Schriften zur Informatik und angewandten Mathematik

Ein Beitrag zur Bestimmung von Rados $\Sigma(5)$

oder

Wie fängt man fleißige Biber?

von

Michael Buro

Bericht Nr. 146 November 1990

Rheinisch-Westfälische Technische Hochschule Aachen

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung von Turingmaschinen hinsichtlich der von T. Rado in [Rad62] eingeführten Funktionen Σ und S und einer weiteren Funktion σ. Gegeben ist hierbei eine bestimmte endliche Menge von Turingmaschinen mit n Arbeitszuständen über dem Arbeitsalphabet {0,1}. Gesucht werden die maximalen Anzahlen von hinterlassenen Einsen auf dem Band, von Kopfbewegungen bzw. von verschiedenen bedruckten Feldern stoppender Turingmaschinen aus jener Menge, wobei das leere Wort als Eingabe dient. T. Rado zeigte in seiner Arbeit, daß die Funktionen Σ und S nicht (turing-)berechenbar sind, es also keinen (Turingmaschinen-)Algorithmus gibt, der für beliebig vorgegebenes n die Werte $\Sigma(n)$ oder $\mathrm{S}(n)$ bestimmen kann. Offensichtlich genügt zur Bestimmung der Funktionswerte die Untersuchung aller Turingmaschinen aus der Menge hinsichtlich ihrer Stoppeigenschaft. Im allgemeinen Fall ist dieses Problem nicht entscheidbar, sonst wäre S berechenbar. Aber selbst die Entscheidung, ob eine bestimmte vorgelegte Turingmaschine stoppt oder nicht, kann mit erheblichen Schwierigkeiten verbunden sein. So fällt es nicht schwer, eine Turingmaschine zu konstruieren, die genau dann stoppt, wenn die Fermatsche Vermutung falsch ist, d.h. wenn ganze Zahlen k, x, y, z existieren mit $k > 2,\ x, y, z \neq 0$ und $x^k + y^k = z^k$.

Die allgemeine Aussage der Nichtberechenbarkeit hält nicht von dem Versuch ab, für kleine Zustandsanzahlen die Funktionswerte zu bestimmen. In [LinRad65] wurde $\Sigma(2) = 4$ und $\Sigma(3) = 6$ bestimmt. Hierbei mußte für 20.736 bzw. ca. $17 \cdot 10^6$ Turingmaschinen die Stoppeigenschaft entschieden werden. Die Arbeiten [WeiCasFen73] und [Bra75] zeigten $\Sigma(4) = 13$, wobei hier schon $2.56 \cdot 10^{10}$ Maschinen zu untersuchen waren. Offen sind die Werte für Turingmaschinen mit mehr als vier Arbeitszuständen. Im Jahr 1982 fand U. Schult [LudSchWan83] durch einen Computer-Kraftakt eine Maschine mit fünf Arbeitszuständen, die 501 Einsen auf dem Band hinterläßt, indem er eine Normalform nutzend in ca. 800 Stunden Rechenzeit die zu untersuchenden Turingmaschinen 500.000 Schritte lang simulierte. Am Ende des Jahres 1984 wurde von G. Uhing [Dew85] eine Maschine auf ähnliche Weise gefunden, die 1901 Einsen hinterläßt. Der aktuelle Stand ist $\Sigma(5) \geq 4.098$. Dieses Ergebnis ist unabhängig in [MarBun90] und bei der Erstellung der vorliegenden Arbeit gefunden worden.

In den folgenden Kapiteln werden zunächst die untersuchten Turingmaschinen und Funktionen eingeführt und grundlegende Eigenschaften dieser dargestellt. Im Anschluß daran werden Methoden beschrieben, die es ermöglichen, die Menge der hinsichtlich ihrer Stoppeigenschaft zu entscheidenden Turingmaschinen zu verkleinern. Den Schwerpunkt bilden zwei Kapitel über Nichtterminationsbeweise. Dort werden für drei Turingmaschinen-Typen Verfahren entwickelt, mit deren Hilfe bewiesen werden kann, daß eine vorgelegte Maschine nicht stoppt. Abschließend wird auf die Anwendung der dargestellten Ansätze auf die Menge der Turingmaschinen mit fünf Arbeitszuständen eingegangen.

2 Grundlagen

Zu Beginn werden die betrachteten Funktionen und einige Notationen eingeführt. Danach wird gezeigt, daß die Funktionen nicht berechenbar sind und sich ihre Werte auch auf eine größere Menge von Turingmaschinen fortsetzen lassen. Im Anschluß daran wird eine erste Möglichkeit aufgezeigt, wie die Anzahl der zur Bestimmung der Funktionswerte zu untersuchenden Turingmaschinen verkleinert werden kann. Eine Konstruktion von Turingmaschinen, die hohe untere Schranken für die betrachteten Werte liefern, beendet das Kapitel.

2.1 Einführende Definitionen und Bezeichnungen

Definition 2.1

Sei M= $(\Lambda, \Gamma, Q, F, \delta, q_0)$ eine Turingmaschine mit

- Eingabealphabet $\Lambda = \{1\}$,
- Arbeitsalphabet $\Gamma = \{0, 1\}$,
- Zustandsmenge $Q = \{0, 1, \ldots, n-1, \mathrm{H}\}$ für ein $n \in \mathbb{N}$,
- Endzustandsmenge $F = \{\mathrm{H}\}$,
- totaler Übergangsfunktion $\delta : Q \setminus F \times \Gamma \to \Gamma \times \{\mathrm{L},\mathrm{R}\} \times Q$ und
- Startzustand $q_0 = 0$.

Dann ist $\underline{\mathcal{TM}_n}$ die Menge aller obigen (binären) Turingmaschinen M mit $n+1$ Zuständen und $\underline{\mathcal{TM}}$ die Vereinigung aller $\mathcal{TM}_n$.
In Abhängigkeit davon, ob eine betrachtete Turingmaschine M bei leerem Wort (ϵ) als Eingabe stoppt (d.h. in der Berechnung der Zustand H erreicht wird) oder nicht, sind die Funktionen $\underline{\Sigma}$, $\underline{\mathrm{S}}$ und $\underline{\sigma}$ folgendermaßen definiert:

$$\Sigma(\mathrm{M}) := \begin{cases} k, & \text{falls M stoppt und } k \text{ Einsen hinterläßt;} \\ 0, & \text{sonst.} \end{cases}$$

$$\mathrm{S}(\mathrm{M}) := \begin{cases} k, & \text{falls M stoppt und } k\text{-mal den Kopf bewegt;} \\ 0, & \text{sonst.} \end{cases}$$

$$\sigma(\mathrm{M}) := \begin{cases} k, & \text{falls M stoppt und } k \text{ verschiedene Felder bedruckt;} \\ 0, & \text{sonst.} \end{cases}$$

Schließlich geben die Funktionen auch Auskunft über die ganze Menge $\mathcal{TM}_n$:

$$\begin{aligned} \Sigma(n) &:= \max\{\Sigma(\mathrm{M}) \mid \mathrm{M} \in \mathcal{TM}_n\} \\ \mathrm{S}(n) &:= \max\{\mathrm{S}(\mathrm{M}) \mid \mathrm{M} \in \mathcal{TM}_n\} \\ \sigma(n) &:= \max\{\sigma(\mathrm{M}) \mid \mathrm{M} \in \mathcal{TM}_n\} \end{aligned}$$

Eine Turingmaschine $\mathrm{M} \in \mathcal{TM}_n$ heißt <u>fleißiger Biber</u> $:\Leftrightarrow \Sigma(\mathrm{M}) = \Sigma(n)$.

Bemerkung 2.2

Es gelten folgende drei Beziehungen:

1. $\forall n \in \mathbb{N} : \mid \mathcal{TM}_n \mid = [4(n+1)]^{2n}$
2. $\forall n \in \mathbb{N} : \ \Sigma(n) \leq \sigma(n) \leq \mathrm{S}(n)$
3. $\forall n \in \mathbb{N} : \ \mathrm{S}(n) \leq 2^{\sigma(n)} n\, \sigma(n)$

Beweis:

1. Pro Eintrag in der Turingtafel (Zeichen, Richtung, Zielzustand) gibt es $2 \cdot 2 \cdot (n+1) = 4 \cdot (n+1)$ Möglichkeiten und es gibt $2n$ Einträge.
2. Aus der Definition folgt für alle $\mathrm{M} \in \mathcal{TM}_n$ direkt $\Sigma(\mathrm{M}) \leq \sigma(\mathrm{M}) \leq \mathrm{S}(\mathrm{M})$. Durch Maximumbildung erst rechts und dann links in den jeweiligen Ungleichungen folgt die Behauptung.
3. Sei M eine stoppende Turingmaschine aus $\mathcal{TM}_n$. Bevor M stoppt, hat sie höchstens $\sigma(n)$ verschiedene Felder bedruckt. Wenn der Haltzustand in der Berechnung noch nicht erreicht worden ist, dann können höchstens $2^{\sigma(n)} n\, \sigma(n) - 1$ Transitionen (d.h. Zustandsübergänge) erfolgt sein, denn die Anzahl verschiedener Konfigurationen ist nach oben durch

   Anzahl der Belegung der maximal $\sigma(n)$ Felder
   $\cdot$ Mächtigkeit der Zustandsmenge (ohne Haltzustand)
   $\cdot$ Anzahl der möglichen Kopfpositionen

   beschränkt. Sind mehr als $2^{\sigma(n)} n\, \sigma(n) - 1$ Transitionen erfolgt, so kam es zu mehr als $2^{\sigma(n)} n\, \sigma(n)$ Konfigurationen. Damit muß eine Konfigurationswiederholung vorliegen und M kann nicht stoppen. Zählt man jetzt noch die Transition in den Haltzustand hinzu und bemerkt, daß jede Transition mit einer Kopfbewegung verbunden ist, so ergibt sich $\mathrm{S}(\mathrm{M}) \leq 2^{\sigma(n)} n\, \sigma(n)$ für alle stoppenden $\mathrm{M} \in \mathcal{TM}_n$. Durch Maximumbildung folgt die gewünschte Ungleichung.

$\diamond$

Im folgenden werden Turingmaschinen aus $\mathcal{TM}$ durch ihre Übergangsfunktion in Gestalt einer speziellen Turingtafel angegeben – beispielsweise

	0	1
0	1R1	1RH
1	0L1	1L0

.

Zu jedem Arbeitszustand gehört eine Zeile. Dort gibt es zwei Einträge, die angeben, was bei Lesen einer 0 bzw. 1 geschehen soll. Der Eintrag in der 2. Zeile und 1. Spalte zeigt z.B. an, daß eine 0 geschrieben, der Kopf nach links bewegt und in Zustand 1 fortgefahren werden soll.

Konfigurationen werden in einer anschaulichen Zellennotation dargestellt:

- Durch Zellen der Form [X] werden endliche Zeichenfolgen auf dem Band zusammengefaßt. Zellen sind also Wörter über dem Arbeitsalphabet; daher können die üblichen Sprachformalismen benutzt werden. Es soll auch möglich sein, daß Zellennamen den Inhalt der Zelle beschreiben, wie z.B. [01101] , und daß Zellen mit ihrem Namen identifiziert werden. Durch den Kontext wird die jeweilige Beschreibungsart deutlich.
- len [X] bezeichnet die Anzahl der Zeichen in der Zelle X.
- [X^i] mit $i \in \mathbb{N}_0$ ist eine Abkürzung für eine Kette von i X-Zellen.
- [$^\omega 0$] bzw. [0^ω] bezeichnen eine unendliche Folge von Nullen (aus dem Arbeitsalphabet) nach links bzw. rechts.
- [Y] zeigt an, daß sich der Kopf der Turingmaschine auf dem Zeichen am rechten Rand $\underset{\mathrm{q}\bullet}{}$ der Y-Zelle befindet und als nächstes Zustand q bearbeitet wird.

