

Burchard v. Braunmühl

# Kleines Skript der Komplexitätstheorie I

## 1 Einführung

In der Rekursionstheorie geht es um die Frage, ob ein Problem oder eine Aufgabe überhaupt mit Hilfe eines Rechners lösbar, also berechenbar ist. Wir wissen, daß dies für verschwindend wenige Probleme der Fall ist. Diese Zahl wird noch einmal gewaltig reduziert, wenn wir verlangen, daß das Problem auch noch in einem vernünftigen Aufwand gelöst werden soll.

Es ist die Komplexitätstheorie, die diesen Aufwand von Problemen betrachtet: wie komplex ist ein Problem im Sinne eines Aufwandsmaßes. Will man den Aufwand messen, muß man konkretisieren, auf was für einem Rechner man das Problem lösen will und was man unter Aufwand versteht, was also das Maß sein soll. Als Rechnermodell könnte man einen PC mit Pentium II und Windows 2000 nehmen oder auch eine Cray, als Maß die Zahl an Sekunden, die das Programm benötigt. Da wird man recht verschiedene Meßzahlen bekommen und wenig Aussage über das Problem selbst.

Was wir wünschen ist ein Modell, das nicht in 10 Jahren überholt ist und Maßeinheiten, die nicht von der Implementierung oder dem Betriebssystem abhängen. Wollen wir also von diesen veränderlichen Größen abstrahieren, nehmen wir einen gewissen Verlust an Anschaulichkeit und Praxisnähe in Kauf.

Modelle die sich anbieten und verwandt werden sind die Turingmaschine (TM) oder die Random Access Maschine (RAM), Maße, die man untersucht, sind u. a. die Zeit und der Speicherplatz. Was diese Maße bedeuten, hängt wieder vom Modell ab. Bei der Turingmaschine zählt man die Zahl der Elementaroperationen (die Schritte) und die Zahl der kleinsten Speicherelemente (die Bandfelder).

Außer der größeren Abstraktion liegen die Vorzüge dieser Modelle auch in ihrer Einfachheit, was vor allem dann von Bedeutung ist, wenn man zeigen will, daß es für gewisse Probleme überhaupt kein Lösungsprogramm geben kann oder doch nur eines, dessen Zeit- oder Speicheraufwand über einer (vernünftigen) Schranke liegt. Ein solcher Beweis kann nämlich nur geführt werden, in dem man auch die Möglichkeiten einer solchen Maschine berücksichtigt, was für aufwendige Maschinen eben selbst entsprechend aufwendig ist.

In der Komplexitätstheorie gibt es unterschiedliche Gebiete, die jeweils andere Aspekte des Aufwands von Problemen behandeln. Das große Gebiet der "Effizienten Algorithmen", umfaßt alle Bemühungen um zeiteffiziente Lösungsverfahren für spezielle Probleme, die meist aus der Praxis stammen und für die triviale Lösungsverfahren zu aufwendig sind. Da diese Probleme aus allen möglichen Anwendungsbereichen stammen können, zerfällt das Gebiet der "Effizienten Algorithmen" selbst wieder in Teile, die ganze Lehrgebiete umfassen, wie Computeralgebra, Computational Geometry, Optimierung etc.

Der größte Teil der Bemühungen richtet sich sicherlich auf das Finden von guten Algorithmen und den Beweis ihrer Güte. Hat man einen Algorithmus für ein Problem gefunden, von dem man weiß oder zeigen kann, daß sein Aufwand unter einer gewissen Schranke liegt, so hat man eine obere Aufwandsschranke für dieses Problem gefunden: man kann es lösen mit einem Aufwand, der unter dieser Schranke liegt.

Ein anderer Teil der Bemühungen allerdings richtet sich auf eine konträre Untersuchung: was ist der Mindestaufwand, der zur Lösung eines Problems getrieben werden muß, egal welche Verfahren, Strategien und Prinzipien man verfolgt. Dies wäre die untere Aufwandsschranke des Problems. Kann man beweisen, daß das Problem in weniger Aufwand nicht lösbar ist, so erübrigt sich die Jagd nach Algorithmen, die diese Schranke unterbieten. Es kann auch bedeuten, daß dieses Problem zwar theoretisch,

aber nicht praktisch lösbar ist, weil eine Lösung zu aufwendig wäre. Eine Lösung eines Problems von vernünftiger Größe kann ein Jahr, aber auch  $10^{30}$  Jahre dauern. Solange wird aber niemand warten wollen.

Haben wir uns für ein Modell und ein Maß entschieden, können wir die Problemklassen betrachten, die innerhalb eines gewissen vorgegebenen Aufwands gelöst werden können, für die dieser Aufwand also eine obere Schranke darstellt. Die Struktur, die diese Klassen bzgl. der Inklusion bilden, ist das Thema der "Strukturellen Komplexitätstheorie".

Ein drittes Gebiet betritt man, wenn man sich fragt, welche Sätze allen vernünftigen Modellen und Maßen gemeinsam sind. Was ein vernünftiges Modell und Maß ist, wird durch zwei elementare, einleuchtende Eigenschaften (Axiome) festgelegt. Gelten diese Axiome, so kann man allein aus ihnen - ohne nähere Kenntnis des Maßes - gewisse Sätze folgern, die dann für alle Maße gelten, die diesen Axiomen genügen. So nennt man dieses Gebiet "Axiomatische Komplexitätstheorie" oder "Abstrakte Komplexitätstheorie".

Ein algorithmisch angehbares Problem kann man immer als eine Funktion auffassen: eine Instanz, ein Fall des Problems wird als Argument (Input) gegeben, die Lösung ist der Wert der Funktion (der Output). Wollen wir in einem Graphen den kürzesten Weg finden, so ist der Graph das Argument und der kürzeste Weg der Wert. Sucht man das Vorkommen eines Wortes in einem Text, so ist der Text und das Wort das Argument, die Position im Text der Wert. Abstrakt gesehen lassen sich alle Probleme als Funktionen auffassen, die Worte (die Kodifizierung der Instanz des Problems) in Worte (die Kodifizierung der Lösung) überführen. Ein Rechner verarbeitet im Grunde nur Bitfolgen. Alle Probleminstanzen müssen also kodiert werden als eine Folge von Bits oder - wie wir auch sagen können - ein Wort über dem Alphabet  $\{0, 1\}$ . Ebenso müssen wir die Lösung aus einem 0, 1-String extrahieren bzw. semantisch herausdeuten.

Wir wollen uns hier - pars pro toto - auf ja-nein-Probleme beschränken, d.h. auf Probleme, deren Lösung in einem Ja oder Nein besteht. Abstrakt gesehen sind das 0,1-wertige Funktionen, d.h. partielle Funktionen aus  $\{0, 1\}^*$  in  $\{0, 1\}$ . Wir können es auch so formulieren, daß wir uns mit dem Aufwand des Erkennens von Sprachen aus  $\{0, 1\}^*$  beschäftigen werden.

Gemessen wird der Aufwand eines Problems immer in der Größe der Problemfalls. Für die Zeit, die die Multiplikation zweier Zahlen benötigt, spielt die Größe der Zahlen eine Rolle. In einem großen Graphen ist der kürzeste Weg schwerer zu finden als in einem kleinen. Darum ist der Aufwand eines Problems immer eine Funktion von der Instanzgröße. Die Größe einer Instanz wiederum messen wir hier einfach in der Länge ihrer Darstellung als Bitfolge, die als Input eingegeben wird. So ist also der Aufwand immer eine (totale) Funktion von  $\mathbb{N}$  nach  $\mathbb{N}$ .

## 2 Grundlagen

Zunächst wollen wir die Modelle und Maße definieren, die hier eine Rolle spielen. Wir setzen voraus, daß die Grundbegriffe der Theoretischen Informatik bekannt sind (Alphabet, Worte, Sprache, Finiter Automat etc.).

### 2.1 Turingmaschinen

Eine 1-Band-Turingmaschine kann man sich anschaulich vorstellen als ein unendlich langes Band, das in diskrete Felder eingeteilt sind, die jedes ein Zeichen tragen können. Auf dem Band läuft ein Schreib-Lese-Kopf, der von einer Kontrolleinheit (KE) gesteuert wird. Der Kopf steht auf einem der Felder, liest das dort stehende Zeichen und gibt es an die KE weiter. Die sagt ihm, was er auf das

Feld drucken - wie er also das gelesene Zeichen ändern - und in welche Richtung er weiterrücken soll (dabei kann er auf dem Feld stehen bleiben oder auf das Feld links oder rechts von ihm gehen). Die KE hat Zustände, reagiert also ja nach Zustand anders auf das gelesene Zeichen. Man kann sich die KE durch eine sequentielle Schaltung realisiert denken, die das vom Kopf gelesene Zeichen als Input bekommt und das zu druckende Zeichen zusammen mit dem Rückbefehl als Output ausgibt. Das Verhalten von sequentiellen Schaltungen läßt sich durch Mealy-Automaten (finite Automaten mit Ausgabe) beschreiben. Also denken wir uns die KE abstrakter als einen finiten Automaten mit Ausgabe.

Ein finiter Automat, der spontane Zustandübergänge hat, also Schritte ohne Eingabe und ohne Ausgabe ausführt (durch  $\varepsilon$ -Befehle ausgelöste  $\varepsilon$ -Schritte), kann immer umgewandelt werden in einen äquivalenten, der keine solche  $\varepsilon$ -Schritte ausführt, der also eine Folge von Schritten ohne Eingabe und Ausgabe in einem einzigen Schritt erledigt. Wenn wir also davon reden, daß die KE etwas intern ausführt, so kann man davon ausgehen, daß dies in einem einzigen Schritt erledigt werden kann. Außerdem können wir annehmen, daß eine TM ihren Kopf immer bewegt oder druckt.

Durch die Zustände kann sich die KE einen endlichen Umfang an Information merken. Dies legt eine spezielle Vorstellung von den Zuständen nahe: die KE hat einen eigenen Arbeitsspeicher, ein gesondertes Zustandsfeld, auf dem sie genau ein Zeichen - das Zustandszeichen - lesen und drucken kann. Die Zahl der Zustandszeichen bestimmt also die Menge an Informationen, die die KE auf diese Weise abspeichern kann. Wählen wir z.B.  $k$ -Tupel von Arbeitssymbolen als Zustandsymbole, so können wir die KE sogar eine Rechnung auf einem Bandstück der Länge  $k$  simulieren lassen..

Die KE kann laso auch durch eine kombinatorische Schaltung realisiert werden, die als Input das Zustandssymbol und das gelesene Bandsymbol (das Arbeitssymbol) bekommt, und als Ausgabewert das neue, zu druckende Zustandssymbol, das neue, zu druckende Bandsymbol und die Kopfrückung liefert. Die Abstraktion der KE ist dann eine Funktion von Zustandsalphabet  $\times$  Arbeitsalphabet in Zustandsalphabet  $\times$  Arbeitsalphabet  $\times$  Kopfrückungen.

Diese Vorstellungen sind nützlich, wenn wir den Ablauf eines Turingprogramms verbal beschreiben, wie wir das in der Regel machen werden. Manchmal sind die einen Vorstellungen hilfreich, manchmal die anderen. Hat man das Gefühl zu schwimmen, so sollte man sich auf die formalen Definitionen zurückziehen.

### Definition 2.1 (Wort)

Endliche Menge nennen wir auch Alphabete. Sei  $\Sigma$  ein Alphabet. Die Folge  $w = a_1 \cdots a_n$  heißt *Wort über  $\Sigma$*  der Länge  $n$ , wenn  $a_i \in \Sigma$  für alle  $i \in [1, n]$ . Für die Länge  $n$  von  $w$  schreiben wir  $|w|$ . Das Wort der Länge 0 nennen wir das leere Wort und schreiben dafür  $\varepsilon$ <sup>1</sup>. Wir definieren  $\Sigma^n =$  Menge der Worte der Länge  $n$  über  $\Sigma$ , insbesondere  $\Sigma^0 = \{\varepsilon\}$  und  $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n =$  Menge aller Worte über  $\Sigma$ . Auf dieser Menge  $\Sigma^*$  definieren wir eine Operation, die *Konkatenation*, die zwei Worte  $u$  und  $v$  zu einem Wort  $uv$  verknüpft, das durch Hintereinanderschreiben entsteht: sind  $u = a_1 \cdots a_n$  und  $v = b_1 \cdots b_m$  zwei Worte über  $\Sigma$ , so ist ihre Konkatenation  $uv = c_1 \cdots c_{n+m}$  mit  $c_1 \cdots c_n = a_1 \cdots a_n$  und  $c_{n+1} \cdots c_{n+m} = b_1 \cdots b_m$ . Insbesondere gilt für jedes Wort  $w \in \Sigma^*$ :  $w\varepsilon = \varepsilon w = w$ .

Jede Menge von Worten  $A \subseteq \Sigma^*$  nennen wir eine Sprache über  $\Sigma$ . Für die leere Sprache schreiben wir  $\emptyset$  (dasselbe Zeichen, wie für die leere Menge, da die leere Sprache eben die (eine) leere Menge ist). Wir verallgemeinern die Konkatenation von Worten auf Sprachen und definieren für Sprachen  $A$  und  $B$  über  $\Sigma$ :  $A \cdot B = \{w \mid \exists u \in A, v \in B : w = uv\}$ . Insbesondere gilt:  $A \cdot \emptyset = \emptyset \cdot A = \emptyset$ . Weiter definieren wir die iterierte Konkatenation:  $A^0 = \{\varepsilon\}$ ,  $A^{n+1} = A^n \cdot A$ ,  $A^* = \bigcup_{n \in \mathbb{N}} A^n$  und  $A^+ = \bigcup_{n \in \mathbb{N}_+} A^n$  ( $\mathbb{N}_+ = \mathbb{N} - \{0\}$ ).

Wir benützen ein fest vorgegebenes, universelles Sondersymbol, das Blank (i.Z.  $\square$ ). Wir verwenden

<sup>1</sup>Wir verabreden, daß  $a_1 \cdots a_0$  bzw.  $a_0 \cdots a_{-1}$  das leere Wort meine, genauso wie  $\{1, \dots, 0\} = \{0, \dots, -1\} = \emptyset$



Hülle von  $\vdash_M$  schreiben wir  $\vdash_M^*$ , also  $s \vdash_M^* s'$ , wenn  $s'$  aus  $s$  durch Anwendung einer Folge von Befehlen hervorgeht, also wenn es Situationen  $s_0, \dots, s_n$  gibt mit  $s = s_0$  und  $s' = s_n$  und  $s_i \vdash_M s_{i+1}$  für alle  $i \in \{0, \dots, n-1\}$ <sup>2</sup>. Die Folge  $s_0, \dots, s_n$  der durchlaufenen Situationen nennen wir einen *Lauf* der Länge  $n$  von  $s$  nach  $s'$  (der Lauf hat zwar  $n+1$  Situationen, aber es werden nur  $n$  Befehle ausgeführt,  $n$  Schritte getan).

Die TM  $M$  stoppt in einer Situation  $s$ , wenn in  $s$  kein Befehl anwendbar ist. Wir nennen  $s$  dann eine *Stopsituation*. Ist der Zustand einer Stopsituation akzeptierend (aus  $F$ ), so ist  $s$  eine *akzeptierende Situation*, andernfalls ist  $s$  eine *verwerfende Situation*. Die TM startet immer in einer Situation der Form  $(q_0, \# w)$  mit  $w \in \Sigma^*$ . Diese Situation nennen wir auch die zum Input  $w$  gehörige *Startsituation*  $s(w)$ .

Ein Lauf ist *maximal*, wenn er nicht verlängerbar ist (mit einer Stopsituation endet) oder wenn er unendlich ist. Ein *w-Lauf* ist ein maximaler Lauf, der mit der zu  $w$  gehörigen Startsituation  $s(w)$  beginnt.

Wir nennen die TM *deterministisch (DTM)*, wenn es keine zwei Befehle aus  $\delta$  gibt, die in den ersten 2 Symbolen (Zustand, gelesenes Zeichen) übereinstimmen. D.h. für eine deterministische TM gibt es in jeder Situation höchstens einen anwendbaren Befehl. Das bedeutet auch, daß es für jeden Input  $w$  genau einen *s(w)-Lauf* gibt.

Die TM  $M$  *erkennt* die Sprache  $L(M) = \{w \in \Sigma^* \mid s(w) \vdash_M^* s \text{ für eine akzeptierende Stopsituation } s\}$ . Die TM akzeptiert also den Input  $w$ , wenn wenigstens ein *s(w)-Lauf* in einer Situation mit akzeptierendem Zustand endet. Wir nennen  $L(M)$  auch die Sprache der TM  $M$ .

Ist die TM  $M$  deterministisch, so ist die von  $M$  *berechnete n-stellige Funktion* ist:  $f_M^n : (\Sigma^*)^n \rightarrow \Sigma^*$  mit  $f_M^n(u_1, \dots, u_n) = w$  gdw.  $(q_0, \#u_1 \square \dots \square u_n) \vdash_M^* (q, \# w)$  für ein  $q \in F$ . Die von der TM  $M$  berechnete Funktion ist also auf dem Input definiert, wenn  $M$  in einem akzeptierenden Zustand stoppt und der Bandinhalt dann ein Wort aus  $\Sigma^*$  ist und der Kopf auf dem ersten Zeichen dieses Wortes steht. Dieses Wort ist dann der Output bzw. das Bild des Inputs unter der Funktion. In allen anderen Fällen sei die von der TM berechnete Funktion nicht definiert.

### Notation 2.3

Manchmal ist es bequem, sich die Felder eines Bandes durch  $\mathbb{Z}$  nummeriert vorzustellen. Dann wollen wir dem Feld, auf dem der Kopf in der Startsituation steht, die Nummer 1 geben. Damit sind die Nummern der übrigen Felder festgelegt: von links nach rechts in der Ordnung von  $\mathbb{Z}$ . Der Input beginnt somit auf Feld 1.

### Aufgabe 2.4

Zeigen Sie, daß es zu jeder NTM eine äquivalente DTM gibt (die dieselbe Sprache erkennt).

### Beispiel 2.5

**Eine TM, die die Binärdarstellung von  $n$  in die von  $n+1$  überführt:** Input  $\text{bin}(n)$ , Output  $\text{bin}(n+1)$  ( $\text{bin}(n)$  = Binärdarstellung von  $n$ ),  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  mit  $q_0 = 0$ ,  $F = \{3\}$  und Befehlstafel  $\delta$  (wir schreiben für  $d = -1, 0, 1$  auch  $L, N, R$ ):

<sup>2</sup>Beachte die Verabredung  $\{0, \dots, -1\} = \emptyset$

$p$	$a$	$p'$	$a'$	$d$	
0	0	0	0	$R$	
0	1	0	1	$R$	laufe über den Input
0	$\square$	1	$\square$	$L$	
1	0	2	1	$L$	
1	1	1	0	$L$	addiere 1 mit Übertrag
1	$\square$	2	1	$L$	
2	0	2	0	$L$	
2	1	2	1	$L$	laufe zum Anfang des Outputs
2	$\square$	3	$\square$	$R$	

**Beispiel 2.6**

**Eine TM, die  $1^n$  in  $1^{2n}$  überführt:** Input  $1^n$ , Output  $1^{2n}$ ,  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  mit  $q_0 = 0$ ,  $F = \{6\}$  und Befehlstafel  $\delta$  (wir schreiben für  $d = -1, 0, 1$  auch  $L, N, R$ ):

$p$	$a$	$p'$	$a'$	$d$	
0	1	1	$\square$	$R$	löscht die erste 1
0	$\square$	6	$\square$	$R$	stoppt
1	1	1	1	$R$	läuft über das nächste $\square$ rechts
1	$\square$	2	$\square$	$R$	
2	1	2	1	$R$	druckt 1 auf das nächste $\square$ rechts
2	$\square$	3	1	$R$	
3	$\square$	4	1	$L$	druckt eine (zweite) 1
4	1	4	1	$L$	läuft über das nächste $\square$ links
4	$\square$	5	$\square$	$L$	
5	1	5	1	$L$	läuft vor nächste $\square$ links
5	$\square$	0	$\square$	$R$	

**Definition 2.7 (Mehrband-Turingmaschine)**

Seien  $Q$  und  $\Gamma$  endliche Alphabete (Zustandsalphabet und Arbeitsalphabet),  $\Sigma \subseteq \Gamma$  (Inputalphabet),  $q_0 \in Q$  (Startzustand),  $F \subseteq Q$  (Menge der akzeptierenden Zustände),  $\square$  ein Sondersymbol nicht aus  $\Gamma$  (bezeichnet ein leeres Feld) und  $\delta \subseteq Q \times \Gamma_o^k \times Q \times \Gamma_o^k \times \{-1, 0, 1\}^k$  (Menge der Befehle).

Dann ist die Struktur

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

eine nichtdeterministische  $k$ -Band-Turingmaschine ( $k$ -NTM).

Die *Situation*, in der sich die Turingmaschine befindet, wird beschrieben durch  $(q, u_1\#v_1, \dots, u_k\#v_k) \in Q \times (\Gamma_o^* \{ \# \} \Gamma_o^+)^k$  ( $\# \notin \Gamma$ ), wobei  $q$  der Zustand der TM ist,  $u_i v_i$  der Inhalt des  $i$ -ten Bandes, wobei der Kopf des  $i$ -ten Bandes auf dem ersten Symbol von  $v_i$  steht (das auch  $\square$  sein kann). Eine *Startsituation* hat die Form  $(q_0, \# w, \#, \dots, \#)$  mit  $w \in \Sigma_o^*$ .

In der Situation  $(p, u_1\#b_1v_1, \dots, u_k\#b_kv_k)$  mit  $p \in Q$ ,  $u_i, v_i \in \Gamma_o^*$ ,  $b_i \in \Gamma_o$  ist jeder Befehl anwendbar, der mit  $(p, b_1, \dots, b_k)$  beginnt, also jeder Befehl der Form  $(p, b_1, \dots, b_k, q, b'_1, \dots, b'_k, d)$  für gewisse  $q, b'_1, \dots, b'_k$  und  $d$ .

Die Menge der Situationen ist  $S = Q \times (\Gamma_o^* \{ \# \} \Gamma_o^+)^k$ . Die Schrittrelation  $\vdash_M \subseteq S \times S$  und die Mehrschrittrelation  $\vdash_M^*$  definieren wir analog. Die TM startet immer in einer Situation der Form  $(q_0, \#w, \#\square, \dots, \#\square)$  mit  $w \in \Sigma^*$ . Diese Situation nennen wir auch die zum Input  $w$  gehörige *Startsituation*  $s(w)$ . Lauf, maximalen Lauf,  $w$ -Lauf, Stopsituation, und akzeptierende Stopsituation definieren wir analog wie in Definition 2.2.

Wir nennen die TM *deterministisch* ( $k$ -DTM), wenn es keine zwei Befehle aus  $\delta$  gibt, die im Zustand und in den gelesenen Bandsymbolen übereinstimmen.

Die *Sprache* der TM ist  $L(M) = \{w \in \Sigma^* \mid s(w) \vdash_M^* s \text{ für eine akzeptierende Stopsituation } s\}$ . Die TM akzeptiert also den Input  $w$ , wenn wenigstens ein  $w$ -Lauf in einer Situation mit akzeptierendem Zustand endet.

Ist die TM  $M$  deterministisch, so ist die von  $M$  berechnete  $n$ -stellige Funktion:  $f_M^n : (\Sigma^*)^n \rightarrow \Sigma^*$  mit  $f_M^n(u_1, \dots, u_n) = w$  gdw.  $(q_0, \#u_1 \square \dots \square u_n, \#, \dots, \#) \vdash_M^* (q, \#w, \#, \dots, \#)$  für ein  $q \in F$ . Die von der TM  $M$  berechnete Funktion ist also auf dem Input definiert, wenn  $M$  in einem akzeptierenden Zustand stoppt und der Inhalt von Band 1 dann ein Wort aus  $\Sigma^*$  ist und der Kopf von Band 1 auf dem ersten Zeichen dieses Wortes steht und die übrigen Bänder leer sind. Dieses Wort ist dann der Output bzw. das Bild des Inputs unter der Funktion. In allen anderen Fällen ist die von der TM berechnete Funktion nicht definiert.

### Aufgabe 2.8

Zeigen sie, daß es zu jeder  $k$ -NTM eine äquivalente  $k$ -DTM gibt (die dieselbe Sprache erkennt bzw. dieselbe Funktion berechnet).

### Definition 2.9 (one-way, two-way)

Wir nennen eine Turingmaschine eine *two-way-Turingmaschine*, wenn das erste Band (das in der Startsituation den Input trägt) ein read-only-Band ist (auf dem also nur gelesen werden darf). Wir sagen: die Turingmaschine hat ein *two-way Inputband*. Wir nennen eine Turingmaschine eine *one-way-Turingmaschine*, wenn das erste Band ein read-only-Band ist, das außerdem nur von links nach rechts gelesen werden darf. Wir sagen dann: die Turingmaschine hat ein *two-way Inputband*. Im Gegensatz zum reinen Leseband, das wir Inputband nennen, bezeichnen wir die anderen Bändern zur Unterscheidung als *Arbeitsbänder*. Entsprechend heißt der Kopf auf dem Inputband *Inputkopf* und die Köpfe auf den Arbeitsbändern *Arbeitsköpfe*.

Manchmal interessieren uns der Zustand einer Turingmaschine  $M$ , ihre Arbeitsbandinhalte und die Stellungen ihrer Arbeitsköpfe unabhängig vom Inputband, also die Situation der  $M$  ausschließlich des Inputbandes, wenn man so will, den Großzustand den  $M$  einnimmt, wenn ihr Inputkopf ein gewisse Inputfeld liest. Dies wollen wir die *Konfiguration* von  $M$  nennen. Wenn  $M$  in der Situation  $(q, u_0 \# v_0, \dots, u_k \# v_k)$  ist (Band 0 sei das Inputband, Band 1 bis  $k$  die Arbeitsbänder), dann befindet sich  $M$  in der Konfiguration  $(q, u_1 \# v_1, \dots, u_k \# v_k)$ .

Eine Turingmaschine, die nur Arbeitsbänder hat, die also den Input zu Anfang auf dem ersten Arbeitsband stehen hat, nennen wir eine *Turingmaschine ohne Inputband*.

### Notation 2.10

Eine  $i\text{XT}^k$ -TM ( $i = 0, 1, 2; X = D, N$ ) ist eine TM ohne Inputband, wenn  $i = 0$ , hat ein 1-way Inputband, wenn  $i = 1$ , und ein 2-way Inputband, wenn  $i = 2$ , sie ist deterministisch, wenn  $X=D$ , und nichtdeterministisch, wenn  $X=N$ . Weiterhin hat sie genau  $k$  Arbeitsbänder (d.h. ein read-only-Band wird nicht mitgezählt). Mit  $i\text{XT}^k$  selbst bezeichnen wir die Klasse der Sprachen, die von einer  $i\text{XT}^k$ -TM erkannt werden. D.h. wir haben die Sprachklassen:  $0\text{DT}^k$ ,  $0\text{NT}^k$ ,  $1\text{DT}^k$ ,  $1\text{NT}^k$ ,  $2\text{DT}^k$ ,  $2\text{NT}^k$ . Weiter sei  $i\text{XT}^* = \bigcup_{k \in \mathbb{N}} i\text{XT}^k$  und  $i\text{XT}^k \langle F \rangle = \bigcup_{f \in F} i\text{XT}^k \langle f \rangle$ , wo  $F \subseteq \mathbb{N}^{\mathbb{N}}$  eine Menge von totalen, arithmetischen Funktionen sei und  $i = 0, 1, 2$  und  $X=D, N$ .

## 2.2 Aufwand von Turingmaschinen

Wir werden uns in dieser Vorlesung mit zwei Aufwandsmaßen beschäftigen, mit dem Aufwand an Zeit (*Zeitkomplexität*) und dem Aufwand an (Speicher-)Platz (*Bandkomplexität*), den Turingmaschinen treiben, um ein Problem zu lösen (eine Funktion zu berechnen).

Ist  $M$  eine Turingmaschine und  $R$  ein  $w$ -Lauf von  $M$ , so nennen wir die Länge  $t(R)$  dieses Laufs, die von  $M$  auf  $w$  verbrauchte Zeit (den Zeitaufwand des Laufes  $R$ ), und die Zahl  $s(R)$  der während  $R$  berührten Arbeitsfelder, den benötigten Platz (den Bandaufwand des Laufes  $R$ ). Ist der  $w$ -Lauf  $R$  unendlich, so ist  $t(R)$  undefiniert. Berührt  $R$  zudem unendlich viele Felder, so ist auch  $s(R)$  undefiniert. Jetzt können wir definieren, wann eine Turingmaschine innerhalb eines gewissen Aufwands arbeite. Dabei gibt es eine starke und eine schwache Vorstellung.

### Definition 2.11 (stark und schwach $f$ -beschränkt)

Sei  $f$  eine totale arithmetische Funktion. Dann nennt man  $M$  *stark  $f$ -zeitbeschränkt* bzw. *stark  $f$ -bandbeschränkt*, wenn für alle  $w$ -Läufe  $R$  mit einem  $w$  der Länge  $n$  der Aufwand  $t(R)$  bzw.  $s(R)$  definiert und kleinergleich  $f(n)$  ist.  $M$  heißt *schwach  $f$ -zeitbeschränkt* bzw. *schwach  $f$ -bandbeschränkt*, wenn es für alle  $n$  und für alle von  $M$  akzeptierten Inputs  $w$  der Länge  $n$  einen  $w$ -Lauf  $R$  gibt, dessen Aufwand  $t(R)$  bzw.  $s(R)$  kleinergleich  $f(n)$  ist.

Wir können die obigen Bedingungen für  *$f$ -zeitbeschränkt* bzw.  *$f$ -bandbeschränkt* etwas lockern und anstelle von “für alle  $n$ ” auch “für fast alle  $n$ ” setzen.

### Aufgabe 2.12

Zeigen Sie: Ist  $M$  eine Turingmaschine, die nur für fast alle  $n$  einen Zeit- oder Bandaufwand kleinergleich  $f(n)$  hat, also etwa nur für alle  $n \geq k$ , so können wir dazu eine TM  $M'$  konstruieren, die diesen Aufwand für alle  $n$  einhält.

### Notation 2.13

Der eingeschränkte Quantor  $\forall^\infty n$  bedeute “für fast alle  $n$ ” bzw. “für alle bis auf endlich viele  $n$ ”. Ist  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine arithmetische Funktion, so würde man “ $f(n)$  ist für fast alle  $n$  gerade” auch schreiben als: “ $\forall^\infty n$  ( $2$  teilt  $f(n)$ )”.

### Definition 2.14 (starker, schwacher Zeit- und Bandaufwand)

Wollen wir den realen Aufwand einer Turingmaschine bzgl. eines Inputs  $w$  messen, so unterscheiden wir wieder zwischen einer strengeren Aufwandsauffassung und einer schwächeren. Bei der strengeren Auffassung messen wir den maximalen Aufwand aller  $w$ -Läufe von  $M$ , bei der schwächeren zählen wir nur den billigsten akzeptierenden Lauf.

streng :

$$t(w) = \begin{cases} \max\{t(R) \mid R \text{ ist } w\text{-Lauf}\}, & \text{falls alle } w\text{-Läufe endlich} \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

$$s(w) = \begin{cases} \max\{s(R) \mid R \text{ ist } w\text{-Lauf}\}, & \text{falls alle } w\text{-Läufe nur endlich viele Felder berühren} \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

$$s(n) = \max\{s(w) \mid |w| = n\} \quad (s(n) \text{ undefiniert, falls } s(w) \text{ undefiniert für ein } w \in \Sigma^n)$$

schwach :

$$t(w) = \begin{cases} \min\{t(R) \mid R \text{ ist akzeptierender } w\text{-Lauf}\}, & \text{falls } w \text{ akzeptiert wird} \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

$$s(w) = \begin{cases} \min\{s(R) \mid R \text{ ist akzeptierender } w\text{-Lauf}\}, & \text{falls } w \text{ akzeptiert wird} \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

$$s(n) = \max\{s(w) \mid |w| = n \wedge w \in L(M)\}$$

Diese beiden arithmetischen (partiell berechenbaren) Funktionen  $t$  und  $s$  nennt man den *Zeitaufwand* bzw. den *Bandaufwand* von  $M$  (jeweils in der starken und in der schwachen Auffassung).

### Aufgabe 2.15

Zeigen Sie, daß die oben definierten Funktionen  $t$  und  $s$  (partiell) berechenbar sind und zwar sowohl in der starken, wie auch in der schwachen Version.

### Notation 2.16

Ist  $K$  eine Klasse von Turingmaschinen, so schreiben wir  $K\langle f \rangle$  für die Menge der von  $f$ -zeitbeschränkten  $K$ -Turingmaschinen erkannten Sprachen und  $K[f]$  für die Menge der von  $f$ -bandbeschränkten  $K$ -Turingmaschinen erkannten Sprachen. Wir schreiben "strong" oder "weak" davor, wenn wir angeben wollen, ob wir starke oder schwache Beschränkung meinen. (z.B. weak-2NT $^k\langle f \rangle$ ). Wenn strong oder weak nicht erwähnt wird, betrachten wir in der Regel den Fall weak.

O.B.d.A. können wir annehmen, daß unsere Turingmaschinen Felder nie löschen: sie drucken einfach ein Pseudo-Blank, das signalisiert: dieses Feld ist gelöscht. Dann kann man den bis dahin gebrauchten Bandaufwand an der Situation eines Laufes ablesen: es ist die Zahl der bedruckten Arbeitsfelder. Der Bandaufwand steigt während eines Laufs monoton an.

### Aufgabe 2.17

Sei  $L = \{wcv \mid w \in \{a, b\}^*\}$ . Dann ist

$$\begin{aligned} L &\in \text{1DT} < \frac{3}{2}(n+1) > \\ &\in \text{0DT} < \frac{1}{2}(n^2 + 3n) > \\ &\in \text{0DT} [n] \\ &\in \text{2DT} [\log(\frac{n+1}{2})] \end{aligned}$$

## 3 Speed-up

Es ist eine Eigenart unseres Modells, daß es eine gewisse Toleranz aufweist, was den Aufwand an Zeit oder Band angeht. Da wir die Größe des Arbeitsalphabets einer Turingmaschine nicht festlegen oder beschränken wollen, kann man den Bandaufwand und auch den Zeitaufwand reduzieren, wenn man dafür das Arbeitsalphabet vergrößert. Auch in einem konkreten Rechner kann man die Geschwindigkeit erhöhen, wenn man die Busbreite heraufsetzt oder den Speicheraufwand reduzieren, wenn man ihn statt in Bits in Bytes oder Worten mißt. Diese 'Toleranz' wollen wir untersuchen bevor wir uns an Aufwandsüberlegungen machen.

### Satz 3.1 (Bandkompression)

Zu jeder  $iXT^k[f]$ -TM  $M$  und zu jedem  $c \in \mathbb{N}_+$  existiert eine äquivalente  $iXT^k[\frac{1}{c}f]$ -TM  $M'$  ( $i = 1, 2$ ;  $X = D, N$ ).

**Beweis:** Statt einzelner Felder betrachten wir Feldblöcke der Länge  $c$ . Der Inhalt eines Felderblocks - also ein Wort der Länge  $c$  über dem Arbeitsalphabet  $\Gamma$  - kann auch durch ein einzelnes Zeichen repräsentiert werden. Ist  $\gamma = |\Gamma|$  die Größe des Arbeitsalphabets  $\Gamma$ , dann gibt es  $\gamma^c$  viele Worte dieser Länge  $c$  über  $\Gamma$ . Setzen wir für jedes dieser Worte ein eigenes Zeichen, nehmen wir also ein Alphabet  $\Delta$  der Größe  $\gamma^c$  (z.B.  $\Delta = \Gamma^c$ ) - wir können die  $c$ -Tupel über  $\Gamma$  selbst als neue Zeichen definieren, so können wir 1 Zeichen anstelle von  $c$  Zeichen setzen. Damit reduziert sich der Bandbedarf der TM um  $\frac{1}{c}$ . Die TM  $M'$  liest ein Zeichen aus  $\Delta$ , speichert es in seiner Kontrolleinheit als ein Wort der Länge  $c$  über  $\Gamma$ , simuliert die Rechnung von  $M$  auf diesem Block und schreibt den veränderten Inhalt des Blockes wieder als ein Zeichen aus  $\Delta$  auf sein Band. Diese Simulation in der KE kostet keine Zeit, sie kann in einem Schritt ausgeführt werden, weil sie intern ist und keinen Input von außen braucht (vgl. Einleitung von Abschnitt 2.1). ■

**Aufgabe 3.2**

Stellen Sie im Beweis von Satz 3.1 aus dem Turingprogramm von  $M$  das von  $M'$  her.

**Bemerkung 3.3 (Spurentechnik)**

Die Tatsache, daß ein jede endliche Menge als Alphabet fungieren kann, also auch die  $k$ -Tupel aus  $\Gamma^k$ , können wir anschaulich interpretieren. Steht das  $k$ -Tupel  $(a_1, \dots, a_k)$  auf einem Feld, so denken wir uns das Feld in  $k$  Zellen unterteilt, so daß die  $i$ -te Zelle das Zeichen  $a_i$  enthält. Faßt man nun die  $i$ -ten Zellen aller Felder zusammen zu einer  $i$ -ten Spur, so hat man das Bild, daß das Band aus  $k$  übereinanderliegenden Spuren aufgebaut ist, jede Spur wie ein eigenes Band, mit dem einzigen Unterschied, daß sich alle  $k$  Spuren einen Kopf teilen müssen. Im obigen Satz 3.1 können wir uns also vorstellen, daß die  $c$  Zeichen, die zunächst in Feldern nebeneinander stehen, in der Maschine  $M'$  auf einem einzigen Feld in  $c$  Spuren untereinander stehen.

**Satz 3.4 (Lineares Speed-up)**

Zu jeder  $0XT^k\langle f \rangle$ -TM  $M$  und zu jedem  $c \in \mathbb{N}_+$  existiert eine äquivalente  $0XT^k\langle \frac{1}{c} f \rangle$ -TM  $M'$  ( $X = D, N$ ), falls  $\lim \frac{n}{f(n)} = 0$  ( $k \geq 2$ ) bzw.  $\lim \frac{n^2}{f(n)} = 0$  ( $k = 1$ ).

**Beweis:** Die Idee besteht darin, daß auf einem komprimierten Band, wie wir es im vorigen Satz geschaffen haben, auch die Rechnung schneller gehen wird: statt einem Zeichen werden gleich  $m$  Zeichen gelesen und verarbeitet. Da dies in der KE geschieht, kostet die Verarbeitung der  $m$  Zeichen nur einen Schritt.

