar 2.5: Ist M eine Turingmaschine, u eine Gödelisierung von M und $v \in \{0,1,\#\}^*$, so hält M angesetzt auf v genau dann, wenn M_u angesetzt auf v hält. Ist $f_M(v) \in \{0,1,\#\}^*$, so ist $f_M(v) = f_{M_u}(v)$.

2.1.3. Das spezielle Halteproblem

Wir definieren eine spezielle Menge und zeigen, daß sie nicht entscheidbar ist. Alle weiteren Nichtentscheidbarkeitssätze, die wir zeigen, sind Folgerungen der Nichtentscheidbarkeit dieser Menge.

Definition 2.6: Die Menge

$$K = \{u \mid u \in \{0,1\}^*,\ M_u \text{ angesetzt auf u hält}\}$$

heißt das spezielle Halteproblem.

Satz 2.7: Das spezielle Halteproblem ist nicht entscheidbar. Der Beweis wird durch Widerspruch geführt und benutzt wieder ein Diagonalverfahren. Ist die charakteristische Funktion χ_K des speziellen Halteproblems eine totale turingberechenbare Funktion, so gibt es nach Satz 1.37 schon eine 1-Band-Turingmaschine Q, die χ_K berechnet, d.h. Q hält für alle Eingaben u und

$$f_Q(u) = \begin{cases} 1 & \text{falls } M_u \text{ angesetzt auf u hält} \\ 0 & \text{sonst.} \end{cases}$$

Aus der Maschine Q erhält man leicht eine Maschine R mit

$$f_R(u) = \begin{cases} 1 & \text{falls } f_Q(u)=0 \\ \text{nicht definiert} & \text{falls } f_Q(u)=1 \end{cases}$$

Ist r eine Gödelisierung von R, so folgt der Widerspruch:

R angesetzt auf r hält $\Longleftrightarrow$

$f_Q(r) = 0 \Longleftrightarrow$

M_r angesetzt auf r hält nicht $\Longleftrightarrow$

R angesetzt auf r hält nicht. □

2.1.4. Reduzierbarkeit

Reduzierbarkeit ist ein einfaches, aber außerordentlich wichtiges und vielseitig verwendbares Konzept, um Nichtentscheidbarkeitsaussagen herzuleiten.

Definition 2.8: Sind $L,L' \subset A^*$ und gibt es eine totale berechenbare Funktion $f:A^* \to A^*$, so daß für alle $w \in A^*$ gilt:
$w \in L \Longleftrightarrow f(w) \in L'$, dann heißt L (vermittels f) reduzierbar auf L' ($L \leq L'$).

Zwei unmittelbare Folgerungen sind

Satz 2.9: a) Die Relation $\leq$ ist transitiv.
b) Ist $L \leq L'$ und L ist nicht entscheidbar, so ist auch L' nicht entscheidbar.

Beweis: a) Ist $L \leq L'$ vermittels der Funktion f und $L' \leq L''$ vermittels der Funktion g, so ist $L \leq L''$ vermittels gof.
b) Ist $L \leq L'$ vermittels der Funktion f, so ist $\chi_L = \chi_{L'}$of. Wäre also $\chi_{L'}$ eine totale berechenbare Funktion, dann auch χ_L. □

Beispiel 2.1o: Die Menge

$$H = \{u\ v \mid u,v \in \{0,1\}^*,\ M_u \text{ angesetzt auf } v \text{ hält}\}$$

heißt Halteproblem. Es gilt $K \leq H$ vermittels der Funktion $f:\{0,1,\#\}^* \to \{0,1,\#\}^*, f(u) = u\#u$.

Korollar 2.11: Das Halteproblem ist nicht entscheidbar.

Beispiel 2.12: Sei $H_o = \{u \mid u \in \{0,1\}^*, M_u$ angesetzt auf leeres Band hält$\}$. H_o heißt das Halteproblem für Turingmaschinen mit leerem Band. Wir zeigen die Nichtentscheidbarkeit von H_o, indem wir H auf H_o reduzieren.

Jedem Wort $u\#v, u,v \in \{0,1\}^*$ wird eine Turingmaschine M(u,v) zugeordnet, die angesetzt auf das leere Band zunächst das Wort u druckt und sich dann wie M_v verhält (in der Schreibweise von

Kapitel 1.3: M(u,v) = "Drucke v; M_u"). Man kann eine Maschine Q angeben, die angesetzt auf u#v eine Gödelisierung von M(u,v) erzeugt, und M(u,v) hält angesetzt auf leeres Band genau dann, wenn M_u angesetzt auf v hält; also gilt $H \leq H_o$ vermittels f_Q.

Wir zeigen jetzt einen sehr allgemeinen Nichtentscheidbarkeitssatz, der grob gesprochen besagt, daß es kein allgemeines Verfahren gibt, einer Turingmaschine irgendeine Eigenschaft der von ihr berechneten Funktion anzusehen.

Satz 2.13 (Rice): Sei R die Menge aller turingberechenbaren Funktionen und $S \subset R$. Dann gilt: Die Menge $C(S)=\{u \mid f_{M_u} \in S\}$ ist genau dann entscheidbar, wenn $S=\emptyset$ oder $S=R$.

Beweis: Sei $S \neq \emptyset$ und $S \neq R$. Dann ist Ω, die überall nicht definierte Funktion, in S oder nicht in S.

Ist $\Omega \in S$, so gibt es eine berechenbare Funktion $q \notin S$, da $S \neq R$. Sei Q eine Turingmaschine, die q berechnet und $w \in \{0,1\}^*$ beliebig. w wird eine Maschine M(w) zugeordnet, die folgendermaßen arbeitet: Angesetzt auf $y \in \{0,1\}^*$ ignoriert M(w) zunächst die Eingabe und verhält sich wie M_w angesetzt auf leeres Band. Hält M_w, so verhält sich M(w) wie Q angesetzt auf y. Hält M_w angesetzt auf leeres Band, so ist $f_{M(w)}=q \notin S$. Andernfalls ist $f_{M(w)}=\Omega \in S$. Da man überdies aus w effektiv eine Gödelisierung von M(w) berechnen kann, folgt $\overline{H}_o \leq C(S)$. Im Fall $\Omega \in S$ zeigt man ebenso $H_o \leq C(S)$. □

Beispiel 2.14: Die Menge $\{u \mid M_u$ berechnet eine konstante Funktion$\}$ ist nicht entscheidbar.

2.1.5. Rekursiv aufzählbare Mengen

Definition 2.15: Eine Menge $L \subset A^*$ heißt rekursiv aufzählbar, wenn $L=\emptyset$ oder wenn es eine totale berechenbare Funktion $f:A^* \to A^*$ mit $L=f(A^*)$ gibt. Wir sagen in diesem Fall: f zählt L auf.

Ähnlich definiert man rekursiv aufzählbare Mengen $L \subset \mathbb{N}$ bzw. $L \subset \mathbb{N}^r$. Die Definition rekursiv aufzählbarer Mengen gleicht stark der Definition von höchstens abzählbar unendlichen Mengen: eine Menge L ist endlich oder abzählbar genau dann, wenn es eine Surjektion $f:\mathbb{N} \to L$ gibt. Damit L rekursiv aufzählbar ist, muß f zusätzlich total berechenbar sein. Der folgende Satz stellt einen Zusammenhang zwischen entscheidbaren und rekursiv aufzählbaren Mengen her.

Satz 2.16: Eine Menge L ist entscheidbar genau dann, wenn L und $\overline{L}$ rekursiv aufzählbar sind. Insbesondere folgt, daß jede entscheidbare Menge rekursiv aufzählbar ist.

Beweis von Satz 1.12: Sei $L \subset A^*$ entscheidbar. Ist $L=\emptyset$ oder $L=A^*$, so ist nichts zu zeigen. Sei also $L \neq \emptyset, A^*$ und $w \in L, v \notin L$.

Die Funktionen f und g seien definiert durch

$$f(x) = \begin{cases} x & \text{falls } \chi_L(x)=1 \\ w & \text{sonst} \end{cases}$$

$$g(x) = \begin{cases} x & \text{falls } \chi_L(x)=0 \\ v & \text{sonst} \end{cases}$$

Da χ_L nach Voraussetzung total berechenbar ist, zählen f und g die Mengen L und $\overline{L}$ auf.

Seien andererseits f und g totale berechenbare Funktionen, die L und $\overline{L}$ aufzählen. Wir konstruieren hieraus ein Verfahren, das L entscheidet. Das Verfahren arbeitet in "Stufen" und startet in Stufe 1. In Stufe i werden die folgenden Mengen berechnet:

$$A^i = \{w \mid w \in A^*, |w|=i\}$$

$$B^i = \{f(w) \mid w \in A^i\}$$

$$C^i = \{g(w) \mid w \in A^i\}.$$

$y \in A^*$ sei die Eingabe. Ist $y \in B^i$, folgt $y \in L$; ist $y \in C^i$, folgt $y \in \overline{L}$. In beiden Fällen bricht das Verfahren nach Stufe i ab. Ist $y \notin B^i \cup C^i$, wird zur Stufe i+1 übergegangen.

Da $L = \bigcup_i B^i$ und $\overline{L} = \bigcup_i C^i$, gibt es für jede Eingabe $y \in A^*$ ein i, so daß $y \in B^i$ oder ein j, so daß $y \in C^j$. Also bricht das Verfahren für jede Eingabe y nach endlich vielen Schritten ab. □

Dieses und ähnliche Verfahren, mehrere Mengen parallel aufzuzählen, nennt man auch "Dovetailing" (dovetail - englisch: Schwalbenschwanz).

2.1.6. Universelle Turingmaschinen

Wir zeigen, daß es Turingmaschinen U gibt, die alle anderen Turingmaschinen M in folgendem Sinn simulieren können: bei Eingabe u#v mit $u \in \{0,1\}^*$ und $v \in \{0,1,\#\}^*$ liefert U im wesentlichen

$f_{M_u}(v)$. Solche Maschinen nennt man universell. Baut man physikalisch eine universelle Maschine, dann kann man sie so programmieren, daß sie sich wie jede andere Maschine verhält; d.h. man kann jede berechenbare Funktion programmieren. Beispielsweise sind elektronische Rechenanlagen universelle Maschinen (potentiell beliebig großer Speicher vorausgesetzt).

Satz 2.17: Es gibt eine universelle 1-Band-Turingmaschine U, so daß für alle $u\in\{0,1\}^*$ und $v\in\{0,1,\#\}^*$ gilt:

a) U angesetzt auf u#v hält genau dann, wenn M_u angesetzt auf v hält. Ist $f_{M_u}(v)\in\{0,1,\#\}^*$, so ist $f_U(u\#v)=f_{M_u}(v)$.

b) Benutzt M_u angesetzt auf v s Felder, so benutzt U angesetzt auf u#v $O(|u|\cdot s)$ Felder.

c) Macht M_u angesetzt auf v t Schritte und ist $|f_{M_u}(v)|=\ell$, so macht U angesetzt auf u#v $O(|u|^2\cdot(|v|^2+t+\ell^2))$ Schritte.

Beweis: Angesetzt auf u#v testet die Maschine U zunächst nach dem Kriterium von Lemma 2.4, ob u Gödelisierung einer 1-Band-Turingmaschine ist. Verläuft dieser Test negativ, hält U nicht, andernfalls simuliert U die Maschine M_u angesetzt auf v Schritt für Schritt.

Sei $M_u=(Z,A,\delta,z_0,E)$, $A=\{a_0,\ldots,a_r\}$ und u' mit h(u')=u die Kodierung von M_u in $\{0,1,\#\}^*$. Die Maschine U hat ein festes Alphabet, muß aber Maschinen mit beliebig großem Alphabet simulieren können. Deshalb werden die Alphabetzeichen aus A als Zeichenreihen aus $\{0,1\}^*$ kodiert, und zwar a_i als $g(a_i)$=Binärdarstellung von i mit Länge $\lfloor\log r\rfloor+1$. Das Band von U hat 3 Spuren; U simuliert M_u so, daß für alle i gilt:

(*) Hat M_u angesetzt auf v nach i Schritten Zustand z_j, Bandinschrift $a_{i_1}\ldots a_{i_s}$ und steht der Kopf von M_u auf Zeichen a_{i_q}, so gilt nach der Simulation dieses Schritts durch U: die Inschrift von Spur 1 ist $\#g(i_1)\#\ldots\#g(i_s)\#$, die Inschrift von Spur 2 ist ##bin(j), die Inschrift von Spur 3 ist u', und die letzten Zeichen von $g(i_q)$ auf Spur 1, bin(j) auf Spur 2 und u' auf Spur 3 stehen auf dem gleichen Feld des Bandes von U.

Beispiel 2.18: Ist M_u die Maschine aus Beispiel 2.2, so wird die Startkonfiguration von M_u angesetzt auf 0#1 dargestellt als

#00#10#01#	Spur 1
##0	Spur 2
u'	Spur 3

Ist $v=a_{j_1}\ldots a_{j_{|v|}}$, so wird zu Beginn der Simulation v durch $\#g(j_1)\#\ldots\#g(j_{|v|})\#$ ersetzt und auf Spur 1 kopiert, auf Spur 2 wird die Zeichenreihe ##0 erzeugt, u' wird auf Spur 3 geschrieben, und die Inschriften von Spur 2 und 3 werden so zurechtgerückt, daß die letzten Zeichen von $bin(j_1)$ auf Spur 1, ##0 auf Spur 2 und u' auf Spur 3 das gleiche Feld belegen. Damit gilt (*) für i=0.

Sei die Bandinschrift von U wie in (*) beschrieben; zur Simulation eines Schritts von M_u geschieht folgendes:

i) Aus $g(i_q)$ erhält man durch Streichen führender Nullen $bin(i_q)$; die Inschrift von Spur 2 wird um $\#bin(i_q)\#$ verlängert. Im obigen Beispiel wird Spur 2 zu ##0#0#.

ii) Durch Verschieben der neuen Inschrift von Spur 2 entlang u' wird in u' eine Teilzeichenreihe gesucht, die mit der Inschrift von Spur 2 übereinstimmt; da u' Kodierung einer Maschine ist, folgt auf diese Teilzeichenreihe in u' eine Teilzeichenreihe der Form bin(k)#bin(ℓ)#bin(m)#. Die Inschrift von Spur 2 wird durch ##bin(k)#bin(ℓ)#bin(m)# ersetzt. Im Beispiel wird Spur 2 zu ##0#1#1#.

iii) Die Inschrift von Spur 2 wird verschoben, bis das letzte Zeichen von bin(ℓ) und von u' das gleiche Feld belegen. Im Beispiel erhält man

#00#10#01#	Spur 1
##0#1#1#	Spur 2
u'	Spur 3

iv) bin(ℓ) wird auf Spur 1 kopiert, alle Felder auf Spur 1 links von bin(ℓ) bis zum nächsten #-Zeichen werden mit 0 bedruckt. Auf Spur 2 wird #bin(ℓ) gelöscht und ##bin(k) an #bin(m)# herangeschoben. Im Beispiel erhält man

#01#10#01#	Spur 1
##0#1#	Spur 2
u'	Spur 3

v) Ist m=0, wird bloß #bin(m)# gelöscht. Ist m=1 bzw. m=2, wird #bin(m)# gelöscht und die Inschriften von Spur 2 und Spur 3 werden so lange nach rechts bzw. links verschoben, bis die letzten Zeichen dieser Inschriften ein Feld links vom nächsten #-Zeichen auf Spur 1 stehen. Im Beispiel erhält man

#01#10#01#	Spur 1
##0	Spur 2
u'	Spur 3

Damit ist (*) nach der Simulation eines Schritts erfüllt, falls (*) vorher erfüllt war. Die Simulation bricht ab, wenn in ii) die Inschrift von Spur 2 nicht in u' gefunden werden kann. Ist $f_{M_u}(v) \in \{0,1,\#\}^*$, so kann dieses leicht am Ende der Simulation aus der Inschrift von Spur 1 gewonnen werden.

Benutzt M_u angesetzt auf v s Felder, so benutzt U angesetzt auf u#v höchstens $|u|+(\lfloor \log r \rfloor+2)s+1$ Felder. Wegen $\lfloor \log r \rfloor \leq |u|$ folgt Behauptung b).

Der Test, ob u Gödelisierung einer Turingmaschine ist, kann in Zeit $O(|u|^2)$ durchgeführt werden. Beim Präparieren der Eingabe werden alle Zeichen der Zeichenreihe v durch Zeichenreihen der Länge $\lfloor \log r \rfloor+1 \leq |u|$ ersetzt. Dies kann in Zeit $O((|u|\cdot|v|)^2)$ durchgeführt werden. Die meiste Zeit bei der Simulation eines Schritts erfordert das Verschieben der Inschriften der Spuren 2 und 3. Die Länge dieser Inschriften ist durch $|u|$ beschränkt, und sie werden höchstens um $O(|u|)$ Felder verschoben. Zeit $O(|u|^2)$ genügt daher zur Simulation eines Schritts. Am Ende der Simulation hat die Inschrift von Spur 1 Länge $\ell\cdot(\lfloor \log r \rfloor+2)+1$. Um aus der Inschrift von Spur 1 das Ergebnis der Rechnung von M_u angesetzt auf v zurückzugewinnen, müssen Zeichenreihen der Länge $\lfloor \log r \rfloor+1$ durch einzelne Zeichen ersetzt und die hierdurch entstehenden Fragmente der Bandinschrift zusammengeschoben werden. Das ist in Zeit $O((|u|\cdot\ell)^2)$ möglich. Man erhält also insgesamt als Laufzeit

$O((|u|\cdot|v|)^2)+O(t\cdot|u|^2)+O((|u|\cdot\ell)^2)$. □

Mit genau den gleichen Methoden zeigt man:

Satz 2.19: Sei $k\geq 2$, und M_u bezeichne die k-Bandmaschine mit Gödelisierung u. Es gibt eine universelle k-Band-Turingmaschine U, so daß für alle $u\in\{0,1\}^*$ und $v\in\{0,1,\#\}^*$ gilt:

a) U angesetzt auf u#v hält genau dann, wenn M_u angesetzt auf v hält. Ist $f_{M_u}(v)\in\{0,1,\#\}^*$, so ist $f_U(u\#v)=f_{M_u}(v)$.

b) Benutzt M_u angesetzt auf v s Felder, so benutzt U angesetzt auf v $O(|u|\cdot s)$ Felder.

c) Macht M_u angesetzt auf v t Schritte und ist $t\geq|v|$, so macht U angesetzt auf u#v $O(|u|^2\cdot t)$ Schritte.

Beweis: Band 1 von U hat 3 Spuren, die Bänder 2,...,k von U haben je 2 Spuren. Für $1\leq i\leq k$ wird auf Band i von U Band i von M_u simuliert. Die Spuren 1 eines jeden Bandes von U sowie die Spuren 2 und 3 von Band 1 von U spielen die gleiche Rolle wie im Fall k=1. Die zweiten Spuren der Bänder 2,...,k von U dienen zum Markieren der Kopfpositionen von M_u. Mit Hilfe von 2 Bändern kann die Eingabe schon in Zeit $O(|u|\cdot|v|)$ präpariert werden. Ist $|f_{M_u}(v)|=\ell$, so hat am Ende der Simulation die Inschrift der ersten Spur von Band 1 von U höchstens die Länge $|u|\cdot\ell$. Hieraus kann wieder mit Hilfe von 2 Bändern das Ergebnis der Rechnung von M_u angesetzt auf v schon in Zeit $O(|u|\cdot\ell)$ gewonnen werden. Man erhält als Gesamtlaufzeit $O(|u|\cdot|v|+|u|^2\cdot t+|u|\cdot\ell)$. Wegen $|v|\leq t$ und $\ell\leq|v|+t$ folgt die Behauptung. □

2.1.7. Nicht rekursiv aufzählbare Mengen

Mit Hilfe universeller Maschinen kann man zeigen, daß es rekursiv aufzählbare nichtentscheidbare Mengen gibt, und explizit nicht rekursiv aufzählbare Mengen angeben.

Satz 2.2o: Das Halteproblem $H=\{u\#v\mid M_u$ angesetzt auf v hält$\}$ ist rekursiv aufzählbar.

Beweis: Wir beschreiben eine 2-Band-Turingmaschine V. Angesetzt auf $w\in\{0,1,\#\}^*$ testet V, ob w von der Form u#v#bin(i) mit $u,v\in\{0,1\}^*$ und $i\in\mathbb{N}$ ist. Verläuft dieser Test negativ, druckt V $u_o\#0$, wobei u_o Gödelisierung irgendeiner Maschine ist, die angesetzt auf 0 hält.

Verläuft der Test positiv, simuliert V die Maschine M_u für i Schritte: Zu Beginn der Simulation wird bin(i) auf Band 2 kopiert; auf Band 1 verhält sich V wie die Maschine U aus Satz 2.17 angesetzt auf u#v; nach der Simulation eines jeden Schritts von M_u wird die Zahl auf Band 2 um 1 erniedrigt; die Simulation bricht spätestens ab, wenn auf Band 2 0 steht. Hält M_u angesetzt auf v innerhalb dieser i Schritte, druckt V die Zeichenreihe u#v, andernfalls druckt V u_o#0. Offenbar ist $H=\{f_V(w) \mid w\in\{0,1,\#\}^*\}$. □

Zusammen mit Korollar 2.11 und Satz 2.16 folgt:

Korollar 2.21: $\overline{H}$ ist nicht rekursiv aufzählbar.

Wir sind jetzt auch in der Lage, zu zeigen, daß die Menge aller totalen berechenbaren Funktionen nicht rekursiv aufzählbar ist, genauer:

Satz 2.22: Es gibt keine rekursiv aufzählbare Menge $T\subset\{0,1\}^*$, so daß $\{f_{M_u} \mid u\in T\}$ gleich der Menge aller totalen berechenbaren Funktionen $g:\{0,1\}^*\to\{0,1\}^*$ ist.

Der Beweis benutzt wieder ein Diagonalverfahren. Angenommen ein solches T existiert und h zählt T auf. Dann ist die Funktion $g:\{0,1\}^*\to\{0,1\}^*$,

$$g(w) = \begin{cases} 0 & \text{falls } f_{M_{h(w)}}(w)\neq 0 \\ 1 & \text{sonst} \end{cases}$$

eine totale berechenbare Funktion. Denn eine Maschine V für g erhält man leicht aus einer Maschine für h und einer universellen Turingmaschine. Die Maschine V hält für alle Eingaben w, weil nach Voraussetzung die Maschinen $M_{h(w)}$ für alle Eingaben, insbesondere also auch für Eingabe w halten.

Nach Voraussetzung gibt es nun ein $v\in\{0,1\}^*$, so daß $f_{M_{h(v)}}=f_V=g$ gilt. $M_{h(v)}$ berechnet also g. Hieraus folgt aber der Widerspruch $f_{M_{h(v)}}(v)=g(v)\neq f_{M_{h(v)}}(v)$. □

Übungen

1. Man beschreibe eine Maschine, die angesetzt auf u#v mit $u,v\in\{0,1\}^*$ eine Gödelisierung der Maschine "$M_u;M_v$" erzeugt.
2. a) Man gebe ein Schema zum Gödelisieren von 2-Band-Turingmaschinen an.

b) Sei M eine Maschine wie in Beispiel 1.22 mit Alphabet $\{0,1,\#,B,x\}$. Wie wird M gödelisiert?

3. Welche der folgenden Mengen sind entscheidbar?
   a) $\{u \mid f_{M_u}(v)=v \text{ für alle } v\in\{0,1\}^*\}$
   b) $\{u \mid M_u \text{ hält für jede Eingabe}\}$
   c) $\{u \mid \{f_{M_u}(v) \mid v\in\{0,1\}^*\} \text{ ist entscheidbar}\}$.

4. Man zeige:
   a) $L\subset A^*$ ist genau dann rekursiv aufzählbar, wenn es eine partielle berechenbare Funktion $\tilde{\chi}_L: A^*\to A^*$ gibt mit

$$\tilde{\chi}_L(w) = \begin{cases} 1 & \text{falls } w\in L \\ 0 & \text{oder nicht definiert sonst} \end{cases}$$

   b) $L\subset A^*$ ist genau dann rekursiv aufzählbar, wenn es eine partielle berechenbare surjektive Funktion $f: A^*\to L$ gibt.
   c) $L\subset A^*$ ist genau dann unendlich und rekursiv aufzählbar, wenn es eine totale berechenbare bijektive Funktion $f: A^*\to L$ gibt.

5. Ist die Menge

$$B=\{u \mid \text{Für alle } v\in\{0,1\}^* \text{ ist } f_{M_u}(v)\in\{0,1\}^* \text{ oder nicht definiert}\}$$

   a) entscheidbar?
   b) rekursiv aufzählbar?
   c) ist $\overline{B}$ rekursiv aufzählbar?
   d) gibt es eine entscheidbare Menge $B'\subseteq\{0,1\}^*$, so daß $\{f_{M_u} \mid u\in B\}=\{f_{M_u} \mid u\in B'\}$ gilt.

6. Man benutze die Techniken aus Kapitel 1, um die universelle 1-Band-Turingmaschine aus Satz 2.17 vollständig zu spezifizieren.

2.2. Die Unvollständigkeit der Arithmetik

Zum Abschluß unserer Untersuchungen über nicht berechenbare Funktionen leiten wir die Gödel'schen Unvollständigkeitssätze her, welche eine fundamentale und unerwartete Eigenschaft der Mathematik aufdecken: die Mathematik läßt sich nicht ein für allemal forma-

lisieren. In Kapitel 6 erhalten wir durch einfache Modifikation der hier entwickelten Techniken einen Beweis dafür, daß ein gewisses entscheidbares Problem notwendig enormen Rechenaufwand erfordert.

2.2.1. Motivation

In der Mathematik stellt man Behauptungen über mathematische Objekte (z.B. über Zahlen) auf und beweist diese Behauptungen in einer Folge von Beweisschritten. In einem Beweisschritt beruft man sich entweder auf Grundtatsachen, die selbst nicht bewiesen werden (Axiome), oder man schließt nach gewissen Regeln (Schlußregeln) von Axiomen und bereits bewiesenen Aussagen auf neue Aussagen. Beispielsweise ist die Aussage: "Jede natürliche Zahl hat einen Nachfolger" eines der Peano'schen Axiome. Eine gängige Schlußregel ist: wenn man "A" bewiesen hat und wenn man "aus A folgt B" bewiesen hat, dann darf man "B" schließen.

Sind einmal Axiome und Schlußregeln spezifiziert, ist das Aufstellen eines Beweises nur noch ein formales Spiel, in dem man Axiome und bereits bewiesene Aussagen im Rahmen der Schlußregeln frei kombinieren kann.

Aussagen, die auf diese Weise "mathematisch" bewiesen sind, genießen wie kaum andere Aussagen den Ruf, "absolut wahr" zu sein, was immer das heißen mag. Um persönlich mathematisch bewiesene Aussagen für wahr halten zu können, muß man jedoch zunächst einmal glauben, daß

1) die Axiome wahr sind
   und
2) daß die Schlußregeln korrekt sind, d.h. daß wieder wahre Aussagen herauskommen, wenn man die Schlußregeln auf wahre Aussagen anwendet.

Zu dem Glauben, daß etwa ein bestimmtes Axiom wahr ist, kann man z.B. kommen, indem man sich das Axiom so lange in Ruhe anschaut, bis einem die eigene Intuition sagt: "jawohl, das ist wirklich wahr" (Die Intuition kann man durch Experimente stützen, z.B. indem man viele Striche auf einen großen, leeren Sandstrand macht und sich damit überzeugt, daß zumindest sehr viele Zahlen einen Nachfolger haben).

Wo das subjektive Ermessen vieler Personen ins Spiel kommt, kann man im allgemeinen nicht darauf hoffen, daß alle Beteiligten in

allen Fragen der gleichen Meinung sind. In der Tat lehnen einige Mathematiker etwa das Prinzip vom ausgeschlossenen Dritten ab; sie würden unseren früher gegebenen Beweis für die Aussage

"Die Funktion

$$f(x) = \begin{cases} 1 & \text{falls Fermat's Vermutung richtig ist} \\ 0 & \text{falls Fermat's Vermutung falsch ist} \end{cases}$$

ist berechenbar"

zwar als Beweis in unserem Sinn erkennen, aber darin für sich selbst keinen Anlaß sehen, diese Aussage für wahr zu halten. Tröstlich ist, daß man sich in der Mathematik nur auf relativ wenige wohldefinierte Axiome und Schlußregeln zu verlassen braucht und damit schon sehr viel herleiten kann. Den größten Teil der bekannten Mathematik kann man z.B. aus den Axiomen und Schlußregeln von Prädikatenlogik und Mengenlehre herleiten, die sich mühelos auf wenigen Seiten vollständig spezifizieren lassen.

Nun gibt es von Zeit zu Zeit immer wieder jemanden, der ein neues Axiom oder eine neue Schlußweise erfindet und damit schöne neue Sätze beweisen kann. Pascal etwa führte als erster Induktionsbeweise, und Cantor postulierte, daß es zu jeder Menge M eine Menge $P(M)=\{X|X\subseteq M\}$ gibt. In einer solchen Situation müssen alle Beteiligten wieder an ihre Intuition appellieren; das neue Axiom bzw. die neue Schlußweise wird allgemein akzeptiert oder als Unfug abgetan, oder eine Diskussion entbrennt.

Eine naheliegende Frage ist hier, ob das immer so weitergehen muß. Kann es sein, daß man irgendwann einmal alle Axiome und Schlußweisen gefunden hat, die man jemals brauchen wird und die es gestatten, jedes mathematische Problem zu lösen, oder muß man immer wieder an die Intuition appellieren?

Im folgenden zeigen wir unter anderem, daß schon für ein Teilgebiet der Mathematik, nämlich die Zahlentheorie, die zweite Alternative zutrifft.

2.2.2. Beweissysteme

In mathematischen Texten opfert man oft literarischen Stil zugunsten präziser Ausdrucksweise. In der Tat kann man die Sprache, in der man Axiome, Schlußregeln und Beweise formuliert, so vollständig formalisieren, daß sich Beweise per Computer

verifizieren lassen.

Definition 2.23: Ein Beweissystem wird spezifiziert durch ein Quadrupel $S=(\Sigma,L,A,S)$, wobei gilt:

Σ ist ein endliches Alphabet.

$L\subset\Sigma^*$ ist eine entscheidbare Menge und heißt die Sprache von S.

Jede Zeichenreihe $w\in L$ soll in eindeutiger Weise als Aussage über mathematische Objekte interpretiert werden können.

$A\subset L$ ist eine entscheidbare Menge und heißt Menge der Axiome von S.

S ist eine entscheidbare Menge von Worten der Form $w_1|\ldots|w_r$, $w_i\in L, |\notin\Sigma$ und heißt Menge der Schlußregeln von S. Für "$w_1|\ldots|w_r\in S$" sagt man auch "w_r folgt direkt aus $w_1,\ldots,w_{r-1}$". Ein ausführliches Beispiel für ein Beweissystem wird in § 1.2.3 gegeben.

Definition 2.24: $S=(\Sigma,L,A,S)$ sei ein Beweissystem. Eine Zeichenreihe $b=w_1|\ldots|w_s, w_i\in L$, heißt Beweis von w_s in S, falls für alle $i\leq s$ gilt: $w_i\in A$ oder es gibt Indizes $i_1,\ldots,i_r<i$ mit $w_{i_1}|\ldots|w_{i_r}|w_i\in S$.

Gibt es einen Beweis von w in S sagt man auch: w ist in S herleitbar. Es ist nicht schwer, unter Verwendung von Entscheidungsverfahren für A und S ein Rechenverfahren anzugeben, das bei Eingabe von Zeichenreihen $b\in(\Sigma\cup\{|\})^*$ entscheidet, ob b Definition 2.24 erfüllt, d.h. es gilt

Lemma 2.25: Ist S ein Beweissystem, so ist die Menge $\{b|b$ ist ein Beweis in $S\}$ entscheidbar.

Eine leichte Folgerung ist

Korollar 2.26: Ist S ein Beweissystem, so ist die Menge $\{w|w$ ist in S herleitbar$\}$ rekursiv aufzählbar.

Beweis: w_o sei irgendeine in S herleitbare Aussage. M sei eine Maschine, die bei Eingabe b zunächst testet, ob b ein Beweis in S ist. Ist b ein Beweis in S, druckt M die durch b bewiesene Aussage aus, andernfalls druckt M die Zeichenreihe w_o aus. Offenbar ist $\{w|w$ in S herleitbar$\} = \{f_M(x)|x\in(\Sigma\cup\{|\})^*\}$. □

2.2.3. Das System Z_E

Als Beispiel für ein Beweissystem definieren wir ein System $Z_E=(\Sigma_E,L_E,A_E,S_E)$ für die elementare Zahlentheorie.

$\Sigma_E=\{0,1,x,(,),t,\cdot,=,\wedge,\vee,\sim,\rightarrow,\forall,\exists\}$

$V=\{xw|w\in\{0,1\}^+\}$ ist die Menge der Variablen.

Definition 2.27: Die Menge der Terme ist die kleinste Menge T für die gilt:

1) $0\in T, 1\in T, V\subset T$.
2) Sind $a,b\in T$, so auch $(a+b)$ und $(a\cdot b)$.

Beispielsweise ist $((x10+1)\cdot 1)$ ein Term.

Definition 2.28: Die Menge der arithmetischen Prädikate ist die kleinste Menge P, für die gilt:

1) Sind a und b Terme, so ist $(a=b)\in P$.
2) Sind A und $B\in P$, so auch $(\sim A),(A\wedge B),(A\vee B),(A\rightarrow B)$.
3) Ist $y\in V$ und $A\in P$, so sind $(\exists yA)$ und $(\forall yA)\in P$.

Beispielsweise ist

$$(\forall x0(\forall x1((x0+x1) = ((x1+x0)+x10))))$$

ein arithmetisches Prädikat.

Definition 2.29: y sei eine Variable.

1) Sind a und b Terme und kommt y in a oder b vor, so kommt y in $(a=b)$ frei vor.
2) Ist A ein arithmetisches Prädikat und kommt y in A vor, so kommt y in $(\exists yA)$ und in $(\forall yA)$ gebunden vor.
3) Sind A und B arithmetische Prädikate und kommt y in A oder B frei bzw. gebunden vor, so kommt y in $(A\wedge B),(A\vee B)$ und $(A\rightarrow B)$ frei bzw. gebunden vor. Kommt y in A frei bzw. gebunden vor, so auch in $(\sim A)$.

Im obigen Beispiel kommen x0 und x1 gebunden und x10 frei vor. Wir bemerken, daß unsere Definition zuläßt, daß die gleiche Variable durch mehrere Quantoren gebunden wird; die gleiche Variable kann auch in einem Prädikat an verschiedenen Stellen frei und gebunden vorkommen. In Zukunft bezeichnet $A(x_1,\ldots,x_n)$ ein Prädikat, in dem die Variablen $x_1,\ldots,x_n$ frei vorkommen. Ein arithmetisches Prädikat, in dem keine Variable frei vorkommt,

heißt arithmetische Aussage.

Lemma 2.3o: Die Menge $L_E=\{A|A$ ist arithmetisches Prädikat$\}$ ist entscheidbar.

Beweis: Ein Programm zu schreiben, das bei Eingabe von Zeichenreihen $w\in\Sigma_E^*$ entscheidet, ob w ein arithmetisches Prädikat im Sinne von Definition 2.29 ist, wäre nicht schwierig, sondern bloß mühsam. □

Wir haben bisher nur Mengen von Zeichenreihen definiert und müssen jetzt angeben, wann ein Prädikat bzw. eine Aussage wahr sein soll. Die folgende Definition formalisiert das Einsetzen von Werten für die Variablen eines Terms.

Definition 2.31: Eine Abbildung $\varphi: V\to\mathbb{N}$ heißt Belegung. Für Belegungen φ und Terme t wird $\varphi(t)$, der Wert von t bei Belegung φ induktiv definiert durch:

$$\varphi(0)=0,\ \varphi(1)=1,\ \varphi(a+b)=\varphi(a)+\varphi(b),\ \varphi(a\cdot b)=\varphi(a)\cdot\varphi(b).$$

Beispiel 2.32: Sei $t=((x0+x1)\cdot(x0+x1))$ und $\varphi(x0)=1, \varphi(x1)=2$; dann ist $\varphi(t)=\varphi(x0+x1)\cdot\varphi(x0+x1)=(\varphi(x0)+\varphi(x1))\cdot(\varphi(x0)+\varphi(x1))=(1+2)\cdot(1+2)=9$.

Für Prädikate A, Variablen x und Terme t bezeichne $A|_{x=t}$ das Prädikat, das man erhält, wenn man jedes freie Vorkommen der Variablen x in A durch den Term t ersetzt. Für $A(x_1,\ldots,x_n)|_{x_1=t_1,\ldots,x_n=t_n}$ schreiben wir auch $A(t_1,\ldots,t_n)$.

Für $n\in\mathbb{N}$ bezeichne $\overline{n}$ den Term $(1+(\ldots+1)\ldots)$ (n-mal).

Definition 2.33: Für alle Terme a,b, Prädikate A,B und Variablen x gilt:

a=b ist wahr, falls $\varphi(a)=\varphi(b)$ für alle Belegungen φ gilt.
$A\wedge B$ ist wahr, falls A und B beide wahr sind.
$A\vee B$ ist wahr, falls A oder B oder beide wahr sind.
$\sim A$ ist wahr, falls A nicht wahr ist.
$A\to B$ ist wahr, falls $(B\vee(\sim A))$ wahr ist.
$\exists xA$ ist wahr, falls es eine Zahl $n\in\mathbb{N}$ gibt, so daß $A|_{x=\overline{n}}$ wahr ist.
$\forall xA$ ist wahr, falls $A|_{x=\overline{n}}$ für alle $n\in\mathbb{N}$ wahr ist.

Beispiel 2.34: Die Prädikate

$$((x0+x1)\cdot(x0+x1)) = ((x0\cdot x0)+(((1+1)\cdot(x0\cdot x1))+(x1\cdot x1)))$$

und

$$(\exists x1((x0+x1)=x0))$$

sind wahr.

Die Axiome und Schlußregeln von Z_E spezifizieren wir hauptsächlich, um interessierten Lesern ein vollständiges Beispiel eines sinnvollen Beweissystems zu geben. Einige der im folgenden bewiesenen Sätze beziehen sich nur auf die Sprache von Beweissystemen. In den übrigen Fällen beziehen wir uns nur dadurch auf Axiome und Schlußregeln, daß wir ohne Beweis behaupten, daß sich gewisse Beweise mit den angegebenen Axiomen und Schlußregeln formalisieren lassen. Das spezielle System von Axiomen und Schlußregeln sowie Beispiel 2.38 sind aus [37]. Zugunsten übersichtlicherer Schreibweise haben wir einige Klammern weggelassen. Die unvollständig geklammerten Zeichenreihen sind als Abkürzung für die vollständig geklammerten Zeichenreihen zu verstehen, wobei die üblichen Prioritäten gelten:

$$\cdot,\ +,\ =,\ \forall,\ \exists,\ \sim,\ \wedge,\ \vee,\ \rightarrow\ .$$

Definition 2.35: Axiome und Schlußregeln für logische Verknüpfungen:
Für alle arithmetischen Prädikate A,B und C sind 1-8 Axiome bzw. Schlußweisen:

1. a) $A\rightarrow(B\rightarrow A)$
1. b) $(A\rightarrow B)\rightarrow((A\rightarrow(B\rightarrow C))\rightarrow(A\rightarrow C))$
2. $A \mid A\rightarrow B \mid B$ (Schlußweise)
3. $A\rightarrow(B\rightarrow A\wedge B)$
4. a) $A\wedge B\rightarrow A$
4. b) $A\wedge B\rightarrow B$
5. a) $A\rightarrow A\vee B$
5. b) $B\rightarrow A\vee B$
6. $(A\rightarrow C)\rightarrow((B\rightarrow C)\rightarrow(A\vee B\rightarrow C))$
7. $(A\rightarrow B)\rightarrow((A\rightarrow\sim B)\rightarrow\sim A)$
8. $\sim\sim A\rightarrow A.$

Für Variablen y, Prädikate A(y) und Terme t heißt t frei für y in A(y), falls in A(t) alle Variablen, die in t vorkommen, frei sind. Beispielsweise sind in dem Prädikat

$$A(x1)=(\exists x0(1+1+x0=x1))\rightarrow(\exists x0(1+x0=x1))$$

die Terme 1, 1+1 oder x10 frei für x1, aber x0 oder 1+1+x0 sind nicht frei für x1. Ist t nicht frei für y in A(y), so kann A(t) falsch werden, obwohl A(y) wahr ist. A(x1) oben ist wahr. A(1+1+x0) ist falsch.

Definition 2.36: Axiome und Schlußweisen für Quantoren.
Für alle Prädikate A(y), in denen y frei vorkommt, für alle Terme t, die frei für y in A(y) sind, und für alle Prädikate C, in denen y nicht frei vorkommt, sind 9-12 Axiome bzw. Schlußweisen

9. $C\rightarrow A(y) \mid C\rightarrow\forall yA(y)$
10. $\forall yA(y)\rightarrow A(t)$
11. $A(t)\rightarrow\exists yA(y)$
12. $A(y)\rightarrow C \mid \exists yA(y)\rightarrow C$.

Definition 2.37: Axiome für natürliche Zahlen.
Für alle Prädikate A(y), in denen y frei vorkommt, und für alle Terme a,b sind 13-21 Axiome

13. $A(0)\wedge\forall y(A(y)\rightarrow A(y+1))\rightarrow A(y)$ (Induktionsaxiome)
14. $a+1=b+1\rightarrow a=b$
15. $\sim a+1=0$
16. $a=b\rightarrow(a=c\rightarrow b=c)$
17. $a=b\rightarrow a+1=b+1$

18. $a+0=a$ } Axiome für Addition
19. $a+(b+1)=(a+b)+1$ }

20. $a\cdot 0=0$ } Axiome für Multiplikation
21. $a\cdot(b+1)=a\cdot b+a$ }

Wir erklären A_E als die Menge aller durch 1, 3-8, 10, 11, 13-20 definierten Zeichenreihen und S_E als die Menge aller durch 2, 9, 12 definierten Zeichenreihen. Man kann wieder ohne viel Schwierigkeiten Programme angeben, die zu gegebener Zeichenreihe w testen, ob w in A_E bzw. S_E liegt. Zusammen mit Lemma 2.30 folgt, daß wir ein Beweissystem $Z_E=(\Sigma_E,L_E,A_E,S_E)$ im Sinne von Definition 2.23 spezifiziert haben.

Wir geben als Beispiel in Z_E einen Beweis für das Prädikat x0=x0. Die eingeklammerten Zeilennummern und die eingeklammerten Kommentare, die angeben, welche Regel benutzt wurde, sind nicht Teil des formalen Beweises in Z_E.

Beispiel 2.38:

(1) $x0=x1\rightarrow(x0=x10\rightarrow x1=x10)|$ (Axiom 16)

(2) $0=0\rightarrow(0=0\rightarrow 0=0)|$ (Axiom 1a)

(3) $(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))\rightarrow((0=0\rightarrow(0=0\rightarrow 0=0))\rightarrow(x0=x1\rightarrow(x0=x10\rightarrow x1=x10)))|$ (Axiom 1a)

(4) $(0=0\rightarrow(0=0\rightarrow 0=0))\rightarrow(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))|$ (Regel 2)

(5) $(0=0\rightarrow(0=0\rightarrow 0=0))\rightarrow\forall x10(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))|$ (Regel 9)

(6) $(0=0\rightarrow(0=0\rightarrow 0=0))\rightarrow\forall x1\forall x10(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))|$ (Regel 9)

(7) $(0=0\rightarrow(0=0\rightarrow 0=0))\rightarrow\forall x0\forall x1\forall x10(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))|$ (Regel 9)

(8) $\forall x0\forall x1\forall x10(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))|$ (Regel 2)

(9) $\forall x0\forall x1\forall x10(x0=x1\rightarrow(x0=x10\rightarrow x1=x10))\rightarrow\forall x1\forall x10(x0+0=x1\rightarrow(x0+0=x10\rightarrow(x1=x10))|$ (Axiom 1o)

(1o) $\forall x1\forall x10(x0+0=x1\rightarrow(x0+0=x10\rightarrow x1=x10))|$ (Regel 2)

(11) $\forall x1\forall x10(x0+0=x1\rightarrow(x0+0=x10\rightarrow x1=x10))\rightarrow\forall x10(x0+0=x0\rightarrow(x0+0=x10\rightarrow x0=x10))|$ (Axiom 1o)

(12) $\forall x10(x0+0=x0\rightarrow(x0+0=x10\rightarrow x0=x10))|$ (Regel 2)

(13) $\forall x10(x0+0=x0\rightarrow(x0+0=x10\rightarrow x0=x10))\rightarrow(x0+0=x0\rightarrow(x0+0=x0\rightarrow x0=x0))|$ (Axiom 1o)

(14) $x0+0=x0\rightarrow(x0+0=x0\rightarrow x0=x0)|$ (Regel 2)

(15) $x0+0=x0|$ (Axiom 18)

(16) $x0+0=x0\rightarrow x0=x0|$ (Regel 2)

(17) $x0=x0$ (Regel 2).

Selbst die Herleitung ganz einfacher Zusammenhänge ist in dem System Z_E offenbar schon sehr mühsam. Dennoch lassen sich z.B. in Z_E alle Sätze einer Standardvorlesung über elementare Zahlentheorie herleiten. Die Situation ist ähnlich wie bei Turingmaschinen: man hat einen einfachen, aber schwerfälligen Formalismus, dem man nicht auf Anhieb ansieht, was in ihm steckt, aber nach einiger Gewöhnung bekommt man ein Gefühl dafür, was turingberechenbar bzw. in Z_E herleitbar ist, ohne explizit Turingmaschinen bzw. Beweise in Z_E hinschreiben zu müssen.

2.2.4. Korrektheit, Konsistenz und Vollständigkeit

Wir führen einige wünschenswerte Eigenschaften von Beweissystemen ein.

Definition 2.39: Ein Beweissystem $S=(\Sigma,L,A,S)$ heißt korrekt, wenn alle in S herleitbaren Aussagen wahr sind. S heißt konsistent (widerspruchsfrei), falls für kein $w\in L$ sowohl w als auch $\sim w$ in S

herleitbar ist. S heißt vollständig, falls für jedes $w \in L$ w oder aber $\sim w$ in S herleitbar ist.("~" soll stets "nicht" bedeuten).

Jedes korrekte System ist konsistent, weil wir glauben, daß für keine Aussage w sowohl w als auch ~w wahr ist. Im Gegensatz zu Korrektheit hat Konsistenz nichts mit der Bedeutung und Wahrheit von Aussagen zu tun, sondern ist eine rein kombinatorische Eigenschaft von Beweissystemen. Konsistenz ist somit für formale mathematische Untersuchungen viel leichter zugänglich als Korrektheit.

Die Korrektheit eines Systems S kann man intuitiv so beweisen:

1) Die Axiome von S sind wahr.
2) Die Schlußregeln von S erhalten Wahrheit.
3) Aus 1) und 2) folgt per Induktion, daß alle in S herleitbaren Aussagen wahr sind.

Die Intuition, die einen dazu (ver-)leitet, an die Korrektheit von Axiomen und Schlußregeln zu glauben, kann trügen. Beweissysteme, die schon weitgehend als korrekt akzeptiert worden waren, haben sich als inkonsistent erwiesen (Stichwort: Russel'sches Paradoxon). Das System Z_E wird von vielen (aber nicht allen) Mathematikern für korrekt gehalten (Stein des Anstoßes ist " $\sim \sim A \rightarrow A$ ").

In einem Beweissystem, das gleichzeitig korrekt und vollständig ist, kann man die Wahrheit jeder Aussage, die in der Sprache des Systems formuliert werden kann, per Computer entscheiden:

Lemma 2.4o: Ist $S=(\Sigma,L,A,S)$ ein korrektes und vollständiges Beweissystem, so ist die Menge $\{w \in L \mid w \text{ wahr}\}$ entscheidbar.

Beweis: Das folgende Verfahren leistet das Gewünschte: bei Eingabe $w \in L$ zählt man in irgendeiner Ordnung die Zeichenreihen $b \in (\Sigma \cup \{|\})^+$ auf (etwa nach wachsender Länge und unter Worten gleicher Länge in alphabetischer Ordnung) und testet für jede Zeichenreihe b, ob b ein Beweis in S von w oder von ~w ist. Ist ein Beweis von w bzw. ~w gefunden, wird "w wahr" bzw. "w falsch" entschieden, und das Verfahren bricht ab.

Da das System S vollständig ist, bricht das Verfahren für jede Eingabe w ab. Das Verfahren ist korrekt, weil nach Voraussetzung S korrekt ist (diese Bemerkung ist keine Linsenspalterei; denn die Herleitbarkeit einer Zeichenreihe w in einem formalen System sagt

allein noch gar nichts über die Wahrheit der mit w assoziierten Aussage). □

2.2.5. Die Gödel'schen Unvollständigkeitssätze

Wir zeigen zunächst einen sehr allgemeinen Satz über die Menge der wahren Aussagen, die man in der Sprache eines Beweissystems formulieren kann.

Satz 2.41: Sei $S=(\Sigma,L,A,S)$ ein Beweissystem, und es gebe eine berechenbare Funktion $H:\{0,1,\#\}^*\to L$, so daß für alle $u,v\in\{0,1\}^*$ gilt: $H(u\#v)$ ist genau dann wahr, wenn die 1-Band-Turingmaschine M_u angesetzt auf v hält.

Dann ist die Menge $\{h\in L \mid h \text{ wahr}\}$ nicht entscheidbar.

Beweis: Die Voraussetzung des Satzes besagt gerade, daß das Halteproblem auf die Menge $\{h\in L \mid h \text{ wahr}\}$ reduzierbar ist. □

Zusammen mit Lemma 2.4o folgt sofort

Korollar 2.42: Erfüllt S die Voraussetzungen von Satz 2.41 und ist S korrekt, so ist S unvollständig.

Haupt-Lemma 2.43: Z_E erfüllt die Voraussetzungen von Satz 2.41.

Der Beweis ist eine längere Übung im Formulieren von Aussagen in L_E und wird in § 2.2.6 geführt.

Korollar 2.44: Die Menge der wahren arithmetischen Aussagen ist nicht entscheidbar, und es gibt arithmetische Aussagen, die in Z_E weder beweisbar noch widerlegbar sind (falls Z_E korrekt ist).

Man könnte nun hoffen, daß durch Zufügen einiger Axiome und Schlußregeln zu Z_E das System vollständig gemacht werden kann (die Mengen der neuen Axiome und Schlußregeln müssen natürlich entscheidbar sein; sonst könnte man A_E einfach zu $\{h\in L_E \mid h \text{ wahr}\}$ erweitern). Wir sagen, daß ein Beweissystem $S=(\Sigma,L,A,S)$ das System Z_E umfaßt, falls $L\supseteq L_E$, $A\supseteq A_E$ und $S\supseteq S_E$ gilt. Satz 2.41 und Korollar 2.42 gelten aber auch für jedes Beweissystem, das Z_E umfaßt, d.h. es gilt

Korollar 2.45: Ist S ein korrektes Beweissystem, das Z_E umfaßt, so ist S unvollständig.

Die obigen Ergebnisse sind reine Existenzsätze. Es gilt jedoch

Satz 2.46 (Erster Gödel'scher Unvollständigkeitssatz):

Ist $S=(\Sigma,L,A,S)$ ein korrektes Beweissystem und erfüllt S die Voraussetzungen von Satz 2.41, dann kann man explizit eine Aussage angeben, die in S weder beweisbar noch widerlegbar ist. Umfaßt S überdies Z_E, so kann man intuitiv (d.h. außerhalb des Systems S) zeigen, daß diese Aussage falsch ist.

Beweis: Sei Q eine Maschine, die angesetzt auf $w\in\{0,1\}^*$ alle Beweise von S aufzählt, bis sie eventuell einen Beweis für die Aussage $\sim H(w\#w)$ findet. Nur wenn Q einen solchen Beweis in S findet, stoppt Q. Sei $q\in\{0,1\}^*$ eine Gödelisierung von Q. Wir behaupten, daß die Aussage $H(q\#q)$ in S weder beweisbar noch widerlegbar ist.

Wäre $H(q\#q)$ in S beweisbar, so wäre - weil S korrekt, folglich konsistent ist - $\sim H(q\#q)$ in S nicht beweisbar. Dann hält aber - nach Konstruktion von Q - die Maschine Q angesetzt auf q nicht; folglich wäre S nicht korrekt, weil $H(q\#q)$ falsch, aber in S herleitbar wäre.

Wäre andererseits $\sim H(q\#q)$ in S beweisbar, so würde - nach Konstruktion von Q - die Maschine angesetzt auf q halten, woraus wieder folgen würde, daß S nicht korrekt ist.

Zum Beweis des zweiten Teils des Satzes benutzt man

Lemma 2.47: Ist $H(u\#v)$ wahr, so ist $H(u\#v)$ in Z_E herleitbar.

Beweis: Intuitiv ist eine endliche Rechnung von M_u angesetzt auf v im Sinne von Definition 1.18 ein Beweis dafür, daß M_u angesetzt auf v hält. Auch formal kann man in Z_E aus einer solchen Rechnung einen Beweis für $H(u\#v)$ gewinnen. Statt Einzelheiten verweisen wir auf die Bemerkung am Ende von § 2.2.3. □

Aus Lemma 2.47 folgt der zweite Teil des Satzes, denn würde Maschine Q angesetzt auf q halten, wäre $H(q\#q)$ in Z_E - also auch in S - herleitbar, was nach dem Beweis des ersten Teils nicht möglich ist. □

Wir untersuchen nun näher, wie es um die Aussicht steht, die Konsistenz eines Systems zu beweisen. Da Konsistenz nichts mit der Bedeutung von Axiomen und Schlußweisen zu tun hat, könnte man hoffen, mit relativ einfachen Mitteln (Z_E) schon die Konsistenz höherer Systeme (etwa der axiomatischen Mengenlehre) nachweisen zu können, selbst wenn die Bedeutung einiger Axiome des höheren Systems vergleichsweise unglaublich ist. Die Aufgabe, dies zu tun,

ist als "das Hilbert'sche Programm" [29] bekannt. Auf Anhieb ist nicht einmal klar, daß man innerhalb eines Systems S die Aussage "S ist konsistent" überhaupt formulieren kann. Man sieht jedoch leicht, daß dies sogar schon in Z_E möglich ist:

Lemma 2.48: Ist S ein beliebiges Beweissystem, dann gibt es eine arithmetische Aussage mit der Bedeutung: S ist konsistent.

Beweis: Sei R eine Turingmaschine, die angesetzt auf die Zeichenreihe 0 alle Beweise in S aufzählt, die bewiesenen Aussagen w auf das Band schreibt und jede neue bewiesene Aussage mit allen vorher bewiesenen Aussagen w' vergleicht. Die Maschine R stoppt genau dann, wenn sie hierbei ein Paar von Aussagen w und w' findet, so daß w=~w' oder w'=~w gilt.

Ist r eine Gödelisierung von R, so ist offenbar ~H(r#0) eine arithmetische Aussage mit der Bedeutung "S ist konsistent". □

Wir schreiben für "~H(r#0)" auch "Consis S".

Satz 2.49 (Zweiter Gödel'scher Unvollständigkeitssatz):

Ist $S=(\Sigma,L,A,S)$ ein konsistentes Beweissystem, das Z_E umfaßt, dann ist "Consis S" nicht in S herleitbar.

Beweis: Wir beginnen wie im Beweis von Satz 2.46. Sei Q eine Maschine, die angesetzt auf w so lange alle Beweise in S aufzählt, bis sie (eventuell) einen Beweis für ~H(w#w) findet und dann stoppt. q sei eine Gödelisierung von Q.

Lemma 2.5o: Ist S konsistent, dann ist ~H(q#q) nicht in S herleitbar.

Beweis: Ist ~H(q#q) in S herleitbar, findet Q angesetzt auf q den Beweis von ~H(q#q) und hält. Dann ist aber nach Lemma 2.47 auch H(q#q) in Z_E - also auch in S - herleitbar, und S ist inkonsistent. □

Analog zu Lemma 2.48 gilt:

Lemma 2.51: Ist $v \in L$, so gibt es ein arithmetisches Prädikat, das bedeutet: "v ist in S herleitbar".

Beweis: Sei M eine Maschine, die bei Eingabe v alle Beweise in S aufzählt, bis sie einen Beweis von v findet und dann stoppt. Ist m eine Gödelisierung von M, so ist H(m#v) ein arithmetisches Prädikat für "v ist in S herleitbar". □

Damit kann Lemma 2.5o in der Sprache von Z_E formuliert werden. Es gilt sogar:

Lemma 2.52: Die Aussage

(1) "Consis $S \rightarrow (\sim H(q\#q)$ ist nicht in S herleitbar)"
ist in Z_E (also auch in S) herleitbar.

Beweis: Der Beweis von Lemma 2.5o kann in Z_E formalisiert werden. Statt Details siehe Schluß von § 2.2.3. □

Wäre nun "Consis S" in S herleitbar, so könnte man aus (1) in Z_E (also auch in S) schließen:

(2) "$\sim H(q\#q)$ ist nicht in S herleitbar".

Aus (2) folgt nach Definition von Q

(3) "$\sim H(q\#q)$",

und dies kann ebenfalls in Z_E geschlossen werden.

Lemma 2.53: Ist $v \in L$ in S herleitbar, so ist auch eine arithmetische Aussage mit Bedeutung "v ist in S herleitbar" in Z_E (also auch in S) herleitbar.

Beweis: Beweis unmittelbar aus dem Beweis von Lemma 2.51 und Lemma 2.47. □

Da "$\sim H(q\#q)$" in S herleitbar ist, ist daher auch die Aussage

(4) "$\sim H(q\#q)$ ist in S herleitbar"
in S herleitbar.

Somit sind sowohl (2) als auch (4) in S herleitbar, aber
(2) = $\sim$ (4) im Widerspruch zur Konsistenz von S. □

2.2.6. Beweis des Haupt-Lemmas

Zu beliebiger 1-Band-Turingmaschine M und $v \in \{0,1\}^*$ konstruieren wir eine arithmetische Aussage mit Bedeutung: " M angesetzt auf v hält". Unsere Konstruktion ist völlig explizit, so daß diese Aussage aus v und einer Gödelisierung von M per Computer generiert werden könnte.

Sei $M=(Z,A,\delta,z_o,E)$ und / sei ein neues Zeichen, d.h. / $\notin$AUZ.

M angesetzt auf v hält genau dann, wenn es ein Wort
$w=/k_1/.../k_T/$ gibt mit

(1) k_1 hat die Form $B...Bz_ovB...B$ (B = Leerzeichen)

(2) $k_i \vdash k_{i+1}$ für alle $i \leq T-1$

(3) $|k_i| = |k_j|$ für alle i,j

(4) In keinem k_i steht ein Zeichen $z \in Z$ am Rand

(5) k_T ist Endkonfiguration.

Die Bedingungen (3) und (4) helfen später, lästige Fallunterscheidungen zu vermeiden.

Da wir in der Sprache von Z_E explizit nur von Zahlen reden können, müssen wir erst wieder wie in § 1.3.6 Zeichenreihen als Zahlen kodieren und dann die Bedingungen (1) bis (5) als Beziehungen zwischen diesen Zahlen darstellen. Wir wählen beinahe die gleiche Kodierung wie in § 1.3.6.

Sei p eine Primzahl, $p \geq |Z|+|A|+2$ und $\psi: ZUAU\{/\} \to \{1,\ldots,p-1\}$ sei injektiv. Wir kürzen $\psi(a)$ durch $\hat{a}$ ab und setzen ψ auf $(ZUAU\{/\})^*$ fort durch $\psi(a_1 \ldots a_s) = \sum_{i=1}^{s} \hat{a}_i p^{s-i}$, $a_i \in ZUAU\{/\}$,

ψ (das leere Wort) = 0 (vgl. § 1.1.2).

$\hat{a}_1 \ldots \hat{a}_s$ ist also Darstellung der Zahl $\psi(a_1 \ldots a_s)$ zur Basis p. Wir drücken zuerst die Konkatenation, d.h. das Hintereinanderschreiben von Zeichenreihen als Relation zwischen den Kodenummern der Zeichenreihen aus.

Lemma 2.54: Es gibt ein arithmetisches Prädikat $z=[x,y]$, so daß für alle Zeichenreihen $X,Y \in (ZUAU\{/\})^*$ gilt: Ist $x=\psi(X)$ und $y=\psi(Y)$, so ist $z=[x,y]$ genau dann wahr, wenn $z=\psi(XY)$ ist.

Beweis: Für $y \in \mathbb{N}$ bezeichne $\psi^{-1}(y)$ die durch y kodierte Zeichenreihe, falls es eine solche Zeichenreihe gibt, und $L(y)=|\psi^{-1}(y)|$ bezeichne die Länge dieser Zeichenreihe. Wir definieren das gewünschte Prädikat in einer Reihe von Definitionsschritten, wobei wir immer Prädikate auf der linken Seite des $\equiv$-zeichens durch einfachere Prädikate auf der rechten Seite ersetzen. Dies wird fortgesetzt, bis alle Prädikate durch arithmetische Prädikate im Sinne von Definition 2.28 ersetzt sind. Wir lassen wieder nach den üblichen Prioritätsregeln Klammern weg. $\overline{p}$ bezeichne $1+\ldots+1$ (p mal).