Operationen einer Turingmaschine M auf Zellen werden durch Regeln der folgenden Art beschrieben:

- Die Regel

  [A][B] $\vdash_{\mathrm{M}}^{*}$ [A'][B']
  $\bullet q_1$ $\bullet q_2$

  gibt an, daß eine (endliche) Berechnung von M existiert, die am rechten Rand der Zelle A in Zustand q_1 beginnend A und B in A' bzw. B' überführt und am linken Rand der Zelle B in Zustand q_2 endet, ohne die beiden Zellen verlassen zu haben.

- Die Regel

  [A] $\vdash_{\mathrm{M}}^{*}$ [A']$\vdash$
  $\bullet q_1$ $\bullet q_2$

  besagt, daß die Turingmaschine M am linken Rand der Zelle A im Zustand q_1 beginnend A zu A' verändert, ohne die Zelle links verlassen zu haben. A und A' haben die gleiche Länge und der Zustand beim erstmaligen Verlassen der Zelle ist q_2.

Eine typische Konfiguration in obiger Notation dargestellt ist z.B.

[$^\omega 0$][L][M'][X^i][R][0^ω] für ein $i \in \mathbb{N}_0$.
$\bullet q_1$

2.2 Sätze zur Nichtberechenbarkeit von Σ, S und σ

Der Churchschen-These folgend werden nachstehend die Begriffe Berechenbarkeit und Turingberechenbarkeit miteinander identifiziert.

Satz 2.3

Die Funktion S ist nicht berechenbar.

Beweis:

Wenn es eine Turingmaschine gäbe, die zu beliebig vorgegebenem $n \in \mathbb{N}$ die maximale Schrittzahl einer bei leerer Eingabe stoppenden Turingmaschine aus $\mathcal{TM}_n$ berechnen kann, so wäre das initiale Halteproblem für (binäre) Turingmaschinen entscheidbar. Dieses ist aber ein Widerspruch, wie bekannterweise mittels Gödelisierung und Diagonalisierung gezeigt werden kann.

$\diamond$

Satz 2.4

Die Funktion σ ist nicht berechenbar.

Beweis:

Wenn wir die Turingberechenbarkeit von σ annehmen, so ist mit Bemerkung 2.2 eine obere Schranke für die Funktion S berechenbar. Somit ist wieder das initiale Halteproblem entscheidbar – ein Widerspruch.

$\diamond$

Satz 2.5

Die Funktion Σ ist nicht berechenbar.

Beweis 1:

Wir zeigen $\forall n \in \mathbb{N}: \ \Sigma(9n) \geq \mathrm{S}(n)$.
Betrachte dazu eine Turingmaschine $\mathrm{M} \in \mathcal{TM}_n$, die angesetzt auf das leere Wort $\mathrm{S}(n)$ Schritte macht und konstruiere aus dieser eine Turingmaschine M' $\in \mathcal{TM}_{9n}$, die stoppt und mindestens $\mathrm{S}(n)$ Einsen auf dem Band hinterläßt.
Hierzu wird jeder (Nicht-Halt-) Zustand der Maschine M durch 9 Zustände ersetzt, die folgendes leisten:
Das Band wird in Tupeln verwaltet. In der ersten Komponente wird das Band von M simuliert und in der zweiten wird pro Schritt von M eine Eins erzeugt, so daß sich nach dem Stoppen von M' mindestens $\mathrm{S}(n)$ Einsen auf dem Band befinden.

Das Anfügen einer Eins in der zweiten Komponente und die Simulation des ersetzten Zustandes der Turingmaschine M kann mittels 9 Zuständen kodiert werden, wie die nachstehende Ersetzung zeigt.
Ein Zustand $q : ABC\ DEF$ mit $A, D \in \{0,1\}$, $B, E \in \{\mathrm{L}, \mathrm{R}\}$, $C, F \in \{0, \ldots, n-1, \mathrm{H}\}$ wird durch folgende 9 Zustände ersetzt:

	0	1
q_0	$0\mathrm{R}q_1$	$1\mathrm{R}q_1$
q_1	$1\mathrm{L}q_6$	$0\mathrm{R}q_2$
q_2	$0\mathrm{R}q_3$	$1\mathrm{R}q_3$
q_3	$1\mathrm{L}q_4$	$1\mathrm{R}q_2$
q_4	$0\mathrm{L}q_5$	$1\mathrm{L}q_5$

	0	1
q_5	$1\mathrm{L}q_6$	$1\mathrm{L}q_4$
q_6	ABq_7	DEq_8
q_7	$0BC_0$	$1BC_0$
q_8	$0EF_0$	$1EF_0$

($\mathrm{H}_0 := \mathrm{H}$, und 0_0 ist Startzustand von M')
Somit ist Σ nicht berechenbar, denn sonst hätten wir wieder eine berechenbare obere Schranke für S.

$\diamond$

Beweis 2 [Rad62]:

Wir zeigen, daß Σ schneller wächst als jede auf $\mathbb{N}$ definierte berechenbare Funktion. Hierzu erfolgt zunächst eine

Definition: Seien $f, g: \ \mathbb{N} \to \mathbb{N}$ gegeben. Dann definiere

$$f \triangleright g \quad :\Leftrightarrow \ \exists x_0 \in \mathbb{N}\ \forall x > x_0: \ f(x) > g(x)$$

Sei nun eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ beliebig vorgegeben. Wir definieren zu dieser Funktion die folgende Hilfsfunktion $F : \mathbb{N} \to \mathbb{N}$:

$$F(x) := \sum_{i=1}^{x} f(i) + i^2$$

F ist berechenbar (etwa durch eine Turingmaschine $\mathrm{M}_F \in \mathcal{TM}_c$) und es gelten die Ungleichungen $\forall x \in \mathbb{N}$:

$$\begin{aligned} F(x) &\geq f(x) \\ F(x) &\geq x^2 \\ F(x+1) &> F(x) \end{aligned}$$

M_F soll ohne Einschränkung folgender Konvention genügen:
Startend auf der Eins am rechten Rand einer (unär kodierten) Eingabe x stoppt M_F schließlich, und der Kopf steht unter der Eins am rechten Ende des (unär kodierten) Funktionswertes $F(x)$.
Wähle nun ein $x \in \mathbb{N}, x > 1$; dann gibt es eine Turingmaschine $\mathrm{M}^{(x)} \in \mathcal{TM}_x$, die auf das leere Band x zusammenhängende Einsen schreibt und unter der am rechten Rand stoppt.

Betrachtet man die zusammengesetzte Turingmaschine $M_F^{(x)} = M_F \circ M_F \circ M^{(x)} \in \mathcal{TM}_{x+2c}$, so wird diese zunächst x Einsen auf das leere Band schreiben, danach $F(x)$ berechnen und dann $F(F(x))$.
Damit gilt also

$$\Sigma(x+2c) \geq F(F(x)).$$

Mit $(\cdot)^2 \triangleright (\cdot) + 2c$ und $F(x) \geq x^2$ folgt

$$F(\cdot) \triangleright (\cdot) + 2c.$$

Weil F streng monoton steigend ist, hat man $F(F(\cdot)) \triangleright F((\cdot)+2c)$ und daher insgesamt

$$\Sigma((\cdot)+2c) \triangleright F((\cdot)+2c).$$

Da $F(x+2c) \geq f(x+2c)$ gilt und x frei wählbar war, ist $\Sigma \triangleright f$ bewiesen und damit die Behauptung.

$\diamond$

Bemerkung 2.6

1. Aus der Behauptung im zweiten Beweis zur Nichtberechenbarkeit von Σ und den Beziehungen aus Bemerkung 2.2 folgt direkt, daß S und σ nicht berechenbar sind. Man kann also beim Beweis der Sätze ganz auf Diagonalisierung verzichten.
2. Wachsen die Werte einer totalen Funktion Φ schneller als die einer jeden berechenbaren Funktion, so ist dieses hinreichend für die Nichtberechenbarkeit von Φ, aber natürlich nicht notwendig. Denn betrachtet man z.B. die Anzahl STOP(n) aller auf leerer Eingabe stoppenden Turingmaschinen aus $\mathcal{TM}_n$, so gilt mit Bemerkung 2.2 STOP$(n) \leq [4(n+1)]^{2n}$. Aber STOP kann nicht berechenbar sein, sonst wäre es S auch.

2.3 Erweiterung auf eine größere Menge von Turingmaschinen

Im folgenden wird gezeigt, daß $\mathcal{TM}$ erweitert werden kann, ohne daß sich die Werte der betrachteten Funktionen ändern.

Definition 2.7

Die Menge $\underline{\mathcal{TM}'}$ ist wie die Menge $\mathcal{TM}$ definiert. Der Bildbereich der Übergangsfunktion soll jetzt aber erweitert werden zu

$$\Gamma \times \{\mathrm{L,N,R}\} \times Q,$$

d.h. der Kopf muß bei einer Transition nicht mehr bewegt werden. $\underline{\Sigma'}$, $\underline{\mathrm{S}'}$ und $\underline{\sigma'}$ sind analog zu Σ, S und σ auf $\mathcal{TM}'$ definiert.

Satz 2.8

Für alle $n \in \mathbb{N}$ gilt:
Zu jeder stoppenden Turingmaschine $\mathrm{M}' \in \mathcal{TM}'_n$ gibt es eine stoppende Turingmaschine $\mathrm{M} \in \mathcal{TM}_n$ mit $\Sigma(\mathrm{M}) = \Sigma'(\mathrm{M}')$ und $\sigma(\mathrm{M}) = \sigma'(\mathrm{M}')$.

Beweis:

Gegeben sei eine stoppende Turingmaschine $\mathrm{M}' \in \mathcal{TM}'_n$.
Ohne Einschränkung existiert dann ein Eintrag in der Turingtafel von M′ mit Kopfbewegung N (sonst ist $\mathrm{M}' \in \mathcal{TM}_n$ und die Aussage ist trivialerweise richtig). Da bei Ausführung dieses Eintrages ein Zeichen aus Γ auf das Band geschrieben wird und der Kopf sich nicht bewegt, gibt es im Folgezustand keine Alternative: Dort wird der Eintrag bearbeitet, der das geschriebene Zeichen auf dem Band erwartet. Folglich kann jener Eintrag durch diesen ersetzt werden, ohne die Berechnung zu verändern (im Sinne der Veränderung auf dem Band und der Anzahl der bedruckten Felder; die Anzahl der Zustandsübergänge verringert sich natürlich). Führt man diesen Ersetzungsprozeß sukzessiv fort, so gelangt man schließlich zu einer Turingtafel, in deren Einträgen höchstens bei Erreichen des Endzustandes der Kopf nicht bewegt wird. Dieses hat aber keinen Einfluß auf die Anzahl der hinterlassenen Einsen und der bedruckten Felder. N kann also in diesem Fall ohne Einschränkung durch R ersetzt werden. Damit ist die aus M′ erhaltene Turingmaschine M aus $\mathcal{TM}_n$ mit $\Sigma(\mathrm{M}) = \Sigma'(\mathrm{M}')$ und $\sigma(\mathrm{M}) = \sigma'(\mathrm{M}')$.

$\diamond$

Folgerung 2.9

Für alle $n \in \mathbb{N}$ gilt $\Sigma(n) = \Sigma'(n)$ und $\sigma(n) = \sigma'(n)$.

Beweis:

Aus Satz 2.8 folgt $\Sigma'(n) \leq \Sigma(n)$ und $\sigma'(n) \leq \sigma(n)$ für alle $n \in \mathbb{N}$, denn

$$\begin{aligned} & \max\{\Sigma'(\mathrm{M}') \mid \mathrm{M}' \in \mathcal{TM}'_n\} \\ = \; & \max\{\Sigma(\mathrm{M}(\mathrm{M}')) \mid \mathrm{M}' \in \mathcal{TM}'_n\} \\ \leq \; & \max\{\Sigma(\mathrm{M}) \mid \mathrm{M} \in \mathcal{TM}_n\} \end{aligned}$$

(für σ und σ' analog).
Die Ungleichungen in der anderen Richtung folgen aus $\mathcal{TM}_n \subset \mathcal{TM}'_n \; \forall n \in \mathbb{N}$.