Dies ist aber nur ein Vorteil, wenn dieser eine Schritt auch höchstens alle  $m$  Schritte von  $M$  gemacht werden muß. Ein Problem ist es aber, wenn der Kopf von  $M$  ständig über eine Blockgrenze hin- und herwandert, sozusagen an einer Blockgrenze oszilliert. Dann würde nach jedem Schritt von  $M$  ein neuer Block gelesen und bearbeitet werden müssen. Um dies zu vermeiden hält die TM nicht einen, sondern zwei Blöcke in der KE. Zu Beginn der Rechnung von  $M$  steht der Kopf am Anfang eines Blocks und die KE von  $M'$  liest den Block mit dem Kopf und den Block links davon in ihren Speicher.

Eine Operation von  $M'$  besteht nun darin, daß die KE die Rechnung von  $M$  auf diesen beiden Blöcken in einem Schritt simuliert bis der Kopf von  $M$  sie verläßt, d.h. also mindestens  $m$  Schritte lang. Hat der Kopf die Blöcke nach rechts verlassen, so schreibt die KE den linken der beiden Blöcke aufs Band (ein Feld) und liest den gerade betretenen Block vom Band in ihren Speicher. Dies ist eine Operation, die höchstens alle  $m$  Schritte von  $M$  von  $M'$  gemacht wird.

Eine solche Operation kostet 3 Schritte. Die Simulationszeit ist dann  $\frac{3}{m}f(n)$  auf einem Band, auf  $k$  Bändern ungünstigstenfalls  $\frac{3k}{m}f(n)$ . Dazu kommt allerdings noch die Zeit, die  $M'$  braucht, den Input, der ja anfänglich nicht komprimiert vorliegt, in  $m$ -Kompression zu bringen, was bei einem Band höchstens  $\frac{n^2}{m} + m$  Schritte dauert, bei zwei Bändern aber höchstens  $2n$  (während der Input auf dem einen Band gelesen wird, wird auf dem anderen der Input in komprimierter Form gespeichert). Bei zwei Bändern ist die Gesamtzeit von  $M'$  also höchstens  $2n + \frac{3k}{m}f(n) \leq \frac{3k+1}{m}f(n)$  für alle  $n$  mit  $2n < \frac{f(n)}{m}$ , d.h. nach Voraussetzung für fast alle  $n$ , bei einem Band ist sie höchstens  $\frac{n^2}{m} + m + \frac{3k}{m}f(n) \leq \frac{3k+1}{m}f(n)$  für alle  $n$  mit  $\frac{n^2}{m} + m < \frac{f(n)}{m}$ , d.h. wieder nach Voraussetzung für fast alle  $n$ . Wählen wir  $m \geq (3k+1)c$ , so ergibt sich obige Behauptung.

Für die endlich vielen Worte bis zu einer Länge sagen wir  $n_0$ , für die möglicherweise  $n < \frac{f(n)}{m}$  nicht gilt, bauen wir einen finiten Automaten (der in real-time arbeitet) und geben ihn  $M'$  mit, wo er parallel mitarbeitet. Wenn das Inputwort kürzer ist als  $n_0$ , so entscheidet der finite Automat, ansonsten entscheidet  $M'$ . ■

**Notation 3.5**

Turingmaschinen, deren Zeitaufwand durch die Identität beschränkt wird, die also für die Rechnung nicht mehr Zeit benötigen, als zum Lesen des Inputs nötig ist, nennen wir real-time Maschinen.

Wir können auch Turingmaschinen mit linearem Zeitaufwand beschleunigen, wenn nicht auf real-time, so doch auf einen Zeitaufwand, der sich von Identität nur um einen beliebig kleinen Bruchteil der Inputlänge  $n$  unterscheidet.

**Notation 3.6**

Wir schreiben analog zu  $\mathbb{N}_+$  auch  $\mathbb{Q}_+ := \{x \in \mathbb{Q} \mid x > 0\}$

**Satz 3.7 (Lineares Speed-up in der Nähe von real time)**

Sei  $t : \mathbb{N} \rightarrow \mathbb{N}$  eine arithmetische Funktion mit  $\exists \varepsilon \in \mathbb{Q}_+ \forall n \in \mathbb{N} (t(n) \geq n + \varepsilon \cdot n)$ .

Dann gilt:  $\forall c \in \mathbb{N}_+ : \text{ODT}^k \langle c \cdot t(n) \rangle \subseteq \text{ODT}^k \langle t(n) \rangle$  (für  $k \geq 2$ ).

**Beweis:** Für jedes  $m \in \mathbb{N}$  können wir  $c \cdot t(n)$  beschleunigen auf

$$z(n) = n + \frac{n}{m} + \frac{3c}{m} \cdot t(n) :$$

das Lesen des Inputs und Schreiben der  $m$ -Kompression kostet  $n$  Schritte, das Zurücklaufen zum Anfang der Kompression kostet  $\frac{n}{m}$  Schritte, und  $\frac{3c}{m} \cdot t(n)$  Schritte kostet die beschleunigte Simulation. Wir zeigen, daß  $z(n) \leq t(n)$  bei geeigneter Wahl von  $m$ . Sei

$$m \geq 3c + \frac{3c + 1}{\varepsilon} \quad \text{oder} \quad \varepsilon \geq \frac{3c + 1}{m - 3c}.$$

Nach Voraussetzung ist

$$t(n) \geq (1 + \varepsilon) \cdot n \geq \frac{m + 1}{m - 3c} \cdot n \quad \text{oder} \quad n \leq \frac{m - 3c}{m + 1} \cdot t(n).$$

Damit ist

$$z(n) = \left(1 + \frac{1}{m}\right) \cdot n + \frac{3c}{m} \cdot t(n) \leq \frac{m + 1}{m} \cdot \frac{m - 3c}{m + 1} \cdot t(n) + \frac{3c}{m} \cdot t(n) = t(n).$$

■

**Korollar 3.8**

$\forall c \in \mathbb{N}_+ \forall \varepsilon \in \mathbb{R}^+ : \text{ODT}^k \langle c \cdot n \rangle \subseteq \text{ODT}^k \langle (1 + \varepsilon) \cdot n \rangle$  (für  $k \geq 2$ ).

**Aufgabe 3.9**

Formulieren Sie das Speed-up Theorem auch für 1-way und 2-way Turingmaschinen.

**Notation 3.10**

Ist  $F$  eine Klasse von Funktionen, so sei  $iXT^k \langle F \rangle = \bigcup \{iXT^k \langle f \rangle \mid f \in F\}$ . Beispiele für solche Klassen sind

- $\text{lin} = \{c \cdot n + d \mid c, d \in \mathbb{R}\}$  die Klasse der linearen Funktionen,
- $\text{poly} = \{a_0 + a_1 n + \dots + a_k n^k \mid a_0, \dots, a_k \in \mathbb{R}, k \in \mathbb{N}\}$  die Klasse der polynomiellen Funktionen,
- $\text{ex} = \{2^f \mid f \in \text{lin}\}$  die Klasse der exponentiellen Funktionen,
- $\text{exp} = \{2^f \mid f \in \text{poly}\}$  die Klasse der polyexponentiellen Funktionen.

Im allgemeinen kann man also den Aufwand, der zum Erkennen einer Sprache benötigt wird, nur bis auf einen konstanten Faktor genau angeben. Diese Beschleunigung durch Vergrößerung des Alphabets hat allerdings dort seine Grenzen, wo der Aufwand für die Komprimierung des Inputs nicht durch den sonstigen Aufwand subsummiert wird. Man kann sogar beweisen, daß die Beschleunigung dann nicht mehr funktioniert. Linearzeit-Maschinen nämlich können wir - wie wir gesehen haben - nahe an real-time heran beschleunigen, der nächste Satz zeigt aber, daß - wenigstens im deterministischen Fall - eine Beschleunigung auf exakt real-time nicht möglich ist, selbst wenn man noch beliebig viele Bänder dazu gibt.

**Satz 3.11**

$$1DT^2\langle lin \rangle \not\subseteq 1DT^*\langle id \rangle$$

**Beweis:** Wir geben eine Sprache an, die diese beiden Klassen trennt, die also in  $1DT^2\langle lin \rangle$  liegt, nicht aber in  $1DT^*\langle id \rangle$ . Sei

$$L := \{w_1\$ \cdots \$w_k\$w \mid k \in \mathbb{N}, \forall i \in \{1, \dots, k\} : w_i \in \{0, 1\}^* \wedge |w_i| = |w|, \exists i : w_i = w\}$$

Ein Wort aus  $L$  besteht also aus einer Folge von gleich langen Teilworten, deren letztes mit einem der vorhergehenden übereinstimmen muß. Solche Worte wollen wir hier einmal salopp  $\$$ -Worte nennen. Man überlegt sich leicht, daß  $L$  aus  $1DT^3\langle 2n \rangle$ , aus  $1DT^2\langle 3n \rangle$ , aus  $2DT^2\langle 2n \rangle$  und aus  $2DT^1\langle 3n \rangle$  ist (vergl. unten stehende Aufgabe). Wir zeigen:  $L \not\subseteq 1DT^*\langle id \rangle$ . Angenommen, die  $1DT^k\langle id \rangle$ -TM  $M$  erkennt  $L$ . Wir betrachten die Menge  $\{0, 1\}^m$  aller Worte der Länge  $m$  über  $\{0, 1\}$ .

1. Wir nennen zwei  $\$$ -Worte  $u = u_1\$ \cdots \$u_l$  und  $v = v_1\$ \cdots \$v_m$  ( $u_i, v_i \in \{0, 1\}^m$ ) trennbar durch  $z \in \{0, 1\}^m$ , gdw.  $u\$z \in L \iff v\$z \notin L$ . Zwei  $\$$ -Worte  $u$  und  $v$  sind also trennbar, wenn das Anhängen eines Wortes  $z$  aus  $\{0, 1\}^m$  das eine Wort nach  $L$ , das andere aber in das Komplement von  $L$  überführt. Offenbar kann man zwei  $\$$ -Worte  $u$  und  $v$  genau dann trennen, wenn die Menge der Teilworte aus  $\{0, 1\}^m$ , die  $u$  enthält, verschieden ist von der, die  $v$  enthält. Es gibt aber genau  $2^m$  Worte aus  $\{0, 1\}^m$ , also  $2^{2^m}$  verschiedene Teilmengen von  $\{0, 1\}^m$ . Damit gibt es auch  $2^{2^m}$  verschiedene Klassen von Worten, wenn wir die Worte, die sich nicht trennen lassen in einer Klasse versammeln. Umgekehrt gibt es  $2^{2^m}$   $\$$ -Worte, die sich alle voneinander trennen lassen.
2. Sei  $M$  eine  $1DT^k\langle id \rangle$ -TM mit  $\gamma$  Arbeitssymbolen und  $p$  Zuständen. Eine Situation, in der die Komponenten des Inputbandes nicht angegeben ist, nennen wir Konfiguration (vgl. Definition 2.9).  $M$  hat bei einem Input der Länge  $n$  höchstens  $p(\gamma^n)^k n^k$  verschiedene Konfigurationen. Die Konfigurationen  $C = (q, U_1\#V_1, \dots, U_k\#V_k)$  und  $C' = (q', U'_1\#V'_1, \dots, U'_k\#V'_k)$  von  $M$  nennen wir  $m$ -äquivalent gdw. gilt:

- $q = q'$
- $U_i, U'_i$  stimmen in den  $m$  letzten Symbolen überein ( $i = 1, \dots, k$ )
- $V_i, V'_i$  stimmen in den  $m + 1$  ersten Symbolen überein ( $i = 1, \dots, k$ )

Es gibt  $p(\gamma^{2m+1})^k$  Klassen von  $m$ -äquivalenten Konfigurationen, die sich innerhalb der  $2m + 1$  großen Blöcke um die Köpfe herum nicht unterscheiden.

Für genügend große  $m$  gilt aber  $p(\gamma^{2m+1})^k \leq 2^{k(2m+1) \cdot \log \gamma + \log p} < 2^{2^m}$ . Also muß es 2 trennbare Worte  $u, v$  geben, nach deren Lesen  $M$  in Konfigurationen  $C_u$  bzw.  $C_v$  gelangt, die  $m$ -äquivalent sind. Da  $M$  beim Lesen von  $z$  nur noch  $|z| = m$  Schritte tun kann ( $M$  ist ja eine real-time Maschine), muß  $M$  damit die Worte  $u\$z$  und  $v\$z$  gleichermaßen akzeptieren bzw. verwerfen ( $M$  kann die Konfigurationen  $C_u$  bzw.  $C_v$  innerhalb von  $m$  Schritten nicht unterscheiden).

Eine Bemerkung noch dazu, daß  $M$  seine Inputs auch wirklich bis zu Ende liest. Gäbe es ein  $\$$ -Wort  $u$ , das  $M$  akzeptiert oder verwirft, ohne es bis zu Ende zu lesen, so würde  $M$  natürlich auch alle Verlängerungen von  $u$  gleichermaßen akzeptieren oder verwerfen. Akzeptiert  $M$  nun  $u$  und ist  $z$  ein Wort aus  $\{0, 1\}^m$ , das in  $u$  nicht vorkommt, so müßte  $M$  die Verlängerung  $u\$z$  eigentlich verwerfen. Verwirft  $M$  das Wort  $u$ , und ist  $z$  ein Wort aus  $\{0, 1\}^m$ , das in  $u$  vorkommt, so müßte  $M$  die Verlängerung  $u\$z$  eigentlich akzeptieren. Also muß  $M$  alle  $\$$ -Worte, die nicht schon alle möglichen Teilworte enthalten bis zum Ende lesen. ■

**Aufgabe 3.12**

Zeigen Sie, daß die Sprache  $L$  aus vorigem Beweis aus  $1DT^3\langle 2n \rangle$ , aus  $1DT^2\langle 3n \rangle$ , aus  $2DT^2\langle 2n \rangle$  und aus  $2DT^1\langle 3n \rangle$  ist.

**Bemerkung 3.13 (Nichtdeterminismus)**

Wir wollen hier kurz betrachten, wie man sich Nichtdeterminismus veranschaulichen kann. Eine nichtdeterministische TM (NTM) hat in einer Situation mehrere Möglichkeiten zu reagieren (kann

verschiedene Befehle ausführen). Die möglichen  $w$ -Läufe einer solchen NTM bilden also einen Baum - wir nennen ihn den Berechnungsbaum von  $w$ . Der Input  $w$  wird genau dann akzeptiert, wenn einer der Pfade in diesem Baum akzeptierend ist, wenn der Baum also eine akzeptierende Blattsituation hat.

Die NTM muß sich für einen der Befehle, die sie in der jeweiligen Situation anwenden kann, entscheiden: wir sagen frei, die NTM rät den richtigen Befehl. Ob es der richtige ist, also einer, der zum Erfolg, d.h. zum Akzeptieren, führt, muß freilich überprüft werden, wir sagen: verifiziert werden. Wir können jede NTM durch eine NTM simulieren, die sich zunächst die richtige Auswahl der Befehle in der Folge der Situationen rät, aufs Band schreibt und dann nur noch deterministisch arbeitet, indem sie in der jeweiligen Situation den Befehl wählt, der in der vorher geratenen Befehlsfolge steht. Kommt sie dabei in eine akzeptierende Situation, so hat sie die geratene Befehlsfolge verifiziert.

Was heißt: die NTM rät eine Befehlsfolge aufs Band? "Eine NTM rät ein Wort aus  $\Sigma^*$  aufs Band" heißt, daß die Maschine irgendein beliebiges Wort über  $\Sigma$  aufs Band schreiben kann, daß sie also z.B. folgende Befehlsmenge enthält:  $\{(p, \square, p, a, 1) \mid a \in \Sigma\} \cup \{(p, \square, q, \square, 0)\}$ . Eine Befehlsfolge wird repräsentiert durch ein Wort. Statt also in jeder erreichten Situation einen der anwendbaren Befehle auszuführen, könnte die NTM auch im Voraus die Folge der gewählten Befehle raten und diese dann völlig deterministisch ausführen. Auf diese Weise kann jede NTM zerlegt werden in eine einfache nichtdeterministische Teilmaschine, die nur ein Wort aufs Band rät, und eine dahintergeschaltete deterministische Maschine.

Haben wir es mit nichtdeterministischen Turingmaschinen zu tun, dann können wir die Stärke des Ratens dazu einsetzen, den Zeitverlust bei der Kompression des Inputs zu vermeiden. Damit können wir dann doch von linear-time auf real-time beschleunigen, wenn wir noch zwei Bänder dazugeben.

### Satz 3.14

$$1NT^k\langle lin \rangle \subseteq 1NT^{k+2}\langle id \rangle$$

**Beweis:** Sei also  $M$  eine  $1NT^k\langle cn \rangle$ -TM für ein  $c \in \mathbb{N}_+$ . Diese simulieren wir dann wie folgt: Es gibt zwei Phasen. In Phase I arbeiten zwei Module SIM und KOP parallel (gleichzeitig). Phase II dient nur der Verifikation und enthält die parallelen Module VV und VH. Zuerst die Phase I:

#### Modul SIM:

1. Rate den Input  $w$  von  $M$  in  $m$ -Kompression auf Band 2 und laufe auf Band 2 zum Anfang zurück.
2. Simuliere  $M$  auf den Bändern 2 bis  $k+2$  beschleunigt (vgl. 3.4), wobei Band 2 die Rolle des Inputbands spielt.
3. Kopiere den geratenen, komprimierten Input von hinten von Band 2 auf Band 3 (das jetzt frei geworden ist) ein Stück weit (wieweit wird geraten).

1. kostet höchstens  $2\frac{n}{m}$  Schritte, 2. kostet höchstens  $k\frac{3cn}{m}$  Schritte und 3. kostet höchstens nochmal  $\frac{n}{m}$  Schritte, das sind insgesamt  $(3kc + 3)\frac{n}{m}$ . Wählen wir  $m > 2(3kc + 3)$ , dann sind das weniger als  $\frac{n}{2}$  Schritte.

**Modul KOP:** Simultan zu SIM kopiere den Input von Band 0 auf Band 1. Am Ende von Phase I hat KOP weniger als die Hälfte des Inputs kopiert.

In der Phase II wird jetzt der geratene, komprimierte Input mit dem wahren Input verglichen, um zu verifizieren, ob richtig geraten wurde. Das tun parallel die Module VV und VH. VH verifiziert den hinteren, noch nicht gelesenen Teil des Inputs, indem die Köpfe auf Band 0 und auf Band 1 simultan nach rechts laufen. VV verifiziert den vorderen, gelesenen und auf Band 1 kopierten Teil des Inputs, indem die Köpfe auf Band 1 und auf Band 2 simultan nach links laufen. Nach dem Lesen, nach  $n$  Schritten, ist die ganze Simulation, samt Verifikation beendet. Dies klappt freilich nur, wenn der in

Phase I von Modul SIM in 3. kopierte Teil der Kompression auf Band 3 dem Teil des Inputs entspricht, der von Modul KOP noch nicht auf Band 1 kopiert worden ist, d.h. Modul SIM muß hier richtig raten. ■

In diesem Abschnitt haben wir gesehen, daß wir den Aufwand einer Sprache bis auf einige Ausnahmen des Zeitaufwandes in der Nähe von real-time, nur bis auf einen konstanten Faktor genau angeben können. Kann die Sprache in einem Aufwand  $f$  erkannt werden, so auch mit dem Aufwand  $\frac{1}{c}f$  für jedes  $c \in \mathbb{N}_+$ . So wollen wir uns über Klassen von Funktionen unterhalten, die sich eben um einen konstanten Faktor unterscheiden.

### 3.1 Die Landau-Symbole

Für jede reelle Folge  $S = (s_i)_{i \in \mathbb{N}}$ , die beschränkt ist (es existiert eine endliche obere Schranke  $c \in \mathbb{R}$  mit  $\forall n : s_n < c$ ), muß es Stellen in  $\mathbb{R}$  geben, wo sich die Glieder von  $S$  häufen, sogenannte Häufungspunkte, also Punkte, an denen sich beliebig nah unendlich viele Glieder von  $S$  befinden. Formal sagt man:  $x$  ist ein Häufungspunkt von  $S$ , wenn es in jeder Umgebung von  $x$  ein Element von  $S$  gibt:

$$x \text{ ist Häufungspunkt der Folge } S \text{ gdw. } \forall \varepsilon > 0 \exists i \in \mathbb{N} : |x - s_i| < \varepsilon.$$

Wir können auch sagen: es gibt eine Teilfolge  $s_{i_1}, s_{i_2}, \dots$  von  $S$ , die gegen  $x$  konvergiert, also eine Indexmenge  $I \in \mathbb{N}$ , mit  $\lim_{\nu \in I} s_\nu = x$ . Ist  $x$  der kleinste bzw. größte Häufungspunkt, so schreiben wir auch  $\liminf S$  bzw.  $\limsup S$  ( $\liminf = \text{limes inferior}$ ,  $\limsup = \text{limes superior}$ ). Wenn  $\liminf S = \limsup \frac{f}{g} = x$ , so hat  $S$  also nur einen Häufungspunkt  $x$ , den wir dann auch den Limes der Folge nennen: ( $\lim_{n \rightarrow \infty} s_n = \lim S = x$ ).

Betrachten wir zwei totale arithmetische Funktionen

$$f, g : \mathbb{N} \longrightarrow \mathbb{N} \text{ und die Folge } S = \left( \frac{f(n)}{g(n)} \right)_{n \in \mathbb{N}}.$$

$S$  ist eine Folge von positiven rationalen Zahlen. Ist  $S$  beschränkt, so gibt es also Häufungspunkte. Für den kleinsten bzw. größten Häufungspunkt, schreiben wir

$$\begin{aligned} \liminf_{n \rightarrow \infty} \frac{f(n)}{g(n)} = x \text{ oder kurz } \inf \frac{f}{g} = x \text{ bzw.} \\ \limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} = x \text{ oder kurz } \sup \frac{f}{g} = x. \end{aligned}$$

(Achtung Notation: in der Mathematik bezeichnet man bei einer Menge  $M$  mit  $\inf M$  bzw.  $\sup M$  die größte untere Schranke bzw. kleinste obere Schranke von  $M$ .)

Es kann natürlich sein, daß  $S$  nicht beschränkt ist ( $\forall m \exists n : \frac{f(n)}{g(n)} > m$ ), d.h. daß  $\frac{f(n)}{g(n)}$  beliebig groß wird. Um nicht immer diese Fallunterscheidung zu machen, wollen wir  $\infty$  als (virtuellen) reellen Punkt mit aufnehmen, und schreiben  $\sup \frac{f}{g} = \infty$ , wenn  $S$  nicht beschränkt ist.

Wir wollen uns nun die Fälle im einzelnen ansehen.

1.  $\sup \frac{f}{g} = 0$ . Der oberste Häufungspunkt ist 0. Damit gibt es nur einen, d.h.  $\lim \frac{f}{g} = 0$ . Es gilt also:  $\forall \varepsilon \in \mathbb{Q}_+ \exists n_0 \forall n > n_0 : f(n) \leq \varepsilon \cdot g(n)$  oder  $\forall c \in \mathbb{N} \exists n_0 \forall n > n_0 : cf(n) \leq g(n)$ .  $f$  sinkt unter jedem Bruchteil von  $g$ , selbst wenn wir  $f$  mit einer großen Konstanten  $c$  multiplizieren, (im Bild, wenn wir die Kurve von  $f$  um einen noch so großen Winkel  $< 90^\circ$  hochdrehen) wird  $f$  immer noch, ab einer gewissen Stelle unter  $g$  sinken. Wir wollen sagen:  $f$  ist *limeskleiner als*  $g$  oder  $f$  beschleunigt langsamer als  $g$ , d.h. die Steigung von  $f$  wächst langsamer als die von  $g$  und schreiben  $f \prec g$  oder  $f \in o(g)$ .

2.  $\sup \frac{f}{g} < \infty$ . Der oberste Häufigkeitspunkt ist endlich. Da  $\sup$  der größte Häufigkeitspunkt ist, liegen höchstens noch endlich viele Elemente von  $S$  über  $\sup$ . Sei  $c$  das größte unter diesen, dann gilt  $\forall n : \frac{f(n)}{g(n)} < c$ , d.h. die Menge  $S$  ist durch  $c$  beschränkt:  $\exists c \in \mathbb{N} \forall n \in \mathbb{N} f(n) < cg(n)$ . Das bedeutet, daß  $f$  zwar über  $g$  liegen kann, aber  $f$  bleibt immer unter einem festen Vielfachen von  $g$  (beschleunigt nicht schneller als  $g$ ). Wir sagen:  $f$  *limes-kleinergleich*  $g$  und schreiben  $f \preceq g$  oder  $f \in O(g)$ .
3.  $\sup \frac{f}{g} = \infty$  (Das ist genau die Negation von  $\sup \frac{f}{g} < \infty$ , d.h.  $f \not\preceq g, f \notin O(g)$ ). Damit gilt  $\forall c \in \mathbb{N}_+ \exists n : \frac{f(n)}{g(n)} > c$ . Es gibt also eine unendliche Folge von Zahlen  $(n_i)_{i \in \mathbb{N}}$  mit  $f(n_i) > cg(n_i)$ , d.h.  $f$  übersteigt unendlich oft jedes Vielfache von  $g$ . Sind  $f'$  und  $g'$  die auf die Stellen aus  $\{n_i \mid i \in \mathbb{N}\}$  eingeschränkten Funktionen  $f$  und  $g$ , so beschleunigt  $f'$  schneller als  $g'$ . Wir sagen:  $f$  ist unendlich oft *limes-größer*  $g$  und schreiben  $f \succ^\infty g$  oder  $f \in \omega(g)$ .
4.  $0 < \sup \frac{f}{g}$  (Das ist die Negation von  $\sup \frac{f}{g} = 0$ , d.h.  $f \not\prec g, f \notin o(g)$ ). Der oberste Häufigkeitspunkt von  $S$  - nennen wir ihn  $\sup$  - kann  $\infty$  sein, aber nicht 0, also  $\exists \varepsilon > 0 \exists n : \varepsilon \cdot g(n) < f(n)$ .  $f$  kann unter  $g$  liegen, aber  $f$  muß unendlich oft über einem festen Bruchteil von  $g$  liegen. Betrachtet man wieder nur diese unendlich vielen Stellen, so kann man sagen  $f$  beschleunigt mindestens so schnell wie  $g$ . Wir sagen:  $f$  ist *limes-größergleich*  $g$  und schreiben  $f \succeq g$  oder  $f \in \Omega(g)$ .  
Man beachte: Wenn  $\inf \frac{f}{g} = 0, > 0, < \infty, = \infty$ , so ist  $\sup \frac{g}{f} = \infty, < \infty, > 0, = 0$ .
5.  $0 < \sup \frac{f}{g} < \infty$  (Das ist eine Verknüpfung von 2. und 4., d.h.  $f \preceq g$ , aber  $f \not\prec g$ ).  $f$  bleibt also immer unter einem Vielfachen von  $g$ , aber  $f$  kann nicht langsamer beschleunigen als  $g$ . Allerdings könnte immer noch der kleinste Häufigkeitspunkt 0 sein ( $\inf \frac{f}{g} = 0$ ), d.h. - beschränkt auf unendlich viele Stellen - könnte  $f$  langsamer beschleunigen als  $g$  ( $f \not\succeq g$ ). Wir schreiben  $f \in \Theta(g)$ .
6.  $0 < \inf \frac{f}{g} \leq \sup \frac{f}{g} < \infty$ . Dann gilt  $f \preceq g$  und  $g \preceq f$ . ( $f \in O(g), g \in O(f)$ ).  $f$  kann ein Vielfaches von  $g$  nicht übersteigen, aber auch  $g$  kann ein Vielfaches von  $f$  nicht übersteigen. Also ist  $\frac{1}{c}g(n) < f(n) < cg(n)$  für ein  $c \in \mathbb{N}$  und alle  $n \in \mathbb{N}$ . Ebenso ist  $\frac{1}{c}f(n) < g(n) < cf(n)$  für ein  $c \in \mathbb{N}$  und alle  $n \in \mathbb{N}$ .  $f$  und  $g$  beschleunigen also gleich schnell und wir sagen:  $f$  *limes-gleich*  $g$  (i.Z.  $f \asymp g$ ).

Wir wollen noch einmal zusammenfassen:

- |    |                        |        |                                 |  |
|----|------------------------|--------|---------------------------------|--|
| 1. | $f \in o(g)$           | $\iff$ | $0 = \sup \frac{f}{g}$          | $f \prec g$                            |
| 2. | $f \in O(g)$           | $\iff$ | $\sup \frac{f}{g} < \infty$     | $f \preceq g$                          |
| 3. | $f \in \omega(g)$      | $\iff$ | $\sup \frac{f}{g} = \infty$     | $f \succ^\infty g \quad f \notin O(g)$ |
| 4. | $f \in \Omega(g)$      | $\iff$ | $0 < \sup \frac{f}{g}$          | $f \succeq g \quad f \notin o(g)$      |
| 5. | $f \in \Theta(g)$      | $\iff$ | $0 < \sup \frac{f}{g} < \infty$ | $f \in O(g) \wedge f \in \Omega(g)$    |
| 6. | $0 < \inf \frac{f}{g}$ | $\leq$ | $\sup \frac{f}{g} < \infty$     | $f \asymp g$                           |
| 7. | $0 = \inf \frac{f}{g}$ |        |                                 | $f \succ g$                            |
| 8. |                        |        | $\inf \frac{f}{g} < \infty$     | $f \preceq g$                          |

**Bemerkung 3.15**

Die Bedeutung von  $\omega$  und  $\Omega$  ist nicht ganz einheitlich. D.E.Knuth hat 1976 vorgeschlagen, diese Symbole als Umkehrung von  $o$  und  $O$  zu gebrauchen, d.h.  $f \in \Omega(g) \iff g \in O(f)$  und  $f \in \omega(g) \iff g \in o(f)$ . Die Notation  $\Omega(g)$  wird in der Regel für untere Schranken gebraucht:  $g$  ist eine untere

Schranke für den Aufwand einer Klasse von Programmen, der Aufwand ist aus  $\Omega(g)$ . Das ist aber auch schon der Fall, wenn der Aufwand unendlich oft  $\varepsilon g$  erreicht für ein  $\varepsilon > 0$ , wenn der Aufwand also nicht aus  $o(g)$  ist. Darum verwenden sehr viele Autoren - und auch wir - dieses Symbol eben genau in dieser Bedeutung:  $f \in \Omega(g) \iff f \notin o(g)$  und  $f \in \omega(g) \iff f \notin O(g)$ . In beiden Nomenklaturen wird aber  $\Theta$  in der gleichen Weise aus  $O$  und  $\Omega$  definiert:  $f \in \Theta(g) \iff f \in O(g) \wedge f \in \Omega(g)$  (für das  $\Theta$  von Knuth benützen wir hier das Zeichen  $\asymp$ ). In beiden Auffassungen gilt auch  $\omega(g) \subseteq \Omega(g)$ , sowie ja  $o(g) \subseteq O(g)$ .

### Aufgabe 3.16

1. Gegeben seien folgende Funktionen:  $\log_2 \sqrt[9]{n}$ ,  $\log_9 n$ ,  $\sqrt[9]{n}$ ,  $\text{id}$ ,  $2^n$ ,  $2^{\sqrt[9]{n}}$ ,  $n^{2^{\log_9 n}}$ ,  $n^{\log n}$ . Zeigen Sie für jedes Paar  $(f, g)$  dieser Funktionen, in welcher Beziehung  $f$  und  $g$  zueinander stehen ( $f \in o(g)$ ,  $g \in o(f)$ ,  $f \in O(g)$  oder  $g \in O(f)$ ).
2. Sei  $kN(n)$  der kleinste Nichtteiler von  $n$  und  $f(n) = \begin{cases} \log(n^2), & \text{falls } n \text{ gerade} \\ \log \log n, & \text{sonst.} \end{cases}$   
Zeigen Sie, daß  $f \in \Theta(\log)$  und  $f \in \omega(kN)$ .

## 4 Tape-Reduction

Hier geht es darum, ob ein Turingmaschine mit mehreren Bändern schneller arbeitet als eine mit weniger Bändern. Es gibt die beiden Aspekte, den algorithmischen Aspekt und den Zählaspekt (obere und untere Schranken). Wie gewöhnlich ist es leichter, zu beweisen, daß etwas geht (wir schreiben einen Algorithmus), als daß es nicht geht (wir zählen die Ressourcen des Modells.)

Wir befassen uns zunächst mit dem algorithmischen Aspekt. Die erste Feststellung ist eine einfache: Wenn wir den Zeitaufwand quadrieren, so können wir Turingmaschinen mit beliebig vielen Bändern durch Turingmaschinen mit einem einzigen Band simulieren.

### Satz 4.1 (k-Band auf 1-Band)

$$iXT^k\langle t \rangle \subseteq iXT^1\langle t^2 \rangle \quad (i = 0, 1, 2; X = D, N; k \geq 2)$$

**Beweis:** Sei  $M$  eine  $iXT^k\langle t \rangle$ -TM. Wir konstruieren eine äquivalente TM  $M'$  mit nur einem Arbeitsband. Für jedes Arbeitsband von  $M$  schaffen wir eine Spur auf dem Arbeitsband von  $M'$  und zusätzlich eine Spur für eine Marke, die uns das Feld 0 signalisiert (Wir stellen uns die Felder nummeriert vor, wobei das Feld, auf dem der Kopf in der Startsituation steht, die Nummer 0 bekommt). Das Feld 0 enthält in jeder Zelle (Spur) das geprüfte, aktuelle Arbeitssymbol des entsprechenden Bandes von  $M$ . Den Zustand von  $M$  merkt sich  $M'$  in der Kontrolleinheit. Damit hat  $M'$  alle Information, über die  $M$  verfügt.

Simuliert  $M'$  nun einen Schritt von  $M$ , so entnimmt  $M'$  dem Feld 0 die aktuellen Arbeitssymbole und führt den dazugehörigen Befehl von  $M$  aus, indem es erst in Feld 0 die neuen Symbole druckt und dann den gesamten Inhalt einer Spur ein Feld nach links (rechts) verschiebt, wenn der Kopf des zugehörigen Bandes von  $M$  nach rechts (links) rücken würde. Um das zu tun, läuft  $M'$  von Feld 0 aus an die rechte Peripherie des Bandes (an das rechte Ende des Bandinhaltes), von dort aus an die linke Peripherie, wobei  $M'$  alle Spureinhalte verschiebt, die nach links verschoben werden müssen, dann wieder an die rechte Peripherie, wobei es die Spureinhalte verschiebt, die nach rechts verschoben werden müssen, und dann wieder auf Feld 0. Schließlich merkt sich  $M'$  den Folgezustand und beginnt mit der Simulation des nächsten Schrittes. Um die Peripherie zu erkennen, kann  $M'$  zwei zusätzliche Endmarken (linkes Ende, rechtes Ende) mitführen oder aber statt Blank ( $\square$ ) immer nur ein Pseudoblank ( $\boxtimes$ ) drucken.  $M'$  steht dann an der Peripherie, wenn  $M'$  eine Endmarke bzw. ein Blank ( $\square$ ) liest.)

Damit macht  $M'$  zur Simulation eines Schrittes von  $M$  höchstens  $6t$  Schritte, da der Inhalt der Bänder von  $M$  nicht länger als  $t$  werden kann und somit der Inhalt des Bandes von  $M'$  nicht länger als  $2t$ . Der Zeitaufwand von  $M'$  ist also höchstens  $6t^2 \in O(t^2)$ .

Bei  $ODT^k \langle t \rangle$ -TMen steht auf dem ersten Arbeitsband von  $M$  ein Input. Ein Zeichen des Inputs muß also als ein  $k$ -Tupel interpretiert werden, dessen erste Komponente dem Inputzeichen entspricht. ■

**Satz 4.2 (k-Band auf 2-Band, determ.)**

$$iDT^k \langle t \rangle \subseteq iDT^2 \langle t \cdot \log t \rangle \quad (i = 1, 2)$$

**Beweis:** Sei  $M$  eine  $iDT^k \langle t \rangle$ -TM. Wir konstruieren eine äquivalente  $iDT^2$ -TM  $M'$ . Für jedes der  $k$  Bänder von  $M$  schaffen wir 2 Spuren auf dem ersten Band von  $M'$ . Betrachten wir also ein Band von  $M$ . Rückt der Kopf auf diesem Band, so verschieben wir den Inhalt der Spuren so, daß das aktuelle Arbeitssymbol in Feld 0 zu stehen kommt.

Verschieben wir bei jedem simulierten Schritt von  $M$  den gesamten Bandinhalt, so ist der Simulationsaufwand  $O(t^2)$ . Vielleicht haben wir mehr Erfolg, wenn wir statt das gesamte Band zu verschieben nur ein jeweils benötigtes Stück des Inhalts bewegen. Das bedeutet, daß wir eine zweite Spur benötigen, in die der bewegte Inhalt geschrieben werden kann, da die der nicht bewegte Teil des Bandes ja schon beschrieben sein kann.

Würden wir immer nur den Inhalt des Feldes nach Feld 0 bewegen, der gerade benötigt (von  $M$  gelesen) wird, so würden wir im schlimmsten Fall, wenn das der Arbeitskopf von  $M$  immer nach rechts rückt, wir also die Inhalte immer nach links verschöben, der Reihe nach die Inhalte von Feld 1, 2, 3, ... nach Feld 0 holen und die bisher geholten Inhalte auf der zweiten Spur immer weiter nach links schieben. Dies würde aber mindestens einen Aufwand von  $c(1+2+\dots+t) \in O(t^2)$  für ein  $c$  bedeuten. Ähnliches gälte, wenn wir den Inhalt eines Blocks einer festen Felderzahl holten.

Die Idee ist nun: wenn wir schon  $l$  Schritte nach links laufen, sollten wir nicht nur 1 Symbol holen, sondern so viele, daß sich dieser Zeitaufwand auch lohnt. Da wir schon Zeit  $l$  verbrauchen, schadet es nichts, wenn wir dafür gleich  $l$  Symbole in die Nähe von Feld 0 bringen, wenn wir dafür nicht mehr als  $cl$  Schritte brauchen für eine feste Konstante  $c$ . Leider ist der Zeitaufwand dafür bei einer 1-Band Turingmaschine aber nicht  $O(l)$ , sondern  $O(l^2)$ . Hier kommt ins Spiel, daß wir ein zweites Arbeitsband haben. Dies dient uns nur zum schnellen Verschieben von Inhalten auf Band 1: wir kopieren den zu verschiebenden Block von  $l$  Feldern auf Band 2, laufen die  $l$  Felder, um die wir verschieben wollen, weiter und kopieren dann die  $l$  Felder von Band 2 wieder auf Band 1. Das kostet uns nur  $4l$  Schritte.