$z=[x,y] \equiv (y=0 \wedge z=x) \vee (\sim y=0 \wedge \exists u((z=x.u+y) \wedge "(u=p^{L(y)})"))$

-- wir erinnern an $159346 = 159 \cdot 10^3 + 346$ im Dezimalsystem --.

$"(u=p^{L(y)})" \equiv "u$ ist Potenz von $p" \wedge "y<u" \wedge "u \leq p \cdot y"$

"u ist Potenz von p" $\equiv \forall r \forall s (r \cdot s = u \rightarrow r = 1 \vee$ "p teilt r")*)

"p teilt r" $\equiv \exists t (r = \overline{p} \cdot t)$.

"y < u" $\equiv \exists v (u = y + v)$

"u $\leq$ p·y" $\equiv u = \overline{p} \cdot y \vee \exists v (u = \overline{p} \cdot y + v)$. □

Entsprechend stellt man die Konkatenation von 3 oder mehr Worten dar durch

$$u=[x,y,z] \equiv \exists w (w=[x,y] \wedge u=[w,z])$$

usw.

Wir formulieren jetzt der Reihe nach die Eigenschaften (1) bis (5) der Zeichenreihe w als arithmetische Prädikate.

(1) $\equiv \exists k \exists m \exists a \exists b \exists c$
$(w=[\overset{\wedge}{/},k,\overset{\wedge}{/},m] \wedge k=[a,b,c] \wedge$
"a=ψ(B...B)" $\wedge$ "b=$\psi(z_o v)$" $\wedge$ "c=ψ(B...B)").

"a=ψ(B...B)" $\equiv \forall d$("d codiert ein einzelnes Zeichen in ψ^{-1}(a)"$\rightarrow d=\hat{B}$)

"d codiert ein einzelnes Zeichen in ψ^{-1}(a)"$\equiv d<\overline{p} \wedge (\exists e \exists f (a=[e,d,f]))$

"b=$\psi(z_o v)$" $\equiv \exists y_o \ldots \exists y_n$
$(y_o = z_o$
$\wedge y_1 = \underbrace{y_o + \ldots + y_o}_{(p\text{-mal})} + \hat{v}_1$
$\vdots$
$\wedge y_n = \overbrace{y_{n-1} + \ldots + y_{n-1}}^{(p\text{-mal})} + \hat{v}_n$
$\wedge b = y_n)$.

(2)$\wedge$(3) $\equiv \forall r \forall x \forall y \forall z$
$((w=[r,\overset{\wedge}{/},x,\overset{\wedge}{/},y,\overset{\wedge}{/},z]$
$\wedge$"/ kommt in x nicht vor"
$\wedge$"/ kommt in y nicht vor")
$\rightarrow$"x$\vdash$y").

"/ kommt in x nicht vor" $\equiv \forall d$ ("d codiert ein einzelnes Zeichen in ψ^{-1}(x)"$\rightarrow\sim d=\overset{\wedge}{/}$).

---

*) Dies ist nur für Primzahlen p korrekt.

$$\text{"}x\vdash y\text{"} \equiv \exists u \exists v$$

$$\Big(\bigvee_{\delta(z,a)=(z',b,N)} (x=[u,\hat{z},\hat{a},v] \wedge y=[u,\hat{z}',\hat{b},v])$$

$$\vee \bigvee_{\delta(z,a)=(z',b,R)} (x=[u,\hat{z},\hat{a},v] \wedge y=[u,\hat{b},\hat{z},v])$$

$$\vee \bigvee_{\delta(z,a)=(z',b,L)} \bigvee_{c\in A} x=[u,\hat{c},\hat{z},\hat{a},v] \wedge y=[u,\hat{z}',\hat{c},\hat{b},v])$$

Teilformeln der Form $\bigvee_{R(\beta)} A(\beta)$ stehen hier und auch später im Text als Abkürzung für $A(\beta_1) \vee \ldots \vee A(\beta_s)$, wobei $\{\beta_1,\ldots,\beta_s\}=\{\beta \mid R(\beta)\}$ eine endliche Menge ist.

$$(4) \equiv \forall x \forall y \forall z \forall d$$
$$((w=[x,\hat{/},y,\hat{/},z] \wedge \text{"}/ \text{ kommt in y nicht vor"}$$
$$\wedge \text{"d kodiert ein einzelnes Zeichen in } \psi^{-1}(y)\text{"} \wedge \bigvee_{q\in z} d=\hat{q})$$
$$\rightarrow\sim \exists e(y=[d,e] \vee y=[e,d])).$$

$$(5) \equiv \exists u,v \bigvee_{e\in E} w=[u,\hat{e},v].$$

Die arithmetische Aussage $\exists w((1)\wedge(2)\wedge(3)\wedge(4)\wedge(5))$ besagt dann gerade "M angesetzt auf x hält". Um Definition 2.28 formal zu erfüllen, müssen nur noch in allen Formeln die kleinen Buchstaben, die Variablen bezeichnen, durch Zeichenreihen der Form $xu, u\in\{0,1\}^*$, ersetzt werden.

Übungen

1. Man zeige: Z_E ist genau dann inkonsistent, wenn alle Prädikate aus L_E in Z_E herleitbar sind.
2. Definition: Ein Beweissystem S heißt ω-konsistent, wenn für kein Prädikat A(y) sowohl jede der Aussagen A(0), A(1), A(2), ... als auch $\exists y(\sim A(y))$ in S herleitbar ist.
   Man zeige: a) Ist Z_E ω-konsistent, so ist Z_E konsistent
   b) $Z_E'=(\Sigma_E, L_E, A_E \cup \sim \text{Consis } Z_E, S_E)$ ist konsistent, aber nicht ω-konsistent. "Konsistenz ist eine rauhe Approximation für Wahrheit, (Korrektheit) ω-Konsistenz eine etwas bessere" [5].

3. Man zeige: Die Menge {w|w ist in Z_E herleitbar} ist nicht entscheidbar.

4. Man zeige: a) Ist Z_E ω-konsistent, so ist Z_E unvollständig.
   b) Ist Z_E ω-konsistent, so kann man explizit Aussagen in L_E angeben, die in Z_E weder beweisbar noch widerlegbar sind.

5. Man gebe arithmetische Aussagen an, die folgendes besagen:
   a) i ist Primzahl
   b) Es gibt unendlich viele Primzahlzwillinge
   c) i ist die 3. Primzahl.

6. Sei M eine 1-Band-Turingmaschine, v eine Eingabe für M und $t \in \mathbb{N}$. Man gebe ein arithmetisches Prädikat an, das genau dann wahr ist, wenn M angesetzt auf v nach höchstens t Schritten hält.

3. Realistische Komplexitätsmaße

Wir haben bisher den Begriff der Berechenbarkeit formalisiert und einige nichtberechenbare Funktionen kennengelernt. Nun wollen wir die berechenbaren Funktionen nach ihrer Komplexität, d.h. nach dem zu ihrer Berechnung erforderlichen Rechenaufwand klassifizieren.

In Kapitel 1 haben ganz verschiedene Modelle (1-Band-Turingmaschinen, Mehrband-Turingmaschinen, μ-rekursive Funktionen) die gleiche Klasse von Funktionen definiert. Im Gegensatz dazu hängt die Komplexität von Funktionen sehr wohl vom zugrundeliegenden Modell ab; wir werden z.B. später zeigen, daß die Berechnung gewisser Funktionen auf 1-Band-Turingmaschinen mehr Zeit als auf 2-Band-Turingmaschinen erfordert.

Turingmaschinen studiert man gerne, weil sie besonders einfach sind. Wenn aber zumindest einige Komplexitätsaussagen vom zugrundeliegenden Maschinenmodell abhängen, dann muß man sich fragen, ob Turingmaschinen überhaupt hierfür ein realistisches Modell sind. Haben Komplexitätsaussagen über Turingmaschinen irgendeine Relevanz für reale Rechenanlagen? Ist eine Funktion, die auf Turingmaschinen leicht bzw. schwer berechenbar ist, deswegen auch auf einer modernen Rechenanlage leicht bzw. schwer berechenbar?

Um diese Fragen beantworten zu können, führen wir ein weiteres Maschinenmodell ein, das an der Struktur realer Rechenanlagen orientiert ist, und studieren dann, inwieweit sich Komplexitätsaussagen für Turingmaschinen auf dieses Modell und umgekehrt von diesem Modell auf Turingmaschinen übertragen lassen.

3.1. Registermaschinen

3.1.1. Beschreibung des Modells

Registermaschinen sind ein abstraktes Modell für die zentrale Recheneinheit und den Arbeitsspeicher von Rechenanlagen.

Eine Registermaschine besteht aus einer zentralen Recheneinheit, einem Speicher, einem Eingabeband, einem Ausgabeband und einem Programm. Der Speicher besteht aus abzählbar vielen Registern, die mit 1,2,... numeriert sind. Die Nummer eines Registers heißt auch seine Adresse. Jedes Register kann eine beliebige natürliche Zahl

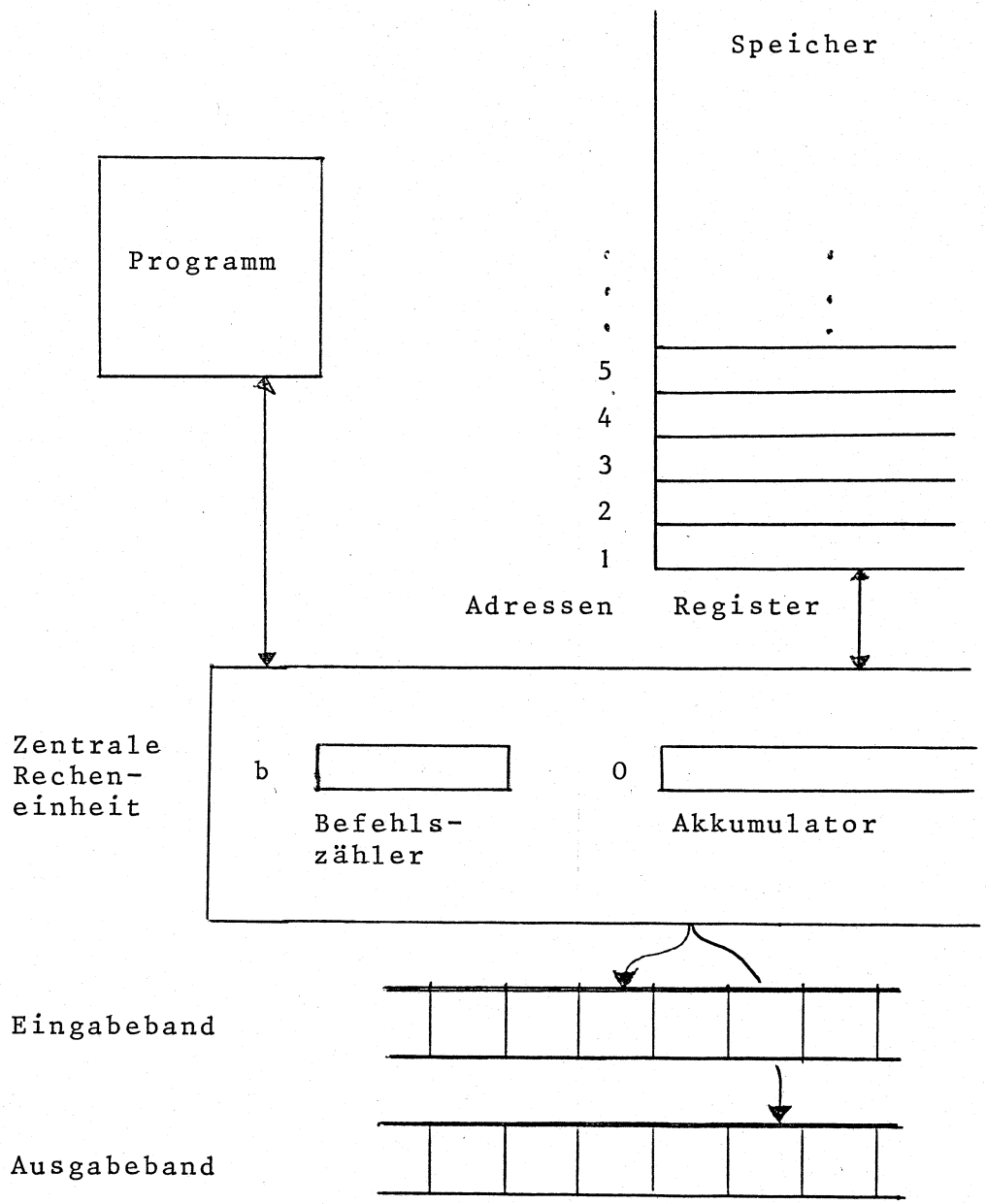


Bild: Aufbau einer Registermaschine

speichern. Das Programm ist eine endliche numerierte Folge von Befehlen aus einer Befehlsliste, die noch spezifiziert wird. Die zentrale Recheneinheit besteht aus einem speziellen Register mit Adresse 0, das auch Akkumulator genannt wird, und dem Befehlszähler. Der Akkumulator kann außer Zahlen auch einzelne Zeichen aus einem endlichen Alphabet A speichern. Der Befehlszähler kann eine Zahl speichern. Dieser Zahl entspricht in naheliegender Weise ein Befehl des Programms: steht die Zahl b im Befehlszähler, so

wird der mit b numerierte Befehl des Programms ausgeführt; der Inhalt des Befehlszählers entspricht dem Zustand von Turingmaschinen.

Die Ein- und Ausgabeinheiten sind (wie bei realen Rechenanlagen) etwas künstliche Anhängsel, die dafür sorgen, daß die Maschinen Funktionen berechnen, die Zeichenreihen in Zeichenreihen abbilden. Das Eingabeband kann Zeichenreihen $w \in A^*$ enthalten. Die Maschine kann den Kopf auf dem Eingabeband nur von links nach rechts bewegen (sie kann also jedes Zeichen nur einmal lesen) und kann nicht auf dem Band schreiben; man kann das Eingabeband als Modell für einen Lochkartenleser ansehen. Auf dem Ausgabeband kann die Maschine den Kopf nur von rechts nach links bewegen und Zeichen aus A drucken; das Ausgabeband kann man als Modell für einen Drucker ansehen.

Zu jedem Zeitpunkt ist eine Registermaschine vollständig beschrieben durch ein Quintupel (b,c,w,p,x): hierbei ist

$b \in \mathbb{N}$ der Inhalt des Befehlszählers;
$c: \mathbb{N} \to \mathbb{N}$ ist eine Funktion,
$c(i)$= Inhalt von Register i;
$w \in A^*$ ist der Inhalt des Eingabebands,
$p \in \{1,\ldots,|w|\}$ ist die Position des Lesekopfs auf dem Eingabeband und
$x \in A^*$ ist die Inschrift des Ausgabebands.

Zu Beginn jeder Rechnung ist der Befehlszähler gleich 1 und der Inhalt aller Register gleich 0.

In einem Schritt werden in Abhängigkeit von dem Befehl, der gerade ausgeführt wird, der Inhalt höchstens eines Registers, der Befehlszähler und eventuell die Position des Lesekopfs geändert oder die Inschrift des Ausgabebands verlängert.

Die folgende Tabelle enthält einige Befehle und ihre Wirkung.

Befehlsliste, 1. Teil:

Befehl	Wirkung
Load i	$c(0) \leftarrow c(i)$; $b \leftarrow b+1$
Store i	$c(i) \leftarrow c(0)$; $b \leftarrow b+1$
add i	$c(0) \leftarrow c(0)+c(i)$; $b \leftarrow b+1$
sub i	$c(0) \leftarrow c(0) \dot{-} c(i)$; $b \leftarrow b+1$
read	$c(0) \leftarrow w_p$; $p \leftarrow p+1$; $b \leftarrow b+1$
print a	$x \leftarrow ax$; $b \leftarrow b+1$ ($a \in A$)

shift	$c(0) \leftarrow \lfloor c(0)/2 \rfloor$
goto i	$b \leftarrow i$
if c(0)=i goto j	$b \leftarrow \begin{cases} j & \text{falls } c(0)=i \\ b+1 & \text{sonst } (i\in\mathbb{N} \text{ oder } i\in A) \end{cases}$
end	Ende der Rechnung

Beispiel 3.1: Auf dem Eingabeband steht eine Eingabe der Form $x\#y\#$, $x,y\in\{0,1\}^*$. Die Maschine soll x und y als Binärzahlen interpretieren und mit der Binärdarstellung der Summe dieser Zahlen auf dem Ausgabeband halten. Das folgende Programm tut das Gewünschte. Die Befehlsfolge 1-8 wird $|x|$-mal durchlaufen und verwandelt

$$x=x_1\ldots x_{|x|}\in\{0,1\}^* \text{ in } \sum_{i=1}^{|x|} x_i 2^{|x|-i}.$$

Ähnliches gilt für die Befehlsfolge 9-16 und y. Die eigentliche Addition findet in den Befehlen 17-18 statt. Die Befehle 19-32 erzeugen die Binärdarstellung der Summe auf dem Ausgabeband.

```
1    read
2    if c(0)=# goto 9
3    store 3
4    load  1          } Wirkung: c(0) ← 2·c(1)
5    add   1          }
6    add   3
7    store 1
8    goto  1
9    read
1o   if c(0)=# goto 17
11   store 3
12   load  2          } Wirkung: c(0) ← 2·c(2)
13   add   2          }
14   add   3
15   store 2
16   goto  9
17   load  1
18   add   2
```

```
19  store 1
2o  shift
21  store 2
22  add   2
23  store 3
24  load  1
25  sub   3
26  if c(0)=0 goto 29
27  print 1
28  goto  30
29  print 0
3o  load  2
31  if c(0)=0 goto 33
32  goto  19
33  end
```

$c(0) \leftarrow \text{Rest}\ (c(0),2)$

$c(2) \leftarrow \lfloor c(0)/2 \rfloor$

Wir erweitern die Befehlsliste, indem wir erlauben, vor die Befehle load, store, das Präfix ind (für indirekte Adressierung) zu schreiben, und vor die Befehle add, sub, load, die Präfixe ind oder c (für Konstanten) zu schreiben:

Befehlsliste, 2. Teil:

Befehl		Wirkung
ind load	i	$c(0) \leftarrow c(c(i))$; $b \leftarrow b+1$
ind store	i	$c(c(i)) \leftarrow c(0)$; $b \leftarrow b+1$
ind add	i	$c(0) \leftarrow c(0)+c(c(i))$; $b \leftarrow b+1$
ind sub	i	$c(0) \leftarrow c(0) \dot{-} c(c(i))$; $b \leftarrow b+1$
c load	i	$c(0) \leftarrow i$
c add	i	$c(0) \leftarrow c(0)+i$
c sub	i	$c(0) \leftarrow c(0) \dot{-} i$.

Beispiel 3.2: Das folgende Programm hält nie und füllt der Reihe nach die Register 3,4,5,... mit den Zahlen $2^3, 2^4, 2^5$,. Im Register 1 wird gezählt, wie weit man damit schon gekommen ist. Im Register 2 werden Zwischenergebnisse gespeichert.

```
1   c load 8
2   store  3
3   c load 3
4   store  1
5   ind load 1
```

6	ind add 1	} $c(2) \leftarrow 2 \cdot c(c(1))$
7	store 2	
8	load 1	} $c(1) \leftarrow c(1)+1$
9	c add 1	
1o	store 1	
11	load 2	} $c(c(1)) \leftarrow c(2)$
12	ind store 1	
13	goto 5	

Wir bemerken noch, daß unsere Befehlsliste nicht die kleinstmögliche ist, sondern der Bequemlichkeit halber einige Befehle enthält, die durch eine Folge von anderen Befehlen simuliert werden können.

3.1.2. Komplexitätsmaße für Registermaschinen

Wir definieren zuerst auf naheliegende Weise die Zeitkomplexität von Registermaschinen.

Definition 3.3: Sei M eine Registermaschine und $x \in \{0,1,2\}^*$. Die uniforme Zeitkomplexität von M angesetzt auf x ist die Anzahl von Schritten, die M angesetzt auf x macht.

In dieser Definition wird das Ausführen eines Schritts als eine Zeiteinheit gezählt, gleichgültig, welche Art von Befehl ausgeführt wird und wie große Zahlen dabei verarbeitet werden. Bis auf einen Faktor 2 wird gerade die Anzahl von Speicherzugriffen gemessen (ein Befehl mit indirekter Adressierung entspricht 2 Speicherzugriffen. Das ist ziemlich realistisch: in Rechenanlagen besteht (bei heutiger Technologie) die zentrale Recheneinheit aus sehr schnellen Bausteinen, und der Arbeitsspeicher besteht aus Kostengründen aus viel langsameren billigeren Bausteinen, so daß die Speicherzugriffszeit größer ist als die Zeit, die benötigt wird, um in der zentralen Recheneinheit eine Operation auszuführen.

In manchen Rechnungen treten Zahlen auf, die so groß sind, daß sie nicht mehr in ein Register einer (realen) Rechenanlage passen, sondern in mehreren Registern gespeichert werden müssen. Um diesem Fall Rechnung zu tragen, betrachtet man oft ein weiteres Kostenmaß, in dem man die Kosten eines Schritts gleich der Summe der Längen der Binärdarstellungen aller in dem Schritt verarbeiteten Registerinhalte setzt (die Befehle "goto i" und "end" erhalten

Kosten 1).

Die folgende Tabelle enthält einige Beispiele:

Befehl	Kosten
load i	$\|bin(c(0))\|+\|bin(c(i))\|$
add i	$\|bin(c(0))\|+\|bin(c(i))\|$
ind store i	$\|bin(c(0))\|+\|bin(c(i))\|+\|bin\ c(c(i))\|$
c add i	$\|bin(c(0))\|$
if c(0)=0 goto i	$\|bin(c(0))\|$

Definition 3.4: Es sei M eine Registermaschine und $x\in\{0,1,2\}^*$. Die logarithmische Zeitkomplexität von M angesetzt auf x ist die Summe der Kosten aller Schritte, die M angesetzt auf x macht.

Der Name logarithmische Zeitkomplexität wird benutzt wegen

$$|bin(i)| = \lfloor \log i \rfloor + 1.$$

Definition3.5: $t:\mathbb{N}\to\mathbb{N}$ sei eine Funktion. Eine Registermaschine M heißt uniform (logarithmisch) t(n)-zeitbeschränkt, wenn für alle $x\in\{0,1,2\}^*$ die uniforme (logarithmische) Zeitkomplexität von M angesetzt auf x kleiner oder gleich $t(|x|)$ ist.

Beispiel 3.6: Die Maschine in Beispiel 3.1 ist uniform O(n)-zeitbeschränkt und logarithmisch O(n log n)-zeitbeschränkt. Mancher Leser wird in der Befehlsliste für Registermaschinen Multiplikations- und Divisionsbefehle vermißt haben. Man studiert in der Tat auch Registermaschinen mit solchen Befehlen; aber in diesem Fall wird das uniforme Komplexitätsmaß unrealistisch:

Beispiel 3.7: In dem folgenden Programm habe der Befehl mult i die Wirkung $c(0)\leftarrow c(0)\cdot c(i)$.

```
1    c load 2
2    store  1
3    load   1
4    mult   1
5    store  1
6    goto   3
```

$$c(1) \leftarrow (c(1))^2$$

Bei jedem Durchlauf der Schleife 3-6 wird c(1) quadriert. Nach 4m+2 Schritten = m Durchläufen ist

$$c(1)=(\ldots((2)^2)^2\ldots)^2=2^{2\cdot\cdot\cdot 2}=2^{2^m}.$$

Damit hat man mit Kosten O(m) eine Zahl erzeugt, deren Binärdar-

stellung die Länge 2^m hat, was der Intuition widerspricht.

Analog zur Zeitkomplexität kann man für Registermaschinen ein uniformes und ein logarithmisches Maß für Speicherplatz definieren.

Definition 3.8: Sei M eine Registermaschine und $x \in \{0,1,2\}^*$. Die uniforme Platzkomplexität von M angesetzt auf x ist die Anzahl von Speicherzellen, die M angesetzt auf x benutzt.

Sei $M(x) = \max \{i \mid M \text{ angesetzt auf } x \text{ benutzt Register } i\}$ und $s(i,x) = \max\{|bin(j)| : j \text{ kommt während der Rechnung von M angesetzt auf x in Register i vor}\}$. Dann ist die logarithmische Platzkomplexität von M angesetzt auf x gleich $\sum_{i \leq M(x)} s(i,x)$.

M heißt uniform (logarithmisch) t(n)-platzbeschränkt, wenn für alle Eingaben $x \in \{0,1,\#\}^*$ die uniforme (logarithmische) Platzkomplexität von M angesetzt auf x durch $t(|x|)$ beschränkt ist.

Beispiel 3.9: Die Maschine aus Beispiel 3.1 ist uniform 3-platzbeschränkt und logarithmisch O(n)-platzbeschränkt.

Das uniforme Platzkomplexitätsmaß für Registermaschinen ist unrealistisch: es wird überhaupt nicht berücksichtigt, wie große Zahlen man in den Registern speichert, und in der Tat zeigt sich, daß jede berechenbare Funktion bereits in 2 Registern berechnet werden kann.

Übungen

1. a) Man verkürze das Programm aus Beispiel 3.1.
   b) Man gebe eine Registermaschine M an, die bei Eingabe $bin(k)\#bin(i_1)\#\ldots\#bin(i_k)\#$ mit $\sum_{i=1}^{k} i_j$ im Akkumulator hält.
   c) Man gebe uniforme und logarithmische Zeitkomplexität sowie die logarithmische Platzkomplexität der Maschine M an.

2. a) Man gebe ein Programm an, das die Register 5,6,7,... mit den Zahlen $2^5, 2^6, 2^7$ ... füllt und nicht die Befehle ind add und c add benutzt.
   b) Man gebe ein Programm an, das die Register 4,5,6,... mit den Zahlen $2^4, 2^6, 2^7$ ... füllt und nicht die Befehle ind add benutzt.

3. a) Man gebe eine uniform O(n)-zeitbeschränkte Registermaschine an, die bei Eingabe von bin(x)#bin(y)# mit dem Produkt von x und y im Akkumulator hält.
   b) Man gebe (größenordnungsmäßig) die logarithmische Zeitkomplexität dieser Maschine an.

4. a) Man gebe eine möglichst schnelle Registermaschine an, die gestartet mit den Zahlen x und y in den Registern 1 und 2 mit

$$f(x,y) = \begin{cases} \text{die y-te Stelle von bin(x)} & \text{falls } y \leq \lceil \log x \rceil \\ 0 & \text{sonst} \end{cases}$$

   im Akkumulator hält.
   b) Man gebe die uniforme und logarithmische Zeitkomplexität dieser Maschine als Funktion von n=(|bin(x)|+|bin(y)|) an.

5. Man löse Aufgabe 4 für f(x,y)=x·y.

3.2. Gegenseitige Simulation von Turingmaschinen und Registermaschinen

3.2.1. Polynomiell verknüpfte Funktionen

Den Anstoß zu diesem Kapitel über realistische Komplexitätsmaße gab die Frage, ob Komplexitätsaussagen, die für Turingmaschinen hergeleitet sind, auch für reale Rechenanlagen relevant sind. Diese Frage wird jetzt im wesentlichen positiv beantwortet:

Jede t(n)-zeitbeschränkte Registermaschine kann von einer "ungefähr t(n)"-zeitbeschränkten Turingmaschine simuliert werden (Satz 3.12) und jede t(n)-zeitbeschränkte Turingmaschine kann von einer "ungefähr t(n)"-zeitbeschränkten Registermaschine simuliert werden (Satz 3.15). Entsprechendes gilt für Speicherplatz. Deswegen ist alles, was auf Turingmaschinen schwer, d.h. nur mit großem Aufwand berechenbar ist, auch für Registermaschinen schwer berechenbar, und alles, was auf Turingmaschinen leicht berechenbar ist, ist auch für Registermaschinen leicht berechenbar.

Der Begriff "t'(n) ist ungefähr so groß wie t(n)" und wird präzisiert durch

Definition 3.1o: $t,t':\mathbb{N}\to\mathbb{N}$ seien Funktionen. t und t' heißen polynomiell verknüpft, falls es ein Polynom p gibt, so daß

$t(n) \leq p(t'(n))$ und
$t'(n) \leq p(t(n))$.

Beispiel 3.11: $t(n)=n$ und $t'(n)=n^3$ sind polynomiell verknüpft. $t(n)=n$ und $t'(n)=2^n$ sind nicht polynomiell verknüpft.

3.2.2. Simulation von Registermaschinen durch Turingmaschinen

Satz 3.12: a) Jede logarithmisch $t(n)$-zeitbeschränkte Registermaschine kann durch eine $O(t^2(n))$-zeitbeschränkte 3-Band-Turingmaschine simuliert werden.*)

b) Jede uniform $t(n)$-zeitbeschränkte Registermaschine kann von einer $O(t^3(n))$-zeitbeschränkten 3-Band-Turingmaschine simuliert werden.*)

Beweis: Gegeben sei das Programm P einer Registermaschine. Das Programm habe p Befehle. Wir beschreiben informal eine 3-Band-Turingmaschine M, die die Registermaschine Schritt für Schritt simuliert. Die Maschine M ist aus p+1 Maschinen $M_0,\ldots,M_p$ in einer Weise zusammengeschaltet, die noch spezifiziert wird. Für $1\leq j\leq p$ simuliert M_j gerade den j-ten Befehl des Programms der Registermaschine. Die Bänder 1 und 2 von M haben je eine Spur, Band 3 von M hat 2 Spuren. M simuliert die Registermaschine so, daß für alle i gilt:

(*) Ist die Registermaschine angesetzt auf Eingabe w nach i Schritten in Konfiguration (b,c,w,q,x) und ist $\{j|c(j)\neq 0\}=\{i_1,\ldots,i_m\}$, dann gilt nach der Simulation dieses Schritts durch M: Der Zustand von M ist der Anfangszustand von M_b, die Inschrift von Band 1 von M ist x, die Inschrift von Band 2 ist

$$\#\#0\#c'(0)\#\#bin(i_1)\#bin(c(i_1))\#\#\ldots\#\#bin(i_m)\#bin(c(i_m))\#\#$$

$$\text{mit } c'(0) = \begin{cases} bin(c(0)) & \text{falls } c(0)\in\mathbb{N} \\ c(0) & \text{sonst.} \end{cases}$$

Spur 1 von Band 3 enthält $w_q\ldots w_{|w|}$, d.h. den Teil der Eingabe, der noch gelesen wird, Spur 2 dient als Hilfsspeicherplatz für Zwischenrechnungen. Die Köpfe stehen am linken Ende der jeweiligen Bandinschrift.

Beispiel 3.13: Bei einer Registermaschine sei der Befehlszähler b=3, die Speicherzustandsfunktion sei

*) $t^i(n) := (t(n))^i$

$$c(i) = \begin{cases} 2i & \text{für } 0 \leq i \leq 3 \\ 0 & \text{sonst,} \end{cases}$$

die Eingabe w sei 1011, der Lesekopf stehe auf dem dritten Zeichen von w, d.h. q=3, und auf dem Ausgabeband stehe 1110. Dies wird auf der simulierenden Turingmaschine dargestellt durch

1110	Band 1
##0#0 ##1#10##10#100##11#110##	Band 2
	Band 3
11	Spur 1
Hilfsspeicherplatz	Spur 2

Der Anfangszustand von M ist der Anfangszustand von M_o. M_o kopiert die Eingabe von Band 1 auf Spur 1 von Band 3, löscht hierbei Band 1, schreibt auf Band 2 die Zeichenreihe ##0#0## und bewegt alle Köpfe an das linke Ende ihrer Inschrift. Damit gilt (*) für i=0.

Man müßte nun für jeden Befehl aus der Befehlsliste der Registermaschine angeben, wie er simuliert wird. Wir tun dies stellvertretend für 2 Befehlstypen:

a) Eine Maschine, die den Befehl add i simuliert, tut das folgende:
   - bin(i) wird auf Spur 2 von Band 3 geschrieben;
   - auf Band 2 wird eine Zeichenreihe ##bin(i)#c(i)## gesucht. Wird eine solche Zeichenreihe gefunden, so wird Spur 3 von Band 2 mit c(i) überschrieben, sonst mit 0.
   - Auf Band 2 wird die Zeichenreihe ##0#c'(0)## gesucht (linkes Ende der Bandinschrift). Der Inhalt von Spur 2 von Band 3 wird zu c'(0) addiert. Da hierbei c'(0) länger werden kann, muß eventuell die Zeichenreihe ##0# nach links verschoben werden.
   - Spur 2 von Band 3 wird gelöscht, alle Köpfe gehen an das linke Ende der Inschrift ihres Bandes.

b) Eine Maschine, die einen Befehl der Form "if c(0)=i then

goto..." simuliert, hat zwei Endzustände "ja" und "nein" und muß

- die Zeichenreihe ##0#c'(0)## auf Band 2 aufsuchen
- testen, ob c(0)=i gilt und das Ergebnis im endlichen Gedächtnis speichern (vgl. Beispiel 1.23)
- alle Köpfe an das linke Ende der Bandinschriften bewegen
- falls c(0)=i, gilt in Zustand "ja", sonst in Zustand "nein" gehen.

Die Maschinen M_j sind wie folgt zusammengeschaltet:

start → M_0;

$M_j \rightarrow M_k$	falls der j-te Befehl des Programms "goto k" ist
$M_j \overset{ja}{\rightarrow} M_k$ nein ↓ M_{i+1}	falls der j-te Befehl des Programms von der form "if c(0){=, >}i then goto k" ist
$M_j \rightarrow$ stop	falls der j-te Befehl des Programms "end" ist
$M_j \rightarrow M_{j+1}$	sonst.

Damit gilt (*) für i, falls (*) für i-1 gilt, und M simuliert die Registermaschine.

Zeitanalyse: Die Zeit zur Simulation eines Schritts der Registermaschine ist im ungünstigsten Fall proportional zur Länge der Inschrift von Band 2. Ist die Registermaschine logarithmisch t(n)-zeitbeschränkt, so ist auch diese Bandinschrift nie länger als 4t(n)+7, denn bei der Simulation eines Schritts mit Kosten k wird die Inschrift von Band 2 um höchstens k+3≤4k Felder verlängert. Es folgt Teil a) des Satzes. Ist die Maschine uniform t(n)-zeitbeschränkt, so können wir die Länge von Band 1 in folgender Weise abschätzen:

In einem Schritt der Registermaschine kann die Länge der größten Zahl, die in irgendeinem Register steht, um höchstens 1 wachsen, etwa durch Addition dieser Zahl zu sich selbst (dies gilt nicht mehr, wenn man Registermaschinen mit Multiplikation betrachtet, vgl. Beispiel 3.7). Nach t(n) Schritten ist also kein Registerinhalt länger als O(t(n)). Deshalb kann auch kein Register mit einer

Adresse, die länger als $O(t(n))$ ist, benutzt werden. Damit hat aber jeder Eintrag ##bin(i)#bin(c(i)) höchstens die Länge $O(t(n))$.

In $t(n)$ Schritten können höchstens $t(n)$ Register benutzt werden. Man erhält somit eine Schranke $O(t^2(n))$ für die Länge der Inschrift von Band 2 und $O(t^3(n))$ für die Gesamtdauer der Simulation. □

Korollar 3.14: a) Jede logarithmisch $t(n)$-zeitbeschränkte Registermaschine kann durch eine $O(t^4(n))$-zeitbeschränkte 1-Band-Turingmaschine simuliert werden.

b) Jede uniform $t(n)$-zeitbeschränkte Registermaschine kann durch eine $O(t^6(n))$-zeitbeschränkte 1-Band-Turingmaschine simuliert werden.

Beweis: Folgt unmittelbar aus Satz 3.12 und Satz 1.37. □

3.2.3. Simulation von Turingmaschinen durch Registermaschinen

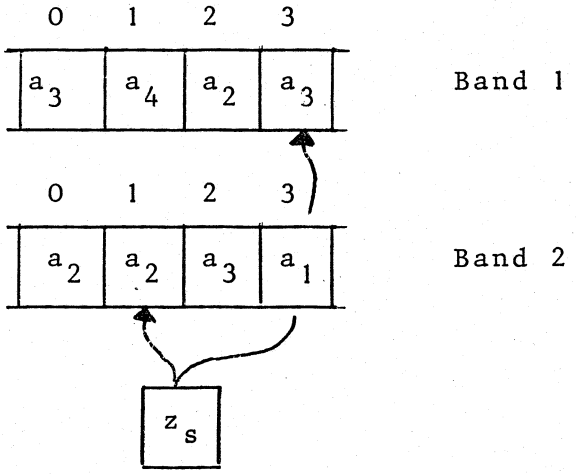
Satz 3.15: Jede $t(n)$-zeitbeschränkte Mehrband-Turingmaschine kann durch eine uniform $O(t(n))$-zeitbeschränkte bzw. logarithmisch $O(t(n)\log t(n))$-zeitbeschränkte Registermaschine simuliert werden.

Beweis: Sei $M=(Z,A,\delta,z_0,E)$ die k-Bandmaschine, die simuliert werden soll, $Z=\{z_1,\ldots,z_t\}$ und $A=\{a_1,\ldots,a_{|A|}\}$. Wir stellen uns die Felder eines jeden Bandes von M mit den ganzen Zahlen $\ldots-2,-1,0,1,2,\ldots$ durchnumeriert vor. Nach Satz 1.34 kann OBdA angenommen werden, daß M auf keinem Band Felder links vom Feld 0 benutzt. $C\in\mathbb{N}$ sei eine Konstante, $C>2k+1$. Die Registermaschine simuliert M so, daß für alle i gilt:

(*) Ist die Maschine M nach dem i-ten Schritt in Zustand z_s, steht der Kopf von Band j auf Zelle r und ist a_m der Inhalt von Zelle p von Band j, so gilt nach Simulation dieses Schritts durch die Registermaschine: $c(j)=r$, $c(k+1)=s$ und $c(C+kp+j)=m$.

DieRegister $k+2,\ldots,C-1$ können für Zwischenrechnungen benutzt werden.

Beispiel 3.16:



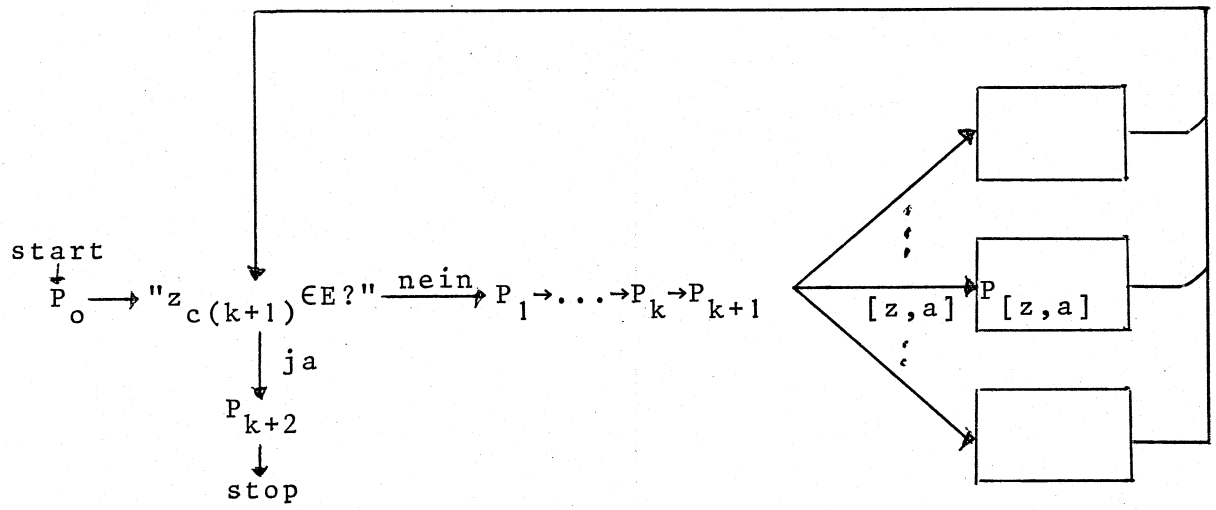
wird auf der Registermaschine dargestellt durch

i	c(i)
C +8	1
C +7	3
C +6	3
C +5	2
C +4	2
C +3	4
C +2	2
C +1	3
⋮	
3	s
2	1
1	3

Konstruktion des Programms der simulierenden Maschine: Eine fortlaufend, aber nicht notwendig von 1 an durchnumerierte Folge von Befehlen nennen wir ein Programmstück. Sind $P_1,\ldots,P_\ell$ Programmstücke, wobei die Befehle von P_1 von 1 bis p_1, die Befehle von P_2 von p_1+1 bis p_2... und die Befehle von P_ℓ durchnumeriert sind, so kann man diese Programmstücke - ebenso wie man dies bei Turingmaschinen tut - zu einem einzigen Programm zusammenschalten. Will man etwa den Befehl "k end" in P_i mit dem ersten Befehl von P_j

identifizieren, so ersetzt man den "end"-Befehl einfach durch "k goto $p_{j-1}+1$". Hierfür schreiben wir $P_i \overset{k}{\rightarrow} P_j$.

Das Programm der simulierenden Registermaschine ist ähnlich aufgebaut wie die simulierende Turingmaschine im Beweis von Satz 1.37.



P_o liest die Eingabe und sorgt dafür, daß (*) für i=0 erfüllt ist. Das Programmstück "$z_{c(k+1)} \in E$?" hat zwei "end"-Befehle, deren Nummern wir mit "ja" und "nein" bezeichnen. Ist c(k+1) die Nummer eines Endzustands von M , wird der "end"-Befehl mit Nummer "ja", sonst der "end"-Befehl mit Nummer "nein" erreicht.

Für $1 \leq j \leq k$ ist der Effekt von P_j: $c(k+1+j) \leftarrow c(C+c(j).k+j)$, d.h. die Nummer des Zeichens, das die Turingmaschine gerade auf Band j liest, wird in das Register c(k+1+j) geholt.

Für jedes Tupel $(z,a) \in Z \times A^k$ hat das Programmstück P_{k+1} einen eigenen "end"-Befehl, dessen Nummer wir mit [z,a] bezeichnen. Ist $(c(k+1),c(k+2),\ldots,c(2k+1))=(s,i_1,\ldots,i_k)$, so erreicht P_{k+1} den Endbefehl mit Nummer $[z_s,a_{i_1},\ldots,a_{i_k}]$. (Wenn man das Programm P_{k+1} geschickt organisiert, kommt man mit $|Z|.|A|^k+1$ Befehlen der Form "if c(0)=i goto j" aus).

Programm $P_{[z_s,a_{i_1},\ldots,a_{i_k}]}$ simuliert, was die Turingmaschine tut, wenn sie im Zustand z_s die Zeichen $a_{i_1},\ldots,a_{i_k}$ liest. d.h. ist $\delta(z_s,a_{i_1},\ldots,a_{i_k})=(z_r,a_{\ell_1},\ldots,a_{\ell_k},s_1,\ldots,s_k)$, so ist der Effekt dieses Programmstücks:

$c(k+1) \leftarrow r$,

$c(C+c(j)\cdot k+j) \leftarrow \ell j \qquad (1 \le j \le k)$

$$c(j) \leftarrow \begin{cases} c(j)-1 & \text{falls } s_j = L \\ c(j) & \text{falls } s_j = N \\ c(j)+1 & \text{falls } s_j = R \end{cases} \qquad (1 \le j \le k).$$

Damit gilt (*) für i, falls (*) für i-1 gilt. Enthält nach der Simulation eines Schritts von M Register k+1 die Nummer eines Endzustands, so gibt P_{k+2} die Inschrift von Band 1 von M aus (die Kodierung dieser Inschrift steht in den Registern

$$C+1, C+k+1, \ldots, C+Lk+1 \text{ mit } L=\min\{\ell \mid c(C+\ell\cdot k+1)=0\}).$$

Da jedes der Programmstücke $P_1, \ldots, P_{k+1}$ und jedes Programmstück der Form $P_{[z,a]}$ nur eine konstante Anzahl von Schritten macht, wird jeder Schritt der Turingmaschine mit einer konstanten Anzahl von Schritten simuliert. Die Anzahl der Schritte von P_o bzw. P_{k+2} ist proportional zur Länge der Eingabe bzw. des Ergebnisses. Hieraus folgt die Behauptung für das uniforme Kostenmaß.

Da die Turingmaschine in t(n) Schritten kein Feld mit einer Nummer größer als t(n) besucht, ist die größte Registeradresse, die bei der Simulation vorkommt, $C+kt(n)+k=O(t(n))$. Der größte Registerinhalt, der während der Simulation vorkommt, ist ebenfalls höchstens $O(t(n))$. Die logarithmischen Kosten eines jeden Schritts während der Simulation können also durch $O(\log t(n))$ abgeschätzt werden. Hieraus folgt die Abschätzung für das logarithmische Kostenmaß. □

Korollar 3.17: Die auf Registermaschinen berechenbaren Funktionen sind genau die turingberechenbaren Funktionen.

Insbesondere folgt - selbst wenn man die Church'sche These nicht akzeptiert -, daß alle Funktionen, die auf Turingmaschinen nicht berechenbar sind, auch auf modernen Rechenanlagen nicht berechnet werden können.

Wir nennen zwei Komplexitätsmaße polynomiell verknüpft, wenn es ein Polynom p gibt, so daß jede Funktion, die bezüglich eines der Maße mit Komplexität t(n) berechenbar ist, bezüglich des anderen Maßes mit Komplexität p(t(n)) berechenbar ist. Die Sätze 1.37, 3.12 und 3.15 kann man nun so zusammenfassen:

Satz 3.18: Zeitkomplexität von 1-Band-Turingmaschinen, Zeitkomplexität von Mehrband-Turingmaschinen sowie uniforme und logarith-

mische Zeitkomplexität von Registermaschinen sind alle polynomiell verknüpft.

Übungen

1. Man verschärfe Korollar 3.14 b).

2. Man formuliere und beweise zu den Sätzen 3.12 und 3.15 analoge Sätze für Bandkomplexität von Turingmaschinen und logarithmische Platzkomplexität von Registermaschinen.

3. Man kann statt Turingmaschinen mit linearen (1-dimensionalen) Bändern auch Turingmaschinen mit mehrdimensionalen Bändern betrachten und hierfür in naheliegender Weise ein Zeitkomplexitätsmaß definieren.
   Man zeige: Die Zeitkomplexität von Turingmaschinen mit 1-dimensionalen Bändern und die Zeitkomplexität von Turingmaschinen mit mehrdimensionalen Bändern sind polynomiell verknüpft.

4. Man gebe ein Schema zum Gödelisieren von Registermaschinen an.

5. Für $u \in \{0,1\}^*$ sei R_u eine Registermaschine mit Gödelisierung u, falls es eine solche Maschine gibt; sonst sei R_u eine Maschine, die für keine Eingabe hält.
   Man zeige: Es gibt eine universelle Registermaschine U, so daß für alle $u \in \{0,1\}^*$ und $v \in \{0,1,\ \}^*$ gilt:
   a) U angesetzt auf u#v hält genau dann, wenn R_u angesetzt auf v hält. Erzeugt R_u angesetzt auf v eine Ausgabe $\in \{0,1,\#\}^*$, so erzeugt U angesetzt auf u#v die gleiche Ausgabe.
   b) Macht R_u angesetzt auf v t Schritte, so macht U angesetzt auf u#v $C_u \cdot t$ Schritte, wobei C_u nur von u abhängt.
   c) wie b) für logarithmische Zeitkomplexität.

6. Man zeige: Die logarithmischen Zeitkostenmaße für Registermaschinen mit und ohne Multiplikation/Division sind polynomiell verknüpft.

7. Ein Pushdown-Band ist ein nach links unendliches Band, auf dem ein Lese-Schreibkopf operiert. Das rechte Bandende ist mit einem Sonderzeichen markiert. Jede Rechnung beginnt mit dem Kopf auf dem Sonderzeichen. Bewegt sich der Kopf nach links, muß ein vom Sonderzeichen verschiedenes Zeichen gedruckt werden. Bewegt sich der Kopf nach rechts, muß das zuvor gelesene Zeichen gelöscht (mit dem Leerzeichen überdruckt) werden.

Ein k-Pushdown-Automat ist eine k+1-Band-Turingmaschine für die gilt:

i) Auf Band 1, wo die Eingabe steht, darf wie beim Eingabeband von Registermaschinen der Kopf nur von links nach rechts bewegt werden.

ii) Die Bänder 2,...,k+1 sind Pushdown-Bänder.
Man zeige:

a) Es gibt einen 1-Pushdown Automaten, der Eingaben $x\in\{0,1,\#\}^*$ testet, ob sie von der Form $x=x_1\ldots x_s\#x_s\ldots x_1, x_i\in\{0,1\}$ sind.

b) Jede 1-Band-Turingmaschine kann durch einen 2-Pushdown-automaten simuliert werden (das eine Pushdownband enthält den Teil der Bandinschrift der Turingmaschine rechts vom Kopf, das andere den Teil links vom Kopf).

Ein Zähler ist ein Pushdown-Band, auf das nicht geschrieben werden darf. Ein k-Zähler Automat ist ein k-Pushdown-Automat, dessen sämtliche Pushdown-Bänder Zähler sind. Der Abstand s des Kopfs vom Sonderzeichen kann als die natürliche Zahl s interpretiert werden. In einem Schritt kann s um 1 erhöht werden oder um 1 erniedrigt werden, und durch Abfragen, ob der Kopf auf dem Sonderzeichen steht, kann "s=0?" getestet werden.

Für $a\in\{0,1\}$ und $n\in\mathbb{N}$ bezeichne a^n die Zeichenreihe $\underbrace{a\ldots a}_{(n\text{-mal})}$.

Man zeige:

c) Es gibt einen 1-Zähler-Automaten, der Eingaben $x\in\{0,1\}^*$ testet, ob sie von der Form $x=0^n1^n, n\in\mathbb{N}$, sind.

d) Jeder k-Pushdown-Automat kann durch einen 2k-Zählerautomaten simuliert werden. (Je 2 Zähler simulieren ein Pushdown-Band. OBdA kann für das Bandalphabet A des Pushdown-Automaten $A\subset\{1,2,\ldots\}$ angenommen werden. Die Bandinschrift $w=w_1\ldots w_s, w_i\in A$, wird als $\sum_{i=1}^{s} w_i(|A|+1)^{i-1}$ dargestellt).

e) Jeder k-Zähler-Automat kann durch einen 2-Zähler-Automaten simuliert werden ($P_1,\ldots,P_k$ seien k verschiedene Primzahlen. Die Zählerstände $z_1,\ldots,z_k$ werden als $\prod_{i=1}^{k} P_i^{z_i}$ dargestellt).

f) Jede berechenbare Funktion kann auf einer Registermaschine mit 2 Registern berechnet werden.

3.3. Schnelle Simulation von Turingmaschinen durch Registermaschinen

In diesem Abschnitt wird Satz 3.15 um einen Faktor log t(n) verschärft (alle Logarithmen in diesem Buch sind zur Basis 2).

Satz 3.19: Ist M eine t(n)-zeitbeschränkte k-Band-Turingmaschine und erzeugt M angesetzt auf Eingaben der Länge n Ausgaben der Länge höchstens L(n), so kann M von einer uniform $O(n+L(n)+t(n)/\log t(n))$-zeitbeschränkten und logarithmisch $O(n \log n+L(n)\log t(n)+t(n))$-zeitbeschränkten Registermaschine simuliert werden.

Für den Beweis benötigen wir einige Vorbereitungen.

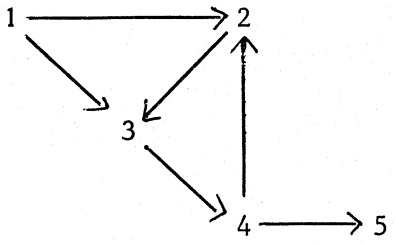
3.3.1. Graphen

Wir führen einige einfache Begriffe aus der Graphentheorie ein.

Definition 3.2o: Ein gerichteter Graph wird spezifiziert durch ein Paar G=(V,E). V ist eine Menge und heißt Menge der Knoten von G. $E \subset V \times V$ heißt Menge der Kanten von G. G heißt endlich, falls V endlich ist.

Endliche gerichtete Graphen kann man aufmalen, indem man die Knoten als Punkte darstellt und für jede Kante $(i,j) \in E$ einen Pfeil von Punkt i nach Punkt j malt.

Beispiel 3.21:



$G=(V,E)$,

$V=\{1,2,3,4,5\}$,

$E=\{(1,2),(2,3),(1,3),(3,4),$
$(4,2),(4,5)\}$.

Sei G=(V,E) ein gerichteter Graph. Ein Knoten $v \in V$ heißt Quelle von G, falls für alle $u \in V$ gilt: $(u,v) \notin E$. Ein Knoten v heißt Senke von G, falls für alle $u \in V$ gilt: $(v,u) \notin E$. Sei $\{v_1,\ldots,v_n\} \subset V$ eine Menge von Knoten, und für alle $i \in \{1,\ldots,n-1\}$ sei die Kante (v_i,v_{i+1}) in E. Dann heißt die Folge von Kanten

$$p=((v_1,v_2),\ldots,(v_{n-1},v_n))$$

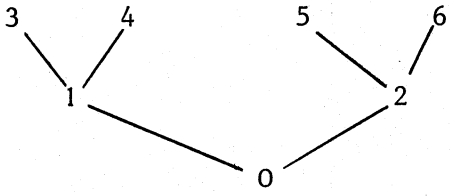
ein Pfad von v_1 nach v_n; n heißt die Länge des Pfades. Ist $v_n=v_1$, so heißt p ein Zyklus. Ein Graph, in dem es keine Zyklen gibt, heißt zykelfrei. In Beispiel 3.21 ist 1 eine Quelle, 5 ist eine Senke und ((3,4),(4,2),(2,3)) ist ein Zyklus der Länge 3.

Definition 3.22: Ein gerichteter Baum ist ein gerichteter Graph G=(V,E), für den gilt:
i) G hat genau eine Quelle $w\in V$
ii) für alle Knoten $v\in V\setminus\{w\}$ gibt es genau einen Pfad von w nach v.

Der Knoten w heißt Wurzel des Baums. Die Senken von Bäumen heißen Blätter, die übrigen Knoten heißen innere Knoten. Für Knoten $v\in V\setminus\{w\}$ heißt die Länge des Pfads von w nach v die Tiefe von v; die Tiefe der Wurzel wird per Definition gleich 0 gesetzt. Die Tiefe eines endlichen Baums wird definiert als max{Tiefe von v|$v\in V$}. Ein einzelner Knoten ist ein Baum der Tiefe 0.

Für $(u,v)\in E$ sagt man auch: v ist ein Sohn von u, und u ist der Vater von v. Sei $k\in\mathbb{N}$. Ein k-närer Baum ist ein Baum, in dem jeder Knoten höchstens k Söhne hat. Ein vollständiger k-närer Baum ist ein Baum, in dem jeder innere Knoten genau k Söhne hat und in dem alle Blätter die gleiche Tiefe haben.

Beispiel 3.23: Ein vollständiger binärer Baum mit Tiefe 3



Durch Induktion über d zeigt man leicht, daß jeder vollständige k-näre Baum der Tiefe d genau k^d Blätter und $\sum_{i=0}^{d-1} k^i=(k^{d-1}-1)/(k-1)$ innere Knoten hat.

3.3.2. Suchbäume

In einem vollständigen W-nären Baum der Tiefe d seien für jeden inneren Knoten die Kanten, die von ihm ausgehen, von 0 bis W-1 durchnumeriert. Für $r\leq d$ beschreibt damit jedes r-Tupel $e=(e_1,\ldots,e_r)\in\{0,\ldots,W-1\}^r$ einen von der Wurzel verschiedenen Knoten v(e) des Baums, nämlich den Knoten, den man erreicht, wenn man von der Wurzel des Baums aus erst der Kante mit Nummer e_1 folgt, dann der Kante mit Nummer e_2 usw.

Beispiel 3.24: Numeriert man für jeden inneren Knoten in dem Baum aus Beispiel 3.23 die nach links oben gehende Kante mit 0 und die nach rechts oben gehende Kante mit 1, so ist $v(0,0)=3$ und $v(1)=2$. Wir beschreiben nun ein Schema zum Abspeichern von Funktionen $f:\{0,\ldots,W-1\}^d \to \mathbb{N}$ auf Registermaschinen, das einem solchen Baum nachgebildet ist. Intuitiv wird für alle $e\in\{0,\ldots,W-1\}^d$ der Funktionswert $f(e)$ im Blatt $v(e)$ gespeichert. Das Speichern bzw. Lesen eines Funktionswerts $f(e)$ wird dann dem Durchlaufen eines Pfads von der Wurzel zu einem Blatt entsprechen und in $O(d)$ Schritten möglich sein.

Wir sagen, daß auf einer Registermaschine ein W-närer Suchbaum (V,E) der Tiefe d implementiert ist , falls gilt:

> Jedem inneren Knoten v entsprechen W aufeinanderfolgende Register mit Adressen $A(v),\ldots,A(v)+W-1$, jedem Blatt v entspricht ein Register $A(v)$; ist im Baum w ein Sohn von v und ist die Kante von v nach w mit i markiert, so ist $c(A(v)+i)=A(w)$.

Hat man einmal einen solchen Suchbaum implementiert und speichert man für $f:\{0,\ldots,W-1\}^d \to \mathbb{N}$ und $e=(e_1,\ldots,e_d)\in\{0,\ldots,W-1\}^d$ den Funktionswert $f(e)$ in Register $A(v(e))$, so kann man für beliebige Register r Befehle mit Wirkung $f(e)\leftarrow c(r)$ oder $c(r)\leftarrow f(e)$ in $O(d)$ Schritten ausführen; denn ist w die Wurzel des Baums und $H=A(w)$, so ist $A(v(e))=c(c(\ldots c(H+e_1)+e_2)\ldots+e_d))$.

Beispiel 3.25: Markiert man die Kanten des Baums aus Beispiel 3.23 wie in Beispiel 3.24 beschrieben, so ist folgendes eine Implementierung dieses Baums mit $A(v)=H+2v$ $(0\le v\le 6)$, wobei in den Blättern des Baums die Funktion $f:\{0,1\}^2\to\{0,1\}$, $f(i,j)=i+j$ gespeichert ist.

i	c(i)
H + 6	0
H + 5	H + 12
H + 4	H + 10
H + 3	H + 8
H + 2	H + 6
H + 1	H + 4
H	H + 2

i	c(i)
H + 12	2
H + 11	
H + 10	1
H + 9	
H + 8	1
H + 7	

Es ist $f(0,1)=c(c(c(H+0)+1))=c(c(H+3))=c(H+8)=1$.

Dieses Schema kann man zum Implementieren beliebiger Suchbäume verallgemeinern.

Lemma 3.26: Setzt man in einer Registermaschine für $H,W,d\in\mathbb{N}$ und alle $m\in\{0,\ldots,W\sum_{i=0}^{d-1}W^i-1\}$ $c(H+m)=H+W\cdot(m+1)$, so implementiert man damit einen W-nären Suchbaum der Tiefe d.

Beweis: Für $0\le j\le d$ seien die Graphen $G_j=(V_j,E_j)$ definiert durch

$$V_j = \{0,\ldots,\sum_{i=0}^{j}W^i-1\},$$

$$E_0 = \emptyset,$$

$$E_j = \{(u,v)\in V_{j-1}\times V_j \mid H+W\cdot v\in\{c(H+W\cdot u+i)\mid 0\le i\le W-1\}\}.$$

Durch Induktion über j zeigt man leicht, daß G_j ein vollständiger W-närer Baum der Tiefe j ist. Markiert man in G_j die Kante (u,v) mit i, falls $H+W\cdot v=c(H+W\cdot u+i)$ gilt, so erhält man einen Suchbaum, der mit $A(v)=H+W\cdot v$ für alle $v\in V_j$ implementiert ist. □

Satz 3.27: Man kann ein Programmstück schreiben, das, gestartet mit Zahlen H,W,d in irgendwelchen fest vorgegebenen Registern, mit $O(W^d)$ Schritten in den Registern H,H+1,... einen W-nären Suchbaum der Tiefe d aufbaut.