$\diamond$

In der Menge $\mathcal{TM}'$ ist eine Transition nicht notwendig mit einer Kopfbewegung verbunden. Deshalb läßt sich für S und S′ nur der folgende Satz formulieren.

Satz 2.10

Für alle $n \in \mathbb{N}$ gilt:
Zu jeder stoppenden Turingmaschine $M' \in \mathcal{TM}'_n$ gibt es eine Turingmaschine $M \in \mathcal{TM}_n$ mit $S'(M') = S(M)$ oder $S'(M') + 1 = S(M)$.

Beweis:

Die Beweisführung erfolgt analog zu der im Satz 2.8. Wird der Endzustand in M' mit einer Kopfbewegung erreicht, so gilt $S'(M') = S(M)$. Sonst erzwingt man dieses und erhält $S'(M') + 1 = S(M)$.

◇

Folgerung 2.11

Für alle $n \in \mathbb{N}$ gilt $S(n) = S'(n)$.

Beweis:

Aus Satz 2.10 folgt $S'(n) \leq S(n)\ \forall n \in \mathbb{N}$. Die Behauptung ergibt sich dann aus $\mathcal{TM}_n \subset \mathcal{TM}'_n\ \forall n \in \mathbb{N}$.

◇

Insgesamt gelten also die Σ-, S- und σ-Werte von $\mathcal{TM}_n$ auch für $\mathcal{TM}'_n$.

2.4 Eine einfache Reduktion

In diesem Abschnitt wird sich herausstellen, daß man bei der Bestimmung der Werte $\Sigma(n)$, $S(n)$ und $\sigma(n)$ nicht alle Turingmaschinen aus $\mathcal{TM}_n$ untersuchen muß.

Definition 2.12

$\underline{\mathcal{TM}^*_n}$ ist die Menge aller Turingmaschinen aus $\mathcal{TM}_n$ mit Startzustand der Form (0: 1R1 ???) für $n > 1$ und (0: 1RH ???) für $n = 1$, wobei ??? für einen beliebigen Eintrag steht. $\underline{\Sigma^*}$, $\underline{S^*}$ und $\underline{\sigma^*}$ sind analog zu Σ, S und σ auf $\mathcal{TM}^*$ definiert.

Satz 2.13

Für alle $n \in \mathbb{N}$ gilt $\Sigma(n) = \Sigma^*(n)$.

Beweis:

Für $n = 1$ ist die Behauptung wahr, denn es gilt $\Sigma(1) = 1 = \Sigma^*(1)$.
Sei also $n > 1$ und der Startzustand von der Form (0: 1R1 ???).
Ohne Einschränkung kann in dem bei Lesen einer Null bearbeitenden Eintrag des Startzustandes

- der Kopf nach rechts bewegt werden, da sich durch Vertauschen der Richtungen L und R in δ die Anzahl der hinterlassenen Einsen nicht ändert.
- der Zustand 1 angesprungen werden. Zielzustand 0 ist nicht relevant, da die Maschine dann nicht stoppen kann. Zielzustand H braucht nicht betrachtet zu werden, da für $n > 1$ $\Sigma(n) > 1$ gilt. Durch Permutation der Zustandsteilmenge $\{1, \ldots, n-1\}$ läßt sich Zielzustand 1 erreichen.
- eine Eins auf das Band geschrieben werden. Falls eine Turingmaschine $M \in \mathcal{TM}_n$ mit $\Sigma(M) = \Sigma(n)$ an dieser Stelle eine Null schreibt, so muß in der Berechnung von M ein Zustand q erreicht werden, der als erster bei Lesen einer Null eine Eins auf das Band schreibt (sonst stoppt M nach höchstens n Schritten, hinterläßt keine Eins und kann demnach keine in bezug auf Σ maximale Turingmaschine sein). Wählt man q als neuen Startzustand, so hinterläßt die so entstandene Maschine auch $\Sigma(n)$ Einsen auf dem Band und hat einen Startzustand der Form (0: 1?? ???).

◇

Satz 2.14

Für alle $n \in \mathbb{N}$ gilt

$$S^*(n) \leq S(n) \leq n - 1 + S^*(n) \quad \text{und} \quad \sigma^*(n) \leq \sigma(n) \leq n - 1 + \sigma^*(n).$$

Beweis:

$S^*(n) \leq S(n)$ und $\sigma^*(n) \leq \sigma(n)$ folgen aus $\mathcal{TM}^*_n \subset \mathcal{TM}_n$.
Wir zeigen $S(M) \leq n - 1 + S^*(n)$ für alle stoppenden $M \in \mathcal{TM}_n$.

1.Fall M hat keinen Zustand der Form (q: 1?? ???). Dann stoppt M nach höchstens n Schritten. Mit $S^*(n) \geq 1\ \forall n \in \mathbb{N}$ ist dann die Behauptung gezeigt.

2.Fall M hat mindestens einen Zustand q der Form (q: 1?? ???), der ohne Einschränkung in der Berechnung von M der erste Zustand ist, der eine Eins schreibt. Bevor q zum erstenmal erreicht wird, können höchstens $n - 1$ Schritte gemacht worden sein, bei denen Nullen geschrieben wurden (sonst stoppt M nicht). Danach fährt die Maschine fort, wie eine Turingmaschine mit Startzustand der Form (0: 1?? ???), was die Behauptung auch in diesem Fall zeigt.

Bildet man in der Behauptung das Maximum über alle stoppenden $M \in \mathcal{TM}_n$, so ist der Satz für S bewiesen. Analog beweist man ihn für σ.

◇

Satz 2.15

Für alle $n \in \mathbb{N}$, $n > 1$ genügt zur Bestimmung der Werte $\Sigma(n)$, $S(n)$ und $\sigma(n)$ die Untersuchung der Turingmaschinen aus $\mathcal{TM}_n^*$.

Beweis:

Satz 2.13 beweist die Behauptung für Σ. Um sie für S zu zeigen, betrachtet man die Menge

$$I := \{M^* \in \mathcal{TM}_n^* \mid S^*(M^*) \geq S^*(n) - n + 1\}.$$

Ist die Menge $\mathcal{TM}_n^*$ hinsichtlich S^* vollständig untersucht worden, so ist $S^*(n)$ bekannt und somit auch I. Zu einer stoppenden Turingmaschine $M^* \in \mathcal{TM}_n^*$ können nun durch Ketten-Zustands-Permutationen stoppende Turingmaschinen $M \in \mathcal{TM}_n$ gefunden werden, die hinsichtlich S mindestens soviel leisten wie M^*, indem man versucht, Zustandsketten der Form

M:	q_1:	0?q_2	???
	q_2:	0?q_3	???
	⋮	⋮	⋮
	q_i:	0?0	???
	0:	1R1	???
	⋮	⋮	⋮

zu erzeugen. Es gilt $\otimes$ $S(M) \leq S^*(M^*) + n - 1$ für diese Konstruktion, da spätestens nach $n-1$ Schritten eine Eins geschrieben werden muß und danach die Berechnungen der beiden Maschinen identisch sind. Die Funktion p liefert die Menge der so aus M^* konstruierbaren Turingmaschinen M.

$$p(M^*) := \{M \in \mathcal{TM}_n \mid \exists \text{ Ketten-Zustands-Permutation } \pi_0, \text{die } M^* \text{ in } M \text{ überführt}\}$$

Definiert man noch

$$p(I) := \bigcup_{M^* \in I} p(M^*),$$

so genügt es,

$$M \in \mathcal{TM}_n \text{ mit } S(M) = S(n) \quad \Rightarrow \quad M \in p(I)$$

zu zeigen. Dieses besagt, daß hinsichtlich S beste Turingmaschinen aus $\mathcal{TM}_n$ in der Menge $p(I)$ gefunden werden können.
Sei also $M \in \mathcal{TM}_n$ mit $S(M) = S(n)$. Da $n > 1$ vorausgesetzt ist und damit wegen $S(2) = 6 > 2 \quad S(n) > n$ gilt, muß in M ein Zustand der Form (q: 1rq' ???) vorkommen, der ohne Einschränkung als erster Zustand eine Eins auf das Band schreibt. q' ist ungleich q (sonst stoppt M nicht) und ungleich H (sonst ist $S(M) \leq n < S(n)$). Ohne

Einschränkung ist r = R. Demnach existiert eine Zustandspermutation π, unter der M in ein $M^* \in \mathcal{TM}_n^*$ überführt wird mit (unter Beachtung von $\otimes$)

$$S^*(M^*) \geq S(M) - n + 1 = S(n) - n + 1.$$

Umgekehrt existiert eine Ketten-Zustands-Permutation $\pi_0 = \pi^{-1}$, die M^* in M überführt. Somit bleibt zu zeigen, daß M^* aus I ist, d.h.

$$S^*(M^*) \geq S^*(n) - n + 1.$$

Laut Satz 2.14 ist aber $S(n) \geq S^*(n)$ und insgesamt also $M \in p(I)$.

Für σ beweist man die Behauptung analog.

◇

Beispiel 2.16

	0	1
0	1R1	1R2
1	1L4	0L3
2	1R3	1LH
3	1L4	1R4
4	0R0	0L1

hinterläßt 1.471 Einsen
nach 2.358.064 Schritten

	0	1
4	0R0	0L1
0	1R1	1R2
1	1L4	0L3
2	1R3	1LH
3	1L4	1R4

(Startzustand 4)
hinterläßt 1.471 Einsen
nach 2.358.065 Schritten

2.5 Hohe untere Schranken für $\Sigma(n)$

Um sich einen Eindruck zu verschaffen, wie schnell $\Sigma(n)$ für steigendes n wächst, wird hier ein auf M. Green [Gre64] zurückgehendes iteratives Schema zur Konstruktion von Turingmaschinen vorgestellt, die schon bei kleiner Zustandsanzahl enorme Anzahlen von Einsen auf dem Band hinterlassen. Wir betrachten Turingmaschinen M mit folgenden Eigenschaften:

- Als Eingabe akzeptiert M entweder eine Kette von $k \geq 1$ Einsen oder das leere Wort ($k = 0$), wobei der Kopf bei Start der Berechnung unter der Eins am rechten Rand stehen soll.
- Die Ausgabe ist eine Kette von Einsen der Länge $M(k)$.
- Die Position der Eins am rechten Rand der Ausgabekette soll gleich der Startposition sein.

- Der Haltzustand soll durch Lesen einer Null auf der Position rechts von der Startposition erreicht werden
- Zu keinem Zeitpunkt der Berechnung soll der Kopf rechts mehr als eine Position von der Startposition entfernt sein.

Eine einfache Turingmaschine, die die Bedingungen erfüllt, ist zum Beispiel

	0	1
0	1R1	1L0
1	0LH	1R1

Diese bezeichnen wir mit M_2, da sie zwei Arbeitszustände besitzt. Angesetzt auf eine Kette von $k \geq 0$ Einsen hängt sie am linken Ende eine Eins an und kehrt zur Startposition zurück. Wir haben also $\forall k \geq 0 : \ M_2(k) = k + 1$.

Durch Hinzufügen von zwei Zuständen ist es jetzt möglich, aus einer gegebenen Turingmaschine M_n mit obigen Eigenschaften eine neue Turingmaschine M_{n+2} , die ihrerseits die Bedingungen erfüllt, nach einem einfachen Schema zu erzeugen:

M_{n+2} :

	0	1
0	$0\mathrm{L}0_n$	1L0
0_n	M_n	
1_n	1R1	
1	0LH	$0\mathrm{L}0_n$

Angesetzt auf eine Kette von k Einsen bewegt M_{n+2} den Kopf nach links bis zur ersten Null, dann wird M_n gestartet, um $M_n(0)$ zu berechnen. Nach der Berechnung wird eine Eins von der originalen Eingabe eine Position nach links geschoben und M_n erneut gestartet – nun $M_n(1+M_n(0))$ berechnend. Dieser Vorgang wird solange wiederholt, bis alle Einsen der Eingabe abgearbeitet sind, also $(k+1)$-mal. Nachdem M_{n+2} gestoppt hat, stehen daher

$$M_{n+2}(k) = \underbrace{1 + M_n[1 + M_n[\cdots[1 + M_n(0)]\cdots]]}_{\text{von innen nach außen } M_n \ (k+1)\text{-mal anwenden}} \qquad (1)$$

Einsen auf dem Band. Etwas kürzer kann man Gleichung (1) als Funktional-Differenzen-Gleichung darstellen:

$$M_{n+2}(k) = \begin{cases} 1 + M_n(M_{n+2}(k-1)), & \text{falls } k > 0 \\ M_{n+2}(0) = 1 + M_n(0), & \text{falls } k = 0 \end{cases} \qquad (2)$$

Gleichung (2) benutzen wir jetzt, um $M_{2i}(k)$ für kleine i zu bestimmen:

- $M_2(k) = k + 1$ wie oben gesehen.