Wir beschreiben die Methode genauer, aber nur für ein Band von  $M$ : Wir denken uns das erste Band von  $M'$  zerlegt in  $2k$  Spuren, also eine Doppelspur statt einer einfachen Spur für jedes Band von  $M$ . Eine Doppelspur ist eingeteilt in Blöcke  $B_{-i}, \dots, B_{-1}, B_0, B_1, \dots, B_i$ , wobei die Länge  $|B_i| = |B_{-i}| = 2^{i-1}$  ( $i > 0$ ) und  $|B_0| = 1$ . Folgende Invariante, soll nach jeder Simulationsphase eines  $M$ -Schrittes für alle  $i \in \mathbb{N}$  gelten:

- $B_i$  ist voll und  $B_{-i}$  ist leer oder  
 $B_i$  ist leer und  $B_{-i}$  ist voll oder  
 $B_i$  und  $B_{-i}$  sind beide halbvoll, d.h. nur die untere Spur ist gefüllt.
- Für  $i \in \mathbb{N}_+$  liegen die Symbole der oberen Spur von  $B_i$  auf dem originalen  $M$ -Band links von denen der unteren Spur und  
für  $i \in \mathbb{N}_-$  liegen die Symbole der oberen Spur von  $B_i$  auf dem originalen  $M$ -Band rechts von denen der unteren Spur.
- Die Symbole von  $B_i$  liegen im Originalband links von denen von  $B_j$ , wenn  $i < j$  ( $i, j \in \mathbb{N}$ ).
- In  $B_0$  trägt nur die untere Zelle ein Symbol. Die obere enthält eine feste Marke, die Feld 0 signalisiert.

Wenn die Zellen der Blöcke  $B_1, \dots, B_{i-1}$  voll sind, Block  $B_i$  aber nicht, und wir nach rechts verschieben

müssen, so führen wir eine  $B_i$ -Operation aus (die spiegelbildliche Aufgabe, nach links zu verschieben, wenn die Blöcke  $B_{-i+1}, \dots, B_1$  voll sind, löst eine analoge  $B_{-i}$ -Operation aus):

1. Schreibe den Inhalt von  $B_0, B_1, \dots, B_{i-1}$  in die unteren Spuren von  $B_1, \dots, B_{i-1}$  und in die untere bzw. obere Spur von  $B_i$ , je nachdem ob  $B_i$  leer oder halbvoll ist (dann sind die Blöcke  $B_1, \dots, B_{i-1}$  halbvoll und  $B_i$  ist halbvoll oder voll).
2. Schreibe die obere (untere) Spur von  $B_{-i}$  (je nachdem ob  $B_{-i}$  voll oder halbvoll ist) in die unteren Spuren von  $B_{-i+1}, \dots, B_0$  (dann sind die Blöcke  $B_{-i+1}, \dots, B_{-1}$  halbvoll, der Block  $B_{-i}$  halbvoll oder leer).

Eine solche  $B_i$ -Operation kostet einen Aufwand von  $5 \cdot 2^i$  Schritten: wir schreiben den Inhalt der Blöcke  $B_0, B_1, \dots, B_{i-1}$ , d.h. beide Spuren in der richtigen Reihenfolge, auf Band 2 ab (das sind  $3 \cdot 2^{i-1}$  Schritte), laufen auf Band 1 ans Ende von Block  $B_i$  (das sind  $2^{i-1}$  Schritte), schreiben dann den Inhalt von Band 2 rückwärts in jeweils eine Spur der Blöcke  $B_1, \dots, B_i$  (das sind  $2 \cdot 2^{i-1}$  Schritte). Dann sind wir auf Feld 0, laufen  $2^{i-1}$  Schritte nach links ans Ende des Blockes  $B_{-i}$ , laufen weitere  $2^{i-1}$  Schritte nach links an den Anfang des Blockes  $B_{-i}$ , wobei wir die obere (wenn leer, die untere) Spur auf Band 2 schreiben, dann laufen wir  $2^{i-1}$  Schritte nach rechts ans Ende des Blockes  $B_{-i}$ , und dann weitere  $2^{i-1}$  Schritte nach rechts auf Feld 0, wobei wir den Inhalt von Band 2 auf die untere Spur der Blöcke  $B_{-i+1}, \dots, B_0$  kopieren. Das sind  $10 \cdot 2^{i-1} = 5 \cdot 2^i$  Schritte.

Die erste  $B_i$ -Operation kann frühestens nach  $2^i$  Schritten ausgeführt werden (solange dauert es, bis die Blöcke davor gefüllt sind). Danach sind die Blöcke davor halbvoll und es dauert mindestens  $2^{i-1}$  Schritte bis sie wieder voll sind. Um eine  $B_i$ -Operation auszuführen, muß also  $t \geq 2^i$  sein oder  $i \leq \log t$ . Damit ist der Gesamtaufwand von  $M'$  höchstens  $T(n) = \sum_{i=1}^{\log t} \frac{t}{2^{i-1}} 5 \cdot 2^i = 10t \log t$ . ■

Wie so oft kann eine nichtdeterministische Maschine Zeit sparen. Sie kann  $k$  Bänder i.a. sogar ohne Zeitverlust auf 2 Bändern simulieren, wie der folgende Satz zeigt.

**Satz 4.3 (k-Band auf 2-Band, nichtdet.)**

$$iNT^k\langle t \rangle \subseteq iNT^2\langle t \rangle \text{ für } t \geq (1 + \varepsilon) \text{ id } (i = 0, 1, 2).$$

**Beweis:** Sei  $M$  eine  $iNT^k\langle t \rangle$ -TM. Wir konstruieren eine 2-Band-TM  $M'$  wie folgt:  $M'$  angewandt auf den Input  $w$  rät sich auf Band 1 eine beliebige endliche Folge von Befehlen von  $M$  (ein Befehl pro Feld). Die Befehle haben die Form

$$(p, A_0, A_1, \dots, A_k, q, B_1, \dots, B_k, d_0, d_1, \dots, d_k),$$

wobei  $A_0$  und  $d_0$  im Falle  $i = 0$  wegfällt.  $M'$  überprüft nun, ob diese Befehlsfolge zu einem akzeptierenden  $w$ -Lauf von  $M$  führen würde. Wenn  $M$  den Input  $w$  akzeptiert, dann muß es eine solche Befehlsfolge geben. Dazu prüft  $M'$  zuerst, ob der erste Befehl mit dem Startzustand  $q_0$  beginnt und ob der letzte Befehl in einen akzeptierenden Zustand  $q_e$  überführt und ob der Folgezustand jeden Befehls gleich dem ersten Zustand des folgenden Befehls ist. Dann überprüft  $M'$  für jedes Band  $i$  einzeln ( $i = 0, \dots, k$ ), ob die Befehle an den entsprechenden Komponenten  $A_i, B_i$  und  $d_i$  zusammenpassen, indem  $M'$  die Rechnung auf Band  $i$  von  $M$  anhand der Befehlsfolge auf seinem zweiten Band simuliert. Außerdem merkt sich  $M'$  nach der  $i$ -ten Simulation das Symbol  $a_i$ , das vom Kopf nach dem letzten Befehl gelesen wird.

Hat jede Simulation geklappt, und gibt es keinen Befehl von  $M$ , der mit  $(q_e, a_0, \dots, a_k)$  beginnt, d.h. die Befehlsfolge könnte nicht weitergeführt werden, so war die geratenene Befehlsfolge korrekt. Der Zeitaufwand von  $M'$  setzt sich zusammen aus: Raten der Befehlsfolge, wobei das Überprüfen der Zustände gleichzeitig geschehen kann ( $t$  Schritte) und den  $k$  bzw.  $k + 1$  Simulationen von jeweils  $t$  Schritten. Damit ist der Aufwand aus  $O(t)$ . Da  $t \geq (1 + \varepsilon) \text{ id}$  können wir wegen des Speed-up-Theorems die Simulation auch mit  $t$  Schritten ausführen. ■

Dieser Satz gilt nur für  $t \geq (1 + \varepsilon) \text{ id}$ , was uns garantiert, daß wir den Zeitverlust um einen konstanten Faktor durch Beschleunigen wieder wettmachen können. Was ist für kleinere  $t$ ? Dafür müssen wir

etwas mehr Aufwand treiben. Wir zeigen, daß man für kleinere  $t$  nicht nur die Bandanzahl reduzieren, sondern die Zeit sogar noch auf real-time beschleunigen kann. Dies tun wir in zwei Schritten: zuerst zeigen wir  $2NT^*\langle \text{lin} \rangle \subseteq 1NT^4\langle \text{id} \rangle$  und dann  $1NT^4\langle \text{id} \rangle \subseteq 1NT^2\langle \text{id} \rangle$ .

**Satz 4.4**

$$2NT^*\langle \text{lin} \rangle \subseteq 1NT^4\langle \text{id} \rangle.$$

**Beweis:** Durch kopieren des Inputs auf ein zusätzliches Arbeitsband, machen wir aus einer  $2NT^*\langle \text{lin} \rangle$ -TM wieder eine  $1NT^*\langle \text{lin} \rangle$ -TM. Sei also  $M$  eine  $1NT^k\langle cn \rangle$ -TM für ein  $c \in \mathbb{Q}$ . Diese simulieren wir dann wie folgt: Es gibt zwei Phasen. In Phase I arbeiten zwei Module SIM und KOP parallel (gleichzeitig). Phase II dient nur der Verifikation und enthält die parallelen Module VV und VH.

**Phase I:**

**Modul SIM:** (Raten einer Befehlsfolge und eines komprimierten Inputs und Verifikation der geratenen Befehlsfolge)

1. Rate eine Folge  $B_1, \dots, B_t$  von Befehlen von  $M$  komprimiert -  $m$  Befehle pro Feld - auf Band 1, so daß  $B_1$  mit dem Startzustand beginnt,  $B_t$  mit einem akzeptierenden Zustand endet und der Folgezustand von  $B_i$  gerade der Zustand ist, mit dem  $B_{i+1}$  beginnt ( $i = 1, \dots, t-1$ ). Dann laufe zu Feld 0 zurück.
2. Rate den Input  $w$  von  $M$  in  $m$ -Kompression auf Band 2 und laufe auf Band 2 zurück zum Feld 0.
3. Separat für Band 2 (geratener Input) und für jedes der  $k$  Arbeitsbänder von  $M$  führe die Befehlsfolge  $B_1, \dots, B_t$  beschleunigt aus (vgl. 3.4). Dazu benütze Band 3. Im Anschluß löscht Kopf 3 Band 3 und läuft zum Anfang zurück. Außerdem merke bei jedem Band, was nach der Ausführung des letzten Befehls  $B_t$  gelesen wurde. Wenn das geklappt hat, d.h. wenn der Kopf immer gelesen hat, was im nächsten Befehl erwartet wurde, und wenn kein Befehl von  $M$  mit dem Endzustand von  $B_t$  und den gemerkten Zeichen anfängt, dann ist die Befehlsfolge eine korrekte Befehlsfolge eines Laufes von  $M$  angewandt auf den geratenen Input.
4. Kopiere den geratenen Input von hinten von Band 2 auf Band 1 (das jetzt frei geworden ist) ein Stück weit (wieweit wird geraten).

Aufwand: Stufe 1 kostet  $2c\frac{n}{m}$  Schritte, Stufe 2 kostet  $2\frac{n}{m}$  Schritte, Stufe 3 kostet  $3k\frac{3cn}{m}$  Schritte und Stufe 4 nochmal  $\frac{n}{m}$  Schritte, das sind insgesamt  $(9kc + 2c + 3)\frac{n}{m}$ . Wählen wir  $m > 2(9kc + 2c + 3)$ , dann sind das weniger als  $\frac{n}{2}$  Schritte.

**Modul KOP:** (Unkomprimiertes Kopieren des Inputs)

Parallel zu SIM kopiere den Input von Band 0 auf Band 4. Am Ende von Phase I hat SIM weniger als die Hälfte des Inputs kopiert.

**Phase II** (Verifikation des geratenen Inputs):

In der Phase II wird jetzt der geratene, komprimierte Input auf Band 2 und Band 1 mit dem wahren Input verglichen, um zu verifizieren, ob richtig geraten wurde. Das tun parallel die Module VV und VH. VH verifiziert den hinteren, noch nicht gelesenen Teil des Inputs, indem die Köpfe auf Band 0 und auf Band 1 simultan nach rechts laufen. VV verifiziert den vorderen, gelesenen und auf Band 4 kopierten Teil des Inputs, indem die Köpfe auf Band 4 und auf Band 2 simultan nach links laufen. Da die Phase I weniger als  $\frac{n}{2}$  Schritte gebraucht hat, ist das Modul VV vor dem Modul VH fertig, d.h. der Inputkopf braucht nie zu warten. Damit ist die ganze Simulation, samt Verifikation nach  $n$  Schritten beendet. ■

**Satz 4.5**

$$1NT^4\langle \text{id} \rangle \subseteq 1NT^2\langle \text{id} \rangle.$$

**Beweis:** Sei  $M$  eine  $1NT^4\langle \text{id} \rangle$ -TM, und  $w$  der Input. Wir beschreiben eine  $1NT^2\langle \text{id} \rangle$ -TM  $M'$ ,

die dieselbe Sprache erkennt. O.B.d.A. können wir annehmen, daß  $M$  nur in Stopzuständen hält, d.h. in Zuständen, in denen kein Befehl mehr ausführbar ist [wir können eine TM leicht so umbauen: wir machen die Endzustandmenge zunächst leer und definieren den neuen akzeptierenden Zustand  $a$  und den neuen verwerfenden Zustand  $q_v$ . Wenn kein Befehl mit dem Tupel  $(p, a, A, B, C, D)$  ( $a \in \Sigma$ ;  $A, B, C, D \in \Gamma$ ) beginnt, so definieren wir einen Befehl  $(p, a, A, B, C, D, s, A, B, C, D, 0, 0, 0, 0, 0)$ , wobei  $s = q_a$ , wenn  $p$  akzeptierend war, und  $s = q_v$  sonst]. Das Prinzip ist das gleiche wie im vorigen Satz 4.4. Wir haben wieder 2 Phasen, in denen parallele Module laufen. In Phase I laufen die Module RAT und SIM parallel zum Modul KOP, in Phase II laufen die beiden Module VV und VH parallel zueinander.  $M'$  bewegt seinen Kopf auf Band 1 immer in Sweeps: in einem Sweep läuft der Kopf von Feld 0 bis Feld  $n/24$  und wieder zurück und dies ohne anzuhalten, d.h. in jedem Schritt läuft er auch ein Feld weiter.

### Phase I:

**Modul RAT:** (Raten einer Befehlsfolge und eines komprimierten Inputs)

Band 1 hat 15 Spuren. Die Spuren 4 - 15 dienen zum Kopieren des Inputs. Auf Spur 1 bis 3 rät  $M'$  ein 3-spuriges Wort und läuft dann zurück, d.h. der Kopf macht einen Sweep. Wenn das Wort richtig geraten ist, hat es folgende Form:

1. In Spur 1 steht eine komprimierte Folge  $B_1, \dots, B_n$  von Befehlen von  $M$ , 24 Befehle pro Feld, so daß  $B_1$  mit dem Startzustand beginnt,  $B_n$  mit einem akzeptierenden Zustand endet und der Folgezustand von  $B_i$  gerade der Zustand ist, mit dem  $B_{i+1}$  beginnt ( $i = 1, \dots, n - 1$ ),
2. in Spur 2 steht die erste Hälfte des Inputwortes  $w$  komprimiert (24 Symbole pro Feld).
3. in Spur 3 steht die gespiegelte zweite Hälfte des Inputwortes  $w$  komprimiert (24 Symbole pro Feld).

Die in 2. und 3. geratene Inputkompression dient der späteren Verifikation des in der Befehlsfolge mitgeratenen Inputs mit dem Input auf Band 0 und wird nicht bei der späteren Simulation benötigt. Der in der Befehlsfolge enthaltene Input soll natürlich mit der Inputkompression in Spur 2 und 3 übereinstimmen. Darum verifiziert  $M'$  beim Raten gleich, daß die erste Hälfte des geratenen Inputs auf Spur 2 übereinstimmt mit dem in der ersten Hälfte der Befehlsfolge niedergelegten Input. Gleichzeitig schreibt  $M'$  den Inhalt der Spur 3 auf Band 1 auch auf Band 2. Beim Raten der zweiten Hälfte der Befehlsfolge vergleicht dann der zurücklaufende Kopf auf Band 2, ob auch die zweite Hälfte der geratenen Inputkompression mit dem in der zweiten Hälfte der Befehlsfolge niedergelegten Input übereinstimmt. Anschließend läuft der Kopf auf Band 1 zurück. Nun ist gesichert, daß der auf Spur 2 und 3 geratene und in der Befehlsfolge verzeichneten Input der gleiche ist. (Der Kopf 2 lösche dabei sein Band und macht es frei für die weitere Arbeit.)

**Modul SIM:** (Verifikation der geratenen Befehlsfolge)

In 4 weiteren Sweeps führt  $M'$  nun separat für jedes der 4 Arbeitsbänder von  $M$  die Befehlsfolge  $B_1, \dots, B_n$  24-fach beschleunigt auf Band 2 aus (vgl. 3.4), d.h. auf Band 2 arbeitet der Modul mit einer 72-fachen Kompression (der Input steht bereits in der Befehlsfolge und braucht hier nicht vom Band gelesen zu werden). Wenn das geklappt hat, ist die Befehlsfolge die korrekte Befehlsfolge eines Laufes von  $M$  angewandt auf den geratenen Input. Anschließend machen beide Köpfe einen 6. Sweep, und kopieren dabei den geratenen, komprimierten Input von Spur 2 und 3 auf Band 1 in die Spuren 1 und 2 auf Band 2, wobei der Kopf von Band 2 am Ende der Kopie stehen bleibt. (Kopf 1 aber vollzieht einen vollen Sweep, damit die Zeiten stimmen.)

**Modul KOP:** (Unkomprimiertes Kopieren des Inputs)

Parallel zu der Arbeit in den Modulen RAT und SIM kopiert der Kopf auf Band 1 während seiner 6 Sweeps in die 12 Spuren 4 bis 15 den Input von Band 0, der in dieser Zeit von Kopf 0 gelesen wird. Am Ende steht der gelesene Input also hin- und hergefaltet in den Spuren 4 bis 15: in Spur 1 vorwärts, in Spur 2 rückwärts, in Spur 3 vorwärts, usw.

In der Phase I werden also  $12 \frac{n}{24} = \frac{n}{2}$  Schritte gemacht und genau der halbe Input von Band 0 kopiert.

**Phase II:** (Verifikation des geratenen Inputs)

Es bleibt noch zu überprüfen, daß der geratenene, komprimierte Input mit dem wahren Input auf Band 0 übereinstimmt. Am Ende der Phase I stehen Kopf 1 auf Feld 0 und Kopf 2 hinter seinem Bandinhalt. Während der Kopf auf Band 0 die zweite Hälfte des Inputs liest, vergleicht der Kopf 2 diesen mit dem Inhalt seiner 2. Spur (wo die 2. Hälfte des geratenen Inputs in gespiegelter Form steht). Gleichzeitig aber vergleicht dieser Kopf den Inhalt seiner 1. Spur (wo die 1. Hälfte des geratenen Inputs steht) mit dem, was Kopf 1 liest, der jetzt den von Band 0 auf Spur 4 bis 15 kopierten Input rückwärts in weiteren 6 Sweeps wieder abliest. Sind diese Vergleiche alle erfolgreich, so ist der geratene Input richtig gewesen. ■

#### Bemerkung 4.6

Man kann sich überlegen, daß man auf diese Weise sogar  $1NT\langle \text{lin} \rangle \subseteq 1NSP\langle \text{id} \rangle$  zeigen kann, wobei P ein Pushdownband (Keller) und S ein Stackband symbolisiert (Man kann also eine linear-zeit Mehrband-Maschine durch eine mit einem Stack und einem Keller ohne Zeitverlust simulieren.)

### 4.1 Untere Schranken: Crossing Sequence

Als nächstes fragen wir uns, ob die Simulation von  $k$  Bändern auf einem Band mit einem quadratischen Mehraufwand wirklich der Weisheit letzter Schluß ist. Kann man dasselbe vielleicht mit einem geringeren Zeitaufwand. Wir fragen also nach der unteren Schranke des Zeitaufwandes einer solchen Simulation. Ist  $t$  der Zeitaufwand der  $k$ -Band TM, so ist  $t^2$  eine obere Schranke und  $t$  eine triviale untere Schranke. Wieweit können wir diese triviale untere Schranke heraufsetzen? Wir wollen zeigen: soll die Simulation von einer TM mit nur einem Arbeitsband ohne Inputband ( $0NT^1$ -TM) geleistet werden, so ist die untere Schranke  $t^2$ , d.h. es geht mit quadratischem Mehraufwand, aber nicht mit weniger. Unser Verfahren ist zwar einfach, aber optimal.

Etwas genauer: wir nennen die Funktion  $s_u$  eine untere Schranke der Mehrband-Einband-Simulation, wenn es eine Funktion  $t$  und eine Sprache  $L$  gibt, so daß  $L \in 1DT^k\langle t \rangle$ , aber  $L \notin 0DT^1\langle o(s_u \circ t) \rangle$ . Weiter nennen wir die Funktion  $s_o$  eine obere Schranke, wenn für alle  $t$  gilt:  $1DT^k\langle t \rangle \subseteq 0DT^1\langle s_o \circ t \rangle$ . In dieser Nomenklatur ist  $id$  eine triviale untere Schranke und  $quad$  eine obere Schranke.

Etwas schwieriger ist es (und wir werden es hier auch nicht behandeln), wenn wir zulassen, daß die simulierende Einband-TM ein 1-way Inputband hat, d.h. also eigentlich 2 Bänder, wenn auch eines davon extrem eingeschränkt ist. Auch hier gelingt es im deterministischen Fall ( $1DT^1$ ) immer noch - allerdings komplizierter - zu zeigen, daß die untere Schranke  $t^2$  ist. Im nichtdeterministischen Falle ( $1NT^1$ ) kann man die untere Schranke sehr nahe an  $t^2$  heranzuführen, aber bisher ist es noch nicht gelungen, sie exakt auf  $t^2$  zu setzen. Die untere Schranke, die man erreicht hat, ist  $\frac{t^2}{\log^s n}$  für beliebiges, aber festes  $s$  ( $\log^s n = \log \circ \dots \circ \log n$ ,  $s$ -mal iteriert).

Für den Beweis genügen unsere bisherigen Mittel nicht. So holen wir ein bißchen aus. Wenn wir zeigen wollen, daß ein Maschinentyp eine Sprache nicht erkennen kann, brauchen wir ein Zählargument. In Theorem 3.11 haben wir die  $m$ -Äquivalenzklassen von Konfigurationen der Maschine gezählt. Nicht immer kann man so naheliegende Aspekte heranziehen. Was wollen wir zählen, wenn wir zeigen wollen, daß eine real-time Turingmaschine mit  $k$  Bändern von einer 1-Band-Turingmaschine (ohne Inputband) nicht unter einem Zeitaufwand von  $O(n^2)$  simuliert werden kann? Wir zählen Crossing Sequences.

#### Definition 4.7 (Crossing Sequence)

Wir betrachten eine (deterministische oder nichtdeterministische) 1-Band Turingmaschine  $M$  ohne Inputband, einen Lauf  $R$  von  $M$  und 2 benachbarte Felder  $k$  und  $k+1$  ( $k \in \mathbb{Z}$ ) auf ihrem Band. Dann ist die Crossing Sequence  $C(R, k)$  des Laufes  $R$  an der Stelle  $k$  die Folge der Zustände, die während

des Laufes  $R$  von  $M$  jeweils eingenommen werden, nachdem der Kopf diese Grenze zwischen Feld  $k$  und Feld  $k + 1$  gerade überquert hat,

Wir können uns eine Crossing Sequence als den Niederschlag der über diese Grenze hinweg transportierten Information vorstellen: kein Zeichen kann über diese Grenze gebracht werden, ohne daß es am Zustand sichtbar wird, der nach dem Überqueren dieser Grenze erreicht wird, denn nur über den Zustand kann die Maschine sich dieses Zeichen merken, um es hinter der Grenze wieder abzulegen.

Betrachten wir Mehrband-Turingmaschinen, so könnten wir dieselbe Feststellung machen, wenn wir in der Crossing Sequence nicht nur den Zustand verzeichneten, der nach Überqueren der Grenze erreicht wird, sondern die gesamte Information der Maschine, die außerhalb des betrachteten Bandes liegt, wenn wir so wollen den Großzustand der Maschine, der aus dem eigentlichen Zustand und den Inhalten und Kopfstellungen der anderen Bänder besteht. Auch solche verallgemeinerten Crossing Sequences werden wir betrachten, aber hier begnügen wir uns zunächst mit dieser einfacheren Form.

#### Notation 4.8

Liegt der Lauf  $R$  fest, so schreiben wir kürzer  $C(k)$ . Wenn die Grenze im Bereich des Inputs  $uv$  zwischen den Teilinputs  $u$  und  $v$  verläuft, so schreiben wir statt  $C(R, |u|)$  auch  $C_R(u, v)$ .

Um die Nützlichkeit dieser Definition zu verstehen, machen wir uns ein paar einfache Fakten klar, die wir hier ohne Beweis formulieren.

#### Lemma 4.9

Sei  $M$  eine 1-Band-Turingmaschine ohne Inputband,  $R$  ein  $uv$ -Lauf von  $M$ ,  $R'$  ein  $u'v'$ -Lauf von  $M$  und  $C_R(u, v) = C_{R'}(u', v')$ . Dann gibt es einen  $uv'$ -Lauf  $R_1$  und einen  $u'v$ -Lauf  $R_2$  mit

1. ist  $|C_R(u, v)|$  ungerade, so gilt:  
( $R_1$  akzeptierend  $\iff R'$  akzeptierend) und ( $R_2$  akzeptierend  $\iff R$  akzeptierend),
2. ist  $|C_R(u, v)|$  gerade, so gilt:  
( $R_1$  akzeptierend  $\iff R$  akzeptierend) und ( $R_2$  akzeptierend  $\iff R'$  akzeptierend),
3. sind  $R$  und  $R'$  akzeptierend, so sind auch  $R_1$  und  $R_2$  akzeptierend.

#### Korollar 4.10

Sei  $M$  eine 1-Band-Turingmaschine ohne Inputband,  $uv$  und  $u'v'$  aus  $L(M)$  und  $R$  bzw.  $R'$  akzeptierender Lauf für  $uv$  bzw.  $u'v'$  mit  $C_R(u, v) = C_{R'}(u', v')$ . Dann sind auch  $uv'$  und  $u'v$  aus  $L(M)$ .

#### Korollar 4.11

Sei  $M$  eine 1-Band-Turingmaschine ohne Inputband,  $R$  ein akzeptierender Lauf von  $M$  für den Input  $uvw$  und  $C(R, |u|) = C(R, |uv|)$ . Dann gibt es auch für  $uw$  einen akzeptierenden Lauf von  $M$ .

**Beweis:** Man überlege sich, daß  $M$  nicht auf dem von  $v$  besetzten Bandteil stoppen kann. ■

Jetzt sind wir bereit den oben angekündigten Satz zu zeigen: Eine 1-Band-Turingmaschine ohne Inputband kann eine real-time  $k$ -Band Turingmaschine nicht mit weniger als  $n^2$  Zeitaufwand simulieren, auch wenn diese noch ein zusätzliches 1-way Inputband hat und auch, wenn wir zulassen, daß die simulierende Turingmaschine nichtdeterministisch ist.

#### Satz 4.12 (quadratische untere Schranke)

$1DT^k\langle n \rangle \not\subseteq ONT^1\langle o(n^2) \rangle$ , d.h.  $1DT^k\langle n \rangle \not\subseteq ONT^1\langle t \rangle$ , wenn  $t \in o(n^2)$  ( $k > 0$ ).

**Beweis:** Wir betrachten die Sprache  $L = \{u\#^m u \mid u \in \Sigma^m, m \in \mathbb{N}\}$ .  $L$  ist aus  $1DT^1\langle \text{id} \rangle$  und aus  $0DT^2\langle \text{id} \rangle$ . Wir zeigen:  $L$  ist nicht aus  $0NT^1\langle o(n^2) \rangle$ .

Sei also  $M$  die  $0NT^1$ -Maschine, die  $L$  erkennt und  $Q$  ihre Zustandsmenge. Wir betrachten einmal die  $2^m$  Worte aus  $L$  der Länge  $n = 3m$  und in jedem dieser Worte  $w$  eine kürzeste Crossing Sequence

$C_w$  im mittleren #-Teil. Nach den obigen Überlegungen müssen alle diese  $2^m$  Crossing Sequences  $C_w$  ( $w \in L$ ,  $|w| = 3m$ ) verschieden sein. Wir fragen uns, wie lang also die längste dieser kürzesten Crossing Sequences sein muß, denn diese Länge gibt uns eine untere Schranke für die Rechenzeit der Maschine auf dem zugehörigen Wort.

Eine untere Schranke für die Länge der längsten dieser  $2^m$  verschiedenen Crossing Sequences bekommen wir, wenn wir uns die  $2^m$  kürzesten Crossing Sequences ansehen, d.h. die  $2^m$  kürzesten Worte über dem Alphabet  $Q$ . Sei  $|Q| = q$ . Es gibt  $q^0$  Crossing Sequences der Länge 0,  $q^i$  der Länge  $i$  und  $\sum_{i=0}^l q^i = \frac{q^{l+1}-1}{q-1} \leq q^{l+1}$  bis zur Länge  $l$ . Da wir  $2^m$  verschiedene Crossing Sequences brauchen, müssen wir  $l$  so groß wählen, daß die Zahl der Crossing Sequences bis zur Länge  $l$  größergleich  $2^m$  ist, also wenigstens muß  $q^{l+1} \geq 2^m$ , d.h.  $l \geq \frac{m}{\log q} - 1$  sein.

Wir haben festgestellt, daß wenigstens für eines der betrachteten  $2^m$  Worte  $w$  die kürzeste Crossing Sequence  $C_w$  die Länge  $\frac{m}{\log q} - 1$  haben muß. Alle anderen  $m-1$  Crossing Sequences in dem mittleren #-Teil von  $w$  haben also eine Länge die größer oder gleich ist. Damit ist die Zeit, die die Turingmaschine  $M$  allein auf dem mittleren Teil von  $w$  verbringt wenigstens

$$m\left(\frac{m}{\log q} - 1\right) = \frac{n}{3}\left(\frac{n}{3\log q} - 1\right) \geq \frac{1}{c}n^2 \text{ für ein } c \in \mathbb{N}.$$

Für alle durch 3 teilbaren Wortlängen  $n$  gilt also, daß es ein Wort gibt, auf dem  $M$  wenigstens  $n^2/c$  Schritte rechnet. Für diese  $n$  also muß die Aufwandschranke  $t(n)$  oberhalb eines festen Bruchteils von  $n^2$  liegen, was bedeutet, daß  $\sup(t/n^2) > 0$  und somit  $t \notin o(n^2)$ . ■

## 5 Bandhierarchien

In den vorigen Kapiteln haben wir erfahren, daß eine Klasse von aufwandbeschränkten (in Zeit oder Band) Turingmaschinen nicht mehr Sprachen erkennt, wenn man in die Klasse noch Turingmaschinen aufnimmt, deren Aufwand ein festes Vielfaches des erlaubten Aufwand der Klasse ist (salopp gesprochen: Turingmaschinen können auch dann nicht mehr, wenn ihr Aufwand um einen festen Betrag vervielfacht wird).

Die Frage stellt sich nun, um wieviel muß man den Aufwand erhöhen, damit die Erkennungsleistung von Turingmaschinen wirklich zunimmt. Genauer: Sei  $K$  ein Modell von Turingmaschinen bzw. die Klasse von Turingmaschinen eines bestimmten Typs und  $K[f]$  die Menge der von diesem Modell, diesen Turingmaschinen, bei einer Bandschranke von  $f$  erkannten Sprachen, so fragen wir, um wieviel muß  $g$  größer sein als  $f$ , damit die Klasse  $K[g]$  echt größer ist als die Klasse  $K[f]$  ( $K[f] \subset K[g]$ ). Doch zieht diese Frage gleich weitere nach sich: was heißt „größer“, hat man eine Antwort für  $f$ , gilt sie dann auch für jede andere Schranke, gibt es besonders interessante Schranken  $f$  oder Klassen von Schranken?

### 5.1 Turingprogramm, UTM, Konstruierbarkeit

Eine wichtige Beweismethode wird sein, daß Turingmaschinen selbst Input sein können, daß Maschinen auf Maschinen - z.B. auf sich selbst - angewandt werden können. Das erlaubt uns universelle Simulation und Diagonalisierung. Diese Tatsache ist so wichtig, daß sie sogar zu einem Axiom für Nummerierungen  $\varphi$  der partiell rekursiven Funktionen  $\mathbb{P}$  gemacht wird.

Da Maschinen nur Worte bearbeiten können, müssen alle Aufgaben kodiert werden in Zeichenreihen, in Worte. Ein Problemfall muß in ein Inputwort umformuliert, kodiert werden. Sollen nun Maschinen auf Maschinen angewandt werden, so müssen auch Turingmaschinen kodiert werden als ein Wort über einem festen Alphabet. Ein solches Wort, das eine Turingmaschine kodiert, wollen wir ein

Turingprogramm oder auch ein Turingwort nennen. Eine mögliche Art Turingmaschinen zu kodieren geben wir hier an:

Sei  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  eine 1-Band-Turingmaschine (ohne Inputband) mit  $F = \{q_1, \dots, q_k\}$  und  $\delta = \{b_1, \dots, b_m\}$ . O.B.d.A. seien alle betrachteten Alphabete  $Q$ ,  $\Sigma$  und  $\Gamma$  aus  $\mathbb{N}_+$ . Wir kodieren  $M$  als binäres Wort  $w_M$  (das Turingprogramm der Maschine  $M$ ) vermöge:

$$w_M = \text{cod}(M) = 1^{q_0}01^{q_1}0 \dots 01^{q_k}00 \text{cod}(b_1)0 \dots 0 \text{cod}(b_m),$$

wobei für einen Befehl  $b = (p, A, q, B, d)$  aus  $\delta$  die Kodierung  $\text{cod}(b) = 1^p01^A01^q01^B01^{d+2}$  sei.

Auf diese Weise können wir jeder TM ein Wort über  $\{0, 1\}$  zuordnen. Es gibt allerdings noch viele binäre Worte, die nicht als ein Turingwort interpretiert werden können, weil sie nicht die richtige Form haben. Um das zu ändern, definieren wir einfach alle diese Worte als Turingworte einer festen Turingmaschine, z.B. der Turingmaschine  $\perp$ , die nur den Startzustand hat, aber keine Befehle und keine Arbeitssymbole. Damit haben wir eine totale, surjektive (Interpretations-) Funktion  $T$  von  $\{0, 1\}^*$  auf die Menge der Turingmaschinen, die jedes binäre Wort  $u$  als eine Turingmaschine  $T_u$  interpretiert, sodaß  $T_u = \perp$  oder  $\text{cod}(T_u) = u$ .

Nummerieren wir die Worte aus  $\{0, 1\}^*$  (zählen sie auf), so nummerieren wir auch die Menge  $\mathcal{T}$  der Turingmaschinen:  $(T_u)_{u \in \{0, 1\}^*}$ , wobei freilich die leere Turingmaschine unendlich viele Turingworte hat, in dieser Nummerierung also unendlich oft vorkommt. Eine Nummerierung von  $I = \{1\}\{0, 1\}^* \cup \{\varepsilon\}$  erzeugt ebenfalls eine (andere) Nummerierung von  $\mathcal{T}$ :  $(T_u)_{u \in I}$ .

Jetzt macht es Sinn, davon zu reden, daß eine TM auf eine andere TM  $M$  angewandt wird, sie wird nämlich auf den Input  $\text{cod}(M) = w_M$  angewandt. Wenn eine Turingmaschine  $U$  angewandt auf den Input  $u \square w$  ( $u \in \Sigma^*$ ,  $w \in (\Sigma^* \cdot \{\square\})^n$ ) denselben Output liefert wie  $T_u(w)$ , so nennen wir  $U$  eine universelle Turingmaschine (UTM), da sie alle Turingmaschinen zu simulieren vermag, da sie also ein einheitliches, kanonisches oder eben universelles Verfahren repräsentiert, mit dem die Rechnungen aller Turingmaschinen nachvollzogen werden können.

Wir beschreiben eine universelle Turingmaschine  $U$  und deren Arbeitsweise: Sei  $u$  ein Turingprogramm und  $w \in \Sigma^*$ . Bei Input  $u \square w$  simuliert  $U$  die durch  $u$  beschriebene TM  $T_u$  angewandt auf Input  $w$  wie folgt:

1.  $U$  sucht in  $u$  den größten 1-String (längstes Symbol) (die Länge sei  $c_u$ ).
2.  $U$  kopiert den Input  $w$  in unärer Form auf Band 2, wobei jetzt jedes Symbol in einen Block von  $c_u$  Feldern geschrieben wird. Ist also  $w = a_1 \dots a_n$ , so schreibt  $U$  auf Band 2 das Wort  $1^{a_1}0^{c_u - a_1} \dots 1^{a_n}0^{c_u - a_n}$ .
3.  $U$  schreibt den Startzustand in der Form  $1^{q_0}0^{c_u - q_0}$  auf Band 3.
4.  $U$  simuliert einen Schritt von  $T_u$  wie folgt: ist  $1^p0^{c_u - p}$  der aktuelle Blockinhalt in Band 3 und  $1^A0^{c_u - A}$  der aktuelle Blockinhalt in Band 2, so sucht  $U$  in  $u$  auf Band 1 einen Befehl, der mit  $1^p01^A$  beginnt. Findet  $U$  einen solchen Befehl  $1^p01^A01^q01^B01^{d+2}$ , so schreibt  $U$  in den Block auf Band 3 den neuen Zustand  $1^q0^{c_u - q}$  und in den aktuellen Block auf Band 2 das neue Symbol  $1^B0^{c_u - B}$  und rückt auf Band 2 um  $d$  Blöcke nach rechts.

Wir definieren jetzt eine Eigenschaft von Funktionen, die im folgenden eine wichtige Rolle spielt: die Konstruierbarkeit.

**Definition 5.1 (bandkonstruierbar)**

Eine (arithmetische) totale Funktion  $f$  heißt  $iDT^k$ -bandkonstruierbar ( $i = 0, 1, 2$ ,  $k \in \mathbb{N} - \{0\}$ ) gdw. es eine  $iDT^k[f]$ -TM gibt, die  $f$  unär berechnet, genauer: die bei Input  $1^n$  mit  $1^{f(n)}$  auf Band 1 stoppt.

**Definition 5.2 (zeitkonstruierbar)**

Eine (arithmetische) totale Funktion  $f$  heißt  $iDT^k$ -zeitkonstruierbar ( $i = 0, 1, 2$ ,  $k \in \mathbb{N} - \{0\}$ ) gdw. es eine  $iDT^k\langle f \rangle$ -TM gibt, die  $f$  binär berechnet, genauer: die bei Input  $1^n$  mit  $\text{bin}(f(n))$  auf Band 1 stoppt. ( $\text{bin}(n)$  = Binärdarstellung von  $n$ )

**Bemerkung 5.3**

Der Begriff der Konstruierbarkeit wird in der Literatur auf mancherlei Weise definiert.