Beweis: Die einzige Schwierigkeit, die Implementierung aus Lemma 3.26 in ein Programm für Registermaschinen umzusetzen, kommt daher, daß man keine Multiplikationsbefehle zur Verfügung hat. Wegen $W^i=W^{i-1}+\ldots+W^{i-1}$ (W-mal) kann man $W^1,W^2,\ldots,W^i$ in $O(W\cdot i)$ Schritten und $W\sum_{i=0}^{d-1}W^i$ in $O(W\cdot d+d)=o(W^d)$ Schritten berechnen. Die Register H+m besetzt man durch: $c(H)=H+W$, $c(H+m)=c(H+m-1)+W$ für $1\le m\le W\cdot\sum_{i=0}^{d-1}W^i-1$. □

3.3.3. Beweis von Satz 3.19

Sei $M=(Z,A,\delta,z_0,E)$ mit $Z=\{z_1,\ldots,z_r\}$ und $A=\{a_1,\ldots,a_{|A|}\}$ die k-Bandmaschine, die simuliert werden soll. OBdA benutzt M auf keinem Band Felder links von der ursprünglichen Position der Köpfe. Das Verhalten der Maschine in einem Schritt ist vollständig durch die endliche Übergangsfunktion $\delta:Z\times A^k\to Z\times(A\cup\{R,L,0\})^k$ definiert. Wir stellen uns die Felder eines jeden Bandes mit den ganzen Zahlen numeriert vor und teilen jedes Band in Blöcke der Länge s ein,

wobei Block i aus den Feldern mit Nummern si,...,s(i+1)-1 besteht ($i\in\mathbb{N}$). Steht ein Kopf in Block i, so kann er innerhalb der nächsten s Schritte nur Felder in den Blöcken i, i-1 und i+1 erreichen. Der Inhalt von 3 Blöcken ist eine Zeichenreihe aus A^{3s}. Sei $K_s=Z\times\{1,\ldots,s\}^k\times(A^{3s})^k$. Das Verhalten der Maschine M in s Schritten kann durch eine endliche Funktion $\delta_s:K_s\to K_s\times\{0,1,2\}^k$ beschrieben werden. Für $z,z'\in Z$, $r_i\in\{1,\ldots,s\}$, $p_i\in\{0,1,2\}$ und $a_j,b_j,c_j,a_j',b_j',c_j'\in A^s$ bedeutet

$$\delta_s(z,r_1,\ldots,r_k,a_1,b_1,c_1,\ldots,a_k,b_k,c_k) =$$

$$(z',r_1',\ldots,r_k',a_1',b_1',c_1',\ldots,a_k',b_k',c_k',p_1,\ldots,p_k):$$

wird M im Zustand z mit Inschrift $a_jb_jc_j$ auf Band j und dem Kopf auf dem r_j-ten Zeichen von b_j gestartet ($1\le j\le k$), so ist M nach s Schritten (bzw. nach s'<s Schritten, falls M schon nach s' Schritten hält) im Zustand z', die Bandinschriften sind $a_j'b_j'c_j'$ und der Kopf auf Band j steht auf dem r_j'-ten Zeichen von

$$\begin{cases} b_j' & \text{falls } p_j=0 \\ c_j' & \text{falls } p_j=1 \\ a_j' & \text{falls } p_j=2 \end{cases} \qquad (1\le j\le k).$$

Idee der schnellen Simulation ist, für passendes s die Funktion δ_s im voraus zu berechnen; danach kann man s Schritte der Turingmaschine stets in einer konstanten Zahl von Schritten durch Nachsehen in der Tabelle von δ_s simulieren.

Konfigurationen der Turingmaschine stellt man ähnlich wie im Beweis von Satz 3.15 dar, jedoch speichert man in einem Register nicht mehr den Inhalt einer Bandzelle, sondern den Inhalt eines ganzen Blocks; $a=a_{i_1}\ldots a_{i_s}\in A^s$ wird als $\psi_s(a):=\sum_{j=1}^{s} i_j\cdot(|A|+1)^{s-j}\in\mathbb{N}$ gespeichert (vgl. § 1.3.6. Das Kodieren bzw. Dekodieren einer Zeichenreihe a kann auf einer Registermaschine nach dem Muster der Befehle 9-16 bzw. 19-32 aus Beispiel 3.1 in $O(|a|)$ Schritten durchgeführt werden). Statt der Position eines Kopfs speichert man die Nummer des Blocks, in dem sich der Kopf befindet und den Abstand des Kopfs vom linken Ende des Blocks.

Das Programm P der simulierenden Maschine ist aus zwei Programmstücken zusammengesetzt: $P="P_1;P_2"$. Ist $w=a_{i_1}\ldots a_{i_n}\in A^*$ eine Eingabe, so liest P_1 die Eingabe und speichert für eine groß genug

gewählte Konstante C_o die Zahlen $n,i_1,\ldots,i_n$ in den Registern $C_o,C_o+1,\ldots,C_o+n$. Die Register 1 bis C_o-1 können von allen Programmstücken zum Speichern von Zwischenergebnissen benutzt werden. Das entscheidende technische Ergebnis dieses Abschnitts ist

Lemma 3.28: Man kann ein Programmstück Q angeben, das, gestartet mit $c(C_o-1)=t$ und $(c(C_o),c(C_o+1),\ldots,c(C_o+n))=(n,i_1,\ldots,i_n)$ folgendes tut:

Hält M angesetzt auf $w=a_{i_1}\ldots a_{i_n}$ nach höchstens t Schritten mit Inschrift v auf Band 1, so hält Q nach $O(n+t/\log t+|v|)$ Schritten mit $c(C_o-1)=0$ und Inschrift v auf dem Ausgabeband; andernfalls hält Q nach $O(n+t/\log t)$ Schritten mit $c(C_o-1)=t$.

Beweis: i) Q berechnet zunächst einige Zahlen:
$s=\lfloor c\cdot\log t\rfloor$, wobei c eine feste rationale Konstante ist, die noch angegeben wird,
$W=\max\{|Z|+1,s+1,(|A|+1)^s,5k+2\}$, $d=4k+2$ und $B=W\cdot\sum_{i=o}^{d}W^i$.

Der Speicher wird in 4 Zonen eingeteilt: Zone 1 von Register 1 bis C_o-2, Zone 2 von Register C_o-1 bis $C_1=C_o+n$, Zone 3 von Register C_1+1 bis $C_2=C_1+B$ und Zone 4 von Register C_2+1 an. In Zone 3 wird ein W-närer Suchbaum der Tiefe d implementiert.

ii) Aus der Funktion δ_s gewinnt man vermittels der Kodierung ψ_s in natürlicher Weise eine zahlentheoretische Funktion Δ_s: für $S\in K_s$ oder $S\in K_s\times\{o,1,2\}^k$ gehe $\psi_s(S)$ aus S hervor, indem jede Komponente a von S aus A^s durch $\psi_s(a)$ und jede Komponente $z_\ell\in Z$ durch ℓ ersetzt wird. Sei $\psi_s(K_s)=\{\psi_s(S)\mid S\in K_s\}\subseteq\{0,\ldots W-1\}^{4k+1}$.
$\Delta_s:\psi_s(K_s)\times\{1,\ldots,5k+1\}\to\mathbb{N}$ wird definiert durch
$\Delta_s(\psi_s(S),i)=$ i-te Komponente von $\psi_s(\delta_s(S))$.

Die Funktion Δ_s soll tabelliert und in dem Suchbaum gespeichert werden. Hierzu zählt Q in Zone 1 systematisch alle Tupel T aus $\{0,\ldots,W-1\}^{4k+1}$ auf. Für jedes Tupel T wird getestet, ob T von der Form $T=\psi_s(S)$ mit $S\in K_s$ ist. Verläuft dieser Test positiv, wird $\psi_s(\delta_s(S))$ berechnet, indem in Zone 4 nach dem Verfahren von Satz 3.15 die Turingmaschine M für s Schritte simuliert wird. Für $1\le i\le 5k+1$ wird die i-te Komponente von $\psi_s(\delta_s(S))$ im Blatt $v(S,i)$ des Suchbaums gespeichert.

iii) Q simuliert M so, daß für alle $i \leq t$ gilt:

(*) Ist die Maschine nach dem $s \cdot i$-ten Schritt im Zustand z_ℓ, steht der Kopf von Band j auf Zelle r von Block m und ist die Inschrift von Block p von Band j $b \in A^s$, so gilt nach Simulation dieses Schritts durch die Registermaschine:

$$c(j)=m,\ c(k+1)=\ell,\ c(k+j+1)=r,$$
$$c(C_2+kp+j)=\psi_s(b).$$

Zunächst muß ein Programmstück durchlaufen werden, das die Eingabe aufbereitet und dafür sorgt, daß (*) für i=0 gilt. Da die Funktion Δ_s bereits im Suchbaum abgespeichert ist und die Tiefe des Suchbaums nicht von s bzw. t abhängt, können nun s Schritte von M mit einer konstanten Zahl von Schritten der Registermaschine simuliert werden, indem ein Programmstück folgender Wirkung durchlaufen wird:

$c(2k+3_j+q) \leftarrow c(C_2+k\cdot(c(j)+q)+j)$
für alle $j \in \{1,\ldots k\}$ und $q \in \{-1,0,1\}$

-- $(c(k+1),\ldots,c(5k+2))$ hat nun die Form $\psi_s(S)$ für ein $S=(z,r_1,\ldots,r_k,a_1,b_1,c_1,\ldots,a_k,b_k,c_k) \in K_s$ --

$c(6k+i) \leftarrow c(A(v(c(k+1),\ldots,c(5k+1),i)))$ für alle $i \in \{1,\ldots,5k+1\}$

-- $(c(6k+1),\ldots,c(11k+2))=\psi_s(\delta_s(S))$
$=(z',r',\ldots,r_k',a_1',b_1',c_1',\ldots,a_k',b_k',c_k',p_1,\ldots,p_k) \in K_s \times \{0,1,2\}^k$ --

$c(k+i) \leftarrow c(6k+i)$ für alle $i \in \{1,\ldots,k+1\}$

$c(C_2+k(c(j)+q)+j) \leftarrow c(7k+3j+q)$ für alle $j \in \{1,\ldots,k\}$ und $q \in \{-1,0,1\}$

$$c(j) \leftarrow \begin{cases} c(j)-1 & \text{falls } c(10k)+j+1 = 2 \\ c(j) & \text{falls } c(10k)+j+1 = 0 \\ c(j)+1 & \text{falls } c(10k)+j+1 = 1 \end{cases} \quad \text{für } j \in \{1,\ldots,k\}.$$

Stellt sich im Laufe der Simulation heraus, daß M nach t oder weniger Schritten hält, so gewinnt Q aus dem Inhalt der Register C_2+1, C_2+k+1,... die Inschrift von Band 1 von M am Ende der Rechnung zurück und gibt diese aus; andernfalls setzt Q $c(C_o-1)=0$.

Zeitanalyse: Für große t ist

$W=(|A|+1)^s \leq (|A|+1)^{c \log t} = 2^{c \log(|A|+1)\log t} = t^{c(\log|A|+1)}$ und

$W^d = t^{c\cdot(\log|A|+1)(4k+2)} \leq \sqrt{t}$ falls $c \leq (2\log|A|+1)(4k+2)^{-1}$. Wird also c klein genug gewählt, so kann nach Satz 3.27 der Suchbaum in $O(\sqrt{t})$ Schritten implementiert werden.

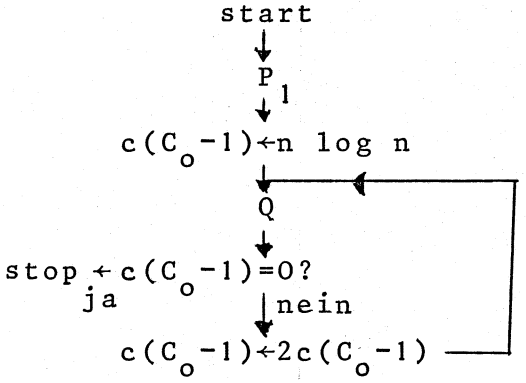
Für jedes Tupel $T\in\{0,\ldots,W-1\}^{4k+1}$ kann man in $O(s)$ Schritten testen, ob $T=\psi_s(S)$ für ein $S\in K_s$ gilt; verläuft dieser Test positiv, kann man nach Satz 3.15 aus T in $O(s)$ Schritten $\psi_s(\delta_s(S))$ berechnen. Die Komponenten von $\psi_s(\delta_s(S))$ können dann mit einer konstanten Zahl von Schritten im Suchbaum abgespeichert werden. Zum Tabellieren der gesamten Funktion Δ_s werden damit für jedes Tupel T aus $\{0,\ldots,W-1\}^{4k+1}$ gerade $O(s)=O(\log t)$ Schritte gemacht, also insgesamt nur $W^{4k+1}O(\log t)\leq W^d O(\log t)\leq O(\sqrt{t}\cdot\log t)$ Schritte.

Das Umkodieren der Eingabe, so daß (*) für i=0 gilt, ist in $O(n)$ Schritten möglich; t Schritte von M können dann mit $O(t/s)=O(t/\log t)$ Schritten der Registermaschine simuliert werden. Das Ergebnis v der Rechnung von M kann in $O(|v|)$ Schritten aus der Kodierung von v zurückgewonnen und ausgegeben werden. □

Satz 3.19 ist beinahe bewiesen. Ist t(n) eine <u>gutartige</u> Funktion in dem Sinn, daß t(n) aus n in $O(t(n)/\log t(n))$ Schritten berechnet werden kann, so genügt das folgende Verfahren:

"P_1; aus der Länge n der Eingabe t(n) berechnen; Q".

Das folgende Programm vermeidet es, t(n) im voraus zu berechnen; das Programmstück Q wird für $t=n\log n;\ 2n\log n,\ldots,2^i n\log n,\ldots$ so oft angewendet, bis die Simulation zu Ende geführt werden kann:



Für alle n mit $t(n)\leq n\log n$ gelingt die Simulation auf Anhieb in $O(n+L(n))$ Schritten. Für alle anderen n sei $d(n)=\lceil t(n)/(n\log n)\rceil$. In diesem Fall erhält man als obere Schranke für die Gesamt-Schrittzahl T:

$$T\leq O\Big(n+L(n)+\sum_{i=0}^{\lceil\log d(n)\rceil}(n+2^i n\log n/\log(2^i n\log n))\Big)$$

$$\leq O\Big(n+L(n)+n\log d(n)+\sum_{j=0}^{\lceil\log t(n)\rceil}2^j/j\Big).$$

Nun ist

$$\sum_{j=0}^{x} 2^j/j \leq \sum_{j=0}^{x-r-1} 2^j + \sum_{j=x-r}^{x} 2^j/j \text{ für jedes } r\in\{1,\ldots,x-1\}$$

$$\leq x\cdot 2^{x-1}/2^r + \frac{1}{x-r}\cdot \sum_{j=0}^{x} 2^j = O(2^x/x) \text{ für } r=2\log x.$$

Also gilt:

$T\leq O(n+L(n)+n \log d(n)+t(n)/\log t(n))$.

Wegen $\log t(n)=O(\log d(n)+\log n)$ folgt

$T\leq O(n+L(n)+[n(\log d(n))^2+n \log n \log d(n)+nd(n)\log n]/\log t(n))$

$=O(n+L(n)+t(n)/\log t(n))$.

Da in der Simulation die logarithmischen Kosten eines jeden Schritts beim Lesen der Eingabe durch $O(\log n)$ und sonst durch $O(\log t(n))$ abgeschätzt werden können, folgt auch die Schranke für die logarithmische Zeitkomplexität. □

Satz 3.12 kann ebenfalls um einen Faktor $\log t(n)$ verbessert werden [54].

Offene Fragen

i) Man interessiert sich dafür, wieweit die Sätze 3.12 bzw. 3.19 verschärft werden können, bzw. gegebenenfalls für Beweise dafür, daß dies in dem einen oder anderen Fall nicht möglich ist.

ii) Gibt es für irgendein $t(n)$ eine 0/1-wertige Funktion, die auf einer uniform $t(n)$-zeitbeschränkten Registermaschine mit Multiplikation und Division berechnet werden kann, aber nicht auf einer (uniform oder logarithmisch) $t(n)$-zeitbeschränkten Registermaschine ohne Multiplikation und Division?

iii) Man kann leicht zeigen, daß die logarithmischen Zeitkostenmaße für Registermaschinen mit und ohne Multiplikation/Division polynomiell verknüpft sind. Gilt das gleiche für uniformes Kostenmaß, wenn man sich auf die Berechnung 0/1-wertiger Funktionen beschränkt?

Diese Frage ist eng verwandt mit der in Kapitel 6 behandelten Frage "P=PTAPE?" [25,67].

4. Komplexität von Turingmaschinen

In diesem Kapitel werden hauptsächlich Turingmaschinen untersucht. Wir zeigen unter anderem, daß es beliebig komplizierte Funktionen gibt. Die wichtigsten Ergebnisse dieses Kapitels lassen sich mit den Sätzen des vorigen Kapitels auf Registermaschinen übertragen.

4.1. Akzeptoren und Komplexitätsklassen

Sei A ein endliches Alphabet. Für $a \in A$ und $n \in \mathbb{N}$ bezeichne im folgenden immer a^n die Zeichenreihe, die aus n a's besteht.

Die Berechnung der Funktion $f:\{0,1\}^* \to \{1\}^*$ mit $f(w)=1^{2^{|w|}}$ erfordert offenbar enormen Rechenaufwand, denn bei Eingabe $|w|$ sind schon $2^{|w|}$ Schritte allein dazu nötig, das Ergebnis hinzuschreiben. Die Funktion f ist also in einer trivialen Weise sehr komplex. Kann man andererseits von einer 0/1-wertigen Funktion zeigen, daß ihre Berechnung großen Rechenaufwand erfordert, so kann dies nur von einem komplizierten Rechenvorgang kommen und nicht, wie eben bei der Funktion f, von trivialen "drucktechnischen" Schwierigkeiten. Wir untersuchen deshalb im folgenden nur noch 0/1-wertige Funktionen bzw., was auf das gleiche hinausläuft, Mengen von Zeichenreihen.

Definition 4.1: A sei ein endliches Alphabet. Eine Teilmenge $L \subseteq A^*$ heißt Sprache. Ein Akzeptor ist eine Turingmaschine M mit zwei Endzuständen z_+ und z_-. Hält M angesetzt auf w im Zustand z_+ (z_-), sagen wir M akzeptiert (verwirft) w. Die Menge $L(M)=\{w \mid M \text{ angesetzt auf } w \text{ hält im Zustand } z_+\}$ heißt die von M akzeptierte Sprache.

Wir definieren einige Komplexitätsklassen, die im folgenden näher untersucht werden. Die Bezeichnungen sind aus der englischsprachigen Literatur übernommen. Maschinen, wie wir sie bisher betrachtet haben, nennt man auch deterministische Maschinen (im Gegensatz zu nichtdeterministischen Maschinen, die im nächsten Kapitel eingeführt werden).

Definition 4.2: Für $b \in \mathbb{N}$ und $s,t:\mathbb{N} \to \mathbb{N}$ mit $s(n) \geq n$ und $t(n) \geq n$ für alle n sei

$\mathrm{DTIME}_k(t(n)) = \{L \mid$ es gibt eine $O(t(n))$-zeitbeschränkte (deterministische) k-Band-Turingmaschine, die L akzeptiert$\}$

$$\mathrm{DTIME}(t(n)) = \bigcup_k \mathrm{DTIME}_k(t(n))$$

$DTAPE_k(s(n))=\{L|$ es gibt eine $O(s(n))$-bandbeschränkte (deterministische) k-Band-Turingmaschine, die L akzeptiert$\}$

$DTAPE(s(n)) = \bigcup_k DTAPE_k(s(n))$

$C_k(s(n),t(n))=\{L|$ es gibt eine simultan $O(s(n))$-bandbeschränkte und $O(t(n))$-zeitbeschränkte (deterministische) k-Band-Turingmaschine, die L akzeptiert$\}$

$C(s(n),t(n)) = \bigcup_k C_k(s(n),t(n))$.

Satz 4.3: a) $C_k(s(n),t(n)) \subseteq C_1(s(n),t^2(n))$

b) $DTIME(t(n)) \subseteq DTIME_1(t^2(n))$

c) $DTAPE(s(n)) = DTAPE_1(s(n))$

d) $DTIME(t(n)) \subseteq DTAPE(t(n))$

e) $DTAPE(s(n)) \subseteq C_1(s(n),2^{O(s(n))}) \subseteq DTIME(2^{O(s(n))})$.

Beweis: a) folgt aus Satz 1.37, b) und c) folgen aus a), d) ist trivial, da jede $O(t(n))$-zeitbeschränkte Maschine auch $O(t(n))$-bandbeschränkt ist.

Für den Beweis von e) sei $L \in DTAPE(s(n))$ und M sei eine $C \cdot s(n)$-bandbeschränkte k-Bandmaschine mit Alphabet A und Zustandsmenge Z, die L akzeptiert. Nach Teil c) kann k=1 vorausgesetzt werden. Wir beschreiben eine simultan $O(s(n))$-bandbeschränkte und $O(2^{O(s(n))})$-zeitbeschränkte 1-Bandmaschine, die ebenfalls L akzeptiert.

Lemma 4.4: Macht M angesetzt auf eine Eingabe w auf s Feldern mehr als $|Z| \cdot s \cdot |A|^s$ Schritte, so hält M angesetzt auf w nicht.

Beweis: Es gibt nur $|Z| \cdot s \cdot |A|^s$ Konfigurationen mit einer Bandinschrift der Länge s. Macht die Maschine auf s Feldern mehr Schritte, muß sich eine Konfiguration wiederholen. Wiederholt sich eine Konfiguration einmal, dann wiederholt sie sich immer wieder, d.h. die Maschine hält nicht. □

Lemma 4.4 bzw. die Schlußweise aus dem Beweis von Lemma 4.4 wird noch häufig benutzt werden. Wir beschreiben nun zunächst eine 2-Bandmaschine Q', die L=L(M) akzeptiert.

Angesetzt auf Eingabe w druckt Q' zuerst auf Band 1 auf die Felder unmittelbar rechts und links von der Eingabe ein Sonderzeichen und erzeugt auf Band 2 die Binärdarstellung von $|Z| \cdot |w| \cdot |A|^{|w|}$.

Band 2 dient als Schrittzähler. Das Band von M wird zwischen den Begrenzungszeichen auf Band 1 von Q' simuliert. Die Begrenzungszeichen werden dabei nach Bedarf nach außen verschoben. Erhöht sich in der Simulation eines Schritts der Abstand der Begrenzungszeichen und ist s der neue Abstand, so wird Band 2 auf $bin(|Z|\cdot s\cdot|A|^s)$ gesetzt, andernfalls wird die Zahl auf Band 2 um 1 erniedrigt. Q' akzeptiert, falls M akzeptiert. Q' verwirft, falls M verwirft oder falls irgendwann die Zahl auf Band 2 gleich 0 ist. Aus Lemma 4.4 folgt $L(Q')=L(M)$.

Es sei $n=|w|$. Da M $C\cdot s(n)$-bandbeschränkt ist, wird die Zahl auf Band 2 von Q' nie größer als $|Z|\cdot C\cdot s(n)\cdot|A|^{C\cdot s(n)}$. Die Simulation eines Schritts von M erfordert das Verändern dieser Zahl. Wegen $|bin(|Z|\cdot C\cdot s(n)\cdot|A|^{C\cdot s(n)})|=O(s(n))$ ist dies in $O(s(n))$ Schritten möglich. Außerdem folgt, daß Q' $O(s(n))$-bandbeschränkt ist. Insgesamt werden höchstens $\sum_{s=1}^{C\cdot s(n)} |Z|\cdot s\cdot|A|^s$ Schritte von M simuliert. Die Laufzeit von Q' ist also beschränkt durch

$$O(s(n)\cdot \sum_{s=1}^{C\cdot s(n)} s|A|^s)\leq O(s^3(n)\cdot|A|^{C\cdot s(n)})\leq O(|A|^{2C\cdot s(n)}).$$

Nach Teil a) gibt es eine 1-Bandmaschine Q, die die gleiche Sprache wie Q' akzeptiert, den gleichen Bandverbrauch hat und höchstens

$$O(|A|^{4C\cdot s(n)}) = O(2^{4C\cdot s(n)\log|A|}) = 2^{O(s(n))}$$

Schritte macht. □

Übungen

1. Lineare Beschleunigung und Bandkompression:

   a) Man zeige, daß für $k\geq 2$ und jede positive Konstante c gilt:
   $$C_k(s(n),t(n))\subseteq C_k(\max\{n+1,c\cdot s(n)\},n+c\cdot t(n)).$$
   Hinweis: Man teile nach dem Muster von § 3.3.3 die Bänder der simulierten Maschine in Blöcke mit passender Länge s ein und simuliere dann s Schritte der simulierten Maschine durch 9 Schritte der simulierenden Maschine.

   b) Man formuliere und beweise einen zu a) analogen Satz für $k=1$.

Die obigen Effekte widersprechen der Erfahrung mit realen Rechenanlagen, wo man üblicherweise nicht weiß, wie man jedes Programm um einen beliebigen konstanten Faktor schneller machen kann. Man hat sich diesen Effekt dadurch eingehandelt, daß man in einer Zelle Zeichen aus einem zwar endlichen, aber beliebig großen Alphabet speichern darf. Realistischer wäre es, nur Turingmaschinen mit Bandalphabet $\{0,1,\#,B\}$ zu betrachten.

Die Form, in der Turingmaschinen hier und in der Literatur üblicherweise definiert werden (mit beliebig großem Bandalphabet), ist einerseits historisch bedingt, andererseits lassen sich damit Maschinen etwas leichter beschreiben; man kann z.B. von Spuren reden.

Die Sätze aus Übung 1 werden in der Literatur häufig implizit verwandt. Wir benutzen diese Sätze so wenig wie möglich. Insbesondere sind alle Definitionen und Sätze in Kapitel 4 so formuliert, daß sie sich wörtlich übertragen, wenn man nur noch Maschinen mit Alphabet $\{0,1,\#,B\}$ betrachtet. Beweise, in denen große Bandalphabete vorkommen, lassen sich auch übertragen:

2. a) Für $k\geq 2$ und $s,t:\mathbb{N}\to\mathbb{N}$ mit $s(n)\geq n$ und $t(n)\geq n$ für alle n gilt: Zu jeder simultan $s(n)$-bandbeschränkten und $t(n)$-zeitbeschränkten k-Band-Turingmaschine M gibt es eine simultan $O(s(n))$-bandbeschränkte und $O(t(n))$-zeitbeschränkte k-Band-Turingmaschine Q mit Alphabet $\{0,1,\#,B\}$, so daß für alle $w\in\{0,1,\#\}^*$ gilt: M angesetzt auf w hält genau dann, wenn Q angesetzt auf w hält. Ist $f_M(w)\in\{0,1,\#\}^*$, so ist $f_M(w)=f_Q(w)$. (Beweis?)

   b) Man formuliere und beweise einen zu a) analogen Satz für 1-Band-Turingmaschinen.

4.2. Hierarchiesätze

Eine zentrale Fragestellung der Komplexitätstheorie ist, wie sehr man die Schranke $t(n)$ einer Komplexitätsklasse zu einer neuen Schranke $T(n)$ vergrößern muß, damit man mit Aufwand $T(n)$ Funktionen berechnen kann, die sich mit Aufwand $t(n)$ nicht berechnen lassen. Einige Auskunft hierüber geben die folgenden Hierarchie-

sätze. Zwei Konzepte, die schon aus Kapitel 2 bekannt sind, kommen wieder vor: Diagonalisierung und universelle Maschinen.

4.2.1. Gödelisierung von Akzeptoren

Akzeptoren gödelisiert man im Prinzip ebenso wie gewöhnliche Turingmaschinen. Da man jedoch der Gödelisierung eines Akzeptors ansehen muß, welches der akzeptierende Endzustand ist, verabreden wir, daß beim Numerieren der Zustände eines Akzeptors der akzeptierende Endzustand die kleinste Nummer von allen Endzuständen bekommt.

Jeder beliebigen Zeichenreihe u kann man nun einen Akzeptor M_u zuordnen, so daß für alle Akzeptoren M gilt: Ist u eine Gödelisierung von M, so ist $L(M)\cap\{0,1,\#\}^*=L(M_u)\cap\{0,1,\#\}^*$ (Im Falle von 1-Bandmaschinen etwa ist M_u ein Akzeptor für die leere Menge, falls u nicht Gödelisierung einer 1-Band-Turingmaschine mit genau zwei Endzuständen ist; andernfalls wird M_u genau wie in § 2.1.2 definiert, wobei der akzeptierende Endzustand der Zustand z_j ist, für den $j=\min\{k \mid z_k \in E\}$ gilt).

4.2.2. Der Bandhierarchiesatz

Für "gutartige" Funktionen s(n) kann man zeigen, daß die kleinste Änderung im asymptotischen Wachstum der Funktion zu einer neuen Bandkomplexitätsklasse führt. Eine Funktion ist gutartig, wenn sie im folgenden Sinn nicht allzu schwer zu berechnen ist:

Definition 4.5: $s:\mathbb{N}\to\mathbb{N}$ heißt bandkonstruierbar, wenn es eine $O(s(n))$-bandbeschränkte Turingmaschine gibt, die bei Eingabe w die Binärdarstellung von $s(|w|)$ erzeugt.

Beispiel 4.6: Die Funktionen $n, n\cdot\lceil\log n\rceil, n^{3/2}, n^2, 2^n, 2^{2^n}$ sind alle bandkonstruierbar.

Satz 4.7: Zu jeder totalen berechenbaren Funktion $s:\mathbb{N}\to\mathbb{N}$ gibt es eine größere bandkonstruierbare Funktion $s':\mathbb{N}\to\mathbb{N}$, d.h. so daß $s'(n)>s(n)$ für alle n gilt.

Beweis: Sei M eine r(n)-bandbeschränkte 1-Band-Turingmaschine, die s berechnet, und $s'(n)=2^{r(n)}$. Offenbar ist $s'(n)>2^{r(n)}-1\geq s(n)$. Die Funktion s' kann sogar schon von einer r(n)+2-bandbeschränkten Maschine berechnet werden, nämlich einer Maschine, die sich erst wie M verhält, dann auf alle Felder, die M benutzt hat, O druckt

und schließlich vor die hierdurch entstandene Zeichenreihe eine 1 druckt. □

Satz 4.8: Ist S(n) bandkonstruierbar und ist $\liminf(s(n)/S(n))=0$, so ist $DTAPE(S(n))\setminus DTAPE(s(n))\neq\emptyset$.

Beweis: Wir beschreiben eine O(S(n))-bandbeschränkte 1-Band-Turingmaschine Q, die alle O(s(n))-bandbeschränkten Maschinen auf folgende Weise überlistet: Es gelingt der Maschine Q, jede O(s(n))-bandbeschränkte Maschine M_u auf irgendeiner Eingabe w_u so lange zu simulieren, daß Q feststellen kann, ob $w_u\in L(M_u)$ gilt. M akzeptiert w_u genau dann, wenn $w_u\notin L(M_u)$ gilt.

Angesetzt auf Eingabe $w\in\{0,1,\#\}^*$ testet Q zunächst, ob es $u,v\in\{0,1\}^*$ mit $w=u\#v$ gibt. Es ist unwesentlich, wie sich Q verhält, wenn dieser Test negativ verläuft. Ist w von der besagten Form, so berechnet Q die Binärdarstellung von $S(|w|)$, setzt zwei Grenzzeichen im Abstand $S(|w|)$ auf das Band und erzeugt $w'=u\#u\#v$.

Q interpretiert u als Gödelisierung eines 1-Band-Akzeptors M_u mit Zustandsmenge Z_u und Bandalphabet A_u, verhält sich wie die universelle Turingmaschine aus Satz 2.17 angesetzt auf w', d.h. Q simuliert die Maschine M_u angesetzt auf Eingabe w, jedoch nur, soweit dies innerhalb der markierten Bandgrenzen möglich ist und höchstens für $|Z_u|\cdot S(|w|)\cdot|A_u|^{S(|w|)}$ Schritte. Es ist unwesentlich, wie sich Q verhält, wenn die Simulation wegen Überschreitung der Bandgrenzen abgebrochen wird. Kann die Simulation zu Ende geführt werden, akzeptiert Q die Eingabe genau dann, wenn M_u verwirft. Bricht die Rechnung wegen Überschreitung der Schrittzahl ab, akzeptiert Q.

Die Schritte von M_u werden in Binärdarstellung gezählt. Wegen $|bin(|Z_u|\cdot S(|w|)\cdot|A_u|^{S(|w|)})|=O(S(|w|)$ ist dies auf Band $O(S(|w|))$ möglich. Also ist die Maschine Q O(S(n))-bandbeschränkt.

Nach Satz 2.17 b) gibt es eine Konstante C, so daß für alle u und s eine Rechnung von M_u mit Bandverbrauch s von Q auf $C\cdot|u|\cdot s$ Feldern simuliert werden kann.

Sei $L\in DTAPE(s(n))=DTAPE_1(s(n))$, M eine $C'\cdot s(n)$-bandbeschränkte 1-Bandmaschine, die L akzeptiert und u eine Gödelisierung von M, d.h. $L(M)\cap\{0,1,\#\}^*=L(M_u)\cap\{0,1,\#\}^*$. Sei $n\in\mathbb{N}$ groß genug, so daß $n>|u|$ und $S(n)>C\cdot C'\cdot|u|\cdot s(n)$ gilt. $v\in\{0,1\}^*$ sei eine Zeichenreihe mit $|v|=n-|u|-1$ und es sei $w_u=u\#v$.

Dann ist $|w_u|=n$, und M_u angesetzt auf w_u braucht höchstens $C'\cdot s(n)$ Felder. Q braucht höchstens $C\cdot C'\cdot|u|\cdot s(n)$ Felder zum Simulieren dieser Rechnung; die Simulation dieser Rechnung bricht also nicht wegen Platzüberschreitung ab. Hält M_u angesetzt auf w_u, so geschieht dies nach Lemma 4.4 in höchstens

$$|Z_u|\cdot C'\cdot s(n)|A_u|^{C'\cdot s(n)}<|Z_u|\cdot S(n)\cdot|A_u|^{S(n)}$$

Schritten. Die Simulation bricht dann auch nicht wegen Zeitüberschreitung ab, und Q akzeptiert w_u genau dann, wenn M_u verwirft. Hält M_u angesetzt auf w_u nicht, bricht die Simulation wegen Zeitüberschreitung ab. Q akzeptiert w_u, aber M_u akzeptiert nicht. In jedem Fall folgt $w_u\in L(M)\Longleftrightarrow w_u\in L(M_u)\Longleftrightarrow w_u\notin L(Q)$, also $L(M)\neq L(Q)$. □

Korollar 4.9: Ist S(n) bandkonstruierbar und ist $\lim(s(n)/(S(n))=0$, so ist $DTAPE(s(n))\subsetneq DTAPE(S(n))$.

Beweis: $s(n)\geq S(n)$ gilt nur für endlich viele n. Also gibt es eine Konstante C, so daß $s(n)\leq C\cdot S(n)$ für alle n gilt; somit ist jede $C'\cdot s(n)$-bandbeschränkte Maschine auch $O(S(n))$-bandbeschränkt, und es gilt $DTAPE(s(n))\subseteq DTAPE(S(n))$. Nach Satz 4.8 ist die Inklusion echt. □

Aus Satz 4.7 und Satz 4.8 folgt

Korollar 2.1o: Für jede totale berechenbare Funktion $s:\mathbb{N}\to\mathbb{N}$ gibt es eine entscheidbare Sprache, die von keiner $O(s(n))$-bandbeschränkten Turingmaschine akzeptiert werden kann.

4.2.3. Zeithierarchiesätze

Auch in den Zeithierarchiesätzen benutzt man eine Voraussetzung über die Gutartigkeit der Funktionen t(n).

Definition 4.1o: $t:\mathbb{N}\to\mathbb{N}$ heißt auf einem Maschinentyp (wie z.B. 1-Bandmaschinen, k-Bandmaschinen) zeitkonstruierbar, wenn es eine $O(t(n))$-zeitbeschränkte Maschine besagten Typs gibt, die angesetzt auf Eingabe w die Binärdarstellung von $t(|w|)$ erzeugt.

Für alle Funktionen t(n) aus Beispiel 4.6 bis auf t(n)=n sieht man leicht, daß sie auf 2-Bandmaschinen zeitkonstruierbar sind. Es gilt aber auch:

Lemma 4.11: Die Funktion t(n)=n ist auf 2-Bandmaschinen zeitkonstruierbar.

Beweis: Wir beschreiben eine 2-Band-Turingmaschine M. Angesetzt

auf Eingabe w druckt M zunächst die Binärdarstellung von 1, d.h. eine 1 auf Band 2. Dann bewegt M den Kopf auf Band 1 über die Eingabe hinweg; jedesmal, wenn dabei ein neues Zeichen gelesen worden ist, wird die Binärzahl auf Band 2 um 1 erhöht und der Kopf auf Band 2 geht an das rechte Ende der Inschrift von Band 2 zurück.

Addition von 1 zu einer Binärzahl geschieht, indem von rechts her alle Einsen bis zur ersten Null durch 0 ersetzt werden, und die erste Null durch 1 ersetzt wird (z.B. 1011+1=1100). Bei jeder 2-ten Addition wird also nur eine Stelle verändert, bei jeder 4-ten Addition werden 2 Stellen verändert, und allgemein werden bei jeder 2^i-ten Addition i Stellen verändert.

Ist $|w|=n$, so erhält man als Gesamtlaufzeit

$$O\Big(\sum_{i=1}^{\lfloor \log n\rfloor+1} \lceil \frac{n}{2^i}\rceil \cdot i\Big) \leq O\Big(\sum_{i=1}^{\lfloor \log n\rfloor+1} i + n \sum_{i=1}^{\infty} \frac{i}{2^i}\Big)$$

$$= O((\log n)^2+n)=O(n). \quad \square$$

Mit der gleichen Argumentation angewandt auf die sukzessive Subtraktion von 1 von Binärzahlen erhält man das sehr nützliche

Lemma 4.12: Es gibt eine 1-Bandmaschine, die angesetzt auf bin(n) in O(n) Schritten von n bis 0 zählt, d.h. nacheinander bin(n-1), bin(n-2),...,bin(0) erzeugt.

Satz 4.13: Für alle $k\geq 2$ gilt: Ist T(n) auf k-Bandmaschinen zeitkonstruierbar und ist $\liminf(t(n)\log t(n)/T(n))=0$, so ist $DTIME_k(T(n))\setminus DTIME_k(t(n))\neq\emptyset$.

Der Beweis gleicht dem Beweis des vorigen Satzes. Wir beschreiben zunächst eine k-Bandmaschine D. Angesetzt auf Eingabe w testet D, ob w von der Form w=u#v mit $u,v\in\{0,1\}^*$ ist. Es ist unwichtig, wie sich D verhält, wenn w nicht von dieser Form ist.

Ist w von dieser Form, interpretiert D die Zeichenreihe u als Gödelisierung einer k-Bandmaschine M_u. D erzeugt w'=u#u#v und verhält sich wie die universelle Maschine aus Satz 2.19 angesetzt auf w', d.h. D simuliert M_u angesetzt auf w. D akzeptiert w genau dann, wenn M_u Eingabe w akzeptiert. Nach Satz 2.19 c) gibt es eine Konstante C, so daß für alle u,v und t gilt: Eine Rechnung der Länge $t\geq|u\ v|$ von M_u angesetzt auf u#v kann von D mit höchstens $C\cdot|u|^2\cdot t$ Schritten simuliert werden.

Wir beschreiben eine weitere k-Bandmaschine Q. Band 1 von Q hat 2 Spuren. Angesetzt auf Eingabe w erzeugt Q zunächst auf Spur 2 von Band 1 die Binärdarstellung von T(n) und dann von $\lceil T(n)/\log T(n)\rceil$. Auf Spur 1 von Band 1 und auf den Bändern 2...k simuliert Q die Maschine D angesetzt auf w. Spur 2 von Band 1 dient als Zähler und wird nach Simulation eines jeden Schritts von D um 1 erniedrigt.

Zu jedem Zeitpunkt der Rechnung gibt es ein Feld auf Band 1, das der Kopfposition auf Band 1 von D zu diesem Zeitpunkt der Rechnung entspricht. Dieses Feld nennen wir das Zentrum. Der Zähler wird anfangs so erzeugt, daß das rechte Ende des Zählers auf dem Zentrum liegt. Bewegt sich das Zentrum bei der Simulation eines Schritts von D nach rechts bzw. links, wird der gesamte Zähler ebenfalls um 1 Feld nach rechts bzw. links verschoben. Damit bleibt das rechte Ende des Zählers immer auf dem Zentrum.

Hält D, bevor der Zähler gleich Null ist, akzeptiert Q die Eingabe w genau dann, wenn D verwirft. Ist der Zähler gleich Null, hält Q ebenfalls; es ist in diesem Fall unwichtig, ob Q akzeptiert oder verwirft.

Die Simulation eines Schritts von D sowie das Erniedrigen um 1 und evtl. das Verschieben des Zählers z um ein Feld erfordern höchstens $O(|z|)=O(|\text{bin}(\lceil T(n)/\log T(n)\rceil)|=O(\log T(n))$ Schritte. Insgesamt werden höchstens $\lceil T(n)/\log T(n)\rceil$ Schritte von D simuliert. Also ist Q $O(T(n))$-zeitbeschränkt.

Sei $L\in DTIME_k(t(n))$ und M eine $C'\cdot t(n)$-zeitbeschränkte k-Bandmaschine, die L akzeptiert. Sei u eine Gödelisierung von M, also $L(M)\cap\{0,1,\#\}^*=L(M_u)\cap\{0,1,\#\}^*$. Sei $n\in\mathbb{N}$ groß genug, so daß $n>|u|$, $T(n)>2C'\cdot C\cdot|u|^2 t(n)\log t(n)$ und $t(n)/(2\log t(n))>C'\cdot C\cdot|u|^2$ gilt. Es folgt $T(n)/\log T(n)>C'\cdot C\cdot|u|^2\cdot t(n)$, denn entweder ist $T(n)<t^2(n)$, also $\log T(n)<2\log t(n)$, oder $T(n)\geq t^2(n)$, also $T(n)/\log T(n)\geq t^2(n)/(2\log t(n))>C'\cdot C\cdot|\bar{u}|^2 t(n)$.

$v\in\{0,1\}^*$ sei eine Zeichenreihe mit $|v|=n-|u|-1$, und es sei $w_u=u\#v$. Offenbar ist $|w_u|=n$, und M_u angesetzt auf w_u macht höchstens $C'\cdot t(n)$ Schritte. D simuliert $C'\cdot t(n)$ Schritte von M_u in $C'\cdot C\cdot|u|^2\cdot t(n)<T(n)/\log T(n)$ Schritten. Also bricht die Simulation von D durch Q bei Eingabe w_u nicht durch Zeitüberschreitung ab. Es folgt $w_u\in L(m)\Leftrightarrow w_u\in L(M_u)\Leftrightarrow w_u\in L(D)\Leftrightarrow w_u\notin L(Q)$, also $L(M)\neq L(Q)$. □

Korollar 4.14: Ist $k \geq 2$, ist $T(n)$ auf k-Band-Turingmaschinen zeitkonstruierbar und ist $\lim t(n)\log t(n)/T(n)=0$, so ist $DTIME_k(t(n)) \subsetneq_+ DTIME_k(T(n))$.

Zusammen mit Satz 1.37 erhält man einen Hierarchiesatz für die Klassen $DTIME(t(n))$:

Korollar 4.15: Ist $T:\mathbb{N}\to\mathbb{N}$ für irgendein k auf k-Band-Turingmaschinen zeitkonstruierbar und ist $\lim t^2(n)\log t(n)/T(n)=0$, so ist $DTIME(t(n)) \subsetneq_+ DTIME(T(n))$.

Beweis: $DTIME(t(n)) \subseteq DTIME_1(t^2(n)) \subseteq DTIME_k(t^2(n)) \subsetneq_+ DTIME_k(T(n)) \subseteq DTIME(T(n))$. □

Aus Satz 4.13 kann man auch Hierarchiesätze für Registermaschinen gewinnen:

Sei $R^u_{t(n)} = \{L \mid$ es gibt eine uniform $O(t(n))$-zeitbeschränkte Registermaschine, die L akzeptiert*)$\}$.

Korollar 4.16: Ist $T(n)$ auf 2-Band-Turingmaschinen zeitkonstruierbar und ist $\lim t^3(n)\log t(n)/T(n)=0$, so ist $R^u_{t(n)} \subsetneq_+ R^u_{T(n)}$.

Beweis: Nach den Sätzen 3.12 und 3.15 gilt

$$R^u_{t(n)} \subseteq DTIME_2(t^3(n)) \subsetneq_+ DTIME_2(T(n)) \subseteq R^u_{T(n)}.$$

Ein ähnliches Korollar ließe sich für logarithmisch zeitbeschränkte Registermaschinen gewinnen. Man kann jedoch durch direktes Übertragen des Beweises von Satz 4.13 auf Registermaschinen erheblich schärfere Sätze gewinnen.

Sätze wie Satz 4.13 liefern auch Schranken dafür, wie weit sich die Simulationen aus Kapitel 3 höchstens verbessern lassen. Beispielsweise gilt

Korollar 4.17: Es gibt $t(n)$-zeitbeschränkte 2-Band-Akzeptoren, die von keiner uniform $O(n+t(n)^{1/3}/(\log t(n)^{2/3})$-zeitbeschränkten Registermaschine simuliert werden können.

Beweis: Andernfalls würde zusammen mit Satz 3.12 b) folgen:

$DTIME_2(n^4) \subseteq R^u_{n^{4/3}/(\log n)^{2/3}} \subseteq DTIME_2(n^4/(\log n)^2)$. □

*) "Registermaschine M akzeptiert L" kann man z.B. definieren durch "M berechnet die charakteristische Funktion von L".

Satz 4.13 kann noch verschärft werden. Sei

$$\log^* n = \min\{k \mid 2^{\cdot^{\cdot^{2}}} \text{ } k\text{-mal} > n\};$$

$\log^i x$ wird induktiv definiert durch:

$\log^0 x = x$, $\log^{i+1} x = \log(\log^i x)$,

also z.B. $\log^2 x = \log(\log x)$.

Satz 4.18: Ist $k \geq 2$, ist $t(n)$ auf k-Bandmaschinen zeitkonstruierbar und ist $\liminf(t(n)\log^* t(n)/T(n)) = 0$, so ist

$$DTIME_k(T(n)) \setminus DTIME_k(t(n)) \neq 0.$$

Ist

$$\lim(t(n)\log^* t(n)/T(n)) = 0,$$

so ist

$$DTIME_k(t(n)) \subsetneq DTIME_k(T(n)).$$

Der Beweis ist eine raffiniertere Variante des Beweises von Satz 4.13. Modifiziert wird nur die Maschine Q. Q simuliert D nicht mehr für $\lceil T(n)/\log T(n) \rceil$, sondern für $\lceil T(n)/\log^* T(n) \rceil$ Schritte. Nun kann aber nicht mehr bei der Simulation jedes Schritts von D der gesamte Zähler z verschoben werden, weil das zuviel Zeit kosten würde. Das ist aber auch nicht nötig, weil bei den meisten Subtraktionen nur wenige Stellen des Zählers verändert werden.

Der Zähler z wird segmentiert in

$$z = z_1 z_2 \ldots z_s,$$

mit

$$|z_i| = \lceil \log^i T(n) \rceil - \lceil \log^{i+1} T(n) \rceil \text{ für } i \neq s,$$

$$|z_s| = \lceil \log^s T(n) \rceil$$

und

$$s = \min\{i \mid \log^i T(n) \leq 25\}.$$

Es folgt $s \leq \log^* T(n)$.

Zu Beginn der Simulation wird $z = bin\lceil T(n)/\log^* T(n) \rceil$ gesetzt, und die Zählersegmente werden so verteilt, daß für alle i das rechte Ende von z_i gerade $3\lceil \log^i T(n) \rceil$ Felder links vom Zentrum steht. Zusätzlich wird das linke Ende von z_1 mit einem Sonderzeichen markiert. Dieses Zeichen markiert das linke Ende aller Zähler und erleichtert später das Testen des Gesamtzählers auf Null. Mit Hilfe einer extra Spur auf Band 2, die als Hilfsspeicherplatz benutzt wird, kann das anfängliche Verteilen der Zählersegmente in Zeit

$O(\log T(n))$ erledigt werden.

Das Erniedrigen des gesamten Zählers um 1 nach der Simulation eines Schritts von D geschieht wie folgt:

Es sei $i=\max\{j \mid z_j \neq 0\ldots0\}$. Dann werden die Zähler $z_{i+1},\ldots,z_s$ auf $1\ldots1$ gesetzt und z_i wird um 1 erniedrigt. Wird hierdurch $z_i=0\ldots0$ und ist das linke Ende von z_i mit dem Sonderzeichen markiert, so wird das linke Ende von z_{i+1} mit dem Sonderzeichen markiert, falls $i\neq s$ gilt (andernfalls bricht die Rechnung ab). Außerdem werden die Zähler $z_i,\ldots,z_s$ so verschoben, daß das rechte Ende von z_i gerade $3\lceil \log^i T(n)\rceil$ Felder links vom Zentrum steht. Dabei wird zuerst z_s bewegt, dann $z_{s-1}\ldots$ usw. bis z_i.

All dies werde nun eine <u>i-Operation</u> genannt. Die Simulation eines beliebigen Schritts von D mit der zugehörigen Korrektur der Zählersegmente nennen wir einen <u>Großschritt</u>. Wir beobachten, daß eine i-Operation alle

$$2^{|z_{i+1}\ldots z_s|} \leq 2^{1+\log^{i+1} T(n)} = 2\ \log^i T(n)$$

Großschritte vorkommt.

Wir müssen zeigen, daß sich die Zähler nie überlappen. Hierfür zeigen wir für beliebiges i durch Induktion über t, daß sich die Zähler z_i und z_{i+1} bis zum t-ten Großschritt nicht überlappen. Für t=0 ist nichts zu zeigen.

<u>Induktionsschritt</u>

t' sei der letzte Großschritt vor t, in dem die Position von z_i korrigiert wurde, falls z_i schon bewegt wurde, t'=0 sonst. Nach dem oben Gesagten ist $t-t' \leq 2\ \log^i T(n)$. $d(i,r)$ bezeichne den Abstand des rechten Endes von z_i vom Zentrum nach Großschritt r. Nach Konstruktion ist $d(i,t')=3\lceil \log^i T(n)\rceil$. In den Großschritten $t'+1,\ldots,t$ hat sich das Zentrum höchstens $t-t' \leq 2\ \log^i T(n)$ Felder auf z_i zubewegt. Nachdem die Position von z_{i+1} in Großschritt t korrigiert wurde, ist $d(i+1,t)=3\lceil \log^{i+1} T(n)\rceil$. Der Abstand des linken Endes von z_{i+1} zum rechten Ende von z_i ist damit mindestens

$$\begin{aligned} & d(i,t')-(t-t')-d(i+1,t)-|z_{i+1}| \\ & \geq 3\lceil \log^i T(n)\rceil - 2\ \log^i T(n) - 3\lceil \log^{i+1} T(n)\rceil - \lceil \log^{i+1} T(n)\rceil \\ & > \log^i T(n) - 4\ \log^{i+1} T(n) - 4. \end{aligned}$$

Nach Konstruktion von s ist $\log^i T(n) \geq 25$. Die Funktion $f(x)=x-4\cdot\log x-4$ wächst im Intervall $[25,\infty]$ strikt monoton, nimmt also ihr Minimum in diesem Intervall für $x=25$ an. Wegen $\log 25<5$ folgt $f(25)>1$.

Es bleibt zu zeigen, daß Q $O(T(n))$-zeitbeschränkt ist.

Wegen $|z_i| \leq \lceil \log^i T(n) \rceil$ kann eine i-Operation mit Hilfe der Extraspur auf Band 2, die zum Hin- und Herkopieren der Zählersegmente z_i benutzt wird, in $O(\log^i T(n))$ Schritten ausgeführt werden. Eine i-Operation findet alle $2^{\lceil \log^{i+1} T(n) \rceil} \geq \log^i T(n)$ Großschritte statt.

Da insgesamt $\lceil T(n)/\log^* T(n) \rceil$ Großschritte ausgeführt werden, erhält man als Gesamtlaufzeit

$$\sum_{i=1}^{s} \frac{T(n)}{\log^* T(n)\ \log^i T(n)}\, O(\log^i T(n)) = s\cdot O\left(\frac{T(n)}{\log^* T(n)}\right)$$

$$= O(T(n)), \text{ da } s \leq \log^* T(n).$$

Der Rest des Beweises folgt genau dem Muster des entsprechenden Teils des Beweises von Satz 4.13. □

Korollar 4.15 kann offenbar verschärft werden, wenn es gelingt, eine schnellere Simulation von beliebig vielen Bändern durch eine feste Anzahl von Bändern zu finden als Satz 1.37. Das ist in der Tat möglich.

Satz 4.19 (Hennie, Stearns): $C(s(n),t(n)) \subseteq C_2(s(n),t(n)\log s(n))$.

Korollar 4.2o: $DTIME(t(n)) \subseteq DTIME_2(t(n)\log t(n))$.

Beweis von Satz 4.19: M sei eine simultan $C\cdot s(n)$-band- und $C\cdot t(n)$-zeitbeschränkte k-Bandmaschine. Wir beschreiben eine 2-Bandmaschine Q, die M simuliert. Bis auf die Repräsentation der Bandinschriften von M und die Simulation der Kopfbewegungen verläuft die Simulation ähnlich wie im Beweis von Satz 3.17. Jedes Band von M wird auf 2 Spuren von Band 1 von Q simuliert. Band 1 von Q ist in Blöcke $B_i, i \in \mathbb{Z}$, eingeteilt, wobei für $|B_i|$, die Länge von Block i, gilt: $|B_i| = 2^{|i|-1}$ falls $i \neq 0$ und $|B_0| = 0$.

Beispiel 4.21:

				3	4	6					12	13	14	15
				1	2	5	7				8	9	10	11
B_{-3}				B_{-2}		B_{-1}	B_o	B_1		B_2			B_3	

Sowie mehr und mehr Zellen von Band 1 benutzt werden, werden die Blockgrenzen auf einer zusätzlichen Spur markiert. Eine Kopfbewegung der Maschine M wird dadurch simuliert, daß die Inschrift der entsprechenden Spuren von Band 1 von Q verschoben wird. Das geschieht so, daß immer das Zeichen eines jeden Bandes, auf dem ein Kopf von M steht, in Block B_o ist. Die Inschrift der Bänder von M muß so gespeichert und verschoben werden, daß zur Simulation der meisten Kopfbewegungen von M nur kleine Teile der Inschrift verschoben werden müssen. Wir betrachten vorläufig nur ein Band von M und die zugehörigen beiden Spuren von Band 1 von Q.

Für $i\in\mathbb{Z}$ bezeichne B_i^1 die untere und B_i^2 die obere Spur von Block B_i. Die Inschrift des Bandes von M wird so in den Blöcken B_i gespeichert, daß nach Simulation jeder Kopfbewegung von M für alle $i,k\in\mathbb{Z}$, $j,\ell\in\{1,2\}$ gilt:

(*) i) B_i^j ist entweder <u>leer</u> oder B_i^j enthält $|B_i|$ konsekutive Zeichen der Bandinschrift von M. In letzterem Fall sagen wir: B_i^j ist <u>voll</u>. B_0^2 ist immer leer.

ii) B_i^1 und B_i^2 sind beide voll (B_i ist voll) oder B_i^1 ist voll und B_i^2 ist leer (B_i ist halb voll) oder B_i^1 und B_i^2 sind beide leer (B_i ist leer).

iii) Sind B_i^j und B_k^ℓ beide voll, ist $i<k$ oder ist $i=k$ und $j<\ell$, so ist der Inhalt von B_i^j ein Teil der Bandinschrift von M links vom Inhalt von B_k^ℓ.

Die Bedingungen (*) kann man zu Beginn der Simulation erfüllen, indem man die Bandinschrift von M auf die untere Spur kopiert und die gesamte obere Spur frei läßt. Zur Simulation einer Kopfbewegung von M nach rechts (Verschieben der Inschrift der entsprechenden Spuren von Band 1 von Q nach links) wird der Reihe nach folgendes getan:

1) Sei $r=\min\{j \mid j\geq 1,\ B_j \text{ ist nicht leer}\}$

$\ell=\max\{j \mid j\leq -1, B_j \text{ ist nicht voll}\}$

2)
$$B^1_\ell \leftarrow \begin{cases} B^1_{\ell+1}\ B^2_{\ell+1} & \text{falls } B^1_\ell \text{ leer und } \ell<-1 \\ B^1_o & \text{falls } B^1_\ell \text{ leer und } \ell=-1 \end{cases}$$

$$B^2_\ell \leftarrow \begin{cases} B^1_{\ell+1}\ B^2_{\ell+1} & \text{falls } B^1_\ell \text{ voll und } \ell<-1 \\ B^1_o & \text{falls } B^1_\ell \text{ voll und } \ell=-1 \end{cases}$$

3) $B^1_i \leftarrow B^1_{i+1} B^2_{i+1}$ für $i=\ell+1,\ldots,-2$ in dieser Reihenfolge

4) $B^1_{-1} \leftarrow B^1_o$ falls $\ell<-1$

5) $B^1_o \ldots B^1_{r-1} \leftarrow B^1_r$

6) $B^1_r \leftarrow B^2_r$ falls B^2_r nicht leer.

Beispiel 4.22: Aus der Inschrift aus Beispiel 4.21 entsteht hierdurch

1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15

B_{-3} B_{-2} B_{-1} B_o B_1 B_2 B_3

Offenbar bleibt (*) nach der Simulation einer Kopfbewegung nach rechts erfüllt, falls (*) vorher erfüllt war. Die Simulation von Kopfbewegungen nach links wird entsprechend gehandhabt. Wir nennen die Simulation einer Kopfbewegung von M einen Großschritt. Eine i-Operation ist das Besuchen der Blöcke $B_o \ldots B_i$ in einem Großschritt. Jeder Großschritt besteht aus einer i-Operation mit $i>0$ und einer i'-Operation mit $i'<0$. Mit Hilfe von Band 2 von Q, das zum Zwischenspeichern von Inschriften der B^j_i benutzt wird, kann eine i-Operation in Zeit $O(2^{|i|})$ ausgeführt werden.

Wir zeigen, daß i-Operationen mit großem $|i|$ selten sind

Lemma 4.23: Für alle $i\geq 0$ gilt: Eine i-Operation kommt frühestens im 2^{i-1}-ten Großschritt und höchstens alle 2^{i-1} Großschritte vor (entsprechendes gilt für $i'<0$).

Beweis: Im Großschritt t komme eine i-Operation vor. Es sei t'

der letzte Großschritt vor t, in dem eine i-Operation stattgefunden hat, falls schon eine i-Operation stattgefunden hat; t'=0 sonst. Nach Durchführung von Großschritt t' sind die Blöcke $B_1 \ldots B_{i-1}$ alle halb voll. Vor Durchführung von Großschritt t sind diese Blöcke alle voll oder alle leer. Also müssen in den Schritten $t'+1,\ldots,t-1$ mindestens $\sum_{j=1}^{i-1} |B_j| = 2^{i-1}-1$ Zeichen über B_o nach links oder rechts geschoben worden sein. In einem Großschritt wird nur ein Zeichen über B_o geschoben. Es folgt $t-t'-1 \geq 2^{i-1}-1$. □

Die Maschine M ist $C \cdot s(n)$-bandbeschränkt und $C \cdot t(n)$-zeitbeschränkt. Es folgt, daß zur Simulation höchstens $C \cdot t(n)$ Großschritte ausgeführt werden und daß nur i-Operationen mit $|B_i| \leq C \cdot s(n)$ bzw. $|i| \leq 1+\log s(n)+ \log C$ ausgeführt werden. Damit ist Q $O(s(n))$-bandbeschränkt, und man erhält als obere Schranke für die Laufzeit von Q

$$2 \sum_{i=1}^{1+\log C+\log s(n)} \frac{C \cdot t(n)}{2^{i-1}} O(2^i) = O(t(n) \log s(n)).$$

Werden in einem Großschritt statt eines Bandes k Bänder verschoben, ändert sich die Laufzeit höchstens um den Faktor k. □

Wir hatten im Beweis von Korollar 4.15 nicht nur einen Faktor t(n) durch die Simulation von vielen Bändern durch ein Band verloren, sondern auch einen Faktor log t(n) durch das Mitführen des Zählers auf einer extra Spur. Wenn man den Zähler jedoch auf einem Extra-Band betreibt, verliert man überhaupt keine Zeit beim Zählen:

Lemma 4.24: Ist $T: \mathbb{N} \to \mathbb{N}$ zeitkonstruierbar auf k+1-Band-Turingmaschinen und ist $\lim t(n)/T(n)=0$, so ist

$$DTIME_k(t(n)) \subsetneq DTIME_{k+1}(T(n)).$$

Beweis: Ebenfalls eine Variante des Beweises von Satz 4.13, bei der nur die Maschine Q modifiziert wird. Statt für $\lceil T(n)/\log T(n) \rceil$ Schritte wird D für T(n) Schritte simuliert. Statt auf einer Extra-Spur wird der Zähler auf einem Extra-Band angelegt. Das Herabzählen des Zählers in Binärdarstellung von T(n) bis 0 ist nach Lemma 4.12 in $O(T(n))$ Schritten möglich. □

Satz 4.25: Ist $T: \mathbb{N} \to \mathbb{N}$ für irgendein k zeitkonstruierbar auf k-Band-Turingmaschinen und ist $\lim t(n)\log t(n)/T(n)=0$, so ist $DTIME(t(n)) \subsetneq DTIME(T(n))$.