- $M_4(k) = 2k + 2$, denn mit Gleichung (2) haben wir für $k > 0$:

$$\begin{aligned} M_4(k) &= 1 + M_2(M_4(k-1)) \\ \Rightarrow M_4(k) &= M_4(k-1) + 2 \end{aligned}$$

  und $M_4(0) = 1 + M_2(0) = 2$.

- $M_6(k) = 3\,(2^{k+1} - 1)$, da für $k > 0$

$$\begin{aligned} M_6(k) &= 1 + M_4(M_6(k-1)) \\ \Rightarrow M_6(k) &= 2\,M_6(k-1) + 3 \end{aligned}$$

  und für $k = 0 \ \ M_6(0) = 3$ gilt.

- $M_8(k) = -2 + \frac{3}{2}\underbrace{A^{A^{A\ldots A^4}}}_{k-\text{mal } A}$ mit $A = \sqrt{8}$, weil für $k > 0$ folgendes gilt:

$$\begin{aligned} M_8(k) &= 1 + M_6(M_8(k-1)) \\ \Rightarrow M_8(k) &= 1 + 3\,(2^{M_8(k-1)+1} - 1) \\ \Rightarrow M_8(k) &= -2 + \tfrac{3}{2}\ 2^{M_8(k-1)+2} \end{aligned}$$

  Setzt man nun $A = 2^{3/2}$ und beachtet $M_8(0) = 4$, so folgt die Behauptung mittels vollständiger Induktion.

Startet man die Turingmaschinen M_{2i} angesetzt auf das leere Wort, so ergeben sich schlechte untere Schranken für $\Sigma(2i)$, denn aus Gleichung (2) folgt

$$\forall i \geq 1 : \ M_{2i}(0) = i.$$

Erweitert man jetzt aber M_{2i} um Zustände, die zunächst eine Kette von Einsen produzieren, so werden große Anzahlen von Einsen erzeugt. Zum Beispiel hinterläßt die Turingmaschine gegeben durch

	0	1
0	1L1	1RH
1	$1\mathrm{R}0_8$	1RH
0_8	M_8	

$M_8(2) = -2 + \frac{3}{2}\,A^{A^4} = -2 + 3 \cdot 2^{95} \approx 1.1 \cdot 10^{29}$ Einsen auf dem Band.

M.Green verbesserte seine Konstruktion noch und erhielt die unteren Schranken

$$\begin{aligned} \Sigma(7) &\geq 22.961 \\ \Sigma(8) &\geq \tfrac{3}{2}(7 \cdot 3^{92} - 1) \ \approx \ 8 \cdot 10^{44}. \end{aligned}$$

3 Effiziente Erzeugung von Turingmaschinen

Eine Möglichkeit, die Funktionswerte $\Sigma(n)$, $S(n)$ und $\sigma(n)$ für kleine n zu bestimmen, liegt in der Erzeugung aller Turingmaschinen aus $\mathcal{TM}_n$ und der anschließenden Untersuchung auf (Nicht-) Termination der Berechnung. Dieser Weg wird hier beschritten.

Dieses Kapitel beschäftigt sich mit der Erzeugung von Turingmaschinen aus $\mathcal{TM}_n$, deren Untersuchung hinsichtlich der betrachteten Funktionen ausreicht. Dazu werden schnelle Verfahren vorgestellt, die es ermöglichen, die zu untersuchende Anzahl schon während der Erzeugung drastisch zu vermindern. Es werden zwei verschiedene Ansätze verfolgt: Zunächst wird versucht, die Anzahl mittels Symmetrie- und Äquivalenz-Betrachtungen zu verringern, was in Abschnitt 2.4 schon zu der Menge $\mathcal{TM}_n^*$ führte. Danach werden einfache (d.h. auch schnell prüfbare) Kriterien für Nichttermination vorgestellt.

3.1 Baumnormale Turingmaschinen

Wie oben erklärt wurde, kann man sich bei der Untersuchung von Turingmaschinen hinsichtlich der Funktionen Σ, S und σ auf diejenigen Maschinen mit Startzustand der Form (0: 1R1 ???) beschränken. Permutiert man nun die restlichen $n-2$ Zustände, so ergeben sich bis zu $(n-2)!$ verschiedene Übergangsfunktionen, die zu Turingmaschinen gehören, die bezüglich der betrachteten Funktionen dasselbe leisten. Ordnet man nun die Zustände nach dem Zeitpunkt ihrer erstmaligen Bearbeitung innerhalb einer gewissen Schrittzahl an und benennt sie entsprechend um (nicht erreichte Einträge werden als undefiniert ($*$) markiert), so erhält man einen Repräsentanten der Äquivalenzklasse, in der alle Turingmaschinen enthalten sind, deren Übergangsfunktionen sich durch Permutation der letzten $n-2$ Zustände ineinander überführen lassen, wobei die Inhalte der undefinierten Einträge frei wählbar sind. Ein solcher Repräsentant wird im weiteren baumnormal genannt. Die Baumnormalform von M nach s Schritten sei mit TNF(M,s) bezeichnet.

Beispiel 3.1

Gegeben sei M durch die Turingtafel

	0	1
0	1R1	1R2
1	1L4	0L3
2	1R3	1LH
3	1L4	1R4
4	0R0	0L1

Ordnet man die Zustände nach den erstmaligen Bearbeitungszeitpunkten an, so ergibt sich die Folge (0,1,4,2,3) und damit die Permutation (0)(1)(423), die zur Baumnormalform von M führt. Die erreichte Baumnormalform ist von der Anzahl der Schritte abhängig, die man M simuliert hat. Nach 3 bzw. 26 Schritten erhält man z.B.

TNF(M,3) =

	0	1
0	1R1	$*$
1	1L2	$*$
2	$*$	0L1
3	$*$	$*$
4	$*$	$*$

TNF(M,26) =

	0	1
0	1R1	1R3
1	1L2	0L4
2	0R0	0L1
3	1R4	$*$
4	1L2	1R2

.

Diese schon in [Bra66] eingesetzte Technik führt zu dem Algorithmus TM-ERZEUGUNG zur Generierung baumnormaler Turingmaschinen; er ist im Anhang aufgeführt.

Der Algorithmus erzeugt rekursiv beginnend mit dem leeren Wort und der fast überall undefinierten Übergangsfunktion bei Erreichen eines undefinierten Eintrages dort alle Möglichkeiten für das zu schreibende Zeichen, für die Richtung und den Zielzustand. Dabei wird festgehalten, welcher Zustand maximal angesprungen werden darf. Dieses erzwingt die gewollte Ordnung auf den Zuständen, die es ermöglicht, nur jeweils einen Repräsentanten zu erzeugen.

Die Simulation wird nach spätestens SCHRITTMAX Schritten abgebrochen, was dazu führen kann, daß noch nicht alle erreichbaren Einträge besucht wurden. Stellt sich bei späteren längeren Simulationen heraus, daß ein undefinierter Eintrag erreicht wird und gibt es mindestens zwei undefinierte Einträge in der Turingtafel, so muß der erreichte Eintrag vollständig expandiert werden. Somit kann später die Anzahl der zu untersuchenden Turingmaschinen noch ansteigen.
Zur Korrektheit des Algorithmus folgender

Satz 3.2

Für alle M $\in \mathcal{TM}_n^*$ ($n \geq 3$) existiert genau eine vom Algorithmus TM-ERZEUGUNG ausgegebene baumnormale Turingmaschine M' mit TNF(M,SCHRITTMAX) = M'.

Beweis:

Zu gegebener M $\in \mathcal{TM}_n^*$ erzeuge man die Baumnormalform innerhalb SCHRITTMAX Schritten durch Ordnung der Zustände wie oben beschrieben. Hieraus ergibt sich eine Ordnung auf den Einträgen nach den erstmaligen Bearbeitungszeitpunkten. Unbesuchte Zustände erhalten die größten Zustandsnummern, und nicht erreichte Einträge werden $*$ gesetzt.
Der Algorithmus generiert bei Erreichen eines undefinierten Eintrages alle dort möglichen Kombinationen für Zeichen, Richtung und Zielzustand. Damit wird jede bei obigem Verfahren erhaltene Folge von Einträgen mindestens einmal erzeugt. Sie wird nicht mehrmals generiert, denn nach vollständiger Expandierung eines Eintrages wird der jeweilige Vorgänger inkrementiert. Wenn es keinen Vorgänger gibt, so terminiert der Algorithmus.

$\diamond$

3.2 Verringerung der Zustandsanzahl

Zwei Zustände sollen genau dann äquivalent genannt werden, wenn sie durch einen Zustand ersetzt werden können, ohne die Berechnung (hinsichtlich der Bandveränderungen und Stoppeigenschaft) zu verändern. Werden bei einer Turingmaschine $M \in \mathcal{TM}_n^*$ zwei äquivalente Zustände erkannt, so gibt es also eine Turingmaschine M' $\in \mathcal{TM}_{n-1}^*$, die die gleiche Stoppeigenschaft wie M besitzt und wenn sie stoppt, die gleiche Anzahl von Einsen auf dem Band hinterläßt. Im folgenden werden vier einfache, schnell prüfbare, hinreichende Bedingungen zur Äquivalenz zweier Zustände angegeben, bei denen auch die Anzahl der Kopfbewegungen erhalten bleibt.

Es seien $U, V \in \{0,1\} \times \{L, R\}$ und $p, q, r, s \in Q$, wobei $p, q \notin \{r\} \cup \{s\}$ gelten soll.

1.

	0	1
p	Ur	Vs
q	Ur	Vs

2.

	0	1
p	Up	Vr
q	Uq	Vr

oder

	0	1
p	Vr	Up
q	Vr	Uq

3.

	0	1
p	Vq	Ur
q	Vp	Ur

oder

	0	1
p	Ur	Vq
q	Ur	Vp

4.

	0	1
p	Vq	Ur
q	Vq	Ur

oder

	0	1
p	Ur	Vq
q	Ur	Vq

Existieren Zustände p und q mit einer dieser Bedingungen, so ist dieses hinreichend für die Äquivalenz von p und q. Die Implikationen sind leicht einzusehen, da in allen Fällen p und q dieselben lokalen Änderungen auf dem Band vornehmen und bei Verlassen von p und q dieselben Folgezustände erreicht werden.

Hat man nun alle Turingmaschinen aus $\mathcal{TM}_n^*$ generiert und die Äquivalenz zweier Zustände einer Turingmaschine M daraus gemäß der obigen Bedingungen festgestellt, so braucht M nicht weiter betrachtet zu werden, da eine Turingmaschine mit weniger Zuständen existiert, die die gleichen Eigenschaften bezüglich der Funktionen Σ, S und σ besitzt.

3.3 Variation des Startzustandes

Betrachtet man eine Turingmaschine $M \in \mathcal{TM}_n^*$ in Baumnormalform, die durch eine Turingtafel der Form

	0	1
0	1R1	???
⋮		
q	1??	???
⋮		

gegeben ist, so besteht die Möglichkeit, einen Zustand $q \neq 0$, der bei Lesen einer Null eine Eins schreibt, als Startzustand zu wählen. Existiert in der dann folgenden Berechnung eine Konfiguration, die auch bei der Berechnung mit Zustand 0 beginnend erreicht wird, so sind die Konfigurationen beider Berechnungen danach identisch.