- So wird auch eine Funktion  $f : \Sigma^* \rightarrow \{1\}^*$ , die auf allen Worten derselben Länge denselben Wert liefert, konstruierbar genannt, wenn sie von einer  $iDT^k[f]$  – TM berechnet wird. [Natürlich sind arithmetische Funktionen und Funktionen auf Worten, deren Wert aber nur von der Wortlänge beeinflußt wird i.W. dasselbe.]
- Oder eine arithmetische Funktion  $f$  heißt  $iDT^k$ -zeitkonstruierbar, wenn eine  $iDT^k\langle O(f) \rangle$  – TM bei Input  $1^n$  nach genau  $f(n)$  Schritten stoppt. Eine solche Maschine kann als Uhr parallel zu einer anderen geschaltet werden, um sicherzustellen, daß sie nicht länger als  $f(n)$  macht, aber dies erhöht die Bänderzahl.
- Oder eine arithmetische Funktion  $f$  heißt  $iDT^k$ -zeitkonstruierbar, wenn eine  $iDT^k\langle O(f) \rangle$  – TM bei Input  $1^n$  mit  $1^{f(n)}$  stoppt. Das hat den Vorteil, daß die Funktion id zeitkonstruierbar ist:

Für Funktionen  $f$  mit  $f(n) \geq (1 + \epsilon)n$  sind diese Varianten von Zeitkonstruierbarkeit äquivalent zur obigen Definition.

**Definition 5.4** ( $\log^* n$ )

$\log^* n = \min\{i \mid \log^i n \leq 1\} = \min\{i \mid i^2 \geq n\}$  (hierbei sei  $\log^i$  der  $i$ -fach iterierte Logarithmus und  $i^2$  die  $i$ -fach iterierte Exponentiation).

**Aufgabe 5.5**

1. Zeigen Sie, daß eine 0DT-Turingmaschine in  $\lceil \frac{5}{2}n \rceil - 2$  Schritten  $n$ -mal binär eine 1 addieren kann, d.h. von 1 bis  $n$  zählen kann.
2. Zeigen Sie, daß folgende Funktionen 1DT<sup>1</sup>-zeitkonstruierbar sind:  $n, n^2, n\sqrt{n}, n \log n, n \log^* n$ .

**Lösung von 1.:** Sei  $n$  eine Zweier-Potenz. Alle  $2^i$  Additionen muß die Turingmaschine  $M$  bis Feld  $i + 1$  gehen, d.h.  $M$  macht nach jedem 2-ten Besuch von Feld  $i$  (alle  $2^{i-1}$  Additionen) noch 2 weitere Schritte. Jede 2-te Addition kostet 2 Schritte [ $t(2) = 2$ ]. Jede 4-te Addition kostet noch zusätzlich 2 Schritte usw.

Die ungeraden Additionen zählen wir extra, sie kosten nur einen Schritt.

$$\begin{aligned} \text{Somit ist } t(n) &= \frac{n}{2} && \text{(alle ungeraden Additionen)} \\ &+ \frac{n}{2} \cdot 2 && \text{(alle 2-ten Additionen)} \\ &+ \frac{n}{4} \cdot 2 && \text{(alle 4-ten Additionen)} \\ &\vdots && \text{usw.} \end{aligned}$$

Ist  $n = 2^k$ , so ist

$$\begin{aligned} t(n) &= \frac{n}{2} + 2 \sum_{i=1}^k \frac{2^k}{2^i} = \frac{n}{2} + 2 \sum_{i=1}^k 2^i \\ &= \frac{n}{2} + 2(2^k - 1) = \frac{n}{2} + 2n - 2 = \frac{5}{2}n - 2 \end{aligned}$$

*Anderer Weg:* Es gilt die Rekursionsgleichung.

$$t(1) = 1, t(2^{i+1}) = 2 \cdot t(2^i) + 2$$

Dann ist  $t(n) = \frac{5}{2}n - 2$  eine Lösung, denn

$$\frac{5}{2}2^{i+1} - 2 = 5 \cdot 2^i - 2 = 5 \cdot 2^i - 4 + 2 = 2 \left( \frac{5}{2} 2^i - 2 \right) + 2$$

Sei  $n = 2^{j_1} + \dots + 2^{j_r}$ , ( $j_1 > j_2 > \dots > j_r$ ), so ist

$$\begin{aligned} t(n) &= t(2^{j_1}) + \dots + t(2^{j_r}) \\ &= \frac{5}{2}(2^{j_1} - 2) + \dots + \frac{5}{2}(2^{j_r} - 2) \\ &= \frac{5}{2}(2^{j_1} + \dots + 2^{j_r}) - 2^{r+1} \leq \frac{5}{2} \cdot n - 2 \end{aligned}$$

## 5.2 Die Band-Hierarchiesätze

Wir zeigen Hierarchien von sich umfassenden Komplexitätsklassen  $\{C_f \mid f \in \mathbb{R}\}$  ( $\mathbb{R}$  ist hier die Menge der totalen rekursiven Funktionen von  $\mathbb{N}$  nach  $\mathbb{N}$  - nicht etwa die Menge der reellen Zahlen). Dazu müssen wir in der Lage sein, eine TM zu definieren, deren Sprache nicht in einer gegebenen Klasse  $C_f$  liegt. Dies tun wir, indem wir eine TM bauen, die über alle TMen diagonalisiert, deren Sprache in der Klasse  $C_f$  liegt. Sei  $(T_u)_{u \in I}$  ( $I = \{1\}\Sigma^* \cup \{\varepsilon\}$ ) eine Nummerierung der Turingmaschinen (alle Indizes, Programme beginnen mit 1 oder sind leer).

Über eine Klasse diagonalisieren heißt gewöhnlich, daß diese Klasse aufzählbar sein muß. Nun wissen wir aber, daß die Klasse  $T_f := \{u \in \Sigma^* \mid L(T_u) \in C_f\}$  gewöhnlich nicht aufzählbar ist - wir sagen  $C_f$  ist nicht stark aufzählbar. Für konstruierbare  $f$  andererseits können wir leicht eine Teilklasse von  $T_f$  angeben, deren Maschinen auch genau die Sprachen von  $C_f$  erkennen und die aufzählbar ist - wir sagen  $C_f$  ist schwach aufzählbar. An jede TM  $M$  setzen wir einfach noch ein Kontrollmodul  $K$ , der  $f$  konstruiert und die Maschine stoppt, sobald sie mehr als  $f$  Aufwand treibt. Diese gekoppelte Maschine  $[MK]$  ist  $f$ -beschränkt, d.h. aus  $T_f$ , jede Sprache aus  $C_f$  kann von einer solchen Maschine erkannt werden (ist  $M$  schon  $f$ -beschränkt, so greift das Modul ja gar nicht ein) und die Teilklasse dieser Maschinen ist aufzählbar (wir zählen alle Turingmaschinen auf und koppeln sie mit  $K$ :  $([T_u K])_{u \in \Sigma^*}$ ). Die Diagonalisierungen, die wir hier definieren, machen genau dies, sie zählen alle Turingworte auf, koppeln sie aber gleich mit dem Kontrollmodul.

### 5.2.1 One-way Maschinen

#### Satz 5.6 (Bandhierarchie one-way I)

Sei  $|\Sigma| \geq 2$  und seien  $f$  und  $g$  totale arithmetische Funktionen,  $g$  1DT-bandkonstruierbar. Dann gilt:

$$f \stackrel{\infty}{\prec} g \wedge g \geq \log \implies \text{weak-2DT}[f] \not\subseteq \text{strong-1DT}[g].$$

**Beweis:** Wir definieren eine Sprache  $L$  mit  $L \in \text{strong-1DT}[g]$ , aber  $L \notin \text{weak-2DT}[f]$ . Dies tun wir, indem wir über die Klasse  $\text{weak-2DT}[f]$  diagonalisieren. Wäre die Klasse  $\text{weak-2DT}[f]$  aufzählbar, so könnten wir einfach eine TM  $\Delta$  bauen, die bei Input  $w$  die  $i$ -te TM der Klasse antisimuliert, wenn  $w$  das  $i$ -te Wort einer Aufzählung von  $\Sigma^*$  ist. Diese TM  $\Delta$  würde sich dann von jeder TM dieser Klasse unterscheiden, d.h. eine Sprache erkennen, die nicht in  $\text{weak-2DT}[f]$  liegt. Leider ist die Klasse  $\text{weak-2DT}[f]$  nicht aufzählbar. Wir helfen uns, indem wir eben eine  $\text{weak-2DT}[f]$  umfassende größere Klasse aufzählen und über diese diagonalisieren. Wenn dies eine  $\text{strong-1DT}[g]$ -TM kann, haben wir den Satz bewiesen.

O.B.d.A sei  $\{0, 1\} \subseteq \Sigma$ . Jedes Wort  $w \in \Sigma^*$  kann geschrieben werden als  $w = xuz$  mit  $x \in \{0, 1\}^*$ ,  $u \in \{0, 1\}^*$ ,  $z \in \Sigma^*$ , wobei  $u$  mit 1 beginnt, wenn  $u \neq \varepsilon$  und  $z$  mit einem Symbol  $a \notin \{0, 1\}^*$  beginnt, wenn  $z \neq \varepsilon$  (d.h.  $x$  ist der maximale 0-Präfix von  $w$  und  $u$  der maximale auf  $x$  folgende  $\{0, 1\}$ -String von  $w$ ). Zur Erinnerung:  $f \stackrel{\infty}{\prec} g$  oder  $\inf \frac{f}{g} = 0$  heißt  $\forall c \in \mathbb{N} \exists n : cf(n) < g(n)$ .

Sei  $(T_u)_{u \in I}$  ( $I = \{1\}\Sigma^* \cup \{\varepsilon\}$ ) eine Nummerierung der two-way 1-Band-Turingmaschinen. Wir definieren eine  $\text{strong-1DT}[g]$ -TM  $\Delta$ , die über  $(T_u)_{u \in I}$  diagonalisiert. Sei  $q_u$  die Zahl der Zustände und

$\gamma_u$  die Zahl der Arbeitssymbole der TM  $T_u$ . Der Input  $w$  habe die oben beschriebene Form  $w = xuz$  und habe die Länge  $n$ . Dann arbeitet  $\Delta(w)$  in zwei Phasen:

**Phase I:**  $\Delta$  konstruiert  $g$  und markiert  $g(n)$  Felder auf Spur 1 [Platzaufwand  $g(n)$ ]. Gleichzeitig kopiert  $\Delta$  den Input  $w = xuz$  auf Spur 2 und 3 und zwar  $x$  binär als  $\text{bin}(|x|)$  in Spur 2 und den Rest  $uz$  direkt darunter in Spur 3, wobei  $\Delta$  sofort stoppt und verwirft, wenn  $|uz| > |\text{bin}(|x|)|$  [Platzaufwand  $\log |x| \leq \log n \leq g(n)$ ].

**Phase II:** Danach antisimuliert  $\Delta$  nun  $T_u(w)$  auf Spur 4, d.h.  $\Delta$  simuliert  $T_u(w)$  ähnlich wie die oben beschriebene UTM mit dem Unterschied, daß der Input eine veränderte Form hat. Solange der Inputkopf von  $T_u$  auf  $x$  weilt, subtrahiert (addiert) der simulierende Kopf von  $\Delta$  von der Binärdarstellung  $\text{bin}(|x|)$  jedesmal eine 1, wenn der Inputkopf von  $T_u$  ein Feld weiterrückt (zurückrückt) [Platzaufwand  $c_u f(n)$ ]. Parallel dazu zählt  $\Delta$  die simulierten Schritte von  $T_u$  binär auf Spur 5, aber nur, solange der markierte Platz von  $g(n)$  Feldern ausreicht.

$\Delta$  akzeptiert, wenn der markierte Platz von  $g(n)$  Feldern zur Simulation nicht ausreicht oder wenn der Platz ausreicht, aber  $T_u(w)$  verwirft. Reicht der Platz aus und  $T_u(w)$  akzeptiert, so verwirft  $\Delta$ . Damit ist  $\Delta$  eine strong-1DT[ $g$ ]-TM.

Reicht der markierte Platz nicht aus, so kann das mehrere Gründe haben:

1.  $T_u$  ist gar keine weak-2DT[ $f$ ]-TM, verwendet also mehr als  $f(n)$  Platz, oder
2.  $T_u(w)$  verwirft (eine weakly beschränkte TM muß sich nur bei akzeptierenden Läufen an die Bandschranke halten), oder
3.  $c_u f(n) > g(n)$  (d.h.  $n$  war nicht geeignet), oder
4.  $T_u$  macht mehr als  $2^{\log n + 2c_u f(n)} = n \cdot 2^{c_u f(n)} \geq n \cdot q_u \gamma_u^{f(n)} f(n)$  Schritte. Dann zyklert  $T_u$ , weil sie mehr Schritte macht als verschiedene Situationen existieren, d.h.  $\log n + 2c_u f(n)$  Felder reichen nicht aus, um die Schritte zu zählen.

Wir nennen eine Antisimulation erfolgreich, wenn der Platz reicht oder höchstens wegen Punkt 2. oder 4. nicht reicht. Wir müssen noch garantieren, daß  $\Delta$  jede weak-2DT[ $f$ ]-TM  $T_u$  für irgendeinen Input auch erfolgreich antisimulieren kann. Für jedes  $u$  existiert aber ein  $x$ , sodaß  $|u| \leq |\text{bin}(|x|)|$  und  $c_u f(|xu|) \leq g(|xu|)$ . Ist  $T_u$  also eine weak-2DT[ $f$ ]-TM, so gibt es einen Vorspann  $x$  zu  $u$ , so daß  $\Delta(xu)$  die TM  $T_u$  auf  $xu$  erfolgreich antisimuliert.

Wäre nun  $L(\Delta) \in \text{weak-2DT}[f]$  und  $T_u$  eine weak-2DT[ $f$ ]-TM, die  $L(\Delta)$  erkennt, so gäbe es doch ein  $x$ , so daß  $\Delta(xu)$  die TM  $T_u$  auf  $xu$  erfolgreich antisimuliert, so daß also  $\Delta(xu)$  akzeptiert gdw.  $T_u(xu)$  verwirft. W! Damit ist  $L(\Delta) \notin \text{weak-2DT}[f]$ . ■

**Korollar 5.7**

1.  $f \in o(g) \wedge g \geq \log \implies \text{strong-1DT}[f] \subsetneq \text{strong-1DT}[g]$ .
2.  $f \in o(g) \wedge g \geq \text{id} \implies \text{weak-1DT}[f] \subsetneq \text{strong-1DT}[g]$ .

**Beweis:**

**ad 1.:**  $\text{strong-1DT}[f] \subseteq \text{strong-1DT}[g]$ , falls  $f \in o(g)$ .

**ad 2.:** Wenn  $g \geq \text{id}$ , dann ist  $\text{weak-1DT}[g] = \text{strong-1DT}[g]$ . Ist nämlich  $M$  eine weak-1DT[ $g$ ]-TM, dann gibt es eine äquivalente strong-1DT[ $g$ ]-TM  $M'$ , die zunächst  $g(n)$  konstruiert und parallel dazu den Input auf das Arbeitsband kopiert (da  $g \geq \text{id}$  reicht der Platz), anschliessend  $M$  auf dem kopierten Input simuliert und dabei verwerfend abbricht, falls mehr als  $g(n)$  Platz verbraucht wird. ■

Was ist, wenn wir im Korollar die Voraussetzung  $g \geq \log$  fallen lassen? Für eine Antwort benötigen wir noch folgende Beobachtung.

**Satz 5.8 (1DT-bandkonstruierbar  $\prec \log$ )**

Ist  $g : \mathbb{N} \rightarrow \mathbb{N}$ , total, unbeschränkt und 1DT-bandkonstruierbar, so ist  $g \in \Omega(\log)$  (d.h.  $\sup \frac{\log}{g} > 0$ ,

d.h.  $\exists c > 0 \exists n : g(n) \geq \frac{1}{c} \log n$ . Anschaulich: eine unbeschränkte, 1DT-bandkonstruierbare Funktion muß unendlich oft einen Wert in der Ordnung von  $\log$  annehmen.

**Beweis:** Sei  $\text{MIN}(g) = \{n \in \mathbb{N} \mid \forall m < n : g(m) < g(n)\}$ , d.h. die Menge der Stellen in  $\mathbb{N}$ , an denen  $g$  jeweils zum ersten Mal einen größeren Wert als an früheren Stellen annimmt. (Da  $g$  unbeschränkt ist, gilt  $|\text{MIN}(g)| = \infty$ .) Sei weiter  $n \in \text{MIN}(g)$  und  $M$  eine 1DT[ $g$ ]-TM, die  $g$  konstruiert. Eine Konfiguration von  $M$  ist gegeben durch Zustand, Arbeitsbandinhalt und Position des Arbeitskopfes, aber ohne Angabe des Inputbandes und der Inputkopfstellung (vgl. Definition 2.9).  $M$  muß beim Verlassen jedes der  $n$  Zeichen von  $1^n$  in einer anderen Konfiguration sein, denn wäre  $M$  beim Verlassen der  $i$ -ten 1 und der  $j$ -ten 1 ( $i < j$ ) in derselben Konfiguration, so würde  $M$  auch auf dem Wort  $1^{n-(j-i)}$  am Ende in derselben Konfiguration sein und somit auch denselben Output liefern wie bei  $1^n$ . Damit wäre also  $g(n) = g(n - j + i)$  im Widerspruch zu  $n \in \text{MIN}(g)$ .  $M$  hat höchstens  $q\gamma^{g(n)}g(n) \leq 2^{cg(n)}$  Konfigurationen für ein  $c$ , benötigt aber mindestens  $n$  viele, d.h. es muß gelten  $2^{cg(n)} \geq n$  oder  $g(n) \geq \frac{1}{c} \log n$ . Dies ist allerdings nur für alle  $n \in \text{MIN}(g)$  zwingend. ■

Nun können wir Voraussetzung  $g \geq \text{id}$  durch die schwächere  $g$  unbeschränkt ersetzen, wenn die Diagonal-TM schwach beschränkt sein darf (also eine mächtigere Maschine):

**Satz 5.9 (Bandhierarchie one-way II)**

Sei  $|\Sigma| \geq 2$  und seien  $f$  und  $g$  totale arithmetische Funktionen,  $g$  1DT-bandkonstruierbar:

$$f \in o(g) \wedge g \text{ unbeschränkt} \implies \text{weak-1DT}[f] \subsetneq \text{weak-1DT}[g].$$

**Beweis:** Wir konstruieren eine one-way-Diagonal-TM  $\Delta$ .  $\Delta$  muß, wenn es  $g$  konstruiert gleichzeitig den Input kopieren, damit er später noch zur Verfügung steht. Ist  $g \geq \text{id}$ , so kann nichts passieren. Ist  $g \geq \log$ , so kann  $\Delta$  den Input immer noch komprimiert kopieren: hat der Input die Form  $w = xuz$  wo  $x$  der maximale Präfix  $\in \{0\}^*$ ,  $u$  der maximale Teil nach  $x$  aus  $\{0, 1\}^*$ ,  $z \in \Sigma^*$  der Rest von  $w$  ist, dann kann  $x$  in binärer Form ( $\text{bin}(|x|)$ ) kopiert werden. Wenn  $x$  genügend groß ist, kann  $\Delta$  den Rest von  $w$  darunter kopieren, ohne den von  $\text{bin}(|x|)$  abgesteckten Platz zu überschreiten.

Nun haben wir im obigen Satz 5.8 gezeigt, daß eine 1DT-bandkonstruierbare Funktion unendlich oft einen Bruchteil von  $\log$  erreichen muß, d.h.  $\frac{1}{c} \log n \geq g(n)$  für unendlich viele  $n$  und ein  $c$ . Sei  $I = \{n \mid \frac{1}{c} \log n \geq g(n)\}$ . Für ein  $n \in I$  könnte  $\Delta$  also den Input  $w = xuz$  (in  $c$ -Kompression) kopieren, wenn  $|w| = n$  und  $\log |x| \geq |uz|$ , da  $\frac{1}{c} \log |x| \leq \frac{1}{c} \log n \leq g(n)$ . Für jedes  $u \in \{0, 1\}^*$  gibt es aber ein  $x \in \{0\}^*$  mit  $\log |x| \geq |u|$ , so daß  $|w| \in I$ .

$\Delta$  arbeitet also genauso wie in dem Hierarchiesatz 5.6 oben mit folgenden Ausnahmen:

1.  $\Delta$  kopiert den Input in der eben beschrieben Weise, d.h. den maximalen Präfix  $x = 0^m$  aus  $\{0\}^*$  als  $\text{bin}(m)$  in  $c$ -Kompression auf Spur 2, was  $\frac{1}{c} \log m$  Felder kostet, und kopiert darunter in Spur 3 den Rest des Inputs.
2.  $\Delta$  verwirft, wenn diese Kopie des Inputs mehr als  $g(n)$  Platz verwandt hat, d.h. wenn  $\frac{1}{c} \log m > g(n)$  oder  $|u| > g(n)$ , was  $\Delta$  nach Konstruktion von  $g$  weiß.

Für jede TM  $T_u$  gibt es nach obiger Überlegung unendlich viele Inputs  $0^m u$ , deren komprimierte Kopien höchstens  $g(|0^m u|)$  Platz benötigen. Da  $f \in o(g)$ , existiert für jedes  $u$  ein Input  $w = 0^m u$ , so daß  $\Delta(w)$  erfolgreich antisimuliert. Alle übrigen Überlegungen laufen wie im Beweis von Satz 5.6 . ■

**Aufgabe 5.10**

Was ist falsch an dem folgenden Beweis zu vorigem Satz 5.9 ?

**Beweis:** Da  $g$  sehr klein sein darf, können wir nicht mehr darauf hoffen, daß Platz  $g(n)$  ausreicht, um den ganzen Input auf das Arbeitsband zu kopieren. Das brauchen wir auch nicht, wenn wir das

Konstruieren von  $g$  und die Simulation parallel machen, so daß beide Module den Input vom 1-way-Band (Band 0) beziehen. Der Input  $w$  hat die Form  $w = uz$  mit  $u \in \{0, 1\}^*$  und  $z \in \Sigma^*$ , wobei  $z$  mit einem Symbol  $a \notin \{0, 1\}^*$  beginnt, wenn  $z \neq \varepsilon$ , d.h.  $u$  ist der maximale  $\{0, 1\}$ -Präfix von  $w$ .

Modul 1 konstruiert  $g$  auf Spur 1. Gleichzeitig arbeitet Modul 2 in mehreren Phasen: In Phase 1 kopiert Modul 2 vom Input  $w = uz$  den Präfix  $u$  auf Spur 2 und 3. Steht der Inputkopf hinter  $u$ , so laufen die Köpfe auf Spur 2 und 3 an den Anfang zurück und es beginnt Phase 2. Hier simuliert Modul 2 die Maschine  $T_u(w)$  auf Spur 4, während der Kopf auf Spur 2 das Turingprogramm  $u$  abliest und der Kopf auf Spur 3 den Inputkopf simuliert. In dieser 2. Phase wartet der Inputkopf (und damit auch Modul 1). Ist der Kopf auf Spur 3 während der Simulation an das Ende von  $u$  gekommen, so endet Phase 2. In Phase 3 fährt Modul 2 mit der Simulation fort, wobei aber jetzt der Inputkopf seine Funktion wieder übernimmt (und auch Modul 1 wieder anläuft). Wenn wir sagen, daß die Module 1 und 2 parallel laufen, so bedeutet das, daß sie nach jeder Inputphase aufeinander warten, bis beide den Inputkopf weiterbewegen wollen. Die Simulation selbst verläuft genau wie im vorigen Beweis beschrieben. Die simulierende Diagonalmaschine  $\Delta$  ist nur weakly beschränkt, weil erst am Ende der Simulation auch Modul 1 den erlaubten Platz  $g(n)$  konstruiert hat und wir erst jetzt wissen, ob die Simulation auf  $g(n)$  Platz stattgefunden hat. In verwerfenden Läufen kann also der Bandaufwand größer als  $g(n)$  werden.<sup>3</sup> ■

Bisher haben wir vorausgesetzt, daß  $|\Sigma| \geq 2$ . Diese Einschränkung hat den Beweis vereinfacht, ist aber nicht notwendig. In einer Theorie mit einem einelementigen Alphabet gelten diese Hierarchiesätze genauso. Beispielsweise:

**Satz 5.11 (Bandhierarchie one-way III)**

Seien  $|\Sigma| = 1$ ,  $f$  und  $g$  totale arithmetische Funktionen,  $g$  1DT-bandkonstruierbar. Dann gilt

$$f \in o(g) \wedge g \geq \log \implies \text{weak-1DT}[f] \subsetneq \text{strong-1DT}[g].$$

**Beweis:** Wir können den Input aufs Arbeitsband kopieren, freilich nicht original, sondern binär. Sei also  $w = 0^n$ , so konstruiert die Diagonalmaschine  $\Delta$  die Funktion  $g$  und schreibt gleichzeitig auf Band 1 die Binärdarstellung  $\text{bin}(n)$ . Sei  $\text{bin}(n) = 1xu$  mit  $x \in \{0\}^*$ ,  $u \in \{0, 1\}^*$ , wobei  $u$  mit 1 beginnt, wenn  $u \neq \varepsilon$ . Da  $g \geq \log$  reicht der Platz für die Antisimulation aus. Ansonsten geht der Beweis ähnlich wie der von Satz 5.6. ■

Unmöglichkeitbeweise (Hierarchien, untere Schranken) führt man in der Regel mit zwei Strategien: Diagonalisierung und Kombinatorik (Zählen der Ressourcen eines Maschinentyps). Die Diagonalisierung erfordert eine Diagonalmaschine, die mächtig genug ist, alle Maschinen einer Klasse zu simulieren. Beim Zählen untersucht man den Index von Äquivalenzen auf  $\Sigma^*$ , die durch Maschinen induziert werden, bzw. durch eine Spracheigenschaft nahegelegt werden. Das erfordert relativ kleine Maschinen, deren Konstellationen überschaubar sind. Bei Maschinentypen, die zum Diagonalisieren zu schwach sind, greift man daher zur zweiten Strategie, den Zählargumenten.

Für kleine Bandschranken können wir Sätze der obigen Art auch mithilfe von Zählargumenten beweisen.

**Satz 5.12 (Bandhierarchie one-way IV)**

Sei  $|\Sigma| \geq 3$  und seien  $f$  und  $g$  totale Funktionen und  $g$  1DT-bandkonstruierbar. Dann gilt

$$f \overset{\infty}{\succ} g \wedge g < \frac{1}{2} \text{id}, g \text{ unbeschränkt} \implies \text{weak-1DT}[g] \not\subseteq \text{weak-1DT}[f].$$

**Beweis:**  $L = \{uxu \mid u \in \{0, 1\}^*, x \in \{\#\}^*, |u| = g(|uxu|)\}$  (Da  $g < \frac{1}{2}n$ , gibt es wenigstens ein  $\#$ .)  $L$  ist aus  $\text{weak-1DT}[g]$ : Die TM  $M$  konstruiert  $g$  und schreibt gleichzeitig  $u$  aufs Arbeitsband und

<sup>3</sup>Fehler: Hat der Platz nicht ausgereicht, weil die simulierte schwach beschränkte TM verwirft und mehr als  $g$  Platz verwendet, dann muß  $\Delta$  akzeptieren. Dann ist  $\Delta$  aber nicht mehr (schwach)  $g$ -beschränkt.

vergleicht anschliessend das Arbeitsband mit dem zweiten  $u$ . Wenn der Input aus  $L$  ist, so reicht der Platz  $g$  aus und  $M$  akzeptiert. Reicht der Platz nicht aus, so verwirft  $M$ , allerdings kann  $M$  dann schon zuviel Platz verbraucht haben, so daß  $M$  nur eine weak-1DT[ $g$ ]-TM ist.

Gäbe es eine weak-1DT[ $f$ ]-TM  $M$  für  $L$ , so würde  $M$  bei zwei der  $2^{g(n)}$  Worte der Länge  $n$  aus  $L$  - etwa bei  $uxu$  und  $v xv$  - beim Lesen des ersten  $\#$ , in derselben Konfiguration (Zustand, Stellung des Arbeitskopfes, Arbeitsbandinhalt) sein, da  $M$  bei Inputs aus  $L$  nur  $2^{cf(n)}$  Konfigurationen einnehmen kann, was weniger ist als  $2^{g(n)}$ , wenn wir ein  $n$  mit  $cf(n) \leq g(n)$  gewählt haben. Dann würde aber  $M$  auch  $uxv$  und  $v xu$  akzeptieren. W! ■

### 5.2.2 Two-way Maschinen

Die Diagonalisierungen gehen bei 2-way-Maschinen ganz analog. Wir geben daher nur die Idee des Beweises an:

#### Satz 5.13 (Bandhierarchie two-way I)

Sei  $|\Sigma| \geq 2$  und seien  $f$  und  $g$  totale arithmetische Funktionen und  $g$  2DT-bandkonstruierbar.

1.  $f \in o(g) \wedge g \geq \log \implies \text{weak-2DT}[f] \subsetneq \text{strong-2DT}[g]$ .
2.  $f \overset{\infty}{\prec} g \wedge g \geq \log \implies \text{weak-2DT}[f] \not\subseteq \text{strong-2DT}[g]$ .

**Beweis:** Der Beweis läuft ganz analog zum 1-way-Fall durch Diagonalisierung. Allerdings braucht die strong-2DT[ $g$ ]-Diagonal-TM  $\Delta$  den Input bei der Konstruktion von  $g$  nicht abschreiben, da sie 2-way ist. Dennoch muß  $g \geq \log$  sein, weil  $\Delta$  die Schritte der TM zählt, die sie antisimuliert, um Zykel zu entdecken. Die hat aber - wenn sie  $f$ -beschränkt ist,  $n2^{cf(n)}$  mögliche Situationen (für ein  $c$ ), da auch sie 2-way ist. Damit muß ein Zähler der Länge  $\log n + cf(n)$  auf  $g(n)$  Feldern Platz finden. Außerdem ist  $\text{weak-2DT}[g] = \text{strong-2DT}[g]$ , wenn  $g \geq \log$  und 2DT-bandkonstruierbar ist. ■

Tatsächlich reicht die Hierarchie aber bis  $\log \log$  hinunter. Bei der Diagonalisierung taucht unter  $\log$  allerdings die Schwierigkeit auf, daß die Diagonalmaschine  $\Delta$  nicht Platz genug hat, die Schritte der simulierten TM zu zählen. Sie hat also keine Möglichkeit zu stoppen, wenn die simulierte TM bei kleinem Bandaufwand zyckelt. Hier hilft uns ein Satz, der besagt, daß es zu jeder Turingmaschine eine äquivalente mit demselben Bandaufwand gibt, die immer stoppt, es sei denn, sie verwendet unendlich viel Band, d.h. die nie auf einem endlichem Bandstück zyckelt.

#### Satz 5.14 (Sipser: haltende bandbeschränkte TMen)

Zu jeder 2DT-TM gibt es eine äquivalente (die dieselbe Sprache erkennt, bzw. dieselbe Funktion berechnet), die denselben Bandaufwand hat und die auf jedem Input, bei dem der Bandaufwand endlich ist, hält.

**Beweis:** Sei  $M$  eine 2DT-TM. OBdA nehmen wir an, daß  $M$  nie löscht, sondern stattdessen ein Pseudoblank  $\boxtimes$  druckt, und nur mit (pseudo-) leerem Band und Inputkopf auf Feld 1 stoppt, d.h. die Stoppsituationen haben die Form  $s_q(k) = (q, \#w, \#\boxtimes^k)$  für ein  $k \in \mathbb{N}$ . Wir konstruieren eine TM  $M'$ , die nicht zyckelt, d.h. unendlich viel Band beschreibt, wenn sie nicht stoppt. Sei  $w$  der Input,  $|w| = n$ ,  $b$  der Bandaufwand von  $M(w)$  ( $M$  angewandt auf  $w$ ) und  $s_0$  die Startsituation von  $M(w)$ .

Sei  $s$  eine Situation, in der  $M$  stoppt oder von der aus  $M$  in einem Schritt eine Situation erreicht, die einen größeren Bandaufwand hat. Wir zeigen zunächst, daß  $M'$  für solche Situationen  $s$  herausfinden kann, ob  $M$  sie von der Startsituation  $s_0$  aus erreicht ( $s_0 \vdash_M^* s$ ). Ist  $\bar{s}$  ein direkter Vorgänger von  $s$  ( $\bar{s} \vdash_M s$ ), so unterscheidet sich  $\bar{s}$  von  $s$  außer im Zustand, in der Inputposition und der Arbeitskopfposition nur noch im Inhalt der drei Felder unter und neben dem Arbeitskopf. Es gibt also höchstens  $l$  viele Vorgängerkandidaten für  $s$  für ein festes  $l$ , von denen man leicht feststellt, welcher tatsächlich Vorgänger ist, indem man den passenden Befehl sucht. Weiter kann man alle Situationen in eine

lineare Ordnung bringen, etwa indem man die Zustände und die Bandinhalte ordnet.  $M'$  kann also zu einer gegebenen Situation  $s$  den ersten direkten Vorgänger  $\bar{s}$  und zu einer Situation  $s$  mit einem direkten Vorgänger  $\bar{s}$  einen nächsten direkten Vorgänger finden oder feststellen, daß es einen nächsten nicht gibt. Zu jedem direkten Vorgänger gibt es eventuell wieder mehrere direkte Vorgänger etc., d.h. wir haben einen Vorgängerbaum.

Die Situation  $s$  wird von  $s_0$  aus erreicht, wenn  $s_0$  ein Blatt des Vorgängerbaums von  $s$  ist.  $M'$  muß also nur diesen Vorgängerbaum im Backtrackingverfahren durchsuchen, um festzustellen, ob  $s_0$  ein Vorgänger von  $s$  ist oder nicht. Man muß sich allerdings vergegenwärtigen, daß dieser Baum auch endlich ist. Dies ist freilich der Fall, da der Bandaufwand der Vorgänger nicht größer ist als der von  $s$ . Da die Zahl der Situationen einer festen Länge  $b$  begrenzt ist durch  $n2^{cb}$  für ein von  $M$  abhängiges  $c$ , kann auch der Pfad des (eingeschränkten) Vorgängerbaums nicht länger als  $n2^{cb}$  sein, wenn wir wissen, daß sich keine Situation auf diesem Pfad wiederholt, d.h. wenn auf diesem Pfad kein Zykel existiert. Da eine deterministische TM einen Zykel nicht mehr verlassen kann, kann auf einem Pfad ein Zykel nur existieren, wenn  $s$  selbst in diesem Zykel liegt. Da  $M$  aber in  $s$  stoppt, bzw. eine Situation mit größerem Bandaufwand erreicht, kann  $s$  in keinem Zykel liegen.

Man merke, daß  $M'$  nur lokales merken muß, um diesen Backtracking-Durchlauf zu realisieren: die gerade betrachtete Situation, und eine unmittelbare Vorgängersituation.

$M'$  untersucht nun für alle Stopsituationen der Form  $(q, \#w, \#\boxtimes^k)$  mit Bandaufwand  $k$ , ob sie von der Startsituation  $s_0$  aus erreichbar sind, dies für  $k = 1, 2, 3, \dots, b$ , wobei  $b$  der Bandaufwand ist, den  $M(w)$  treibt. Wird für alle  $k \in [1, b]$  keine Stopsituation erreicht, so stoppt  $M$  nicht. Dann kann  $M'$  stoppen und verwerfen.

$M'$  kennt freilich dieses  $b$  nicht, weiß also nicht, bei welchem  $k$  abgebrochen werden kann. Um zu prüfen, ob  $M(w)$  überhaupt eine Situation mit Bandbedarf  $k + 1$  erreicht, zählt  $M'$  alle Situationen  $s$  auf, die Bandbedarf  $k$  haben und deren direkter Nachfolger Bandbedarf  $k + 1$  hat, und überprüft, ob  $s$  von  $s_0$  aus erreicht wird. Gibt es keine solche Situation, so kann  $M'$  aufhören, das  $k$  zu erhöhen. Nur wenn  $M(w)$  unendlichen Bandaufwand treibt, wird auch  $M'$  zu keinem Ende kommen.

Bleibt zu betrachten, wie sich  $M'$  eine oder zwei Situationen von  $M$  merkt, und wie  $M'$  Situationen aufzählt. Eine Situation von  $M$  merkt sich  $M'$ , indem es selbst die Inputposition einnimmt und ein Band besitzt mit Inhalt und Kopfposition, wie sie in der Situation angegeben ist. Daneben kann  $M'$  weitere Arbeitsbänder (oder Spuren haben), deren Bandbedarf aber nicht größer sein darf.  $M'$  kann sich so auch zwei Situationen merken, wenn ihre Inputpositionen sich nur gering unterscheiden. Dann merkt sich  $M'$  nämlich nur eine Inputposition und zusätzlich den geringen Unterschied auf einem Arbeitsband. Wenn wir die Ordnung der Situationen so definieren, daß benachbarte Situationen auch benachbarte oder gleiche Inputpositionen haben, so kann  $M'$  immer die nächste Situation in dieser Ordnung konstruieren. ■

Mit diesem Satz nun können wir die Diagonalisierung weitertreiben.

### Satz 5.15 (Bandhierarchie two-way II)

Sei  $|\Sigma| \geq 1$ ,  $f$  und  $g$  totale, arithmetische Funktionen und  $g$  2DT-bandkonstruierbar.

1.  $f \in o(g) \wedge g$  unbeschränkt  $\implies$  weak-2DT[ $f$ ]  $\subsetneq$  strong-2DT[ $g$ ].
2.  $f \overset{\infty}{\prec} g \wedge g$  unbeschränkt  $\implies$  weak-2DT[ $f$ ]  $\not\subseteq$  strong-2DT[ $g$ ].

**Beweis:** Der Beweis läuft ganz wie im Fall  $g \geq \log$  mit dem Unterschied, daß die Diagonalmaschine  $\Delta$  die Schritte der simulierten Maschine nicht mitzählt - was sie auf so wenig Platz ja auch nicht kann. Damit stoppt  $\Delta$  nicht, wenn die simulierte TM bei endlichem Bandbedarf zyckelt, d.h. auch  $\Delta$  zyckelt, obwohl sie den konstruierten Platz  $g(n)$  nicht überschreitet. Nach dem vorigen Satz wissen wir nun, daß es zu  $\Delta$  eine äquivalente TM  $\Delta'$  geben muß, die immer hält. Wir nehmen nur eine kleine Änderung in dem Beweis dieses Satzes vor: wenn  $\Delta'$  feststellt, daß  $\Delta$  zyckelt, so stoppt  $\Delta'$  akzeptierend

(und nicht verwerfend). ■

Auch bei 2-way-Maschinen funktioniert der Beweis mit Zählargumenten, nur muß das Argument etwas anders lauten, weil diese Maschinen mächtiger sind. Wir zählen Objekte, die wir erst definieren müssen.