Beweis: OBdA kann $k \geq 3$ angenommen werden. Dann folgt

$DTIME(t(n)) \subseteq DTIME_2(t(n)\log\ t(n)) \subseteq DTIME_{k-1}(t(n)\log\ t(n))$

$\subsetneq DTIME_k(T(n)) \subseteq DTIME(T(n))$. □

In allen Hierarchiesätzen, die wir bewiesen haben, steckt eine Voraussetzung über die Gutartigkeit der Funktion T(n). Wir werden in Kapitel 7 sehen, daß ohne eine solche Voraussetzung keiner dieser Hierarchiesätze gilt.

4.2.4. Padding

Padding (= engl. ausstopfen) ist ein Trick, mit dem man für nicht zu schnell wachsende Funktionen t(n) Hierarchiesätze verschärfen kann.

Satz 4.26: $t:\mathbb{N}\to\mathbb{N}$ sei zeitkonstruierbar auf Turingmaschinen. Für $p:\mathbb{N}\to\mathbb{N}$ gebe es eine O(n)-zeitbeschränkte Turingmaschine, die bei Eingabe w die Binärdarstellung von $p(|w|)$ berechnet.

Für $\varepsilon>0$ und alle i gelte

$$\lim \frac{t(n)\cdot(\log\ t(n))^{\varepsilon(i+1)}}{t(p(n))\cdot(\log\ t(p(n)))^{\varepsilon i}} \quad \text{existiert}$$

und ist von Null verschieden.

Dann ist $DTIME(t(n)) \subsetneq DTIME(t(n)(\log\ t(n))^{\varepsilon})$.

Korollar 4.27: a) $DTIME(n^k) \subsetneq DTIME(n^k(\log\ n)^{\varepsilon})$

b) $DTIME(2^n) \subsetneq DTIME(n^{\varepsilon}2^n)$.

Beweis des Korollars: a) $p(n)=n(\log\ n)^{\varepsilon}$

b) $p(n)=n+\varepsilon\log\ n$.

Der Beweis von Satz 4.26 wird durch Widerspruch geführt. Wir leiten aus der Annahme

$$DTIME(t(n))=DTIME(t(n)\cdot(\log\ t(n))^{\varepsilon})$$

durch Induktion über i

$$DTIME(t(n)\cdot(\log\ t(n))^{\varepsilon i}) = DTIME(t(n)\cdot(\log\ t(n))^{\varepsilon(i+1)})$$

für alle i her. Es folgt

$$DTIME(t(n))=DTIME(t(n)(\log\ t(n))^{\varepsilon\cdot(\lceil\frac{1}{\varepsilon}\rceil+1)})$$

im Widerspruch zu Satz 4.25.

Der Induktionsanfang (i=0) ist gerade die Annahme. Für den Schluß

von i auf i+1 sei nun $L \in DTIME(t(n)(\log t(n))^{\varepsilon(i+1)})$, $L \subseteq A^*$ und M sei ein $O(t(n)\cdot(\log t(n))^{\varepsilon(i+1)})$-zeitbeschränkter Akzeptor für L. $x \notin A$ sei ein neues (Padding-) Symbol.

Es sei $P=\{wx^{p(|w|)-|w|} \mid w \in A^*\}$

und $L'=\{wx^y \mid w \in L, wx^y \in P\}$.

Lemma 4.28: $L' \in DTIME(t(n)\cdot(\log t(n))^{\varepsilon i})$.

Beweis: Q sei eine Maschine, die angesetzt auf Eingabe w' erst testet, ob $w'=wx^y \in P$ gilt. Dies kann nach Voraussetzung über p(n) (und weil man nach dem Beweis von Lemma 4.12 binär von 0 bis t in O(t) Schritten zählen kann) in Zeit $O(|w'|)$ getestet werden. Ist $w' \notin P$, verwirft Q sofort, andernfalls löscht Q x^y und verhält sich auf w wie M. Offenbar ist $L(Q)=L'$.

Sei $|w'|=n$, $w'=wx^{p(|w|)-w}$ und $m=|w|$, also $n=p(m)$. Dann macht Q angesetzt auf w' höchstens

$$\begin{aligned} &O(n)+O(t(m)(\log t(m))^{\varepsilon(i+1)}) \\ &\quad = O(n)+O(t(p(m))(\log t(p(m)))^{\varepsilon i}) \\ &\quad = O(t(n)(\log t(n))^{\varepsilon i}) \end{aligned}$$

Schritte. □

Aus Lemma 4.28 und der Induktionsvoraussetzung folgt

$$L' \in DTIME(t(n)(\log t(n))^{\varepsilon(i-1)}).$$

Sei Q' ein $O(t(n)(\log t(n))^{\varepsilon(i-1)})$-zeitbeschränkter Akzeptor für L'. M' sei eine Maschine, die angesetzt auf Eingabe w erst $wx^{p(|w|)-|w|}$ erzeugt und sich dann wie Q' verhält. Offenbar ist $L(M')=L$.

Sei $|w|=n$. Angesetzt auf Eingabe w macht M' höchstens

$$O(p(n))+O(t(p(n))\cdot(\log t(p(n))^{\varepsilon(i-1)})=O(t(n)(\log t(n))^{\varepsilon i})$$

Schritte. □

Satz 4.18 kann ebenfalls durch Padding verschärft werden.

Offene Probleme

a) Kann man die Zeithierarchiesätze dieses Abschnitts verschärfen?

b) Kann man beliebig viele Bänder schneller als mit Zeitverlust log t(n) mit einer festen Anzahl von Bändern simulieren? Ist für zwei Bänder die hier angegebene Simulation die best-

mögliche?

Übungen

1. Man zeige:

   a) Die folgenden Funktionen t(n) sind auf 2-Band-Turingmaschinen zeitkonstruierbar:
   $$t(n)=n^2, \lceil n^{3/2} \rceil, \lceil n \log^* n \rceil.$$

   b) Die Funktion $t(n)=n \log n$ ist auf 1-Band-Turingmaschinen zeitkonstruierbar.

   c) Zu jeder berechenbaren Funktion $t:\mathbb{N}\to\mathbb{N}$ gibt es eine größere Funktion $T:\mathbb{N}\to\mathbb{N}$, die auf 1-Band-Turingmaschinen zeitkonstruierbar ist.

2. Man formuliere und beweise einen zu Satz 4.13 analogen Satz für 1-Band-Turingmaschinen.

3. Zwei Mengen A,B heißen unvergleichbar, wenn weder $A\subseteq B$ noch $B\subseteq A$ gilt.
   Man zeige: Es gibt Funktionen $t_1, t_2:\mathbb{N}\to\mathbb{N}$, so daß $DTIME_k(t_1(n))$ und $DTIME_k(t_2(n))$ unvergleichbar sind.

4. Man führe die Beweise von Satz 4.18 und Lemma 4.25 zu Ende.

5. a) Man formuliere und beweise ein zu Korollar 4.16 analoges Ergebnis für Registermaschinen mit logarithmischem Kostenmaß.
   b) Man beweise direkt möglichst scharfe Zeithierarchiesätze für Registermaschinen (für beide Zeitmaße).
   c) Gibt es für die Zeitkomplexität von Registermaschinen einen zu Übung 1 aus Abschnitt 4.1 analogen "Beschleunigungssatz"? Beweis?
   d) Wie weit kann man die Simulationen aus Kapitel 3 höchstens verschärfen?

6. a) Was ist bei der $t(n)\log s(n)$-Simulation von vielen Bändern durch zwei Bänder zur Simulation einer Kopfbewegung nach links zu tun?

   Eine Turingmaschine M heißt "oblivious", wenn die Kopfbewegungen von M angesetzt auf Eingabe w nur von $|w|$ abhängen.

b) Jede simultan O(s(n))-bandbeschränkte und O(t(n))-zeitbeschränkte k-Bandmaschine kann durch eine O(s(n)·t(n))-zeitbeschränkte "oblivious" 1-Bandmaschine simuliert werden.

c) Jede simultan O(s(n))-bandbeschränkte und O(t(n))-zeitbeschränkte k-Bandmaschine kann durch eine O(t(n)·log(s(n))-zeitbeschränkte "oblivious" 2-Bandmaschine simuliert werden.

7. Man zeige: a) $DTIME(2^{n^2}) \subsetneq DTIME(n2^{n^2})$

b) $DTIME_2(n) \subsetneq DTIME_2(n\sqrt{\log^* n})$

c) gilt
$$DTIME(2^{2^n}) \subsetneq DTIME(2^n 2^{2^n})?$$

8. Z sei die elementare Zahlentheorie (vgl. Kapitel 2.2).

$DTIME_k^Z(t(n)) := \{L \mid L=L(M)$ für eine k-Bandmaschine M, und es gibt in Z einen Beweis dafür, daß M O(t(n))-zeitbeschränkt ist$\}$.

Man zeige: Ist $\lim t(n)/T(n)=0$, so ist
$$DTIME_k^Z(t(n)) \subsetneq DTIME_k(T(n)).$$

4.3. Weitere untere Schranken

Wir haben bisher durch Diagonalisierung nachweislich schwer erkennbare Sprachen konstruiert. In diesem Abschnitt leiten wir mit rein kombinatorischen Methoden (d.h. ohne explizite Diagonalisierung) untere Schranken für die Komplexität einiger Sprachen her.

4.3.1. Off-line-Turingmaschinen

Eine Eingabe der Länge n belegt bereits n Felder; deshalb konnten wir bisher nur s(n)-bandbeschränkte Maschinen mit $s(n) \geq n$ betrachten. Eine leichte Modifikation des Turingmaschinenmodells erlaubt es jedoch, auch Bandkomplexitäten, die kleiner als n sind, zu studieren.

Definition 4.29: Eine k-Band-Off-line Turingmaschine ist eine k+1-Band-Turingmaschine mit einem ausgezeichneten Eingabeband, auf dem die Maschine nur lesen darf (im Gegensatz zum Eingabeband von Registermaschinen darf die Off-line Maschine jedoch jede Zelle des Eingabebandes beliebig oft besuchen). Auf den übrigen

Bändern, die auch Arbeitsbänder genannt werden, darf die Maschine sowohl lesen als auch schreiben. Eine Off-line-Maschine M heißt s(n)-bandbeschränkt, wenn M für jede Eingabe der Länge n höchstens s(n) Felder auf den Arbeitsbändern benutzt.

Beispiel 4.3o: Wir beschreiben informal eine (1+log n)-bandbeschränkte 1-Band-Off-line-Maschine M, die die Sprache $\{a^m b^m | m \in M\}$ akzeptiert. Ist das erste Zeichen, das die Maschine auf dem Eingabeband liest, ein b, so verwirft M sofort, andernfalls druckt M die Binärdarstellung von 1, d.h. eine 1 auf das Arbeitsband. Dann bewegt die Maschine den Kopf auf dem Eingabeband von links nach rechts über die Eingabe, bis sie das erste b findet. Hierbei addiert die Maschine nach jeder Bewegung des Kopfs auf dem Eingabeband 1 zu der Binärzahl auf dem Arbeitsband. Während die Maschine b's liest, wird subtrahiert statt addiert. Wir ein von b verschiedenes Zeichen gelesen, während auf dem Arbeitsband noch eine Zahl > 0 steht, ist die Eingabe nicht von der Form $a^i b^i$. Die Eingabe ist genau dann von der gewünschten Form, wenn der eben erwähnte Fall nicht eintritt und wenn nach dem Lesen des letzten Zeichens der Eingabe und der zugehörigen Subtraktion 0 auf dem Arbeitsband steht. Hat die Eingabe die Länge n, so wird auf dem Arbeitsband höchstens bis n gezählt. Die Länge der Binärdarstellung von n ist aber kleiner oder gleich 1+log n.

Für Funktionen $s: \mathbb{N} \to \mathbb{N}$, für die nicht $s(n) \geq n$ für alle n gilt, werden die Klassen $DTAPE_k(s(n))$, $DTAPE(s(n))$, $C_k(s(n),t(n))$ und $C(s(n),t(n))$ definiert, indem man in Definition 4.2 Turingmaschinen durch Off-line-Turingmaschinen ersetzt.

Durch Simulation von k Arbeitsbändern auf k+1 Spuren eines einzigen Arbeitsbands erhält man Satz 1.37 wörtlich für Off-line-Maschinen und damit Satz 4.3 a) und c) für beliebige Funktionen s(n). Aus Lemma 4.4 wird

Lemma 4.3o: Ist M eine 1-Band-Off-line-Turingmaschine mit Alphabet A und Zustandsmenge Z und macht M angesetzt auf Eingabe w mehr als $|z| \cdot |w| \cdot s \cdot |A|^s$ Schritte, ohne mehr als s Felder des Arbeitsbands zu benutzen, so hält M angesetzt auf w nicht.

Schließlich gilt auch

Satz 4.31: a) Für $s(n) = \Omega(\log n)$ ist $DTAPE(s(n)) \subseteq C_1(s(n), 2^{O(s(n))})$,

b) Ist $S: \mathbb{N} \to \mathbb{N}$ bandkonstruierbar, $S(n) = \Omega(\log n)$ und

$\lim\inf(s(n)/S(n))=0$, so ist
$DTAPE(S(n))\setminus DTAPE(s(n))\neq\emptyset$.

Den Beweis erhält man aus den Beweisen von Satz 4.3 e) und Satz 4.8, wobei man Lemma 4.3o anstelle von Lemma 4.4 benutzt. □

4.3.2. Eine Platzschranke

Satz 4.32: Für $L=\{a^m b^m \mid m\in\mathbb{N}\}$ gilt
a) $L\in DTAPE(\log n)$.
b) Ist $s(n)=o(\log n)$, so ist $L\notin DTAPE(s(n))$.

Beweis: Teil a) wurde in Beispiel 4.29 gezeigt. Idee des Beweises der unteren Schranke b) ist, daß eine Maschine bis m zählen können muß, um $a^m b^m$ zu erkennen. Dafür ist ungefähr Platz log m nötig.

Sei also M eine $C\cdot s(n)$-bandbeschränkte k-Band-Off-line-Turingmaschine mit Zustandsmenge Z und Alphabet A, die L akzeptiert. Ist während einer Rechnung z der Zustand von M, v die Inschrift des Arbeitsbandes und r die Kopfposition auf dem Arbeitsband von M, so nennen wir (z,v,r) einen verallgemeinerten Zustand oder kurz V-Zustand von M. Angesetzt auf $a^m b^m$ benutzt M höchstens $C\cdot s(2m)$ Felder des Arbeitsbands, kann also höchstens $C\cdot s(2m)\cdot|Z|\cdot|A|^{C\cdot s(2m)}$ verschiedene V-Zustände annehmen. Ist $s(n)=o(\log n)$, so gibt es eine gerade Zahl $n\in\mathbb{N}$ mit

$$\log|A|\cdot C\cdot s(n)+\log(C\cdot s(n))+\log|Z|<\log n/2$$

bzw.

$$C\cdot s(n)\cdot|Z|\cdot|A|^{C\cdot s(n)}<n/2.$$

Sei $m=n/2$. Wir betrachten Rechnungen von M angesetzt auf Eingaben der Form $a^p b^m$, $p\geq m$. Eine Exkursion ist eine Folge von Schritten von M, für die gilt: in jedem Schritt ist der Kopf auf dem Eingabeband auf einem der a's. Im ersten und im letzten Schritt ist der Kopf auf dem Eingabeband auf dem am weitesten rechts oder auf dem am weitesten links stehenden a. Eine Exkursion heißt klein, wenn sie mit dem Kopf auf der gleichen Seite der a's beginnt und endet, andernfalls heißt die Exkursion groß.

Lemma 4.33: Sei E eine Exkursion von M angesetzt auf $a^m b^m$; E' sei eine Exkursion von M angesetzt auf $a^{m+m!}b^m$, die im gleichen V-Zustand und auf der gleichen Seite wie E beginnt. Dann endet E' auch im gleichen V-Zustand und auf der gleichen Seite wie E.

Beweis: Die Behauptung ist offensichtlich, falls E eine kleine

Exkursion ist. Sei also E eine große Exkursion. Wegen $C\cdot s(2m)|Z|\cdot|A|^{C\cdot s(2m)}<m$ gibt es zwei Felder i und j, so daß M während der Exkursion E beim Lesen der a's auf Feld i und j im gleichen V-Zustand ist (M verzählt sich). Hätte die Folge der a's statt der Länge m eine Länge $m+s|i-j|, s\in\mathbb{N}$, so würde die Exkursion offenbar im gleichen V-Zustand und auf der gleichen Seite enden. Da $|i-j|$ kleiner als m ist, also m! teilt, folgt die Behauptung. □

Aus dem Lemma folgt sofort, daß M mit $a^m b^m$ auch $a^{m+m!}b^m$ akzeptiert, denn die Rechnungen für $a^m b^m$ und $a^{m+m!}b^m$ verlaufen in den Exkursionen wie im Lemma beschrieben und zwischen den Exkursionen identisch. □

4.3.3. Eine Zeitschranke

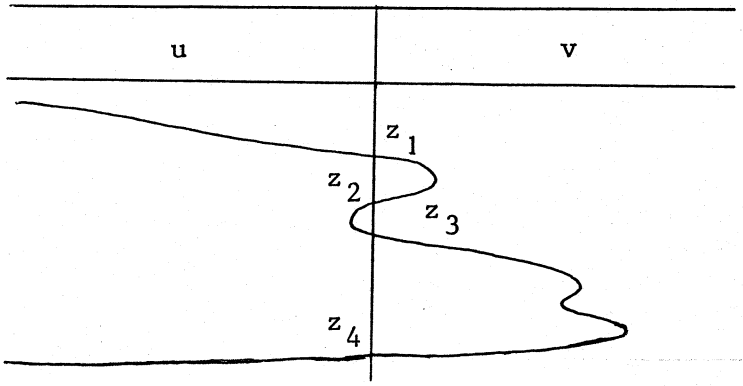
M sei eine 1-Band-Turingmaschine. Eingaben w werden auf den Feldern $0,\ldots,|w|-1$ des Bandes von M eingegeben. Die Maschine M sei so modifiziert, daß sie immer mit dem Kopf auf Feld 0 hält; dadurch kann sich die Zeitkomplexität von M höchstens verdoppeln.

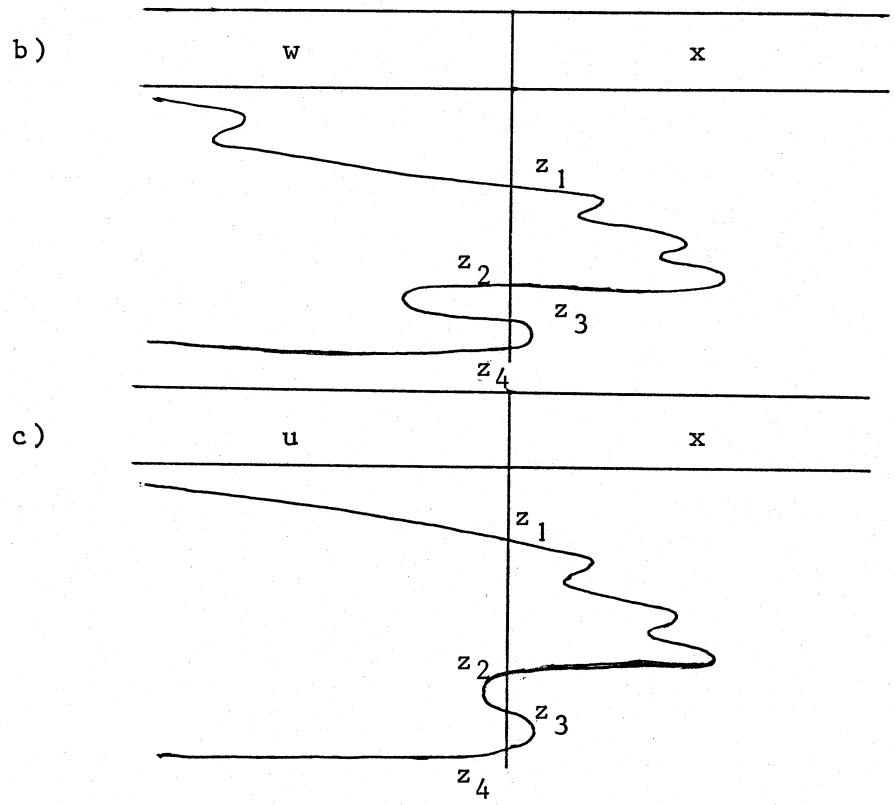
w sei eine Eingabe und i die Nummer eines Felds des Bandes von M. Jedesmal wenn M angesetzt auf w die Grenze zwischen Feld i-1 und Feld i überschreitet, notiere man sich den Zustand von M unmittelbar nach diesem Schritt. Die Folge von Zuständen, die man erhält, heißt die Crossing-Sequenz von M angesetzt auf w an der Stelle i und wird mit CS(w,i) abgekürzt (to cross = engl. kreuzen, hinüberwechseln).

Lemma 4.34: Sind u,v,w,x Zeichenreihen und ist $CS(uv,|u|)=CS(wx,|w|)$, dann gilt $ux\in L(M)$ genau dann, wenn $uv\in L(M)$ gilt.

Der Beweis folgt beinahe unmittelbar aus den folgenden Diagrammen, wo Kopfpositionen als Funktion der Zeit und Crossingsequenzen einer Turingmaschine aufgetragen sind.

a)





Wenn man die Rechnung a) für uv und die Rechnung b) für wx an der Crossing-Sequenz aufschneidet und dann die Stücke für u und x zusammenklebt, erhält man die Rechnung c) für ux. Die Rechnung für ux verläuft links von Feld |u| wie die Rechnung für uv links von Feld |u|. Die Rechnung für ux verläuft rechts von Feld |u|-1 wie die Rechnung für wx rechts von Feld |w|-1. Da M auf Feld 0, also jedenfalls links von Feld |u| hält und damit akzeptiert oder verwirft, folgt die Behauptung. □

Satz 4.35: Sei $x \notin \{0,1\}$ ein Alphabetzeichen. Für die Sprache $L=\{wx^{|w|}w \mid w \in \{0,1\}^*\}$ gilt:

a) $L \in DTIME_1(n^2)$,

b) $L \in DTIME_2(n)$

c) $L \notin DTIME(t(n))$ für $t(n)=o(n^2)$.

Korollar 4.36: Die Simulation von t(n)-zeitbeschränkten Mehrbandmaschinen durch $O(t^2(n))$-zeitbeschränkte 1-Bandmaschinen kann für $t(n)=O(n)$ nicht verbessert werden.

Beweis von Satz 4.35: M sei eine t(n)-zeitbeschränkte 1-Bandmaschine mit Alphabet A und Zustandsmenge Z, die L akzeptiert. Sei $m \in \mathbb{N}$ und $n=3m$. Für $i \in \{m,\ldots,2m-1\}$ und $w,w' \in \{0,1\}^m$, $w \neq w'$ muß

$CS(wx^mw,i) \neq CS(w'x^mw',i)$ gelten, weil M sonst nach Lemma 4.34 mit wx^mw auch wx^ww' akzeptieren würde.

Sei $p(i) = \frac{1}{2^m} \sum_{w\in\{0,1\}^m} |CS(wx^mw,i)|$, d.h. p(i) ist die durchschnittliche Länge der Crossingsequenzen an der Stelle i bei akzeptierenden Berechnungen. Für mindestens die Hälfte aller $w\in\{0,1\}^m$ muß $|CS(wx^mw,i)| \leq 2p(i)$ gelten. Es gibt weniger als $(|Z|+1)^{2p(i)}$ verschiedene Crossingsequenzen der Länge kleiner oder gleich 2p(i). Hiervon muß es nach dem eben Gesagten mindestens 2^{m-1} geben, also $(|Z|+|)^{2p(i)} \geq 2^{m-1}$ bzw. $p(i) \geq (m-1)/(2\ \log(|Z|+1))$. Die Laufzeit von M angesetzt auf $wx^{|w|}w$ ist sicher größer als $\sum_{m<i\leq 2m} |CS(wx^mw,i)|$. Die durchschnittliche Laufzeit bei akzeptierenden Berechnungen ist dann größer als

$$\frac{1}{2^m} \sum_{w\in\{0,1\}^m} \sum_{m\leq i<2m} |CS(wx^mw,i)|$$

$$= \sum_{m<i\leq 2m} p(i) \geq m\cdot(m-1)/(2\ \log(|Z|+|)) = \Omega(n^2).$$

Die Dauer mindestens einer Rechnung muß größer oder gleich der mittleren Laufzeit sein. □

4.3.4. Ein Time-Space-Trade-off

Aus der Programmierpraxis kennt man Fälle, wo man gewisse Funktionen mit verschiedenen Programmen entweder mit wenig Rechenzeit oder mit wenig Speicherplatz berechnen kann, aber man kennt kein Programm, das gleichzeitig wenig Rechenzeit und wenig Speicherplatz benutzt. Das kann daran liegen, daß einem noch kein solches Programm eingefallen ist oder daran, daß es ein solches Programm nicht gibt.

Definition 4.37: Eine Sprache L hat Time-Space-Trade-off, falls es Funktionen $s_1, s_2, t_1, t_2 : \mathbb{N} \to \mathbb{N}$ gibt, so daß gilt:

$s_1(n) = o(s_2(n))$, $t_2(n) = o(t_1(n))$, $L \in C(s_i(n), t_i(n))$ für i=1,2 und $L \notin C(s_1(n), t_2(n))$.

Muß man also in diesem Fall Zeiteffizienz und Platzeffizienz gegeneinander eintauschen (to trade = engl. handeln).

Satz 4.38: Für die Sprache $L=\{wx^{|w|}w \mid w\in\{0,1\}^*\}$ gilt:

a) Ist $s(n)<n$ bandkonstruierbar, so ist $L \in C(s(n), n^2/s(n))$.

b) Ist $s(n) \cdot t(n) = O(n^2)$, so ist $L \notin C(s(n), t(n))$.

Korollar 4.39: L hat Time-Space-Trade-off.

Beweis: $s_1 = \log n, t_1 = n^2/\log n$,

$s_2 = n, t_2 = n$.

Beweis von Satz 4.38: Wir zeigen nur die untere Schranke. M sei eine simultan $Cs(n)$-bandbeschränkte und $C \cdot t(n)$-zeitbeschränkte k-Band-Off-line-Maschine mit Alphabet A und Zustandsmenge Z, die L akzeptiert. Hat M im Laufe einer Rechnung Zustand z, auf den Arbeitsbändern die Inschriften $w_1, \ldots, w_k$ und die Kopfpositionen $v_1, \ldots, v_k$, so nennen wir $(z, w_1, \ldots, w_k, v_1, \ldots, v_k)$ einen V-Zustand von M. Bei Eingabe der Länge n kann M nur $V(n) = (C \cdot s(n))^k \cdot |Z| \cdot |A|^{k \cdot C \cdot s(n)}$ verschiedene V-Zustände annehmen. Nun wiederhole man wörtlich den Beweis von Satz 4.35, wobei man nur "Zustand" durch "V-Zustand" und $|Z|$ durch $V(n)$ ersetzt. Man erhält

$$t(n) > m \cdot (m-1)/(2 \log(V(n)+1))$$
$$= m \cdot (m-1)/(2(\log|Z| + s(n) \cdot k \cdot C \cdot \log|A| + k \log(C \cdot s(n)))),$$
$$= \Omega(n^2/s(n)). \quad \square$$

4.3.5. Eine Komplexitäts-Lücke

Es ist nicht immer der Fall, daß das kleinste Erhöhen im asymptotischen Wachstum der Schranke einer Komplexitätsklasse zu einer neuen Klasse führt.

Satz 4.4o: Ist $t(n) = o(n \log n)$, so gilt $DTIME_1(t(n)) = DTIME_1(n)$.

Beweis: Sei M eine 1-Band-Turingmaschine mit Zustandsmenge Z, die stets hält, aber nicht O(n)-zeitbeschränkt ist. Es wird gezeigt, daß M für unendlich viele Eingaben w mindestens $\Omega(|w| \log |w|)$ Schritte macht.

Lemma 4.41: Besucht eine 1-Band-Turingmaschine angesetzt auf w die Felder i,i', liegen i und i' beide rechts oder beide links von den Feldern $0, \ldots, |w|-1$ und ist $CS(w,i) = CS(w,i')$, so hält die Maschine angesetzt auf w nicht.

Beweis: OBdA sei $|i'| > |i|$. i und i' sind beides Nummern von

Feldern, die ursprünglich leer sind.
$CS(w,i)=CS(w,i')$ impliziert ähnlich wie im Bweis von Lemma 4.34 $CS(w,i')=CS(w,i'+(i'-i))$. Durch Induktion folgt $CS(w,i)=CS(w,i'+s(i'-i))$ für alle $s\in\mathbb{N}$. Die Maschine besucht also unendlich viele Felder und hält nicht. □

Lemma 4.42: Ist $r\in\mathbb{N}$ und ist Q eine 1-Bandmaschine mit Zustandsmenge Z, die immer hält und die für keine Eingabe eine Crossingsequenz mit Länge > r erzeugt, so ist Q O(n)-zeitbeschränkt.

Beweis: Es gibt weniger als $(|Z|+1)^r$ Crossingsequenzen der Länge höchstens r. Nach Lemma 4.41 kann Q angesetzt auf Eingabe w nur die Felder $-(|Z|+|)^r,\ldots,|w|+(|Z|+1)^r-1$ besuchen. Es folgt, daß M $r\cdot(n+2(|Z|+1)^r)$-zeitbeschränkt ist. □

Zurück zum Beweis von Satz 4.4o: M hält für alle Eingaben, ist aber nicht O(n)-zeitbeschränkt. Dann gibt es für jedes r eine Menge von Worten w, so daß M angesetzt auf w eine Crossingsequenz der Länge mindestens r erzeugt. Für jedes r sei w(r) ein kürzestes solches Wort.

Lemma 4.43: Für jedes r gilt: Es gibt keine drei verschiedenen Indizes $i,j,k\in\{0,\ldots,|w(r)|-1\}$, so daß $CS(w(r),i)=CS(w(r),j)=CS(w(r),k))$.

Beweis: Angenommen, es gibt drei solche Indizes; OBdA gilt $i<j<k$. Nach Definition von w(r) gibt es ein Feld ℓ, so daß $|CS(w(r),\ell)|\geq r$ gilt. Ist $\ell\notin\{i,\ldots,j-1\}$, so ist $w_1\ldots w_i w_{j+1}\ldots w(r)_{|w(r)|}$ ein kürzeres Wort als w(r), das eine Crossingsequenz der Länge mindestens r erzeugt; andernfalls ist $w_1\ldots w_j w_{k+1}\ldots w(r)_{|w(r)|}$ ein kürzeres solches Wort. □

Sei $n=|w(r)|$. Die Laufzeit t von M angesetzt auf w(r) ist größer oder gleich $\sum_{1\leq i\leq n}|CS(w(r),i)|$. Da mindestens n/2 verschiedene Crossingsequenzen vorkommen müssen, folgt

$$t \geq \min_{\substack{S\subset Z^* \\ |S|=n/2}} \sum_{v\in S}|v|.$$

Das Minimum wird angenommen, wenn man $S=Z^0UZ^1UZ^2U\ldots Z^{j(n)}US'$ mit $S'\subset Z^{j(n)+1}$ wählt, wobei j(n) möglichst klein gewählt wird. Wegen

$$\frac{n}{2} = |S| < \sum_{i=0}^{j(n)+1}|Z|^i = \frac{|Z|^{j(n)+2}-1}{|Z|-1} \leq |Z|^{j(n)+2}$$

folgt

$$j(n) \geq \frac{\log n-1}{\log|Z|} -2,$$

also

$$t \geq \sum_{v\in S} |v| \geq \sum_{i=0}^{j(n)} i\cdot|Z|^i \geq j(n)|Z|^{j(n)}$$

$$= j(n)2^{(\log|Z|)j(n)} \geq (\frac{\log n-1}{\log|Z|} -2)2^{\log n-1-2\cdot\log|Z|}$$

$$= (\frac{\log n-1}{\log|Z|} -2) \frac{n}{2|Z|^2} = \Omega(n \log n). \quad \square$$

Offene Probleme

Die in diesem Abschnitt gezeigten unteren Schranken sind die größten unteren Schranken für die Komplexität von Turingmaschinen, die man zur Zeit mit rein kombinatorischen Mitteln herleiten kann. Es stellt sich das Problem , mit rein kombinatorischen Methoden folgendes herzuleiten:

a) Untere Schranken $\geq \omega(n^2)$ für die Zeitkomplexität von 1-Band-Turingmaschinen;

b) untere Schranken $\geq \omega(n)$ für die Zeitkomplexität von 2-Bandmaschinen oder Registermaschinen;

c) untere Schranken $\geq \omega(\log n)$ für die Bandkomplexität von Off-line-Turingmaschinen.

Weitere Probleme sind:

d) Ist die Simulation von vielen Bändern durch 1 Band von Satz 1.37 auch für $t(n)=\omega(n)$ optimal?

e) Gibt es beliebig komplizierte Sprachen mit Time-Space-Trade-Off, d.h. gibt es zu jeder berechenbaren Funktion $t(n)$ eine Sprache $L\notin DTIME(t(n))$ die Time-Space-Trade-Off hat?

Übungen

1. Ist $t(n)=0(n^2)$, so ist

$$\{w_1\ldots w_r w_r\ldots w_1 \mid w_i\in\{0,1\}^*, r\in\mathbb{N}\} \notin DTIME_1(t(n)).$$

2. Ist $T:\mathbb{N}\to\mathbb{N}$ auf 1-Bandmaschinen zeitkonstruierbar, $T(n)<n^2$

und $t(n)=O(T(n))$, so ist
$DTIME_1(t(n)) \subsetneq DTIME_1(T(n))$.

3. Ist M eine $t(n)$-zeitbeschränkte 1-Bandmaschine, die bei Eingabe $w_1 \cdots w_{n/(2 \log n)}, w_i \in \{0,1\}^{2 \log n}$, die w_i als Binärzahlen auffaßt und die w_i so umsortiert, daß die entsprechenden Zahlen der Größe nach geordnet sind, dann ist $t(n)=\Omega(n^2)$.

   Hinweis: Man konstruiere aus einer solchen Maschine M eine $O(t(n)+n \log n)$-zeitbeschränkte Maschine Q, die $\{wx^{|w|}w \mid w \in \{0,1\}^*\}$ akzeptiert.

4. $DTIME_1(t(n)) \subseteq C_1(t(n)/\log t(n), t(n))$.

   Hinweis: Man zeige, daß jede $t(n)$-zeitbeschränkte 1-Bandmaschine schon $O(t(n) \log t(n))$-bandbeschränkt ist.

5. Nichtdeterminismus

Wir erweitern das Turingmaschinenmodell:

Bei den bisher betrachteten Maschinen ist das, was in einem Schritt geschieht, vollständig durch die gerade gelesenen Zeichen und den Zustand der Maschinen bestimmt. Hierfür sagen wir: die Maschinen sind deterministisch. Bei dem erweiterten, nichtdeterministischen Modell soll es erlaubt sein, daß eine Maschine in einem Schritt endlich viele Alternativen dafür, was getan werden soll, hat.

Diese Maschinen sehen auf den ersten Blick sehr dubios aus. Sie können etwa, wenn sie zweimal mit der gleichen Eingabe gestartet werden, zu verschiedenen Ergebnissen kommen; eine reale Rechenanlage, die so etwas tut, ist reparaturbedürftig. Es stellt sich jedoch heraus, daß sich diese Maschinen zum Klassifizieren der Komplexität interessanter Sprachfamilien eignen und daß man sie als technisches Hilfsmittel sowohl zum Entwurf deterministischer Algorithmen als auch zum Verkürzen von Beweisen verwenden kann.

5.1. Nichtdeterministische und deterministische Komplexitätsklassen

5.1.1. Nichtdeterministische Turingmaschinen

Wir führen die Definition nichtdeterministischer Maschinen formal nur für 1-Bandmaschinen durch. Die Definition kann in naheliegender Weise auf k-Bandmaschinen ausgedehnt werden.

Definition 5.1: Eine nichtdeterministische 1-Band-Turingmaschine wird durch ein Tupel (Z,A,δ,z_0,E) spezifiziert, wobei gilt: Z und A sind endliche Mengen (Zustandsmenge und Bandalphabet), $Z \cap A = \emptyset$, $\{0,1,\#,B\} \subseteq A$.

$\delta \subset Z \times A \times Z \times A \times \{0,R,L\}$ ist eine Relation (Übergangsrelation).

$z_0 \in Z$ (Anfangszustand, Endzustand).

$E = \{z \in Z \mid \forall a,z',c,s : (z,a,z',c,s) \notin \delta\}$ (Menge der Endzustände).

$(z,a,z',a',s) \in \delta$ bedeutet: wenn die Maschine im Zustand z das Zeichen a liest, kann sie in Zustand z' gehen, das Zeichen a' drucken und die Kopfbewegung s (R = rechts, L = links, 0 = keine Bewegung) ausführen.

Wir schreiben Konfigurationen von 1-Bandmaschinen wieder als Zeichenreihen uzv mit $u,v \in A^*, z \in Z$. (Vgl. Kapitel 1.3).

Definition 5.2: Sind k und k' Konfigurationen, so stehe "$k \vdash k'$" dafür, daß die Maschine aus Konfiguration k in einem Schritt Konfiguration k' erreichen kann. Wir sagen auch: k' ist eine Nachfolgekonfiguration von k. Die formale Definition von "$\vdash$" erhält man, wenn man in Definition 1.17 "$\delta(z,a)=(z',c,s)$" durch $(z,a,z',c,s) \in \delta$ ersetzt.

Beispiel 5.3: Angesetzt auf leeres Band schreibt die folgende Maschine irgendeine Zeichenreihe aus $\{0,1\}^*$ auf das Band:

$$\delta = \{(z_o,B,z_o,0,R),$$
$$(z_o,B,z_o,1,R),$$
$$(z_o,B,z_e,B,0)\}.$$

Für dieses Verhalten sagen wir auch: die Maschine rät eine Zeichenreihe.

Definition 5.4: Nichtdeterministische Akzeptoren werden genau wie im deterministischen Fall definiert, indem man einen akzeptierenden und einen verwerfenden Endzustand einführt.

Definition 5.5: Eine endliche Rechnung von M angesetzt auf w ist eine endliche Folge von Konfigurationen $(k_o,\ldots,k_t)$, für die gilt:

i) k_o hat die Form $B\ldots Bz_owB\ldots B$

ii) $k_i \vdash k_{i+1}$ für alle $i \in \{0,\ldots,t-1\}$

iii) k_t enthält einen Endzustand.

t heißt Länge der Rechnung. Ist $k_o = z_ow$, so heißt die Rechnung eine kanonische Rechnung von M angesetzt auf w und $\max\{|k_i|-1 \mid 0 \leq i \leq t\}$ der Bandverbrauch dieser Rechnung. Ist M ein Akzeptor, so heißt die Rechnung akzeptierend bzw. verwerfend, wenn k_t den akzeptierenden bzw. verwerfenden Endzustand enthält. Eine unendliche Folge von Konfigurationen, die i) und ii) erfüllt, heißt eine unendliche Rechnung von M angesetzt auf w.

Da eine nichtdeterministische Maschine angesetzt auf die gleiche Eingabe verschiedene Rechnungen durchführen kann, die insbesondere in verschiedenen Zuständen enden können, muß neu definiert werden, was die von einer Maschine akzeptierte Sprache ist.

Definition 5.6: Ist M ein Akzeptor, so heißt die Menge M
$L(M)=\{w|$ es gibt eine akzeptierende Berechnung von M angesetzt auf $w\}$ die von M akzeptierte Sprache.

Beispiel 5.7: Sei $L=\{bin(i)|$ i ist keine Primzahl$\}$. Wir beschreiben eine nichtdeterministische Turingmaschine M.

Angesetzt auf Eingabe bin(i) rät (vergl. Beispiel 1.3) M die Binärdarstellungen irgendwelcher Zahlen $j,k\in\mathbb{N}$. Dann testet M, ob $j\neq 1, k\neq 1$ und $j\cdot k=i$ gilt. Verlaufen alle drei Tests positiv, akzeptiert M die Eingabe, andernfalls verwirft M. Offenbar ist $L(M)=L$.

5.1.2. Komplexität nichtdeterministischer Maschinen

Da eine nichtdeterministische Maschine angesetzt auf eine und dieselbe Eingabe verschieden lange Rechnungen bzw. Rechnungen mit verschiedenem Bandverbrauch durchführen kann, muß die Komplexität von nichtdeterministischen Maschinen neu definiert werden. Die Definition ist sehr großzügig: nichtakzeptierende Rechnungen kosten nichts, und von allen akzeptierenden Rechnungen zählt nur die effizienteste.

Definition 5.8: M sei ein nichtdeterministischer Akzeptor, w eine Eingabe.

$$T_M(w) = \begin{cases} \min\{t| \text{ es gibt eine akzeptierende Berechnung der Länge } t \text{ von M angesetzt auf } w\} & \text{falls } w\in L(M) \\ 0 & \text{sonst} \end{cases}$$

$$S_M(w) = \begin{cases} \min\{s| \text{ es gibt eine akzeptierende kanonische Berechnung mit Bandverbrauch } s \text{ von M angesetzt auf } w\} & \text{falls } w\in L(M) \\ 0 & \text{sonst} \end{cases}$$

M heißt t(n)-zeitbeschränkt falls für alle w $T_M(w)\leq t(|w|)$ gilt.
M heißt s(n)-bandbeschränkt falls für alle w $S_M(w)\leq s(|w|)$ gilt.

Beispiel 5.9: Verwendet man bei der Maschine aus Beispiel 5.7 zur Multiplikation von Binärzahlen ein zur Schulmethode analoges Verfahren, so ist die Maschine $O(n^2)$-zeitbeschränkt und O(n)-bandbeschränkt. Man beachte, daß für kein $k\in\mathbb{N}$ eine deterministische $O(n^k)$-zeitbeschränkte Maschine bekannt ist, die $L=\{bin(i)|i$ keine

Primzahl} akzeptiert. Andererseits ist kein Beweis für $L \notin DTIME_2(n)$ bekannt. Komplexitätsklassen werden ebenso wie im deterministischen Fall erklärt.

Definition 5.1o:

$NTIME_k(t(n)) = \{L \mid$ es gibt eine nichtdeterministische $O(t(n))$-zeitbeschränkte k-Band-Turingmaschine, die L akzeptiert$\}$.

$NTIME(t(n)) = \bigcup_k NTIME_k(t(n))$

$NTAPE_k(s(n)) = \{L \mid$ es gibt eine nichtdeterministische $O(s(n))$-bandbeschränkte k-Band-Turingmaschine, die L akzeptiert$\}$.

Gilt $s(n) \geq n$ nicht für alle $n \in \mathbb{N}$, wird $NTAPE_k(s(n))$ durch nichtdeterministische Off-line-Turingmaschinen definiert.

$NTAPE(s(n)) = \bigcup_k NTAPE_k(s(n))$.

Einige einfache Beziehungen zwischen diesen Klassen enthält

Satz 5.11: a) $NTIME(t(n)) \subseteq NTIME_1(t^2(n))$

b) $NTAPE(s(n)) \subset NTAPE_1(s(n))$

c) $NTIME(t(n)) \subseteq NTAPE(t(n))$.

Beweis: Die Simulation von k Bändern auf k+1 Spuren eines Bands von Satz 1.37 läßt sich auf nichtdeterministische Maschinen übertragen. Hieraus folgen a) und b). c) ist trivial. □

Man kann eine zu Kapitel 4 analoge Theorie für nichtdeterministische Turingmaschinen entwickeln. Wir tun dies teilweise in den Übungen und verweisen insbesondere bezüglich Hierarchiesätzen auf die Literatur [7,65].

5.1.3. Beziehungen zwischen Determinismus und Nichtdeterminismus

Einige einfache Beziehungen zwischen deterministischen und nichtdeterministischen Komplexitätsklassen enthält

Satz 5.12: a) $DTIME(t(n)) \subseteq NTIME(t(n))$

b) $DTAPE(s(n)) \subseteq NTAPE(s(n))$

c) Ist t(n) bandkonstruierbar, so ist
$NTIME(t(n)) \subseteq DTAPE(t(n))$
$\subseteq DTIME(2^{O(t(n))})$.

Beweis: a) und b) sind trivial, da jede deterministische Maschine

auch eine nichtdeterministische Maschine ist. Zum Beweis von c) sei M eine $C \cdot t(n)$-zeitbeschränkte nichtdeterministische k-Bandmaschine mit Alphabet A und Zustandsmenge Z. Ein Protokoll einer Rechnung von M ist eine Folge $(P^1,\ldots,P^t)$ von 2k+1-Tupeln, so daß für alle $i \in \{1,\ldots,t\}$ gilt:

$P^i \in Z \times A^k \times \{N,R,L\}^k$

P^i_1 = Zustand von M zu Beginn von Schritt i

P^i_{j+1} = das im Schritt i auf Band j gedruckte Zeichen, $1 \leq j \leq k$

P^i_{k+j+1} = Kopfbewegung auf Band j im i-ten Schritt, $1 \leq j \leq k$.

t heißt Länge des Protokolls.

Wir beschreiben eine deterministische Maschine Q. Angesetzt auf Eingabe w berechnet Q die Binärdarstellung von $t(|w|)$ und zählt dann in lexikographischer Ordnung alle Folgen $(P_1,\ldots,P_{C.t(|w|)})$ mit $P_i \in Z \times A^k \times \{N,R,L\}^k$ auf. Jedesmal, wenn eine neue Folge erzeugt wurde, wird die zuvor erzeugte Folge gelöscht und es wird getestet, ob die neue Folge oder ein Anfangsstück davon Protokoll einer akzeptierenden Berechnung von M angesetzt auf w ist; hierzu simuliert Q die Maschine M Schritt für Schritt; im i-ten Schritt wird die vom Tupel p_i vorgeschriebene Aktion ausgeführt; ist diese Aktion nicht mit der Übergangsrelation von M verträglich, ist die getestete Folge kein Protokoll einer Rechnung von Q.

Ist eine der aufgezählten Folgen oder ein Anfangsstück davon Protokoll einer akzeptierenden Berechnung von M angesetzt auf w, akzeptiert M, andernfalls verwirft M. Offenbar gilt $w \in L(Q) \Leftrightarrow w \in L(M)$. Da stets nur zwei Folgen gleichzeitig gespeichert werden müssen, ist Q $O(t(n))$-bandbeschränkt. Die zweite Inklusion folgt aus Satz 4.3 e). □

Der folgende Satz enthält eine effiziente Simulation von bandbeschränkten nichtdeterministischen Maschinen durch deterministische Maschinen.

Satz 5.13 (Savitch): Ist $s(n) \geq \log n$ bandkonstruierbar, so ist $NTAPE(s(n)) \subseteq DTAPE(s^2(n))$.

Wir führen den Beweis für den Fall, wo $s(n) \geq n$ für alle n gilt. Andernfalls betrachtet man Off-line-Maschinen, und der gleiche Beweis geht mit naheliegenden Änderungen durch.

Sei M eine $C\cdot s(n)$-bandbeschränkte nichtdeterministische 1-Bandmaschine mit Alphabet A und Zustandsmenge Z.

Lemma 5.14: Ist $w\in L(M)$ und $|w|=n$, so gibt es eine akzeptierende Berechnung der Länge höchstens $|Z|\cdot C\cdot s(n)|A|^{C\cdot s(n)}$ von M angesetzt auf w.

Beweis: Da M $C\cdot t(n)$-bandbeschränkt ist, muß sich in jeder längeren Rechnung eine Konfiguration wiederholen. Ist $k_o\vdash..\vdash k'\vdash k..k\vdash k''\vdash..\vdash k_\ell$ eine akzeptierende Berechnung, so auch $k_o\vdash..\vdash k'\vdash k\vdash k''\vdash..\vdash k_\ell$. □

Wir modifizieren die Maschine M so, daß für den akzeptierenden Endzustand z_+ und alle $a\in A$ gilt: $(z_+,a,z_+,a,0)\in\delta$, d.h. nach Erreichen des akzeptierenden Endzustands hält M nun nicht mehr, sondern "läuft weiter, tut aber nichts". Sinn dieser Formalität ist, daß man sich auf die Betrachtung von Berechnungen beschränken kann, deren Länge eine Zweierpotenz ist.

Definition 5.15: M sei eine 1-Band-Turingmaschine mit Alphabet A und Zustandsmenge Z. Für $m\in\mathbb{N}$ sei $K(s)=\{uzv\,|\,u,v\in A^*,z\in Z^*,|uv|\leq s\}$ die Menge der Konfigurationen mit einer Bandinschrift der Länge höchstens s. Für Konfigurationen k,k' und $t,s\in\mathbb{N}$ soll

$$k\vdash^t k'$$

bzw. $$k\vdash_s k'$$

bzw. $$k\vdash^t_s k'$$

bedeuten, daß es eine Folge von Konfigurationen $k_o,\ldots,k_r$ gibt, so daß gilt: $k=k_o \ldots k_r=k'$ und

$$r=t$$

bzw. $$k_i\in K(s) \text{ für alle } i$$

bzw. $$r=t \text{ und } k_i\in K(s) \text{ für alle } i.$$

Aus dem bisher Gesagten folgt

Lemma 5.16: Sei $w\in A^*$, $|w|=n$ und $\tau(n)=\lceil \log|Z|\cdot C\cdot t(n)\cdot|A|^{C\cdot t(n)}\rceil$, dann ist $w\in L(M)$ genau dann, wenn

$$z_o w \vdash^{2^{\tau(n)}}_{C\cdot t(n)} k_+$$

für irgendeine Konfiguration k_+, in der der akzeptierende Endzustand vorkommt, gilt.

Wir beschreiben im folgenden ein Verfahren, für $s,\tau\in\mathbb{N}$ mit $2^\tau\leq 2\cdot|Z|\cdot S\cdot|A|^s$ und beliebige Konfigurationen $k,k'\in K(s)$ auf Platz $O(s\cdot\tau)$ zu testen, ob $k \vdash\!\!\!\!\!\!\frac{2^\tau}{s}\; k'$ gilt. Hieraus erhält man ein Verfahren, für Eingaben w mit $|w|=n$ zu testen, ob $w\in L(M)$ gilt, indem man alle in Frage kommenden Konfigurationen k_+ lexikographisch aufzählt, und für jeden Kandidaten k_+ testet, ob $z_o w \vdash\!\!\!\!\!\!\frac{2^{\tau(n)}}{C\cdot s(n)} k_+$ gilt. Da alle Konfigurationen auf dem gleichen Speicherplatz aufgezählt werden können und auch alle Tests auf dem gleichen Speicherplatz durchgeführt werden können, erhält man als Bandverbrauch dieses Verfahrens $O(s(n))+O(s(n)\cdot\tau(n))=O(s^2(n))$, da $\tau(n)=O(s(n))$.

Wir müssen noch zeigen, wie man auf Band $O(s\cdot\tau)$ testet, ob $k \vdash\!\!\!\!\!\!\frac{2^\tau}{s} k'$ gilt. Die entscheidende Beobachtung ist hier, daß $k\frac{2^\tau}{s}k'$ genau dann gilt, wenn

$$k \vdash\!\!\!\!\!\!\frac{2^{\tau-1}}{s}\; k'' \quad k'' \vdash\!\!\!\!\!\!\frac{2^{\tau-1}}{s}\; k'$$

für irgendein $k''\in K(s)$ gilt. Dies legt die folgende Vorgehensweise nahe: Um $k \frac{2^\tau}{s}k'$ zu testen, zähle man auf Speicherplatz $O(s)$ in lexikographischer Ordnung alle $k''\in K(s)$ auf und teste für jedes k'' zuerst, ob $k \frac{2^{\tau-1}}{s}k''$ gilt, und dann auf dem gleichen Speicherplatz, ob $k'' \frac{2^{\tau-1}}{s}k'$ gilt.

Sei $B(s,\tau)$ der Speicherplatz, der hinreicht, um zu testen, ob $k'' \vdash\!\!\!\!\!\!\frac{2^{\tau-1}}{s}k'$ gilt. Wir zeigen, daß man das eben skizzierte Verfahren so programmieren kann, daß für ein passendes $D\in\mathbb{N}$ $B(s,\tau)\leq B(s,\tau-1)+D\cdot s$ und $B(s,1)\leq D\cdot s$ gilt. Durch Induktion über τ folgt $B(s,\tau)\leq D\cdot s\cdot\tau$.

Zunächst einige Notationen: Die Menge aller Konfigurationen von M sei durch die Relation $\leq$ der Länge nach und unter Folgen gleicher Länge lexikographisch geordnet; $k_{max}(s)$ bzw. $k_{min}(s)$ bezeichne das bezüglich $\leq$ größte bzw. kleinste Element in $K(s)$; $k+1$ bezeichne die bezüglich $\leq$ auf k folgende Konfiguration.

Wir beschreiben eine Maschine Q, aus der man leicht das gesuchte Verfahren gewinnen kann. Die Maschine Q manipuliert Bandinschriften der Form $T_1\ldots T_r$, wobei die T_j Zeichenreihen der Form $(bin(s),k,k',k'',bin(\tau),i)$ mit $k,k',k''\in K(s)$, $s,\tau\in\mathbb{N}$ und $i\in\{0,1\}$ sind. (Die Maschine Q hat Komma und Klammern als Alphabetzeichen). Das Programm der Maschine Q hat drei Phasen (Expandieren, Schrumpfen, neuer Versuch), zwischen denen im Verlauf einer Rechnung hin- und hergesprungen wird. In einer Phase wird immer die aktuell am weitesten rechts stehende Zeichenreihe T_j verarbeitet. Die ein-

zelnen Phasen werden so organisiert, daß gilt:

(*) Wird die Phase "Expandieren" mit einer Bandinschrift $T_1 \ldots T_r T$ mit $T=(\text{bin}(s),k,k',k'',\text{bin}(\tau),i)$ und $k,k',k'' \in K(s)$ begonnen, und gibt es eine Konfiguration $k^{(3)} \in K(s)$, so daß $k^{(3)} \geq k''$ und $k \vdash_s^{2^{\tau-1}} k^{(3)} \vdash_s^{2^{\tau-1}} k'$ gilt, so wird T schließlich gelöscht und das Programm ist danach in der Phase "Schrumpfen". Andernfalls wird T ebenfalls schließlich gelöscht, aber das Programm ist danach in der Phase "Neuer Versuch".

In den einzelnen Phasen geschieht folgendes:

Expandieren:

$\tau=1$: Gilt $k \vdash k'' \vdash k'$, wird T gestrichen und dann in die Schrumpfphase gegangen, andernfalls wird in die Phase "Neuer Versuch" gegangen.

$\tau>1$: Ist $i=0$, wird die Folge der Tupel nach rechts um $(\text{bin}(s),k,k'',k_{min}(s),\text{bin}(\tau-1),0)$, andernfalls um $(\text{bin}(s),k'',k',k_{min}(s),\text{bin}(\tau-1),0)$ verlängert. Danach wird eine neue Expansionsphase begonnen.

Schrumpfen:

Ist $i=0$, wird $i=1$ gesetzt und expandiert. Ist $i=1$, wird T gelöscht und eine neue Schrumpfphase begonnen.

Neuer Versuch:

Ist $k''>k_{max}(s)$, wird k'' auf $k''+1$ und i auf 0 gesetzt, dann wird eine Expansionsphase begonnen. Ist $k''=k_{max}(s)$, wird T gelöscht und eine neue Phase "Neuer Versuch" begonnen.

Durch Induktion über τ zeigt man leicht, daß (*) für alle τ gilt.

Sei $B(s,\tau)$ der Speicherplatz, den die Maschine Q angesetzt auf eine Bandinschrift $T=(\text{bin}(s),k,k',k'',\text{bin}(\tau),i)$ mit $k,k',k'' \in K(s)$ benutzt. Dann ist offenbar

$$B(s,\tau) \leq |T| + B(s,\tau-1)$$
$$\leq 3s+\log s+1+\log \tau+1+8+B(s,\tau-1)$$
$$\leq 14s+\log \tau+B(s,\tau-1).$$

Für $2^\tau \leq 2\cdot|z|\cdot s\cdot|A|^s$ folgt

$$\log \tau \leq 1+\log|Z|+\log s+s\cdot\log|A|$$
$$\leq s(2+\log|Z|+\log|A|).$$

Also

$$B(s,\tau)\leq Ds+B(s,\tau-1) \text{ für } D\leq 16+\log|Z|+\log|A|.$$

Ferner gilt offenbar $B(s,1)\leq 3s+\log s+10\leq Ds$.

Mit Hilfe der Maschine Q kann man nun für Konfigurationen $k,k'\in K(s)$ auf Platz $D\cdot s\cdot\tau$ testen, ob $k \frac{2^\tau}{s} k'$ gilt, indem man Q mit Inschrift $(bin(s),k,k',k_{min}(s),bin(2^\tau),0)$ startet. $k \not\frac{2^\tau}{s} k'$ gilt genau dann, wenn Q schließlich die gesamte Bandinschrift löscht und danach in der Phase "Schrumpfen" ist. □

Offene Probleme:

Sehr wenig ist darüber bekannt, "wieviel Nichtdeterminismus wirklich nützt".

a) Gilt $DTIME(t(n))\subsetneq NTIME(t(n))$ für irgendeine Funktion $t(n)\geq n$?

b) Gilt $DTAPE(s(n))\subsetneq NTAPE(s(n))$ für irgendeine Funktion $s(n)\geq \log n$. Für $s(n)=n$ ist diese Frage als "das LBA-Problem" bekannt (vergl. Kapitel 5.4).

Übungen

1. Jede $t(n)$-zeitbeschränkte und $s(n)$-bandbeschränkte nichtdeterministische Turingmaschine kann durch eine $O(t(n))$ und $O(s(n))$ bandbeschränkte nichtdeterministische Maschine, die in jedem Schritt höchstens 2 Alternativen hat, simuliert werden.

2. a) $L=\{wx^{|w|}w \mid w\in\{0,1\}^*\}\notin NTIME_1(t(n))$, falls $t(n)=o(n^2)$.
   b) Für $\overline{L}=\{0,1,x\}^*\setminus L$ gilt: $\overline{L}\in NTIME_1(n \log n)$.
   c) Ist $t(n)=o(n^2)$, so ist $NTIME_1(n \log n)\setminus DTIME_1(t(n))\neq\emptyset$.
   d) Welche Ergebnisse aus Kapitel 4.3 lassen sich auf den nichtdeterministischen Fall übertragen?

3. a) $NTIME(t(n))=NTIME_2(t(n))$.
   Hinweis: Simulation von einer k-Bandmaschine M durch eine 2-Bandmaschine Q mit nur einem konstanten Faktor Zeitverlust. Q versucht zunächst, ein Protokoll einer Rechnung von M zu raten.

   b) Ist $T(n)$ zeitkonstruierbar und $t(n)=o(T(n))$, so ist $DTIME(t(n))\subsetneq NTIME(T(n))$.

4. a) Aus $DTIME(n)=NTIME(n)$ folgt $DTIME(t(n))=NTIME(t(n))$ für jede zeitkonstruierbare Funktion $t(n)\geq n$.

b) Aus DTAPE(log n)=NTAPE(log n) folgt DTAPE(s(n))=NTAPE(log n) für jede bandkonstruierbare Funktion $s(n) \geq \log n$.
Hinweis: Man benutze Padding.

5.2. Vergleich zwischen Speicherplatz und Rechenzeit

Wir können jetzt zwei ganz einfache Fragen für deterministische Maschinen beantworten.

i) Können O(t(n))-bandbeschränkte Akzeptoren mehr Sprachen erkennen als O(t(n))-zeitbeschränkte Akzeptoren?

ii) Gibt es Zeithierarchien über festem Band, d.h. kann man, wenn man die Bandschranke s(n) festhält und die Zeitschranke t(n) wachsen läßt, mehr und mehr Sprachen erkennen?

Die Evidenz aus der Programmierpraxis dafür, daß zumindest die Antwort auf Frage i) positiv ausfällt, ist überwältigend, aber der Beweis ist nicht trivial.

5.2.1. Ein Spiel auf Graphen

Wir studieren ein Spiel auf Graphen, das wir später zur Analyse des Bandverbrauchs bei einer Simulation brauchen. Für grundlegende Definitionen und Notationen aus der Graphentheorie verweisen wir auf § 3.3.1.

Definition 5.17: Sei $G=(V,E)$ ein gerichteter Graph. Für $(v,w) \in E$ sagen wir auch: v ist Vorgänger von w und w ist Nachfolger von v. Die Anzahl der Vorgänger eines Knotens heißt Ingrad des Knotens. Der Ingrad von G ist das Maximum der Ingrade aller Knoten von G.

$G=(V,E)$ sei ein gerichteter zykelfreier Graph und $v_o \in V$ ein im voraus ausgezeichneter Knoten. Das Spiel wird gespielt, indem man Marken nach folgenden Regeln auf die Knoten von G setzt bzw. von den Knoten entfernt.