Diese Beobachtung führt zu einer weiteren Reduktionsmöglichkeit der zu untersuchenden Turingmaschinen-Menge:
Gibt es einen Zustand q wie oben beschrieben, so wird die neue Turingmaschine (sie hat einen anderen Startzustand!) M' in TNF gebracht. Dieses gelingt direkt, wenn q in M die Form (1R? ???) hat. Ist dagegen q von der Gestalt (1L? ???), so muß vorher L und R in der gesamten Turingtafel vertauscht werden. Nun liegen M und M' in TNF vor. Mit einer auf TNF($\mathcal{TM}_n^*$) definierten (vollständigen) Ordnung $\preceq$ (zum Beispiel: $M_1 \preceq M_2 :\Leftrightarrow$ Binärdarstellung der Turingtafel von $M_1 \leq$ Binärdarstellung der Turingtafel von M_2) können M und M' verglichen werden. Ist $M \prec M'$, so braucht M' nicht weiter betrachtet zu werden, denn zu einem M gibt es nur endlich viele Turingmaschinen M' aus $\mathcal{TM}_n^*$, die nach obigem Verfahren entstehen. Alle diese Turingmaschinen stoppen oder es stoppt keine von ihnen. Damit genügt es, einen Repräsentanten weiter zu untersuchen, und dieser wird zum Beispiel als minimale Turingmaschine unter der obigen Ordnung gewählt.

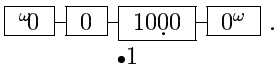
Es muß hier bemerkt werden, daß zwar die Stoppeigenschaften und im Fall des Haltens die Anzahl der hinterlassenen Einsen bei allen Turingmaschinen in der betrachteten Menge gleich sind, die Anzahl der Schritte oder bedruckten Felder sich aber sehrwohl unterscheiden können. Ist also eine hinsichtlich S oder σ "gute" Turingmaschine gefunden worden, so können die Werte u.U. durch Wahl eines anderen Startzustandes verbessert werden.
An einem Beispiel soll das Verfahren illustriert werden.

Beispiel 3.3

Es sei M (in TNF) durch die Turingtafel

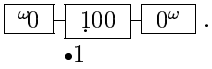
	0	1
0	1R1	0R4
1	0L2	1L1
2	1L3	0L1
3	1R0	0R1
4	0R0	$*$

gegeben. Nach 14 Schritten erreicht M die Konfiguration



.

Hierbei ist die Startposition mit '$\dot{}$' gekennzeichnet und alle Positionen zwischen $\boxed{^\omega 0}$ und $\boxed{0^\omega}$ sind von M bedruckt worden.
Wählt man nun Zustand 3 als Startzustand, so erreicht die neue Turingmaschine M' nach vier Schritten die Konfiguration



.

Die beiden Konfigurationen sind gleich (trotz einer Bandverschiebung um zwei Positionen). M' ist in TNF gegeben durch

	0	1
0	1R1	0R2
1	1R2	0R4
2	0L3	1L2
3	1L0	0L2
4	0R1	$*$

,

und je nach gewählter Ordnung muß nur noch entweder M oder M' betrachtet werden.

3.4 Einfache hinreichende Bedingungen für Nichttermination

Um nicht zu viele Turingmaschinen aus TNF($\mathcal{TM}_n$) abspeichern zu müssen, ist es nützlich, frühzeitig (d.h. schon im Algorithmus TM-ERZEUGUNG) einfache hinreichende Bedingungen für die Nichttermination zu testen. Eine Turingmaschine stoppt nicht, wenn

- der Haltzustand vom aktuellen Zustand nicht erreichbar ist.

  Ein Spezialfall dieses Testes ist im Algorithmus TM–ERZEUGUNG schon implementiert worden: Mindestens ein Eintrag muß undefiniert sein, sonst kann die Turingmaschine nicht in den Haltzustand gelangen. Beginnend beim aktuellen Zustand während einer Simulation wird die Menge der erreichbaren Zustände durch eine Breadth–First–Suche im Übergangsgraphen bestimmt. Gibt es in diesen Zuständen keinen undefinierten Eintrag, so stoppt die Turingmaschine nicht.

- sich eine Konfiguration wiederholt.

  Hierzu wird während einer Simulation der betrachteten Turingmaschine die Folge der Nummern abgearbeiteter Einträge betrachtet. Zu einem Zeitpunkt t_2 wird die erreichte Bandposition zu Null festgelegt und davor nach Zeitpunkten t_1 und t_0 gesucht, wo Position Null erreicht wurde. Sind nun die Nummern–Teilfolgen $A = (e_t)_{t=t_0}^{t_1-1}$ und $B = (e_t)_{t=t_1}^{t_2-1}$ gleich, so sind auch die Konfigurationen zu den Zeitpunkten t_1 und t_2 gleich und die Turingmaschine stoppt nicht, denn bei der Abarbeitung der Einträge in A wird ein gewisser Bandinhalt von der Turingmaschine erwartet, und dieser ist gleich dem, den die Bearbeitung von A hinterläßt, da im Anschluß $B = A$ abgearbeitet wurde.

- das Erreichen des Haltzustandes zu einem Widerspruch geführt werden kann.

  Notwendig zum Erreichen des Haltzustandes ist die Erreichbarkeit eines undefinierten Eintrages in der Turingtafel. Kann sie nun für alle undefinierten Einträge zum Widerspruch geführt werden, so stoppt die betrachtete Turingmaschine nicht.

  Für jeden undefinierten Eintrag wird jetzt nacheinander angenommen, daß er der erste erreichte ist und versucht, dieses durch Rückwärtslauf der Turingmaschine zu widerlegen. Dabei entsteht ein Konfigurationsbaum: Die Wurzel ist die als erreicht angenommene Konfiguration. Die Söhne eines Knotens bilden die Konfigurationen, bei denen die Turingmaschine nach Abarbeitung des aktuellen Zustandes in den Zustand der Vaterkonfiguration gelangt. Dabei kann es zu einem widersprüchlichem Bandinhalt kommen, so daß manche Söhne ausscheiden. Brechen alle Pfade von der Wurzel aus derart ab, kann also die angenommene Konfiguration nicht erreicht worden sein. Wenn dieses bei allen undefinierten Einträgen der Fall ist, dann stoppt die Turingmaschine nicht.

  Zur Vereinfachung dieses Testes ist es zweckmäßig, die untersuchte Turingmaschine vorwärts mindestens solange simuliert zu haben, wie die maximale vorgegebene

Tiefe des Baumes angibt, da dann nicht in jedem Knoten geprüft werden muß, ob die Startkonfiguration erreicht worden ist.

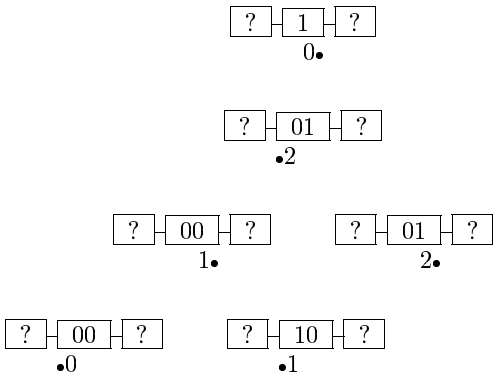
Beispiel 3.4

Gegeben ist die Turingmaschine M $\in \mathcal{TM}_3$ in TNF durch

	0	1
0	1R1	$*$
1	1L2	1R1
2	0R0	0L2

.

Nun wird angenommen, daß der mit '$*$' markierte undefinierte Eintrag erreicht wird. Hieraus ergibt sich der folgende Konfigurationsbaum:



Hierbei sind die mit '?' bezeichneten Zellen irrelevant und haben nichts miteinander zu tun. Bei allen Blattkonfigurationen liegt ein Widerspruch vor, denn die Ausführung des jeweiligen Eintrages führt nicht zu der gewünschten Vaterkonfiguration. M stoppt demnach nicht.

- sich der Kopf periodisch in eine Richtung bewegt und dabei immer neue Bandpositionen erreicht.

  Hier wird wieder die Folge der Nummern der bearbeiteten Einträge einer Turingtafel während einer Simulation untersucht. Zusätzlich sollen nun Nummern, bei denen eine vorher noch nicht bedruckte Position erreicht wird, als Exstremstelle an der jeweiligen Seite des Bandes markiert werden. Ab einem Zeitpunkt t_2 der Berechnung, an dem eine Extremstelle vorliegt, werden nun nach vorne zwei weitere Extremstellen der gleichen Bandseite in der Folge gesucht. Dazwischen sollen keine Extremstellen der anderen Seite vorkommen. Die Zeitpunkte seien t_1 und t_0. Sind nun die Teilfolgen $A = (e_t)_{t=t_0}^{t_1-1}$ und $B = (e_t)_{t=t_1}^{t_2-1}$ gleich (Markierungen sind dabei relevant), so wird ab t_2 wieder die

Folge von Einträgen abgearbeitet, deren Nummern in A gegeben sind. Damit ist durch vollständige Induktion bewiesen, daß die betrachtete Turingmaschine nicht stoppt.

Der behauptete Induktionsschluß läßt sich wie folgt zeigen:

Die Bandposition zu Beginn der Ausführung von A sei gleich Null. Ohne Einschränkung kommen $l > 0$ Extremstellen am rechten Rand in A vor. Die minimale erreichte Position in A sei $m \leq 0$. A erwartet also links von der Startposition auf $-m$ Positionen einen bestimmten Bandinhalt und ab der Startposition l Nullen nach rechts. B tut dieses auch. Demnach muß die Abarbeitung von A auf den $-m$ von B nach links erwarteten Bandpositionen den richtigen Bandinhalt hinterlassen haben. Somit wird nach Bearbeitung von B $(= A)$ wieder A ausgeführt.

4 Nichtterminationsbeweis durch Zustandsfolgenbetrachtung

In diesem Kapitel wird ein neues Verfahren zum Beweis der Nichttermination für eine große Teilmenge von $\mathcal{TM}_5$ entwickelt. Ausgehend von einem Anfangsstück der Zustandsfolge bei der Berechnung einer Turingmaschine M wird dabei eine Gesamtdarstellung der (unendlichen) Folge vermutet und schließlich versucht, diese mittels vollständiger Induktion zu beweisen. Wenn dieses gelingt, stoppt M nicht.

4.1 Motivation

Die Arbeitsweise der Turingmaschinen, die hier untersucht werden sollen, läßt sich anschaulich durch Kopfpositions-Zeit-Diagramme beschreiben. Hierbei wird skizzenhaft angedeutet, wie sich der Kopf der Turingmaschine qualitativ mit der Zeit über das Band bewegt. Wir unterscheiden die fünf in Abbildung 1 gezeigten Muster. Bei Turingmaschinen der ersten Art tritt eine Konfigurationswiederholung auf. Diese sind zusammen mit den durch Diagramm 2 beschriebenen Turingmaschinen schon in Abschnitt 3.4 behandelt worden. Interessant sind die übrigen Arten, denn sie stellen den weitaus größten Anteil der nach Anwendung der einfachen Beweisalgorithmen noch verbliebenen Turingmaschinen aus $\mathcal{TM}_5$. A. Brady hat zur Bestimmung von $\Sigma(4)$ für all diese spezielle, auf Bandbetrachtung (siehe Kapitel 5) beruhende Beweisalgorithmen entwickelt, welche aber letztlich nur Spezialfälle lösen konnten. In $\mathcal{TM}_4$ reichen diese Methoden aus, um die Nichttermination bis auf wenige Ausnahmen zu zeigen. Betrachtet man die in $\mathcal{TM}_5$ vorkommenden Variationen, so ergibt sich schnell die Notwendigkeit einer Verallgemeinerung, denn die Anzahl der auftretenden Fälle steigt beträchtlich an. Die Lösung liegt in der Untersuchung der in der Berechnung einer Turingmaschine abgearbeiteten Zustandsfolge.