**Definition 5.16 (Crossing-Sequence, Crossing-Menge)**

Sei  $M$  eine (deterministische oder nichtdeterministische)  $k$ -Band-TM (one-way oder two-way) und  $R$  ein Lauf von  $M$ . Wir betrachten 2 benachbarte Felder  $l$  und  $l + 1$  ( $l \in \mathbb{Z}$ ) auf Band  $j$ .

- Eine  $j$ -Konfiguration von  $M$  hat die Form  $(q, u_0 \# v_0, \dots, u_{j-1} \# v_{j-1}, u_{j+1} \# v_{j+1}, \dots, u_k \# v_k)$  (d.h. eine  $j$ -Konfiguration ist eine Situation ohne die Information von Band  $j$ ). Ist  $k = 1$ , so besteht die 1-Konfiguration nur noch aus dem Zustand.
- Die Crossing Sequence  $C_j(R, l)$  [Crossing-Menge  $CM_j(R, l)$ ] des Laufes  $R$  an der Stelle  $l$  des Bandes  $j$  ist die Folge [Menge] der  $j$ -Konfigurationen, die während des Laufes  $R$  von  $M$  eingenommen werden, immer nachdem der Kopf die Grenze zwischen Feld  $l$  und Feld  $l + 1$  auf Band  $j$  gerade überquert hat.

**Bemerkung 5.17**

- Bei 0-Band-Maschinen ( $k = 0$ , nur ein Inputband) besteht eine Crossing-Sequence (Crossing-Menge) aus einer Folge (Menge) von Zuständen.
- In der Regel werden wir Crossing-Sequences und -Mengen auf dem Inputband ( $j = 0$ ) betrachten. Dann schreiben wir für  $C_0(R, i)$  auch  $C(R, i)$  oder  $C_R(i)$  oder einfach  $C(i)$ . Bei Input  $w = uv$  schreiben wir  $C_R(u, v)$  für  $C_0(R, |u|)$ . (analog bei Crossing-Mengen)

Ohne Beweis führen wir zwei einfache Lemmata an.

**Lemma 5.18**

Sei  $M$  eine (deterministische oder nichtdeterministische) 2-way  $k$ -Band-Turingmaschine,  $R$  ein  $uv$ -Lauf von  $M$ ,  $R'$  ein  $u'v'$ -Lauf von  $M$  und  $C_R(u, v) = C_{R'}(u', v')$ . Dann gibt es einen  $u'v'$ -Lauf  $R_1$  und einen  $uv'$ -Lauf  $R_2$  mit

1. ist  $|C_R(u, v)|$  gerade, so gilt:  
( $R_1$  akzeptierend  $\iff R'$  akzeptierend) und ( $R_2$  akzeptierend  $\iff R$  akzeptierend),
2. ist  $|C_R(u, v)|$  ungerade, so gilt:  
( $R_1$  akzeptierend  $\iff R$  akzeptierend) und ( $R_2$  akzeptierend  $\iff R'$  akzeptierend),
3. sind  $R$  und  $R'$  akzeptierend, so sind auch  $R_1$  und  $R_2$  akzeptierend.

**Lemma 5.19**

Ist  $M$  eine deterministische 2-way  $k$ -Band-Turingmaschine, so gilt obiges Lemma 5.18 ebenfalls, wenn wir statt der Crossing Sequence  $C_R(u, v)$  die Crossing-Menge  $CM_R(u, v)$  setzen.

**Satz 5.20 (Bandhierarchie two-way III)**

Seien  $|\Sigma| \geq 3$ ,  $f$  und  $g$  totale Funktionen und  $g$  2DT-bandkonstruierbar. Dann gilt:

$$f \succ g \wedge g < \log \implies \text{strong-2DT}[g] \not\subseteq \text{weak-2DT}[f].$$

**Beweis:**  $L = \{uxu \mid u \in \{0, 1\}^*, x \in \{\#\}^*, |u| = 2^{g(|uxu|)}\}$  (Da  $g < \log$ , ist  $|u| < \frac{n}{2}$ , d.h. es gibt wenigstens ein  $\#$ ).  $L$  ist aus strong-1DT[ $g$ ]: Die TM  $M$  konstruiert  $g$ . Dann vergleicht sie den Teil vor den  $\#$ 'en mit dem nach den  $\#$ 'en, wobei sie die Inputposition jeweils auf dem Arbeitsband binär speichert. Reicht der Platz dazu nicht aus, so verwirft  $M$ . Reicht der Platz  $g$  aus und sind alle Vergleiche erfolgreich, so akzeptiert  $M$ .

Angenommen es gibt eine weak-2DT[ $f$ ]-TM  $M$  für  $L$ . Da es nur  $2^{2^{cf(n)}}$  verschiedene Crossingmengen bei einem Bandaufwand  $\leq f(n)$  gibt, gibt es wenigstens zwei unter den  $2^{2^{g(n)}}$  Worten der Länge  $n$  aus

$L$  (etwa  $uxu$  und  $v xv$ ), die am ersten  $\#$  dieselbe Crossing Menge  $CM(u, \#v) = CM(u', \#v')$  haben. Dann muß  $M$  aber auch  $uxv'$  und  $u'xv$  akzeptieren, Worte, die nicht zu  $L$  gehören. W!! ■

Was ist, wenn das Inputalphabet  $\Sigma$  einelementig ist, d.h. die Inputworte unär sind? Für  $f \in o(g)$  und  $g \geq \log$  können wir dies wie im one-way-Fall beweisen.

**Satz 5.21 (Bandhierarchie two-way IV)**

Sei  $|\Sigma| = 1$  und  $f$  und  $g$  totale arithmetische Funktionen und  $g$  2DT-bandkonstruierbar. Dann gilt:

$$f \in o(g) \wedge g \geq \log \implies \text{weak-2DT}[f] \subsetneq \text{strong-2DT}[g].$$

**Beweis:** Ist  $|\Sigma| = 1$ , so wird wieder das Wort binär aufs Arbeitsband geschrieben, um dem Input die zu antisimulierende Turingmaschine zu entnehmen.

Bei vorliegendem Input müssen wir uns entscheiden, welche Turingmaschine antisimuliert werden soll. Dabei muß garantiert werden, daß jede Turingmaschine bei unendlich vielen Worten antisimuliert wird, da die Antisimulation erst bei Worten genügend großen Länge erfolgreich sein wird, eine Länge, die wir aber nicht kennen. Die oben beschriebene Weise ist eine Möglichkeit, dies zu tun. Aber auch jede andere Möglichkeit, die das gesagte berücksichtigt, erfüllt diesen Zweck. So können wir auch die Nummer der kleinsten Primzahl nehmen, die  $n$  teilt, wenn  $n$  die Länge des Inputs ist. Dann wird die  $i$ -te Turingmaschine bei allen Worten antisimuliert, für die die  $i$ -te Primzahl  $p_i$  der kleinste Teiler ist. Davon gibt es unendlich viele. ■

Der Fall  $f \overset{\infty}{\prec} g$  ( $\inf \frac{f}{g} = 0$ ) bereitet hier allerdings Schwierigkeiten: Wie auch immer wir die zu antisimulierende Turingmaschine aus dem Input bestimmen, so wird es für jede Länge  $n$  immer nur eine Turingmaschine sein. Wenn nun gerade für alle die  $n$ , die eine bestimmte Turingmaschine - etwa die  $i$ -te - liefern,  $f(n) \geq g(n)$  ist, so wird diese  $i$ -te Turingmaschine nie erfolgreich antisimuliert, d.h. die Diagonalisierung klappt nicht. Dennoch haben wir Erfolg, wenn wir unsere Strategie etwas verfeinern.

**Aufgabe 5.22**

Sei  $|\Sigma| = 1$ ,  $f$  und  $g$  totale arithmetische Funktionen, beide 2DT-bandkonstruierbar .Behauptung:

$$f \overset{\infty}{\prec} g \wedge g \geq \log \implies \text{weak-2DT}[f] \subsetneq \text{strong-2DT}[g].$$

Was ist falsch am folgenden Beweis?

**Beweis:** Da  $f$  2DT-bandkonstruierbar ist  $\text{weak-2DT}[f] = \text{strong-2DT}[f]$ . Wir antisimulieren daher über strong bandbeschränkte TMen. Wir versuchen zu gegebenem  $c$  den Wert  $f(n)$  auf Platz  $\frac{1}{c}g(n)$  auszurechnen. Reicht der Platz  $g(n)$  nicht, so wissen wir auch, daß  $cf(n) > g(n)$ , da die Berechnung von  $f(n)$  nicht mehr Platz benötigt, als der Wert  $f(n)$ . Reicht der Platz, so wissen wir, daß  $cf(n) \leq g(n)$ . Auf diese Weise können wir für jedes  $c$  und für alle  $n$  auf Platz  $\frac{1}{c}g(n)$  feststellen, ob  $cf(n) \leq g(n)$ , d.h. ob wir es mit einem  $n$  zu tun haben, auf dem eine Antisimulation erfolgreich sein kann. Sei  $N$  die Menge dieser aussichtsreichen  $n$ . Wir bauen eine Diagonalmaschine  $\Delta_N$ , die nicht auf allen Inputs, sondern nur auf den aussichtsreichen diagonalisiert, also nicht auf  $\mathbb{N}$ , sondern nur auf  $N$ . Inputs, die nicht aus  $N$  stammen, akzeptiert  $\Delta_N$ . Ist der Input  $n$  das  $i$ -te Element aus  $N$ , so antisimuliert  $\Delta_N$  die  $i$ -te Turingmaschine auf  $n$ .<sup>4</sup> ■

<sup>4</sup>Fehler: Um festzustellen, das wievielte Element  $n$  aus  $N$  ist, müssen alle  $m < n$  überprüft werden, was auf  $g(n)$  Platz nur geht, wenn  $g(\nu) \leq g(n)$  für alle  $\nu < n$ , wenn  $g$  also monoton ist. Der Beweis könnte richtig gestellt werden, wenn man zeigen kann, daß es eine auf  $g(n)$  Platz berechenbare Funktion  $s$  gibt, die  $N$  surjektiv auf  $\mathbb{N}$  abbildet. Dann könnte man in einer der üblichen Weisen aus  $s(n)$  die TM bilden, die auf  $n$  antisimuliert werden soll.

**Satz 5.23 (Bandhierarchie two-way V)**

Sei  $|\Sigma| = 1$  und  $f$  und  $g$  totale arithmetische Funktionen und  $g$  2DT-bandkonstruierbar. Dann gilt:

$$f \stackrel{\infty}{\prec} g \wedge g \geq \log \implies \text{strong-2DT}[f] \subsetneq \text{strong-2DT}[g].$$

**Beweis:** Wir betrachten eine feste Nummerierung  $(T_i)_{i \in \mathbb{N}}$  der Turingmaschinen. Bisher haben wir bei jedem Input schon zu Anfang eine TM bestimmt, um sie zu antisimulieren. Nun wissen wir nicht, ob für all die Inputs  $n$ , auf denen wir die  $i$ -te TM antisimulieren, nicht gerade  $cf(n) > g(n)$  gilt, so daß eine erfolgreiche Antisimulation unmöglich ist. (Die Antisimulation auf Input  $1^n$  ist erfolgreich, wenn der Platz  $g(n)$  reicht - wenn der Schrittzähler überläuft, so gilt dies auch noch als erfolgreich). Darum wollen wir bei einem gegebenen Input  $1^n$  nicht eine feste TM antisimulieren, die sich aus  $n$  ergibt, sondern eine der ersten  $\log n$  TMen in unserer Nummerierung, die wir bisher noch nicht erfolgreich antisimuliert haben. Damit wird eine TM, ab einem festen  $n$  immer wieder - versuchsweise - antisimuliert, spätestens, wenn alle davorliegenden TMen erfolgreich antisimuliert wurden, die überhaupt erfolgreich simuliert werden können. Eine strong-2DT[ $f$ ]-TM wird dann auf jeden Fall einmal erfolgreich antisimuliert.

**A.** Nehmen wir zunächst einmal an,  $g$  sei monoton wachsend. Die Diagonalmaschine  $\Delta$ , angewandt auf  $1^n$ , muß erst einmal herausfinden, welche unter den ersten  $\log n$  TMen noch nicht bei früheren (kleineren) Inputs erfolgreich antisimuliert wurden. Unter diesen sucht  $\Delta$  eine erste, die sie erfolgreich simulieren kann (Jede Simulation endet - spätestens, wenn der Schrittzähler überläuft). Findet sie eine, so markiert sie diese, findet sie keine, so stoppt sie.

Um die nichtmarkierten unter den ersten  $\log n$  TMen zu finden, muß  $\Delta$  diese Simulationen auch für alle kleineren Inputs durchführen. Für jedes  $\nu < n$  wird höchstens eine TM markiert, nämlich die erste TM, die noch nicht bei Inputs kleiner  $\nu$  markiert wurde und die  $\Delta(\nu)$  erfolgreich antisimuliert. Gibt es eine solche nicht, so wird für dieses  $\nu$  keine TM markiert.  $\Delta$  angewandt auf Input  $n$  rechnet also auch auf allen Inputs  $\nu < n$ . Das bedeutet, daß  $\Delta(n)$  nicht nur  $g(n)$  berechnen muß, sondern auch  $g(\nu)$  für alle  $\nu < n$ . Da  $g(\nu)$  auf Platz  $g(n)$  berechnet werden muß, muß  $g$  also monoton wachsend sein. Weiter muß  $\Delta$  TMen auf Input  $\nu < n$  simulieren, d.h.  $\Delta$  muß  $\nu$  speichern können, ein Grund, warum  $g \geq \log$  sein muß.

$\Delta$  muß sich natürlich merken, welche der ersten  $\log n$  TMen sie schon markiert hat. Dazu führt sie ein binäres Wort der Länge  $\log n$  mit, das an der Stelle  $i$  eine 1 enthält, wenn die  $i$ -te TM markiert wurde. Da  $g \geq \log$  haben wir dafür genügend Platz. Immerhin ist das der Grund, warum wir bei Input  $n$  nur die ersten  $\log n$  TMen testen. Aber auch wenn wir nur die ersten  $\log^*(n)$  TMen testen, benötigen wir doch  $\log(n)$  Band, wie wir im vorigen Absatz gesehen haben.

Wir wollen diese Diagonalisierung „fast-überall-Diagonalisierung“ nennen, weil jede TM fast überall versuchsweise simuliert wird. Formaler definieren wir eine Funktion  $s : \mathbb{N} \rightarrow \mathbb{N}$  rekursiv wie folgt:

$$s(0) = 0 \text{ falls TM } M_0(0) \text{ erfolgreich antisimuliert werden kann, } = \infty, \text{ sonst. Und für } n > 0 : \\ s(n) = \min \{ i \leq \log n \mid \forall j < \log n : s(j) \neq i \text{ und TM } M_i(n) \text{ kann erfolgreich antisimuliert werden} \}.$$

Merke, daß  $\min \emptyset = \infty$ . Wir wollen hier  $\infty$  als undefiniert interpretieren.  $\Delta(n)$  antisimuliert die  $s(n)$ -te TM, falls  $s(n)$  definiert ist.

Folgende Prozeduren sollen das Gesagte präzisieren:

```
function Erfolg(n, i) (Wert:boolean);
  if  $\Delta(1^n)$  die  $i$ -te TM auf  $g(n)$  Platz erfolgreich simuliert
  then return true else return false;
```

```
function Mark(n) (Wert:{0,1}-Wort der Länge  $\log n$ );
  if  $n = 1$  then return  $\varepsilon$ ;
```

```

if  $n = 2$  then begin
  if Erfolg(1,1) then  $w := 1$  {markiere TM 1} else  $w := 0$  ; return  $w$  ;
end;
 $w := \text{Mark}(n - 1)$  ;
if  $\lfloor \log(n - 1) \rfloor < \lfloor \log n \rfloor$  then  $w := w0$ ;  $i := 1$ ;
while  $i \leq |w|$  do
  if  $w[i] = 1$  {TM  $i$  markiert} then  $i := i + 1$  else
    if Erfolg( $n, i$ ) then begin  $w[i] := 1$ ; return  $w$  end else  $i := i + 1$ ;
return  $w$ ;

```

(Bemerke, daß Erfolg( $n, i$ ) = true gdw.  $s(n) = i$ .)

**B.** Wenn  $g$  nicht monoton wächst, so kann es geschehen, daß  $\Delta$  nicht für alle  $\nu < n$  den Wert  $g(\nu)$  auf Platz  $g(n)$  errechnen kann,  $\Delta$  also nicht feststellen kann, welche TM bei Input  $\nu$  markiert wurde. In diesem Fall soll  $\Delta$  keine TM bei Input  $\nu$  markieren, also keine 1 in dem Bit-Vektor eintragen. Wenn die TM  $i$  bei Input  $\nu$  markiert wurde, so wird  $\Delta$  dies nicht feststellen, und eventuell noch einmal versuchen diese TM  $i$  zu antisimulieren. Dies ist eigentlich nichts Störendes oder gar Falsches, der einzige Nachteil ist, daß auf diese Weise TMen mehrmals antisimuliert und daher markiert werden können, was nur auf Kosten der Zeit geht.

Da  $g$  nicht beschränkt ist, gibt es immer wieder eine Stelle  $n$ , an der  $g(n)$  größer ist als alle Werte an früheren Stellen. An solchen Stellen können alle früheren Werte  $g(\nu)$  berechnet werden und alle TMen markiert werden, die früher schon erfolgreich antisimuliert wurden. Da es unendlich viele solche Stellen gibt, ist garantiert, daß nicht immer nur „vergessene“ TMen nachsimuliert werden, sondern auch immer wieder neue, die noch nie erfolgreich simuliert wurden.

Nun könnte es sein, daß ausgerechnet diese Stellen  $n$ , an denen  $g$  zum ersten Mal einen Wert annimmt, der größer ist als alle früheren, schlecht sind in dem Sinn, daß  $c_i f(n) > g(n)$ , wenn an der Stelle  $n$  die  $i$ -te TM antisimuliert werden soll. Dies kann dann gar nicht erfolgreich sein, d.h. an diesen Stellen wird keine TM markiert. Darum betrachten wir nicht diese Stellen, sondern eine Folge  $(n_i)_{i \in \mathbb{N}}$  mit

1.  $d_i f(n_i) < g(n_i)$ , wo  $d_i = \max\{c_j \mid j \leq \log n_i\}$  und  $c_j$  der Faktor ist, um den der Bandaufwand von  $\Delta$  bei der Simulation der TM  $j$  wächst, also z.B.  $c_j = \max\{|Q_j|, |\Gamma_j|\}$ , (d.h. die Stellen sind gut,  $\Delta$  kann alle kleineren TMen antisimulieren) und
2.  $\forall i : g(n_i) < g(n_{i+1})$  (die Folge der Werte steigt monoton).

Solch eine Folge muß es geben, da  $\forall c \exists n : c f(n) < g(n)$ . Z.B. wählen wir  $n_i = \min\{m \mid d_i f(m) < g(m) \wedge g(m) > g(n_{i-1})\}$ .  $n_i$  existiert, denn auch für  $c = \max\{d_i, g(n_{i-1})\}$  muß es ein erstes  $m$  geben, mit  $c f(m) < g(m)$ .

Bei dieser Folge können wir sicher sein, daß an jeder Stelle  $n_i$  der Teil des Binärwortes, der real überprüft wurde, und nicht eine 0 trägt, weil  $g$  an dieser Stelle nicht ausgerechnet werden konnte, größer ist als an den früheren Stellen. Damit werden immer mehr TMen antisimuliert, die noch nicht erfolgreich antisimuliert worden waren. ■

Der Fall  $g \leq \log$  hat im unären Fall eine weitere Schwierigkeit: wir können nicht den Input binär aufs Band schreiben. Da  $g \leq \log$ , können wir keine „fast-überall-Diagonalisierung“ machen. Wir müssen also dem Input  $n$  eine TM entnehmen, deren Turingwort eine Länge hat, die  $g(n)$  nicht überschreitet. Dabei muß garantiert werden, daß eine TM bei unendlich vielen Inputs  $n$  ausgewählt wird.

Eine Möglichkeit ist, den Logarithmus des kleinsten Nichtteiler von  $n$  zu nehmen. Für jedes  $i$  gibt es unendlich viele Inputs  $n$  mit  $\log(\text{kN}(n)) = i$ , da es für jedes  $i$  unendlich viele  $n$  gibt mit  $\text{kN}(n) = 2^i$ . Wie wir später beweisen werden, liegt die Funktion  $\text{kN}(n)$  (kleinster Nichtteiler von  $n$ ) unter  $c \log n$  für ein  $c$ , d.h. er läßt sich auf  $\log \log$  Band niederschreiben, was genügt, wenn  $g \succeq \log \log$  oder wenigstens

$g \succeq \text{kN}(n)$ .

Wir können aber auch für jedes konstruierbare  $g$  unter log die Hierarchie zeigen:

**Satz 5.24 (Bandhierarchie two-way VI)**

Sei  $|\Sigma| = 1$  und  $f$  und  $g$  totale arithmetische Funktionen mit  $f \in o(g)$  und  $g \in o(\log)$ , unbeschränkt und 2DT-bandkonstruierbar. Dann gilt:

$$f \in o(g) \wedge g \in o(\log) \implies \text{weak-2DT}[f] \subsetneq \text{strong-2DT}[g]$$

**Beweis:** Wie wir später in Satz 6.26 zeigen werden, muß  $g$  unendlich oft über  $\log \log$  liegen. Wir könnten wie oben vorgeschlagen als Turingmaschine die mit dem Index  $\log \text{kN}(g(n))$  wählen, wenn wir nur glauben könnten, daß  $g$  surjektiv ist, d.h.  $g(n)$  jeden Wert aus  $\mathbb{N}$  annimmt. Da wir dies aber kaum voraussetzen können, müssen wir eine weitere Überlegung anstellen.

Betrachten wir die Stellen  $n$ , an denen  $g(n) \geq \log \log n$  ist. Mit einem Argument aus dem ebenfalls später geführten Beweis zu Satz 6.17 können wir schließen, daß es für jedes solche  $n$  ein erstes  $n'$  geben muß mit  $n' \leq n^n$  und  $g(n) < g(n')$ , da es sonst überhaupt kein  $m > n$  mehr gäbe, auf dem  $g$  einen größeren Wert annimmt als  $g(n)$ , was aber sein muß, da  $g$  unbeschränkt ist. Da  $g \in o(\log)$ , kann der Sprung von  $g(n)$  nach  $g(n')$  aber höchstens  $\log n' \leq \log n^n = n \log n$  sein. Um aus  $g$  eine surjektive unbeschränkte Funktion zu erhalten, drücken wir den Graphen von  $g$  so sehr zusammen, daß ein Sprung von  $n \log n$  zu 1 wird. Dazu verwenden wir die Funktion  $\alpha = \frac{1}{3} \log^*$ , denn

$$\log^* g(n') \leq \log^* g(n^n) \leq \log^*(n \log n) \leq \log^* 2^n \leq \log^*(\log \log n) + 3 \leq \log^* g(n) + 3$$

und somit  $\alpha g(n') \leq \alpha g(n) + 1$ . Bei Input  $n$  wird also die TM mit Index  $\log \text{kN}(\alpha g(n))$  gewählt. ■

## 6 Band unter log

### 6.1 Die Band-Gaps unter log

In diesem Abschnitt geht es darum, daß die Bandhierarchie nach unten begrenzt ist. Geht man bei den one-way-Turingmaschinen mit dem Bandaufwand unter log, (bei den two-way TMen unter  $\log \log$ ), so bricht die Hierarchie auf die Klasse REG der regulären Sprachen zusammen: es entsteht ein Loch, eine Lücke (engl. gap) in der Hierarchie, in der unterschiedlicher Aufwand nicht mehr zu unterschiedlichen Sprachklassen führt. Wir wollen folgendes Programm abhandeln:

1.  $\text{REG} \subsetneq \text{strong-1DT}[\log] \ (\subseteq \text{weak-1DT}[\log])$
2.  $\text{REG} = \text{weak-1DT}[o(\log)] \ (= \text{strong-1DT}[o(\log)])$
3.  $\text{REG} = \text{strong-1NT}[o(\log)]$
4.  $\text{REG} \subsetneq \text{weak-1NT}[\log \log]$
5.  $\text{REG} \subsetneq \text{strong-2DT}[\log \log] \ (\subseteq \text{weak-2DT}[\log \log] \subseteq \text{weak-2NT}[\log \log])$
6.  $\text{REG} = \text{weak-2DT}[o(\log \log)] \ (= \text{strong-2DT}[o(\log \log)])$
7.  $\text{REG} = \text{strong-2NT}[o(\log \log)]$
8.  $\text{REG} = \text{weak-2DT}[o(\log \log)]$   
und für das Kapitel Zeitkomplexität wollen wir uns vormerken:
9.  $\text{REG} = \text{strong-0NT}^1\langle o(\text{id} \cdot \log) \rangle$
10.  $\text{REG} \subsetneq \text{weak-0NT}^1\langle \text{id} \cdot \log \log \rangle$

### 6.1.1 One-way Turingmaschinen

#### Satz 6.1

Es gibt keine 1DT-Turingmaschine mit einem starken oder schwachen unbeschränkten Bandaufwand  $s$  der aus  $o(\log)$  ist ( $s \prec \log$ ).

**Beweis:** Sei  $M$  eine weak-1DT-TM mit einem unbeschränkten, schwachen Bandaufwand  $s \in o(\log)$ . O.B.d.A. lösche  $M$  nie, z.B. drucke  $\boxtimes$  statt  $\square$ . Für alle  $c \in \mathbb{N}$  gibt es ein  $n_0$ , so daß  $s(n) < \frac{1}{c} \log n$  für alle  $n > n_0$ . Sei

$$\text{MIN} = \{n > n_0 \mid s(n) \text{ definiert und } \forall m < n : (s(m) \text{ definiert} \implies s(m) < s(n))\}$$

d.h. die Menge der Stellen, an denen  $s$  zum ersten Mal einen größeren Wert annimmt. Sei  $w \in L(M)$  mit  $|w| = n \in \text{MIN}$  und  $s(w) = s(n)$ , d.h. ein kürzestes Wort aus  $L(M)$ , dessen akzeptierender Lauf  $R$  den Bandaufwand  $s(n)$  hat.  $R$  hat höchstens  $2^{c \cdot s} < n$  Konfigurationen für ein von  $M$  abhängiges  $c$ , also gibt es 2 Felder  $k_1, k_2$  in  $w$  ( $k_1 < k_2$ ), die in  $R$  mit derselben Konfiguration verlassen werden. Sei  $v = w(1) \cdots w(k_1)w(k_2 + 1) \cdots w(n)$ . Dann existiert auch für  $v$  ein akzeptierender Lauf  $R'$  mit demselben Platzbedarf  $s(n)$  im Widerspruch zur Wahl von  $w$  bzw.  $n$  [ $s(n - (k_2 - k_1)) = s(n)$ , d.h.  $n \notin \text{MIN}$ ]. ■

Man beachte, wie wichtig der Determinismus für dieses Argument ist: Hätten wir es mit nichtdeterministischen Maschinen zu tun, gäbe es also möglicherweise mehrere akzeptierende Läufe für  $v$ , so wäre es möglich, daß  $v$  einen billigeren akzeptierenden Lauf hätte und somit kein Widerspruch entstünde. Bei der strengen Aufwandsauffassung brauchen wir uns nicht auf die billigsten Läufe einzuschränken und können diesen Satz weiter unten auch für nichtdeterministische Maschinen beweisen. Bestehen wir aber auf der schwachen Aufwandsauffassung, so können nichtdeterministische Maschinen - wie wir noch zeigen werden - sogar noch mit einem Bandaufwand von  $\log \log$  etwas nichtreguläres erkennen, d.h. ihr Gap fällt mit dem der 2-way Maschinen zusammen. (Programmpunkt 4)

#### Korollar 6.2

$\text{weak-1DT}[o(\log)] = \text{weak-1DT}[\text{const}] = \text{REG}$ .

Aber schon ab Bandaufwand  $\log$  können nichtreguläre Sprachen erkannt werden:

#### Satz 6.3

$\text{strong-1DT}[\log] \neq \text{REG}$

**Beweis:**  $\{a^n b^n \mid n \in \mathbb{N}\}$ . ■

Nun zu dem eben angekündigten Satz:

#### Satz 6.4

Ist der Bandaufwand  $s$  einer strong-1NT-TM unbeschränkt, so ist  $s \notin o(\log)$ .

**Beweis:** Im Unterschied zum deterministischen Fall gibt es zu einem Wort  $w$  der Länge  $n$  eventuell mehrere  $w$ -Läufe, deren Aufwand aber wegen der strengen Aufwandsauffassung bei allen durch  $s(n)$  beschränkt ist. Sei  $n$  minimal mit  $s(n) = b$  für ein vorgegebenes  $b \in \text{Bild } s$  und  $w_s$  das Wort der Länge  $n$ , das einen  $w$ -Lauf  $R$  mit dem Bandaufwand  $b$  hat. Ist nun  $b < \log n$ , so können wir ein kürzeres Wort  $w'$  finden, das einen  $w'$ -Lauf mit eben diesem Bandaufwand  $b$  hat. Damit ist aber  $n$  nicht minimal im Widerspruch zu unserer Annahme. ■

#### Korollar 6.5

$\text{strong-1NT}[o(\log)] = \text{REG}$ .

### 6.1.2 Two-way Turingmaschinen

#### Satz 6.6

Es gibt keine weak-2DT-TM mit unbeschränktem Bandaufwand  $s \in o(\log \log)$ .

**Beweis:** Der Beweis geht völlig analog zum 1-way-Fall, nur betrachten wir Crossingmengen anstelle von Konfigurationen. ■

#### Korollar 6.7

weak-2DT[ $o(\log \log)$ ] = REG.

Ab  $\log \log$  aber werden auch nichtreguläre Sprachen erkannt:

#### Satz 6.8

strong-2DT[ $\log \log$ ]  $\neq$  REG.

**Beweis:** Die Sprache  $\text{BIN} = \{ \overleftarrow{\text{bin}}(0) \# \cdots \# \overleftarrow{\text{bin}}(n) \mid n \in \mathbb{N} \}$ , wo  $\overleftarrow{\text{bin}}(i)$  die gespiegelte Binärdarstellung von  $i$  sei, ist nicht regulär, läßt sich aber auf Band  $\log \log$  erkennen. ■

Wie im 1-way-Fall kann man bei der strengen Aufwandsauffassung das Gap auch für nichtdeterministische Maschinen beweisen:

#### Satz 6.9

Für jede strong-2NT-TM gilt: ist ihr Bandaufwand  $s \in o(\log \log)$ , so ist er schon beschränkt.

**Beweis:** Aufgabe. ■

#### Korollar 6.10

strong-2NT[ $o(\log \log)$ ] = REG.

Auch hier erhebt sich die Frage, wie das Gap aussieht, wenn man schwachbeschränkte nichtdeterministische Maschinen betrachtet. Dafür bedienen wir uns einer neuen Methode: wir zählen Transitionsmatrizen, die sich im Gegensatz zu Crossingmengen nicht auf einen einzelnen Lauf beziehen, sondern alle Läufe eines gewissen Aufwands abstrahieren, die sich auf einem Inputabschnitt abspielen.

### 6.1.3 Methode Transitionsmatrix

Die Transitionsmatrizen beschreiben, in welcher Konfiguration eine Turingmaschine einen Input verläßt, den es in einer bestimmten Konfiguration betreten hat.

#### Definition 6.11 (Transitionsmatrix)

Gegeben sei eine 2NT-TM  $M$ ,  $s \in \mathbb{N}$ . Ein Lauf  $R$  mit Bandaufwand  $s$  von  $M$  hat  $z \leq 2^{c \cdot s}$  Konfigurationen  $\{k_1, \dots, k_z\}$  für eine  $c \in \mathbb{N}$ . Wir definieren eine Boolesche  $z \times z$ -Matrix  $A_v^s$  vermöge:

$A_M^s(v)(i, j) = 1$  gdw. es einen  $v$ -Lauf  $R$  von  $M$  mit einem Aufwand  $\leq s$  gibt, der in der Konfiguration  $k_i$  auf dem ersten Symbol von  $v$  startet und in der Konfiguration  $k_j$  auf dem Feld links neben  $v$  endet.

Diese Matrix nennen wir *Transitionsmatrix* des Wortes  $v$  bei Bandaufwand  $s$ . Die Zahl der Transitionsmatrizen  $A_M^s(v)$  ( $v \in \Sigma^*$ ) ist kleinergleich  $2^{z^2} \leq 2^{2^{c \cdot s}}$ .

#### Lemma 6.12

Gegeben sei eine 2NT-TM  $M$ . Sei  $uv \in L(M)$ , sei  $R$  ein akzeptierender  $uv$ -Lauf mit einem Bandaufwand  $\leq s$ , der auf Feld 0 stoppt, und sei  $A_M^s(v) = A_M^s(x)$ , für ein  $x$ . Dann existiert auch für  $ux$  ein akzeptierender Lauf mit einem Bandaufwand  $\leq s$ .

**Lemma 6.13**

Gegeben sei eine 2NT-TM  $M$ . Sei  $uvw \in L(M)$ , sei  $R$  ein akzeptierender  $uvw$ -Lauf mit einem Bandaufwand  $\leq s$ , der auf Feld 0 stoppt, und sei  $A_M^s(vw) = A_M^s(w)$ . Dann existiert auch für  $uw$  ein akzeptierender Lauf mit einem Bandaufwand  $\leq s$ .

Weder das Zählen von Konfigurationen, noch das von Crossingmengen hat ausgereicht, das Gap unter loglog auch für nichtdeterministische schwach beschränkte Maschinen zu beweisen. Mit den Transitionsmatrizen aber haben wir eine Methode, mit der uns das gelingt. Das liegt daran, daß sich Transitionsmatrizen im Gegensatz zu Crossingmengen nicht Eigenschaften einzelner Läufe sind, sondern Eigenschaften von Inputworten, die das Ergebnis aller möglichen Läufe auf dem Wort beschreiben, die sich an einen bestimmten Höchstaufwand halten.

**Satz 6.14**

Für jede weak-2NT-TM gilt: ist ihr Bandaufwand  $s \in o(\log \log)$ , so ist er schon beschränkt, d.h. durch eine Konstante majorisierbar.

**Beweis:** Sei  $M$  eine 2NT-TM mit einem schwachen Bandaufwand  $s \in o(\log \log)$  und  $s$  unbeschränkt. O.B.d.A. lösche  $M$  nie und stoppe nur, wenn der Inputkopf auf Feld 0 steht. Die Zahl der Konfigurationen eines Laufs mit Bandaufwand  $s$  ist  $\leq 2^{c \cdot s}$  für ein  $c \in \mathbb{N}$ . Da  $s \in o(\log \log)$ , ist  $s(n) < \frac{1}{2c} \log \log n$  für genügend große  $n$ . Sei  $s = s(n)$  und  $w_s$  ein kürzestes Wort ( $|w_s| = n$ ) aus  $L(M)$ , dessen billigster akzeptierender Lauf  $R$  den Bandaufwand  $s(n)$  hat. Die Zahl der Transitionsmatrizen  $A_M^s(v)$  ( $v \in \Sigma^*$ , Bandaufwand  $s$ ) ist höchstens

$$2^{2^{c \cdot s}} < 2^{2^{\log \log n}} \leq n.$$

Also gibt es 2 Positionen in  $w_s$  mit derselben Transitionsmatrix, d.h. es gibt Worte  $x, y, z$ , sodaß  $w_s = xyz$  mit  $A_M^s(yz) = A_M^s(z)$ . Dann gibt es aber auch für  $xz$  einen akzeptierenden Lauf  $R'$  mit demselben Platzbedarf  $s$ . Dieser Lauf  $R'$  ist auch der billigste Lauf für  $xz$ , denn gäbe es einen billigeren Lauf für  $xz$ , so müßte es auch einen billigeren für  $w_s = xyz$  geben. Das steht aber im Widerspruch zur Wahl von  $w_s$ . ■

**Korollar 6.15**

weak-2NT[ $o(\log \log)$ ] = REG.

Folgende Aufgabe zeigt, daß man sich manchmal auf die Intuition verlassen kann. Die Sprache der Palindrome kann nur erkannt werden, wenn man wenigstens  $\log$  Platz zu Verfügung hat, selbst dann, wenn die TM nichtdeterministisch und nur schwach beschränkt ist.

**Aufgabe 6.16**

$L = \{w\#\overline{w} \mid w \in \{0, 1\}^*\} \notin \text{weak-2NT}[f]$ , falls  $\inf \frac{f}{\log} = 0$  (falls  $f$  also unendlich oft unter log driftet).

**Beweis:** Idee: betrachte Transitionsmatrizen. ■

Ein klassisches Argument von Juris Hartmanis wird in dem folgenden Satz verwandt, den wir zur Behandlung unserer Programmpunkte brauchen. Wir wollen dieses Argument darum auch künftig das *Hartmanis-Argument* nennen.

**Satz 6.17**

$L = \{a^n b^n \mid n \in \mathbb{N}\} \notin \text{weak-2NT}[f]$ , falls  $\inf \frac{f}{\log} = 0$  (falls  $f$  also unendlich oft unter log driftet).

**Beweis:** Sei  $M$  eine weak-2NT[ $f$ ]-TM, die auf einem unären Inputblock  $b^n$  arbeitet. Wir nennen ein Stück des  $b^n$ -Laufes, der auf dem ersten Feld von  $b^n$  beginnt und hinter dem letzten von  $b^n$  endet oder umgekehrt auf dem letzten Feld von  $b^n$  beginnt und vor dem ersten Feld von  $b^n$  endet einen Sweep, wenn die Maschine zwischendurch den Inputbereich  $b^n$  nicht verläßt. Bei einem gegebenen Sweep betrachten wir für jedes Feld in  $b^n$  die Konfiguration, in der  $M$  dieses Feld zum letzten Mal verläßt.

Hat  $M$  weniger als  $n$  Konfigurationen, so muß diese Konfiguration beim letzten Verlassen für zwei Felder die gleiche sein. Damit könnte aber  $M$  auf dem zweiten Feld so weiterlaufen, wie auf dem ersten und die Distanz  $d$  zwischen diesen beiden Feldern noch einmal zurücklegen, um wieder in dieser Konfiguration ein drittes Feld zum letzten Mal zu verlassen. Würden wir den Block  $b^n$  um  $d$  Felder verlängern, so gäbe es immer noch einen Sweep über diesen Block, der mit derselben Konfiguration anfängt und mit derselben endet, d.h. das äußere Verhalten dieses Sweeps wäre für beide Blöcke dasselbe. Diese Distanz  $d$  kann aber auch mit demselben Effekt mehrmals, ja beliebig oft eingefügt werden. Wir wollen sie so oft einfügen, daß wir den Block  $b^n$  insgesamt um  $n!$  verlängert haben.