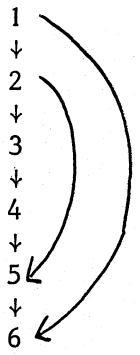
i) Haben alle Vorgänger eines Knotens v eine Marke, so darf man eine Marke auf v setzen. Insbesondere darf man immer eine Marke auf eine Quelle setzen.

ii) Man darf jederzeit eine Marke entfernen.

Ein Zug ist das Setzen oder das Entfernen einer Marke. Ziel des Spiels ist es, den ausgezeichneten Knoten v_o zu markieren und

dabei die Anzahl von Marken, die gleichzeitig auf dem Graphen G sind, möglichst klein zu halten.

Beispiel 5.18: G sei der folgende Graph:



v_o=6 soll markiert werden.

Strategie 1: Markiere 1,...,6 in dieser Reihenfolge.
(6 Marken, 6 Züge)

Strategie 2: (3 Marken, 14 Züge)

Zug →	1	2	3	4	5	6	7	8	9	1o	11	12	13	14
Markiere	1	2		3		4		1	2		5		1	6
Entferne Marke von			1		2		3			1		2		

Dieses Spiel ist eingehend studiert worden [8,3o,39,48-51,53]. Intuitiv ist dieses Spiel in folgendem Sinn ein Modell für Rechnungen. Die Knoten des Graphen stellen das Ergebnis von Operationen dar. Hat ein Knoten v die Vorgänger $v_1,\ldots,v_k$, so heißt das, daß zur Ausführung der zu v gehörigen Operation die Ergebnisse der zu $v_1,\ldots,v_k$ gehörigen Operationen vorliegen müssen. Die Marken stellen Speicherplätze dar. Das Speichern des Ergebnisses einer Operation erfordert einen Speicherplatz. Regel i) gibt dann gerade wieder, daß eine Operation nur ausgeführt (und das Ergebnis gespeichert) werden kann, wenn alle Operanden schon berechnet und irgendwo gespeichert sind. Um Speicherplatz zu sparen, ist es möglich, Zwischenergebnisse zu löschen (Regel 2) und sie neu zu berechnen, wenn man sie wieder braucht.

Wir zeigen erst, daß man in gerichteten Graphen jeden Knoten überhaupt markieren kann, und geben dann an, wie man große Graphen, die kleinen Ingrad haben, mit wenig Marken markieren kann.

Lemma 5.19: Die Knoten jedes gerichteten zykelfreien Graphen

$G=(V,E)$ können so von 1 bis $|V|$ numeriert werden, daß aus $(v,w)\in E$ folgt: Nummer von v < Nummer von w.

Beweis durch Induktion über $|V|$: Hat G nur einen Knoten, wird dieser mit 1 numeriert.

Sei $|V|>1$. Da G zykelfrei ist, hat G eine Senke s.

Sei $G_s=(V_s,E_s)$, $V_s=V\setminus\{s\}$ (also $|V_s|=|V|-1$), $E_s=\{(u,v)\in E \mid v\neq s\}$.

Die gewünschte Numerierung erhält man, indem man die Knoten von G_s nach Induktionsvoraussetzung mit $1,\ldots,|V_s|$ numeriert und s mit $|V|$ numeriert. □

Lemma 5.2o: Jeder Knoten eines gerichteten zykelfreien Graphen $G=(V,E)$ kann mit $|V|$ Marken in $|V|$ Zügen markiert werden.

Beweis: Die Knoten von G seien nach Lemma 5.19 numeriert. Man setze Marken auf die Knoten in der Reihenfolge ihrer Nummern. □

Definition 5.21: $G=(V,E)$ sei ein gerichteter, azyklischer Graph. Zwei Graphen $G_1=(V_1,E_1)$ und $G_2=(V_2,E_2)$ heißen eine Zerlegung von G, falls $V=V_1\cup V_2$, $V_1\cap V_2=\emptyset$, $E_i=\{(u,v)\in E \mid u,v\in V_i\}$ für $i=1,2$ und falls es in G keinen Pfad von einem Knoten in G_2 zu einem Knoten in G_1 gibt.

Lemma 5.22: Jeder gerichtete zykelfreie Graph $G=(V,E)$ mit n Kanten und Ingrad k hat eine Zerlegung G_1,G_2 mit $\frac{n}{2} \leq |E_2| \leq \frac{n}{2} + k$.

Beweis: G_1,G_2 sei eine beliebige Zerlegung von G. s sei eine Senke von G_1. Bildet man

$$G_1'=(V_1\setminus\{s\},\ E_1\setminus\{(u,v)\in E \mid u\in V_1, v=s\})$$

$$G_2'=(V_2\cup\{s\},\ E_2\cup\{(u,v)\in E \mid u=s, v\in V_2\}),$$

so ist G_1', G_2' eine neue Zerlegung mit $|E_2'|\leq|E_2|+k$.

Man starte mit $G_1=G$ und $G_2=(\emptyset,\emptyset)$ und wiederhole das obige Verfahren so lange, bis G_2 die gewünschte Größe hat. □

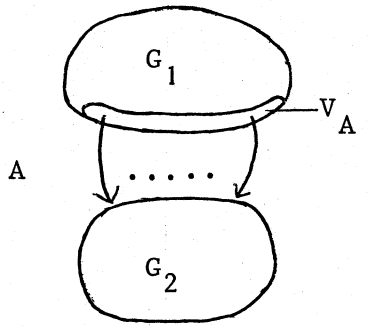
Satz 5.23: Jeder gerichtete zykelfreie Graph mit n Knoten und Ingrad k kann mit $O((k \log k)\cdot n/\log n)$ Marken und 2^n Zügen markiert werden.

Beweis: Auf einem Graphen mit n Knoten gibt es nur 2^n verschiedene Muster, in denen Marken auf den Knoten des Graphen liegen können. In jeder Strategie, die mehr als 2^n Züge hat, wiederholt sich ein Muster und die Strategie kann verkürzt werden. Daraus

folgt die Zeitschranke.

Sei $P_k(n)$ die kleinste Zahl von Marken, die hinreicht, um in jedem gerichteten azyklischen Graphen mit Ingrad k und n Kanten jeden beliebigen Knoten markieren zu können. Wir zeigen $P_k(n) \leq O((\log k) \cdot n/\log n)$. Daraus folgt der Satz, da jeder Graph mit n Knoten und Ingrad k höchstens kn Kanten hat.

Sei $G=(V,E)$ ein gerichteter azyklischer Graph mit Ingrad k und n Kanten, und $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$ sei eine Zerlegung nach Lemma 5.22. Es sei $A=\{(u,v) \in E \mid u \in V_1, v \in V_2\}$ und $V_A=\{u \in V_1 \mid \exists v \in V_2 : (u,v) \in E\}$



Lemma 5.24: $P_k(n) \leq \max\{P_k(n/2+k)+P_k(n/2-2n/\log n)+k, P_k(n/2+k)+2n/\log n\}$.

Beweis: Sei $v_o \in V$, und v_o soll markiert werden.

1. Fall: $v \in G_1$: Wegen $|E_1| \leq n/2$ genügen $P_k(n/2)$ Marken

2. Fall: $v \in G_2$ und $|A| \leq 2n/\log n$.
Man markiere in G_1 nacheinander alle Knoten in V_A. Jedesmal, wenn ein neuer Knoten in V_A markiert wurde, werden die Marken in G_1 entfernt, außer denen, die auf Knoten aus V_A liegen. Das geht mit $P_k(n/2)+|V_A| \leq P_k(n/2)+2n/\log n$ Marken. Dann markiere man v_o in G_2. Wegen $|V_2| \leq \frac{n}{2}+k$ geht das mit $P_k(\frac{n}{2}+k)$ Marken auf G_2 bzw. $P_k(\frac{n}{2}+k)+2n/\log n$ Marken auf G_2 und V_A.

3. Fall: $v \in G_2$ und $|A| \geq 2n/\log n$.
Man beginne eine Strategie S, die v_o in G_2 markiert und nur $P_k(\frac{n}{2}+k)$ Marken auf G_2 benutzt. Jedesmal, wenn hierbei eine Marke auf eine Quelle q von G_2 gesetzt werden soll, muß die Strategie S unterbrochen werden, und es werden zuerst nacheinander alle Vorgänger von q in G_1 markiert. Jedesmal, wenn ein neuer Vorgänger von q markiert wurde, werden alle Marken von G_1, die nicht auf Vorgängern von q liegen, entfernt. Sind alle Vorgänger

von q markiert, wird S markiert, alle Marken auf G_1 werden entfernt, und die Strategie S wird fortgesetzt.

Wegen $|A| \geq 2n/\log n$ ist $|E_1| \leq n/2 - 2n/\log n$. Auf G_1 kommt man deshalb mit $P_k(n/2-2n(\log n))+k$ Marken aus. □

Sei $n_o \in N$, so daß für alle $n \geq n_o$ gilt:

i) $\dfrac{n-2k \log n}{2 \log n(\log n-2)} \geq k$

ii) $n \geq 4k$

iii) $n/2-2n/\log n \geq n/4$

iv) $\log(n/2+k) \geq \frac{7}{8} \log n$.

Sei $C=\max\{\log n_o, 14\}$, also $C=O(\log k)$. Wir zeigen durch Induktion über $n: P_k(n) \leq C \cdot n/\log n$.

Wegen $C \geq \log n_o$ ist das sicher richtig für $n \leq n_o$. Um von $m<n$ auf n zu schließen, muß man zwei Fälle betrachten:

1. Fall: In Lemma 5.24 ist der erste Ausdruck das Maximum. Es folgt:

$$P_k(n) \leq C \frac{n/2+k}{\log(n/2+k)} + C \frac{n/2-2n/\log n}{\log(n/2-2n/\log n)} + k$$

$$\leq \frac{Cn/2}{\log n-1} + C \frac{n/2-2n/\log n}{\log n-2} + \frac{Ck}{\log n-2} + k$$

wegen iii). Wir benutzen

$$\frac{1}{x-a} = \frac{1}{x} + \frac{a}{x(x-a)}$$

und erhalten

$$P_k(n) \leq \frac{Cn}{2 \log n} + \frac{Cn}{2(\log n)(\log n-1)} + \frac{Cn}{2 \log n} + \frac{Cn}{\log n(\log n-2)}$$

$$- \frac{2Cn}{\log n(\log n-2)} + \frac{Ck}{\log n-2} + k$$

$$\leq \frac{Cn}{\log n} - \frac{Cn}{2 \log n(\log n-2)} + \frac{Ck}{\log n-2} + k \leq \frac{Cn}{\log n} \text{ wegen i).}$$

2. Fall: In Lemma 5.24 ist der zweite Ausdruck das Maximum. Es folgt:

$$P_k(n) \leq C \frac{n/2+k}{\log(n/2+k)} + \frac{2n}{\log n}$$

$$\leq \frac{3Cn}{4 \cdot \log(n/2+k)} + \frac{2n}{\log n} \text{ wegen ii)}$$

$$\leq \frac{6}{7} \frac{Cn}{\log n} + \frac{2n}{\log n} \text{ wegen iv)}$$

$$\leq \frac{Cn}{\log n}, \text{ da } C \geq 14. \quad \square$$

Durch sorgfältige Analyse kann C verbessert werden, aber nicht der Term n/log n [51].

Für späteren Gebrauch genügt uns nicht die Existenz effizienter Markierungsstrategien, sondern wir müssen diese auch, ohne zu viel Platz zu verwenden, finden können. Der Beweis von Lemma 5.24 enthält implizit ein rekursives Markierungsverfahren [51], aber die Konstruktion einer Turingmaschine, die dieses Verfahren ausführt, würde technisch einigen Aufwand erfordern. Wir wählen einen leichteren Weg.

Zunächst kümmern wir uns um die Repräsentation von Graphen. Sei $G=(V,E), V=(1,\ldots,n)$ ein gerichteter Graph mit n Knoten. Für $i \in V$ sei $A(i)=\{j \mid (j,i) \in E\}$ (Vorgängerliste von Knoten i). Den Graphen G kann man offenbar durch die Menge der Vorgängerlisten aller Knoten repräsentieren. Repräsentiert man die Knoten durch ihre Binärdarstellung, so hat diese Repräsentation r_G von Graph G die Länge $O(|E|\log|V|)$. Hat G Ingrad k, ist $|E| \leq k|V|$. Um einen mit Marken belegten Graphen zu repräsentieren, genügt ein zusätzliches Bit in der Vorgängerliste eines jeden Knotens i, das auf 1 gesetzt wird, wenn Knoten i eine Marke hat. Für zykelfreie Graphen G kann eine Markierungsstrategie als eine Folge von Anweisungen der Form $\mu=<1,\text{bin } i>$ (markiere Knoten i) oder $\mu=<0,\text{bin}(i)>$ (entferne die Marke auf Knoten i) aufgeschrieben werden.

Satz 5.25: Es gibt eine $O(n^2)$-bandbeschränkte deterministische Turingmaschine Q, die angesetzt auf $(r_G,\text{bin}(v),\text{bin}(i))$ nacheinander die Züge einer Strategie generiert, die Knoten v in G mit höchstens i Marken markiert, falls eine solche Strategie existiert (jedesmal wenn Q einen neuen Zug ausdruckt, löscht Q den vorherigen Zug).

Beweis: Sei Q' eine nichtdeterministische Maschine, die angesetzt auf Eingabe $(r_G, bin(v), bin(i), bin(j), \mu)$ mit r_G = Repräsentation eines mit Marken belegten Graphen G, μ ein Zug einer Markierungsstrategie, folgendes tut:

Q' notiert den Effekt von Zug μ auf die Belegung des Graphen mit Marken und spielt dann nichtdeterministisch weiter, aber höchstens für weitere j-1 Züge und so, daß höchstens i Marken gleichzeitig auf dem Graphen sind. Wird hierbei Knoten v markiert, hält Q' im akzeptierenden Endzustand, sonst hält Q' im verwerfenden Zustand. Man kann Q' leicht so konstruieren, daß Q' O(n)-bandbeschränkt ist, und es gilt:

$L(Q') = \{(r_G, bin(v), bin(i), bin(j), \mu) |$ es gibt eine Strategie mit erstem Zug μ und höchstens j Zügen, die Knoten v in Graph G ausgehend von der gegebenen Markierung markiert und höchstens i Marken benutzt$\}$.

Nach Satz 5.13 gibt es eine $O(n^2)$-bandbeschränkte deterministische Version Q" von Q'. Wir beschreiben eine Maschine Q, die das Gewünschte tut.

Sei m die Zahl der Knoten von G; $m \leq |r_G|$. Wenn Q schon die ersten s-1 Züge $\mu_1, \ldots, \mu_{s-1}$ der gewünschten Strategie generiert hat und dabei die Repräsentation des hierdurch mit Marken belegten Graphen r_G^{s-1} erzeugt hat, findet Q den s-ten Zug wie folgt:

Es gibt nur m mögliche s-te Züge μ. Für jeden dieser Züge testet Q, ob $(r_G^{s-1}, bin(v), bin(i), bin(2^m - s), \mu) \in L(Q)$ gilt; falls ja, wird μ ausgedruckt und r_G^s aus r_G^{s-1} gebildet, indem der Effekt von Zug μ auf die Markierung notiert wird. □

5.2.2. Berechnungsgraphen und Stenogramme

Im folgenden sei M immer eine beliebige, aber feste deterministische $C \cdot t(n)$-zeitbeschränkte k-Band-Turingmaschine mit Alphabet A und Zustandsmenge Z. Die Zellen der Bänder von M seien mit den ganzen Zahlen durchnumeriert und in Blöcke der Größe $\Delta \in \mathbb{N}$ partitioniert, so daß auf jedem Band Block b gerade die Zellen mit Nummern $\Delta b, \ldots, \Delta \cdot (b+1) - 1$ enthält. w sei eine Eingabe für M, $|w| = n$. Wir unterteilen die Rechnung von M angesetzt auf w nach der folgenden Vorschrift in Zeitintervalle:

Das erste Zeitintervall beginnt mit dem ersten Schritt. Für $i \geq 1$

und $j\in\{1,\ldots,k\}$ sei $p(w,\Delta,i,j)$ die Position des Kopfs auf Band j zu Beginn des Zeitintervalls i und

$$u(w,\Delta,i,j)=\{\lfloor p(w,\Delta,i,j)/\Delta\rfloor+q \mid q=-1,0,1\},$$

d.h. $u(w,\Delta,i,j)$ besteht aus der Nummer des Blocks, in dem sich der Kopf auf Band j zu Beginn von Zeitintervall i befindet, und den Nummern der angrenzenden Blöcke. Das i-te Zeitintervall endet und das i+1-te Zeitintervall beginnt, sobald einer der Köpfe j einen Block mit Nummer $\notin u(w,\Delta,i,j)$ betritt.

Jedes Zeitintervall dauert als Teil einer Rechnung von $C\cdot t(n)$ Schritten höchstens $C\cdot t(n)$ Schritte. Weil vom Beginn zum Ende eines Zeitintervalls mindestens ein Kopf einen ganzen Block durchläuft, dauert jedes Zeitintervall (außer evtl. dem letzten) mindestens Δ Schritte. Für die Anzahl m der auftretenden Zeitintervalle folgt $m\leq\lceil C\cdot t(n)/\Delta\rceil$.

Für $i\geq 1$ sei $p(w,\Delta,i)=(\text{bin}(p(w,\Delta,i,1)),\ldots,\text{bin}(p(w,\Delta,i,k)))$. Die Zeichenreihe $p(w,\Delta,1)\ldots p(w,\Delta,m)$ hat Länge höchstens $D\cdot t(n)\log t(n)/\Delta$ für ein passendes $D\in\mathbb{N}$ und heißt das <u>Δ-Stenogramm</u> der Rechnung von M angesetzt auf w.

Eine beliebige Folge

$$p=p_1\ldots p_r \text{ mit } p_i=(\text{bin}(p_{i1}),\ldots,\text{bin}(p_{ik})) \quad r,p_{ij}\in\mathbb{N}$$

heißt ein <u>potentielles Stenogramm</u>.

Es sei $B(w,\Delta,i,j)\in A^{3\Delta}$ die Inschrift der drei Blöcke auf Band j mit Nummern aus $u(w,\Delta,i,j)$ am Ende von Zeitintervall i und $z(w,\Delta,i)\in Z$ der Zustand von M am Ende von Zeitintervall i.

$\text{Res}(w,\Delta,i)=(z(w,\Delta,i),B(w,\Delta,i,1),\ldots,B(w,\Delta,i,k),\text{bin}(p(w,\Delta,i+1,1))$ $\ldots,\text{bin}(p(w,\Delta,i+1,k))$ heißt das Δ-<u>Resultat</u> von Zeitintervall i der Rechnung von M angesetzt auf w und hat Länge $O(\Delta+\log t(n))$.

Jedes $R=(z,B_1,\ldots,B_k,\text{bin } q_1,\ldots,\text{bin } q_k)$ mit $z\in Z, B_i\in A^*$ und $q_i\in\mathbb{Z}$ heißt ein <u>potentielles Resultat</u>.

Eine Zeichenreihe $(\text{bin}(i_1),\text{Res}(w,\Delta,i_1))\ldots(\text{bin}(i_s),\text{Res}(w,\Delta,i_s))$ heißt ein <u>Δ-Fragment</u> der Rechnung von M angesetzt auf w. Eine Zeichenreihe $(\text{bin}(i_1),R_{i_1})\ldots(\text{bin}(i_s),R_{i_s})$ heißt ein <u>potentielles Fragment</u>, wenn die R_{i_j} potentielle Resultate sind.

Für $\Delta\in\mathbb{N}$ und potentielle Stenogramme $p=p_1\ldots p_r$ ordnet man dem Paar (Δ,p) einen Graphen $G(\Delta,p)=(V(\Delta,p),E(\Delta,p))$ zu, der wie folgt defi-

niert wird:

$V(\Delta,p) = \{0,\ldots,r\}$.

Sei $u_{ij}=\{\lfloor p_{ij}/\Delta\rfloor+q \mid q=-1,0,1\}$.

Es gilt $(i',i)\in E(\Delta,p)$, wenn mindestens einer der folgenden Fälle vorliegt:

(1) $i'+1=i$

(2) $\exists j \exists b\in u_{ij}: i'=\max\{i''<i \mid b\in u_{i''j}\}$

(3) $i'=0$ und $\exists j \exists b\in u_{ij} \forall i''<i: b\notin u_{i''j}$.

$G(\Delta,p)$ hat Ingrad $3k+1$ und heißt potentieller Berechnungsgraph. Die Repräsentation von $G(\Delta,p)$ kann aus p auf Platz $O(|p|)$ und in Zeit $O(|p|^2)$ erzeugt werden. Ist p das Δ-Stenogramm einer Rechnung, so heißt $G(\Delta,p)$ der Δ-Berechnungsgraph von M angesetzt auf w. Bedingung (2) hat dann die Bedeutung: In Zeitintervall i wird auf Band j möglicherweise ein Block b besucht, der schon in der Umgebung $u(w,\Delta,i',j)$ war, und dessen Inschrift garantiert seit dem Ende von Zeitintervall i' nicht verändert wurde. (3) bedeutet, daß im Zeitintervall i auf Band j ein Block besucht wird, der vorher noch nicht besucht wurde.

Beispiel 5.26: Auf Seite 16o sind die Kopfbewegungen einer 2-Band-Turingmaschine als Funktion der Zeit und ein zugehöriger Berechnungsgraph aufgetragen.

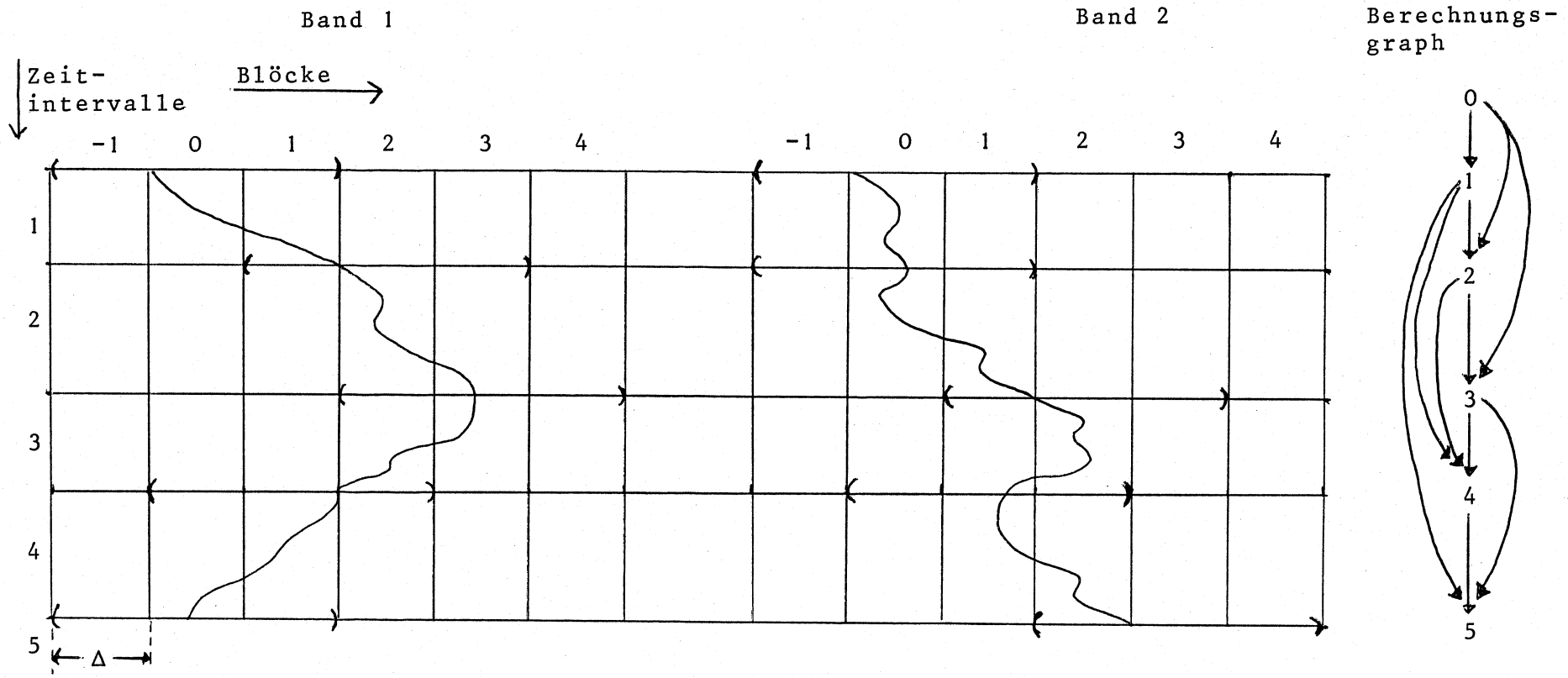
Das folgende Lemma gibt an, wie man aus Stenogrammen und Fragmenten von Rechnungen Resultate von gewissen Zeitintervallen berechnen kann.

Lemma 5.27: Sei $k\geq 2$ und M eine deterministische $C\cdot t(n)$-zeitbeschränkte k-Band-Turingmaschine. Dann gibt es eine deterministische k-Band-Turingmaschine R, für die gilt:

Ist $I=\{i' \mid i' \text{ ist Vorgänger von } i \text{ im } \Delta\text{-Berechnungsgraph von M angesetzt auf w}\}\subseteq\{i_1,\ldots,i_s\}$ und ist p das Δ-Stenogramm der Rechnung von M angesetzt auf w, so berechnet R angesetzt auf die Eingabe (w,E) mit

$$E=(\text{bin}(\Delta),p(w,\Delta,1)\ldots p(w,\Delta,i),(\text{bin}(i_1),\text{Res}(w,\Delta,i_1))\ldots$$
$$\ldots(\text{bin}(i_s),\text{Res}(w,\Delta,i_s)))$$

als Ergebnis $\text{Res}(w,\Delta,i)$. R braucht hierbei Band $O(|w|+|E|)$ und macht höchstens $O(t(|w|)+|E|)$ Schritte.



Kopfbewegungen einer 2-Band-Turingmaschine, angesetzt auf eine Eingabe w.
Zu Beginn eines jeden Zeitintervalls i ($1 \leq i \leq 5$) ist auf jedem Band j (j=1,2) die Umgebung u(w,Δ,i,j) eingetragen.

Beweis: Wir stellen uns die Felder eines jeden Bandes von R mit den ganzen Zahlen durchnumeriert vor. R tut das folgende:

1) Der Zustand z von M zu Beginn von Zeitintervall i wird bestimmt. z ist die erste Komponente von Res(w,Δ,i-1).

2) Auf den Feldern $-\Delta,\ldots,2\Delta-1$ eines jeden Bandes von R werden drei Blöcke mit Nummern $-1,0,1$ zu je Δ Feldern markiert.

3) Für alle $j\in\{1,\ldots,k\}$ werden die drei markierten Blöcke auf Band j von M mit der Inschrift zu Beginn von Zeitintervall i derjenigen drei Blöcke von Band j von M beschriftet, die M während Zeitintervall i erreichen kann. Für $q\in\{-1,0,1\}$ findet R die Inschrift des q-ten Blocks von Band j wie folgt:

   Sei $b=\lfloor p(w,\Delta,i,j)/\Delta\rfloor+q$ und

   $$i' = \begin{cases} \max\{i''<i \mid b\in u(w,\Delta,i'',j)\} & \text{falls dieses existiert} \\ 0 & \text{sonst} \end{cases}$$

   Ist $i'\neq 0$, so ist $b=\lfloor p(w,\Delta,i',j)/\Delta\rfloor+q'$ für ein $q'\in\{-1,0,1\}$ und Block q von Band j von R wird mit den ersten oder zweiten oder dritten Δ Zeichen von B(w,Δ,i',j) beschriftet, je nachdem, ob $q'=-1$ oder $q'=0$ oder $q'=1$ gilt.

   Ist $i'=0$ und ist $j\neq 1$ oder $\Delta\cdot b\geq|w|$ oder $b<0$, so bleibt Block q von Band j von R leer. Ist $i=0$ und $j=1$ und $0\leq\Delta\cdot b<|w|$, so wird Block q von Band j von R beschriftet mit:

   $$\begin{cases} w_{\Delta\cdot b+1}\cdots w_{\Delta\cdot(b+1)} & \text{falls } \Delta(b+1)\leq|w| \\ w_{\Delta\cdot b+1}\cdots w_{|w|}B\ldots B & \text{sonst.} \end{cases}$$

4) Für alle $j\in\{1\ldots k\}$ wird der Kopf auf Band j von R an diejenige Stelle der Inschrift gebracht, auf der der Kopf auf Band j von M zu Beginn des i-ten Zeitintervalls steht. Dies ist das Feld mit Nummer Rest(p(w,Δ,i,j),Δ).

5) R simuliert M Schritt für Schritt, bis einer der Köpfe die markierten Blöcke verläßt.

6) Am Ende dieser Simulation ist der Zustand von M gleich z(w,Δ,i). Für alle $j\in\{1,\ldots,k\}$ ist die Inschrift der Blöcke $-1,0,1$ von Band j von R gleich B(w,Δ,i,j). Steht der Kopf auf Band j von R auf Feld r_j, so ist $p(w,\Delta,i+1,j)=\Delta\cdot\lfloor p(w,\Delta,i,j)/\Delta\rfloor+r_j$.
   R kommt offenbar mit Speicherplatz proportional zur Länge der

Eingabe aus.

Zur Ausführung von 1) bis 4 und 6) muß beschränkt oft folgendes getan werden:

i) Absuchen der Eingabe in Zeit $O(|w|+|E|)$.

ii) Multiplikation/Division von Zahlen $\leq C\cdot t(|w|)$. Die Binärdarstellung dieser Zahlen hat die Länge $O(\log t(|w|))$; die Multiplikationen/Divisionen können daher nach der Schulmethode in Zeit $O((\log t(|w|))^2)=o(t(|w|))$ durchgeführt werden.

iii) Zählen bis höchstens $\Delta\leq C\cdot t(|w|)$ auf einem Band und gleichzeitig Bewegung eines Kopfs auf einem anderen Band. Dies ist nach dem Beweis von Lemma 4.11 bzw. nach Lemma 4.12 in $O(t(|w|)$ Schritten möglich.

Teil 5 ist ein Teil einer Rechnung der Länge höchstens $C\cdot t(|w|)$, dauert also nicht länger als $C\cdot t(|w|)$ Schritte.

Man erhält als Gesamtlaufzeit $O(|E|+t(|w|))$. □

Das entscheidende technische Resultat dieses Abschnitts ist

Satz 5.28: Sei $k\geq 2$ und M eine deterministische $C\cdot t(n)$-zeitbeschränkte k-Bandmaschine. Dann gibt es eine deterministische k-Bandmaschine Q, die eine Eingabe $(r_G,w,bin(\Delta))$ mit $G=(V,E)$, $V=\{0,\ldots,m-1\}$ genau dann akzeptiert, wenn G der Δ-Berechnungsgraph der Rechnung von M angesetzt auf w ist und M Eingabe w akzeptiert.

Ist $|w|=n$, so benutzt Q hierfür Band

$$O(|r_G|^2+n+(\Delta+\log t(n))m/\log m + t(n)\log t(n)/\Delta)$$

und Zeit

$$O(2^{O(|r_G|^2)}+2^m(t(n)+(\Delta+\log t(n))m/\log m))+t(n)\log t(n)/\Delta).$$

Beweis: Ist Ingrad $G>3k+1$ oder ist $(i,i+1)\notin E$ für irgendein $i\in\{0,\ldots,m-2\}$ oder ist eine Kante $(i,j)\in E$ mit $i\geq j$, verwirft Q sofort. Andernfalls ist G zykelfrei und hat Ingrad höchstens $3k+1$, kann also nach Satz 5.23 mit $O(m/\log m)$ Marken markiert werden. Sei $\mu_1,\ldots,\mu_T$, $T\leq 2^m$, eine Markierungsstrategie, die dies erreicht, und sei $V_x=\{v\in V|v \text{ hat eine Marke nach Zug } \mu_x\}$, $W_x=\bigcup_{y\leq x} V_y$ und $h(x)=\max\{v\in W_x\}$.

Q generiert eine Folge $F_0,F_1\ldots$ von potentiellen Fragmenten F_x und

eine Folge $p^0,p^1,\ldots$ von potentiellen Stenogrammen p^x. Es wird $F_o=\emptyset$ und $p^o=(0,\ldots,0)$ gesetzt, und die Berechnung von F_x und p^x aus F_{x-1} und p^{x-1} wird so organisiert, daß für alle x gilt:

(*) Wird mit der Berechnung von F_x begonnen, so gilt:

a) F_{x-1} besteht aus allen Einträgen $(bin(i),Res(w,\Delta,i))$ mit $i\in V_x$;

b) $p^{x-1}=p(w,\Delta,1)\ldots p(w,\Delta,h(x-1)+1)$;

c) für alle $u\in\{0,\ldots,m\}$ und $v\in\{0,\ldots,h(x-1)+1\}$ gilt $(u,v)\in E$ genau dann, wenn (u,v) eine Kante im Δ-Berechnungsgraph von M angesetzt auf w ist.

(*) gilt sicher am Anfang der Rechnung. Wir geben an, wie F_x und p^x berechnet werden.

Fall i): Zug μ_x markiert Knoten i:

Sei $p^{x-1}=p_1\ldots p_\ell$ und

$$F_{x-1}=(bin(i_1),Res(w,\Delta,i_1))\ldots(bin(i_s),Res(w,\Delta i_s)).$$

Q berechnet aus der Zeichenreihe $(w,bin(\Delta),p_1\ldots p_i,F_{x-1})$ nach dem Verfahren von Lemma 5.27 ein potentielles Resultat

$$R_i=(z,B_1,\ldots,B_k,bin(q_1),\ldots,bin(q_k)).$$

Wird während dieser Simulation festgestellt, daß M akzeptiert bzw. verwirft, akzeptiert bzw. verwirft Q. Q setzt

$$p^x=\begin{cases} p^{x-1}(bin(q_1),\ldots,bin(q_k)) & \text{falls } i=h(x-1)+1 \\ p^{x-1} & \text{sonst}\end{cases}$$

und $F^x=F^{x-1}(bin(i),R_i)$.

Wurde p^{x-1} zu p^x verlängert, wird die Repräsentation von $G(\Delta,p^x)$ erzeugt und getestet, ob die Menge der Vorgänger von Knoten i+1 in $G(\Delta,p^x)$ gleich der Menge der Vorgänger von Knoten i+1 in G ist. Falls ja, wird mit der Berechnung von F_{x+1} und p^{x+1} begonnen, andernfalls wird verworfen.

Fall ii): Wird in Zug M_x eine Marke von Knoten i entfernt, löscht Q aus F^x den Eintrag $(bin(i),R_i)$ und schiebt die Bandinschrift zusammen, falls durch das Löschen eine Lücke

entstanden ist. $p^x=p^{x-1}$.

Wird F_T erreicht und hat M bis dahin weder akzeptiert noch verworfen, verwirft Q. Wir zeigen, daß (*) für x gilt, falls (*) für x-1 gilt.

Fall i): In Zug μ_x wird Knoten i markiert.

Wegen $(i-1,i)\in E$ und nach Regel i) des Markenspiels muß i-1 markiert sein, also $i-1\leq h(x-1)$, bzw. $i\leq h(x-1)+1$.

Also gilt nach Induktionsvoraussetzung c): $I=\{i'|(i',i)\in E, i'\neq 0\}=\{i'\neq 0$ und i' ist Vorgänger von i im Δ-Berechnungsgraph von M angesetzt auf w$\}$. Nach Regel ii) des Markenspiels sind die Knoten aus I alle markiert, also gilt $I\subseteq\{i_1,\ldots,i_s\}$ nach Induktionsvoraussetzung a).

Nach Induktionsvoraussetzung b) ist $p^{x-1}=p(w,\Delta,1)\ldots p(w,\Delta,i)$ oder eine Verlängerung hiervon.

Also berechnet Q $R_i=Res(w,\Delta,i)$. Insbesondere folgt $q_j=p(w,\Delta,i+1,j)$ für $1\leq j\leq k$ und damit $p^x=p(w,\Delta,1)\ldots p(w,\Delta,i+1)$, falls $i=h(x-1)+1$.

Damit sind a) und b) gezeigt. Ist $i<h(x-1)+1$, ist für c) nichts zu zeigen. Ist $i=h(x-1)+1$, muß c) noch für $v=h(x)+1=i+1$ gezeigt werden. Genau die Gültigkeit hiervon wird jedoch von Q getestet, bevor mit der Berechnung von F_{x+1} begonnen wird.

Fall ii): In Zug μ_x wird eine Marke von Knoten i entfernt. Dieser Fall ist wegen $h(x)=h(x-1)$ trivial.

Damit ist die Korrektheit von Q bewiesen.

Analyse von Speicherplatz und Rechenzeit:

Der anfängliche Test von G ist auf Platz $|r_G|$ und in Zeit $O(|r_G|^2)$ durchführbar. Das Finden der Markierungsstrategie ist nach Satz 5.25 auf Band $O(|r_G|^2)$ und damit nach Satz 4.3 e) in Zeit $2^{O(|r_G|^2)}$ möglich.

Speichern eines Fragments F_x erfordert höchstens Platz $O((\Delta+\log t(n))\cdot|V_x|)=O((\Delta+\log t(n))m/\log m)$. p^x hat höchstens

Länge $O(t(n)\log t(n)/\Delta)$. Hieraus folgt die Schranke für den Bandverbrauch.

Das Verfahren aus Lemma 5.27 wird nur mit Eingaben der Länge $O(n+(\Delta+\log t(n))m/\log m+t(n)\log t(n)/\Delta)$ durchgeführt. Daher kostet die Berechnung eines Resultats $Res(w,\Delta,i)$ nach Lemma 5.27 höchstens $O(t(n)+(\Delta+\log t(n))m/\log m+t(n)\log t(n)/\Delta)$ Schritte. Dies muß höchstens 2^m mal wiederholt werden.

Höchstens m mal wird p^{x-1} zu p^x verlängert. Das Erzeugen der Repräsentation von $G(\Delta,p^x)$ aus p^x und der Vergleich der Vorgängerlisten von Knoten i+1 in G und $G(\Delta,p^x)$ ist in Zeit $O(|r_G|^2)$ möglich. Hier werden zusätzliche $O(m\cdot|r_G|^2)\leq O(|r_G|^3)=o(2^{O(|r_G|^2)})$ Schritte gemacht. □

5.2.3. Platzeffiziente Simulationen und Zeithierarchien über festem Band

Satz 5.29: Sei $k\geq 2$, $t,d:\mathbb{N}\to\mathbb{N}$, $d(n)\leq O(\log t(n))$, $\lim d(n)=\infty$, $t(n)/d(n)\geq n$ für alle n, und es gebe eine simultan $O(t(n))$-zeitbeschränkte und $O(t(n)/d(n))$-bandbeschränkte k-Bandmaschine, die angesetzt auf Eingabe w die Binärdarstellungen von $t(|w|)$ und $d(|w|)$ erzeugt, dann ist

$$DTIME_k(t(n))\subseteq C_k(t(n)/d(n),t(n)\cdot 2^{2^{2d(n)}}).$$

Beweis: Nach Definition der Klassen $C_k(s(n),t(n))$ genügt es, die Behauptung für $d(n)\leq\frac{1}{2}\log t(n)$ zu zeigen.

Sei M eine $C\cdot t(n)$-zeitbeschränkte k-Bandmaschine, w eine Eingabe für M und $|w|=n$. Für $\Delta(n)=C\cdot t(n)d^2(n)/2^{d(n)}$ folgt: die Berechnung von M angesetzt auf w hat $m\leq 2^{d(n)}/d^2(n)+1$ Zeitintervalle. Aus $\Delta(n)\geq\frac{C}{4}\cdot t(n)(\log t(n))^2/\sqrt{t(n)}$ folgt $\Delta(n)+\log t(n)=O(\Delta(n))$, $\Delta(n)m/\log m$ $O(t(n)/d(n))$ und $t(n)\log t(n)/\Delta(n)\leq O(\sqrt{t(n)}/\log t(n))$. Die Repräsentation r_G des $\Delta(n)$-Berechnungsgraphen G von M angesetzt auf w hat Länge $|r_G|\leq D\cdot 2^{d(n)}/d(n)$ für eine passende (von n unabhängige) Konstante D.

Wir beschreiben eine k-Bandmaschine Q, die M simuliert. Angesetzt auf Eingabe w mit $|w|=n$ zählt Q systematisch alle Repräsentationen r_G irgendwelcher Graphen mit $|r_G|\leq D2^{d(n)}/d(n)$ auf, testet für jedes r_G, ob r_G die Repräsentation des $\Delta(n)$-Berechnungsgraphen von M angesetzt auf w ist und falls ja, ob M w akzeptiert. Ist dies

für irgendein r_G der Fall, akzeptiert Q. Da der $\Delta(n)$-Berechnungsgraph sicher mit aufgezählt wird, folgt $L(Q)=L(M)$.

Jeder Test kann nach Satz 5.28 auf Band

$O(n+2^{2d(n)}/d^2(n)+t(n)/d(n)+\sqrt{t(n)}/\log t(n))=O(t(n)/d(n))$

durchgeführt werden. Da alle Tests nacheinander auf dem gleichen Platz durchgeführt werden können, folgt die Platzschranke des Satzes.

Die Durchführung eines Tests geht nach Satz 5.28 in Zeit $O(2^{O(2^{2d(n)}/d^2(n))}+(2^{2^{d(n)}/d^2(n)})t(n))$. Die Durchführung aller Tests liefert für die Zeit einen Faktor $2^{O(2^{d(n)}/d(n))}$ = Anzahl von Repräsentationen von Graphen mit Länge $O(2^{d(n)}/d(n))$. Damit erhält man insgesamt als Schranke für den Zeitverbrauch

$O(2^{O(2^{2d(n)}/d^2(n)+2^{d(n)}/d(n))}+(2^{2^{d(n)}/d^2(n)+O(2^{d(n)}/d(n)}).t(n))=$

$O(2^{O(2^{2d(n)}/d^2(n))}+2^{O(2^{d(n)}/d(n))}t(n))\leq O(2^{2^{2d(n)}}\cdot t(n))$. □

Korollar 5.3o: Ist $t(n)/\log t(n)$ bandkonstruierbar und ist $t(n)/\log t(n)=\Omega(n)$, so ist $DTIME(t(n))\subseteq DTAPE(t(n)/\log t(n))$.

Mit einem Trick ähnlich wie im Beweis von Satz 3.19 kann man hier die Voraussetzung über die Gutartigkeit der Funktion t(n) umgehen.

Satz 5.31: Ist $t(n)/\log t(n)=\Omega(n)$, so ist

$$DTIME(t(n))\subseteq DTAPE(t(n)/\log t(n)).$$

Beweis: M sei eine $C\cdot t(n)$-zeitbeschränkte k-Bandmaschine. Wir beschreiben eine Maschine Q; w sei eine Eingabe. Q arbeitet in mehreren Stufen und beginnt in Stufe $|w|$. In Stufe t wird folgendes getan:

Es wird so getan, als sei die Laufzeit von M angesetzt auf w gleich t, und die Simulation aus dem Beweis von Satz 5.28 wird durchgeführt mit $d=\frac{1}{2}\log t$ und $\Delta=\sqrt{t}(\log t)^2$. Dabei wird Platz $O(t/\log t)$ benutzt. Gelingt es in Stufe t nicht, M angesetzt auf w zu Ende zu simulieren, wird zu Stufe t+1 übergegangen.

Da M $C\cdot t(n)$-zeitbeschränkt ist, gelingt die Simulation spätestens in Stufe $C\cdot t(n)$. □

Die Voraussetzung über das Wachstum der Funktion t(n) kann man ebenfalls umgehen, indem man zu Off-line-Maschinen übergeht:

Satz 5.32: $DTIME(t(n)) \subseteq DTAPE(t(n)/\log t(n))$.

Beweis: Man modifiziert für Off-line-Maschinen Lemma 5.27 wie folgt: Die Eingabe (w,E) wird auf zwei Bändern eingegeben: w auf dem Eingabeband und E auf dem ersten Arbeitsband; man erhält als Bandverbrauch auf den Arbeitsbändern $O(|E|+\log(|w|))$. Ebenso wird Satz 5.28 abgeändert, indem w auf dem Eingabeband und $(r_G, bin(\Delta))$ auf dem ersten Arbeitsband eingegeben wird; man erhält hier als Bandverbrauch auf den Arbeitsbändern

$$O(|r_G|^2+\log n+(\Delta+\log t(n))m/\log m+t(n)\log t(n)/\Delta).$$

Der Rest des Beweises verläuft genau nach dem Muster der Beweise von Satz 5.29 und Satz 5.3o. □

Korollar 5.33: Ist s(n) bandkonstruierbar, $\lim t(n)/(s(n)\log t(n)) = 0$ so ist $DTIME(t(n)) \subsetneq DTAPE(s(n))$.

Beweis: $DTIME(t(n)) \subseteq DTAPE(t(n)/\log t(n)) \subsetneq DTAPE(s(n))$ nach Satz 4.31 b). □

Korollar 5.34: Ist s(n) bandkonstruierbar, so ist $DTIME(s(n)) \subsetneq DTAPE(s(n))$.

Dies beantwortet Frage i), die wir zu Beginn dieses Abschnitts gestellt hatten. Der folgende Satz gibt eine Antwort auf Frage ii):

Satz 5.35: Ist $k \geq 2$, $s(n)=\Omega(n)$ und ist s(n) auf k-Bandmaschinen zeitkonstruierbar, so gibt es eine Folge von Funktionen $t_i: \mathbb{N} \to \mathbb{N}, i=0,1,\ldots$, so daß

$$C_k(s(n),t_i(n)) \subseteq C_k(s(n),t_{i+1}(n)) \quad i=0,1,\ldots .$$

Beweis: Sei $t_o(n)=s(n)$

$$f_j(n) = t_i(n)=s(n)2^{\cdot^{\cdot^{\cdot^{2}}}\Big\}\log^* s(n)}\Big\} 4i \quad .$$

Aus der Definition der Funktion log*( ) folgert man leicht

$$2^{\cdot^{\cdot^{\cdot^{2}}}\Big\}\log^* s(n)}\Big\} j = o(\log s(n)) \text{ für alle } j,$$

und damit

$$\log t_i(n) = \log s(n) + o(\log s(n)).$$

Also

$$C_k(s(n),t_i(n)) \subseteq DTIME_k(t_i(n))$$
$$\subsetneq DTIME_k(t_i(n)(\log^* s(n))^2) \text{ nach Satz 4.18}$$

$$\subseteq \mathrm{DTIME}_k(s(n)\cdot 2^{\cdot^{\cdot^{2}}\}\,4i+1}) \quad \text{(log*s(n) Zweier)}$$

$$\subseteq C_k(s(n), s(n)2^{2^{2\cdot 2^{\cdot^{\cdot^{\cdot}}}}}) \quad \text{(log*s(n) Zweier, } 4i+1 \text{ Höhe)} \quad \text{nach Satz 5.29}$$

$$\subseteq C_k(s(n), s(n)2^{\cdot^{\cdot^{\cdot^{2}}\}\,4i+4}}) = C_k(s(n), t_{i+1}(n)). \quad \square$$

Offene Fragen

Wir haben mit beträchtlichem Aufwand einige Ergebnisse über Beziehungen zwischen Speicherplatz und Rechenzeit hergeleitet. Verglichen mit den Lücken, die in unserem Wissen über diese Probleme klaffen, nehmen sich diese Ergebnisse sehr bescheiden aus:

a) Entweder kann Satz 5.32 verbessert werden oder es gilt $\mathrm{DTIME}(n \log n) \setminus \mathrm{DTAPE}(n) \neq \emptyset$, aber nicht einmal

b) $\mathrm{NTIME}(2^{O(n)}) \setminus \mathrm{DTAPE}(n) \neq \emptyset$? oder

$\bigcup_k \mathrm{NTIME}(n^k) \setminus \mathrm{DTAPE}(\log n) \neq \emptyset$? ist bekannt.

Da Satz 5.23 nicht verbessert werden kann [51], sind zum Verbessern von Satz 5.32 neue Techniken nötig.

c) Entweder kann Satz 4.3 e) verbessert werden oder es gilt $\mathrm{DTAPE}(n) \setminus \mathrm{DTIME}(2^{c \cdot n}) \neq \emptyset$ für irgendein c. Aber nicht einmal
$\mathrm{NTAPE}(n) \setminus \mathrm{DTIME}(n \log n) \neq \emptyset$? oder
$\mathrm{NTAPE}(n) \setminus \mathrm{NTIME}(n) \neq \emptyset$?
ist bekannt.

d) Gilt ein ähnlicher Satz wie Satz 5.32 für Registermaschinen?

e) Gilt Satz 5.35 für die Klassen C(s(n),t(n)) ?

f) Gilt $C(n, n^i) \subsetneq C(n, n^{i+1})$ für i=1,2,... ?

g) Jeder gerichtete zykelfreie Graph G_n mit n Knoten und Ingrad k kann mit O(n/log n) Marken in $\binom{n}{O(n/\log n)} = 2^{O(n/\log n)}$ Zügen markiert werden. Das Markieren gewisser Familien von Graphen G_n vom Ingrad 2 mit O(n/log n) Marken erfordert mindestens $\Omega \cdot (n \log \log n)$ Züge [53]. Was ist die minimale Zahl von Zügen, mit der jeder Graph mit n Knoten und Ingrad k mit O(n/log n) Marken markiert werden kann?

Übungen

1. a) Wir betrachten gerichtete binäre Räume, wo alle Kanten in Richtung der Wurzel orientiert sind.

   Man zeige: In jedem solchen Baum mit n Blättern kann die Wurzel mit $\log_2 n+2$ Marken markiert werden.

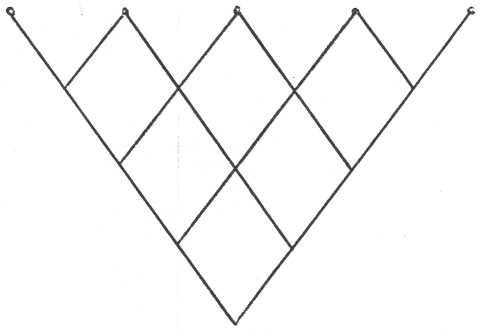
   b) Die Tiefe eines Knotens v ist die Anzahl der Kanten auf dem längsten Pfad von einer Quelle zu v.

   Man zeige: Jeder Knoten mit Tiefe d in einem gerichteten zykelfreien Graph mit Ingrad k kann mit $d(k-1)+2$ Marken markiert werden.

2. a) Ist B ein vollständiger binärer Baum mit 2^k Blättern, so sind mindestens k Marken nötig, um die Wurzel zu markieren.

   Hinweis: Ein Pfad heißt offen, wenn er keine Marke enthält. Man zähle die Marken auf dem Graph nach dem ersten Zug, nach dem kein Pfad von einem Blatt zur Wurzel mehr offen ist.

   b) Der folgende Graph heißt 5-Pyramide



   Es ist naheliegend, was eine m-Pyramide ist. Markieren der Wurzel einer m-Pyramide erfordert m Marken.

3. In [12] wird gezeigt: In jedem Graph $G=(V,E)$ mit n Kanten gibt es eine Menge $E'\subset E, |E'|\leq O(n \log\log n/\log n)$, so daß in $G'=(V,E\setminus E')$ kein Pfad eine Länge größer als $O(n/\log n)$ hat. Man folgere hieraus:

   Jeder Knoten in jedem gerichteten zykelfreien Graph mit n Knoten und Ingrad k kann mit $O(n \log\log n/\log n)$ Marken markiert werden.

4. Ein Graph heißt planar, falls man ihn ohne Überkreuzung von

Kanten in der Ebene einbetten kann. In [38] wird gezeigt: Ist $G=(V,E)$ ein planarer Graph, so gibt es eine Partition V_1, V_2, V_3 von V, so daß $\frac{1}{3}|V| \leq |V_i| \leq \frac{2}{3}|V|$ für $i=1,2$ und $|V_3| \leq O(\sqrt{|V|})$ gilt, und es gibt keine Kante zwischen V_1 und V_2.

a) Man folgere hieraus:
Jeder gerichtete zykelfreie planare Graph mit n Knoten und Ingrad k kann mit $O(\sqrt{n})$ Marken markiert werden.

b) Hinweis: $DTIME_1(t(n)) \subseteq DTAPE(t^{2/3}(n))$.
Die Berechnungsgraphen von 1-Band-Turingmaschinen sind planar.

5. M sei eine 1-Band-Turingmaschine, w eine Eingabe für M, $|w|=n$.

Ein CS-Stenogramm der Rechnung von M angesetzt auf w ist eine Zeichenreihe

$$(bin(i_1),CS(w,i_1))\ldots(bin(i_s),CS(w,i_s)),$$

$CS(w,i_j)$=Crossingsequenz der Rechnung von M angesetzt auf w an der Stelle i_j.

a) Ist M $O(t(n))$-zeitbeschränkt, so gibt es ein CS-Stenogramm der Länge $O(\sqrt{t(n)}\log t(n))$ von der Rechnung von M angesetzt auf w. (Man unterteile das Band in Blöcke und nehme aus jedem Block die kürzeste Crossingsequenz.)

b) Ist $\sqrt{t(n)}$ bandkonstruierbar, so ist $DTIME_1(t(n)) \subseteq DTAPE(\sqrt{t(n)}\log t(n))$.

c) Man zeige b) ohne die Voraussetzung über die Bandkonstruierbarkeit von $\sqrt{t(n)}$.

d) $DTIME_1(t(n)) \subseteq DTAPE(\sqrt{t(n)})$.

Hinweis: Man benutze zusätzlich Übung 4 aus Kapitel 4.3.

In [47] wird $NTIME_1(t(n)) \subseteq DTAPE(\sqrt{t(n)})$ gezeigt.

5.3. Grammatiken

Formale Grammatiken sind ein Hilfsmittel, mit dem man formale Sprachen definieren kann. Sie wurden ursprünglich studiert, weil man hoffte, auf diese Art z.B. eine formale Definition natürlicher Sprachen geben zu können. Das ist bisher nicht gelungen; jedoch sind viele Programmiersprachen teilweise oder sogar voll-

ständig durch formale Grammatiken beschrieben worden.

5.3.1. Chomsky-Grammatiken

Definition 5.36: Eine Chomsky-Grammatik G wird beschrieben durch ein 4-Tupel G=(A,T,P,s), wobei gilt:

A ist eine endliche Menge (Alphabet)
$T \subset A$ (Terminalalphabet)
P ist eine endliche Teilmenge von $(A \setminus T)^* \times A^*$ (Produktionensystem)
$s \in A \setminus T$ Startsymbol
$A \setminus T$ heißt Nichtterminalalphabet.

Definition 5.37: Für $w,w' \in A^*$ sagen wir: w' ist in G direkt aus w ableitbar ($w \underset{G}{\rightarrow} w'$), falls es Worte $u,v,x,y \in A^*$ gibt, so daß $w=uxv, w'=uyv$ und $(x,y) \in P$ gilt.

Die Elemente von P werden auch Produktionen bzw. Produktionsregeln genannt. Eine Produktion $(x,y) \in P$ hat also die Bedeutung, daß man beim Ableiten in einer Zeichenreihe die Teilzeichenreihe x durch die Zeichenreihe y ersetzen darf.

Definition 5.38: Eine Folge von Worten $(w_1,\ldots,w_s), w_i \in A^*$, heißt Ableitung von w_s aus w_1 in G, falls für alle $i \in \{1,\ldots,s-1\}$ gilt: $w_i \underset{G}{\rightarrow} w_{i+1}$. w' heißt in G aus w ableitbar ($w \underset{G}{\rightarrow}^* w'$), falls es eine Ableitung $(w_1,\ldots,w_s)$ mit $w=w_1, w_s=w'$ gibt. Wenn klar ist, welche Grammatik gemeint ist, schreiben wir statt $w \underset{G}{\rightarrow} w'$ bzw. $w \underset{G}{\rightarrow}^* w'$ einfach $w \rightarrow w'$ bzw. $w \rightarrow^* w'$.

$L(G)=\{w \mid w \in T^*, s \underset{G}{\rightarrow}^* w\}$ heißt die von der Grammatik G erzeugte Sprache. $L \subseteq A^*$ heißt Chomsky-Sprache, falls es eine Chomsky-Grammatik G mit L=L(G) gibt.

Beispiel 5.39: $G=(A,T,P,s)$, $A=\{s,t,a,b\}$, $T=\{a,b\}$,
$P=\{(s,as),(s,at),(t,bt),(t,b)\}$
$L(G)=\{a^m b^n \mid m,n \in \{1,2,\ldots\}\}$.

Beispiel 5.4o: $G=(A,T,P,s)$,
$A=\{s,0,1,x\}$, $T=\{0,1,x\}$
$P=\{(s,0s0), (s,1s1), (s,x)\}$
$L(G)=\{w_1 \ldots w_r x w_r \ldots w_1 \mid w_i \in \{0,1\}, r \in \mathbb{N}\}$

Für "$(u,v) \in P$" wird auch die Schreibweise "$u \rightarrow v$" verwandt. Für "$(u,v_1),\ldots,(u,v_s) \in P$" schreibt man auch "$u \rightarrow v_1 \mid v_2 \mid \ldots \mid v_s$". Man beschreibt also etwa das Produktionensystem aus Beispiel 5.4o auch

durch "s→0s0|1s1|x". Man klassifiziert Grammatiken nach der Form ihrer Produktionensysteme.

Definition 5.41: Eine Chomsky-Grammatik heißt vom

- Typ 3 bzw. regulär, falls alle Produktionen von der Form u→av oder u→a mit u,v∈A\T,a∈T sind
- Typ 2 bzw. kontextfrei, falls für alle Produktionen u→v gilt: u∈A\T,|v|≥1
- Typ 1 bzw. kontextsensitiv, falls für alle Produktionen u→v gilt: |u|≤|v|
- Typ 0 immer.

Eine Chomskysprache heißt regulär bzw. kontextfrei bzw. kontextsensitiv, wenn sie von einer regulären bzw. kontextfreien bzw. kontextsensitiven Grammatik erzeugt wird. Beispielsweise ist die Grammatik aus Beispiel 5.39 regulär, Die Grammatik aus Beispiel 5.4o ist kontextfrei. Das folgende Beispiel deutet an, wie man kontextfreie Grammatiken zur Definition von Programmiersprachen verwenden kann.

Beispiel 5.42: G=(A,T,P,s)
A={s,(,),+,-,·,:,y,0,1,v,v'}
T={(,),+,-,·,1,y,0,1}
v→yv'|v'
v'→0v'|1v'|0|1
s→(s+s)|(s-s)|(s·s)|(s:s)|v.

L(G) besteht aus allen vollständig geklammerten arithmetischen Ausdrücken wie etwa (((y0+y1)·y01)+(1000-y1)).

Die Programmiersprache ALGOL ist zum großen Teil durch eine kontextfreie Grammatik beschrieben. Formal ist eine Programmiersprache eine gewisse Menge korrekter Programme. Man will nun gerne mit möglichst geringem Aufwand testen können, ob irgendeine Zeichenreihe ein korrektes Programm der Programmiersprache ist. Dies motiviert die Untersuchung der Komplexität von Sprachen, die durch Grammatiken erzeugt werden.

5.3.2. Die Komplexität kontextsensitiver Sprachen

Alle in Definition 5.41 aufgeführten Sprachklassen können durch Klassen von Maschinen, die sie akzeptieren, charakterisiert werden

[32]. Wir führen dies stellvertretend für die kontextsensitiven Sprachen durch.

Satz 5.43: Ist L kontextsensitiv, so ist $L \in NTAPE(n)$.

Beweis: $G=(A,T,P,s)$ sei eine kontextsensitive Grammatik. Wir beschreiben eine nichtdeterministische 3-Band-Turingmaschine M. w sei eine Eingabe für M. M arbeitet in mehreren Stufen:

In Stufe 0 wird s auf Band 2 geschrieben. Für $t \geq 1$ wird in Stufe t folgendes getan:

Sei u_{t-1} die Inschrift von Band 2 nach Stufe t-1. M rät nichtdeterministisch auf Band 3 irgendeine Zeichenreihe $u_t \in A^*$, dann testet M, ob $u_{t-1} \rightarrow u_t$ gilt.

Gilt $u_{t-1} \rightarrow u_t$ nicht, verwirft M;

gilt $u_{t-1} \rightarrow u_t$ und $u_t = w$, akzeptiert M;

gilt $u_{t-1} \rightarrow u_t$ und $u_t \neq w$, kopiert M die Inschrift von Band 3 auf Band 2 und geht in Stufe t+1.

Akzeptiert M die Eingabe w in Stufe t, so ist $(t, u_1, \ldots, u_t)$ offenbar eine Ableitung von w aus t. Ist andererseits $w \in L(G)$, so gibt es eine solche Ableitung; da M diese Ableitung raten kann, akzeptiert M die Eingabe w. Aus der Definition kontextsensitiver Grammatiken folgt $|u_i| \leq |w|$ für alle i. Also ist M auch O(n)-bandbeschränkt. □

Die entscheidende Beobachtung für die Umkehrung von Satz 5.43 ist, daß eine Produktion der Form za→az zu Ableitungen wie zaaa→azaa→aaza→aaaz führt. Das ist aber mit etwas Phantasie schon die Kopfbewegung einer Turingmaschine.