4.2 XMAS-Turingmaschinen

Zunächst werden die im folgenden benötigten Begriffe eingeführt.

Definition 4.1

Sei $M \in \mathcal{TM}_k$, und die Berechnung von M auf ϵ terminiere nicht. Dann ist die Laufformel F_M das unendliche Wort gebildet aus den Nummern der in der Berechnung abgearbeiteten Einträge der Turingtafel von M. Es ist also

$$F_M = (e_i)_{i=1}^{\infty} \in E^{\omega} \text{ mit } e_i = q_i \cdot 2 + bit_i \in E := \{0, \ldots, 2k-1\}.$$

Hierbei ist q_i der zum Zeitpunkt i der Berechnung erreichte Zustand und $bit_i \in \{0,1\}$ das Zeichen unter dem Kopf zu diesem Zeitpunkt.
Als Extremstellen-Eintragnummer wird eine Nummer in der Laufformel bezeichnet, bei deren Bearbeitung eine vorher noch nicht erreichte Bandposition bedruckt wird. Diese Nummern spielen in den folgenden Betrachtungen eine große Rolle und werden

später mit $^+$ bzw. $_-$ markiert, je nachdem, ob am rechten oder linken Rand des Bandes eine neue Position erreicht wird.
Im weiteren bezeichnen $\alpha, \beta, \gamma \ldots$ endliche Wörter über E, $\alpha \cdot \beta$ bezeichnet die Konkatenation von α und β, $|\alpha|$ die Länge und $\alpha[k]$ das k-te Zeichen von α. $(\alpha)^j$ ist für $j \geq 0$ induktiv durch $(\alpha)^0 := \epsilon$, $(\alpha)^{j+1} := (\alpha)^j \cdot \alpha$ definiert.

Beispiel 4.2

Es sei $M3 \in \mathcal{TM}_3$ durch

	0	1
0	1R1	1L0
1	1L0	1R2
2	1RH	1R1

mit Eintragnummern

	0	1
0	0	1
1	2	3
2	4	5

gegeben. M3 stoppt auf ϵ nicht (wie mit der Methode aus diesem Kapitel bewiesen werden kann) und das Anfangsstück von F_{M3} nach z.B. 35 Schritten ist

$$F_{M3}^{35} = 02103521110353521111103535352111111$$

Mit diesen Begriffen können jetzt die betrachteten Turingmaschinen definiert werden.

Definition 4.3

$M \in \mathcal{TM}_k$ heißt XMAS-Turingmaschine $:\Leftrightarrow$

1. Die Berechnung von M auf ϵ terminiert nicht.
2. $\exists n \in \mathbb{N}_0, \forall 0 \leq j \leq n\ \exists \alpha_j \in E^+, \exists \alpha_{n+1} \in E^*, \forall 1 \leq j \leq n\ \exists \beta_j \in E^+$ mit

$$F_M = \alpha_0 \prod_{i=1}^{\infty} \underbrace{[\alpha_1 (\beta_1)^i \alpha_2 (\beta_2)^i \cdots (\beta_n)^i \alpha_{n+1}]}_{\text{Hauptschleife}}$$

Bemerkung 4.4

1. Es gibt XMAS-Turingmaschinen, denn z.B. alle Turingmaschinen M, bei deren Berechnung eine Konfigurationswiederholung vorkommt, haben ein F_M der Form $\alpha_0 \prod_{i=1}^{\infty} [\alpha_1]$.
2. Anschaulich beschreiben die Klammerungen $(\beta_j)^i$ die sich verlängernden Strecken im Kopfpositions-Zeit-Diagramm und die α_j in der Hauptschleife die Aktionen an den Rändern.

Beispiel 4.5

M3 ist eine XMAS-Turingmaschine mit $F_{M3} = 0210 \cdot \prod_{i=1}^{\infty}[352(1)^i1(1)^i0(35)^i]$. Der Beweis dieser Behauptung erfolgt später.

Den ersten Schritt hin zu einer Normalform von F_M, die zu einer schnellen Erkennung von etwaigen XMAS-Turingmaschinen und zu einer einfacheren Beweistechnik führt, macht der folgende

Satz 4.6

Sei M eine XMAS-Turingmaschine mit

$$F_M = \alpha_0 \prod_{i=1}^{\infty}[\alpha_1(\beta_1)^i\alpha_2(\beta_2)^i \cdots (\beta_n)^i\alpha_{n+1}].$$

Dann gibt es eine Darstellung

$$F_M = \alpha'_0 \prod_{i=1}^{\infty}[\alpha'_1(\beta_1)^i\alpha'_2(\beta_2)^i \cdots (\beta_n)^i\alpha'_{n+1}],$$

bei der etwaige Extremstellen bei festen Eintragnummern in den α'_j erreicht werden.

Beweis:

Zu Beginn werden nützliche Abkürzungen vereinbart:

- $a_j := \sum_{k=1}^{|\alpha_j|} \text{Richtung}_M(\alpha_j[k])$ für $1 \le j \le n+1$ gibt die Differenz der Kopfpositionen vor und nach der Abarbeitung der Einträge in α_j an. Hierbei bezeichnet $\alpha_j[k]$ die k-te Nummer in α_j, und $\text{Richtung}_M(e)$ ist gleich 1 bzw. -1, wenn der zu e gehörige Eintrag in der Turingtafel von M den Kopf nach rechts bzw. links bewegt.
- $b_j := \sum_{k=1}^{|\beta_j|} \text{Richtung}_M(\beta_j[k])$ für $1 \le j \le n$
- $a := \sum_{j=1}^{n+1} a_j$ ist die Summe aller Schritte bei Abarbeitung aller Einträge in den α_j der Hauptschleife.
- $b := \sum_{j=1}^{n} b_j$ ist die Summe aller Schritte bei Abarbeitung aller β_j.

1.Fall $n = 0$

F_M ist also gleich $\alpha_0 \prod_{i=1}^{\infty}[\alpha_1]$. Ist $a_1 = 0$, so wird bei $i \ge 2$ in α_1 keine Extremstelle mehr erreicht; die gewünschte Darstellung ist demnach $\alpha'_0 = \alpha_0\alpha_1, \alpha'_1 = \alpha_1$.

Falls $a_1 \neq 0$ ist, so existiert ein $i_o \in \mathbb{N}$, so daß ab $i = i_0$ pro Durchlauf der Hauptschleife genau a_1 Extremstellen erreicht werden, denn der Kopf bewegt sich in jedem Durchlauf eine konstante Anzahl (a_1) von Feldern in eine Richtung weiter. Für die Kopfpositionen bei Erreichen einer festen Nummer e in α_1 im $(i+1)$-ten Durchlauf gilt $\text{pos}^e_{i+1} = \text{pos}^e_i + a_1$. Ist nun bei e eine Extremstelle im Durchlauf i erreicht worden, so wird demnach auch im $(i+1)$-ten Durchlauf der Hauptschleife bei e eine Extremstelle erreicht. Das heißt aber, daß alle Extremstellen ab $i = i_0$ bei festen Eintragnummern in α_1 erreicht werden.

2.Fall $n \ge 1$, $\exists j \in \{1, \ldots, n\}$ mit $b_j = 0$

D.h. die Start- und Endposition auf dem Band bei einer Bearbeitung von β_j sind gleich.

Ist nun $i \ge 2$, so tritt ein Konfigurationswiederholung auf, denn nach Abarbeitung von β_j hat sich der Bandinhalt nicht verändert (β_j wird nochmals auf demselben Bandstück bearbeitet) und die Turingmaschine befindet sich in demselben Zustand auf derselben Bandposition wie zu Beginn der ersten Bearbeitung von β_j. Damit hat F_M auch die Darstellung $\alpha'_0 \prod_{i=1}^{\infty}[\alpha'_1]$ mit $\alpha'_1 = \beta_j$ für ein $j \in \{1, \ldots, n\}$ und Fall 1 liefert die Behauptung.

3.Fall $n \ge 1$, $\forall j \in \{1, \ldots, n\}$ ist $b_j \neq 0$, $b \neq 0$ ($\mathbb{E}$ $b > 0$)

Für F_M sei im folgenden die Darstellung

$$F_M = \alpha''_0 \prod_{i=1}^{\infty}[\gamma_0(\beta_1)^i\gamma_1(\beta_2)^i \cdots (\beta_n)^i\gamma_n]$$

gewählt mit $\gamma_0 = \alpha_1\beta_1$, $\gamma_j = \beta_j\alpha_{j+1}\beta_{j+1}$ für $1 \le j < n$ und $\gamma_n = \beta_n\alpha_{n+1}$.

Beh.1 Es existiert ein $i_0 \in \mathbb{N}$, so daß für alle $i \ge i_0$ die Maximalstellen (d.h. Bandpositionen möglichst weit rechts) im i-ten Durchlauf der Hauptschleife nur bei Nummern in γ_l erreicht werden mit $\sum_{j=1}^{l} b_j \stackrel{!}{=} \max =: \Delta$, wobei $l \in \{0, \ldots, n\}$ ist.

Bew. Innerhalb der $(\beta_j)^i$ werden bei keinem i Maximalstellen erreicht, da alle $(\beta_j)^i$ vorne und hinten von β_j umgeben sind und $\forall j\ b_j \neq 0$ gilt.
Sei $A = \sum_{j=1}^{n+1} |\alpha_j|$ und $B = \sum_{j=1}^{n} |\beta_j|$. Mit $i_0 := 2A + B - 1$ gilt die Behauptung. Denn setzt man die Bandposition bei Beginn von α_1 zu Null, so wird in γ_l eine Position $\ge (i+2)\Delta - A$ erreicht, da bis zu α_{l+1} (in γ_l) alle β_j $(i+2)$-mal bearbeitet werden und für die Abschätzung angenommen werden kann, daß alle α_j den Kopf nur nach links bewegen.
In den γ_k mit $\sum_{j=1}^{k} b_j < \Delta$ gilt für die Bandposition pos zu Beginn von α_{k+1}

$$\text{pos} = \sum_{j=1}^{k} a_j + (i+2)\sum_{j=1}^{k} b_j \ \le \ \sum_{j=1}^{k} a_j + (i+2)(\Delta - 1).$$

Somit hat man aber für alle in γ_k erreichten Positionen pos' (mit Setzung $|\beta_{n+1}| := 0$)

$$\begin{aligned} \text{pos'} &\le \sum_{j=1}^{k} a_j + (i+2)(\Delta - 1) + |\beta_k| + |\alpha_{k+1}| + |\beta_{k+1}| \\ &\le (i+2)(\Delta - 1) + A + B \ < \ (i+2)\Delta - A \ \text{ für } i \ge i_0. \end{aligned}$$

Für die Bandpositionen bei der Bearbeitung einer festen Eintragnummer e in γ_l in aufeinanderfolgenden Hauptschleifendurchläufen gilt

$$\text{pos}^e_{i+1} = \ \text{pos}^e_i + a + ib + \sum_{j=1}^{l} b_j \ ,$$

da bis zum nächsten Erreichen von e alle α_j abgearbeitet werden, dazu alle β_j mit $j > l$ i-mal und alle β_j mit $j \leq l$ $(i+1)$-mal. Damit erhält man auch für die maximalen Bandpositionen im $(i+1)$-ten Durchlauf

$$\max_{i+1} = \max_i + a + ib + \Delta.$$

Es existiert also ein $i_1 \in \mathbb{N}$, $i_1 \geq i_0$, so daß für alle $i \geq i_1$ im $(i+1)$-ten Durchlauf $a + ib + \Delta$ Maximum-Extremstellen erreicht werden. Da es aber nur endlich viele Eintragnummern außerhalb der $(\beta_j)^i$ gibt, müssen in den Klammerungen beliebig viele Extremstellen erreicht werden können. Damit muß es mindestens eine Klammerung $(\beta_j)^i$ geben, bei der beliebig viele Extremstellen erreicht werden können. Werden dort aber mehr als $2 \cdot b_j - 2$ erreicht, so wurde ein β_j bearbeitet, bei dem alle rechts neu betretenen Felder Extremstellen waren, da β_j den Kopf um b_j Positionen nach rechts verschiebt. Damit akzeptiert β_j auf den Feldern, die bei der Bearbeitung des vorherigen β_j nicht betreten wurden, Nullen. Dieses bedeutet aber, daß im Anschluß nur noch β_j ausgeführt wird. F_M hat also auch hier die Darstellung $\alpha'_0 \prod_{i=1}^{\infty}[\alpha'_1]$ mit $\alpha'_1 = \beta_j$ für ein $j \in \{1, \ldots, n\}$, und die Behauptung ist mit Fall 1 gezeigt.