Diese Betrachtung können wir für jeden Sweep durchführen, und erhalten, daß  $M$  auf  $b^{n+n!}$  alle seine Sweeps, die  $M$  auf  $b^n$  ausführte in veränderter Form auch auf  $b^{n+n!}$  ausführen kann, ohne beim Austritt aus dem Block eine andere Konfiguration einzunehmen. Für die Laufteile auf dem Block, die keinen Sweep darstellen (etwa:  $M$  betritt den Block von links und verläßt ihn beim nächsten Mal auch wieder links), ist eine Verlängerung des Blocks unbedeutend, da sie das andere Ende des Blocks nicht berühren und somit auch nicht erkennen. Verlangen wir, daß  $M$  immer nur außerhalb des Blockes stoppt, wird  $M$  jedes Inputwort, das diesen Block  $b^n$  enthält, genauso behandeln (akzeptieren oder verwerfen), wenn der Block auf  $b^{n+n!}$  verlängert wird.

Das gilt natürlich auch für das Wort  $a^n b^n$  und seine verlängerte Variante  $a^n b^{n+n!}$ . O.B.d.A. stoppe  $M$  nur auf Feld 1.  $M$  hat höchstens  $2^{cf(n)}$  Konfigurationen für ein  $c \in \mathbb{N}$ . Wählen wir  $n$  so daß  $2cf(n) < \log n$ , dann ist die Zahl der Konfigurationen höchstens  $\sqrt{n} < \frac{1}{2}n$ . Dann erkennt  $M$  mit  $a^n b^n$  auch  $a^n b^{n+n!}$  - ein Wort, das nicht zu  $L$  gehört. ■

#### 6.1.4 Primzahlen

Um unsere Programmpunkte 4 und 10 zu zeigen, müssen wir etwas ausholen.

##### Notation 6.18

Wir bezeichnen mit  $\pi(n)$  die Anzahl der Primzahlen unter  $n$  und mit  $p_n$  die  $n$ -te Primzahl angefangen bei 2 ( $p_1 = 2$ ).

Ohne Beweis zitieren wir den berühmten Primzahlsatz der Zahlentheorie, den erst 1896 der Franzose Jaques-Salomon Hadamard (1865-1963) und unabhängig davon der Belgier Charles de la Vallée-Poussin (1866-1962) bewiesen haben. (Selbst der berühmte Euler (1707-1783) scheiterte an diesem Problem. Tschebyschew (1821-1894) fand immerhin das dahinter stehende Korollar. Entsprechend der Legende, daß der Bezwinger dieses Problems unsterblich würde, sind Hadamard und Vallée-Poussin, beide, fast 100 Jahr alt geworden.)

##### Satz 6.19 (Primzahlsatz)

$$\lim \pi(n) \cdot \frac{\ln(n)}{n} = 1$$

( $\ln$  ist der natürliche Logarithmus zur Basis  $e$ ).

##### Korollar 6.20

Für genügend große  $n$  gilt:  $1,4 \cdot \frac{n}{\log n} < \pi(n) < 1,5 \cdot \frac{n}{\log n}$ .

**Beweis:** Nach obigem Primzahlsatz gilt für genügende große  $n$ :

$$\begin{aligned} (1 - \varepsilon) \frac{n}{\ln n} &< \pi(n) < (1 + \varepsilon) \frac{n}{\ln n} \\ (1 - \varepsilon) \log e \frac{n}{\log n} &< \pi(n) < (1 + \varepsilon) \log e \frac{n}{\log n} \\ 1,4 \cdot \frac{n}{\log n} &< \pi(n) < 1,5 \cdot \frac{n}{\log n} \quad (1,4 < \log e < 1,5) \end{aligned}$$

**Korollar 6.21**

Für genügend große  $n$  gilt:  $\frac{1}{2}n \log n < p_n < 2n \cdot \log n$ .

**Beweis:**

$$\frac{1}{2}n \log n < \frac{1}{2}n \cdot \log p_n = \frac{1}{2}\pi(p_n) \cdot \log p_n < p_n.$$

$$\pi(2n \log n) > \frac{2n \log n}{\log n + \log \log n + \log 2} > n, \text{ und daher } p_n < 2n \log n.$$

**Korollar 6.22**

Zwischen 2 Zweierpotenzen liegt fast immer eine Primzahl.

**Beweis:** Für genügende große  $n$  gibt es bis  $2^n$  höchstens  $1,5 \cdot \frac{2^n}{n}$  Primzahlen, bis  $2^{n+1}$ , aber mindestens  $\frac{2^{n+1}}{n+1}$ , d.h. zwischen  $2^n$  und  $2^{n+1}$  liegen Primzahlen, da  $1,5(n+1) \cdot 2^n < 2 \cdot n \cdot 2^n$ .

**Satz 6.23 (Chinesischer Restsatz)**

Sind  $q_1, \dots, q_k$  teilerfremde Zahlen aus  $\mathbb{N}$ , so ist jedes  $n$ , das kleiner ist als das Produkt  $\prod_{i=1}^k q_i$ , eindeutig festgelegt durch seine Reste  $r_i \pmod{q_i}$ .

**Beweis:** Hätten zwei verschiedene Zahlen  $n, m \leq \prod_{i=1}^k q_i$  immer dieselben Reste, so gälte:

$$\forall i : |n - m| \equiv 0 \pmod{q_i}, \text{ d.h. } \forall i : q_i \mid |n - m|, \text{ d.h. } \prod_{i=1}^k q_i \mid |n - m|. \text{ W!}$$

Wir sagen "a teilt b", i.Z.:  $a \mid b$ , wenn  $\exists c > 0 : ac = b$ .

**Korollar 6.24**

$n = m$  gdw.  $n \equiv m \pmod{p_i}$  für alle Primzahlen  $p_i \leq 8 \log(n + m)$ .

**Beweis:** Wir zeigen: wenn  $k = \frac{4 \log n}{\log \log n}$ , so ist  $n < \prod_{i=1}^k p_i$ .

$$\prod_{i=1}^k p_i > k! > \left(\frac{k}{2}\right)^{\binom{k}{2}} = 2^{\frac{k}{2} \log \frac{k}{2}} > 2^{\log n} = n,$$

da

$$\frac{k}{2} \cdot \log \frac{k}{2} = \frac{2 \log n}{\log \log n} \cdot (\log 2 + \log \log n - \log \log \log n) > \frac{2 \log n}{\log \log n} \cdot \frac{1}{2} \log \log n = \log n.$$

Weiter ist  $p_k < 2k \log k$  (Korollar 6.20). Also

$$p_k < \frac{8 \log n}{\log \log n} (\log 4 + \log \log n - \log \log \log n) < 8 \log n.$$

**Satz 6.25**

Sei  $L = \{a^n b^m \mid n \neq m\}$ .

1.  $L \in \text{weak-1NT}[\log \log]$ ,
2.  $L \in \text{weak-2DT}[\log \log]$ ,
3.  $L \in \text{weak-0NT}^1\langle n \cdot \log \log \rangle$ .

**Beweis:** Es geht darum, eine Zahl  $p$  zu finden, so daß das Teilen durch  $p$  bei  $n$  und  $m$  zu verschiedenen Resten führt. Dann ist klar:  $n \neq m$ . Wir wissen, wenn  $n \neq m$ , so gibt es eine solche Zahl  $p$  schon unter  $8 \log n$ , die binär auf Band  $\log \log n$  Platz findet, andernfalls gibt es eine solche Zahl überhaupt nicht.

In Fall 1 rät die TM binär ein  $p$  auf das Arbeitsband, und teilt erst  $n$  und dann  $m$ , indem es  $p$  in jedem Inputschritt eins herunterzählt bis auf Null und dann wieder von vorn beginnt.

Im Fall 2 probiert die TM jedes  $p \in \mathbb{N}$  durch  $(2, 3, 4, \dots)$  bis es eines findet, bei dem die Reste ungleich sind. Sie stoppt genau dann, wenn  $n \neq m$ .

In Fall 3 arbeitet die Maschine wie in Fall 1, nur wird das  $p$  auf das eine Arbeitsband geschrieben (in eine 2. Spur) und während der Arbeitskopf über den Input läuft in Spur 2 mitgezogen, was für jeden Schritt des Kopfes auf dem Input zusätzlich  $O(\log \log)$  Schritte kostet.

In allen Fällen sind die Maschinen nur schwach beschränkt: ist  $n = m$ , so gibt es Läufe, die jeden Aufwand überschreiten, ist aber  $n \neq m$ , so gibt es immer einen akzeptierenden Lauf, der mit dem Aufwand auskommt. ■

## 6.2 Bandkonstruierbarkeit unter log

Wir stehen jetzt in einem gewissen Dilemma: Satz 5.15 erzählt uns von einer Hierarchie, die durch keine Einschränkung nach unten begrenzt wird. Andererseits haben wir gerade bewiesen, daß unterhalb von  $\log \log$  ein Gap existiert, d.h. die Hierarchie endet bei  $\log \log$ . Es kann dafür nur einen Grund geben: die einzige Bedingung an  $g$  ist nicht mehr erfüllt, die Bedingung der Konstruierbarkeit. Wir müssen daraus schließen, daß es unterhalb von  $\log \log$  keine konstruierbaren Funktionen mehr gibt. Das führt uns zu der Frage, ob wir die konstruierbaren in diesem unteren Bereich - sagen wir unterhalb von  $\log$  - charakterisieren können. Dies versuchen wir mit dem folgenden Satz:

### Satz 6.26 (Charakterisierung der 2DT-bandkonstruierbaren Funktionen unter log)

1. Eine arithmetische Funktion  $f$  mit  $\inf \frac{f}{\log} = 0$  und  $\inf \frac{f}{\log \log} > 0$  kann nicht 2DT-bandkonstruierbar sein. (Eine Funktion die an unendlich vielen Stellen unter  $\log$  driftet, aber fast nie unter  $\log \log$ , kann nicht konstruierbar sein)
2. Für eine arithmetische Funktion  $f$  mit  $\inf \frac{f}{\log} = 0$ , die 2DT-bandkonstruierbar ist, gilt:
  1.  $\inf f < \infty$  ( $f$  muß an unendlich vielen Stellen unter einem konstanten Wert liegen)
  2.  $\inf \frac{\log \log}{f} < \infty$  ( $f$  muß an unendlich vielen Stellen über einem festen Bruchteil von  $\log \log$  liegen)

D.h. diese Funktion  $f$  muß laufend zwischen einer Konstanten und  $\varepsilon \log \log$  oszillieren für ein  $\varepsilon > 0$ .

### Beweis:

1. Wenn  $L_{=} = \{a^n b^n \mid n \in \mathbb{N}\} \in \text{weak-2NT}[f]$  so muß gelten:  $\inf \frac{f}{\log} > 0$  (vgl. Satz 6.17) (d.h.  $f$  driftet nicht unendlich oft unter  $\log \log$ ).  $L_{\neq} = \{a^n b^n \mid m \neq n; m, n \in \mathbb{N}\} \in \text{weak-2DT}[\log \log]$  (vgl. Satz 6.25). Aus dem Beweis von Satz 6.25 geht aber auch hervor, daß  $L_{\neq}$  auch von einer stark beschränkten Maschine erkannt würde, gäbe es ein Abbruchkriterium. Dieses läge vor, wäre die Aufwandsfunktion konstruierbar. D.h.  $L_{\neq} \in \text{strong-2DT}[f]$  für jedes 2DT-bandkonstruierbare  $f$  mit  $\inf \frac{f}{\log \log} > 0$  (fast immer über  $\varepsilon \log \log$  für ein  $\varepsilon > 0$ ). Eine stark beschränkte Maschine entscheidet aber ihre Sprache, d.h. sie akzeptiert und verwirft in endlicher Zeit und demselben Aufwand  $f$ . Diese Maschine würde also auch leicht umgebaut werden können zu einer  $\text{strong-2DT}[f]$ -TM, die  $L_{=}$  erkennt. Aus Satz 6.17 müssen wir also schließen, daß für  $f$  dann auch  $\inf \frac{f}{\log} \neq 0$  gilt.

1. Sei  $f$  eine unbeschränkte, 2DT-bandkonstruierbare Funktion mit  $\inf \frac{f}{\log} = 0$  (d.h.  $f \asymp^\infty \log$ ). Dann gibt es eine 2DT[ $f$ ]-TM  $M$ , die auf Input  $1^n$  mit Output  $1^{f(n)}$  stoppt ( $M(1^n) \downarrow 1^{f(n)}$ ). Nach dem Hartmanisargument aus dem Beweis von Satz 6.17 muß dann auch für unendlich viele genügend große  $n$  gelten:  $M(1^{n+in!}) \downarrow 1^{f(n)}$ , d.h. für diese  $n$  ist  $f(n+in!) = f(n)$  für alle  $i \in \mathbb{N}$ . Wir haben also unendlich viele Konstanten  $f(n)$  und auf jede dieser Konstanten muß die Funktion  $f$  unendlich oft zurückfallen.
2. Sei  $f$  eine unbeschränkte 2DT-bandkonstruierbare Funktion mit  $\inf \frac{f}{\log} = 0$  (d.h.  $f \asymp^\infty \log$ ). Damit ist  $f$  Aufwandsfunktion einer 2DT-Turingmaschine, die eine unendliche Sprache erkennt (z.B. der, die erst  $f$  konstruiert und dann akzeptierend stoppt). Nach Satz 6.14 darf dann  $f$  nicht aus  $o(\log \log)$  sein, d.h.  $\inf \frac{\log \log}{f} < \infty$  ( $f \asymp^\infty \log \log$ ) (an unendlich vielen Stellen muß der Aufwand größer als  $\varepsilon \log \log$  sein für ein  $\varepsilon > 0$ ).

■

### 6.2.1 Größter kleinster Nichtteiler (gkN)

Es erhebt sich natürlich die Frage, ob es unter den Bedingungen 2.1 und 2.2 des Korollars 6.20 überhaupt eine unbeschränkte 2DT-bandkonstruierbare Funktion mit  $\inf \frac{f}{\log} = 0$  ( $f \asymp^\infty \log$ ) gibt. Darum wollen wir hier eine angeben:

#### Notation 6.27

Sind  $n, m \in \mathbb{N}$ , so sagen wir:  $n$  teilt  $m$  (in Zeichen:  $n \mid m$ ), wenn  $m$  ein positives Vielfaches von  $n$  ist. Andernfalls sagen wir:  $n$  teilt  $m$  nicht (in Zeichen:  $n \nmid m$ ). Mit  $\text{kN}$  bezeichnen wir den kleinsten Nichtteiler von  $n$  und mit  $\text{gkN}$  den größten der kleinsten Nichtteiler der natürlichen Zahlen von 1 bis  $n$ . Damit ist die Funktion  $\text{gkN}$  eine Glättung der Funktion  $\text{kN}$ : die kleinste,  $\text{kN}$  majorisierende Treppenfunktion. Formal:

$$\begin{aligned} \text{kN}(n) &:= \min\{t \in \mathbb{N} : t \nmid n, t > 1\} \\ \text{gkN}(n) &:= \max\{\text{kN}(\nu) : \nu \in \{3, \dots, n\}\}. \end{aligned}$$

Wir nennen  $n$  eine Sprungstelle von  $\text{gkN}$ , wenn  $\text{gkN}(n) > \text{gkN}(n-1)$ .

#### Notation 6.28

Seien  $\{p_i : i \in \mathbb{N}\}$  die Primzahlen in ihrer natürlichen Ordnung. Jede natürliche Zahl kann auf eindeutige Weise dargestellt werden als ein Produkt ihrer Primteiler:

$$n = \prod_{i=1}^{\infty} \{p_i^{\nu_i} : p_i^{\nu_i} \mid n \wedge p_i^{\nu_i+1} \nmid n\}$$

oder

$$n = p_1^{\nu_1} \cdots p_k^{\nu_k} \cdots, \text{ wobei } \nu_i = \max\{j \in \mathbb{N} : p_i^j \mid n\}.$$

Dieses Produkt von Primzahlpotenzen nennen wir die *Primzahlpotenzdarstellung (PPD)*. Schreiben wir nur den endlichen Anfang der PPD: das Produkt  $p_1^{\nu_1} \cdots p_k^{\nu_k}$ , wo  $p_k$  die größte Primzahl ist, die  $n$  teilt ( $k = \max\{i : \nu_i \neq 0\}$ ), so reden wir von der *Standard-Primzahlpotenzdarstellung (SPPD)*.

Damit wird also jede natürlich Zahl auf eindeutige Weise repräsentiert von dem unendlichen Tupel  $(\nu_1, \dots, \nu_i, \dots)$  der Primzahlexponenten  $\nu_i = \max\{j \in \mathbb{N} : p_i^j \mid n\}$  bzw. dem endlichen Anfang dieses Tupels  $(\nu_1, \dots, \nu_k)$ , wo  $k = \max\{i : \nu_i \neq 0\}$ . In der PPD von  $n$  werden alle Primteiler von  $n$  aufgezählt und alle Teiler von  $n$  sind Produkte dieser Primteiler. Ein Nichtteiler von  $n$  muß also in seiner PPD eine Primzahl wenigstens einmal öfter enthalten, als die PPD von  $n$ . Damit ist der kleinste Nichtteiler von  $n$  offenbar gerade  $\text{kN}(n) = \min\{p_i^{\nu_i+1} : i \in \mathbb{N}\}$ .

**Definition 6.29 (ausgewogen)**

Wir nennen eine PPD *ausgewogen*, wenn gilt: erhöht man den Exponenten einer Primzahlpotenz in der PPD um 1, so wird diese Primzahlpotenz größer als jede andere Primzahlpotenz in der PPD:  $\forall i, j : p_i^{\nu_i+1} > p_j^{\nu_j}$ .

**Bemerkung:** Man kann sich die Primzahlpotenzen in einer PPD vorstellen wie Türme. Erhöhen wir die Potenz um 1, so stocken wir den Turm auf. In einer ausgewogenen PPD, überragt ein aufgestockter Turm jeden anderen Turm der PPD. Für die SPPD  $p_1^{\nu_1} \cdots p_k^{\nu_k}$  gilt außerdem: alle Potenzen  $\nu_1, \dots, \nu_k$  sind ungleich 0, d.h. jede der ersten  $k$  Primzahlen teilt  $n$ .

**Lemma 6.30 (Sprungstelle)**

$n$  ist eine Sprungstelle von gkN gdw. die PPD von  $n$  ausgewogen ist.

**Beweis:** Angenommen,  $n$  ist eine Sprungstelle, d.h.  $\text{gkN}(n) > \text{gkN}(n-1)$ . Wäre die PPD von  $n$  nicht ausgewogen, dann gäbe es  $i$  und  $j$  mit  $p_i^{\nu_i+1} \leq p_j^{\nu_j}$ . Also kann  $\text{kN}(n) = \min\{p_i^{\nu_i+1} : i \in \mathbb{N}\}$  nicht  $p_j^{\nu_j+1}$  sein. Dann aber hat  $m = n/p_j$  denselben kleinsten Nichtteiler ( $\text{kN}(m) = \text{kN}(n)$ ), da  $n$  und  $m$  in allen anderen Primzahlpotenzen übereinstimmen.

Angenommen die PPD von  $n$  ist ausgewogen. Ist  $m < n$ , so gibt es eine Primzahlpotenz  $p_i^{\nu_i}$  in der PPD von  $n$ , die  $m$  nicht teilt ( $p_i^{\nu_i} \nmid m$ ), d.h.  $\text{kN}(m) \leq p_i^{\nu_i} < p_j^{\nu_j+1}$  für alle  $j$  (ausgewogen), damit ist  $\text{kN}(m) < \min\{p_i^{\nu_i+1} \mid i \in \mathbb{N}\} = \text{kN}(n)$ . ■

**Lemma 6.31 (Wachstum von kN)**

Für alle großen Sprungstellen  $n$  gilt:  $\frac{1}{2} \log n \leq \text{kN}(n) \leq 8 \log n$ .

**Beweis:** Sei  $n = p_1^{\nu_1} \cdots p_k^{\nu_k}$  (SPPD) eine Sprungstelle (die PPD also ausgewogen) und  $r$  der Wert der maximalen Primzahlpotenz von  $n$ . Dann ist

1.  $k = \pi(r)$  [die ersten  $k$  Primzahlen liegen wegen der Ausgewogenheit unter  $r$ , da sie alle  $n$  teilen. Alle späteren Primzahlen  $p_j$  müssen über  $r$  liegen, da ihre Potenz in der PPD 0 ist, also  $p_j^1$  (der aufgestockte Turm) über jeder Primzahlpotenz der PPD, also auch über  $r$  liegen muß] und damit  $\frac{r}{\log r} \leq k \leq 2 \frac{r}{\log r}$  (Korollar 6.20). Nun ist
2.  $k! \leq n \leq r^k$  [die ersten  $k$  Primzahlen teilen  $n$  wegen der Ausgewogenheit], d.h.  $\log k! \leq \log n \leq k \cdot \log r$  und
3.  $k! \geq \left(\frac{k}{2}\right)^{\frac{k}{2}} = 2^{\frac{k}{2} \cdot \log \frac{k}{2}}$ , d.h.  $\log k! \geq \frac{k}{2} \cdot \log \frac{k}{2} \geq \frac{r}{2 \log r} (\log r - \log \log r - 1) \geq \frac{r}{4}$  (wegen 1.) Und weiter wegen 1.
4.  $r^k \leq r^{2 \frac{r}{\log r}}$ , d.h.  $k \cdot \log r \leq 2r$ . Hieraus folgt
5.  $\frac{r}{4} \stackrel{(3)}{\leq} \log k! \stackrel{(2)}{\leq} \log n \stackrel{(2)}{\leq} k \log r \stackrel{(4)}{\leq} 2r$  und somit
6.  $\frac{1}{2} \log n \stackrel{(5)}{\leq} r \stackrel{(a)}{\leq} \text{kN}(n) \leq 2^{\nu_1+1} \leq 2r \stackrel{(5)}{\leq} 8 \log n$  ((a) = ausgewogen).

**Lemma 6.32 (Wachstum von gkN)**

$\frac{1}{4} \log n \leq \text{gkN}(n) \leq 8 \log n$  für alle  $n \in \mathbb{N}$ .

**Beweis:** Sei  $n = p_1^{\nu_1} \cdots p_k^{\nu_k}$  Sprungstelle (die PPD also ausgewogen) und  $\text{kN}(n) = p_i^{\nu_i+1}$ . Wegen der Ausgewogenheit ist  $\text{kN}(n) > p_j^{\nu_j}$  für alle  $j \in \mathbb{N}$ . Damit ist die nächste Sprungstelle (nächste ausgewogene Stelle)  $n' = np_i$ .

- Sei  $i > 1$ , d.h.  $p_i \neq 2$  : Da  $2^{\nu_1} < p_i^{\nu_i+1} \implies 2 \cdot 2^{\nu_1} < p_i p_i^{\nu_i+1}$ , d.h.  $p_i^{\nu_i+2}$  kann nicht  $\text{kN}(n')$  sein. Sei  $\text{kN}(n') = p_j^{\nu_j+1}$  ( $j \neq i$ ), so ist  $\text{kN}(n') = p_j^{\nu_j+1} \leq 2^{\nu_1+1} \leq 2 \cdot p_i^{\nu_i+1} \leq 2 \cdot \text{kN}(n)$ .
- Sei  $i = 1$ , d.h.  $p_i = 2$  : Dann ist  $n' = 2n$  und  $\text{kN}(n') \leq 2^{\nu_1+1} \leq 2p_i^{\nu_i+1} = 2\text{kN}(n)$ .

Beidesmal also springt der Wert von  $\text{gkN}$  an jeder Sprungstelle auf höchstens das Doppelte des vorigen Wertes. Da nach dem Lemma 6.31 an den Sprungstellen  $\text{gkN}(n)$  zwischen  $\frac{1}{2} \log n$  und  $8 \log n$  liegt und  $\text{gkN}(n) = \text{gkN}(\nu)$  für alle Argumente  $\nu$  zwischen den Sprungstellen  $n$  und  $n'$ , gilt:

$$8 \log \nu \geq 8 \log n \geq \text{gkN}(n) = \text{gkN}(\nu) \geq 1/2 \text{gkN}(n') \geq 1/4 \log n' \geq 1/4 \log \nu.$$

■

### Korollar 6.33

1. Die Funktion  $\log \text{kN}$  ist 2DT-bandkonstruierbar.
2. Es ist  $\inf \log \text{kN} = 2$  und  $\inf(\log \log / \log \text{kN}) = 1$ .
3.  $\text{gkN} \asymp \log \log$  (d.h.  $\text{gkN} \in O(\log \log)$  und  $\log \log \in O(\text{gkN})$ ). Damit kann sie nach Satz 6.26 nicht 2DT-bandkonstruierbar sein.

**Beweis:** ad 2.: Für alle ungeraden Stellen  $n$  ist  $\text{kN}(n) = 2$ . An den Sprungstellen ist  $\log \log(n) - 2 \leq \log \text{kN}(n) \leq \log \log(n) + 3$ , also

$$1 - \frac{2}{\log \log n} \leq \frac{\log \text{kN}(n)}{\log \log n} \leq 1 + \frac{3}{\log \log n}.$$

Der verführerische Gedanke, die Turingmaschine für  $\text{kN}$  umzubauen in eine für  $\text{gkN}$ , scheitert daran, daß sie auf dem Inputband keine Marke setzen kann. Für Turingmaschinen, die auf dem Inputband eine einzige Marke verschieben dürfen, wäre  $\text{gkN}$  tatsächlich konstruierbar. ■

### Notation 6.34

Noch eine Bemerkung über die Sprungstellen von  $\text{gkN}$  und die Funktion  $n!$ :

Sind  $n_1, \dots, n_k$  natürliche Zahlen, so wollen wir  $n_1 \cdots n_k$  einen Produktausdruck über  $\mathbb{N}$  nennen und hierfür folgende Operation  $R(n_1 \cdots n_k)$  erklären: Wir streichen jedes  $n_i$ , dessen PPD mehr als eine Primzahlpotenz enthält und dann noch die, die ein anderes übriggebliebenes  $n_j$  teilen, die also eine Potenz derselben Primzahl darstellen, aber eine kleinere. Es bleiben die Faktoren  $n_i$ , die größte Primzahlpotenzen waren.  $R(n_1 \cdots n_k)$  sei der Wert des so reduzierten Produktes. Wir wollen den Produktausdruck  $1 \cdots n$  mit  $n!$  abkürzen, wohl wissend, daß zwischen Ausdruck und Wert des Ausdrucks zu unterscheiden ist und daß hier wieder eine gewollte Verwechslung nahegelegt wird. Dann gilt:

### Lemma 6.35

Die Menge der Sprungstellen von  $\text{gkN}$  ist gerade  $\{R(n!) : n \in \mathbb{N}\}$ .

### Beweis:

1.  $R(n!)$  hat eine ausgewogene PPD. Denn ist  $p_i^{\nu_i}$  eine Primzahlpotenz, so ist  $p_i^{\nu_i+1} > n$ , da sonst  $p_i^{\nu_i+1}$  Faktor von  $n!$  wäre und anstelle von  $p_i^{\nu_i}$  übriggeblieben wäre. Damit ist  $p_i^{\nu_i+1}$  auch größer als alle Primzahlpotenzen von  $R(n!)$ .
2. Ist  $n$  eine Zahl mit ausgewogener PPD und  $p_i^{\nu_i}$  die größte Primzahlpotenz, dann ist  $n = R(p_i^{\nu_i}!)$ . Denn alle Primzahlpotenzen  $p_j^{\nu_j}$  der PPD von  $n$  sind Faktoren von  $p_i^{\nu_i}!$ , aber wegen der Ausgewogenheit keine größeren ( $p_j^{\nu_j+1} > p_i^{\nu_i}$ ). D.h. die Primzahlpotenzen der PPD von  $n$  sind gerade die größten Primzahlpotenzen, die Faktoren von  $p_i^{\nu_i}!$  sind. ■

An dieser Stelle können wir eine unäre nichtreguläre Sprache  $L \in \text{strong-1DT}[\log \log]$  angeben.

### Satz 6.36

Sei  $L = \{n \mid \exists k : p_1, \dots, p_k \text{ teilen } n, p_1^2, \dots, p_k^2 \text{ und } p_{k+1} \text{ teilen } n \text{ nicht}\}$ . Dann ist  $L \in \text{strong-1DT}[\log \log]$ , aber  $L \notin \text{REG}$ .

**Beweis:**  $n \in L$  bedeutet, daß  $n = p_1 \cdots p_k \cdot r$  für ein  $r$ , das nur Primzahlen  $> p_{k+1}$  enthält. Bei Input  $n$  arbeitet die strong-2DT[log log]-TM  $M$  wie folgt ( $M$  kann für jede Zahl  $m$  auf log  $m$ -Platz testen, ob  $m$  Primzahl ist oder nicht):

```

p := 2
while p teilt n do
  if p2 teilt n then reject else p := nächste Primzahl;
accept.

```

$M$  benötigt höchstens log  $p_{k+1}$  Platz, wenn  $p_{k+1}$  die erste Primzahl ist, bei der der Algorithmus abbricht ( $p_{k+1}$  teilt  $n$  nicht oder  $p_{k+1}^2$  teilt  $n$ ). Ist  $p_{k+1}$  die erste Primzahl, die  $n$  nicht teilt, so ist

1.  $n \geq p_1 \cdots p_k > k! > 2^{\frac{k}{2} \cdot \log \frac{k}{2}}$  bzw.  $\log n \geq \log(p_1 \cdots p_k) > \frac{k}{2} \log \frac{k}{2}$ ,
2.  $k \log k \geq 2k + \log k + 1 + 3k$ , da  $2k + \log k + 1 + 3k \leq 5k + \log k + 1 \leq 7k \leq k \log k$ ,
3.  $p_{k+1} \leq 2(k+1) \log(k+1) \leq 2k \log k + 2k + \log k + 1 \leq 3k \log k - 3k \leq 3k \log(k-1) = 6 \frac{k}{2} \log(\frac{k}{2}) \leq 6 \log n$

Also benötigt  $M$  für jeden Input nur log log  $n + 3$  Platz. ■

## 7 Nichtdeterministische Maschinen

### Satz 7.1 (Savitch)

$v$ -2NT[ $f$ ]( $t$ )  $\subseteq$  v-2DT[ $f \cdot \log t$ ]( $2^{c \cdot f \cdot \log t}$ ) für ein  $c$ , falls  $f \geq \log$  ( $v = \text{strong, weak}$ ).

**Beweis:** Sei  $M$  eine weak-2NT[ $f$ ]( $t$ )-TM angewandt auf den Input  $w$  mit der Länge  $n$ , sei  $s(w)$  die Startsituation von  $M$  bei  $w$ . O.B.d.A. drucke  $M$  nie  $\square$ . Wir geben eine weak-2DT[ $f \cdot \log t$ ]( $2^{c \cdot f \cdot \log t}$ )-TM  $M'$  an, die dieselbe Sprache erkennt.  $M'$  will herausfinden, ob  $M$  von  $s(w)$  aus innerhalb von  $t$  Schritten eine akzeptierende Stopsituation erreicht. Dazu überprüft  $M'$  für jede mögliche akzeptierende Stopsituation  $s$  mit Bandbedarf  $f$ , ob es einen Lauf einer Länge  $\leq t$  von  $s(w)$  nach  $s$  gibt.

Wir nehmen einmal an, daß  $M'$  sich zwei Situationen  $s_1, s_2$  mit Bandaufwand  $b$  und die Zeit  $t$  notiert hat, um zu überprüfen, ob es einen Lauf der Länge  $t$  von  $s_1$  nach  $s_2$  gibt. Gibt es einen solchen Lauf, so gibt es eine Situation  $s$ , die in der Mitte des Laufs liegt und die den Lauf in zwei gleich lange Teile teilt, für die also gilt: es gibt einen Lauf der Länge  $\frac{t}{2}$  von  $s_1$  nach  $s$  und einen solchen von  $s$  nach  $s_2$ . Diese Situation versucht  $M'$  zu finden, indem es für alle Situationen mit Bandaufwand  $b$  versucht diese beiden - halb so großen - Teilfragen zu beantworten. Bei diesen beiden Teilfragen geht  $M'$  aber rekursiv genauso vor, halbiert die Zeit und versucht wiederum eine Situation dazwischen zu finden. Diese rekursive Prozedur geben wir genauer so an:

```

function LAUF( $s_1, s_2, t$ ); {= true, falls es Lauf von  $s_1$  nach  $s_2$  einer Länge  $\leq t$  gibt, = false sonst}
  if ( $t = 0$  and  $s_1 = s_2$ ) then return(true);
  if  $t = 1$  and  $s_1 \vdash s_2$  then return(true);
  if  $t > 1$  then
    for each  $s$  of length  $\leq f$  do
      if LAUF( $s_1, s, \lfloor \frac{t}{2} \rfloor$ ) and LAUF( $s, s_2, \lceil \frac{t}{2} \rceil$ ) then return(true);
  return(false);
end.

```

*Bandaufwand:* Die Funktion hat eine Schachtelungstiefe log  $t$  und Parameter der Länge  $f$ , benötigt also  $f \cdot \log t$  Platz.

*Zeitaufwand:* Wir könnten es uns leicht machen und feststellen, daß  $M'$  auf Band  $f \log t$  ohne zu zykeln gar nicht mehr Zeit arbeiten kann als  $2^{c' f \log t}$ . Dies ist auch richtig für ein  $c'$ , das nur von  $M'$  abhängt, genauer von der Größe der Alphabete von  $M'$ . Wir können aber auch zeigen, daß  $M'$  nicht

mehr als  $2^{cf \log t}$  Schritte tut, für ein  $c$ , das nur von  $M$  abhängt.

Es gibt  $c^f$  Situationen einer Größe  $\leq f$  (für ein nur von  $M$  abhängiges  $c$ ), die in der for-Schleife in jedem Aufruf alle durchprobiert werden. Darum ist die Zeit durch folgende Rekursionsgleichung bestimmt:

$$T(0) = T(1) = f, \text{ und für } t > 1 : \\ T(t) = 2 \cdot c^f \cdot T\left(\frac{t}{2}\right) + f.$$

Für die Lösung dieser Rekursionsgleichung ist:  $T(t) \in O(c^{f \cdot \log t})$ , wie wir im Anschluß sehen.

Bisher sind wir davon ausgegangen, daß  $M'$  die Werte  $f$  und  $t$  kennt. Dies ist der Fall, wenn beide Funktionen auf  $f \log t$ -Platz berechenbar sind. Wenn das nicht so ist, so müssen wir eine zusätzliche Überlegung anstellen, wobei wir jetzt zwischen einer schwachen und starken Beschränkung unterscheiden müssen (wir nehmen an, daß die Eigenschaft stark oder schwach beschränkt, eine Eigenschaft der Maschine und nicht des Maßes ist. Es wirkt nicht natürlich, Maschinen zu betrachten, die schwach bandbeschränkt, aber stark zeitbeschränkt sind, oder umgekehrt.)

Sei  $M$  schwach beschränkt, also eine weak-2NT $[f]\langle t \rangle$ -TM. Dann gibt  $M'$  versuchsweise einen eigenen Bandaufwand  $B$  vor, der bei Mißerfolg wächst:  $B = 1, 2, 3, \dots$ . Bei jedem  $B$  führt  $M'$  sein Verfahren durch, und zwar für alle möglichen Bandaufwände  $s$  und Zeitaufwände  $t$  von  $M$  mit  $s \log t = B$ . Wenn  $M'$  in keinem Fall Erfolg hat, erhöht  $M'$  seinen Bandaufwand  $B$  um 1. Wenn  $M$  verwirft, so wird  $M'$  nicht aufhören,  $B$  zu erhöhen, und somit unendlich viel Band und Zeit verbrauchen. Darum ist auch  $M'$  nur schwach beschränkt. Wenn  $M$  aber akzeptiert, etwa mit dem Bandaufwand  $s$  und dem Zeitaufwand  $t$ , so wird  $M'$  einmal Erfolg haben. Es ist klar, daß der Zeitaufwand von  $M'$  wächst, da  $M'$  das Verfahren für jede Bandschranke  $\sigma$  und Zeitschranke  $\tau$  mit  $\sigma \log \tau < s \log t$  durchführt. Es gibt aber höchstens  $s \log t \cdot 2^{s \log t}$  solche Schrankenpaare  $(\sigma, \tau)$ , d.h. beim Zeitaufwand von  $M'$  erhöht sich lediglich die Konstante  $c$  im Exponenten.

Sei  $M$  stark beschränkt, also eine strong-2NT $[f]\langle t \rangle$ -TM.  $M'$  versucht wieder den eigenen Bandaufwand  $B$  vorzugeben und läßt ihn wachsen, wenn es keinen Erfolg hat. Damit  $M'$  aber stark beschränkt ist, müssen wir ein Abbruchkriterium für das Erhöhen von  $B$  angeben. Wir nennen  $C$  eine  $B$ -Situation, wenn  $s \log t \leq B$  ist, wobei  $C$  den Platzbedarf  $s$  hat und in der Zeit  $t$  erreicht wird.  $M'$  kann  $B$  erhöhen, wenn es noch eine  $B$ -Situation  $C$  gibt mit einer unmittelbaren Nachfolgersituation  $C'$ , die  $B'$ -Situation ist für ein  $B' > B$ . Da  $M$  stark beschränkt, darf  $M'$  dann auch  $B'$ -Platz verbrauchen. Bevor also  $M'$  seinen Bandaufwand  $B$  erhöht, muß  $M'$  mit seinem Verfahren für alle Situationen überprüfen, ob sie  $B$ -Situationen sind und ob auf sie eine Situation folgen kann, für deren Platzbedarf  $s$  und deren Erreichbarkeitszeit  $t$  gilt:  $s \log t > B$ . Auch hier überlegt man sich, daß der Zeitaufwand nur polynomiell ist im Zeitaufwand des Verfahrens, sich also nur die Konstante im Exponenten ändert. ■

### Lemma 7.2

Die Rekursionsgleichung

$$T(n) = \begin{cases} b & \text{falls } n = 1 \\ aT\left(\frac{n}{c}\right) + bn & \text{falls } n > 1 \end{cases}$$

hat eine Lösung

$$T(n) \in \begin{cases} O(n) & \text{falls } a < c \\ O(n \log n) & \text{falls } a = c \\ O(n^{\log_c a}) & \text{falls } a > c \end{cases}$$

**Beweis:**

$$T(n) = bn \left( 1 + \frac{a}{c} + \frac{a^2}{c^2} + \dots + \frac{a^t}{c^t} \right)$$

mit  $t = \log_c n$  ist eine Lösung der Rekursionsgleichung.