Satz 5.44: Ist $L \in NTAPE(n)$, $L \subseteq A^*$ und $B \notin A$, so ist die Sprache $\{BwB \mid w \in L\}$ kontextsensitiv.

Beweis: Sei Q eine nichtdeterministische $C \cdot t(n)$ bandbeschränkte Turingmaschine, die L akzeptiert. Nach Satz 5.11 b) kann angenommen werden, daß Q nur ein Band hat. Weiter gilt für deterministische Maschinen:

Lemma 5.45: Wird eine Sprache L von einer $C \cdot n$-bandbeschränkten 1-Band-Turingmaschine Q akzeptiert, so wird L auch von einer n+1-bandbeschränkten 1-Band-Turingmaschine M akzeptiert, für die gilt:
a) M druckt auf den Feldern, auf denen die Eingabe stand, nie das

Leerzeichen.

b) Der Kopf von M überschreitet die Felder, auf denen die Eingabe stand, nie nach links und höchstens um 1 Feld nach rechts. Auf diesem Feld druckt M nie etwas anderes als das Leerzeichen.

Beweis: Die Maschine Q wird zunächst nach dem Muster des Beweises von Lemma 1.35 so modifiziert, daß nie Felder links von der ursprünglichen Position des Kopfs besucht werden. Sei A das Alphabet der modifizierten Maschine Q. Das Alphabet von M ist $AUA^{\lceil C \rceil}$. M simuliert Q Schritt für Schritt und stellt dabei je $\lceil C \rceil$ konsekutive Zeichen der Bandinschrift von Q als ein einziges Alphabetzeichen dar. □

Sei nun $M=(A_M, Z_M, \delta, z_o, z_+, z_-)$ eine Maschine wie in Lemma 5.45 beschrieben; $B \in A_M$ sei das Leerzeichen. Wir geben eine Grammatik $G=(A,T,P,s)$ an mit $L(G)=\{BwB \mid w \in L(M)\}$. Es ist

$$A = A_M \cup \{[az] \mid a \in A_M, z \in Z_M\} \cup \{z_+, s\},$$

$$T = A_M,$$

und für alle $a,b,c \in A_M$ und $z,z' \in Z_M$ enthält P die folgenden Produktionen:

1) $B[z_o a] \to Ba$

2) $c[z'b] \to [za]b$ falls $(z,a,z',c,R) \in \delta_M$
   $[z'b]c \to b[za]$ falls $(z,a,z',c,L) \in \delta_M$
   $[z'c] \to [za]$ falls $(z,a,z',c,N) \in \delta_M$

3) $z_+ \to [z_+ a] \mid z_+ a \mid a z_+$
   $s' \to z_+ B$
   $s \to Bs'$.

Sei $w \in L(M)$ und $(k_o, \ldots, k_N)$ mit $k_o = Bz_o wB$, $k_N = Buz_+ vB$ sei eine akzeptierende Berechnung von M angesetzt auf w, wobei für jede Konfiguration k_i mit B beginnt und endet. Für jedes i gehe $\hat{k}_i$ aus k_i dadurch hervor, daß man die Teilzeichenreihe der Form $za, z \in Z_M, a \in A_M$, durch das Zeichen [za] ersetzt. Es folgt

$$s \to Bs' \to Bz_+ B \to^* Bu[z_+ v_1]v_2 \ldots v_{|v|}{}^{B=\hat{k}_N \to \hat{k}_{N-1} \to \ldots}$$

$$\ldots \to \hat{k}_o = B[z_o w_1]w_2 \ldots w_{|w|}{}^{B \to BwB},$$

also $BwB \in L(G)$.

Sei andererseits $v \in L(G)$ und $(s, Bs', s_1, \ldots, s_N)$ mit $s_1 = Bz_+ B$, $s_N = v \in T^*$

eine Ableitung von v aus s. Wir beobachten:

i) Das erste und letzte Zeichen von jedem s_i ist gleich B oder von der Form [zB], $z \in Z$ (das liegt daran, daß die Maschine M das Leerzeichen nie überdruckt).

ii) Es gibt ein s_i, in dem ein Zeichen der Form $[z_+a]$, $a \in A_M$, erscheint. s_m sei die erste Zeichenreihe, für die dies gilt

iii) Da $v \in T^*$, muß im letzten Schritt der Ableitung eine Produktion aus Gruppe 1) angewendet worden sein. Diese Anwendung kann nur am linken Ende von s_{N-1} erfolgt sein (das liegt daran, daß die Maschine M nie von B verschiedene Zeichen mit B überdruckt). Also $v=BwB$ für ein $w \in A_M^*$.

Für $i \in \{m,\ldots,N-1\}$ entstehe k_i aus s_i, indem man das Zeichen der Form [za] durch die Zeichenreihe za ersetzt. Nach dem oben Gesagten und nach Konstruktion der Produktionen aus Gruppe 2 folgt leicht: $(k_{N-1},\ldots,k_m)$ ist eine akzeptierende Berechnung von M angesetzt auf w, also $w \in L(M)$. □

Um die Umkehrung von Satz 5.43 zu erhalten, genügt es jetzt, das folgende technische Lemma zu zeigen:

Lemma 5.46: Sei $G=(A,T,P,s)$ eine kontextsensitive Grammatik, für die gilt:

1) Alle Produktionen in P sind von der Form $x_1 \to y_1 y_2$ oder $x_1 x_2 \to y_1 y_2; x_i, y_i \in A$;

2) s kommt nie auf der rechten Seite einer Produktion vor.

Dann ist die Sprache

$$L'=\{w \mid \exists a,b \in T: awb \in L(G)\}$$

kontextsensitiv.

Beweis: Sei $S_5=\{w \mid s \overset{*}{\underset{G}{\to}} w, |w|=5\}$.

Wir geben eine Grammatik $G'=(A',T',P',s')$ an:

$$A'=A \cup \{(ab) \mid a,b \in A\} \cup \{[ab] \mid a,b \in A\}$$
$$T'=T, s'=s.$$

P' enthält die folgenden Regeln:

1) $s \to w_2 \ldots w_{|w|-1}$ für alle $w \in L(G), |w| \leq 4$

2) $s \to (w_1 w_2) w_3 [w_4 w_5]$ für alle $w \in S_5$

3) Für jede Regel $x_1 \to y_1 y_2$ aus P und alle $a \in A$ enthält P' die Regeln:

$$x_1 \to y_1 y_2$$
$$(x_1 a) \to (y_1 y_2) a$$
$$(a x_1) \to (a y_1) y_2$$
$$[x_1 a] \to y_1 [y_2 a]$$
$$[a x_1] \to a [y_1 y_2]$$

Für jede Regel $x_1 x_2 \to y_1 y_2$ aus P und für alle $a \in A$ enthält P' die Regeln:

$$x_1 x_2 \to y_1 y_2$$
$$(x_1 x_2) \to (y_1 y_2)$$
$$(a x_1) x_2 \to (a y_1) y_2$$
$$[x_1 x_2] \to [y_1 y_2]$$
$$x_1 [x_2 a] \to y_1 [y_2 a].$$

4) Für alle $a,b \in T$ enthält P' die Regeln:

$$(ab) \to b$$
$$[ab] \to a.$$

Wir zeigen, daß $L'=L(G')$ gilt. Sei $w \in L'$, $a,b \in A$ und $u=awb \in L(G)$. Ist $|u|<5$, so ist $w \in L(G')$ nach Regelgruppe 1 von G'. Andernfalls sei $(u^0,\ldots,u^n)$ eine Ableitung von u aus s in G, d.h. $s=u^0 \underset{G}{\to} \ldots \underset{G}{\to} u^n = u$.

Sei $m=\min\{j \mid 5=|u^j|\}$, und für u mit $|u| \geq 5$ sei

$$\tilde{u}=(u_1 u_2) u_3 \ldots u_{|u|-2} [u_{|u|-1}, u_{|u|}].$$

Es folgt

$s \underset{G'}{\to} \tilde{u}^m$ nach Regegruppe 2 von G',

$\tilde{u}^m \underset{G'}{\to} {}^*\tilde{u}^n$ nach Regelgruppe 3 von G' und

$\tilde{u}^n \underset{G'}{\to} {}^*w$ nach Regelgruppe 4 von G',

also $w \in L(G')$.

Sei andererseits $w \in L(G')$. Ist w in G' direkt aus s ableitbar, so kann dies nur nach Regelgruppe 1 möglich sein, d.h. $|w| \leq 2$, und es gibt $a,b \in A$, so daß $awb \in L(G)$ gilt, also $w \in L'$. Andernfalls sei

$(u^o,\ldots,u^n)$ eine Ableitung von w aus s in G', d.h.

$$s=u^o \underset{G'}{\rightarrow} \ldots \underset{G'}{\rightarrow} u^n=w \text{ mit } n>1.$$

$s \underset{G'}{\rightarrow} u^1$ kann in diesem Fall nur nach Regelgruppe 2 gelten. Also ist u^1 von der Form $(z_1z_2)z_3[z_4z_5], z_i \in A$, wobei $s \underset{G}{\rightarrow}^* z_1z_2z_3z_4z_5$ gilt.

Wir behaupten, daß man die Ableitung $(u^o,\ldots,u^n)$ so modifizieren kann, daß gilt:

$$u^{i-1} \underset{G'}{\rightarrow} u^i \text{ nach Regelgruppe } \begin{cases} 3 & \text{für } 1\leq i<n-2 \\ 4 & \text{für } n-2\leq i\leq n \end{cases}$$

Denn gilt etwa $(ab)u[cd] \underset{G'}{\rightarrow} bu[cd] \underset{G'}{\rightarrow} bu'[c'd']$, so gilt auch $(ab)u[cd] \underset{G'}{\rightarrow} (ab)u'[c'd'] \underset{G'}{\rightarrow} bu[c'd']$, d.h. man kann in Ableitungen die Anwendung einer Regel aus Regelgruppe 4 immer "nach hinten schieben".

Für Zeichenreihen $u\in(A')^*$ gehe $\tilde{u}$ aus u hervor, indem alle Zeichen Zeichen der Form (ab) bzw. [ab] durch die Zeichenreihe ab ersetzt werden. Die Ableitung $(u^o,\ldots,u^n)$ sei modifiziert wie oben angegeben. Dann folgt $s \underset{G}{\rightarrow}^* \tilde{u}^1 \underset{G}{\rightarrow}^* \tilde{u}^{n-2}=awb$ für gewisse Zeichen $a,b\in A$, also $w\in L'$. □

Aus Satz 5.43, Satz 5.44 und Lemma 5.46 folgt

Satz 5.47: $\{L|L \text{ ist kontextsensitiv}\}=NTIME(n)$.

Korollar 5.48: Jede kontextsensitive Sprache kann von einer deterministischen $O(n^2)$-bandbeschränkten Turingmaschine akzeptiert werden.

Beweis: folgt unmittelbar aus Satz 5.13.

Korollar 5.49: Es gibt kontextsensitive Sprachen, die von keiner o(n log n)-zeitbeschränkten Mehrband-Turingmaschine akzeptiert werden können.

Beweis: Nach Satz 5.32 und Satz 4.31 b) gilt:

$DTIME(o(n \log n))\subseteq DTAPE(o(n))\subsetneq DTAPE(n)\subseteq NTAPE(n)=\{L|L \text{ ist kontextsensitiv}\}$. □

In ähnlicher Weise kann man aus Satz 5.32 für mehrere andere Probleme nichtlineare untere Zeitschranken herleiten [35].

Übungen

1. Man führe den Beweis von Lemma 5.46 zu Ende.

2. Man zeige: L ist Typ-0-Sprache genau dann, wenn es eine Turingmaschine M mit L=L(M) gibt.

3. Ein (nicht-) deterministischer endlicher Automat ist eine (nicht-) deterministische 1-Band-Turingmaschine $M=(A,Z,\delta,z_o,E)$, für die gilt:

   i) M darf auf dem Band nicht schreiben und muß bei jedem Schritt den Kopf um 1 Feld nach rechts bewegen.

   ii) $\delta(z,a)$ darf auch für Endzustände $z\in E$ definiert sein.

   $L(M)=\{w|$ es gibt eine Rechnung $(z_ow,\ldots,wz_e)$ mit $z_e\in Z_E$ von M angesetzt auf $w\}$.

   Man zeige:

   a) Ist L Typ-3-Sprache, so gibt es einen nichtdeterministischen endlichen Automaten M mit L=L(M).

   b) Ist $M=(A_M,Z_M,\delta_M,z_o,Z_{EM})$ ein nichtdeterministischer endlicher Automat, so gibt es eine reguläre Grammatik G=(A,T,P,s) mit L(G)=L(M).

      Hinweis: $A\setminus T=\{Z|Z\subseteq Z_M\}$.

   c) Zu jedem nichtdeterministischen Automat M gibt es einen deterministischen Automaten Q, so daß L(M)=L(Q) gilt.

      Hinweis: $Z_Q=\{Z|Z\subseteq Z_M\}$.

4. a) Ist M eine deterministische 1-Band-Turingmaschine, die für alle Eingaben hält und nie eine Crossingsequenz der Länge größer als c, $c\in\mathbb{N}$, generiert. Dann gibt es einen nichtdeterministischen endlichen Automaten Q mit L(M)=L(Q).

      Hinweis: $Z_Q=\bigcup_{i=0}^{c} Z_M^i$.

   b) Ist L=L(M) für eine o(n log n)-zeitbeschränkte 1-Band-Turingmaschine, so ist L regulär.

      Hinweis: Man verwende den Beweis von Satz 4.4o.

5. (Chomsky-Normalform):

   Zu jeder kontextfreien Grammatik G=(A,T,P,s) gibt es eine

kontextfreie Grammatik $G'=(A',T,P',s')$, so daß $L(G)=L(G')$ gilt, und alle Produktionen in G' sind von der Form $u\to v_1v_2$ oder $u\to t$ mit $u,v_1,v_2\in A'\setminus T, t\in T$.

6. $\{L|L$ ist kontextfrei$\}=\{L|L$ wird von einem nichtdeterministischen 1-Pushdown-Automaten akzeptiert$\}$.

7. Sei $G=(A,T,P,s)$ eine kontextfreie Grammatik in Chomsky-Normalform und $w\in T^*, |w|=n$.

   Für $i,j\in\{1,\ldots,n\}$ sei $S_{i,j}=\{a|a\in A, a\to^* w_i\ldots w_j\}$.

   Man gebe ein Verfahren an, um

   a) $S_{i,i}$ zu berechnen;

   b) alle $S_{i,j}$ mit $j-i=k$ zu berechnen, falls alle S_{ij} mit $j-i=k-1$ bekannt sind.

   c) $\{L|L$ kontextfrei$\}\subseteq DTIME(n^3)$.

8. $\{L|L$ kontextfrei$\}\subseteq NTAPE(\log n)\subseteq DTAPE((\log n)^2)$.

   Hinweis: Man benutze Übung 1 a) aus Kapitel 5.2.

9. a) $\{L|L$ kontextfrei$\}\setminus DTIME_1(o(n^2))\neq\emptyset$

   b) $\{L|L$ kontextfrei$\}\setminus DTAPE(o(\log(n))\neq\emptyset$.

Offene Probleme

Die offensichtlichen Probleme sind, scharfe obere und untere Schranken für den Platz- bzw. Zeitbedarf herzuleiten, der zum Erkennen von kontextfreien bzw. kontextsensitiven Sprachen nötig ist. Dies führt sämtlich auf Spezialfälle von früher genannten offenen Problemen (warum?).

Das prominenteste dieser Probleme ist das LBA-Problem: $NTAPE(n)=DTAPE(n)$? Der Name kommt daher, daß man $O(n)$-bandbeschränkte deterministische bzw. nichtdeterministische Maschinen auch "deterministic linear bounded automaton (DLBA)" bzw. "linear bounded automaton (LBA)" nennt. Dieses Problem ist aus folgendem Grund interessant: Wir haben gesehen, daß die Frage nach der Bandkomplexität kontextsensitiver Sprachen auf das LBA-Problem führt. Es stellt sich heraus, daß die Frage nach der Komplexität vieler anderer natürlicher Probleme ebenfalls auf dieses Problem führt [23]. Wir bemerken abschließend noch, daß die Zeitschranke für die

Erkennung kontextfreier Sprachen aus Übung 6 auf

$$O(n^{\log_2 7}) \approx O(n^{2.81})$$

verbessert werden kann [72].

6. Vollständige Probleme

Wir studieren in diesem Kapitel eine einfache Technik, die es gestattet, Resultate der Art: "Die Sprachen L und L' sind ungefähr gleich schwer zu erkennen" und "L ist beinahe die schwierigste Sprache in der Sprachklasse K" herzuleiten. Hierdurch gelingt es, die Schwierigkeit einer überraschend großen Zahl von Problemen zu klassifizieren und auch von gewissen Problemen zu zeigen, daß sie nur mit enormem Rechenaufwand zu lösen sind.

6.1. Effiziente Reduzierbarkeit

6.1.1. Praktisch durchführbare Rechnungen

Wir untersuchen im folgenden insbesondere die folgenden Sprachklassen:

Definition 6.1:

$$P := \bigcup_k DTIME(n^k)$$

$$NP := \bigcup_k NTIME(n^k)$$

$$PTAPE := \bigcup_k DTAPE(n^k)$$

$$EXPTIME := DTIME(2^{O(n)})$$

$$EXPTAPE := DTAPE(2^{O(n)}).$$

Alle diese Klassen sind zwar durch Turingmaschinen definiert, aber aus Satz 3.18 folgt, daß man genau die gleiche Klasse P bzw. EXPTIME erhalten würde, wenn man sie über Registermaschinen definieren würde; diese Klassen sind vom zugrundeliegenden Maschinenmodell unabhängig, solange man sich nur auf Modelle beschränkt, deren Zeitkomplexitätsmaß mit dem der Turingmaschinen polynomiell verknüpft ist. Auf die gleiche Weise könnte man die Klassen PTAPE bzw. EXPTAPE durch platzbeschränkte Registermaschinen oder die Klasse NP durch nichtdeterministische Registermaschinen definieren, und man würde die gleichen Sprachklassen erhalten.

Man kann sich darüber streiten, ob jede Funktion, die in polynomieller Zeit berechenbar ist, auch vom praktischen Standpunkt aus berechenbar ist (etwa wenn bei Eingabe der Länge n n^{215} Schritte oder auch nur n^4 Schritte auszuführen sind). Ist andererseits eine Funktion nicht in polynomieller Zeit berechenbar, so heißt das, daß man sie in der Praxis höchstens noch für ganz

kleine Eingaben auswerten kann, weil sonst das Ende der Welt dem Ende der Rechnung zuvorkäme. Insofern bedeutet der Nachweis, daß eine Sprache nicht in P liegt, daß ihre charakteristische Funktion für praktische Zwecke als nicht berechenbar angesehen werden kann. Wir werden im folgenden unter anderem für eine interessante Sprache diesen Nachweis führen.

6.1.2. Polynomielle Reduzierbarkeit

Definition 6.2: Eine Funktion $f:A^*\to A^*$ heißt in polynomieller Zeit berechenbar, falls es ein $k\in\mathbb{N}$ und eine deterministische $O(n^k)$-zeitbeschränkte Turingmaschine M gibt, die, angesetzt auf Eingabe $w\in A^*$, die Ausgabe $f(w)$ liefert.

Lemma 6.3: Sind $f,g:A^*\to A^*$ in polynomieller Zeit berechenbar, so auch $f\circ g$.

Beweis: Sei M eine $O(n^k)$-zeitbeschränkte Maschine, die f berechnet und Q eine $O(n^s)$-zeitbeschränkte Maschine, die g berechnet. Es folgt $|g(w)|\leq O(|w|^s)$ für alle $w\in A$.

M' sei eine Maschine, die sich, angesetzt auf Eingabe w, erst wie Q verhält, d.h. g(w) produziert, und sich dann wie M angesetzt auf g(w) verhält. M' berechnet fog und macht bei Eingabe w höchstens $O(|w|^s)+O(|g(w)|^k)=O(|w|^{ks})$ Schritte. □

Definition 6.4: $L,L'\subseteq A^*$ seien Sprachen. L' heißt polynomiell reduzierbar auf L ($L'\leq_p L$), falls es eine in polynomieller Zeit berechenbare Funktion $f:A^*\to A^*$ gibt, so daß für alle $w\in A^*$ gilt: $w\in L'\Longleftrightarrow f(w)\in L$. Wir sagen hierfür auch: L' wird vermittels f auf L reduziert. Polynomielle Reduzierbarkeit ist eine spezielle Form der Reduzierbarkeit aus Kapitel 2.1. Die Relation $\leq_p$ ist transitiv, denn ist $L\leq_p L'$ vermittels g und $L'\leq_p L''$ vermittels f, so ist $w\in L\Longleftrightarrow f(g(w))\in L''$. Nach Lemma 6.3 ist fog in polynomieller Zeit berechenbar.

Definition 6.5: Zwei Sprachen L und L' heißen polynomiell äquivalent ($L\equiv_p L'$), falls $L\leq_p L'$ und $L'\leq_p L$ gilt.

Nach dem oben Gesagten folgt sofort, daß $\equiv_p$ eine Äquivalenzrelation ist. Außerdem folgt aus $L\equiv_p L'$: $L\in P\Longleftrightarrow L'\in P$; d.h. die Sprachen L und L' sind entweder beide mit erträglichen Aufwand erkennbar oder keine. Wir lernen im folgenden Klassen polynomiell äquivalenter Sprachen kennen, für deren Erkennung kein polynomiell

zeitbeschränktes Verfahren bekannt ist, aber es ist auch nicht bewiesen, daß es ein solches Verfahren nicht gibt. Da diese Sprachen polynomiell äquivalent sind, hat man jedoch die Erkennung aller dieser Sprachen vom Komplexitätsstandpunkt aus als ein und dasselbe Problem identifiziert.

Definition 6.6: Sei K eine Klasse von Sprachen, z.B. K=NP, PSPACE, EXPTIME. Eine Sprache L heißt (bezüglich $\leq_p$) K-hart, falls für jede Sprache L' aus K gilt: $L' \leq_p L$.

Lemma 6.7: $L \subseteq A^*$ sei K-hart. Es gilt:

a) $L \in P \Rightarrow K \subseteq P$

b) $K \setminus P \neq \emptyset \Rightarrow L \notin P$.

Beweis: a) Sei χ_L, die charakteristische Funktion von L, in polynomieller Zeit berechenbar. Ist $L' \in K$ beliebig und gilt $L' \leq_p L$ vermittels f, so ist $\chi_{L'} = \chi_L \circ f$ nach Lemma 6.3 in polynomieller Zeit berechenbar, also $L' \in P$.

b) folgt direkt aus a). □

Korollar 6.8: Ist L EXPTIME-hart, so folgt $L \notin P$.

Beweis: $EXPTIME \setminus P \neq \emptyset$ nach dem Zeithierarchiesatz. □

Wir werden von einer gewissen Sprache nachweisen, daß sie schwer entscheidbar ist, indem wir zeigen, daß sie EXPTIME-hart ist.

Definition 6.9: Sei K eine Klasse von Sprachen. L heißt (bezüglich $\leq_p$) K-vollständig, falls $L \in K$ gilt und L K-hart ist.

Lemma 6.1o: a) Ist L K-vollständig, so gilt $L \in P \Longleftrightarrow K \subseteq P$.

b) Ist L K-vollständig, $L' \in K$ und $L \leq_p L'$, so ist L' K-vollständig.

c) Sind L und L' K-vollständig, so gilt $L \equiv_p L'$.

Beweis: a) folgt aus Lemma 6.7.

b) Ist $L'' \in K$, so gilt $L'' \leq_p L \leq_p L'$, also $L'' \leq_p L'$ wegen der Transitivität von $\leq_p$.

c) Da L K-hart ist und $L' \in K$ gilt, folgt $L' \leq_p L$. Ebenso folgt $L \leq L'$. □

Nach Teil a) dieses Lemmas repräsentiert eine vollständige Sprache allein schon die Komplexität ihrer Klasse. Teil b) ist das perfekte Gegenstück zu Satz 2.9 b) aus dem Kapitel über nicht be-

rechenbare Funktionen. Dort konnte man durch Reduktion einer nicht berechenbaren Sprache L auf eine andere Sprache L' zeigen, daß L' ebenfalls nicht berechenbar ist. Hier gewinnt man durch Reduktion aus vollständigen Problemen weitere vollständige Probleme.

Das Konzept der polynomiellen Reduzierbarkeit hat man aus der allgemeinen Reduzierbarkeit erhalten, indem man zusätzlich eine Bedingung über die Komplexität der Funktion f gestellt hat. Die Bedingung, in polynomieller Zeit berechenbar zu sein, ist nicht die einzig mögliche; in den Übungen behandeln wir einige andere Formen von Reduzierbarkeit mit Komplexitätsbedingungen. Polynomielle Reduzierbarkeit hat sich jedoch von diesen Formen bisher als am fruchtbarsten erwiesen.

Übungen

1. Ist $L \subseteq A^*$, $L \neq \emptyset$ und $L \neq A^*$, so ist L (bezüglich $\leq_p$) P-vollständig.

2. $NPTAPE := \bigcup_k NTAPE(n^k)$

   Man zeige: NPTAPE = PTAPE.

3. $L = \{u\#v\#^s \mid u,v \in \{0,1\}^*, s \in \mathbb{N}$, u ist Gödelnummer einer deterministischen Turingmaschine M_u, M_u akzeptiert v und benutzt dabei höchstens $s/|u|$ Bandfelder$\}$.

   a) $L \in DTAPE(n)$

   b) L ist PTAPE-vollständig.

4. Man gebe Sprachen an, die bezüglich $\leq_p$

   a) NTAPE(n)-vollständig

   b) NP-vollständig

   sind.

5. $f: A^* \to A^*$ heißt in linearer Zeit berechenbar, falls es eine deterministische O(n)-zeitbeschränkte Turingmaschine gibt, die bei Eingabe $w \in A^*$ als Ergebnis f(w) liefert. Reduzierbarkeit in linearer Zeit ($\leq_{lintime}$) und Vollständigkeit werden analog zu polynomieller Reduzierbarkeit und Vollständigkeit definiert:

   a) Die Sprache aus Übung 3 ist bezüglich $\leq_{lintime}$ DTAPE(n)-vollständig

   b) Es gibt keine bezüglich $\leq_{lintime}$ P-vollständige Sprache

c) P≠DTAPE(n)

d) P≠NTAPE(n).

6. f:A*→A* heißt auf logarithmischem Band berechenbar, falls es eine deterministische Off-line-Turingmaschine M gibt, so daß gilt:

   i) M hat ein spezielles Ausgabeband, auf dem nur geschrieben werden darf.

   ii) Angesetzt auf Eingabe w∈A* druckt M auf das Ausgabeband f(w).

   iii) M benutzt auf dem Arbeitsband bei Eingabe w nur O(log|w|) Felder.

   Reduzierbarkeit auf logarithmischem Band ($\leq_{\text{logtape}}$) und Vollständigkeit werden auf naheliegende Weise definiert.

   Man formuliere und beweise die Sätze von § 6.1.2 für $\leq_{\text{logtape}}$.

7. Sei L={(r_G,a,b) | r_G ist Repräsentation eines gerichteten Graphen G=(V,E), a,b∈V, und es gibt einen Pfad in G von a nach b}.

   a) L∈NTAPE(log n);

   b) L ist NTAPE(log n)-vollständig bezüglich $\leq_{\text{logtape}}$.

   Hinweis: Sei L'∈NTAPE(log n) und M ein nichtdeterministischer C·log n-bandbeschränkter 1-Band-Akzeptor für L' mit Alphabet A und Zustandsmenge Z. Eingaben w mit |w|=n bilde man auf Graphen $G_w=(V_w,E_w)$ ab, wobei gilt:

$$V_n=\{1,\ldots,n\}\times\{1,\ldots,C\cdot\log n\}\times Z\times A^{C\cdot\log n}$$

   (bis auf die Inschrift des Eingabebandes ist dies die Menge der möglichen Konfigurationen von M bei Eingaben der Länge n).

Offene Fragen

Für DTAPE(log n) ist auch die Bezeichnung log-TAPE üblich. Es gilt

(*) log-TAPE ⊆ P ⊆ NP ⊆ PTAPE.

Überdies gilt nach dem Bandhierarchiesatz log-TAPE$\subsetneq$PTAPE. Also ist mindestens eine der Inklusionen in (*) echt. Welche? Diese Fragen

hängen engstens mit den in Kapitel 5.2 genannten offenen Fragen zusammen, vergl. etwa Übung 4 aus Kapitel 5.1.

6.2. NP-vollständige Probleme

Wir beschreiben eine spezielle Sprache, die auch "das Erfüllbarkeitsproblem" genannt wird, und weisen die NP-Vollständigkeit dieser Sprache nach. Durch Reduktion dieser Sprache auf andere Sprachen kann man leicht die NP-Vollständigkeit weiterer Sprachen zeigen.

6.2.1. Das Erfüllbarkeitsproblem

V sei eine abzählbare Menge von Variablen, die später Werte aus $\{0,1\}$ annehmen. Wir bezeichnen Variablen aus V meistens mit $x_1, x_2, \ldots$.

Die folgende Definition gleicht der Definition der arithmetischen Ausdrücke aus § 2.2.3.

Definition 6.11: Die Menge der Boole'schen Ausdrücke ist die kleinste Menge B, für die gilt:

1) $0 \in B$, $1 \in B$;
2) $x \in V \Rightarrow x \in B$;
3) sind $b_1, b_2 \in B$, so auch $(b_1 \vee b_2)$, $(b_1 \wedge b_2)$, $(\sim b_1)$.

Beispiel 6.12: $((x_1 \wedge x_2) \vee (x_1 \wedge (\sim x_3)))$ ist ein Boole'scher Ausdruck.

Boole'sche Ausdrücke sind Zeichenreihen über dem unendlichen Alphabet $A=\{\wedge,\vee,\sim,(,)\} \cup V$. Die Länge eines Boole'schen Ausdrucks ist die Länge dieser Zeichenreihe. Der Ausdruck in Beispiel 6.12 hat Länge 16.

Wenn man in arithmetische Ausdrücke für die Variablen Zahlen einsetzt, erhält man dadurch im allgemeinen in naheliegender Weise eine Zahl; setzt man etwa in $(x_1+x_2)\cdot x_1$ für $x_1=2$ und $x_2=1$, so liefert der Ausdruck den Wert 6. Wir tun das gleiche für Boole'sche Ausdrücke und interpretieren die logischen Verknüpfungen $\wedge,\vee,\sim$ jetzt als Funktionen $\wedge,\vee:\{0,1\}^2 \to \{0,1\}$, $\sim:\{0,1\} \to \{0,1\}$. Für $x_i, x_j \in \{0,1\}$ ist:

$$x_i \wedge x_j = \begin{cases} 1 & \text{falls } x_i = x_j = 1 \\ 0 & \text{sonst} \end{cases}$$

$$x_i \vee x_j = \begin{cases} 0 & \text{falls } x_i = x_j = 0 \\ 1 & \text{sonst} \end{cases}$$

$$\sim x_i = \begin{cases} 0 & \text{falls } x_i = 1 \\ 1 & \text{sonst} \end{cases}$$

Die Funktionen $\wedge, \vee, \sim$ nennt man auch Konjunktion, Disjunktion und Negation.

Definition 6.13: Eine Belegung ist eine Abbildung $\varphi: V \to \{0,1\}$. Ist b ein Boole'scher Ausdruck, so wird $\varphi(b)$, der Wert von b bei Belegung φ, rekursiv definiert durch:

$$\varphi(b) = \begin{cases} b & \text{falls } b \in \{0,1\} \\ \varphi(x_i) & \text{falls } b = x_i, x_i \in V \\ \sim\varphi(b_1) & \text{falls } b = (\sim b_1) \\ \varphi(b_1) \vee \varphi(b_2) & \text{falls } b = (b_1 \vee b_2) \\ \varphi(b_1) \wedge \varphi(b_2) & \text{falls } b = (b_1 \wedge b_2). \end{cases}$$

Beispiel 6.14: $b = ((x_1 \vee x_2) \wedge x_1), \varphi(x_1) = 1, \varphi(x_2) = 0, \varphi(b) = 1.$

Für $\varphi(b)=1$ sagen wir auch: φ erfüllt b. Ein Boole'scher Ausdruck b heißt erfüllbar, wenn es eine Belegung φ mit $\varphi(b)=1$ gibt. Der Ausdruck aus Beispiel 6.14 ist erfüllbar. $(x_1 \vee (\sim x_1))$ ist nicht erfüllbar.

Mit arithmetischen Ausdrücken kann man rechnen, d.h. innerhalb gewisser Regeln verschiedene Ausdrücke gleichsetzen, z.B. $(a+b)\cdot(a+b) = a\cdot a + (1+1)\cdot a\cdot b + b\cdot b$. Wir erklären jetzt, wann man Boole'sche Ausdrücke gleichsetzen darf:

Definition 6.15: Boole'sche Ausdrücke b_1 und b_2 heißen äquivalent ($b_1 = b_2$), falls $\varphi(b_1) = \varphi(b_2)$ für alle Belegungen φ gilt.

Lemma 6.16: Für alle Boole'schen Ausdrücke a,b,c gilt:

a) $(a \vee b) = (b \vee a)$	$(a \wedge b) = (b \wedge a)$
b) $(a \vee (b \vee c)) = ((a \vee b) \vee c)$	$(a \wedge (b \wedge c)) = ((a \wedge b) \wedge c)$
c) $(a \vee (a \wedge b)) = a$	$(a \wedge (a \vee b)) = a$
d) $(a \vee (b \wedge c)) = ((a \vee b) \wedge (a \vee c))$	$(a \wedge (b \vee c)) = ((a \wedge b) \vee (a \wedge c))$
e) $(a \vee (b \wedge (\sim b))) = a$	$(a \wedge (b \vee (\sim b))) = a$
f) $(a \vee (\sim a)) = 1$	$a \wedge (\sim a) = 0$

Beweis: a) $\varphi(a \vee b)=\varphi(a)\vee\varphi(b)=\varphi(b)\vee\varphi(a)=\varphi(b \vee a)$.

$\varphi(a\wedge b)=\ldots$ usw. □

Ist M eine Menge und sind $\wedge,\vee: M^2 \to M$ und $\sim: M \to M$ Abbildungen, so daß a) - f) für alle $a,b,c \in M$ gelten, so heißt $(M,\wedge,\vee,\sim)$ eine Boole'sche Algebra.

Teil b) von Lemma 6.16 rechtfertigt, Boole'sche Ausdrücke der Form $(b_1 \vee \ldots \vee b_s)$ bzw. $(b_1 \wedge \ldots \wedge b_s)$ zu bilden, da es auf die Reihenfolge der Klammerung nicht ankommt. Zur weiteren Vereinfachung der Notation vereinbaren wir die Schreibweise $\overline{x}_i$ für $(\sim x_i), x_i \in V$; um beim Schreiben Klammern zu sparen, vereinbaren wir folgende Prioritäten: $\sim$ bindet vor $\wedge$, und $\wedge$ bindet vor $\vee$. Ähnlich wie beim Multiplikationspunkt in arithmetischen Ausdrücken läßt man in Boole'schen Ausdrücken das Zeichen $\wedge$ oft weg.

Beispiel 6.17: Für $((x_1\wedge((\sim x_2)\wedge x_3))\vee((\sim x_1)\wedge x_2))$ schreibt man auch $x_1\overline{x}_2x_3 \vee \overline{x}_1x_2$.

Definition 6.18: Eine Klausel mit s Literalen ist ein Boole'scher Ausdruck der Form $(y_1 \vee \ldots \vee y_s)$, $y_i \in \{x_1,x_2,\ldots\} \cup \{\overline{x}_1,\overline{x}_2,\ldots\} \cup \{0,1\}$. Ein Boole'scher Ausdruck b heißt in konjunktiver Normalform, falls b von der Form $K_1 \wedge \ldots \wedge K_t$ ist, wobei die K_i Klauseln sind.

Satz 6.19: Zu jedem Boole'schen Ausdruck b mit n Variablen kann man einen äquivalenten Boole'schen Ausdruck b' in konjunktiver Normalform mit höchstens 2^n Klauseln mit je n Literalen angeben.

Beweis: Ist $\varphi(b)=1$ für alle Belegungen, so ist $b'=1$. Andernfalls seien OBdA $x_1,\ldots,x_n$ die Variablen in b.

Jeder Belegung mit $\varphi(b)=0$ wird eine Klausel $K(\varphi)=(y_1(\varphi)\vee\ldots\vee y_n(\varphi))$ zugeordnet durch:

$$y_j(\varphi) = \begin{cases} x_j & \text{falls } \varphi(x_j)=0 \\ \overline{x}_j & \text{falls } \varphi(x_j)=1. \end{cases}$$

Man erhält höchstens $s=2^n$ verschiedene solche Klauseln $K_1,\ldots,K_s$. Sei $b'=K_1\wedge\ldots\wedge K_s$.

Ist $\varphi(b)=0$, folgt $K(\varphi)=0$, also $\varphi(b')=0$.

Ist andererseits φ' eine Belegung mit $\varphi'(b')=0$, so ist $\varphi'(K_i)=0$ für irgendeine Klausel K_i. Aber $K_i=y_1(\varphi)\vee\ldots\vee y_n(\varphi)$ für irgendeine Belegung φ mit $\varphi(b)=0$. Es folgt $\varphi'(y_j(\varphi))=0$ für alle y_j. Nach Konstruktion der $y_j(\varphi)$ folgt $\varphi'(x_j)=\varphi(x_j)$ für alle x_j, also

$\varphi'(b)=\varphi(b)=0$. □

Der Ausdruck b' heißt die vollständige konjunktive Normalform von b. Das Erfüllbarkeitsproblem (englisch: Satisfiability problem) ist, zu gegebenem Boole'schen Ausdruck b in konjunktiver Normalform zu entscheiden, ob b erfüllbar ist. Um dieses Problem als Sprach-erkennungsproblem formulieren zu können, kodiert man Boole'sche Ausdrücke als Zeichenreihen über einem endlichen Alphabet A, z.B. $A=\{(,),x,0,1,\wedge,\vee,\sim\}$, wobei x_i durch xbin(i) ersetzt wird.

Beispiel 6.2o: $((x_1 \vee x_2) \wedge (x_3))$ wird kodiert in

((x1∨x1o)∧(x11)).

6.2.2. NP-Vollständigkeit des Erfüllbarkeitsproblems

Satz 6.21 (Cook): Die Sprache

SAT={b|b ist Kodierung eines erfüllbaren Boole'schen Ausdrucks in konjunktiver Form},

ist NP-vollständig.

Beweis: SAT∈NP ist leicht zu zeigen; man muß nur eine polynomiell zeitbeschränkte, nichtdeterministische Maschine M angeben, die SAT akzeptiert. Wir beschreiben eine von vielen verschiedenen Möglichkeiten:

Angesetzt auf Eingabe w testet M zuerst, ob w überhaupt Kodierung eines Boole'schen Ausdrucks in konjunktiver Normalform ist. Ist w keine solche Kodierung, verwirft M sofort. Ist w Kodierung eines Boole'schen Ausdrucks, erstellt M eine Liste aller in w vorkommenden Variablen (und entfernt insbesondere doppelt vorkommende Variablen aus dieser Liste). Dann rät M für jede Variable x_i aus der Liste eine Zahl $\varphi(x_i)\in\{0,1\}$ und ersetzt in w jedes Vorkommen von x_i durch $\varphi(x_i)$. Schließlich wertet M den hierdurch entstandenen Ausdruck, in dem keine Variablen mehr vorkommen, aus und akzeptiert genau dann, wenn 1 herauskommt.

M akzeptiert SAT , und es ist nicht schwer, M so zu organisieren, daß M $O(n^2)$-zeitbeschränkt ist, also $SAT\in NTIME(n^2)\subseteq NP$.

Es bleibt zu zeigen, daß die Sprache SAT NP-hart ist. Sei also $L\in NP$ beliebig, p(n) ein Polynom und $M=(A,Z,\delta,z_o,z_+,z_-)$ eine p(n)-zeitbeschränkte nichtdeterministische Turingmaschine, die L akzeptiert. Nach Satz 5.11 a) kann angenommen werden, daß M nur

ein Band hat. Weiter sei M so modifiziert, daß $(z_+,a,z_+,a,N)\in\delta$ für alle $a\in A$ gilt.

Wir müssen zeigen, daß es eine polynomiell zeitbeschränkte deterministische Maschine gibt, die, angesetzt auf Eingabe w, einen Boole'schen Ausdruck F(w) in konjunktiver Normalform erzeugt, der genau dann erfüllbar ist, wenn es eine akzeptierende Berechnung der Länge $p(|w|)$ von M angesetzt auf w gibt. Wir geben zu beliebig vorgegebenem w diesen Ausdruck explizit an:

Sei $w\in A^*, |w|=n$. Statt der Variablen $x_1,x_2\ldots$ benutzen wir für $0\le t\le p(n)$ die folgenden Mengen V_t von Variablen:

$$V_t=\{c_{i,a,t}\mid a\in A,-p(n)\le i\le p(n)\}$$
$$\cup\{h_{i,t}\mid -p(n)\le i\le p(n)\}$$
$$\cup\{s_{z,t}\mid z\in Z\}.$$

Mit diesen Variablen bilden wir einen Ausdruck F(w), der genau dann erfüllbar sein soll, wenn es eine Folge $(k_o,\ldots,k_{p(n)})$ von Konfigurationen der Maschine M gibt mit:

$$k_o = B^{p(n)}z_o wB^{p(n)-n+1}$$
$$k_t \vdash k_{t+1} \text{ für } 0\le t\le p(n) \text{ und}$$
$$z_+ \text{ kommt in } k_{p(n)} \text{ vor.}$$

Der Ausdruck F(w) wird so gebildet, daß man aus einer Belegung φ, die F(w) erfüllt, die gewünschte Konfigurationenfolge gewinnen kann. Wir geben im voraus an, wie das geschehen soll.

Definition 6.22: Sei φ eine Belegung von V_t, so daß gilt:

(*) $\varphi(s_{z,t})=1$ für genau ein $z\in Z$,

$\varphi(h_{i,t})=1$ für genau ein $i\in\{-p(n),\ldots,p(n)\}$ und

für alle $i\in\{-p(n),\ldots,p(n)\}$ gilt $\varphi(c_{i,a,t})=1$ für genau ein $a\in A$. Dann wird die Konfiguration $k_t(\varphi)$ definiert durch:

Zustand von k_t=z falls $\varphi(s_{z,t})=1$,

Inhalt von Zelle i von k_t=a falls $\varphi(c_{i,a,t})=1$,

Kopfposition von k_t=i falls $\varphi(h_{i,t})=1$.

Der Ausdruck F(w) darf nur durch solche Belegungen φ erfüllt werden, für die (*) für alle $t\in\{0,\ldots,p(n)\}$ gilt. Der Ausdruck

$$u(x_1,\ldots,x_r)=(x_1\vee\ldots\vee x_r)\wedge\bigwedge_{i\neq j}(\bar{x}_i\vee\bar{x}_j)$$

hat Länge $O(r^2)$, und es gilt $\varphi(u(x_1,\ldots,x_r))=1$ genau dann, wenn $\varphi(x_i)=1$ für genau ein $i\in\{1,\ldots,r\}$ gilt.

Sei $Z=\{z_1,\ldots,z_q\}$ und $A=\{a_1,\ldots,a_r\}$.

Der Ausdruck

$$\begin{aligned}CONF(V_t) = {} & u(s_{z_1,t},\ldots,s_{z_q,t})\\ & \wedge u(h_{-p(n),t},\ldots,h_{p(n),t})\\ & \wedge \bigwedge_{-p(n)\leq i\leq p(n)} u(c_{i,a_1,t},\ldots,c_{i,a_r,t})\end{aligned}$$

hat Länge $O(p^2(n))$, und es gilt $\varphi(CONF(V_t))=1$ genau dann, wenn φ (*) erfüllt. Aus jeder Belegung φ, die den Ausdruck $CONF(V_t)$ erfüllt, kann man also nach Definition 6.22 eine Konfiguration $k_t(\varphi)$ gewinnen. Wir suchen aber nur solche Belegungen, für die auch $k_t(\varphi)\vdash k_{t+1}(\varphi)$ für $0\leq t\leq p(n)$ gilt.

Wir definieren $x\equiv y:=(\bar{x}\vee y)\wedge(x\vee\bar{y})$. Es gilt $\varphi(x\equiv y)=1$ genau dann, wenn $\varphi(x)=\varphi(y)$ gilt.

Der Ausdruck

$$D_t = \bigwedge_{\substack{-p(n)\leq i\leq p(n)\\ a\in A}} (h_{i,t}\vee(c_{i,a,t}\equiv c_{i,a,t+1}))$$

hat Länge $O(p(n))$.

Erfüllt φ den Ausdruck

$$CONF(V_t)\wedge CONF(V_{t+1})\wedge D_t,$$

so erfüllt φ insbesondere alle Ausdrücke $h_{i,t}\vee(c_{i,a,t}\equiv c_{i,a,t+1})$. Da φ $CONF(V_t)$ erfüllt, gilt $\varphi(h_{i_o,t})=1$ für genau ein i_o. Für alle anderen i und alle a muß deshalb $\varphi(c_{i,a,t})=\varphi(c_{i,a,t+1})$ gelten, d.h. die Bandinschriften von $k_t(\varphi)$ und $k_{t+1}(\varphi)$ unterscheiden sich höchstens an der Stelle, wo der Kopf in $k_t(\varphi)$ steht.

Wir brauchen noch einen Ausdruck, der sicherstellt, daß gilt:

(**): Die Inschrift von Zelle i_o sowie die Kopfposition von $k_{t+1}(\varphi)$ geht aus $k_t(\varphi)$ durch einen Schritt der Maschine M gemäß der Übergangsrelation δ hervor.

Sei

$$\delta_{iazt} = \delta^1_{iazt} \vee \delta^2_{iazt}$$

$$\delta^1_{iazt} = \overline{c}_{i,a,t} \vee \overline{h}_{i,t} \vee \overline{s}_{z,t}$$

$$\delta^2_{iazt} = \bigvee_{(z,a,q,b,R)\in\delta} c_{i,b,t+1} h_{i+1,t+1} s_{q,t+1}$$

$$\vee \bigvee_{(z,a,q,b,L)\in\delta} c_{i,b,t+1} h_{i-1,t+1} s_{q,t+1}$$

$$\vee \bigvee_{(z,a,q,b,N)\in\delta} c_{i,b,t+1} h_{i,t+1}, s_{q,t+1}.$$

Der Ausdruck

$$\delta_t = \bigwedge_{\substack{-p(n)\le i\le p(n)\\ a\in A\\ z\in Z}} \delta_{iazt}$$

hat Länge $O(p(n))$, und es gilt

Lemma 6.23: Erfüllt φ den Ausdruck

$$CONF(V_t) \wedge CONF(V_{t+1}) \wedge D_t \wedge \delta_t,$$

so gilt $k_t(\varphi) \vdash k_{t+1}(\varphi)$.

Beweis: Sei $\varphi(h_{i,t})=1$, $\varphi(c_{i,a,t})=1$ und $\varphi(s_{i,t})=1$. Es folgt $\varphi(\delta^1_{iazt})=0$.

Da φ insbesondere δ_t erfüllt, muß $\varphi(\delta^2_{iazt})=1$ gelten, also muß φ eines der Produkte in δ^2_{iazt} erfüllen.

Sei $\varphi(h_{i',t+1})=1$, $\varphi(c_{i,b,t})=1$ und $\varphi(s_{q,t+1})=1$. Es folgt $i'=i-1$ oder $i'=i$ oder $i'=i+1$, da sonst keines der Produkte in δ^2_{iazt} erfüllt wäre.

Ist $i'=i+1$, muß $(z,a,b,q,R)\in\delta$ gelten.

Ist $i'=i$, muß $(z,a,b,q,N)\in\delta$ gelten,

ist $i'=i-1$, muß $(z,a,b,q,L)\in\delta$ gelten.

Daraus folgt aber (**). □

Der Ausdruck $D_t \wedge \delta_t$ ist nicht in konjunktiver Normalform.

Der Ausdruck

$$"V_t \vdash V_{t+1}"$$

entstehe aus $D_t \wedge \delta_t$, indem man die Ausdrücke der Form $z \vee (x \equiv y)$ in D_t und die Ausdrücke δ_{iazt} in δ_t nach Satz 6.19 durch ihre vollständige konjunktive Normalform ersetzt. Da die Anzahl der Variablen in diesen Ausdrücken nicht von n abhängt, verlängert sich dadurch der Ausdruck nur um einen konstanten Faktor.

Der Ausdruck

$$A(w) = \bigwedge_{0 \le i < n} c_{i,w_{i+1},0} \wedge \bigwedge_{\substack{-p(n) \le i \le 0 \\ n \le i \le p(n)}} c_{i,B,0} \wedge h_{0,0} \wedge s_{z_o,0}$$

hat Länge $O(p(n))$, und φ erfüllt $A(w)$ genau dann, wenn $k_o(\varphi) = B^{p(n)} z_o w B^{p(n)-n+1}$ gilt.

Sei schließlich

$$F(w) = A(w) \wedge \bigwedge_{0 \le t \le p(n)} CONF(V_t) \wedge \bigwedge_{0 \le t < p(n)} "V_t \vdash V_{t+1}" \wedge s_{z_+,p(n)}.$$

$F(w)$ hat Länge $O(p^3(n))$.

Ist $\varphi(F(w))=1$, so ist nach dem oben Gesagten $(k_o(\varphi),\ldots,k_{p(n)}(\varphi))$ eine akzeptierende Berechnung von M angesetzt auf w, also $w \in L$.

Sei andererseits $w \in L$ und $(k_o,\ldots,k_{p(n)})$ eine akzeptierende Berechnung von M angesetzt auf w. Man verifiziert sofort, daß die folgende Belegung $F(w)$ erfüllt:

$$\varphi(c_{i,a,t}) = \begin{cases} 1 & \text{falls a der Inhalt von Zelle i in } k_t \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

$$\varphi(h_{i,t}) = \begin{cases} 1 & \text{falls der Kopf in } k_t \text{ auf Zelle i steht} \\ 0 & \text{sonst} \end{cases}$$

$$\varphi(s_{z,t}) = \begin{cases} 1 & \text{falls z der Zustand in } k_t \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Also ist $F(w)$ genau dann erfüllbar, wenn $w \in L$ gilt. Da in $F(w)$ insgesamt $O(p(n))$ Variablen vorkommen, hat die Kodierung von $F(w)$ die Länge $O(p^3(n) \log n)$, und es ist nicht schwer, sondern nur mühsam, eine polynomiell zeitbeschränkte deterministische Maschine anzugeben, aus w die Kodierung von $F(w)$ erzeugt.

6.2.3. Weitere NP-vollständige Probleme

Wir präsentieren ein graphentheoretisches NP-vollständiges Er-

kennungsproblem.

Definition 6.24: Ein ungerichteter Graph wird spezifiziert durch ein Paar $G=(V,E)$. V ist eine Menge und heißt Menge der Knoten von G. $E\subseteq\{\{u,v\}\mid u,v\in V\}$ heißt Menge der Kanten von G. G heißt endlich, falls V endlich ist.

Endliche ungerichtete Graphen kann man aufmalen, indem man die Knoten als Punkte darstellt und für jede Kante $\{u,v\}\in E$ die Punkte u und v mit einem Strich verbindet. Für $\{u,v\}\in E$ sagt man auch: u und v sind benachbart.

Ungerichtete Graphen kann man ähnlich wie gerichtete Graphen als Zeichenreihen kodieren. Anstelle der Vorgängerlisten (vgl. § 5.5.1) bildet man für jeden Knoten die Liste der zu ihm benachbarten Knoten.

Definition 6.25: Ein ungerichteter Graph $G=(V,E)$ hat einen vollständigen Subgraph mit k Knoten (k-Clique), falls es eine Menge von Knoten $K\subseteq V$ gibt, so daß $|K|=k$ und $\{u,v\}\in E$ für alle $u,v\in K$ gilt.

Satz 6.26: Die Sprache

CLIQUE = {(Repräsentation eines Graphen $(G,bin(k))\mid$ G hat eine k-Clique}

ist NP-vollständig.

Beweis: Man kann leicht eine polynomiell zeitbeschränkte nichtdeterministische Maschine angeben, die angesetzt auf die Repräsentation eines Graphen $G=(V,E)$ und $bin(k)$ eine Teilmenge K von k Knoten rät und dann verifiziert, daß in G alle Kanten zwischen Knoten aus K vorhanden sind, also CLIQUE $\in$ NP. Um SAT$\leq_p$CLIQUE zu zeigen, ordnen wir jedem Boole'schen Ausdruck A in konjunktiver Normalform mit k Klauseln einen Graphen $G_A=(V_A,E_A)$ zu, der genau dann eine k-Clique hat, wenn A erfüllbar ist.

Sei

$$A = K_1\wedge\ldots\wedge K_k$$

$$K_i = (y_{i1}\vee\ldots\vee y_{in_i})$$

$$y_{ij}\in\{x_1,x_2,\ldots\}\cup\{\overline{x}_1,\overline{x}_2,\ldots\}.$$

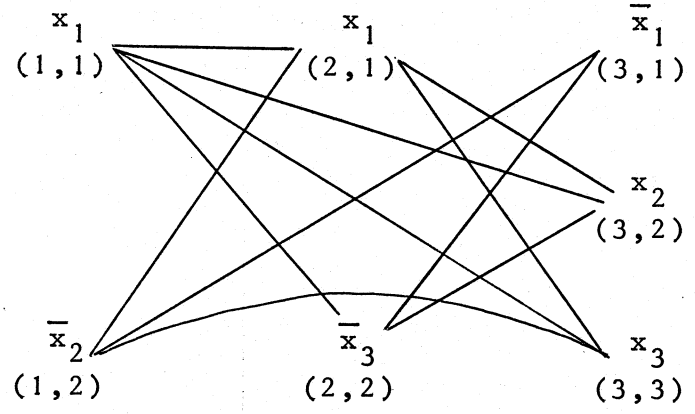
Dann setzen wir

$$V_A = \{(i,j)\mid i\in\{1,\ldots,k\},j\in\{1,\ldots,n_i\}\}\quad\text{und}$$

$$E_A = \{\{(i,j),(i',j')\} \mid i \neq i' \text{ und } y_{ij} \neq \overline{y}_{i'j'}\}$$

Beispiel 6.27: $A = (x_1 \vee \overline{x}_2)(x_1 \vee \overline{x}_3)(\overline{x}_1 \vee x_2 \vee x_3)$

$G_A =$



Über jedem Knoten (i,j) ist y_{ij} notiert.

Sei φ eine Belegung mit $\varphi(A)=1$. Dann gibt es in jeder Klausel K_i ein $y_{ij(i)}$ mit $\varphi(y_{ij(i)})=1$. Für alle i und i' folgt: $y_{ij(i)} \neq \overline{y}_{i'j(i')}$. Also bildet $\{(1,j(1)),\ldots,(k,j(k))\} \subseteq V_A$ eine k-Clique in G_a.

Sei andererseits $K=\{(i_1,j_1),\ldots,(i_k,j_k)\}$ eine k-Clique in G_A. Da keine Knoten (i,j) und (i',j') mit $y_{ij}=\overline{y}_{i'j'}$ mit einer Kante verbunden sind, gibt es eine Belegung φ mit $\varphi(y_{i_1j_1})=\ldots=\varphi(y_{i_kj_k})=1$. Da keine Knoten mit gleicher erster Komponente mit Kanten verbunden sind, folgt $\{i_1,\ldots,i_k\}=\{1,\ldots,k\}$. Es folgt $\varphi(K_i)=1$ für jede Klausel K_i und damit $\varphi(A)=1$. Hat der Ausdruck A die Länge n, so hat der Graph G_A höchstens n Knoten und n^2 Kanten, und man kann leicht eine polynomiell zeitbeschränkte Maschine angeben, die aus der Kodierung von A die Repräsentation von G_A erzeugt. □

Man kennt zur Zeit schätzungsweise 200 in nichttrivialer Weise verschiedene NP-vollständige Erkennungsprobleme aus allen möglichen Gebieten der Mathematik. Der Nachweis der NP-Vollständigkeit einer neuen Sprache L verläuft stets nach dem Muster von Satz 6.26. "L∈NP" ist meistens trivial und "L ist NP-hart" erhält man durch polynomielle Reduktion von L auf irgendein bereits bekanntes NP-vollständiges Problem (nicht notwendig SAT). Einige weitere Beispiele hierfür sind in den Übungen. Sammlungen von NP-vollständigen Problemen findet man in [2,36,42,69].

Einige besonders interessante NP-vollständige Sprachen werden in [16,17,4o] präsentiert.

Aus Lemma 6.1o folgt, daß alle NP-vollständigen Probleme polynomiell äquivalent, d.h. im wesentlichen gleich schwer sind. Dies ist der für die Praxis relevante Aspekt dieser Ergebnisse. Die Einführung des (nichtrealistischen) Konzepts nichtdeterministischer Maschinen erweist sich auch als beweistechnisch sehr nützlich, da beim Nachweis von $L \leq_p L'$ und $L' \leq_p L$ für NP-vollständige Sprachen eine der beiden Richtungen im allgemeinen trivial wird.

Übungen

1. Ist $SAT \in DTIME(t(n))$, so kann man in Zeit $O(nt(n))$ zu vorgegebenem Boole'schen Ausdruck in konjunktiver Normalform eine Belegung φ finden, die A erfüllt, falls es eine solche Belegung gibt.

   Ist $SAT \in DTIME(t(n))$, so gibt es eine $O(nt(n))$-zeitbeschränkte deterministische Turingmaschine, die, angesetzt auf die Kodierung eines Boole'schen Ausdrucks b in konjunktiver Normalform eine Belegung φ findet, die b erfüllt, falls es eine solche Belegung gibt.

2. Die folgenden Mengen können durch einfache Kodierung in Sprachen umgewandelt werden:

   3-SAT = {A|A hat höchstens 3 Literalen y_{ij} pro Klausel}.

   KNOTEN-ÜBERDECKUNG = {(G,k)| Im Graph G gibt es k Knoten, so daß jede Kante in G einen dieser Knoten berührt}.

   MENGEN-ÜBERDECKUNG = $\{(S_1,\ldots,S_n,k) \mid$ Die S_i sind endliche Mengen und es gibt $i_1,\ldots,i_k$, so daß $\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{i=1}^{n} S_i\}$.

   a) Alle diese Sprachen sind in NP.
   b) CLIQUE $\leq_p$ KNOTEN-ÜBERDECKUNG.
   c) KNOTEN-ÜBERDECKUNG $\leq_p$ MENGEN-ÜBERDECKUNG.
   d) SAT $\leq_p$ 3-SAT.

3. Für $L \subseteq A^*$ sei $\overline{L} = A^* \setminus L$. Sei $\overline{NP} = \{\overline{L} \mid L \in NP\}$.
   Man gebe $\overline{NP}$-vollständige Sprachen an.

4. $L = \{w \mid w \in \{0,1\}^*, w \neq 0\}$. Ist L nicht NP-vollständig, so folgt $P \neq NP$.

5. Eine Funktion $f:\{0,1\}^n \to \{0,1\}$ heißt Schaltfunktion.

Ein Schaltkreis wird spezifiziert durch ein Tupel (n,G,ρ):

$n \in \mathbb{N}$ gibt die Zahl der Eingänge an.

$G=(V,E)$ ist ein gerichteter zykelfreier Graph von Ingrad 2 mit n Quellen.

$\rho: V \to \{\wedge, \vee, \sim, x_1, x_2, \ldots, x_n\}$ ist eine Abbildung, so daß für alle Knoten $v \in V$ gilt:

$\rho(v) \in \{\wedge, \vee\}$ falls Ingrad $(v)=2$

$\rho(v) = \sim$ falls Ingrad $(v)=1$

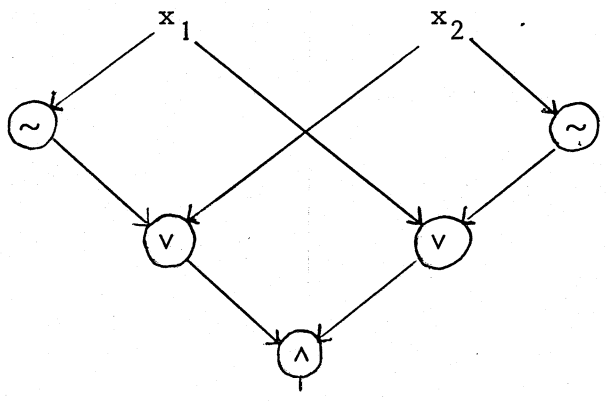
$\rho(v) \in \{x_1, \ldots, x_n\}$ falls Ingrad $(v)=0$.

Für Knoten $v \in V$ definieren wir induktiv die am Knoten v berechnete Funktion $res_v:\{0,1\}^n \to \{0,1\}$.