Der Fall $b < 0$ wird analog behandelt.

4.Fall $n \geq 1$, $\forall j \in \{1, \ldots, n\}$ ist $b_j \neq 0$, $b = 0$

Ähnlich wie im 3. Fall erhält man für alle $i \geq 0$

$$\begin{aligned} \max_{i+1} &= \max_i + \Delta_{\max} \\ \min_{i+1} &= \min_i + \Delta_{\min} \end{aligned} \tag{3}$$

mit

$$\begin{aligned} \Delta_{\max} &= a + \max\{\sum_{j=1}^{l} b_j \mid 0 \leq l \leq n\} \\ \Delta_{\min} &= a + \min\{\sum_{j=1}^{l} b_j \mid 0 \leq l \leq n\} \end{aligned}$$

Ab einem bestimmten i_0 kommen also in der Hauptschleife eine konstante Anzahl von Extremstellen vor. Diese Anzahl ist größer Null, weil $\Delta_{\max} > 0$ oder $\Delta_{\min} < 0$ wegen $n \geq 1$ und $\forall j$ $b_j \neq 0$ gilt. Exemplarisch wird die Behauptung des Satzes für Maximum-Extremstellen gezeigt.

Beh.2 Sei $\Delta_{\max} > 0$ und $b = 0$. Dann existiert $i_1 \in \mathbb{N}$, so daß Maximum-Extremstellen für $i \geq i_1$ bei festen Eintragnummern in den γ_l mit $\sum_{j=1}^{l} b_j \stackrel{!}{=} \max(= \Delta)$ erreicht werden.

Bew. Wegen Gleichung (3) werden ab einem gewissen i_0 in jedem Hauptschleifendurchlauf $\Delta_{\max}$ Maximum-Extremstellen erreicht. Unter Beachtung der Beziehungen aus dem Beweis von Behauptung 1 ergibt sich die Existenz eines

$i_1 \in \mathbb{N}$, $i_1 \geq i_0$, so daß für alle $i \geq i_1$ die Bandpositionen in den γ_k mit $\sum_{j=1}^{k} b_j < \Delta$ im $(i+1)$-ten Durchlauf $\leq \max_{i+1} - \Delta_{\max} = \max_i$ sind, denn man hat für Bandpositionen pos in solchen γ_k

$$\begin{aligned} & \max_{i+1} - \text{pos} \\ \geq\ & (i+1+2)\Delta - A - [(i+1+2)(\Delta - 1) + A + B] \\ =\ & i + 3 - 2A - B. \end{aligned}$$

und somit $\text{pos} \leq \max_{i+1} - (i+3) + 2A + B$.

Auch in Klammerungen $(\beta_j)^i$ können keine Extremstellen erreicht werden, sonst würde wie im Fall 3 nur noch β_j für ein j bearbeitet und damit $F_M = \alpha'_0 \prod_{i=1}^{\infty}[\beta_j]$ sein, weil der Klammerung $(\beta_j)^i$ ein β_j folgt, bei dem dann alle neu betretenen Positionen Extremstellen sind. Diese Darstellung steht aber im Widerspruch zur alten, in der es ein β_k mit $b_k < 0$ und ein $\beta_{k'}$ mit $b_{k'} > 0$ gibt, da $n \geq 1$, $\forall j$ $b_j \neq 0$ und $b = 0$ gilt.

Somit werden alle Maximum-Extremstellen bei $i \geq i_1$ in den γ_l mit $\sum_{j=1}^{l} b_j = \Delta$ erreicht. Wird nun im i-ten Durchlauf bei einer bestimmten Eintragnummer e in einem γ_l eine Maximum-Extremstelle pos_i^e betreten, so wird dort im Durchlauf $i+1$ die Extremstelle pos_{i+1}^e erreicht. Es gilt nämlich

$$\text{pos}_{i+1}^e = \text{pos}_i^e + \Delta_{\max}.$$

Diese Kopfposition kann im i-ten Durchlauf nicht erreicht werden, da alle erreichbaren Positionen $\leq \max_{i-1} + \Delta_{\max} < \text{pos}_i^e + \Delta_{\max}$ sind. Wird die Kopfposition pos_{i+1}^e im $(i+1)$-ten Durchlauf früher als bei der betrachteten Nummer e erreicht, so wäre dort im vorherigen Durchlauf die Position pos_i^e auch früher betreten worden – ein Widerspruch dazu, daß pos_i^e eine Extremstelle war.

Analog wird die Behauptung für Minimum-Extremstellen gezeigt.

◇

Bemerkung 4.7

Durch die Zahlen $\Delta_{\max}$ und $\Delta_{\min}$ lassen sich die Turingmaschinen-Typen 3), 4) und 5) aus Abbildung 1 unterscheiden: Im Fall 3) ist $\Delta_{\min} < 0$ und $\Delta_{\max} > 0$. Bei Turingmaschinen gemäß 4) ist $\Delta_{\min} = 0$ und $\Delta_{\max} > 0$ und bei 5) schließlich $\Delta_{\min}, \Delta_{\max} > 0$ und $\Delta_{\min} < \Delta_{\max}$.

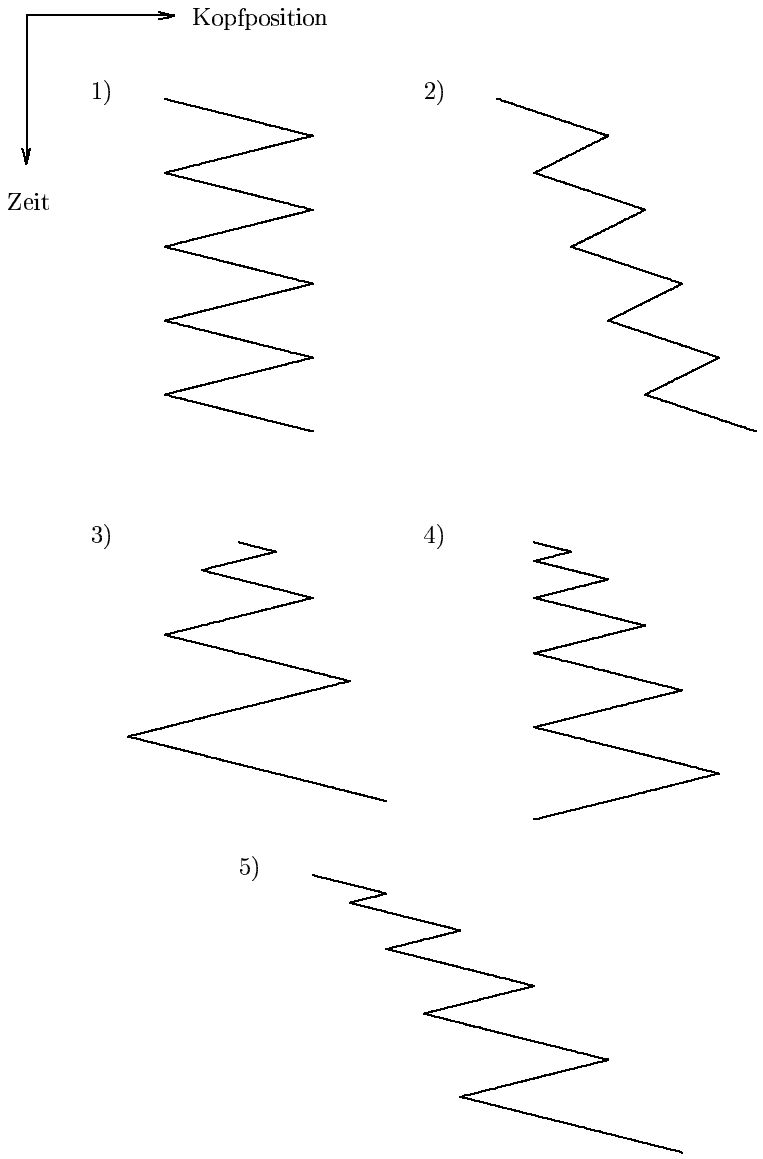


Abbildung 1: Kopfpositions-Zeit-Diagramme der betrachteten Turingmaschinen

Beispiel 4.8

Markiert man Extremstellen-Eintragnummern mit $^+$ bzw. $\_$, so ergibt sich

$$\mathrm{F}_{\mathrm{M3}}^{35} = 0\overset{+}{\_}2^+10\_352^+1110\_35352^+111110\_3535352^+111111,$$

und allgemein (hier ohne Beweis) erhält man z.B.

$$\mathrm{F}_{\mathrm{M3}} = 0\overset{+}{\_}2^+10\_\prod_{i=1}^{\infty}[352^+(1)^i1(1)^i0\_(35)^i] \; .$$

Diese Form genügt der Behauptung des Satzes, und die Größen im 4. Fall des Beweises sind:

$$\begin{aligned}
n &= 3 & \\
\alpha_1 &= 352 & a_1 &= 1+1-1=1 \\
\beta_1 &= 1 & b_1 &= -1 \\
\alpha_2 &= 1 & a_2 &= -1 \\
\beta_2 &= 1 & b_2 &= -1 \\
\alpha_3 &= 0 & a_3 &= 1 \\
\beta_3 &= 35 & b_3 &= 1+1=2 \\
\alpha_4 &= \epsilon & a_4 &= 0 \\
a = 1-1+1 &= 1 & b &= -1-1+2=0
\end{aligned}$$

$$\begin{aligned}
\Delta_{\min} &= a + \min\{\sum_{j=1}^{l} b_j \mid 0 \le l \le n\} = -1 \\
\Delta_{\max} &= a + \max\{\sum_{j=1}^{l} b_j \mid 0 \le l \le n\} = 1
\end{aligned}$$

Die Form der Laufformel kann noch weiter eingeschränkt werden, wie der folgende Satz zeigt.

Satz 4.9

Sei M eine XMAS-Turingmaschine mit

$$\mathrm{F_M} = \alpha_0 \prod_{i=1}^{\infty} [\alpha_1(\beta_1)^i \alpha_2(\beta_2)^i \cdots (\beta_n)^i \alpha_{n+1}].$$

Dann existiert eine Darstellung

$$\mathrm{F_M} = \gamma_0 \prod_{i=1}^{\infty} [\gamma_1(\delta_1)^i \gamma_2(\delta_2)^i \cdots (\delta_m)^i \gamma_{m+1}]$$

mit den folgenden Eigenschaften:

- $\gamma_i, \delta_i \in \mathrm{E}^+$ für $1 \leq i \leq m$,
  $\gamma_0 \in \mathrm{E}^*$ und $\gamma_{m+1} \in \mathrm{E}^*$, falls $m \geq 1$, sonst $\gamma_1 \in \mathrm{E}^+$.
- Extremstellen werden nur bei festen Eintragnummern in den γ_j erreicht.
- Wenn überhaupt Extremstellen in der Hauptschleife erreicht werden, dann auch bei $\gamma_1[1]$.
- $(\delta_j[1] \neq \gamma_{j+1}[1] \ \vee \ \gamma_{j+1}[1]$ Extremstellen-Nummer) für $1 \leq j < m$ und
  $([\gamma_{m+1} \neq \epsilon \ \wedge \ (\delta_m[1] \neq \gamma_{m+1}[1] \ \vee \ \gamma_{m+1}[1]$ Extremstelle-Nummer) $] \ \vee \ \gamma_{m+1} = \epsilon)$.