Für  $a < c$  ist die Summe in der Klammer gerade

$$\frac{1 - \left(\frac{a}{c}\right)^{t+1}}{1 - \frac{a}{c}} \leq \frac{1}{1 - \frac{a}{c}} = \frac{c}{c - a}, \text{ also } T(n) \in O(n).$$

Für  $a = c$  ist

$$T(n) = bn \cdot t = bn \cdot \log_c n \in O(n \cdot \log n).$$

Für  $a > c$  ist

$$\begin{aligned} T(n) &= bn \cdot \frac{\left(\frac{a}{c}\right)^{t+1} - 1}{\frac{a}{c} - 1} = \frac{b}{\frac{a}{c} - 1} \cdot n \left( \frac{a^{t+1}}{c^{t+1}} - 1 \right) \\ &= \frac{b}{\frac{a}{c} - 1} \cdot n \cdot \frac{a \cdot a^{\log_c n}}{c \cdot n} - \frac{b}{\frac{a}{c} - 1} \cdot n = O(a^{\log_c n}) = O(n^{\log_c a}). \end{aligned}$$

■

### Korollar 7.3

$v\text{-}2\text{NT}[f] \subseteq v\text{-}2\text{DT}[f^2]$ , falls  $f \geq \log$  ( $v = \text{strong, weak}$ ).

Auch für den Satz von Savitch gibt es eine Variante mit einem Bandaufwand unter  $\log$ .

### Satz 7.4 (Savitch unter $\log$ )

$f \in o(\log) \implies 2\text{NT}[f] \subseteq 2\text{DT}[f \cdot \log]$ .

**Beweis:** Sei  $M$  eine  $2\text{NT}[f]$ -TM ( $f \in o(\log)$ ). Wir konstruieren eine äquivalente  $2\text{DT}[f \cdot \log]$ -TM  $M'$ . Ist  $s$  eine Situation, so sei  $\text{pos}(s)$  die Inputposition in der Situation  $s$ . Ist  $s_1 \vdash^* s_4$  ein Lauf, in dem der Inputkopf das Intervall  $[l, r]$  nicht verläßt und  $\text{pos}(s_1), \text{pos}(s_4) \in \{l, r\}$ , so existieren Situationen  $s_2$  und  $s_3$  mit  $\text{pos}(s_2), \text{pos}(s_3) \in \{l, \frac{l+r}{2}, r\}$ , so daß

- $s_1 \vdash^* s_2 \vdash^* s_3 \vdash^* s_4$ ,
- die Läufe  $s_1 \vdash^* s_2$  und  $s_3 \vdash^* s_4$  höchstens halb so lang sind wie der Lauf  $s_1 \vdash^* s_4$  und
- die Inputpositionen während des Laufs  $s_2 \vdash^* s_3$  ganz im Inputintervall  $[l', r']$  liegen, wobei

$$\begin{aligned} l' &= l \text{ und } r' = \frac{l+r}{2} \text{ falls } \text{pos}(s_2), \text{pos}(s_3) \in \{l, \frac{l+r}{2}\}, \\ l' &= \frac{l+r}{2} \text{ und } r' = r \text{ falls } \text{pos}(s_2), \text{pos}(s_3) \in \{\frac{l+r}{2}, r\}. \end{aligned}$$

Bei  $\text{pos}(s_2), \text{pos}(s_3) = \frac{l+r}{2}$  gibt es 2 Möglichkeiten für  $l', r'$ .

Wir schreiben für die Situation  $s$  das Paar  $(c, i)$ , wenn  $i = \text{pos}(s)$  und  $c$  die Konfiguration von  $s$  ist, d.h. Inhalt und Kopfstellung des Arbeitsbandes und den Zustand der Kontrolleinheit repräsentiert. Die Prozedur  $\text{Path}(c_1, i_1, c_4, i_4, l, r, t)$  überprüft, ob es einen Lauf  $s_1 = (c_1, i_1) \vdash^* (c_4, i_4) = s_4$  einer Länge  $\leq t$  gibt, bei dem der Inputkopf das Inputintervall  $[l, r]$  nicht verläßt.

```
procedure Path( $c_1, i_1, c_4, i_4, l, r, t$ )
```

```
if  $t = 1$  then ...
```

```
  else for each  $(s_2 = (c_2, i_2), s_3 = (c_3, i_3))$  with  $|c_1| \leq |c_2| \leq |c_3| \leq |c_4|$ 
```

```
    and  $i_2, i_3 \in \{l, \frac{l+r}{2}, r\}$  and  $d \in \{0, 1\}$  do
```

```
      if Path( $c_1, i_1, c_2, i_2, l, r, \frac{t}{2}$ ) and Path( $c_3, i_3, c_4, i_4, l, r, \frac{t}{2}$ ) and Path( $c_2, i_2, c_3, i_3, l', r', t$ )
```

```
        then return(true);
```

```
return false
```

( $l', r'$  wie oben, wobei  $d \in \{0, 1\}$  die Zweideutigkeit für  $l', r'$  determiniert).

Die  $2\text{DT}$ -TM  $M'$  notiert von den Parametern der Prozedur nur  $c_1, c_4$ . Aber bei der Wahl "for each ..." merkt sie sich, welche der 3 Möglichkeiten  $i_2$  und  $i_3$  jeweils eingenommen haben und wie  $d$  aussieht.

Aus der Folge dieser Wahlen gehen die Inputpositionen  $i_1$  und  $i_4$  und die aktuellen Intervallgrenzen  $l, r$  hervor.

Die Prozedur ruft 3 Prozeduren auf, bei denen sich entweder die Zeit oder das Inputintervall halbiert. Spätestens nach Aufruftiefe  $\log t + \log n \leq \log n + f(n) + \log n$  (Inputlänge  $n$ ,  $t \leq n \cdot 2^{f(n)}$ ) läßt sich weder Zeit noch Inputintervall teilen und  $M$  berechnet das Prädikat Path direkt. Der Rekursionstack hat somit  $3 \cdot \log n$  Parameterblöcke der Größe  $2f(n) + 3$   $[(c, c', w_1, w_2, d)]$  wobei  $c, c'$  Konfigurationen,  $w_1, w_2$  Wahlen aus  $\{1, 2, 3\}$  und  $d \in \{0, 1\}$ , d.h. eine Höhe von höchstens  $6 \cdot \log n \cdot f(n)$ . ■

Für das Folgende wollen wir hier Notationen aufführen, die sich von der unseren unterscheiden, aber in der Literatur sehr gebräuchlich sind.

**Notation 7.5**

$$\begin{aligned} \text{DTime}(f) &= \text{DT}(f) &= 2\text{DT}^*(f) &= \bigcup_{k \in \mathbb{N}} 2\text{DT}^k(f) \\ \text{DSpace}(f) &= \text{DS}(f) &= 2\text{DT}[f], \\ &&\text{DT}(\text{lin}) &= \bigcup_{c \in \mathbb{N}} \text{DT}(c \cdot n), \\ &&\text{DT}(\text{poly}) &= \bigcup_{k \in \mathbb{N}} \text{DT}(n^k), \\ &&\text{DT}(\text{ex}) &= \text{DT}(2^{\text{lin}}). &= \bigcup_{c \in \mathbb{N}} \text{DT}(2^{c \cdot n}) \end{aligned}$$

Analog definieren wir  $\text{DS}(\text{lin}), \text{DS}(\text{poly}), \text{DS}(\text{ex})$  und alle Klassen entsprechend auch nichtdeterministisch. Weiter sei

$$\begin{aligned} \text{P} &:= \text{DT}(\text{poly}), \\ \text{NP} &:= \text{NT}(\text{poly}), \\ \text{PSPACE} &:= \text{DS}(\text{poly}) = \text{NS}(\text{poly}) \quad \text{Satz 7.1 (Savitch)}, \\ \text{EXTIME} &:= \text{DT}(\text{ex}). \end{aligned}$$

**Lemma 7.6 (Translationslemma)**

Sei  $f, g > \log$ ,  $h > \text{id}$  und  $h$  auf  $\log h$ -Band berechenbar. Dann gilt für  $a, b \in \{\text{weak}, \text{strong}\}$ ,  $X, Y \in \{\text{N}, \text{D}\}$ :

$$a\text{-}2\text{XT}[f] \subseteq b\text{-}2\text{YT}[g] \implies a\text{-}2\text{XT}[f \circ h] \subseteq b\text{-}2\text{YT}[g \circ h].$$

**Beweis:** Sei  $L \in 2\text{XT}[f \circ h]$  vermöge  $M_1$ . Sei  $L^\# = \{w\#^i \mid w \in L, i = h(n) - n, |w| = n\}$ .

$$\begin{array}{ccccccc} L(M_1) & = & L & \in & 2\text{XT}[f \circ h] & ? & 2\text{YT}[g \circ h] & \ni & L & = & L(M_4) \\ & & \downarrow & & \downarrow & & \uparrow & & \uparrow & & \\ L(M_2) & = & L^\# & \in & 2\text{XT}[f] & \subseteq & 2\text{YT}[g] & \ni & L^\# & = & L(M_3) \end{array}$$

Dann ist  $L^\# \in 2\text{XT}[f]$  vermöge der TM  $M_2$ .  $M_2(w\#^i)$  arbeitet wie folgt:

1.  $M_2$  markiert  $\log(n + i)$  Felder und berechnet darauf  $h(n)$ , wozu nach Voraussetzung  $\log h(n)$  Felder reichen. Wenn der Platz also nicht reicht, ist  $n + i \neq h(n)$  und  $M_2$  verwirft. Ansonsten prüft  $M_2$ , ob  $h(n) = n + i$ .
2.  $M_2$  simuliert  $M_1(w)$ .  $M_1$  benötigt  $fh(n) = f(n + i)$  Platz (Ist  $M_1$  stark bandbeschränkt, so ist dies auch  $M_2$ ).

Nach Voraussetzung ist  $L^\# \in 2\text{YT}[g]$  vermöge einer TM  $M_3$ . Dann ist  $L \in 2\text{YT}[g \circ h]$  vermöge der TM  $M_4$ .  $M_4(w)$  arbeitet wie folgt ( $|w| = n$ ):

1.  $M_4$  konstruiert  $h(n)$  auf  $\log h(n) < gh(n)$  Platz.
2.  $M_4$  simuliert  $M_3(w\#^{h(n)-n})$ . Kommt dabei der Inputkopf von  $M_3$  in den  $\#$ -Bereich, so läßt  $M_4$  seine Inputkopf am Ende des Inputs stehen und hält auf einem  $(h(n) - n)$ -Zähler auf Platz  $\log h(n) < gh(n)$  die Position des Inputkopfes von  $M_3$  mit.

Ist  $M_3$  stark-(schwach-) beschränkt, so ist dies auch  $M_4$ . ■

Das Translationslemma kann man in einer veränderten Form auch für einen Bandaufwand unter log zeigen:

**Lemma 7.7 (Translationslemma unter log (Szepietowski 89))**

$$2DT[\log \log] = 2NT[\log \log] \implies 2DT[\log] = 2NT[\log]$$

**Beweis:**

1. Das kleinste gemeinsame Vielfache  $\text{kgV}(n_1, \dots, n_k)$  von  $k$  Zahlen  $n_1, \dots, n_k$  ist das Produkt der größten Primzahlpotenzen, die in den Primzahlpotenzdarstellungen (PPDs) von  $n_1, \dots, n_k$  vorkommen. Betrachten wir die  $k$  ersten Zahlen  $1, \dots, k$ . Alle Primzahlpotenzen  $\leq k$  kommen in den PPDs der Zahlen  $1, \dots, k$  vor. Ist  $p^l$  eine größte Primzahlpotenz  $\leq k$ , so ist  $l \leq \log_p k$  und  $l + 1 > \log_p k$ , d.h.  $l = \lfloor \log_p k \rfloor$ . Damit gilt:
2. Das kleinste gemeinsame Vielfache  $m(k)$  der Zahlen  $1, \dots, k$  ist  $\prod \{p^{\lfloor \log_p k \rfloor} \mid p \text{ prim} \wedge p \leq k\}$ .
3. Für die Zahl  $\pi(k)$  der Primzahlen unter  $k$  gilt:

$$\frac{k}{\log k} < \pi(k) < \frac{2k}{\log k}.$$

Die größten Primzahlpotenzen  $\leq k$  liegen aber über  $\sqrt{k}$  (sonst könnte man sie quadrieren und sie lägen immer noch unter  $k$ ). Also ist

$$\begin{aligned} (\sqrt{k})^{\frac{k}{\log k}} &\leq m(k) \leq k^{\frac{2k}{\log k}} \\ \text{bzw. } 2^{\frac{k}{2}} &\leq m(k) \leq 2^{2k}. \end{aligned}$$

Das bedeutet

$$\forall k : (\sqrt{2})^k \leq m(k) \leq 4^k.$$

Sei  $L \in 2NT[\log]$  vermöge  $M_1$ . Sei

$$\begin{aligned} L^\# &= \{w\#^l \mid w \in L, i \leq |w| \implies i \mid l\} \\ &= \{w\#^l \mid l = \text{irgendein gemeinsames Vielfaches von } \{1, \dots, |w|\}\}. \end{aligned}$$

Dann ist  $L^\# \in 2NT[\log \log]$  vermöge der TM  $M_2$ :  $M_2(w\#^l)$  arbeitet wie folgt:

$M_2$  prüft, ob die Länge  $l$  des  $\#$ -Strings teilbar ist durch alle  $i \leq |w| = n$ , was höchstens  $\log i$  Platz benötigt. Dazu errechnet  $M_2$  den kleinsten Nichtteiler von  $l$ . Der muß größer als  $|w|$  sein. Der kleinste Nichtteiler von  $l$  ist  $\text{kN}(l) \leq 8 \log l$  (Lemma 6.31). Damit kann die erste Zahl, die  $l$  nicht teilt, auf  $\log \log l$  Platz geschrieben werden. Ist der Test erfolgreich, so ist  $l \geq m(n) \geq (\sqrt{2})^n$ , also  $\log \log(n+l) \geq \frac{1}{2} \log n$ . Damit hat  $M_2$  genügende Platz, um  $M_1$  auf  $w$  zu simulieren.

$$\begin{array}{ccccccc} L(M_1) & = & L & \in & 2NT[\log] & ? & 2DT[\log] & \ni & L & = & L(M_4) \\ & & \downarrow & & \downarrow & & \uparrow & & \uparrow & & \\ L(M_2) & = & L^\# & \in & 2NT[\log \log] & \subseteq & 2DT[\log \log] & \ni & L^\# & = & L(M_3) \end{array}$$

Nach Voraussetzung gibt es eine zu  $M_2$  äquivalente  $2DT[\log \log]$ -TM  $M_3$ , die ebenfalls  $L^\#$  erkennt. Dann erkennt die  $2DT[\log]$ -TM  $M_4$  die Sprache  $L$  wie folgt:  $M_4(w)$  simuliert  $M_3(w\#^{m(|w|)})$ .  $M_3$  hat bei Input  $w\#^{m(|w|)}$  höchstens  $2^{c \cdot \log \log(n+m(n))} \leq 2^{d \log n} = n^e$  Konfigurationen für Konstanten  $c, d, e$  ( $|w| = n$ ).

Wenn  $M_3$  in den  $\#$ -String kommt, so zählt  $M_4$  die Inputpositionen bis  $n^e$  mit. Überschreitet  $M_3$  bei Input  $w\#^{m(|w|)}$  das Feld  $n + n^e$ , so läuft  $M_3$  in eine Rechtsperiode, d.h. in einen Teillauf, der mit

derselben Konfiguration beginnt und endet und den Inputkopf nach rechts bewegt. Halten wir auf einem 2. Zähler das jeweils rechteste erreichte Feld fest, so können wir die Periode so beginnen lassen, daß sie zuletzt ein noch nie betretenes Feld erreicht. Die Periodendistanz (nach rechts)  $d$  stellt  $M_4$  wir fest, indem  $M_4$  sich die Konfiguration merkt, wenn ein Feld zum 1. Mal betreten wurde und dann solange simuliert, bis diese Periode wieder auftritt. Jetzt weiß  $M_4$ , daß  $M_3$  den  $\#$ -String in  $d$ -Schritten durchläuft. Die Frage ist nur wie  $M_3$  rechts ankommt. Lassen wir die 1. Periode auf Feld  $k$  im  $\#$ -String starten, so ist die Strecke  $m(n)-k$  zu durchlaufen, d.h. die letzte ganze Periode endet auf Feld  $(m(n) - k) \bmod d$ . Da  $(a + b) \bmod d = (a \bmod d + b \bmod d) \bmod d$  kann  $M_4$  dies ausrechnen, da keine Zahl  $> n^e$  vorkommt. Auf dem rechten Block von  $n^e$  Feldern, kann  $M_4$  wieder zählend simulieren, bis  $M_3$  wieder zuweit nach links gelangt. Dann wird wieder gerechnet.

$$w \in L \text{ gdw. } w \#^{m(|w|)} \in L\#$$

■

### Bemerkung 7.8

Ist  $M$  stark (schwach) beschränkt, so auch  $M'$ .

Das folgende Theorem löst ein Problem, das unter dem Namen „zweites LBA-Problem“ berühmt geworden ist, weil es 20 Jahre ungelöst geblieben war. 1987 hat Neil Immerman die Lösung gefunden und durch Email gleich verbreitet. Dennoch war ihm der Slowake Róbert Szelepcsényi knapp zuvorgekommen. Er hatte diesen kurz zuvor in einem technischen Report in tschechischer Sprache veröffentlicht. Einer der nicht seltenen Fälle, wo mehrere Personen oder Gruppen zur selben Zeit an derselben Sache arbeiten, ohne voneinander zu wissen.

### Satz 7.9 (Immerman)

strong-2NT[ $f$ ] ist abgeschlossen unter Komplement, falls  $f \geq \log$ ,

weak-2NT[ $f$ ] ist abgeschlossen unter Komplement, falls  $f \geq \log$  und  $f$  2DT-bandkonstruierbar.

**Beweis:** Gegeben sei eine 2NT[ $f$ ]-TM  $M$ . Wir suchen eine 2NT[ $f$ ]-TM  $\overline{M}$  mit  $L(\overline{M}) = \overline{L(M)}$ . Sei  $w$  der Input,  $s(w)$  die Startsituation mit Input  $w$ ,  $|w| = n$ . Es gibt höchstens  $n2^{cf(n)} = N$  Situationen von  $M$  mit Input  $w$ . Sei  $S = \{s_1, \dots, s_N\}$  die Menge dieser Situationen in einer Ordnung, die von  $M$  berechnet werden kann, z.B. geordnet zuerst nach Inputposition, dann nach Arbeitsposition und dann nach der lexikographischen Ordnung des Bandinhalts. Sei  $E = \{s \in S \mid s(w) \vdash^* s\}$  die Menge dieser Situationen, die von der Startsituation aus erreicht werden können und  $E_k = \{s \mid s(w) \vdash^{\leq k} s\}$  die Menge dieser Situationen, die von der Startsituation aus in höchstens  $k$  Schritten erreicht werden können. Seien weiter  $e = |E|$  und  $e_k = |E_k|$  die Mächtigkeiten dieser Mengen.

Wenn  $\overline{M}$  die Mächtigkeit  $e$  kennt, so kann  $\overline{M}$  die TM  $M$  wie folgt simulieren:  $\overline{M}$  zählt die Situationen  $s_1, \dots, s_N$  in dieser Reihenfolge auf, d.h.  $\overline{M}$  nimmt der Reihe nach diese Situationen ein (setzt den Inputkopf an die richtige Stelle, und speichert Arbeitsinhalt und Arbeitskopfposition aufs Band). Bei jeder dieser Situationen  $s$  versucht  $\overline{M}$  nun einen Lauf von  $M$  zu finden, der von  $s(w)$  nach  $s$  führt, indem  $\overline{M}$  einfach die TM  $M$  simuliert und überprüft, ob die Rechnung zur Situation  $s$  führt. Hat  $\overline{M}$  dabei Erfolg, so erhöht  $\overline{M}$  einen Zähler, der die Erfolge mitzählt. Ist  $s$  eine akzeptierende Stopsituation, so weiß  $\overline{M}$ , daß  $M$  akzeptieren kann und stoppt sofort verwerfend. Andernfalls geht  $\overline{M}$  zur nächsten Situation über. Hat  $\overline{M}$  schließlich alle Situationen aufgezählt und enthält der Erfolgswähler die Zahl  $e$ , so weiß  $\overline{M}$ , daß sie alle erreichbaren Situationen gefunden und geprüft hat. Da offenbar keine akzeptierende darunter waren, muß  $M$  verwerfen, was  $\overline{M}$  zum Anlaß nimmt, zu akzeptieren.

Bleibt die Frage, wie  $\overline{M}$  die Mächtigkeit  $e$  konstruiert. Zuerst konstruiert sich  $\overline{M}$  die Mächtigkeit  $e_0$ . Die ist 1, da nur die Startsituation selbst in 0 Schritten erreichbar ist. Jetzt beschreiben wir, wie  $\overline{M}$  die Mächtigkeit  $e_{i+1}$  konstruiert, wenn sie die Mächtigkeit  $e_i$  schon kennt. Dazu gibt es zwei Möglichkeiten:

**1. Möglichkeit:**  $\overline{M}$  geht wie oben alle  $N$  Situationen aus  $S$  durch und versucht bei jeder Situation  $s$  sie selbst oder einen Vorgänger von ihr in  $E_i$  zu finden und bei Erfolg einen Zähler  $z$  zu erhöhen, der die gefundenen Situationen aus  $E_{i+1}$  zählt. Dazu geht  $\overline{M}$  nochmal die Situationen aus  $S$  durch, wobei  $\overline{M}$  bei jeder dieser Situationen  $s'$  versucht einen Lauf einer Länge  $\leq i$  von  $M$  zu finden, der von  $s(w)$  nach  $s'$  führt, indem  $\overline{M}$  die TM  $M$  maximal  $i$  Schritte lang simuliert und überprüft, ob die Rechnung zur Situation  $s'$  führt. Hat  $\overline{M}$  dabei Erfolg, so erhöht  $\overline{M}$  einen Zähler, der die Erfolge mitzählt. Ist  $s'$  ein Vorgänger von  $s$  oder  $s' = s$ , so weiß  $\overline{M}$ , daß  $s$  zu  $E_{i+1}$  gehört und erhöht den Zähler, der die so gefundenen Situationen aus  $E_{i+1}$  mithält und geht zur der nächsten Situation nach  $s$  über. Hat  $\overline{M}$  schließlich alle Situationen aufgezählt und enthält der Erfolgswähler die Zahl  $e_i$ , so weiß  $\overline{M}$ , daß es alle Situationen aus  $E_i$  gefunden und geprüft hat. Da offenbar keine Vorgänger von  $s$  dabei waren, geht  $\overline{M}$  zur der nächsten Situation nach  $s$  über, ohne den Zähler  $z$  zu erhöhen. Am Ende steht im Zähler  $z$  die Mächtigkeit  $e_{i+1}$ .

**2. Möglichkeit:**  $\overline{M}$  geht alle Situationen  $s$  aus  $E_i$  durch und zählt diese und ihre unmittelbaren Nachfolger, allerdings nur, wenn sie nicht vorher schon in  $E_i$  oder als Nachfolger einer Situation aus  $E_i$  vorgekommen sind. Dazu geht  $\overline{M}$  noch einmal die Situationen aus  $E_i$  durch überprüft aber nur die Situationen bis  $s$  (wobei freilich alle  $e_i$  Situationen aus  $E_i$  gefunden werden müssen, um sicherzugehen, daß keine ausgelassen wurde).

$\overline{M}$  geht also wie oben alle  $N$  Situationen aus  $S$  durch und versucht bei jeder Situation  $s$  einen Lauf einer Länge  $\leq i$  von  $M$  zu finden, der von  $s(w)$  nach  $s$  führt, indem  $\overline{M}$  die TM  $M$  maximal  $i$  Schritte lang simuliert und überprüft, ob die Rechnung zur Situation  $s$  führt. Hat  $\overline{M}$  dabei Erfolg, so erhöht  $\overline{M}$  einen Zähler  $z$ , der die Erfolge mitzählt, und merkt sich  $s$  und die unmittelbaren Nachfolger von  $s$  bzgl. der Befehlstafel von  $M$ .

Für jede Situation  $s'$  mit  $s' = s$  oder  $s'$  Nachfolger von  $s$  überprüft  $\overline{M}$ , ob  $s'$  schon einmal vorher bei einer Situation  $\hat{s}$  vorkam, die in der Aufzählung von  $E_i$  vor  $s$  erfolgreich aus  $s(w)$  abgeleitet wurde.

Dazu geht  $\overline{M}$  wieder alle  $N$  Situationen aus  $S$  durch und versucht bei jeder Situation  $\hat{s}$  einen Lauf einer Länge  $\leq i$  von  $M$  zu finden, der von  $s(w)$  nach  $\hat{s}$  führt, indem  $\overline{M}$  die TM  $M$  maximal  $i$  Schritte lang simuliert und überprüft, ob die Rechnung zur Situation  $s$  führt. Hat  $\overline{M}$  dabei Erfolg, so erhöht  $\overline{M}$  einen Zähler  $\hat{z}$ , der die Erfolge mitzählt. Ist  $\hat{s} < s$  und  $\hat{s}$  oder ein Nachfolger von  $\hat{s}$  gerade  $s'$ , so bricht  $\overline{M}$  diese Überprüfung ab, weil klar ist, daß  $s'$  nicht noch einmal gezählt werden darf. Ist  $\hat{s} < s$  und kein Nachfolger von  $\hat{s}$  gerade  $s'$  oder ist  $\hat{s} > s$ , so geht  $\overline{M}$  zu der nächsten auf  $\hat{s}$  folgenden Situation, bis schließlich alle  $N$  Situationen aufgeführt sind. Steht am Ende  $e_i$  im Zähler  $z$ , so weiß  $\overline{M}$ , daß es alle Situationen aus  $E_i$  betrachtet hat und die Überprüfung ergeben hat, daß wir  $s'$  nicht schon aus einer vorher betrachteten Situation  $\hat{s}$  erhalten haben. Dann erhöht  $\overline{M}$  den Zähler  $z^+$ , der die so gefundenen Situationen aus  $E_{i+1}$  mithält. Am Ende enthält der Zähler  $z^+$  gerade die Mächtigkeit  $e_{i+1}$ .

Dies alles funktioniert, wenn  $f$  2NT-bandkonstruierbar ist, denn dann kann  $\overline{M}$  alle Situationen mit Platz  $f$  aufzählen. Ist  $f$  nicht 2NT-bandkonstruierbar, so muß  $\overline{M}$  das Band langsam anwachsen lassen ( $f(n) = 1, 2, 3, \dots$ ) bis die Mächtigkeit  $e$  nicht mehr anwächst. Dann ist klar, daß  $f(n)$  so groß ist, daß keine Situation mit mehr Platz noch erreichbar ist. Nach diesen verbalen Ausführungen geben wir noch ein paar Prozeduren an, die das nochmal anders darstellen.

program  $\overline{M}$ ;

```
function Aufz( $k, e_k, j$ ) {liefert die  $j$ -te Situation von  $E_k$ };
 $s := \varepsilon$ ;  $z := 0$ ;
for  $i = 1$  to  $N$  do
  if verified ( $s(w) \vdash^{\leq k} s_i$ ) then begin
     $z := z + 1$ ;
    if  $z = j$  then  $s := s_j$ ;
```

```

    endif;
endfor;
if  $z < e_k$  then verwerfe else return  $s$ ;

function Find1( $k, e_k$ ) {liefert  $e_{k+1}$ }
 $e_{k+1} = 0$ ;
for  $i = 1$  to  $N$  do
    for  $j = 1$  to  $e_k$  do
        if (Aufz( $k, e_k, j$ )  $\vdash s_i$  oder  $= s_i$ ) then  $e_{k+1} = e_{k+1} + 1$ ;
    endfor;
endfor;
return  $e_{k+1}$ ;

function Find2( $k, e_k$ ) {liefert  $e_{k+1}$ };
 $e_{k+1} = 0$ ;
for  $i = 1$  to  $e_k$  do begin
     $s = \text{Aufz}(k, e_k, i)$ ;
    for all  $s' \in \text{Nachfolger}(s) \cup \{s\}$  do begin
         $v = 0$ ;
        for  $j = 1$  to  $i$  do begin
            if (Aufz( $k, e_k, j$ )  $\vdash s'$  oder  $= s'$ ) then  $v = 1$ ;
        end for;
        if  $v = 0$  then  $e_{k+1} = e_{k+1} + 1$ ;
    end for;
end for;
return  $e_{k+1}$ ;

function Find {liefert  $e$  und maximale Zeit  $k$ };
 $k := 0$ ;  $e' := 1$ ; {da  $e_0 = 1$ }
while  $e \neq e'$  do begin  $k := k + 1$ ;  $e := e'$ ;  $e' = \text{Find1}(k, e)$  end;
return ( $e, k$ );

```

```

begin
    ( $e, k$ ) := Find;
    for  $i = 1$  to  $e$  do begin
         $s := \text{Aufz}(k, e, i)$ ;
        if  $s$  akzeptierend then verwerfe;
    endfor;
    akzeptiere;
end.

```

**Bemerkung 7.10**

Dieser Satz ist für weak-2NT[ $f$ ] mit  $f$  unter log definitiv falsch, wie wir aus den Sätzen 6.17 und 6.25 wissen, ob auch für strong-2NT[ $f$ ], wissen wir nicht, aber unser Beweis funktioniert auf jedenfall nicht mehr, wenn  $f$  unter log.

**7.1 Nichtdeterministische Bandhierarchien**

Mit Hilfe des Translationslemmas und des Satzes von Savitch können wir eine nichtdeterministische Bandhierarchie folgender Art beweisen:

**Satz 7.11 (Nondeterministische Bandhierarchie I)**

$\forall \varepsilon > 0 \quad \forall r : 2NT[n^r] \subsetneq 2NT[n^{r+\varepsilon}]$ .

**Beweis:** Für ein  $\varepsilon > 0$  und ein  $r \in \mathbb{N}$  wähle  $s, t$ , sodaß  $r \leq \frac{s}{t}$  und  $\frac{s+1}{t} \leq r + \varepsilon$ .

Annahme, die Aussage sei falsch, d.h.

$$2NT[n^{\frac{s+1}{t}}] \subseteq 2NT[n^{\frac{s}{t}}].$$

Definiere die Funktion  $h_i$  durch  $h_i(n) = n^{(s+i)t}$ . Die Funktion  $h_i$  ist auf  $\log h_i$  Platz berechenbar, d.h. wir können das Translationslemma 7.6 anwenden und die Annahme wird zu:

$$2NT[n^{(s+1)(s+i)}] \subseteq 2NT[n^{s(s+i)}] \quad \text{für alle } i \text{ und } s. \tag{1}$$

Wegen  $s(s+i) \leq (s+1)(s+i-1)$  für  $i \geq 1$  gilt außerdem

$$2NT[n^{s(s+i)}] \subseteq 2NT[n^{(s+1)(s+i-1)}] \tag{2}$$

Somit erhalten wir:

$$\begin{aligned} 2NT[n^{2s^2+2s}] = 2NT[n^{(s+1)2s}] & \stackrel{(1)}{\subseteq} 2NT[n^{s(2s)}] & \stackrel{(2)}{\subseteq} 2NT[n^{(s+1)(2s-1)}] \\ & \stackrel{(1)}{\subseteq} 2NT[n^{s(2s-1)}] & \stackrel{(2)}{\subseteq} 2NT[n^{(s+1)(2s-2)}] \\ & \dots & \dots \\ & \dots & \stackrel{(2)}{\subseteq} 2NT[n^{(s+1)s}] \\ & \stackrel{(1)}{\subseteq} 2NT[n^{s^2}] & \\ \text{wegen Savitch} & \subseteq 2DT[n^{2s^2}] & \subsetneq 2DT[n^{2s^2+2s}] \quad \text{W!} \end{aligned}$$

■

Eine interessante Diagonalisierung führt zu folgendem Satz, der von J.I.Seiferas, M.J.Fischer und A.R.Meyer 1973 vorgestellt wurde. Wir zeigen diesen Satz allerdings ohne den Rekursionssatz zu verwenden, auf nichtkonstruktive Art.

**Aufgabe 7.12**

Zeigen Sie, daß es keine arithmetische Funktion  $f$ , sodaß  $\text{strong-}2NT[f]$  alle entscheidbaren Sprachen enthält.

**Satz 7.13 (Nondeterministische Bandhierarchie two-way II)**

Seien  $f$  und  $g$  arithmetische Funktionen,  $g$  2DT-bandkonstruierbar (suc ist die Nachfolgerfunktion in  $\mathbb{N}$ ). Dann gilt:

$$f \circ \text{suc} \in o(g) \implies \text{strong-}2NT[f] \subsetneq \text{strong-}2NT[g].$$

**Beweis:** Sei  $(M_i)$  eine Aufzählung der 2NT-TMen,  $L$  eine entscheidbare Sprache, die für kein  $k \in \mathbb{N}$  in  $2NT[g(n+k)]$  liegt, und  $M$  eine immer haltende TM für  $L$ . Wir definieren jetzt folgende 2NT[ $g$ ]-TM  $N$ :

Den String  $0^i x 0^p$  mit  $x \in \{1, 2\}^*$  wollen wir abkürzen durch  $(i, x, p)$ .  $N$  angewandt auf  $(i, x, p)$  konstruiert und markiert  $g(|i, x, p|)$  Felder und simuliert  $M(x)$  auf diesen  $g(|i, x, p|)$  Feldern. Falls die Simulation klappt ohne daß die markierten Felder verlassen werden, liefert  $N$  denselben Output. Falls die markierten Felder nicht ausreichen, weil der Bandaufwand von  $M(x)$  zu groß ist, dann simuliert  $N$  die TM  $M_i$  auf  $(i, x, p+1)$ , wiederum auf diesen  $g(|i, x, p|)$  Feldern. Dabei führt  $N$  in einem binären Zähler der Länge  $g(|i, x, p|)$  die Zahl der simulierten Schritte mit. Falls die markierten Felder zur

Simulation und zum Zählen reichen, liefert  $N$  den Output von  $M_i$  auf  $(i, x, p + 1)$ . Reicht der Platz aber auch hier nicht, so stoppt  $N$  verwerfend. Etwas salopp schreiben wir das so:

$$N(i, x, p) = \begin{cases} M(x), & \text{falls Platz } g(|i, x, p|) \text{ für die Simulation reicht, sonst} \\ M_i(i, x, p + 1), & \text{falls Platz } g(|i, x, p|) \text{ für die Simulation reicht und die Simulation stoppt, sonst} \\ 0 & \end{cases}$$

( $M(x) = 0$  bzw.  $1$  bedeutet:  $M$  angewandt auf  $x$  verwirft bzw. akzeptiert).

Offensichtlich ist  $N$  eine strong-2NT[ $g$ ]-TM, die immer hält. Annahme, der Satz sei falsch, d.h. strong-2NT[ $g$ ]  $\subseteq$  strong-2NT[ $f$ ]. Dann gibt es eine strong-2NT[ $f$ ]-TM, die zu  $N$  äquivalent ist, d.h. dieselbe Sprache erkennt. Zu dieser gibt es aber eine äquivalente immer haltende strong-2NT[ $f$ ]-TM  $M_k$ . Beachten Sie, daß die TMen  $M, N$  und  $M_k$  immer halten. Nun gilt:

- Für alle  $x$  gibt es ein erstes  $p_x$ , so daß  $N(k, x, p_x) = M(x)$ .
- Zur Simulation von  $M_k(k, x, p + 1)$  benötigt  $N$  möglicherweise  $c_k f(|k, x, p + 1|)$  Platz für eine von  $M_k$  abhängige Konstante  $c_k$ .
- Es gilt:  $\exists n_0 \forall p \geq 0 \forall x (|x| \geq n_0) : g(|k, x, p|) \leq c_k f(|k, x, p + 1|)$ . Also verfügt  $N(k, x, p)$  bei Worten  $x$  einer Länge  $|x| \geq n_0$  über genügend Platz, um  $M_k(k, x, p + 1)$  erfolgreich zu simulieren. Damit ist  $M_k(k, x, p) = N(k, x, p) = M_k(k, x, p + 1)$ , falls  $p < p_x$ .
- Also gilt für alle  $x$  mit  $|x| \geq n_0$  :

$$M_k(k, x) = M_k(k, x, 1) = M_k(k, x, 2) = \dots = M_k(k, x, p_x) = M(x),$$

d.h.  $M_k(k, x)$  akzeptiert gdw.  $M(x)$  akzeptiert.

- Um  $L$  zu erkennen, können wir also eine TM  $M'$  konstruieren, deren Programm aus dem Programm von  $M_k$  besteht, vor das wir ein Programm setzen, das  $0^k$  vor den Input schreibt.  $M'$  ist dann eine 2NT[ $g(n + k)$ ]-TM, also  $L \in 2NT[ $g(n + k)$ ]$  im Widerspruch zur Annahme. ■

Mit dem Satz von Immerman können wir schließlich eine Bandhierarchie für 2NT-TMen beweisen, die genau so fein ist, wie die für die deterministischen TMen.

### Satz 7.14 (Nondeterministische Bandhierarchie two-way III)

Sei  $|\Sigma| \geq 3$ ,  $f$  und  $g$  totale, arithmetische Funktionen,  $g \geq \log \log$  und 2DT-bandkonstruierbar. Dann  $f \in o(g) \implies \text{weak-2NT}[f] \not\subseteq \text{strong-2NT}[g]$ .

**Beweis:** Sei  $(M_i)_{i \in \mathbb{N}}$  eine Aufzählung der 2NT-TMen. Wir konstruieren eine Diagonalmaschine  $D$ , die auf Input  $w$  ( $|w| = n$ ) wie folgt arbeitet: Ist  $w = u2v$  mit  $u \in \{0, 1\}^*$ , d.h.  $u$  ist der Präfix von  $w$  bis zur ersten 2, dann simuliert  $D$  die TM  $M_u(w)$  auf  $g(n)$  Platz, den sie vorher konstruiert. Reicht der Platz nicht, so verwirft  $D$ . Reicht der Platz, so verhält sich  $D$  wie  $M_u$ , d.h.  $D(w)$  hat genau dann einen akzeptierenden Lauf, wenn  $M_u(w)$  einen akzeptierenden Lauf hat, den  $D$  auf Platz  $g(n)$  simulieren kann. Ist  $M_u$  eine weak-2NT[ $f$ ]-TM und akzeptiert  $M_u$  den Input  $w$ , so gibt es einen akzeptierenden  $w$ -Lauf von  $M_u$  mit Platzaufwand  $\leq c_u f(n)$ , für eine von  $u$  abhängige Konstante  $c_u$ , was für genügend große  $n$  kleiner ist als  $g(n)$ . D.h. für jede weak-2NT[ $f$ ]-TM  $M_u$  gibt es ein Wort  $w = u2v$ , auf dem  $M_u$  von  $D$  erfolgreich simuliert wird, wenn  $w \in L(M_u)$ .  $D$  ist eine strong-2NT[ $g$ ]-TM, d.h. nach Satz 7.9 (Immerman) gibt es zu  $D$  eine komplementäre TM  $\Delta$ , die genau die Worte erkennt, die  $D$  verwirft.  $\Delta$  wird also auf einem Wort  $w = u2v$  genau dann verwerfen, wenn  $w \in L(M_u)$ , in allen anderen Fällen akzeptiert  $\Delta$ . Damit kann aber  $L(\Delta)$  nicht aus weak-2DT[ $f$ ] sein. ■

## 8 Zeitkomplexität

### 8.1 Die Zeit-Hierarchiesätze

#### 8.1.1 Die ursprüngliche Zeithierarchie

##### Satz 8.1 (Zeithierarchie 1)

Seien  $f, g$  totale Funktionen größer  $c \cdot \text{id}$  für ein  $c \in \mathbb{N}$ ,  $g$   $2\text{DT}^{k+1}$ -zeitkonstruierbar. Dann gilt:

$$f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^{k+1} \langle g \rangle.$$

**Beweis:** Der Beweis geht analog zu den Beweisen der Bandhierarchien: die Diagonal-TM  $\Delta$  arbeitet auf dem Input  $w$  der Länge  $n$  und der Form  $w = u2v$  ( $u \in \{0,1\}^*$ ) wie folgt:

1.  $\Delta$  konstruiert  $g$  und markiert  $\log g(n)$  Felder auf Band  $k+1$ .
2.  $\Delta$  kopiert  $u$  vom Inputband auf Spur 1 von Band 1.
3.  $\Delta$  antisimuliert  $M_u(w)$  auf Band  $1, \dots, k$ . Auf Band 1 wird dabei das Turingprogramm  $u$  auf der Sonderspur 1 bei jedem simulierten Schritt von  $M_u$  so verschoben, daß der Arbeitskopf immer das erste Zeichen von  $u$  liest.
4.  $\Delta$  zählt jeden seiner Schritte aus 3. auf Band  $k+1$  mit, stoppt und akzeptiert, sobald der markierte Bereich nicht ausreicht.