α_i	falls	$\rho(v)=x_i$
$\sim res_w(\alpha)$	falls	$\rho(v)=\sim$ und $(w,v) \in E$
$res_u(\alpha) \circ res_w(\alpha)$	falls	$\rho(v)=\circ \in \{\wedge,\vee\}$ und $(u,v),(w,v) \in E$.

Ein Schaltkreis berechnet $f:\{0,1\}^n \to \{0,1\}$ falls $f=res_v$ für irgendein $v \in V$. Ein Schaltkreis S berechnet $g:\{0,1\}^n \to \{0,1\}^m$, falls S alle Komponenten von g berechnet.

Beispiel: Der Schaltkreis



berechnet $f(\alpha) = \begin{cases} 1 & \text{falls } \alpha_1=\alpha_2 \\ 0 & \text{sonst.} \end{cases}$

a) Zu jeder Schaltfunktion $f:\{0,1\}^n \to \{0,1\}$ gibt es einen Schaltkreis, der f berechnet.

Hinweis: Man benutze Satz 6.19.

b) Man schätze die Anzahl der Bausteine dieses Schaltkreises nach oben ab.

6. Die Größe eines Schaltkreises ist die Anzahl der Knoten mit Ingrad 2. Die Komplexität C(f) einer Schaltfunktion f ist die Größe des kleinsten Schaltkreises, der f berechnet.

a) Mit allen Schaltkreisen der Größe c und n Eingängen kann man höchstens $c^{2(c+n)} \cdot 16^c \cdot n^n$ verschiedene Funktionen $f:\{0,1\}^n \to \{0,1\}^n$ berechnen.

b) $B_n = \{f \mid f:\{0,1\}^n \to \{0,1\}\}$
$|B_n| = ?$

c) $\max\{C(f) \mid f \in B_n\} \geq ?$

7. Für $L \subseteq A^*$ sei $\chi_L^n:\{0,1\}^n \to \{0,1\}$ die charakteristische Funktion von $L \cap \{0,1\}^n$.

a) Ist $M=(A,Z,\delta,z_0,z_+,z_-)$ eine deterministische simultan $t(n)$-zeitbeschränkte und $s(n)$-bandbeschränkte Einbandturingmaschine, so ist $C(\chi^n_{L(M)}) \leq O(s(n) \cdot t(n))$.

Hinweis: Die Funktionen $\beta:Z \to \{0,1\}^{\lceil \log|Z| \rceil}$ und $\gamma:A \to \{0,1\}^{\lceil \log|A| \rceil}$ seien injektiv.

Man baue einen Schaltkreis für $\chi^n_{L(M)}$ unter anderem aus $s(n) \cdot t(n)$ Kopien eines Schaltkreises für die folgende Funktion:

$f:\{0,1\}^{\lceil \log|Z| \rceil + \lceil \log|A| \rceil} \to \{0,1\}^{\lceil \log|Z| \rceil + \lceil \log|A| \rceil + 3}$

$$f(\beta(z),\gamma(a)) = \begin{cases} (\beta(z'),\gamma(c),1,0,0) & \text{falls } \delta(z,a)=(z',c,L) \\ (\beta(z'),\gamma(c),0,1,0) & \text{falls } \delta(z,a)=(z',c,N) \\ (\beta(z'),\gamma(c),0,0,1) & \text{falls } \delta(z,a)=(z',c,R). \end{cases}$$

b) Gilt für irgendeine Sprache $L \in NP \cap \{0,1\}^*$: $\liminf n^k / C(\chi^n_L) = 0$ für alle k, so ist $P \neq NP$.

8. a) Sei M eine "oblivious" (vergl. Übung 6 aus Kapitel 4.2) $t(n)$-zeitbeschränkte deterministische Mehrband-Turingmaschine, $L(M) \subseteq \{0,1\}^*$.

Dann ist $C(\chi^n_{L(M)}) \leq O(t(n))$.

b) Ist M eine deterministische $t(n)$-zeitbeschränkte Mehrbandturingmaschine und $L(M) \subseteq \{0,1\}^*$, so ist

$$C(\chi^n_{L(M)}) = O(t(n)\log t(n)).$$

c) Gilt für irgendeine Sprache $L\in NTIME(n)\cap\{0,1\}^*$:

$\liminf n \log n/C(\chi^n_L)=0$, so ist $DTIME(n)\subsetneq NTIME(n)$.

9. Die Sprache
$L=\{(s,\alpha)\mid \alpha\in\{0,1\}^*$, s ist Kodierung eines Schaltkreises mit $|\alpha|$ Eingängen und einem Ausgang v, und es gilt $res_v(\alpha)=1\}$
ist P-vollständig bezüglich $\leq_{logtape}$.

Offene Fragen

a) Übungen 7 und 8 motivieren die Untersuchung der Komplexität von Schaltfunktionen und verwandter Fragestellungen. Einen Überblick hierüber gibt [59]. Das Herleiten von unteren Schranken für die Komplexität von Schaltfunktionen scheint jedoch nicht leicht zu sein. So ist es z.Zt. unbekannt, ob es Sprachen $L\in NP\cap\{0,1\}^*$ gibt, so daß $C(\chi^n_L)>3n$ gilt.

b) NP = $\overline{NP}$?

6.3. PTAPE-vollständige Probleme

Wir definieren die Menge der Boole'schen Aussagen und zeigen, daß das Problem, die Gültigkeit einer solchen Aussage zu entscheiden, PTAPE-vollständig ist. Hiermit läßt sich die Komplexität kombinatorischer Spiele klassifizieren.

6.3.1. Boole'sche Prädikate

Definition 6.28: Die Menge der Boole'schen Prädikate ist die kleinste Menge M, für die gilt:

1) Ist b ein Boole'scher Ausdruck, so ist $b\in M$.
2) Sind A und $b\in M$, so auch $(\sim A),(A\wedge B),(A\vee B)$.
3) Ist $A\in M$ und ist x eine Variable, so sind $(\exists xA)$ und $(\forall xA)\in M$.

Beispiel 6.29: $((\exists x_1((\sim x_1)\vee x_2))\vee(\forall x_2(x_2\wedge x_3)))$ ist ein Boole'sches Prädikat.

Boole'sche Prädikate sind Zeichenreihen über einem unendlichen Alphabet $A=\{\sim,\wedge,\vee,\exists,\forall,(,)\}\cup V$. Die Länge eines Boole'schen Prädikats ist die Anzahl von Zeichen in dieser Zeichenreihe. Die Länge des Prädikats in Beispiel 6.29 ist 24. Um die Schreibweise zu

vereinfachen, lassen wir oft Klammern weg. Es gelten dann die üblichen Prioritätsregeln (vergl. § 2.2.3).

Definition 6.3o: x sei eine Variable.

1) Ist A ein Boole'scher Ausdruck und kommt x in A vor, so kommt x in A frei vor.

2) Ist A ein Boole'sches Prädikat und kommt x in A vor, so kommt x in $(\exists xA)$ und in $(\forall xA)$ gebunden vor.

3) Sind A und B Boole'sche Prädikate und kommt x in A oder B frei bzw. gebunden vor, so kommt x in $(A\wedge B)$ und $(A\vee B)$ frei bzw. gebunden vor. Kommt x in A frei bzw. gebunden vor, so auch in $(\sim A)$.

In Beispiel 6.29 kommt x_1 gebunden vor, x_3 kommt frei vor, x_2 kommt frei und gebunden vor.

In Zukunft bezeichnet $A(x_1,\ldots,x_s)$ ein Boole'sches Prädikat, in dem genau die Variablen $x_1,\ldots,x_s$ frei vorkommen. Ein Boole'sches Prädikat, in dem keine Variable frei vorkommt, heißt Boole'sche Aussage.

Definition 6.31: Sei A ein Boole'sches Prädikat, in dem die Variablen $x_1,\ldots,x_r$ frei vorkommen, und $\alpha_1,\ldots,\alpha_s\in\{0,1\}UV, s\leq r$. Dann ist $A|_{x_1=\alpha_1\ldots x_s=\alpha_s}$ das Prädikat, das man erhält, wenn man für $1\leq j\leq s$ jedes freie Vorkommen von x_j in A durch α_j ersetzt.

Ist A das Prädikat aus Beispiel 6.29, so ist

$$A|_{x_2=0} = (\exists x_1((\sim x_1)\vee 0))\vee(\forall x_2(x_2\wedge x_3))).$$

Für $A(x_1,\ldots,x_r)|_{x_1=\alpha_1\ldots x_s=\alpha_s}$ schreiben wir auch $A(\alpha_1,\ldots,\alpha_s,x_{s+1}\ldots x_r)$. Ist φ eine Belegung und $A(x_1,\ldots,x_s)$ ein Boole'sches Prädikat, so wird $\varphi(A)=A(\varphi(x_1),\ldots,\varphi(x_s))$ definiert.

Definition 6.32: Ein Boole'scher Ausdruck A gilt (identisch), falls $\varphi(A)=1$ für alle Belegungen φ gilt (z.B. gelten die Ausdrücke $x_1\vee\overline{x}_1$ und $((0\vee 1)\wedge 1)$). Ist A ein Boole'sches Prädikat und x eine Variable, so gilt $\exists xA$, falls $A|_{x=0}$ oder $A|_{x=1}$ oder beide gelten. $\forall xA$ gilt, falls $A|_{x=0}$ und $A|_{x=1}$ gelten. Sind A und B Boole'sche Prädikate, so gilt $A\wedge B$, falls A und B beide gelten. $A\vee B$ gilt, falls A oder B oder beide gelten. $\sim A$ gilt, falls A nicht gilt. Ein Boole'sches Prädikat A heißt erfüllbar, falls es eine Belegung

φ gibt, so daß $\varphi(A)$ gilt.

Beispielsweise gilt $\exists x_1 \ldots \exists x_n A(x_1 \ldots x_n)$ genau dann, wenn $A(x_1 \ldots x_n)$ erfüllbar ist. Die Gültigkeit von (passend kodierten) Boole'schen Aussagen zu entscheiden, ist deshalb sicher NP-hart. Wir zeigen im folgenden, daß schon ein Spezialfall dieses Problems sogar PSPACE-vollständig ist. Wir leiten zuvor einige einfache Ergebnisse über Boole'sche Prädikate her.

Zwei Boole'sche Prädikate A und B heißen äquivalent ($A \equiv B$), falls $\varphi(A)$ genau dann gilt, wenn $\varphi(B)$ gilt. Man verifiziert sofort, daß hierdurch eine Äquivalenzrelation erklärt wird. Ferner gilt

Lemma 6.33: Die Menge der Boole'schen Prädikate zusammen mit den Operationen $\wedge, \vee, \sim$ formt eine Boole'sche Algebra, d.h. Lemma 6.16 bleibt gültig, wenn man Boole'sche Ausdrücke durch Boole'sche Prädikate und = durch $\equiv$ ersetzt.

Beweis: a) $\varphi(A \vee B)$ gilt $\iff$ $\varphi(A)$ oder $\varphi(B)$ gilt $\iff$ $\varphi(B \vee A)$ gilt. $\varphi(A \wedge B)$ gilt ... usw. □

Lemma 6.34: Sind A,B und C Boole'sche Prädikate und ist $A \equiv B$, so folgt:

a) $\exists x A \equiv \exists x B$

b) $\forall x A \equiv \forall x B$

c) $(\sim A) \equiv (\sim B)$

d) $A \vee C \equiv B \vee C$

e) $A \wedge C \equiv B \wedge C$.

Beweis:

a) Sei $A = A(x, x_1, \ldots, x_s)$, $B = B(x, y_1, \ldots, y_r)$ und φ eine Belegung. Es folgt:

$\varphi(\exists x A)$ gilt $\iff$

$\exists x A(x, \varphi(x_1), \ldots, \varphi(x_s))$ gilt $\iff$

$A(\alpha, \varphi(x_1), \ldots, \varphi(x_s))$ gilt für $\alpha=0$ oder $\alpha=1$ $\iff$

$\varphi_\alpha(A)$ gilt für $\alpha=0$ oder $\alpha=1$, wobei

$$\varphi_\alpha(z) = \begin{cases} \alpha & \text{für } z = x \\ \varphi(z) & z \in V \setminus \{x\} \end{cases}$$

Ebenso folgt:

$\varphi(\exists x B)$ gilt $\iff$ $\varphi_\alpha(B)$ gilt für $\alpha=0$ oder $\alpha=1$.

Nach Voraussetzung gilt aber $\varphi_\alpha(A)$ genau dann, wenn $\varphi_\alpha(B)$ gilt.

b)-e): Übung. □

Lemma 6.35: A und B seien Boole'sche Ausdrücke, und die Variable x komme in B nicht vor. Dann gilt:

a) $Bo\exists xA \equiv \exists x(BoA)$ $\quad o \in \{\wedge,\vee\}$

b) $Bo\forall xA \equiv \forall x(BoA)$ $\quad o \in \{\wedge,\vee\}$

c) $\sim\exists xA \equiv \forall x(\sim A)$

d) $\sim\forall xA \equiv \exists x(\sim A)$

Beweis:

a) Sei $A=A(x,x_1,\ldots,x_s)$, $B=B(y_1,\ldots,y_r)$ und φ eine Belegung. Es folgt:

$\varphi(Bo\exists xA)$ gilt $\Longleftrightarrow$

$B(\varphi(y_1),\ldots,\varphi(y_r))oA(\alpha,\varphi(x_1),\ldots,\varphi(x_s))$ gilt für $\alpha=0$ oder $\alpha=1 \Longleftrightarrow$

$\exists xB(\varphi(y_1),\ldots,\varphi(y_r))oA(x,\varphi(x_1),\ldots,\varphi(x_s))$ gilt $\Longleftrightarrow$

$\varphi(\exists x(BoA))$ gilt.

b)-d): Übung. □

Lemma 6.36: A sei ein Prädikat, in dem die Variable x frei vorkommt, und z sei eine Variable, die nicht in A vorkommt, A' entstehe aus A, indem jedes freie Vorkommen von x in A durch z ersetzt wird. Dann gilt für $Q\in\{\exists,\forall\}$:

$$QxA \equiv QzA'.$$

Beweis: Übung. □

Ein Boole'sches Prädikat heißt in Pränex Normalform, falls es von der Form

$$Q_1x_1,\ldots,Q_sx_sA(x_1,\ldots,x_n)$$

ist, wobei $Q_i\in\{\exists,\forall\}$ gilt und A ein Boole'scher Ausdruck ist. Ist A überdies in konjunktiver Normalform, so heißt das Prädikat in Konjunktiver Pränex Normalform.

Satz 6.37: Zu jedem Boole'schen Prädikat der Länge n gibt es ein äquivalentes Prädikat der Länge n in Pränex Normalform.

Beweis durch Induktion über n. Ein Prädikat der Länge 1 ist eine Variable und somit in Pränex Normalform. Ist C ein Prädikat der Länge $n>1$, so liegt einer der folgenden Fälle vor:

a) C ist ein Boole'scher Ausdruck

b) $C=BoA, o\in\{\wedge,\vee\}$, A oder B enthält $\forall$ oder $\exists$

c) $C=\sim A$, A enthält $\forall$ oder $\exists$

d) $C=QxA$, $Q\in\{\exists,\forall\}$.

Im Fall a) ist nichts zu zeigen.

Im Fall b) seien

$$A' = Q_1x_1\ldots Q_sx_sD(x_1,\ldots,x_n) \text{ und}$$

$$B' = P_1y_1\ldots P_ry_rE(y_1,\ldots,y_m)$$

die Pränex-Normalform von A und B. $x_1',\ldots,x_s',y_1'\ldots y_r$ seien Variablen, die weder in A' noch B' vorkommen. Sei

$D' = D(x_1',\ldots,x_s',x_{s+1},\ldots,x_n)$, $A'' = Q_1x_1'\ldots Q_sx_s'D'$

$E' = E(y_1',\ldots,y_r',y_{r+1},\ldots,y_n)$, $B'' = P_1y_1'\ldots P_ry_r'E'$.

Nach Lemma 6.36 folgt $A'\equiv A''$ und $B'\equiv B''$, also

$C = BoA\equiv B'oA'\equiv B''oA''$ nach Lemma 6.34

$\equiv Q_1x_1'\ldots Q_sx_s'(B''oD')$ nach Lemma 6.35

$\equiv Q_1x_1'\ldots Q_sx_s'(D'oB'')$ nach Lemma 6.33 und Lemma 6.34

$\equiv Q_1x_1'\ldots Q_sx_s'P_1y_1'\ldots P_ry_r'(D'oE')$ nach Lemma 6.35.

Fall c) und d): Übung. □

Lemma 6.38: Es gibt eine Konstante c, so daß gilt:

Zu jedem Boole'schen Ausdruck $A(x_1,\ldots,x_s)$ der Länge n gibt es ein äquivalentes Prädikat

$$B=\exists y_1\ldots\exists y_rK(x_1,\ldots,x_s,y_1,\ldots,y_r)$$

der Länge c·n, wobei K ein Boole'scher Ausdruck in konjunktiver Normalform ist.

Beweis durch Induktion über n. Für n=1 ist nichts zu zeigen. Sei n>1 und A nicht in konjunktiver Normalform. Dann gibt es in A einen Teilausdruck der Form (yoz) mit

$$y,z\in\{x_1,\ldots,x_s\}\cup\{\overline{x}_1,\ldots,\overline{x}_s\},\ o\in\{\wedge,\vee\}.$$

A' entstehe aus A, indem man (yoz) durch y_1 ersetzt. A'' sei die konjunktive Normalform von $y_1\equiv(yoz)$. Man verifiziert leicht $A\equiv\exists y_1(A''\wedge A')$. Anwendung der Induktionsvoraussetzung auf A' und Anwendung von Lemma 6.35 liefert das gewünschte Prädikat. Die Konstante c wird bestimmt durch die Länge der konjunktiven Normalformen von $y_1\equiv(y\wedge z)$ und $y_1=(y\vee z)$. □

Aus Satz 6.37 und Lemma 6.38 und Lemma 6.34 folgt:

Satz 6.39: Zu jedem Boole'schen Prädikat der Länge n gibt es ein äquivalentes Prädikat der Länge O(n) in konjunktiver Pränex Normalform.

Wir beobachten abschließend, daß man aus den Beweisen von Satz 6.37 und Lemma 6.38 leicht ein Rechenverfahren mit polynomieller Laufzeit gewinnen kann, das, angesetzt auf die Kodierung eines beliebigen Prädikats, die Kodierung der konjunktiven Pränex Normalform dieses Prädikats liefert.

6.3.2. Das Entscheidungsproblem für Boole'sche Aussagen

Satz 6.4o: Die Sprache

KPNF = {w|w ist Kodierung einer gültigen Boole'schen Aussage in Konjunktiver Pränex Normalform}

ist PTAPE-vollständig.

Der Beweis kombiniert Techniken aus den Beweisen der Sätze von Savitch (5.13) und Cook (6.21).

Lemma 6.41: $KPNF \in DTAPE(n)$.

Beweis: Definition 6.32 legt das folgende rekursive Entscheidungsverfahren nahe: Um für $w_1,\ldots,w_{i-1} \in \{0,1\}$ die Gültigkeit von Aussagen der Form

$$Q_i x_i \ldots Q_n x_n A(w_1,\ldots,w_{i-1},x_i,\ldots,x_n)$$

zu testen, teste man für $w_i=0$ und $w_i=1$ nacheinander auf dem gleichen Speicherplatz die Gültigkeit von

$$Q_{i+1} x_{i+1} \ldots Q_n x_n A(w_1,\ldots,w_i,x_{i+1},\ldots,x_n)$$

und entscheide dann gemäß Definition 6.32. Wir geben eine Art an, dieses Verfahren so zu organisieren, daß für den hierfür benutzten Speicherplatz s(i,n) gilt:

$$s(i,n) \leq s(i+1,n)+1$$

Also

$$s(0,n) \leq s(n,n)+n.$$

Da man Ausdrücke $A(w_1,\ldots,w_n)$, in denen keine Variable vorkommt, auf Band n auswerten kann, folgt die Behauptung.

Das folgende Verfahren hat mehrere Phasen: Expandieren, (+)-Schrumpfen und (-)-Schrumpfen. Zwischen diesen Phasen wird während

des Ablaufs einer Rechnung hin- und hergesprungen. Das Verfahren manipuliert eine Zeichenreihe w'=#w, w∈{0,1}*, und startet in der Phase "Expandieren" mit w'=#. Das Verfahren ist so organisiert, daß gilt:

(*) Wird das Verfahren in der Expansionsphase mit der Zeichenreihe $w'=\#w_1\ldots w_i$ gestartet, so bearbeitet das Verfahren schließlich diese Zeichenreihe in einer Schrumpfphase, und zwar in der Phase (+)-Schrumpfen, falls

$$Q_{i+1}x_{i+1}\ldots Q_n x_n A(w_1,\ldots,w_i,x_{i+1},\ldots,x_n)$$

gilt, andernfalls in der Phase (-)-Schrumpfen.

Expandieren: Ist $|w|=n$ und $A(w_1,\ldots,w_n)\equiv 1$, wird in die Phase (+)-Schrumpfen gegangen.

Ist $|w|=n$ und $A(w_1,\ldots,w_n)\equiv 0$, wird in die Phase (-)-Schrumpfen gegangen.

Ist $|w|<n$, wird die Zeichenreihe w zu w0 verlängert und eine neue Expansionsphase begonnen.

(+)-Schrumpfen: Sei $w=w_1\ldots w_i$.

Ist $Q_i=\forall$ und $w_i=0$, wird w_i durch 1 ersetzt und eine Expansionsphase begonnen.

Ist $Q_i=\forall$ und $w_i=1$ oder $Q_i=\exists$, wird w_i gestrichen und in eine neue (+)-Schrumpfphase gegangen.

(-)-Schrumpfen: Sei $w=w_1\ldots w_i$.

Ist $Q_i=\exists$ und $w_i=0$, wird w_i durch 1 ersetzt und eine neue Expansionsphase begonnen.

Ist $Q_i=\exists$ und $w_i=1$ oder $Q_i=\forall$, wird w_i gestrichen und in eine neue (-)-Schrumpfphase gegangen.

Durch Induktion über i (der Induktionsanfang ist i=n, der Induktionsschluß ist von i auf i-1) zeigt man leicht, daß (*) für alle i gilt, und damit die Korrektheit des Verfahrens.

Die Aussage $Q_1x_1\ldots Q_nx_nA(x_1,\ldots,x_n)$ gilt, wenn das Verfahren schließlich # in der (+)-Schrumpfphase zu bearbeiten versucht. Der Bandverbrauch dieses Entscheidungsverfahrens ist $n+O(+|A(x_1,\ldots,x_n)|)=O(|A(x_1,\ldots,x_n)|)$. □

Lemma 6.42: KPNF ist PTAPE-hart.

Beweis: Sei L∈PTAPE. Dann gibt es ein Polynom p(n) und eine p(n)-bandbeschränkte deterministische 1-Band-Turingmaschine $M=(Z,A,\delta,z_o,z_+,z_-)$, die L akzeptiert. M sei so modifiziert, daß $\delta(z_+,a)=(z_+,a,N)$ für alle a∈A gilt.

Nach Satz 6.39 genügt es zu zeigen, daß man zu jedem w∈A* in polynomieller Zeit irgendeine Boole'sche Aussage f(w) generieren kann, die genau dann gilt, wenn w∈L gilt, d.h. wenn für irgendein t und irgendeine Konfiguration k'

$$B^{p(n)}z_o wB^{p(n)-n+1} \vdash_{p(n)}^{t} k'$$

gilt, wobei z_+ in k' vorkommt (die Notation " $\vdash_s^t$ " wurde in Definition 5.15 erklärt.

Es sei |w|=n).

Aus Lemma 4.4 folgt $t\leq|Z|\cdot p(n)\cdot|A|^{p(n)}$ falls M angesetzt auf w hält. Sei $\tau(n)=\lceil\log(|Z|\cdot p(n)\cdot|A|^{p(n)})\rceil=O(p(n))$. Wegen $\delta(z_+,a)=(z_+,a,N)$ für alle a∈A kann oben $t=2^{\tau(n)}$ angenommen werden. $V_o,V_1,\ldots$ seien Variablenmengen wie im Beweis von Satz 6.22. Für x∈ℕ geben wir induktiv Prädikate "$V_u \vdash_{p(n)}^{2^x} V_w$" an, so daß gilt:

(*) Eine Belegung φ erfüllt

$$CONF(V_u) \wedge CONF(V_w) \wedge "V_u \vdash_{p(n)}^{2^x} V_w"$$

genau dann, wenn

$$k_u(\varphi) \vdash_{p(n)}^{2^x} k_w(\varphi)$$

gilt.

Offenbar kann "$V_u \vdash V_w$" für "$V_u \vdash_{p(n)}^{2^o} V_w$" genommen werden. $\exists V_u$ bzw. $\forall V_u$ bezeichne eine Folge von $|V_u|$ Quantoren ∃ bzw. ∀, die über alle Variablen aus V_u quantifizieren. Die naheliegende induktive Definition wäre

$$"V_u \vdash_{p(n)}^{2^x} V_w" = \exists V_r(CONF(V_r)\wedge("V_u \vdash_{p(n)}^{2^{x-1}} V_r")\wedge("V_r \vdash_{p(n)}^{2^{x-1}} V_w")),$$

aber das würde zu einem Prädikat "$V_u \vdash_{p(n)}^{2^x} V_w$" der Länge mindestens 2^x führen. Wir geben eine kürzere Schreibweise für dieses Prädikat an.

"$V_u=V_w$" bezeichne das Prädikat

$$\bigwedge_{i,a}(c_{i,a,u}\equiv c_{i,a,w}) \wedge \bigwedge_i (h_{i,u}\equiv h_{i,w}) \wedge \bigwedge_z (s_{z,u}=s_{z,w}).$$

Dann sei

$$"V_u \mid\!\!\frac{2^x}{p(n)}\ V_w" = \exists V_r(\mathrm{Conf}(V_r)\wedge\forall V_s\forall V_t E_{uwrst},$$

mit

$$E_{uwrst}=(\sim("V_u=V_s"\wedge"V_r=V_t"))\vee(\sim("V_r=V_s"\wedge"V_w=V_t"))\vee("V_s \mid\!\!\frac{2^{x-1}}{p(n)}\ V_t")).$$

Wir zeigen durch Induktion über x, daß (*) für alle x gilt.
(*) gilt offenbar für x=0.

Ist $\varphi:V\to\{0,1\}$ eine Belegung und $V'\supset V$, so heißt $\varphi':V'\to\{0,1\}$ eine <u>Fortsetzung</u> von φ, falls $\varphi'(x)=\varphi(x)$ für alle $x\in V$ gilt. Sei φ eine Belegung der Variablen aus V_u und V_w, und φ erfülle

$$\mathrm{CONF}(V_u)\wedge\mathrm{CONF}(V_w)\wedge"V_u \mid\!\!\frac{2^x}{p(n)}\ V_w".$$

Nach Konstruktion von $"V_u \mid\!\!\frac{2^x}{p(n)}\ V_w"$ gibt es eine Fortsetzung φ_r von φ auf die Variablen aus V_r, so daß φ_r das Prädikat $\mathrm{CONF}(V_r)$ erfüllt, und für jede Fortsetzung φ_{rst} von φ_r auf die Variablen aus V_r und V_s gilt: φ_{rst} erfüllt E_{uwrst}.
Setzt man insbesondere φ_r so fort, daß φ_{rst} das Prädikat $"V_u=V_s"\wedge"V_r=V_t"$ erfüllt, so folgt, daß φ_{rst} auch

$$\mathrm{CONF}(V_s)\wedge\mathrm{CONF}(V_t)\wedge"V_s \mid\!\!\frac{2^{x-1}}{p(n)}\ V_t"$$

erfüllt. Weiter folgt

$$k_s(\varphi_{rst})=k_u(\varphi_{rst})=k_u(\varphi), k_t(\varphi_{rst})=k_r(\varphi_{rst})=k_r(\varphi_r),$$

und nach Induktionssetzung

$$k_s(\varphi_{rst}) \mid\!\!\frac{2^{x-1}}{p(n)}\ k_t(\varphi_{rst}),\ \text{also}\ k_u(\varphi) \mid\!\!\frac{2^{x-1}}{p(n)}\ k_r(\varphi_r).$$

Setzt man φ_r so fort, daß φ_{rst} das Prädikat $"V_r=V_s"\wedge"V_w=V_t"$ erfüllt, folgt ebenso $k_r(\varphi_r) \mid\!\!\frac{2^{x-1}}{p(n)}\ k_w(\varphi)$, also $k_u(\varphi) \mid\!\!\frac{2^x}{p\,n)}\ k_w(\varphi)$.
Gilt andererseits $k_u(\varphi) \mid\!\!\frac{2^x}{p(n)}\ k_w(\varphi)$, so gibt es eine Fortsetzung φ_r von φ auf die Variablen von V_r, so daß

$$k_u(\varphi_r)=k_u(\varphi) \mid\!\!\frac{2^{x-1}}{p(n)}\ k_r(\varphi_r) \mid\!\!\frac{2^{x-1}}{p(n)}\ k_w(\varphi)=k_w(\varphi_r)$$

gilt. φ_r erfüllt sicher CONF(V_u), CONF(V_w) und CONF(V_r). φ_{rst} sei eine beliebige Fortsetzung von φ_r auf die Variablen aus V_s und V_t. Erfüllt φ_{rst} das Prädikat "$V_u=V_s$"$\wedge$"$V_r=V_t$", so folgt

$k_s(\varphi_{rst})=k_u(\varphi_{rst})$ und $k_r(\varphi_{rst})=k_t(\varphi_{rst})$, also $k_s(\varphi_{rst}) \mid\!\frac{2^{x-1}}{p(n)}\; k_r(\varphi_{rst})$.

Somit erfüllt φ_{rst} nach Induktionsvoraussetzung

$$"V_s \mid\!\frac{2^{x-1}}{p(n)}\; V_t"$$

und damit E_{uwrst}.

Ebenso schließt man, falls φ_{rst} das Prädikat "$V_r=V_s$"$\wedge V_w=V_t$" erfüllt. In allen anderen Fällen erfüllt φ_{rst} das Prädikat $(\sim("V_u=V_s"\wedge"V_r=V_t"))\vee(\sim("V_r=V_s"\wedge"V_w=V_t"))$. Also erfüllt φ

$$CONF(V_u)\wedge CONF(V_w)\wedge"V_u \mid\!\frac{2^{x}}{p(n)}\; V_w".$$

Ist A(w) wie im Beweis von Satz 6.21 erklärt, so gilt nach (*) $w\in L$ genau dann, wenn das Prädikat

$$F'(w)=A(w)\wedge CONF(V_o)\wedge CONF(V_1)\wedge"V_o \mid\!\frac{2^{\tau(n)}}{p(n)}\; V_1"\wedge s_{z_+,1}$$

erfüllbar ist bzw. genau dann, wenn die Aussage

$$F(w)=\exists V_o \exists V_1 \exists s_{z_+,1} F'(w)$$

gilt.

Für die Länge L(x) der Prädikate "$V_u \mid\!\frac{2^{x}}{p(n)} V_w$" gilt offenbar :

$$L(0) = O(p(n))$$

$$L(x) \leq c\cdot p^2(n)+L(x-1)$$

für eine passende Konstante c, also

$$L(x) \leq cxp^2(n)+O(p(n)).$$

Insbesondere $L(\tau(n))\leq O(p^3(n))$. Es folgt, daß F(w) Länge $O(p^3(n))$ hat. Kodiert man alle auftretenden Variablen in ein endliches Alphabet, erhält man eine Zeichenreihe der Länge $O(p(n)\log p(n))= O(p(n)\log n)$. Es ist nicht schwierig, sondern bloß mühsam, aus der obigen Konstruktion ein polynomiell zeitbeschränktes Rechenverfahren anzugeben, das aus w erst die Kodierung von F(w) und hieraus die Kodierung der konjunktiven Pränex Normalform von F(w) generiert. □

6.3.3. Weitere PTAPE-vollständige Probleme

Das folgende Kinderspiel erscheint auf den ersten Blick ziemlich geistlos: Zwei Spieler nennen abwechselnd Ländernamen. Der Name des Landes, das ein Spieler nennt, muß mit dem gleichen Buchstaben beginnen, mit dem der Name des zuvor vom Gegner genannten Landes aufhört. Kein Land darf zweimal genannt werden. Wer zuerst kein Land mehr nennen kann, hat verloren. Nimmt man die Buchstaben als Knoten und die Länder, deren Name mit a_1 anfängt und mit a_2 aufhört, als Kante von a_1 nach a_2, so hat man damit einen Sonderfall des Spiels Geographie, das wir im folgenden untersuchen:

Zwei Spieler A und B entfernen abwechselnd Kanten aus einem gerichteten Graphen. Ein Spieler darf nur eine Kante entfernen, die in einem Knoten beginnt, in dem die zuvor vom Gegenspieler entfernte Kante geendet hat. Verloren hat, wer als erster nicht mehr ziehen kann. Eine Spielsituation S dieses Spiels ist ein Paar $S=(G,v)$, wobei $G=(V,E)$ ein gerichteter Graph und $v\in V$ ist. $|E|$ heißt Größe der Spielsituation und ist eine obere Schranke für die Dauer des Spiels. Eine Situation $S=(G,v)$ hat eine Gewinnstrategie, wenn folgendes gilt: Ist Spieler x auf Graph G am Zug und muß eine Kante, die von v ausgeht, entfernen, so kann x in diesem und allen folgenden Zügen so spielen, daß er gewinnt, gleichgültig wie geschickt der Gegner spielt.

Satz 6.43: Die Sprache

GEO = {w|w ist Kodierung einer Spielsituation des Spiels Geographie, die eine Gewinnstrategie hat}

ist PTAPE-vollständig.

Wir zeigen zunächst

Lemma 6.44: $GEO\in PTAPE$.

Beweis: Sei $S=(G,v)$, $G=(V,E)$, eine Spielsituation; $v_1,\ldots,v_s$ sei die Menge der Nachfolger von v in G. Für $1\leq i\leq s$ sei

$$E_i=E\setminus\{v,v_i)\},\ G_i=(G,E_i) \text{ und } S_i=(G_i,v_i).$$

Offenbar gilt:

(*) S hat genau dann eine Gewinnstrategie, wenn für irgendein i S_i keine Gewinnstrategie hat.

S hat genau dann keine Gewinnstrategie, wenn für alle i S_i eine Gewinnstrategie hat.

Dies legt das folgende rekursive Entscheidungsverfahren nahe: Um zu testen, ob S=(G,v) eine Gewinnstrategie hat, zähle man in irgendeiner Ordnung alle Nachfolger v_i von v in G auf, teste nacheinander auf dem gleichen Platz für alle Spielsituationen S_i, ob S_i eine Gewinnstrategie hat, und entscheide dann nach (*). Dieses Verfahren kann so organisiert werden, daß für den Bandverbrauch B(n) zum Bearbeiten einer Spielsituation der Größe n gilt:

$$B(n) \leq O(\log n) + B(n-1),$$

also

$$B(n) \leq O(n \log n) + B(0).$$

Die Details der Organisation sind analog zu den Beweisen von Satz 5.13 und Lemma 6.41. □

Um zu zeigen, daß die Sprache GEO auch PTAPE-hart ist, untersuchen wir zuerst ein Belegungsspiel auf Boole'schen Aussagen $C=Q_1x_1,\ldots,Q_nx_nD(x_1,\ldots,x_n)$ in Pränex-Normalform. Im Verlauf des Spiels spezifizieren zwei Spieler A und B eine Belegung φ der Variablen $x_1,\ldots,x_n$. Ist $Q_1=\exists$, darf A $\varphi(x_1)$ wählen, sonst darf B $\varphi(x_1)$ wählen.

Hat ein Spieler $\varphi(x_i)$ gewählt und ist $Q_{i+1}=Q_i$, so wählt der gleiche Spieler $\varphi(x_{i+1})$, andernfalls kommt der Gegner zum Zug und darf $\varphi(x_{i+1})$ wählen. A gewinnt, falls schließlich $\varphi(D)=1$ ist. Offenbar wählt A $\varphi(x_i)$ für alle Variablen x_i mit $Q_i=\exists$ und B wählt $\varphi(x_i)$ für $Q_i=\forall$.

Man kann wieder in naheliegender Weise definieren, wann ein Spieler in diesem Spiel eine Gewinnstrategie hat.

Lemma 6.45: Für jede Boole'sche Aussage

$$C=Q_1x_1,\ldots,Q_nx_nD(x_1,\ldots,x_n)$$

in Pränex-Normalform gilt:

A hat im Belegungsspiel auf C eine Gewinnstrategie, wenn C gilt; andernfalls hat B eine Gewinnstrategie.

Beweis: Durch Induktion über n. Für n=0 ist nichts zu zeigen. Sei also $n\geq 1$ und C gelte. Ist $Q_1=\exists$, so gibt es ein $\alpha\in\{0,1\}$, so daß $C_\alpha=Q_2x_2\ldots Q_nx_nA(\alpha,x_2,\ldots,x_n)$ gilt. Wählt A $\varphi(x_1)=\alpha$, so hat A nach

Induktionsvoraussetzung eine Gewinnstrategie auf C_α. Ist $Q_1=\forall$, so gilt C_α für $\alpha=0$ und $\alpha=1$. Gleichgültig, wie B $\varphi(x_1)$ wählt, hat deshalb A nach Induktionsvoraussetzung eine Gewinnstrategie auf C_α. Ebenso zeigt man, daß B eine Gewinnstrategie hat, falls C nicht gilt. □

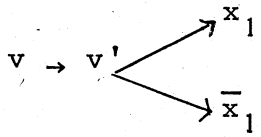
Lemma 6.46: GEO ist PTAPE-hart.

Beweis: Wir zeigen KPNF$\leq_p$GEO. Sei $C=Q_1x_1\ldots Q_nx_nD(x_1,\ldots,x_n)$ eine Boole'sche Aussage der Länge L in Konjunktiver Pränex-Normalform: Wir beschreiben eine Spielsituation $S(C)=(G,v)$ mit in L polynomieller Größe, die genau dann eine Gewinnstrategie hat, wenn A in dem Belegungsspiel auf C eine Gewinnstrategie hat.

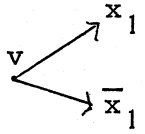
Sei $D=K_1\wedge\ldots\wedge K_s$, wobei die K_i Klauseln sind. OBdA kommt in keiner Klausel 0 oder 1 vor (andernfalls kann D in einfacher Weise so abgeändert werden, daß 0 und 1 nicht vorkommen).

$V\subseteq\{x_1,\ldots,x_n\}\cup\{\overline{x}_1,\ldots,\overline{x}_n\}\cup\{Q_i^j\mid 1\leq j\leq 3, 1\leq i\leq n\}\cup\{K_i\mid 1\leq i\leq s\}\cup\{v,v',D\}$.

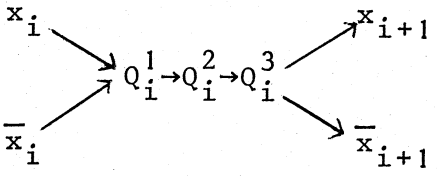
Die folgenden Kanten sind in G:



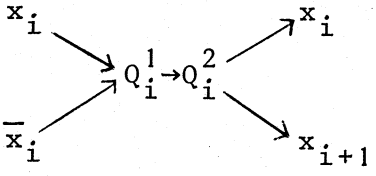
falls $Q_1=\forall$



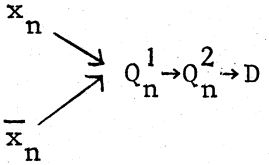
falls $Q_1=\exists$



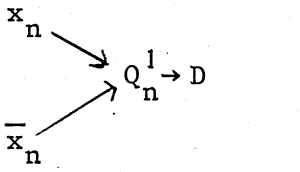
falls $Q_i=Q_{i+1}$



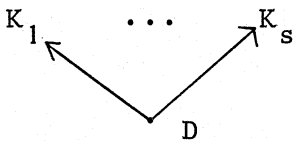
falls $Q_i\neq Q_{i+1}$



falls $Q_n=\forall$

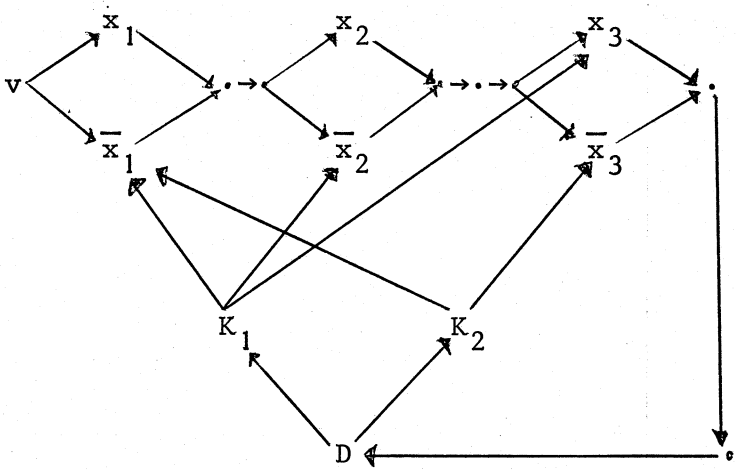


falls $Q_n=\exists$



$K_j \to \tilde{x}_i, \tilde{x}_i \in \{x_i, \bar{x}_i\}$ falls $\tilde{x}_i$ in Klausel j vorkommt.

Beispiel 6.47: $\exists x_1 \forall x_2 \forall x_3 (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3)$



Allgemeiner Spielverlauf: A sei der Spieler, der zuerst eine Kante, die von v ausgeht, entfernt. A und B entfernen zunächst Kanten auf einem Pfad von v nach D. Ist $Q_i=\exists$, so wählt A, ob der Pfad durch x_i oder $\bar{x}_i$ geht, sonst wählt B. Geht der Pfad durch $\tilde{x}_i \in \{x_i, \bar{x}_i\}$, so sagen wir: $\tilde{x}_i$ wurde gewählt.

B entfernt eine Kante von D zu einem K_j. Gibt es von diesem K_j eine Kante zu einem $\bar{x}_i$, das gewählt wurde, gewinnt A, sonst gewinnt B.

Lemma 6.48: Hat A eine Gewinnstrategie auf dem Belegungsspiel auf C, so hat A eine Gewinnstrategie auf S(C); andernfalls hat B eine Gewinnstrategie.

Beweis: A habe eine Gewinnstrategie für das Belegungsspiel auf C. Dann spielt A auf S(C) wie folgt:

Es seien bereits $\tilde{x}_1, \ldots, \tilde{x}_k$ gewählt, $\tilde{x}_i \in \{x_i, \bar{x}_i\}$, $1 \leq i \leq k$.

Für $i \leq k$ sei $\varphi(x_i) = \begin{cases} 1 & \text{falls } \tilde{x}_i = x_i \\ 0 & \text{falls } \tilde{x}_i = \bar{x}_i, \end{cases}$

also $\varphi(\tilde{x}_i)=1$, $1 \leq i \leq k$.

Ist A am Zug, so wählt A x_{i+1}, falls A im Belegungsspiel bei dem durch φ spezifizierten Spielverlauf $\varphi(x_{i+1})=1$ wählen würde, andernfalls wählt A $\bar{x}_{i+1}$. Sind auf diese Weise $\tilde{x}_1,\ldots,\tilde{x}_n$ alle gewählt, so ist damit eine Belegung $\varphi(D)=1$ spezifiziert. B entfernt die Kante von D nach einem K_j. Wegen $\varphi(D)=1$ folgt $\varphi(K_j)=1$, also gibt es in K_j ein $\tilde{x}_i$, so daß $\varphi(\tilde{x}_i)=1$ ist, d.h. $\tilde{x}_i$ wurde gewählt, A entfernt die Kante von K_j nach $\tilde{x}_i$. Da die einzige Kante, die ursprünglich aus $\tilde{x}_i$ herausführte, bereits entfernt ist, verliert B.

Ebenso zeigt man, daß B eine Gewinnstrategie auf S(C) hat, falls B eine Gewinnstrategie in dem Belegungsspiel auf C hat. □

Damit sind auch Lemma 6.46 und Satz 6.43 bewiesen. □□

Auf ähnliche Weise kann man die PTAPE-Vollständigkeit vieler kombinatorischer Spiele herleiten [13,62]. Darüber hinaus sind viele weitere PTAPE-vollständige Probleme bekannt (siehe etwa [2,7o]).

Offene Fragen

a) nxn-Go sei die naheliegende Verallgemeinerung des Spiels Go auf ein nxn-Brett. Ist das Problem zu gegebener Spielsituation in nxn-Go zu entscheiden, ob der Spieler am Zug eine Gewinnstrategie hat, PTAPE-vollständig oder zumindest NP-hart?

   Man kann zeigen, daß die Verallgemeinerung dieses Problems auf ein nxnx3-Gitter PTAPE-hart ist.

b) nxn-Hex sei die naheliegende Verallgemeinerung des Spiels Hex auf ein nxn-Brett. Ist das Problem, zu gegebener Spielsituation in nxn-Hex zu entscheiden, ob der Spieler am Zug eine Gewinnstrategie hat, PTAPE-vollständig oder zumindest NP-hart?

   Man kann zeigen, daß die Verallgemeinerung von Hex auf beliebige Graphen PTAPE-vollständig ist [13].

c) Ist das Problem, zu gegebenem Graphen G und zu gegebener Zahl m zu entscheiden, ob jeder Knoten in G nach den Regeln des Markenspiels aus Kapitel 5 mit m Marken markiert werden kann, PTAPE-vollständig?

Es ist bekannt, daß dieses Problem NP-hart ist [66].

Übungen

1. Man bilde die Konjunktive Pränex-Normalform von
$(\forall x_1 \forall x_2 (x_1 \vee x_3)) \wedge (\sim \forall x_1 (x_1 \equiv x_2 \vee \overline{x}_3))$.

2. Man vollende den Beweis von Lemma 6.48.

3. Ist GEO$\in$DTIME(n), so gibt es eine $O(n \cdot t(n))$-zeitbeschränkte Turingmaschine, die, angesetzt auf eine Spielsituation (G,v), den ersten Zug einer Gewinnstrategie für diese Spielsituation bestimmt, falls die Situation (G,v) eine Gewinnstrategie hat.

4. Es gibt eine Konstante k, so daß gilt:

   Die Sprache GEO ist schon dann PTAPE-vollständig, wenn man sich auf Graphen G beschränkt, in denen kein Knoten mehr als k Vorgänger und Nachfolger hat.

5. Das Spiel "Node-cayless":

   Zwei Spieler setzen abwechselnd Marken auf die Knoten eines ungerichteten Graphen. Wenn ein Nachbar eines Knotens schon eine Marke hat, darf dieser Knoten nicht mehr markiert werden. Wer als erster nicht mehr setzen kann, hat verloren.

   Sei NC={w|w ist Kodierung einer Spielsituation im obigen Spiel, in der der Spieler am Zug eine Gewinnstrategie hat}.

   a) NC $\in$ PTAPE
   b) KPNF $\leq_p$ NC.

4. Ein nachweislich schweres Problem

Der Nachweis der NP-Vollständigkeit bzw. der PTAPE-Vollständigkeit eines Problems liefert bei unserem heutigen Wissensstand zwar Evidenz, aber noch keinen Beweis dafür, daß dieses Problem wirklich schwer ist. Wir lernen jetzt ein nachweislich schwer entscheidbares Problem kennen.

6.4.1. Presburger Arithmetik

Wir erinnern an die Definition der arithmetischen Aussagen in § 2.2.3.

Definition 6.49: Die Prädikate bzw. Aussagen der Presburger-Arithmetik sind genau die arithmetischen Aussagen, in denen das Zeichen · (für Multiplikation) nicht vorkommt.

Es erweist sich als angenehm, in diesen Prädikaten vorübergehend jede Variable als ein einzelnes Zeichen zu betrachten, d.h. wir fassen diese Aussagen als Zeichenreihen über einem abzählbaren Alphabet auf. Die Länge einer solchen Aussage ist die Länge dieser Zeichenreihe. Durch Kodieren von Variable x_i als xbin(i) kann man Prädikate der Länge n in Zeichenreihen der Länge höchstens nlog n über einem endlichen Alphabet kodieren. Sei

Z_P = {w|w ist Kodierung einer wahren Aussage der Presburger-Arithmetik}.

Presburger zeigte 1930, daß Z_P entscheidbar ist [56]; aber jedes Entscheidungsverfahren für Z_P erfordert enormen Zeitaufwand [14]:

Satz 6.50: Z_P ist EXPTIME-hart.

Korollar 6.51: $Z_P \notin P$.

Beweis des Korollars: unmittelbar aus Korollar 6.8.

Beweis des Satzes: Sei $L \in EXPTIME$ und $M=(A,Z,\delta,z_o,z_+,z_-)$ sei eine $2^{C\cdot n}$-zeitbeschränkte 1-Bandmaschine, die L akzeptiert, $v \in A^*$ sei eine Eingabe für M, $|v|=n$.

Sei $/ \notin AUZ$ ein neues Zeichen. M akzeptiert v genau dann, wenn es ein Wort $w=k_o/.../k_T$ gibt, so daß gilt:

(1) k_o hat die Form $B...Bz_ovB...B$

(2) $k_i \vdash k_{i+1}$ für alle $i \leq T-1$

(3) $|k_i|=|k_j|$ für alle i,j

(4) In keinem k_i steht ein Zeichen $z \in Z$ am Rand

(5) k_T ist Endkonfiguration

(6) $|k_i| \leq 2^{C\cdot n}+3$ für alle $i \leq T \leq 2^{C\cdot n}$, also $|w| \leq (2^{C\cdot n}+1)(2^{C\cdot n}+4)-1$.

Kodiert man wie in § 2.2.6 Zeichenreihen aus (AUZU{/})* vermittels einer Kodierung ψ in $\mathbb{N}$, in dem man für eine Primzahl $p \geq |A|+|Z|+2$ eine Zeichenreihe γ als Darstellung der Zahl $\psi(\gamma)$ zur Basis p auffaßt, so folgt

Definition 6.49: Die Prädikate bzw. Aussagen der Presburger-Arithmetik sind genau die arithmetischen Aussagen, in denen das Zeichen · (für Multiplikation) nicht vorkommt.

Es erweist sich als angenehm, in diesen Prädikaten vorübergehend jede Variable als ein einzelnes Zeichen zu betrachten, d.h. wir fassen diese Aussagen als Zeichenreihen über einem abzählbaren Alphabet auf. Die Länge einer solchen Aussage ist die Länge dieser Zeichenreihe. Durch Kodieren von Variable x_i als xbin(i) kann man Prädikate der Länge n in Zeichenreihen der Länge höchstens nlog n über einem endlichen Alphabet kodieren. Sei

Z_P = {w|w ist Kodierung einer wahren Aussage der Presburger-Arithmetik}.

Presburger zeigte 1930, daß Z_P entscheidbar ist [56]; aber jedes Entscheidungsverfahren für Z_P erfordert enormen Zeitaufwand [14]:

Satz 6.50: Z_P ist EXPTIME-hart.

Korollar 6.51: $Z_P \notin P$.

Beweis des Korollars: unmittelbar aus Korollar 6.8.

Beweis des Satzes: Sei $L \in EXPTIME$ und $M=(A,Z,\delta,z_o,z_+,z_-)$ sei eine $2^{C\cdot n}$-zeitbeschränkte 1-Bandmaschine, die L akzeptiert, $v \in A^*$ sei eine Eingabe für M, $|v|=n$.

Sei $/ \notin AUZ$ ein neues Zeichen. M akzeptiert v genau dann, wenn es ein Wort $w=k_o/.../k_T$ gibt, so daß gilt:

(1) k_o hat die Form $B...Bz_ovB...B$

(2) $k_i \vdash k_{i+1}$ für alle $i \leq T-1$

(3) $|k_i|=|k_j|$ für alle i,j

(4) In keinem k_i steht ein Zeichen $z \in Z$ am Rand

(5) k_T ist Endkonfiguration

(6) $|k_i| \leq 2^{C\cdot n}+3$ für alle $i \leq T \leq 2^{C\cdot n}$, also $|w| \leq (2^{C\cdot n}+1)(2^{C\cdot n}+4)-1$.

Kodiert man wie in § 2.2.6 Zeichenreihen aus $(AUZU\{/\})^*$ vermittels einer Kodierung ψ in $\mathbb{N}$, in dem man für eine Primzahl $p \geq |A|+|Z|+2$ eine Zeichenreihe γ als Darstellung der Zahl $\psi(\gamma)$ zur Basis p auffaßt, so folgt

$$\psi(w) \leq_p |w| \leq_2 (\log P)(2^{C\cdot n}+1)(2^{C\cdot n}+4)-1) \leq_2 2^{3Cn}$$

für alle genügend großen Zahlen n.

In § 2.2.6 wurde für eine beliebige Turingmaschine M und Eingabe v eine Aussage der elementaren Arithmetik konstruiert, die besagt: M angesetzt auf x hält im Zustand z_o. In dieser Aussage kommen Teilprädikate der Form a·b=c vor, und man kann ein solches Prädikat auch nicht durch ein Prädikat der Presburger Arithmetik umschreiben, da sonst nach Satz 2.41 Z_P nicht entscheidbar wäre. Es gilt jedoch:

Lemma 6.52: Für alle n gibt es ein Prädikat $m_n(a,b,c)$ der Presburger-Arithmetik mit Länge O(n) und eine Zahl $p_n \geq 2^{2^n}$, so daß $m_n(a,b,c)$ bedeutet: $a \leq p_n \wedge a\cdot b=c$.

Beweis:

$$m_o(a,b,c) \equiv (a=0 \wedge c=0 \vee \bigvee_{i=1}^{4} (a=1+\ldots+1\,(i\text{-mal}) \wedge c=b+\ldots+b\,(i\text{-mal}))).$$

Um m_{n+1} aus m_n zu definieren, benutzt man:

Lemma 6.53: Für jede natürliche Zahl a gibt es natürliche Zahlen $a_1,a_2,a_3,a_4 \leq \lfloor\sqrt{a}\rfloor$, so daß $a=a_1\cdot a_2+a_3+a_4$ gilt.

Beweis: $a_1=a_2=\lfloor\sqrt{a}\rfloor=\sqrt{a}-b, b<1$

$$a_3+a_4=2b\sqrt{a}-b^2=2b(\lfloor\sqrt{a}\rfloor+b)-b^2$$
$$=2b\lfloor\sqrt{a}\rfloor+b^2<2\lfloor\sqrt{a}\rfloor+1. \quad \square$$

Nun läge es nahe zu definieren:

$$m_{n+1}(a,b,c) \equiv \exists a_1,\ldots,a_4,c_1,\ldots,c_4$$
$$m_n(a_2,b,c_1) \wedge m_n(a_1,c_1,c_2) \wedge$$
$$m_n(a_3,b,c_3) \wedge m_n(a_4,b,c_4) \wedge$$
$$c=c_2+c_3+c_4.$$

Dies würde jedoch Prädikate m_n der Länge mindestens 4^n liefern. Ähnlich wie im Beweis von Lemma 6.42 erklärt man deshalb $m_{n+1}(a,b,c) \equiv \exists a_1,\ldots,a_4,c_1,\ldots,c_4 \forall d,e,f$:

$$(((d=a_2 \wedge e=b \wedge f=c_1) \vee$$
$$(d=a_1 \wedge e=c_1 \wedge f=c_2) \vee$$
$$(d=a_3 \wedge e=b \wedge f=c_3) \vee$$

$$\vee\ (d=a_4 \wedge e=b \wedge f=c_4)$$
$$\rightarrow m_n(d,e,f))$$
$$\wedge\ c=c_2+c_3+c_4).$$

Man erhält damit für die Länge L(n) von m_n:

$$L(n+1) = L(n)+O(1)$$
$$L(0) = O(1),$$

also

$$L(n) = O(n).$$

Besagt $m_n(a,b,c)$:

$$a \leq p_n \wedge a \cdot b = c,$$

so besagt $m_{n+1}(a,b,c)$:

$$a \leq p_n^2 + 2p_n \wedge a \cdot b = c,$$

also

$$p_{n+1} > p_n^2.$$

Mit $p_o=4$ folgt $p_n \geq 2^{2^n}$. □

Sei $H_n(M,v)$ die Aussage der Presburger Arithmetik, die man aus der Aussage $\exists w((1)\wedge(2)\wedge(3)\wedge(4)\wedge(5))$ aus § 2.2.6 erhält, indem man Prädikate der Form a.b=c durch $m_n(a,b,c)$ und z_e durch z_+ ersetzt. Dann ist die Länge von $H_n(M,v)=O(n+|v|)$, und M angesetzt auf v hält genau dann, wenn $H_{3C\cdot|v|}(M,v)$ wahr ist.

Die Kodierung von $H_{3C\cdot|v|}(M,v)$ hat Länge $O(|v|\log|v|)$ und kann in polynomieller Zeit aus v gewonnen werden. □

Korollar 6.51 kann mit geringer Mühe noch weiter verschärft werden:

Satz 6.54: Es gibt eine Konstante c, so daß $Z_P \notin DTIME(2^{cn/\log n})$ gilt.

Beweis: Angenommen $Z_P \in DTIME(2^{cn/\log n})$ für alle c. Nach dem Zeithierarchiesatz 4.25 gibt es eine Sprache L, so daß $L \in DTIME(2^n)$, aber $L \notin DTIME(2^n/n^2)$ gilt.

Sei M eine 2^{cn}-zeitbeschränkte 1-Bandmaschine, die L akzeptiert. Dann kann man für Eingaben v der Länge n "$v \in L$?" entscheiden, indem man zuerst in polynomieller Zeit die Kodierung der Aussage $H_{3cn}(M,v)$ erzeugt - diese hat Länge $\ell(n) \leq Dn \log n$ für ein passendes D - und dann in Zeit $O(2^{c\ell(n)/\log \ell(n)})$ die Wahrheit dieser

Aussage testet. Da dies für alle c möglich sein soll, kann man c klein genug wählen, so daß $c\ell(n)/\log \ell(n) \leq n-2 \log n$ für alle außer endlich vielen $n \in \mathbb{N}$ gilt. Dann folgt aber der Widerspruch $L \in DTIME(2^n/n^2)$. □

Sammlungen weiterer nachweislich schwerer Probleme findet man in [2.7o].

Offene Fragen

Satz 6.54 kann weiter verschärft werden. Es gibt eine Konstante c, so daß $Z_p \notin NTIME(2^{2^{cn}})$ gilt [14,69]. Andererseits ist nur $Z_p \in DTIME(2^{2^{2^{O(n \log n)}}})$ bekannt [46].
Welche dieser Schranken kann verbessert werden?

Übungen

1. a) Man zeige:

   Es gibt eine Konstante c, so daß $Z_p \notin NTIME(2^{cn/\log n})$ gilt.

   b) Sei S ein beliebiges vollständiges und korrektes Beweissystem für Z_p.

   Man zeige:

   Es gibt eine Konstante c und unendlich viele wahre Aussagen $w \in Z_p$, so daß jeder Beweis von w in S mindestens Länge $2^{c|w|/\log|w|}$ hat.

2. a) Man modifiziere die Formeln H(M,v) so, daß man stets mit einer festen Anzahl von Variablen auskommt.

   Hinweis: Für mehrfach gebundene Variablen zählt der innerste Quantor, z.B. ist

   $$\forall x(x+y=z \wedge \exists x(\sim x=u))$$

   äquivalent zu

   $$\forall x(x+y=z \wedge \exists x'(\sim x'=u)).$$

   b) Es gibt eine Konstante c, so daß $Z_p \notin NTIME(2^{cn})$.

3. Für eine Sprache $L \subseteq A^*$ gelte:

   Zu jeder Sprache $L' \in DTAPE(2^n)$ mit $L' \subseteq A^*$ gibt es eine auf Band $O(2^{\sqrt{n}})$ berechenbare Funktion $f: A^* \to A^*$, so daß für alle $w \in A^*$ gilt: $w \in L' \iff f(w) \in L$. Man gebe eine möglichst große untere

Schranke für die Bandkomplexität von L an, falls für alle $w \in A^*$ gilt:

a) $|f(w)| \leq c|w|$

b) $|f(w)| \leq |w|\log|w|$

c) $|f(w)| \leq |w|^2$.

7. Abstrakte Komplexitätstheorie

In diesem Abschnitt werden einige offengebliebene Fragen beantwortet. Wir haben nie die Komplexität einer Sprache etwa als die Komplexität des effizientesten Programms, das die Sprache erkennt, eingeführt. Es stellt sich heraus, daß dies auch nicht möglich ist, denn es gibt Sprachen, zu deren Erkennung es kein effizientestes Programm gibt.

Weiter waren offen geblieben, ob die Hierarchiesätze gültig bleiben, wenn man die Voraussetzung über die Gutartigkeit der Funktionen t(n), durch die die Komplexitätsklassen definiert sind, wegfallen läßt. Hier stellt sich heraus, daß man ohne eine solche Voraussage keine generelle quantitative Aussage darüber machen kann, wie "dicht" die verschiedenen Komplexitätsklassen beieinander liegen.

Diese merkwürdigen Phänomene tauchen nicht etwa deswegen auf, weil Rechenzeit und Speicherplatz ungeschickt gewählte Komplexitätsmaße sind, sondern man kann diese Ergebnisse schon unter ganz schwachen Bedingungen darüber, was man für ein Komplexitätsmaß halten will, herleiten.