Reicht der markierte Bereich, antisimuliert  $\Delta$  also erfolgreich, so verwirft  $\Delta$  genau dann, wenn  $M_u$  akzeptiert. Macht  $M_u$   $f(n)$  Schritte, so benötigt  $\Delta$  zur Antisimulation  $c \cdot |u| \cdot \log \gamma_u \cdot f(n)$  Schritte für ein  $c \in \mathbb{N}$ , wenn  $\gamma_u$  die Zahl der Arbeitssymbole und Zustände von  $M_u$  ist.  $\Delta$  macht höchstens  $2g(n)$  Schritte.

Reicht der markierte Bereich nicht aus, so kann das folgende Gründe haben:

- $M_u(w)$  macht mehr als  $f(n)$  Schritte. Dann ist  $M_u$  nicht  $f$ -zeitbeschränkt oder verwirft  $w$  (schwach zeitbeschränkt).
- $c \cdot |u| \cdot \log \gamma_u \cdot f(n) \geq g(n)$ , d.h.  $n$  ist nicht groß genug.

In beiden Fällen akzeptiert  $\Delta$ . Wenn  $M_u$   $f$ -zeitbeschränkt und  $v$  genügend lang relativ zu  $u$ , so akzeptiert  $\Delta(w)$  gdw.  $M_u(w)$  verwirft. Damit ist  $L(\Delta) \neq L(M_u)$ . ■

Schreibt man den Zähler nicht auf Band  $k+1$ , sondern auch auf Band 1 (in eine weitere Spur), und verschiebt man ihn wie  $u$  bei jedem simulierten Schritt von  $M_u$ , so daß sein letztes Feld unter dem Arbeitskopf steht, so spart man ein Band, benötigt aber zum Verschieben  $\log g(n)$  Zeit. Damit bekommt man folgende Variante des Satzes.

##### Satz 8.2 (Zeithierarchie 2)

Seien  $f, g$  totale Funktionen größer  $c \cdot \text{id}$  für ein  $c \in \mathbb{N}$ ,  $g$   $2\text{DT}^k$ -zeitkonstruierbar und  $k \geq 2$ . Dann gilt:

$$f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^k \langle g \log g \rangle.$$

##### Aufgabe 8.3

Zeigen Sie: Seien  $f, g$  totale Funktionen größer  $c \cdot \text{id}$  für ein  $c \in \mathbb{N}$ ,  $g$   $2\text{DT}^k$ -zeitkonstruierbar und  $k \geq 2$ . Dann gilt:

$$f \cdot \log f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^k \langle g \rangle.$$

Mithilfe des Satzes 4.2 über den zeitlichen Mehraufwand, wenn man  $k$  Bänder auf 2 reduziert, bekommen wir sogar einen schärferen Satz:

**Satz 8.4 (Zeithierarchie 2a)**

Seien  $f, g$  totale Funktionen größer  $c \cdot \text{id}$  für ein  $c \in \mathbb{N}$ ,  $g$   $2\text{DT}^k$ -zeitkonstruierbar und  $k \geq 2$ . Dann gilt:

$$f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^2 \langle g \log g \rangle.$$

Die Frage bleibt, ob man die Zeithierarchie genauso „dicht“ bekommt, wie die Bandhierarchie, ob man also den Faktor  $\log f$  wegbekommt. Das läuft auf die Frage hinaus, ob man den Zähler günstiger realisieren kann.

**8.1.2 Der Satz von Paul**

Der folgende Satz kann den Faktor  $\log f$  immerhin reduzieren auf  $\log^* f$

**Satz 8.5 (Zeithierarchie 3, Paul)**

Seien  $f, g$  totale Funktionen größergleich  $c \cdot \text{id}$  für ein  $c \in \mathbb{N}$ ,  $g$   $2\text{DT}^k$ -zeitkonstruierbar und  $k \geq 2$ . Dann gilt:

$$f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^k \langle g \cdot \log^* g \rangle.$$

**Beweis:** Die Idee ist, den Zähler nicht bei jedem simulierten Schritt in seiner ganzen Länge zu verschieben, sondern nur stückweise. Wir zerlegen den Zählerbereich  $z$  in Blöcke  $z = z_s \cdots z_1$  mit  $|z_1| = 16$ ,  $|z_{i+1}| = 2^{|z_i \cdots z_1|} - |z_i \cdots z_1|$ . Damit ist die Länge  $|z_i \cdots z_1| = 2^{\dots^2}$   $\}_{-i+3\text{-mal}}$  und  $s + 4 = \log^* g$ , wenn wir bis  $g$  zählen. Block  $z_{i+1}$  benötigen wir höchstens alle  $2^{|z_i \cdots z_1|} < 2^{|z_{i+1}|}$  Schritte. Dann verschieben wir ihn um  $|z_{i+1}|$  Felder. Die Kosten um zu dem Block zu laufen, um ihn auf Band 2 zu kopieren, um die  $|z_{i+1}|$  Felder zu laufen und ihn wieder auf Band 1 zurückzukopieren, sind jeweils höchstens  $2^{|z_i \cdots z_1|}$  Schritte, zusammen also höchstens  $4 \cdot 2^{|z_i \cdots z_1|} < 8^{|z_{i+1}|}$  Schritte. Der Grund dafür, daß wir wenigstens 2 Bänder voraussetzen, liegt in der Möglichkeit, den Zähler durch Kopieren schnell verschieben zu können. Damit kostet das Zählen bis  $g$  mit Verschieben der Blöcke höchstens  $\sum_{i=1}^s \frac{g}{2^{|z_i|}} \cdot 8 \cdot |z_i| < 4 \cdot g \cdot \log^* g$  Schritte.

Damit die Blöcke nicht ineinander geschoben werden, verschieben wir Block  $z_i$ , so daß er danach  $3 \cdot |z_i|$  Felder links vom Arbeitskopf beginnt. Dann ist der Abstand zwischen  $z_{i+1}$  und  $z_i$  gerade  $2^{|z_{i+1}|} - 3^{|z_i|} = 2(2^{|z_i \cdots z_1|} - |z_i \cdots z_1|) - 3^{|z_i|} > 2 \cdot 2^{|z_i \cdots z_1|} - 5^{|z_i \cdots z_1|}$ . Dieser Abstand verringert sich bis zum nächsten Verschieben von  $z_{i+1}$  um höchstens  $2^{|z_i \cdots z_1|}$ , bleibt also immer größer Null. ■

Man kann diesen Satz auch umformulieren:

**Satz 8.6**

Seien  $f, g$  totale Funktionen,  $f, g \geq \text{id}$ ,  $g$   $2\text{DT}^k$ -zeitkonstruierbar und  $k \geq 2$ . Dann gilt:

$$f \cdot \log^* f \in o(g) \implies 2\text{DT}^k \langle f \rangle \subsetneq 2\text{DT}^k \langle g \rangle.$$

**Beweis:** Folgt aus vorigem Satz vermöge:

1. mit  $g$  ist auch  $\frac{g}{\log^* g}$   $2\text{DT}^k$ -zeitkonstruierbar.
2.  $f \log^* f \in o(g) \implies f \in o\left(\frac{g}{\log^* g}\right)$ , weil
  - ist  $g \leq f^2$ , so ist  $\log^* g \leq \log^* f^2 \leq 2 \log^* f$  und somit  $f \log^* g \leq 2f \log^* f \leq g$
  - ist  $g > f^2$ , so ist  $f \leq \sqrt{g} \leq \frac{g}{\log^* g}$ .

■

8.1.3 Der Satz von Führer

**Satz 8.7 (Zeithierarchie 4, Führer)**

Seien  $f, g$  totale Funktionen,  $f, g \geq \text{id}$ ,  $g$   $2DT^k$ -zeitkonstruierbar,  $f \in o(g)$  und  $k \geq 2$ . Dann gilt

$$2DT^k \langle f \rangle \subsetneq 2DT^k \langle g \rangle.$$

**Beweis:** Wieder betrachten wir nur, wie der Zähler kostengünstig gestaltet werden kann. Der Nachteil des Zählens mit einem konventionellen binären Zähler ist, daß er ein Feld niedrigster Stelligkeit enthält, das bei jedem Zählschritt angelaufen werden muß (indem wir hinlaufen bzw. den Zähler oder Teile von ihm durch Verschieben in der Nähe des Arbeitskopfes halten). Die Idee ist nun einen Zähler zu schaffen, der nicht nur eines, sondern viele Felder enthält, die eine Ziffer mit niedrigster Stelligkeit enthalten. Das hat den Vorteil, daß ein solches Feld schneller anzulaufen ist, solange sich der Arbeitskopf im Bereich des Zählers aufhält. Entsprechend muß es auch mehrere, wenn auch weniger Felder mit der nächsten Stelligkeit geben usw.

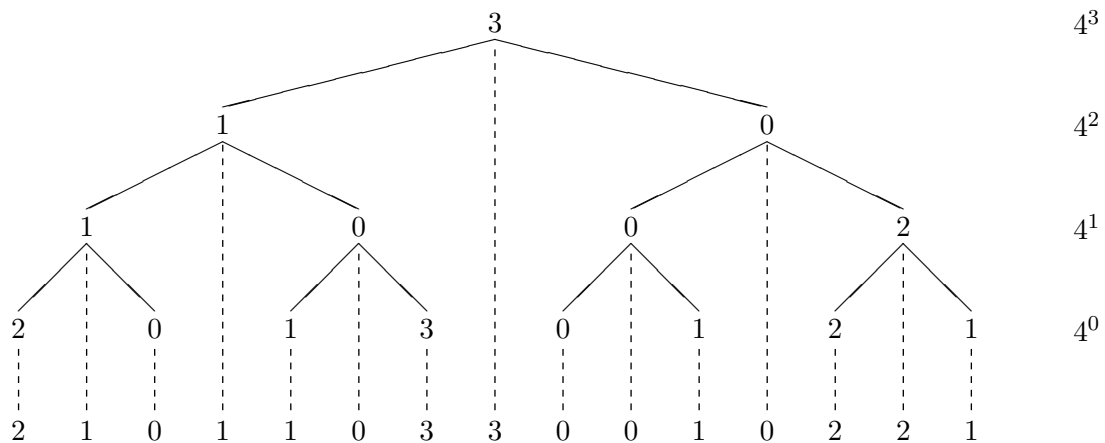
**Baumzähler:** Wir definieren also einen solchen Zähler, den wir einen quaternären Baumzähler nennen wollen: Wir stellen uns einen orientierten, gerichteten, knotenbewerteten, vollständigen, binären Baum vor, dessen Knoten quaternäre Ziffern aus  $\{0, 1, 2, 3\}$  enthalten und zwar die Blätter Ziffern der niedrigsten Stelligkeit, die mit  $4^0$  multipliziert werden, deren Väter Ziffern mit der nächsten Stelligkeit (diese werden mit  $4^1$  multipliziert), deren Väter wieder Ziffern mit der Stelligkeit 2, usw. Die Ziffern in Knoten der Höhe  $i$  haben also die Stelligkeit  $i$ , werden also mit  $4^i$  multipliziert.

Diesen Baum stellen wir als String dar, indem wir die Ebenen des Baum ineinanderschieben: ein Symbol an einer ungeraden Stelle des Wortes ( $1 \bmod 2$ ) steht für die Ziffer eines Blattes (Stelligkeit 0), ein Symbol an einer Stelle  $2 \bmod 4$  im Wort steht für eine Ziffer der Stelligkeit 1, ein Symbol an einer Stelle  $4 \bmod 8$  im Wort steht für eine Ziffer der Stelligkeit 2, allgemein: ein Symbol an einer Stelle  $2^i \bmod 2^{i+1}$  im Wort steht für eine Ziffer der Stelligkeit  $i$ . Benachbarte Ziffern der Stelligkeit  $i$ , die mit  $4^i$  zu multiplizieren sind, stehen  $2^{i+1}$  Felder auseinander.

Das Wort

2 1 0 1 1 0 3 3 0 0 1 0 2 2 1

steht also für den Baum



der folgende die Zahl repräsentiert:

$$(2 + 0 + 1 + 3 + 0 + 1 + 2 + 1) 4^0 + (1 + 0 + 0 + 2) 4^1 + (1 + 0) 4^2 + 3 \cdot 4^3 = 230.$$

Allgemein steht also das Wort  $z_1 \cdots z_l \in \{0, 1, 2, 3\}^l$  für einen Baum der Höhe  $h$  mit  $l = 2^{h+1} - 1$  und repräsentiert folgende Zahl:

$$\begin{aligned}
 & 4^0 (z_1 + z_3 + \dots + z_l) &= 4^0 \sum \{z_i \mid i = 1 + 2j, j = 0, \dots, \frac{l-1}{2}\} \\
 + & 4^1 (z_2 + z_6 + z_{10} + \dots + z_{l-1}) &= 4^1 \sum \{z_i \mid i = 2 + 4j, j = 0, \dots, \frac{l-1}{4}\} \\
 + & 4^2 (z_4 + z_{12} + \dots + z_{l-4}) &= 4^2 \sum \{z_i \mid i = 4 + 8j, j = 0, \dots, \frac{l-1}{8}\} \\
 & \vdots & \vdots \\
 + & 4^i (z_{2^i} + z_{2^i+2^{i+1}} + \dots + z_{l-2^{i+1}}) &= 4^i \sum \{z_i \mid i = 2^i + 2^{i+1}j, j = 0, \dots, \frac{l-1}{2^{i+1}}\} \\
 & \vdots & \vdots \\
 + & 4^h (z_{2^h}) &
 \end{aligned}$$

Solange der Arbeitskopf sich im Bereich der Darstellung dieses Zähler befindet, kann er in einem Schritt eine Ziffer der Stelligkeit 0 erreichen und in  $2^i$  Schritten eine Ziffer der Stelligkeit  $i$ . Damit der Arbeitskopf in der richtigen Richtung geht, wenn er von einer Ziffer der Stelligkeit  $i$  zur nächsten der Stelligkeit  $i + 1$  laufen will (z.B. weil sie 0 ist, und eine 1 abgezogen werden soll), wollen wir in eine zweite Spur unter jede Ziffer eine Richtungsmarke aus  $\{R, L\}$  schreiben, die angibt, in welcher Richtung die nächste höherwertige Ziffer zu finden ist.

Zu Anfang konstruiert die Diagonalmaschine  $\Delta$  den Funktionswert  $g(n)$  und einen Anfangsbaum, der eine Zahl in der Größenordnung  $g(n)$  repräsentiert (im Folgenden  $g := g(n)$ ).

### I. Konstruktion des Anfangsbaums.

Die Länge der Quaternärdarstellung von  $g$  ist  $|\text{quat}(g)| = \lfloor \log_4 g \rfloor + 1 =: k$ , d.h.  $4^k \leq g < 4^{k+1}$ . ( $\text{quat}(4^k - 1) = (3^k)_{\text{quat}}$ , String mit  $k$  Dreien).

Algorithmus:

1. konstruiere  $g$  (Aufwand:  $g$ ),
2. zähle 4-när bis  $g$  (belegt  $k$  Felder) (Aufwand:  $3g$ ),
3. bedrucke  $2^k - 1$  Felder mit 3 (Aufwand:  $\sqrt{g}$ )
4. Setze R,L-Labels in zweite Spur: laufe  $(k + 1)$ -mal über diesen 3-Block und drucke jeweils in Spur 2 auf jedes 2. noch unbedruckte Feld abwechselnd erst  $R$  dann  $L$ . Das Mittelfeld bekommt das Zeichen  $T$  für Top. (Aufwand:  $< k2^k \leq \log \sqrt{g} \cdot \sqrt{g}$ ).

Der gesamte Zeitaufwand dieses Algorithmus ist also  $4g + (1 + \log \sqrt{g})\sqrt{g} \in O(g)$ .

- Die Länge der Baumdarstellung ist  $2^k - 1 \leq \sqrt{2^{2k}} = \sqrt{4^k} \leq \sqrt{g}$ .
- Die Höhe des Baums ist  $h = k - 1$  ( $k = h + 1$ ).
- Der maximale Wert eines Pfades ist  $\pi_h = (3^{h+1})_{\text{quat}} = 4^k - 1 \geq \frac{1}{4}g$ .
- Der maximale Wert des ganzen Baums ist  $B_h = 6 \cdot 4^h - 3 \cdot 2^h = 6 \cdot 4^{k-1} - 3 \cdot 2^{k-1} \leq 2g$ .  
 [Beweis durch Induktion:  $B_1 = 3 \cdot 4^1 + 3 + 3 = 3 \cdot 4 + 3 \cdot 2 = 6 \cdot 4 - 3 \cdot 2$  und  $B_{h+1} = 3 \cdot 4^{h+1} + 2(6 \cdot 4^h - 3 \cdot 2^h) = 6 \cdot 4^{h+1} - 3 \cdot 2^{h+1}$ ].

### II. Das Zählen (Operation $-1$ )

Der Baumzähler ist zu Anfang gefüllt. Die Diagonal-TM  $\Delta$  zieht bei jedem simulierten Schritt eine Eins ab. Sie stoppt, falls die oberste Ziffer an der Wurzel negativ wird, d.h. frühestens, wenn sie einen Pfad völlig entleert hat, d.h. frühestens nach  $4^k - 1$  und spätestens, wenn der ganze Baum entleert ist, d.h. spätestens nach  $6 \cdot 4^{k-1} - 3 \cdot 2^{k-1}$  Schritten.

procedure MinusEins:

1. markiere Kopffosition und suche die nächste  $4^0$ -Stelle (Blatt);  $r := 0$ ; gehe zu (2).
2. Ist Ziffer = 0 und Richtungsmarke =  $T$ , so sind  $\geq \frac{g}{4}$  Schritte getan, Stop.  
 Ist Ziffer = 0 und Richtungsmarke  $\neq T$ , so ersetze Ziffer durch 3 und gehe  $2^r$  Schritte in die

angegebene Richtung;  $r := r + 1$ ; gehe zu (2).

Ist Ziffer  $\neq 0$ , so ziehe 1 von ihr ab und gehe zum markierten Feld zurück, Stop.

Der temporäre Zähler  $r$  hat maximal den Wert  $h$  und wird auf dem zweiten Band geführt.

*Aufwand des Zählens:* Die Zeit um von Schicht  $i - 1$  in Schicht  $i$  zu steigen ( $4^i$ -Stellen) ist  $2^{i-1}$  (Zähler  $r$  steht auf Band 2). Dies geschieht allerdings höchstens alle  $4^i$  Schritte (jedesmal, wenn eine  $4^i$ -Stelle angelaufen wird, müssen hierfür erst  $4^i$  Schritte getan sein), d.h. bis zum Entleeren der Wurzel höchstens  $\frac{6 \cdot 4^h - 3 \cdot 2^h}{4^i} < 6 \cdot 4^{h-i}$  mal. Damit ist die Gesamtzeit des Zählers ist also höchstens  $\sum_{i=0}^h 6 \cdot 4^{h-i} 2^{i-1} \leq 3 \cdot 4^h \sum_{i=0}^h \frac{1}{2^i} \leq 6 \cdot 4^h \leq 6 \cdot 4^{k-1} \leq 1,5 \cdot 4^k < 4g$ .

### III. Verschieben des Zählers

Immer wenn der Arbeitskopf den Zählerbereich (Block von  $2^k - 1$  Felder) verläßt, wird der Zähler neu zentriert (um  $2^{k-1}$  Felder verschoben, so daß die Wurzel unter dem Arbeitsfeld steht). Das kostet weniger als  $3 \cdot 2^k$  Schritte (mithilfe von Band 2) und kommt höchstens alle  $2^h$  Schritte vor. Die Zeit des Verschiebens ist also  $\leq \frac{6 \cdot 4^{k-1}}{2^{k-1}} \cdot 3 \cdot 2^k = 9 \cdot 4^k \leq 9g$ . ■

#### Aufgabe 8.8

1. Sei  $B$  ein binärer Baumzähler der Höhe  $h$  ganz mit der Ziffer 1 ausgelegt. Dann enthielte ein Pfad die Zahl  $2^{h+1} - 1$ , der ganze Baum aber die Zahl  $(h + 1) \cdot 2^h$ .

2. Sei  $B$  ein  $p$ -ärer Baumzähler mit Verzweigungsgrad  $v$  und den Ziffern  $\{0, \dots, p - 1\}$  und sei  $B$  ganz mit der Ziffer  $p - 1$  ausgelegt, so enthielte ein Pfad die Zahl  $\frac{p^{h+1}-1}{p-1}$ , der ganze Baum die Zahl  $z = (p - 1) \sum_{i=0}^h v^i p^{h-i}$ . Ist  $v = p$ , so ist  $z = (p - 1)(h + 1)p^h$ . Ist  $v < p$ , so ist  $z < (p - 1)h^3 p^h$ .

[Hinweis: Sei  $v = (1 - \varepsilon) p$ . Es ist  $(1 - \varepsilon)^k < 1 - k\varepsilon + \frac{k(k-1)}{2} \varepsilon^2$ . Also ist dann  $z = (p - 1) \sum_{i=0}^h v^i p^{h-i} = (p - 1) \sum_{i=0}^h (1 - \varepsilon)^i p^i p^{h-i} = (p - 1) p^h \sum_{i=0}^h (1 - \varepsilon)^i$ . Nun ist  $\sum_{i=0}^h (1 - \varepsilon)^i < (h + 1)(\varepsilon + \varepsilon^2 + \frac{1}{2}h(h - 1))$  (zeigen!). Also ist  $z < (p - 1)(h + 1)p^h(\varepsilon + \varepsilon^2 + \frac{1}{2}h(h - 1))$ .]

#### Aufgabe 8.9

Eine  $2DT^2\langle t \rangle$ -TM ist eine deterministische two-way TM mit 2 Turingbändern, die t-zeitbeschränkt ist. Eine  $2DT^2C^l\langle t \rangle$ -TM ist eine  $2DT^2\langle t \rangle$ -TM mit noch weiteren  $l$  Bändern, die Zählerbänder (Counter). Ein Zählerband genügt folgenden Bedingung:

1. das Arbeitsalphabet des Bandes hat nur ein Symbol (es darf immer nur derselbe Buchstabe gedruckt werden),
2. wenn der Kopf nach links geht, so muß er löschen ( $\square$  drucken),
3. wenn der Kopf nach rechts geht, so muß er das Symbol ( $\neq \square$ ) drucken.

Zeigen Sie mithilfe der obigen Beweisidee, daß man beliebig viele Zähler ohne Zeitverlust weglassen kann, wenn man 2 Turingbänder hat, genauer:

$$2DT^2C^l\langle t \rangle = 2DT^2\langle t \rangle, \text{ falls } t > id.$$

[Hinweis: Zeigen Sie die Behauptung zunächst nur für einen Zähler. Er wird repräsentiert durch einen normalen, üblichen Quaternärzähler  $Z$  auf Band 1, und einen quaternären Baumzähler  $B$  auf Band 2.

1. Erstelle Baumzähler:  
Im üblichen Quaternärzähler  $Z$  stehe die Zahl  $n$  (Länge  $\log_4 n$ ). Sei  $k = \max \{i \mid 16 \cdot 4^i < n\}$  (Länge des Zählers  $Z$  minus 2). Der quaternäre Baumzähler  $B$  der Tiefe  $k$  enthält eine Zahl bis  $6 \cdot 4^k - 3 \cdot 2^k < n$ . Erstelle Baumzähler  $B$  ( $2^{k+1} - 1$  Nullen) Zeit  $\sqrt{n}$ . Der Baumzähler habe die Ziffer  $-3, -2, -1, 0, 1, 2, 3$
2. Zähle mit Baumzähler:  
Dies geht in Linearzeit bis ein Pfad gefüllt ist mit 3 oder  $-3$  nach frühestens  $4 \cdot 4^k$  Schritten. Dann:
3. Reorganisiere Baumzähler:  
Überlaufe den Baumzähler von links nach rechts und addiere die Ziffer entsprechend ihrer Stelligkeit auf  $Z$ . Die Folge der Stelligkeit ist: 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 ...

Um dies auf die richtige Weise zu tun, habe ich unter dem Zähler  $Z$  eine 2. Spur mit Marken  $*$ . Die Markenstrategie ist folgende:

Anfang: Kopf steht auf der letzten Ziffer von  $Z$  (keine Marke)

Addition: Addiere die im Baumzähler gelesene Ziffer an die Stelle von  $Z$ , an der der Kopf steht, und setze dort eine Marke. Suche das rechteste Feld ohne Marke und lösche alle Marken rechts davon (Zeit:  $|Z| = \log n$ ).

$2^{k+1} - 1$  Additionen (Zeit  $\leq \log n \cdot \sqrt{n}$ ). Jetzt erstellen wir einen neuen Baumzähler und 1). Dann holen wir  $Z$  an die Stelle des aktuellen Arbeitsfeldes (Zeit  $2 \cdot 4 \cdot 4^k + 2 \log n \in o(n)$ .)

### 8.1.4 Nichtdeterministische Zeithierarchie

Auch im nichtdeterministischen Fall kann man Zeithierarchien beweisen, allerdings nicht so feine, wie uns das bei der Bandkomplexität gelungen ist. Hier fehlt so etwas wie der Satz von Immerman. Aber dieselbe Diagonalisierung, die wir im Satz 7.13 von J.I.Seiferas, M.J.Fischer und A.R.Meyer 1973 verwandt haben, kann auch für eine nichtdeterministische Zeithierarchie verwandt werden.

#### Satz 8.10 (Nondeterministische Zeithierarchie two-way )

Seien  $f$  und  $g$  arithmetische Funktionen,  $g$  2DT-zeitkonstruierbar,  $g > id$  (suc = Nachfolgerfunktion). Dann gilt:

$$f \circ \text{suc} \in o(g) \implies \text{strong-2NT} \langle f \rangle \subsetneq \text{strong-2NT} \langle g \rangle.$$

**Beweis:** Sei  $(M_i)$  eine Aufzählung der 2NT-TMen,  $L$  eine entscheidbare Sprache, die für kein  $k$  in  $2\text{NT} \langle g(n+k) \rangle$  liegt und  $M$  eine immer haltende TM für  $L$ . Wir definieren jetzt folgende  $2\text{NT} \langle g \rangle$ -TM  $N$ :

Den String  $0^i x 0^p$  mit  $x \in \{1, 2\}^*$  wollen wir abkürzen durch  $(i, x, p)$ .  $N$  angewandt auf  $(i, x, p)$  konstruiert  $g(|i, x, p|)$  und simuliert  $M(x)$  falls die konstruierte Zeit reicht. Falls nicht, so simuliert  $N$  jetzt  $M_i(i, x, p+1)$ , wiederum falls die konstruierte Zeit reicht. Reicht sie auch hier nicht, so stoppt  $N$  verwerfend. Etwas salopp schreiben wir das so:

$$N(i, x, p) = \begin{cases} M(x), & \text{falls Zeit } g(|i, x, p|) \text{ zur Simulation reicht, sonst} \\ M_i(i, x, p+1), & \text{falls Zeit } g(|i, x, p|) \text{ zur Simulation reicht, sonst} \\ 0 & \end{cases}$$

( $M(x) = 0$  bzw.  $1$  bedeutet:  $M$  angewandt auf  $x$  verwirft bzw. akzeptiert).

Offensichtlich ist  $N$  eine  $\text{strong-2NT} \langle g \rangle$ -TM, die immer hält. Annahme, der Satz sei falsch, d.h.  $\text{strong-2NT} \langle g \rangle \subseteq \text{strong-2NT} \langle f \rangle$ . Dann gibt es eine (immer haltende)  $\text{strong-2NT} \langle f \rangle$ -TM  $M_k$ , die zu  $N$  äquivalent ist, d.h. dieselbe Sprache erkennt. Beachten Sie, daß die TM  $M$ ,  $N$  und  $M_k$  immer halten. Nun gilt:

- Für alle  $x$  gibt es ein erstes  $p_x$ , so daß  $N(k, x, p_x) = M(x)$ .
- Zur Simulation von  $M_k(k, x, p+1)$  benötigt  $N$  möglicherweise  $c_k f(|k, x, p+1|)$  Zeit für eine von  $M_k$  abhängige Konstante  $c_k$ .
- Es gilt:  $\exists n_0 \forall p \geq 0 \forall x (|x| \geq n_0) : g(|k, x, p|) \geq c_k f(|k, x, p+1|)$ . Also verfügt  $N(k, x, p)$  bei Worten  $x$  einer Länge  $|x| \geq n_0$  über genügend Zeit, um  $M_k(k, x, p+1)$  erfolgreich zu simulieren. Damit ist  $M_k(k, x, p) = N(k, x, p) = M_k(k, x, p+1)$ , falls  $p < p_x$ .
- Also gilt für alle  $x$  mit  $|x| \geq n_0$ :

$$M_k(k, x) = M_k(k, x, 1) = M_k(k, x, 2) = \dots = M_k(k, x, p_x) = M(x),$$

d.h.  $M_k(k, x)$  akzeptiert gdw.  $M(x)$  akzeptiert.

- Um  $L$  zu erkennen, können wir also eine TM  $M'$  konstruieren, deren Programm aus dem Programm von  $M_k$  besteht, vor das wir ein Programm setzen, das  $0^k$  vor den Input schreibt.  $M'$  ist dann eine  $2\text{NT} \langle g(n+k) \rangle$ -TM, also  $L \in 2\text{NT} \langle g(n+k) \rangle$  im Widerspruch zur Annahme.

■

## 8.2 Unteres Zeit-Gap

Wir wollen hier ein paar Worte über das untere Ende der Zeithierarchien verwenden. Haben wir mehr als ein Band, so können wir bereits mit real-time nichtreguläre Sprachen erkennen, wie etwa  $L_{=} = \{a^n b^n \mid n \in \mathbb{N}\}$ . Hier ist also die unterste vernünftige Zeitgrenze, die man braucht, um wenigstens den Input zu lesen, noch ausreichend, um mehr zu erkennen als Finite Automaten, die ihrerseits mit real-time auskommen. Bei 1-Band Turingmaschinen ohne zusätzliches Inputband, sieht es allerdings anders aus: steht weniger als  $n \log n$  Zeit zur Verfügung, so können nur noch reguläre Sprachen erkannt werden, wenn wir die strenge Zeitauffassung zugrunde legen. Bei der schwachen Zeitauffassung, kann man nichtdeterministisch noch mit  $n \log \log n$  Zeit eine nicht reguläre Sprache erkennen, wie wir in Satz 6.25 bereits bewiesen haben. Unter dieser Zeitschranke allerdings können auch nur noch reguläre Sprachen erkannt werden.

### Satz 8.11

Sei  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  eine  $0NT^1$ -TM und  $L$  die Sprache von  $M$ . Gibt es eine Zahl  $l$ , sodaß jedes Wort  $w$  von  $L$  einen  $w$ -Lauf hat mit Crossing Sequences von höchstens der Länge  $l$ , so ist die Sprache regulär. Genauer:

Falls  $\exists l \in \mathbb{N} \forall w \in L(M) \exists w\text{-Lauf } R \text{ akzeptierend } \forall k \in \mathbb{N} : |C(R, k)| < l$ , so ist  $L(M)$  regulär.

**Beweis:**  $M$  rät das größte nichtberührte Feld links (dort ist die CS leer). Dann rät  $M$  die Nachbar-CS rechts und überprüft gleich, ob die beiden benachbarten CS'e zusammen gehören können (kompatibel sind). War die Prüfung erfolgreich, so rät  $M$  die nächste CS, solange bis sie die leere CS  $\varepsilon$  rät und erfolgreich als kompatibel prüft. Dann existiert offenbar ein  $w$ -Lauf.

Genauer: O.B.d.A. können wir annehmen:

1.  $M$  rückt bei jedem Befehl nach links oder rechts ( $d \neq 0$ )
2.  $Q = Q_L \dot{\cup} Q_R$ , wo  $Q_L$  ( $Q_R$ ) die Menge der Zustände, die in einem Schritt von links (rechts) erreicht werden ( $\dot{\cup}$  meint die disjunkte Vereinigung. Wir nehmen also an, daß die Mengen  $Q_L$  und  $Q_R$  disjunkt gemacht sind).
3.  $M$  macht seinen letzten Schritt von rechts auf Feld 1.

Für ein Paar  $(s, r)$  von CS'e definieren wir  **$a$ -kompatibel von links (rechts):**

1. 2 leere CS'e  $(\varepsilon, \varepsilon)$  sind  $a$ -kompatibel von links und rechts ( $a \in \Sigma$ )
2. Sind  $(s, r)$   $b$ -kompatibel von links und  $(p, a, q, b, L) \in \delta$  und  $p \in Q_L$ , so ist  $(pqs, r)$   $a$ -kompatibel von links bzw.  $p \in Q_R$ , so ist  $(qs, pr)$   $a$ -kompatibel von rechts.
3. Sind  $(s, r)$   $b$ -kompatibel von rechts und  $(p, a, q, b, R) \in \delta$  und  $p \in Q_R$ , so ist  $(s, pqr)$   $a$ -kompatibel von rechts bzw.  $p \in Q_L$ , so ist  $(ps, qr)$   $a$ -kompatibel von links.

Folgender NFA  $A$  erkennt  $L(M)$  ( $w = a_1 \dots a_n$ ):  $A$  rät die rechteste CS  $s$  eines  $w$ -Laufs von  $M$  links vom Input, die nicht leer ist und überprüft, ob  $(\varepsilon, s)$   $\square$ -kompatibel von rechts. Dann rät  $A$  die rechte Nachbar-CS  $t$  zu  $s$  und überprüft, ob  $(s, t)$   $\square$ -kompatibel von rechts, usw. Irgendwann rät  $A$  die CS, die zur Grenze zwischen  $a_1$  und  $a_2$  gehört  $C_R(a_1, a_2 \dots a_n)$ . Ab dann wird die Kompatibilität von links überprüft.  $A$  akzeptiert, wenn  $A$  schließlich die erste leere CS rechts von Input geraten und erfolgreich überprüft hat. ■

### Satz 8.12

weak- $0DT^1 \langle O(n \cdot \log n) \rangle = \text{REG}$ , strong- $0NT^1 \langle O(n \cdot \log n) \rangle = \text{REG}$ .

**Beweis:** Sei  $M$   $0NT^1 \langle t \rangle$ -TM und  $L(M) \notin \text{REG}$ . Dann gibt es nach Satz 8.11 für jedes  $r \in \mathbb{N}$  ein  $w \in L(M)$  mit einem akzeptierenden  $w$ -Lauf, der eine CS einer Länge  $\geq r$  hat. Sei  $w(r)$  ein kürzestes solches Wort und  $R$  der akz. Lauf mit CS länger als  $r$ .

Dann gibt es keine 3 Stellen  $k_1, k_2, k_3$  auf dem Band im Bereich  $w(r)$  mit  $C_R(k_1) = C_R(k_2) = C_R(k_3)$ , denn sonst könnte man aus  $w(r)$  das Stück zwischen  $k_1$  und  $k_2$  oder zwischen  $k_2$  und  $k_3$  ausschneiden und bekäme ein kürzeres Wort  $w'$  und einen akzeptierenden Lauf  $R'$  für  $w'$ , der immer noch eine CS länger  $r$  hat (schneide eben das Stück heraus in dem nicht die einzige längste CS liegt).

Also muß es  $\frac{n}{2}$  verschiedene CS'e geben ( $n = |w(r)|$ ). Wenn wir annehmen, daß nur die kürzesten CS'e vorkommen, so müssen wir die CS'e bis zur Länge  $l$  betrachten, mit

$$1 + q + q^2 + \dots + q^l = \frac{q^{l+1} - 1}{q - 1} > \frac{n}{2}, \quad (q = |Q|)$$

$$\text{d.h. } q^{l+1} \geq \frac{n}{2}(q - 1) + 1, \text{ also } q^l \geq \left(1 - \frac{1}{q}\right) \frac{n}{2}$$

$$\text{Damit ist } t(n) \geq \sum_{i \leq l} iq^i \geq lq^l \geq$$

$$\geq \log_q \frac{n}{2} \cdot \left(1 - \frac{1}{q}\right) \frac{n}{2} \geq c \cdot n \cdot \log n \text{ für ein } c \in \mathbb{N}.$$

Im Fall weak- $0DT^1$  gibt es nur einen Lauf für  $w(r)$ . Im Fall strong- $0NT^1$  müssen alle Läufe von  $w(r)$  die Aufwandschranke einhalten. Wenn also einer teuer ist ( $\geq n \log n$ ), so kann die Schranke nicht darunter liegen. Da dieses Argument für jedes  $n$  gilt, welches Länge eines  $w(r)$  ist für ein  $r$ , ist  $\sup \frac{t}{n \cdot \log n} > 0$ , d.h.  $t \in \Omega(n \cdot \log n)$ . ■

Wir werden später sehen, daß man bei schwacher Komplexität sogar noch mit Zeit  $n \cdot \log \log n$  etwas nicht reguläres erkennen kann:

### Satz 8.13

weak- $0NT^1 \langle n \cdot \log \log n \rangle \notin \text{REG}$ .

**Beweis:** Die Sprache  $L_{\neq} = \{a^n b^m \mid n \neq m\}$  ist aus weak- $0NT^1 \langle n \cdot \log \log n \rangle$  wie wir bereits in Satz 6.25 bewiesen haben, aber nicht aus REG. ■