Im Anhang machen wir einige Bemerkungen über die physikalischen Grenzen, die der Geschwindigkeit von Rechenvorgängen gesetzt sind.

7.1. Abstrakte Komplexitätsmaße

Die Ergebnisse, die wir im folgenden herleiten, sollen nicht nur für ein spezielles Modell der Berechenbarkeit wie etwa Turingmaschinen unabhängig vom gewählten Komplexitätsmaß sein, sondern sie sollen sogar in jedem "halbwegs vernünftigen" Modell der Berechenbarkeit für jedes Komplexitätsmaß gelten. Was ein "halbwegs vernünftiges" Modell sein soll, wird durch den Begriff der zulässigen Aufzählung formal gefaßt. Danach werden abstrakte Komplexitätsmaße eingeführt und zunächst einige einfache Ergebnisse darüber hergeleitet.

7.1.1. Zulässige Aufzählungen

Wir fixieren irgendeine einfache Gödelisierung von 1-Band-Turingmaschinen, d.h. eine einfache Kodierung der Maschinentafeln als Zeichenreihen über einem festen Alphabet bzw. in diesem Zu-

sammenhang: als natürliche Zahlen (man nehme etwa die Gödelisierung aus § 2.1.2). Für $\in\{0,1\}^*$ sei die Maschine M_u definiert wie in § 2.1.2. Für $i\in\mathbb{N}$ bezeichnet in Zukunft $M^i=M_{bin(i)}$, d.h. M^i ist eine Maschine mit Gödelnummer i. Man kann M^i analog zu Definition 1.28 effektiv auf folgende Weise eine partielle berechenbare Funktion $\varphi_i:\mathbb{N}\to\mathbb{N}$ zuordnen: Hält M_i angesetzt auf die Binärdarstellung von j mit der Binärdarstellung einer Zahl k auf dem Band, so ist $\varphi_i(j)=k$, andernfalls ist $\varphi_i(j)$ nicht definiert. Die Folge von Funktionen $(\varphi_0,\varphi_1,\ldots)$ heißt Standardaufzählung.

In Zukunft schreiben wir, dem allgemeinen Sprachgebrauch folgend, für "f ist eine partielle berechenbare Funktion" einfach "f ist partiell rekursiv" und für "f ist eine totale berechenbare Funktion" einfach "f ist rekursiv".

Definition 7.1: Eine Folge $(f_0,f_1,\ldots)$ von Funktionen heißt Aufzählung, falls i) und ii) gelten

i) $\{f_0,f_1,\ldots\}=\{f\mid f:\mathbb{N}\to\mathbb{N}$ ist partiell rekursiv$\}$

ii) Es gibt eine rekursive Funktion $g:\mathbb{N}\to\mathbb{N}$, so daß $f_i=\varphi_{g(i)}$ für alle i gilt.

Ebenso wie bei 1-Band-Turingmaschinen kann man aus Gödelisierungen von Mehrband-Turingmaschinen, Registermaschinen oder μ-rekursiven Schemata Aufzählungen gewinnen. Aufzählungen repräsentieren auf diese Weise Modelle der Berechenbarkeit. Für je zwei der Modelle, die wir bisher betrachtet haben, war es stets möglich, zu jedem "Programm" in einem Modell effektiv ein "Programm" in dem anderen Modell anzugeben, das die gleiche Funktion berechnet. Beispielsweise wurde im Beweis von Satz 1.41 zu jeder 1-Band-Turingmaschine ein μ-rekursives Schema angegeben, das die Arbeitsweise dieser Maschine simuliert. Formaler ausgedrückt: Wir haben eine totale berechenbare Funktion g angegeben, die jede Beschreibung (Gödelnummer) i einer 1-Band-Turingmaschine in die Beschreibung (Gödelnummer) g(i) eines Rekursionsschemas, das die gleiche Funktion wie die Turingmaschine berechnet, abbildet. Wir sehen zwei Modelle als ähnlich an, wenn eine solche effektive Simulation möglich ist und übertragen dies auf Aufzählungen.

Definition 7.2: Zwei Aufzählungen

$$(f_0,f_1,\ldots) \text{ und } (f_0',f_1',\ldots)$$

heißen ähnlich, falls es rekursive Funktionen

$g,h : \mathbb{N} \to \mathbb{N}$,

gibt, so daß

$f_i = f'_{g(i)}$ und $f'_i = f_{h(i)}$ für alle $i \in \mathbb{N}$ gilt.

Ähnlichkeit ist eine Äquivalenzrelation.

Definition 7.3: Eine Aufzählung $(f_0, f_1, \ldots)$ heißt zulässig, wenn sie zur Standardaufzählung ähnlich ist.

Nach dem oben Gesagten sind die Aufzählungen, die man aus einfachen Gödelisierungen von 1-Band-Turingmaschinen, Mehrband-Turingmaschinen, Registermaschinen und Rekursionsschemata erhält, alle zulässig. Der folgende Satz charakterisiert die zulässigen Aufzählungen als diejenigen, in denen es ein universelles Programm (u) und einen uniformen Einsetzmechanismus (s) gibt.

Satz 7.4: Eine Aufzählung $(f_0, f_1, \ldots)$ ist genau dann zulässig, wenn es totale berechenbare Funktionen $b,s: \mathbb{N}^2 \to \mathbb{N}$ und einen Index $u \in \mathbb{N}$ gibt, so daß gilt:

1) b ist bijektiv
2) Für alle $m,n \in \mathbb{N}$ gilt $f_u(b(m,n)) = f_m(n)$
3) Für alle $i,m,n \in \mathbb{N}$ gilt $f_i(b(m,n)) = f_{s(i,m)}(n)$.

Beweis:

Lemma 7.5: Ist $b: \mathbb{N}^2 \in \mathbb{N}$ rekursiv und bijektiv, dann gibt es totale berechenbare Funktionen $b_1, b_2: \mathbb{N} \to \mathbb{N}$, so daß für alle $m,n \in \mathbb{N}$ gilt:

$$b_1(b(m,n)) = m \text{ und } b_2(b(m,n)) = n.$$

Beweis: Sei M eine Turingmaschine, die angesetzt auf bin(i) systematisch alle Zeichenreihen der Form bin(m)#bin(n) aufzählt und für jede dieser Zeichenreihen bin(b(m,n)) berechnet. Da b surjektiv ist, findet M schließlich m und n so, daß i=b(m,n) gilt. □

Lemma 7.6: 1), 2) und 3) gelten für die Standardaufzählung.

Beweis: b sei die Funktion $b^{(2)}$ aus Satz 1.11.

u ist die Gödelnummer einer Maschine, die angesetzt auf bin(i) mit $i \in \mathbb{N}$ die Binärdarstellungen von m und n mit i=b(m,n) berechnet und dann die Maschine mit Gödelnummer m angesetzt auf Eingabe bin(n) simuliert.

S sei eine Maschine, die angesetzt auf bin(i)#bin(m) eine Gödelisierung einer Maschine $Q_{m,i}$ erzeugt, die folgendes tut:

Im "endlichen Gedächtnis" von $Q_{m,i}$ sind die Zahlen m und i gespeichert (vergl. Beweis von Lemma 1.3o). Angesetzt auf Eingabe bin(n) schreibt $Q_{m,i}$ erst bin(m) auf das Band, berechnet bin(b(m,n)) und verhält sich dann wie M^i (die Maschine mit Gödelnummer i) angesetzt auf bin(b(m,n)).

Die Maschine S berechnet die Funktion s. □

Lemma 7.7: $(f_o,f_1,\ldots)$ und $(f'_o,f'_1,\ldots)$ seien Aufzählungen, die beide (1), (2) und (3) für u,s,b bzw. u',s',b' erfüllen. Dann sind diese Aufzählungen ähnlich, d.h. es gibt rekursive Funktionen g und h, so daß $f_i=f'_{g(i)}$ und $f'_i=f_{h(i)}$ für alle i gilt.

Beweis: Es genügt, g anzugeben. h erhält man, indem man die Rollen der beiden Aufzählungen vertauscht. Nach Voraussetzung gilt für alle j und n

$$f_j(n) = f_u(b(j,n)).$$

f_u ist eine partielle berechenbare Funktion.

Also gibt es einen Index v, so daß $f_u=f'_v$, also

$$f_j(n) = f'_v(b(j,n)) = f'_{s'(v,j)}(n).$$

$g(j)=s'(v,j)$ leistet das Gewünschte. □

Es bleibt zu zeigen:

Lemma 7.8: Ist $(f_o,f_1,\ldots)$ zulässig, so gelten (1), (2) und (3).

Beweis: Für die Standardaufzählung $(\varphi_o,\varphi_1,\ldots)$ bezeichne b_o,s_o,u_o die Funktionen b,s bzw. den Index u so, daß (1), (2) und (3) gelten. b_1 und b_2 bezeichnen die Umkehrungen von b nach Lemma 7.5. g und h seien rekursiv und so, daß $f_i=\varphi_{g(i)}$ und $\varphi_i=f_{h(i)}$ für alle $i\in\mathbb{N}$ gilt. Wir setzen für $(f_o,f_1,\ldots)$ $b=b_o$ und müssen u und s für diese Aufzählung angeben.

Für u muß $f_u(b(m,n))=f_m(n)$ gelten, d.h.

$$f_u(i)=f_{b_1(i)}(b_2(i))=\varphi_{g(b_1(i))}b_2(i)=\varphi_{u_o}(b(g(b_1(i)),b_2(i)))$$

$$=\varphi_v(i) \text{ für einen Index } v\in\mathbb{N}$$

$$=f_{h(v)}(i)$$

u=h(v) leistet das Gewünschte.

Für s muß $f_i(b(m,n))=f_{s(i,m)}(n)$ gelten, d.h.

$$f_i(j) = f_{s(i,b_1(j))}(b_2(j)).$$

Es ist

$$f_i(j) = \varphi_{g(i)}(j) = \varphi_{s_o(g(i),b_1(j))}(b_2(j))$$

$$= f_{h(s_o(g(i),b_1(j)))}(b_2(j)).$$

$s(i,m)=h(s_o(g(i),m))$ leistet das Gewünschte. □□

7.1.2. Abstrakte Komplexitätsmaße

Wir isolieren einige Eigenschaften, die alle Komplexitätsmaße, die wir bisher kennengelernt haben, gemeinsam haben. Dann definieren wir abstrakte Komplexitätsmaße durch genau diese Eigenschaften.

Sei M^i die 1-Band-Turingmaschine mit Gödelnummer i, $f_i: \mathbb{N} \to \mathbb{N}$ die durch M^i berechnete Funktion, und für $i,n \in \mathbb{N}$, sei $\psi_i(n)$ die Zeit- oder Bandkomplexität von M^i angesetzt auf die Binärdarstellung von n. Dann gilt

Lemma 7.8:

(1) Ist $f_i(n)$ definiert, so ist auch $\psi_i(n)$ definiert.

(2) Es gibt eine totale berechenbare Funktion $M: \mathbb{N}^3 \to \{0,1,2\}$, so daß gilt

$$M(i,n,m) = \begin{cases} 0 & \text{falls } \psi_i(n) \neq m \\ 1 & \text{falls } \psi_i(n)=m \text{ und } f_i(n) \text{ existiert} \\ 2 & \text{falls } \psi_i(n)=m \text{ und } f_i(n) \text{ existiert nicht.} \end{cases}$$

Beweis:

(1) ist trivial,

(2) für Zeitkomplexität:

Sei Q eine universelle Maschine (vergl. § 2.1.6), die, angesetzt auf Eingabe bin(i)#bin(n)#bin(m), die Maschine M^i angesetzt auf bin(n) für genau m Schritte simuliert. Q gibt 1 aus, falls M^i nach genau m Schritten hält, andernfalls gibt Q 0 aus. Q berechnet M.

(2) für Bandkomplexität:

A_i bzw. Z_i seien das Alphabet bzw. die Zustandsmenge von M^i. Sei Q eine universelle Maschine, die angesetzt auf bin(i)#bin(n)#bin(m)

die Maschine M^i angesetzt auf bin(n) solange simuliert, bis M^i mehr als m Felder benutzt oder mehr als $|Z_i| \cdot m \cdot |A_i|^m$ Schritte macht.

Folgende Fälle sind möglich:

a) Die Simultation kann zu Ende geführt werden und
   α) M^i benutzt genau m Felder
   β) M^i benutzt höchstens m-1 Felder.

b) Die Simultation bricht wegen Zeitüberschreitung ab und
   α) M^i benutzt genau m Felder
   β) M^i benutzt höchstens m-1 Felder.

c) Die Simultation bricht wegen Platzüberschreitung ab.

Die Ausgabe von Q ist:

0 in den Fällen aβ,bβ und c
1 im Fall aα
2 im Fall bα.

Aus Lemma 4.4 folgt, daß Q die Funktion M berechnet. □

Nach dem gleichen Muster zeigt man:

Satz 7.9: Ist M(i) die Mehrband-Turingmaschine oder die Registermaschine mit Gödelnummer i, $f_i: \mathbb{N} \to \mathbb{N}$ die durch M(i) berechnete Funktion und $\psi_i(n)$ die Zeitkomplexität (für Registermaschinen uniform oder logarithmisch), oder die Bandkomplexität bzw. die logarithmische Platzkomplexität oder das Produkt aus Zeitkomplexität von M(i) angesetzt auf bin(n), so gelten (1) und (2) von Lemma 7.8.

Beweis: Übung. □

Überraschenderweise kann man schon eine ganze Theorie der Komplexität (nämlich die Abstrakte Komplexität) entwickeln, wenn man von den Komplexitätsmaßen nur die Aussagen (1) und (2) aus Lemma 7.8 benutzt.

Definition 7.1o: Ein abstraktes Komplexitätsmaß wird spezifiziert durch ein Paar (F,ψ), wobei gilt:

i) $F=(f_o,f_1,\ldots)$ ist eine zulässige Aufzählung

ii) $\psi=(\psi_o,\psi_1,\ldots)$ ist eine Folge von partiellen rekursiven Funktionen $\psi_i: \mathbb{N} \to \mathbb{N}$, so daß die Aussagen (1) und (2) aus Lemma 7.8 gelten.

Die Aussagen (1) und (2) sind eine abgeschwächte Form der sogenannten Blum'schen Axiome [3] und werden künftig Axiom 1 und Axiom 2 genannt. Die Funktionen $\psi_0, \psi_1 \ldots$ heißen Schrittzahlfunktionen. Lemma 7.8 und Satz 7.9 liefern Beispiele für abstrakte Komplexitätsmaße.

Wir geben zwei Folgen von Funktionen ψ_i an, die zusammen mit der Standardaufzählung $(\varphi_0, \varphi_1, \ldots)$ keine abstrakten Komplexitätsmaße liefern. $\psi_i(n)$ nicht definiert für alle i und n verletzt (1).

$$\psi_i(n) = \begin{cases} 1 & \text{falls } \varphi_i(n) \text{ existiert} \\ \text{nicht definiert} & \text{sonst} \end{cases}$$

verletzt (2), denn wenn M berechenbar wäre, so wäre auch

$$H(i,n) = \begin{cases} 1 & \text{falls } M(i,n,1)=1 \\ 0 & \text{sonst} \end{cases}$$

berechenbar, aber H(i,n) ist die charakteristische Funktion des Halteproblems.

7.1.3. Einige einfache Eigenschaften von abstrakten Komplexitätsmaßen

Im folgenden ist immer

$$(F,\psi), F=(f_0, f_1, \ldots), \psi=(\psi=(\psi_1, \ldots)$$

ein beliebiges abstraktes Komplexitätsmaß. Wir sagen, i ist ein Index für $f: \mathbb{N} \to \mathbb{N}$, falls $f=f_i$ gilt. b und u seien für die Aufzählung F wie in Satz 7.4 beschrieben. Wir leiten einige einfache Komplexitätsaussagen her. Zunächst zeigen wir, daß es beliebig komplexe Funktionen gibt.

Satz 7.11: Sei $T: \mathbb{N} \to \mathbb{N}$ rekursiv. Dann gibt es eine rekursive Funktion f, so daß für jeden Index i für f gilt:

$$\psi_i(n) \geq T(n) \text{ für unendlich viele } n.$$

Beweis: $R: \mathbb{N} \to \mathbb{N}$ sei eine rekursive Funktion, so daß es für jedes $i \in \mathbb{N}$ unendlich viele $j \in \mathbb{N}$ mit $R(j)=i$ gibt, z.B.

$$R(0), R(1), R(2), \ldots = 0,0,1,0,1,2,0,1,2,3,\ldots$$

Sei

$$f(n) = \begin{cases} f_{R(n)}(n)+1 & \text{falls } \psi_{R(n)}(n) < T(n) \text{ und } f_{R(n)}(n) \text{ existiert} \\ 0 & \text{sonst.} \end{cases}$$

f(n) ist rekursiv, denn

$$f(n) = \begin{cases} f_u(b(R(n),n))+1 & \text{falls } M(R(n),n,t)=1 \text{ für ein } t<T(n) \\ 0 & \text{sonst.} \end{cases}$$

Ist i ein Index für f und R(n)=i, so folgt $\psi_i(n) \geq T(n)$. □

Der folgende Satz zeigt, daß es sogar Funktionen gibt, die fast überall beliebig komplex sind:

Satz 7.12: Sei $T:\mathbb{N} \to \mathbb{N}$ rekursiv. Dann gibt es eine rekursive 0/1-wertige Funktion f, so daß für jeden Index i für f gilt:

$$\psi_i(n) \geq T(n) \text{ für fast alle } n.$$

"für fast alle n" bedeutet: für alle außer höchstens endlich vielen $n \in \mathbb{N}$.

Beweis: Um f(n) zu definieren, sucht man das kleinste s, so daß $s<n$ und $\psi_s(n)<T(n)$ gilt, und macht $f(n) \neq f_s(n)$, falls dies nicht schon für ein $m<n$ geschehen ist.

Zur formalen Definition von F definieren wir induktiv und simultan Funktionen s(n) und Listen L(n). L(n) enthält alle Indizes derjenigen Funktionen f_i, so daß f(m) bereits für ein $m \leq n$ von $f_i(m)$ verschieden gemacht wurde.

$$L(0) = \emptyset$$

$$s(n) = \begin{cases} \min\{s \mid s \notin L(n-1), s \leq n, \psi_s(n)<T(n), f_s(n) \text{ existiert}\} \\ \qquad \text{falls dieses existiert} \\ \text{nicht definiert} \quad \text{sonst} \end{cases}$$

$$f(n) = \begin{cases} 0 & \text{falls } s(n) \text{ definiert ist und } f_{s(n)}(n)=1 \\ 1 & \text{sonst} \end{cases}$$

$$L(n) = \begin{cases} L(n-1) \cup \{s(n)\} & \text{falls } s(n) \text{ definiert ist} \\ L(n-1) & \text{sonst.} \end{cases}$$

f(n) ist offenbar rekursiv.

Sei nun i ein Index, so daß $\psi_i(n)<T(n)$ für unendlich viele n gilt. Ist f_i nicht total, folgt $f \neq f_i$, denn f ist total. Ist f_i total, so gibt es unendlich viele $n_j \in \mathbb{N}$, $j=1,2,\ldots$, so daß $n_j \geq i$, $\psi_i(n_j)<T(n_j)$ und $f_i(n_j)$ existiert.

Für jedes j mit $i \notin L(n_j-1)$ folgt

$s(n_j) = i$ oder

$s(n_j) = m$ für ein $m<i$, aber $s(n_k)\neq m$ für alle $k>j$, da $m\in L(n_k)$.

Es folgt $s(n_j)=i$ für ein $j\leq i$ und damit $f\neq f_i$. □

Die nächsten Sätze zeigen, daß man das Wachstum einer Funktion durch ihre Komplexität abschätzen kann, aber nicht umgekehrt die Komplexität einer Funktion durch ihr Wachstum.

Satz 7.12: Es gibt eine rekursive Funktion $h:\mathbb{N}^2\to\mathbb{N}$, so daß für alle i gilt:

$$f_i(n)\leq h(n,\psi_i(n)) \text{ für fast alle } n.$$

Beweis: Sei

$$H(j,n,m) = \begin{cases} f_j(n) & \text{falls } \psi_j(n)=m \text{ und } f_j(n) \text{ existiert} \\ 1 & \text{sonst.} \end{cases}$$

H ist rekursiv. Sei

$$h(n,m) = \max_{j\leq n} H(j,n,m).$$

Für alle $n\geq i$, für die $f_i(n)$ definiert ist, folgt:

$$h(n,\psi_i(n)) = \max_{j\leq n} H(j,n,\psi_i(n)) \geq f_i(n). \quad \square$$

Satz 7.13: Es gibt keine rekursive Funktion $h:\mathbb{N}^2\to\mathbb{N}$, so daß für alle i gilt:

$$\psi_i(n) \leq h(n,f_i(n)) \text{ für unendlich viele } n.$$

Beweis: Gäbe es eine solche Funktion h, so würde für jede 0/1-wertige Funktion folgen:

$$\psi_i(n) \leq h(n,0) + h(n,1) \text{ für unendlich viele } n.$$

Dies widerspricht aber Satz 7.11. □

Satz 7.12 zeigt, daß sehr schnell wachsende Funktionen auch sehr komplex sein müssen.

Wir haben in Lemma 4.4 die Zeitkomplexität einer Turingmaschine durch eine rekursive Funktion ihrer Bandkomplexität nach oben abschätzen können. Umgekehrt ist jede t(n)-zeitbeschränkte Maschine stets n+t(n)-bandbeschränkt. Es zeigt sich, daß eine derartige gegenseitige Abschätzung stets für zwei Komplexitätsmaße, denen die gleiche Aufzählung zugrunde liegt, möglich ist.

Satz 7.14: (F,ψ) und (F,ψ') seien abstrakte Komplexitätsmaße. Dann gibt es eine im ersten Argument monoton wachsende rekursive

Funktion $R:\mathbb{N}^2 \to \mathbb{N}$, so daß für alle i und alle $n \geq i$ gilt: $\psi_i^!(n) \leq R(n,\psi_i(n))$ und $\psi_i(n) \leq R(n,\psi_i^!(n))$, falls $f_i(n)$ definiert ist.

Beweis:

$$R(n,m) = \begin{cases} \max\{\psi_j(n), \psi_j^!(n) \mid j \leq n, M(i,n,m)=1 \text{ oder } M'(i,n,m)=1\} \\ \qquad\qquad \text{falls dieses existiert} \\ 1 \qquad \text{sonst.} \end{cases}$$

R ist offenbar im ersten Argument monoton wachsend; ist $i \geq n$ und $f_i(n)$ definiert, so folgt

$$R(n,\psi_i(n)) = \max\{\psi_j(n), \psi_j^!(n) \mid j \leq n, \psi_j(n)=\psi_j(n) \text{ oder } \psi_j^!(n)=\psi_i(n)\} \geq \psi_j^!(n).$$

$R(n,\psi_i^!(n)) \geq \psi_i(n)$ folgt ebenso. □

Wir haben früher nicht nur verschiedene Komplexitätsmaße auf dem gleichen Modell verglichen, sondern auch Komplexitätsmaße auf verschiedenen Modellen miteinander verglichen. Zum Beispiel haben wir in Satz 3.12 b) eine rekursive Funktion R angegeben, so daß jede uniform t(n)-zeitbeschränkte Registermaschine durch eine $O(R(t(n)))$-zeitbeschränkte Mehrband-Turingmaschine simuliert werden kann (nämlich $R(t)=t^3$). Wir zeigen, daß auch eine solche Abschätzung immer möglich ist.

Satz 7.15: (F,ψ) und (F',ψ') seien Komplexitätsmaße, g und h seien rekursive Funktionen, so daß

$$f_i = f'_{g(i)} \text{ und } f_i^! = f_{h(i)}$$

für alle $i \in \mathbb{N}$ gilt.

Dann gibt es eine im ersten Argument monoton und im zweiten Argument streng monoton wachsende rekursive Funktion $R:\mathbb{N}^2 \to \mathbb{N}$, so daß für alle $i \in \mathbb{N}$ i) und ii) für fast alle $n \in \mathbb{N}$ gelten:

i) $\psi_{h(i)}(n) \leq R(n,\psi_i^!(n))$ falls $f_i^!(n)$ definiert ist

ii) $\psi'_{g(i)}(n) \leq R(n,\psi_i(n))$. falls $f_i(n)$ definiert ist.

Beweis: Sei $\psi_i^h(n) = \psi_{h(i)}(n)$ und

$$\psi^h = (\psi_o^h, \psi_1^h, \ldots).$$

Es gilt

Lemma 7.16: (F',ψ^h) ist ein Komplexitätsmaß.

Beweis: Die Axiome 1 und 2 müssen nachgeprüft werden. Existiert

$f'_i(n)=f_{h(i)}(n)$, so existiert auch $\psi_{h(i)}(n)=\psi^h_i(n)$. Erfüllt M Axiom 2 für (F,ψ), so erfüllt $M^h(i,n,m)=M(h(i),n,m)$ Axiom 2 für (F',ψ^h). □

Anwendung von Satz 7.14 auf (F',ψ') und (F',ψ^h) liefert eine rekursive Funktion $R_1:\mathbb{N}^2\to\mathbb{N}$, so daß für alle $i\in\mathbb{N}$ gilt:

$$\psi_{h(i)}(n)=\psi^h_i(n) \leq R_1(n,\psi'_i(n))$$

falls $n\geq i$ und $f'_i(n)$ definiert ist.

Vertauscht man in dem obigen Argument die Rollen von (F,ψ) und (F',ψ'), so erhält man eine rekursive Funktion $R_2:\mathbb{N}^2\to\mathbb{N}$, so daß für alle $i\in\mathbb{N}$ gilt:

$$\psi'_{g(i)} \leq R_2(n,\psi_i(n))$$

falls $n\geq i$ und $f_i(n)$ definiert ist.

$$R(n,m)=\max\{R_1(n,m),R_2(n,m),R(n,m\dot{-}1)+1\}$$

ist monoton im ersten und streng monoton im zweiten Argument und erfüllt i) und ii). □

Übungen

1. a) Man gebe eine rekursive Funktion g an, so daß für

$$\varphi^g = (\varphi^g_o,\varphi^g_1,\ldots)$$

mit

$$\varphi^g_i = \varphi_{g(i)}$$

gilt:

φ^g ist eine Aufzählung, in der es für jede partiell rekursive Funktion $f:\mathbb{N}\to\mathbb{N}$ nur einen Index gibt.

b) Ist φ^g zulässig?

2. Man beweise Satz 7.9 in einigen Fällen.
3. Ist (F,ψ) ein Komplexitätsmaß und $r:\mathbb{N}\to\mathbb{N}$ monoton und rekursiv, so ist $(F,r\circ\psi)$ mit

$$r\circ\psi = (r\circ\psi_0,r\circ\psi_1,\ldots)$$

ein Komplexitätsmaß.

7.2. Speed-up und Gap-Theorem

7.2.1. Das Speed-up-Theorem

Es werden für jedes Komplexitätsmaß Funktionen f angegeben, so daß es zu jedem Programm (Index) für f ein erheblich effizienteres gibt.

Satz 7.17 "Speed-up-theorem": Zu jeder rekursiven Funktion $r:\mathbb{N}\to\mathbb{N}$ gibt es eine rekursive Funktion $f:\mathbb{N}\to\mathbb{N}$, so daß gilt:

Zu jedem Index i für f gibt es einen Index j für f so, daß

$$r(\psi_j(n)) \leq \psi_i(n)$$

für fast alle n gilt.

Beispiel 7.18: Sei $r(n)=2^n$ und ψ_i die Zeitkomplexität von Turingmaschine M_i. Der Satz impliziert, daß es eine Funktion f gibt, so daß es zu jeder Turingmaschine M^i, die f berechnet, eine Turingmaschine $M^{i'}$ gibt, die f berechnet und so daß

$$\psi_{i'}(n) \leq \log \psi_i(n)$$

für fast alle n gilt.

Zu i' gibt es aber wieder einen Index i", so daß $M^{i''}$ ebenfalls f berechnet und so daß

$$\psi_{i''}(n) \leq \log \psi_{i'}(n) \leq \log\log \psi_i(n)$$

für fast alle n gilt usw.

Wir beweisen Satz 7.17 erst für ein spezielles Komplexitätsmaß und schließen dann mit Satz 7.15 auf den allgemeinen Fall.

Sei $\varphi=(\varphi_0,\varphi_1,\ldots)$ die Standardaufzählung, $L_i(n)$ sei die Bandkomplexität von Maschine M^i angesetzt auf bin(n), und es sei $L=(L_0,L_1,\ldots)$.

Lemma 7.19: Satz 7.17 gilt für (φ,L).

Beweis: OBdA kann r streng monoton wachsend, bandkonstruierbar und $r(n)\geq 2n$ vorausgesetzt werden.

Wir erklären

$$r_0(1) = 1$$

$$r_{i+1}(1) = r(r_i(1)) \text{ für } i>1,$$

also

$$r_i(1) = r(\ldots r(1)\ldots) \quad i\text{-mal}.$$

Wir definieren die Funktion f so, daß gilt:

1) Ist $\varphi_i = f$, so ist $L_i(n) \geq r_{n-i}(1)$ für fast alle n.

2) Zu jedem $k \in \mathbb{N}$ gibt es eine Zahl $j(k) \in \mathbb{N}$, so daß $f = \varphi_{j(k)}$ und $L_{j(k)}(n) \leq r_{n-k}(1)$ für fast alle n gilt.

Hieraus folgt, daß f die gewünschten Eigenschaften hat, denn ist $\varphi_i = f$, so folgt $\varphi_{j(i+1)} = f$ und

$$r(L_{j(i+1)}(n)) \leq r(r_{n-i-1}(1)) = r_{n-i}(1) \leq L_i(n).$$

f wird ganz ähnlich definiert wie die fast überall komplexen Funktionen aus Satz 7.12. Wir definieren simultan induktiv Funktionen f(n), s(n) und Listen L(n).

$L(0) = \emptyset$.

$$s(n) = \begin{cases} \min\{s \mid s \notin L(n-1), s \leq n, L_s(n) < r_{n-s}(1) \text{ und } \varphi_s(n) \text{ existiert}\} \\ \text{nicht definiert sonst} \end{cases} \quad \text{falls dieses existiert}$$

$$f(n) = \begin{cases} 0 & \text{falls } s(n) \text{ definiert ist und } \varphi_{s(n)}(n) = 1 \\ 1 & \text{sonst} \end{cases}$$

$$L(n) = \begin{cases} L(n-1) \cup \{s(n)\} & \text{falls } s(n) \text{ definiert ist} \\ L(n-1) & \text{sonst.} \end{cases}$$

f ist offenbar rekursiv. Wir müssen 1) und 2) nachweisen. Der Nachweis von 1) folgt genau dem Beweis von Satz 7.12. Man zeigt $f \neq \varphi_i$ falls $L_i(n) < r_{n-i}(n)$ für unendlich viele n gilt.

Zum Nachweis von 2) sei $k \in \mathbb{N}$ vorgegeben. Wir beobachten, daß es ein $v \in \mathbb{N}$ gibt, so daß jeder Index aus $\{1,\ldots,2k\}$, der jemals in irgendeiner Liste L(n) auftaucht, bereits in L(v) auftaucht.

Wir beschreiben nun die Arbeitsweise der Turingmaschine $M^{j(k)}$ angesetzt auf bin(n):

$M^{j(k)}$ hat im endlichen Gedächtnis $f(1),\ldots,f(v)$ und L(v) gespeichert. Ist $n \leq v$, druckt $M^{j(k)}$ sofort f(v). Ist $n > v$ druckt $M^{j(k)}$ zunächst L(v). Dann arbeitet $M_{j(k)}$ in Stufen $m = v+1,\ldots,n$. In Stufen $m < n$ wird L(m) berechnet; in Stufe n wird f(n) berechnet und ausgegeben. In Stufe m wird folgendes getan:

Für $s = 2k+1,\ldots,m$ in dieser Reihenfolge, aber $s \notin L(m-1)$ versucht $M^{j(k)}$, die Turingmaschine M^s angesetzt auf bin(m) zu simulieren, aber nur solange M^s höchstens $r_{m-s}(1)$ Felder des Bands von M^s

benutzt und höchstens für $|Z_s|\cdot r_{m-s}(1)\cdot|A_s|^{r_{m-s}(1)}$ Schritte (Z_s=Zustandsmenge von M^s, A_s=Alphabet von M^s).

Sobald dies für irgendein s gelingt, so wird $L(m)=L(m-1)\cup\{s\}$ gesetzt. Ist $m<n$, wird m erhöht. Ist $m=n$, wird

$$\begin{cases} 1 & \text{falls } \varphi_s(n)=0 \\ 0 & \text{sonst} \end{cases}$$

ausgegeben.

Gelingt die Simulation für kein s, wird $L(m)=L(m-1)$ gesetzt. Ist $m<n$, wird m erhöht. Ist $m=n$, wird 1 ausgegeben.

$M^{j(k)}$ berechnet offenbar f.

Wir schätzen die Bandkomplexität von $M^{j(k)}$ ab. Rechnungen von $M^s=M_{bin(s)}$ auf $r_{m-s}(1)$ Feldern können nach Satz 2.17. von einer universellen Maschine auf $C\cdot|bin(s)|\cdot r_{m-s}(1)$ Feldern simuliert werden, wobei C eine passende Konstante ist. OBdA kann k groß genug gewählt werden, daß $C\cdot|bin(s)|\leq s$ für alle $s>2k$ gilt. Also werden zur Simulation einer Rechnung von M^s auf $r_{m-s}(1)$ Feldern des Bands von M^s nur $s\cdot r_{m-s}(1)$ Felder des Bands von $M^{j(k)}$ benötigt. Aber

$$\begin{aligned} s r_{m-s}(1) &= 2^{\log s} r_{m-s}(1) \\ &\leq r_{m-s+\log s}(1), \text{ da } r(n)\geq 2n. \end{aligned}$$

OBdA kann k groß genug gewählt werden, daß $s/2\geq\log s$ für alle $s>2k$ gilt. Also $s r_{m-s}(1)\leq r_{m-s/2}(1)\leq r_{n-k}(1)$.

Zum Speichern einer Liste L(m) sind höchstens $O(m\log m)\leq O(n\log n)$ Felder nötig, aber $O(n\log n)\leq 2^{n-k}\leq r_{n-k}(1)$ für fast alle n. □

Beweis von Satz 7.17: Sei $r:\mathbb{N}\to\mathbb{N}$ eine rekursive Funktion, oBdA kann angenommen werden, daß r monoton wächst.

(F,ψ) sei ein beliebiges Komplexitätsmaß, und es gelte

$$f_i=\varphi_{g(i)} \text{ und } \varphi_i=f_{h(i)} \text{ für alle } i,$$

wobei g und h rekursive Funktionen sind. $R:\mathbb{N}^2\to\mathbb{N}$ sei für (F,ψ) und $(F',\psi')=(\varphi,L)$ wie in Satz 7.15. Sei $t(n)=R(2^n,r(R(2^n,n)))$. Nach Lemma 7.19 gibt es eine rekursive Funktion $f:\mathbb{N}\to\mathbb{N}$, so daß es zu jedem Index k mit $\varphi_k=f$ einen Index j mit $\varphi_j=f$ gibt, so daß für fast alle n gilt:

$$t(L_j(n)) \leq L_k(n).$$

Wegen $L_j(n) \geq |bin(n)| \geq \log n$ folgt $2^{L_j(n)} \geq n$ für alle j und n. Also wegen der Monotonie von R im ersten Argument:

$$R(n,r(R(n,L_j(n)))) \leq t(L_j(n)) \leq L_k(n).$$

Wir zeigen, daß f die gewünschten Eigenschaften hat.

Sei i ein Index mit $f_i = f = \varphi_{g(i)}$. Nach Lemma 7.19 gibt es einen Index j mit $\varphi_j = f$ und

$$R(n,r(R(n,L_j(n)))) \leq t(L_j(n)) \leq L_{g(i)}(n).$$

Aus der Monotonie von r und Satz 7.15 folgt:

$$R(n,r(\psi_{h(j)}(n))) \leq L_{g(i)}(n)$$

$$\leq R(n,\psi_i(n)) \text{ für fast alle n.}$$

Wegen der strikten Monotonie von R im zweiten Argument folgt

$$r(\psi_{h(j)}(n)) \leq \psi_i(n) \text{ für fast alle n.}$$

Wegen $f_{h(j)} = \varphi_j = f$ ist damit alles gezeigt. □

Wir bemerken noch, daß im Beweis von Lemma 7.17 nur die Existenz einer Maschine $M^{j(k)}$ mit bestimmten Eigenschaften gezeigt wurde, denn wir haben kein Verfahren zur Berechnung von v angegeben. Man kann in der Tat zeigen, daß man nicht effektiv aus jedem Index i für f einen Index j für f mit $r(\psi_j(n)) \leq \psi_i(n)$ für fast alle n berechnen kann.

7.2.2. Komplexitätslücken

Man kann im abstrakten Fall ebenso wie im konkreten Fall Komplexitätsklassen definieren.

Definition 7.2o: Für rekursive Funktionen $t:\mathbb{N} \to \mathbb{N}$ sei

$$C_{t(n)} = \{f_i \mid \psi_i(n) \leq t(n) \text{ für fast alle n}\}.$$

Wir haben, etwa in den Hierarchiesätzen, monotone rekursive Funktionen $r:\mathbb{N} \to \mathbb{N}$ angegeben, so daß unter gewissen Voraussetzungen über die Gutartigkeit der Funktion t galt: $C_{t(n)} \overset{\subset}{\neq} C_{r(t(n))}$.

Andererseits haben wir auch schon eine Komplexitätslücke kennengelernt (Kapitel 4, Satz 3.9), d.h. eine Situation, wo für $t(n) = O(T(n))$ gilt: $C_{t(n)} = C_{T(n)}$.

Wir zeigen jetzt, daß es in jedem Komplexitätsmaß beliebig große Komplexitätslücken gibt.

Satz 7.21: Zu jeder rekursiven Funktion $r:\mathbb{N}\to\mathbb{N}$ gibt es eine rekursive Funktion $t:\mathbb{N}\to\mathbb{N}$, so daß $C_{t(n)}=C_{r(t(n))}$ gilt.

Beweis: t wird definiert durch

$$t(n)=\min\{m\mid\forall i\leq n:(\psi_i(n)\notin\{m+1,\ldots,r(m)\})\}.$$

t ist partiell rekursiv, denn

$$t(n)=\min\{m\mid\forall i\leq n\forall x\in\{m+1,\ldots,r(m)\}:M(i,n,x)=0\}$$

t ist auch total, denn z.B. für

$$M_n=\max\{\psi_i(n)\mid i\leq n,\psi_i(n)\text{ ist definiert}\}$$

und alle $i\leq n$ ist

$$\psi_i(n)\leq M_n \text{ oder nicht definiert, also}$$

$$\psi_i(n)\notin\{M_n+1,\ldots,r(M_n)\}.$$

Wir weisen $C_{t(n)}=C_{r(t(n))}$ nach, indem wir zeigen, daß jede Schrittzahlfunktion, die durch $r(t(n))$ beschränkt ist, bereits durch $t(n)$ beschränkt ist.

Sei $\psi_j(n)\leq r(t(n))$ und $n\geq j$. $t(n)$ ist eine Zahl m, so daß für alle $i\leq n$ gilt: $\psi_i(n)\notin\{m+1,\ldots,r(m)\}$. Also insbesondere für i=j:

$$\psi_j(n)\notin\{t(n)+1,\ldots,r(t(n))\}.$$

Wegen $\psi_j(n)\leq r(t(n))$ folgt $\psi_j(n)\leq t(n)$. □

Man kann sogar erzwingen, daß die Funktion t in Satz 7.21 monoton ist.

Satz 7.22 "Gap-Theorem": Zu jeder rekursiven Funktion $r:\mathbb{N}\to\mathbb{N}$ gibt es eine monotone rekursive Funktion $t:\mathbb{N}\to\mathbb{N}$, so daß $C_{t(n)}=C_{r(t(n))}$ gilt.

Beweis: t wird definiert durch $t(1)=1$

$$m(n)=\min\{m\mid\forall i\leq n:\psi_i(n)\notin\{t(n)+m+1,\ldots,r(t(n)+m)\}\}$$

$$t(n+1)=t(n)+m(n).$$

Der Rest des Beweises verläuft analog zum Beweis von Satz 7.21 und wird dem Leser als Übung empfohlen. □

7.3. Anhang: Physikalische Grenzen für die Zeitdauer von Schaltvorgängen

Wir haben in diesem Text die Grenzen der Leistungsfähigkeit mechanischer Rechenverfahren untersucht und festgestellt, daß zur Berechnung gewisser Funktionen sehr viele Rechenschritte nötig sind, gleichgültig, welches Verfahren man benutzt. Nun könnte man hoffen, daß es vielleicht physikalisch möglich ist, immer schnellere Schaltelemente und Rechenanlagen zu bauen und auf diese Art den Bereich der Funktionen, die man "praktisch" berechnen kann, beliebig zu erweitern.

Unter ganz einfachen Annahmen über Schaltelemente und mit etwas Optimismus über den Gültigkeitsbereich der bekannten physikalischen Gesetze, zeigt die folgende Überlegung aus [68], daß dies nicht möglich ist und daß jeder Schaltvorgang mindestens $5{,}6 \cdot 10^{-33}$ Sekunden dauert.

Die Annahmen über ein Schaltelement sind:

a) Es besitzt mindestens zwei Zustände.
b) In einem Schaltvorgang (Wechsel des Zustands) wird Energie umverteilt.
c) Das Schaltelement hat endliche Größe.

Bevor irgendein Schaltelement B darauf reagieren kann, daß Schaltelement A geschaltet hat, muß B die Energie in A messen, und zwar mit einem Fehler ΔE, der zumindest kleiner sein muß als E=Maximum der Energie in A vor, während und nach dem Schaltvorgang.

Um Energie mit einem Fehler ΔE zu messen, ist nach Heisenberg's Unschärferelation mindestens Zeit

(*) $t \geq \bar{h}/\Delta E \geq \bar{h}/E$ ($\bar{h}$ = Elementarquantum/π)

nötig.

Wenn das Schaltelement innerhalb einer Kugel vom Radius r eingeschlossen ist, kann die Zeit für einen Schaltvorgang nicht kleiner als

$$t_m = 2r/c \qquad \text{(c: Lichtgeschwindigkeit)}$$

sein.

Das heißt, um die Zeit klein zu machen, muß man sowohl die Energie E groß als auch den Radius r klein machen. Konzentration von zuviel

Energie auf zu wenig Raum führt jedoch zum Gravitationskollaps, und zwar beim Schwarzschild-Radius

$$r_s = 2GE/c^2 \qquad \text{(G: Gravitationskonstante).}$$

Einsetzen von r_s für r in der Formel für t_m liefert

$$t_m = 4GE/c^3$$

$$\geq 4G\bar{h}/t_m c^3 \quad \text{wegen (*).}$$

Also

$$t_m \geq \sqrt{4G\bar{h}/c^3} \approx 5{,}6 \cdot 10^{-33} \text{ sec.}$$

Symbolverzeichnis

$A \setminus B = \{a \in a\ A, a \notin B\}$
$M \times N = \{(a,b) \mid a \in M, b \in N\}$
$M^n = M \times \ldots \times M$ (n-mal)
$\emptyset$ leere Menge
$| \ |$ Mächtigkeit einer Menge
Länge einer Zeichenreihe
Betrag einer Zahl
Σ Summe
log Logarithmus zur Basis 2
□ Ende eines Beweises
[ ] Hinweis auf das Literaturverzeichnis

Im Text definierte Symbole

Kapitel 1

A^+, A^* 1o
$\mathbb{N}$ 1o
$\dot{-}$ 15
$\wedge, \vee, \sim$ 17
$\mu_b f$, μf 18, 22
$\lfloor x \rfloor$, $\lceil x \rceil$ 25
B, 26, 27
δ, z_o, E 27
$\vdash$ 28
$|w|$ 29
bin 34
O, o, Ω, ω 38
ψ 5o

Kapitel 2

χ_L 56
$\overline{L}$ 56
M_u 58
$\leq$ 6o
$\wedge, \vee, \sim, \rightarrow, \exists, \forall$ 72, 73
$H(u\#v)$ 78
ψ 82

Kapitel 3

$c(i)$	88
$t^i(n)$	95
$G=(V,E)$	1o4
ψ	1o8

Kapitel 4

z_+, z_-	113
$L(M)$	113
$DTIME_k(t(n))$, $DTIME(t(n))$	113
$DTAPE(s(n))$	
$C_k(t(n),s(n))$, $C(t(n),s(n))$	114
$\log^*$	123
$CS(w,i)$	135

Kapitel 5

δ, $\vdash$	142
$L(M)$	144
$NTIME_k(t(n))$, $NTIME(t(n))$	
$NTAPE(s(n))$	145
$\vdash^t_s$	147
r_G	156
$\rightarrow$, $\vec{G}$, $\vec{G}^*$	171
$L(G)$	171

Kapitel 6

P, NP, PTAPE, EXPTIME,	
EXPTAPE	181
$\leq_p$, $\equiv_p$	182
$\wedge,\vee,\sim,\exists,\forall$	186, 199
φ	187, 2oo
"$V_i \vdash V_j$"	193

Kapitel 7

M^i	221
φ^i	221
$M(i,n,m)$	224
(F,ψ)	225
L_i	231

Literatur

Abkürzungen:

ACM-STOC : ACM Symposium on Theory of Computing
IEEE-SWAT : IEEE Symposium on Switching and Automata Theory
IEEE-FOCS : IEEE Symposium on Foundations of Computer Science
J.ACM : Journal of the ACM
J.CSS : Journal of Computer and System Sciences

[1] Ackermann, W.: Zum Hilbert'schen Aufbau der reellen Zahlen, Mathematische Annalen 99, 118-133, 1928.

[2] Aho, Hopcroft, Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[3] Blum, M.: A machine independent theory of the complexity of recursive functions, J.ACM, 14, 322-336, 1967.

[4] Chomsky, N.: Three models for the description of language, PGIT, 113-124, 1965.

[5] Cohen, P.: Set Theory and the Continuum Hypothesis, W.A. Benjamin, Inc., Reading, Mass., 1966.

[6] Cook, S.A.: The complexity of theorem proving procedures, 3rd ACM-STOC, 151-158, 1971.

[7] Cook, S.A.: A hierarchy for nondeterministic time complexity, J.CSS, 7, 343-353, 1973.

[8] Cook, S.A.: An observation of time-storage trade off, 5th ACM-STOC, 29-33, 1973.

[9] Cook, Reckhow: Time-bounded random access machines, J.CSS, 7, 354-375, 1973.

[1o] Davis, M.: Computability and Unsolvability, Mc Graw-Hill, New York, 1958.

[11] Eilenberg, Elgot: Recursiveness, Academic Press, New York, 197o.

[12] Erdös, Graham, Szemeredi: On sparse graphs with dense long paths, Technical report, Stanford University.

[13] Even, Tarjan: A combinatorial problem which is complete in polynomial space, 7th ACM-STOC, 66-71, 1975.

[14] Fischer, M.J., Rabin: Super-exponential complexity of Presburger arithmetic, MAC Technical memorandum 43, M.I.T. 1974.

[15] Galil, Z.: Two fast simulations which imply some fast string matching and palindrome-recognition algorithms, Information Processing letters, 85-87, 1976.

[16] Garey, Graham, Johnson: Some NP-complete geometric problems, 8th ACM-STOC, 1o-22, 1976.

[17] Garey, Johnson, Stockmeyer: Some simplified NP-complete problems, 6th ACM-STOC, 47-63, 1974.

[18] Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, Monatshefte für Mathematik und Physik, 38, 173-198, 1931.

[19] Hartmanis, J.: Size arguments in the study of computation speeds, Proc. Symp. on Computers and Automata Polytechnic Inst. of Brooklyn, 1-18, 1965.

[2o] Hartmanis, J.: Computational complexity of one-tape Turing machine computations, J.ACM, 15, 325-339, 1968.

[21] Hartmanis, J.: Relations between Diagonalization, proof systems and complexity Gaps, 9th ACM-STOC, 223-227, 1977.

[22] Hartmanis, Hopcroft: An overview of the theory of computational complexity, J.ACM, 18, 444-475, 1971.

[23] Hartmanis, Hunt: The LBA problem and its importance in the theory of computing, SIAM-AMS Proceedings, 7, 1-26, 1974.

[24] Hartmanis, Lewis, Stearns: Hierarchies of memory limited computations, IEEE-Conference Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 179-19o, 1965.

[25] Hartmanis, Simon: On the power of multiplication in random access machines, 15th IEEE-SWAT, 13-23, 1974.

[26] Hartmanis, Stearns: On the computational complexity of algorithms, Trans. Am. Math. Soc., 117, 288-3o6, 1965.

[27] Hennie, F.C.: One-tape off-line Turing machine computations, Information and Control, 8, 553-578, 1965.

[28] Hennie, Stearns: Two-tape-simulation of Multitape Turing machines, J.ACM, 13, 533-546, 1966.

[29] von Heijenoort, J.: From Frege to Gödel (A source Book in Mathematical Logic, 1919-1931) Harvard University Press, Cambridge, Mass., 1967.

[3o] Hopcroft, Paul, Valiant: On time versus space and related problems, 15th IEEE-FOCS, 57-64, 1975.

[31] Hopcroft, Paul, Valiant: On Time versus Space, J.ACM, 24, 332-337, 1977.

[32] Hopcroft, Ullman: Formal languages and their relation to automata, Addison Wesley, Reading, Mass., 1969.

[33] Hopcroft, Ullman: Some results on tape-bounded Turing machines, J.ACM, 16, 168-177, 1969.

[34] Hotz, Walter: Automatentheorie und formale Sprachen I, B.I.-Hochschulskripten, Mannheim, 1968.

[35] Hunt, H.B. III: Non-linear time bounds on the complexity of regular expression problems, preprint Harvard University, Cambridge, Massachusetts, 1976.

[36] Karp, R.M.: Reducibility among combinatorial problems, Complexity of computer computations (R. Miller and J. Thatcher, ed.) Plenum Press 1972, 85-1o4.

[37] Kleene, S.C.: Introduction to Metamathematics, North Holland Publishing Company, Amsterdam, London 1971.

[38] Lipton, Tarjan: A separator theorem for planar graphs, Preprint, Stanford University, 1977.

[39] Lipton, Tarjan: Applications of a planar separator theorem, 18th IEEE-FOCS, 162-17o, 1977.

[4o] Manders, Adleman: NP-complete decision problems for quadratic polynomials, 8th ACM-STOC, 23-29, 1976.

[41] Mehlhorn, K.: Komplexitätstheorie, Vorlesungsskript, Saarbrücken 1976.

[42] Mehlhorn, K.: Effiziente Algorithmen, Teubner 1977.

[43] Meyer, Stockmeyer: Word problems requiring exponential time, 5th ACM-STOC, 1-9, 1973.

[44] Minsky, M.L.: Recursive unsolvability of Post's problem 'Tag' and other topics in the theory of Turing machines, Ann. of Math., 74, 437-455, 1961.

[45] Myhill, J.: Linear bounded automata, WADD Technical note 6o-165, 196o, Wright Patterson Air Force Base Ohio.

[46] Oppen, D.C.: Elementary bounds for Presburger arithmetic, 5th ACM-STOC, 34-37, 1973.

[47] Paterson, M.S.: Tape bounds for time-bouned Turing machines, J.CSS, 6, 116-124, 1972.

[48] Paterson, Hewitt: Comparative Schematology, Project MAC Conf. on concurrent systems and parallel computation, Woods Hole MA, 119-127, 197o.

[49] Paul, W.J.: On Time Hierarchies, 9th ACM Symposium on Theory of Computing, 218-222, 1977.

[5o] Paul, Tarjan: Time-space trade-off in a pebble game, technical report, Stanford University 1977.

[51] Paul, Tarjan, Celoni: Space bounds for a game on graphs, Math. Syst. Theory, 1o, 239-251, 1977.

[52] Peter, R.: Rekursive Funktionen, Academic Press, New York, 1967.

[53] Pippenger, N.: A time-space trade-off, technical report, IBM Yorktown Hights, 1977.

[54] Pippenger, Fischer: Relations among complexity measures, Preprint IBM Yorktown Hights, 1977.

[55] Pratt, Stockmeyer, Rabin: A characterization of the power of vector machines, 6th ACM-STOC, 122-134, 1974.

[56] Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welcher die Addition als einzige Operation hervortritt, Comptes-rendus du I Congres des mathematiciens des pays slaves, 1929.

[57] Rogers, H.: Theory of Recursive Functions and Effective Computability, Mc Graw-Hill, New-York, 1967.

[58] Ruby, P.C. Fischer: Translational methods and computational complexity, IEEE-Conference Record on Switching Theory and Logical Design, Ann Arbor, Michigan, 173-178, 1965.

[59] Savage, J.: The complexity of computing, John Wiley & Sons, New York, 1976.

[6o] Savage, Swamy: Space-time tradeoffs on the FFT algorithm, Technical Report, Brown University, 1977.

[61] Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities, J.CSS 4, 177-192, 197o.

[62] Schaefer, T.J.: Complexity of finite two-person perfect-information games, 8th ACM-STOC, 41-49, 1976.

[63] Schnorr, C.P.: The network complexity and the Turing machine complexity of finite functions, Acta Informatica 7, 95-1o7, 1976.

[64] Schnorr, C.P.: Rekursive Funktionen und ihre Komplexität, Teubner, Stuttgart, 1974.

[65] Seiferas, M.J. Fischer, A.R. Meyer: Refinements of the nondeterministic time and space hierarchies, IEEE SWAT, 13o-137, 1973.

[66] Sethi, R.: Complete Register Allocation Problems, SIAM J. COMPUT. 4, 226-248, 1975.

[67] Simon, J.: On feasible numbers, 9th ACM-STOC, 195-2o7, 1977.

[68] Simon, J.: Physical limits on the speed of computing, technical report, universidade estadual de Campinas, Brasil.

[69] Specker, Strassen: Komplexität von Entscheidungsproblemen, Springer Lecture Notes in Computer Science, 43, 1976.

[7o] Stockmeyer, L.J.: Thesis, M.I.T., Project MAC.

[71] Tracktenbrot, B.A.: Turing computations with logarithmic delay, Algebra i Logika 3,4, 33-48, 1964.

[72] Valiant, L.G.: Generel context-free recognition in less than cubic time, J.CSS 1o, 3o8-315, 1975.

[73] Walter, H.: Rekursive Funktionen, Vorlesung, Saarbrücken WS 1971/72

[74] Younger, D.H.: Recognition and parsing of context-free languages in time n^3, Information and Control 1o, 189-2o8, 1967.

Sachverzeichnis

Ableitung 14, 171
Abstrakte Komplexitäts-
-Klassen 234
-Maße 225
abzählbar 1o
Ackermann-Funktion 24
Adresse 86
ähnlich 221
Äquivalenz
-Boole'scher Ausdrücke 187
-Boole'scher Prädikate 2o1
-polynomielle 182
Akkumulator 87
Akzeptor, akzeptieren,
akzeptierte Sprache 113, 143
Algorithmus 8
Alphabet 1o, 25, 26, 171
Anfangszustand 26, 27
Arbeitsband 133
arithmetische
-Prädikate 72
-Aussagen 73
Aufzählung 221
-zulässige 222

Band 25
-alphabet 25
-beschränkt 3o, 133, 144
-hierarchiesätze 118, 133
-inschrift 26
-konstruierbar 117
-verbrauch 29
Baum 1o5
Befehl 87
-szähler 87
-sliste 88, 9o
Belegung 73, 187
-sspiel 21o

Berechenbarkeit 78
Δ-Berechnungsgraph 159
beschränkter μ-Operator 18
Beweis 69, 71
-system 71
Binärdarstellung 29
Blatt 1o5
Block 1o7, 157
Blum 226
Bool'sche
-Algebra 187
-Ausdrücke 186
-Aussagen 2oo
-Prädikate 199

charakteristische Funktion 56
Chomsky
-Grammatik 171
-Sprache 171
Church'sche These 53
Clique 194
Cook 189
Crossing-Sequenz 135

Diagonalisierung 12, 59, 117
Disjunktion 187

effiziente Reduzierbarkeit 181
Eingabeband 86, 132
Einsetzung 13
endlicher Automat 178
endliches Gedächtnis 36
Endkonfiguration 27
Endzustand 26, 27
entscheidbar 56
erfüllbar 187, 2oo
Erfüllbarkeitsproblem 189
EXPTIME 181

frei 16, 72, 2oo
Δ-Fragment 158
Funktionen
-berechenbare 8
-partielle 21
-partiell rekursive 221
-primitiv rekursive 13
-rekursive 221
-μ-rekursive 22
-totale 21
-turingberechenbare 35

Gap-theorem 234
gebunden 16, 72, 2oo
gelten 2oo
Geographie 2o9
Gewinnstrategie 2o9
Go 213
Gödel 78, 8o
-isierung 57, 117
-nummer 58
Graph 1o9, 194
gutartig 111, 117, 119

halten 26, 28
Halteproblem 59, 6o
hart 183
Hennie 125
herleitbar 71
Hierarchiesätze 116, 167
-Band 118, 133
-Zeit 119, 167

implementieren 1o6
(Suchbaum)
Ingrad 152
innere Knoten 1o5

kanonische Rechnung 29
Kante 1o4, 194
Klausel 188
Kleene'sche Normalform 53
Knoten 1o4, 194
Komplement 56
Komplexitäts
-lücken 183, 234
-klassen 113, 114, 122, 133, 145
234
-maße 3o, 91, 92, 93, 133, 144,
225
Konfiguration 27
Konjunktion 187
konjunktive
-Normalform 188
-Pränex Normalform 2o2
Konkatenation 5o
konsistent 76
ω-konsistent 84
konstante Funktionen 13
kontext
-frei 172
-sensitiv 172
korrekt 76

Länge 27, 29, 143, 186, 199, 215
LBA-Problem 179
leeres Wort 1o
Leersymbol 26
lineare Zeit 185
Literal 188
Logarithmische
-Zeitkomplexität 91
-Platzkomplexität 93
-s Band 185
Logarithmus 1o4

Markenspiel 152

Nachfolge
-konfiguration 28, 143

-r 152
-rfunktion 13
Negation 187
Nichtdeterminismus 142
Nichtterminalalphabet 171
NP 181
-vollständig 183

oblivious 131
μ-Operator 2o
-beschränkter 18
Off-line-Turingmaschine 132

P 181
Padding 129
partiell 21
-rekursiv 221
Pfad 1o5
Platzkomplexität 93
Platzbeschränkt 93
polynomiell
-äquivalent 182
-reduzierbar 182
-verknüpft 94, 1o1
-e Zeit 182
potentielle
-r Berechnungsgraph 159
-s Fragment 158
-s Resultat 158
Prädikat 16, 72, 199
Pränex-Normalform 2o2
Presburger Arithmetik 215
primitiv rekursive
-Bijektionen 19
-Funktionen 13
-Prädikate 16
Produktionensystem 171
Programm 86
-stück 99
Projektion 13
Protokoll 146
PTAPE 181
-vollständig 183
Pushdown
-automat 1o3
-band 1o2

Quelle 1o4

raten 143
Rechenschritt 26
Rechnung 28, 143
-kanonische 29
reduzierbar 6o
-lintime 184
-logtape 185
-polynomiell 182
Register 86
-maschine 86
regulär 35, 172
rekursiv 221
-aufzählbar 61
μ-rekursiv 22
Resultat 26
Δ-Resultat 158

Savitch 146
Schalt
-funktion 197
-kreis 197
Schlußregel 69, 71
Schrittzahlfunktion 226
segmentierter Zähler 123
Senke 1o4
Simulation 38
Sohn 1o5
Speed-up theorem 231
Speicher 86
Spielsituation 2o8

Sprache 71, 113, 144, 171
Spur 39
Standardaufzählung 221
Startsymbol 171
Stearns 125
Δ-Stenogramm 158
Substitution 13
Suchbaum 1o6

Term 72
Terminalalphabet 171
Tiefe 1o5, 169
Time-Space-Trade-Off 137
totale Funktionen 21
turingberechenbar
Turingmaschine
-deterministische 25, 27
-nichtdeterministische 142
-Off-line 132
-universelle 62

Übergangs
-funktion 26, 27
-relation 142
uniforme
-Zeitkomplexität 92
-Platzkomplexität 92
Unvollständikeit 68, 77
-ssätze 78, 8o

Vater 1o5
verwerfen 113, 143
vollständige
-Bäume 1o5
-Beweissysteme 77
-konjunktive Normalform 189
-Probleme 183
Vorgänger 151
-liste 156

wahr 69, 73
Wort 1o
-leeres 1o
Wurzel 1o5

Zähler 1o3, 121
-automat 1o3
Zeit
-beschränkt 3o, 92, 144
-hierarchiesätze 12o, 122, 123, 128, 129, 167
-intervall 157
-konstruierbar 119
-komplexität 3o, 91, 92
Zentrum 121
zulässige Aufzählung 222
Zyklus 1o5