In dem folgenden Beweis des Satzes werden zwei Hilfsaussagen benutzt, die zunächst formuliert und bewiesen werden.

Lemma 4.10

Seien α und β endliche Wörter über einem Alphabet E. Dann gilt

$$\exists \gamma \in \mathrm{E}^* \ \forall i \in \mathbb{N}_0 : \ (\alpha)^i(\beta)^i = (\gamma)^i$$

$$\Leftrightarrow \ \alpha\beta\alpha\beta = \alpha\alpha\beta\beta \ \Leftrightarrow \ \alpha\beta = \beta\alpha$$

Beweis:

Zu zeigen ist nur die erste Äquivalenz, die zweite ist leicht einzusehen.

"$\Rightarrow$" Man benutzt die Voraussetzung für $i \in \{1, 2\}$ und erhält $\alpha\beta = \gamma$ und $\alpha\alpha\beta\beta = \gamma\gamma$ und damit direkt die Behauptung.

"$\Leftarrow$" Man setzt $\gamma = \alpha\beta$ und zeigt die Behauptung durch vollständige Induktion

$[i \in \{0, 1\}]$ Klar!

$[i \to i + 1]$ Es gelte $(\gamma)^i = (\alpha)^i(\beta)^i$ und damit auch $(\gamma)^{i+1} = \alpha\beta(\alpha)^i(\beta)^i$. Wegen $\alpha\beta = \beta\alpha$ gilt $\forall k \in \{0, \ldots, i-1\} : \alpha(\alpha)^k\beta(\alpha)^{i-k}(\beta)^i = \alpha(\alpha)^{k+1}\beta(\alpha)^{i-k-1}(\beta)^i$. Hieraus ergibt sich $\alpha\beta(\alpha)^i(\beta)^i = \alpha(\alpha)^i\beta(\beta)^i = (\alpha)^{i+1}(\beta)^{i+1}$, also $(\gamma)^{i+1} = (\alpha)^{i+1}(\beta)^{i+1}$.

$\diamond$

Lemma 4.11

Seien α und β endliche Wörter ungleich ϵ über einem Alphabet E mit $a = |\alpha|$, $b = |\beta|$ und $l = \mathrm{kgV}(a, b)$. Dann gilt

$$(\alpha)^{l/a} = (\beta)^{l/b} \quad \Leftrightarrow \quad \alpha\beta = \beta\alpha.$$

Beweis:

"$\Rightarrow$" Aus der Voraussetzung erhält man $(\alpha)^{l/a}(\alpha)^{l/a} = (\beta)^{l/b}(\beta)^{l/b}$.
Daraus folgt $\alpha(\alpha)^{l/a}(\alpha)^{l/a-1} = \beta(\beta)^{l/b}(\beta)^{l/b-1}$ und $\alpha(\beta)^{l/b}(\alpha)^{l/a-1} = \beta(\alpha)^{l/a}(\beta)^{l/b-1}$.
Damit ist $\alpha\beta = \beta\alpha$ und die Behauptung bewiesen.

"$\Leftarrow$" Für alle $k \in \mathbb{N}_0$, $l \in \mathbb{N}$ gilt wegen der Voraussetzung $(\alpha)^k(\beta)^l = \beta(\alpha)^k(\beta)^{l-1}$. Hieraus erhält man $(\alpha)^{l/a}(\beta)^{l/b} = (\beta)^{l/b}(\alpha)^{l/a}$ und damit direkt die Behauptung.

$\diamond$

Beweis von Satz 4.9:

Gemäß Satz 4.6 kann angenommen werden, daß $\mathrm{F_M}$ die Form

1) $\alpha_0 \prod_{i=1}^{\infty}[\alpha_1]$ mit $a_1 = 0$ oder $a_1 \neq 0$ oder
2) $\alpha_0 \prod_{i=1}^{\infty}[\alpha_1(\beta_1)^i\alpha_2(\beta_2)^i \cdots (\beta_n)^i\alpha_{n+1}]$ mit $n \geq 1$, $b_j \neq 0 \ \forall j$ und $b = 0$

hat, wobei Extremstellen nur bei festen Nummern in den α_j erreicht werden.
Im Fall 1 ist bei $a_1 = 0$ nichts zu zeigen. Gilt $a_1 \neq 0$, so folgt die Behauptung des Satzes durch Verschieben von α_1, so daß in der damit erhaltenen Form $\alpha'_0 \prod_{i=1}^{\infty}[\alpha'_1]$ bei $\alpha'_1[1]$ eine Extremstelle erreicht wird.
Zu zeigen bleibt die Behauptung im Fall 2. Hierbei wird wieder durch Verschieben der Hauptschleife erzwungen, daß in der erhaltenen Darstellung

$$\mathrm{F_M} = \alpha'_0 \prod_{i=1}^{\infty} [\alpha'_1(\beta'_1)^i\alpha'_2(\beta'_2)^i \cdots (\beta'_n)^i\alpha'_{n+1}]$$

bei $\alpha'_1[1]$ eine Extremstelle erreicht wird. Jede Klammerung $(\beta'_j)^i$ wird nun beginnend bei $(\beta'_n)^i$ nach links dem folgenden Verfahren unterzogen:

Schritt 1 Schiebe die betrachtete Klammerung nach rechts solange die erste Eintragnummer in der Klammer mit der Nummer hinter der Klammerung übereinstimmt und dort keine Extremstelle erreicht wird. D.h.

$$\begin{aligned} \mathrm{F_M} &= \alpha'_0 \prod_{i=1}^{\infty} [\cdots (e_1 e_2 \cdots e_k)^i e_1 e \cdots] \\ &\downarrow \\ \mathrm{F_M} &= \alpha'_0 \prod_{i=1}^{\infty} [\cdots e_1 (e_2 \cdots e_k e_1)^i e \cdots], \end{aligned}$$

falls e_1 hinter der Klammerung keine Extremstelle erreicht.

Dabei wird F_M nicht verändert, denn es gilt

$$\forall i \in \mathbb{N}_0 : (e_1 e_2 \cdots e_k)^i e_1 = e_1 (e_2 \cdots e_k e_1)^i,$$

wie man leicht durch vollständige Induktion beweist.

Falls hinter der zu verschiebenden Klammerung eine Extremstellen-Eintragnummer oder eine verschiedene Eintragnummer steht, beende die Bearbeitung der Klammerung.

Falls hinter der schließenden Klammer eine neue Klammerung beginnt, so fahre mit Schritt 2 fort, sonst mit Schritt 1

Schritt 2 Stoßen zwei Klammerungen $(\alpha)^i$ und $(\beta)^i$ zusammen, so gibt es zwei Fälle:

a. $(\alpha)^i(\beta)^i = (\gamma)^i \ \forall i \in \mathbb{N}$ für ein $\gamma \in E^+$.
Mit Lemma 4.10 ist diese Bedingung äquivalent zu $\alpha\beta = \beta\alpha$ und demnach leicht zu testen. In diesem Fall können die beiden Klammerungen zu einer neuen zusammengefaßt werden, die nicht mehr nach rechts verschoben werden kann, denn es ist $\alpha[1] = \beta[1]$, und β kann nicht verschoben werden, da es schon behandelt wurde. Hier ist also die Bearbeitung der betrachteten Klammerung beendet.

b. $(\alpha)^i(\beta)^i = (\gamma)^i \ \forall i \in \mathbb{N}$ für kein $\gamma \in E^+$.
In diesem Fall findet man eine andere Darstellung von F_M, bei der die beiden Klammerungen nicht mehr zusammenstoßen:
Es sei $A = |\alpha|$, $B = |\beta|$ und $l = \mathrm{kgV}(A, B)$. Laut Lemma 4.11 gilt dann $(\alpha)^{l/A} \neq (\beta)^{l/B}$. In F_M werden nun alle Klammerungen $(\gamma)^i$ durch $(\gamma)^{l/B+i}$ ersetzt, was bei Anpassung von α'_0 möglich ist, ohne F_M und die bisher erhaltenen Eigenschaften zu verändern. Es liegt also folgende Situation vor:

$$\underbrace{\alpha \cdots \alpha}_{l/B\text{-mal}} (\alpha)^i \underbrace{\beta \cdots \beta}_{l/B\text{-mal}} (\beta)^i.$$

Da aber $(\alpha)^{l/A} \neq (\beta)^{l/B}$ ist, existiert keine Verschiebung der Klammerung $(\alpha)^i$ nach rechts, so daß $(\alpha)^i$ und $(\beta)^i$ wieder zusammenstoßen. Es kann daher mit Schritt 1 fortgefahren werden, ohne bei der betrachteten Klammerung nochmals Schritt 2 ausführen zu müssen.

Nach Anwendung des Verfahrens auf alle Klammerungen liegt eine Darstellung von F_M vor, die der Behauptung des Satzes genügt.

◇

Beispiel 4.12

Das im Beweis von Satz 4.9 vorgestellte Verfahren soll jetzt benutzt werden, um

$$F_{M3} = 0^+_-2^+10_- \prod_{i=1}^{\infty} [352^+(1)^i 1(1)^i 0_-(35)^i]$$

zu normieren.
Im ersten Schritt wird durch Verschiebung das Erreichen einer Extremstelle zu Beginn der Hauptschleife erzwungen.

$$\rightarrow \ F_{M3} = 0^+_-2^+10_-35 \prod_{i=1}^{\infty} [2^+(1)^i 1(1)^i 0_-(35)^i 35]$$

Als nächstes wird die Klammerung $(35)^i$ nach rechts verschoben

$$\rightarrow \ F_{M3} = 0^+_-2^+10_-35 \prod_{i=1}^{\infty} [2^+(1)^i 1(1)^i 0_-3(53)^i 5]$$

und danach $(53)^i$.

$$\rightarrow \ F_{M3} = 0^+_-2^+10_-35 \prod_{i=1}^{\infty} [2^+(1)^i 1(1)^i 0_-35(35)^i]$$

Die folgende Klammerung $(1)^i$ kann nicht verschoben werden, da ihr die Eintragnummer 0 folgt. Aber die Verschiebung der ersten Klammerung ist möglich.

$$\rightarrow \ F_{M3} = 0^+_-2^+10_-35 \prod_{i=1}^{\infty} [2^+1(1)^i (1)^i 0_-35(35)^i]$$

Schließlich wird $(1)^i(1)^i = (11)^i$ zusammengefaßt, und man erhält mit

$$\rightarrow \ F_{M3} = 0^+_-2^+10_-35 \prod_{i=1}^{\infty} [2^+1(11)^i 0_-35(35)^i]$$

die gewünschte Form.

4.3 Ein Algorithmus zur Vermutung der Laufformel

Für XMAS-Turingmaschinen M mit $F_M = \alpha_0 \prod_{i=1}^{\infty} [\alpha_1]$ und $a_1 = 0$ wurde schon in Abschnitt 3.4 ein Nichtterminationsbeweis entwickelt. Es werden also ab jetzt nur noch XMAS-Turingmaschinen betrachtet, in deren Hauptschleife Extremstellen erreicht werden. Diese bilden den Schlüssel zur schnellen Ermittlung (oder besser Vermutung) der allgemeinen Darstellung der Hauptschleife. Hierzu wird die zu untersuchende Turingmaschine M eine gewisse Anzahl s von Schritten simuliert und so die Folge F^s_M von Eintragnummern ermittelt. Bei Erreichen von Extremstellen werden die dazugehörigen Nummern mit $^+$ bzw. $_-$

markiert und die Zeitpunkte in einem Feld gemerkt. Nun werden Teilfolgen F aus F_M^s ermittelt, die selbst wieder aus drei Teilfolgen F_0, F_1 und F_2 bestehen. Dabei sollen die F_i mit der gleichen Extremstellen-Eintragnummer beginnen und ihre Längen linear anwachsen. Mittels der F_i ist es nun möglich, die allgemeine Hauptschleifendarstellung nach dem folgenden Verfahren, das sich auf Satz 4.9 stützt, zu vermuten:

Ist M eine XMAS-Turingmaschine, so gibt es gemäß Satz 4.9 eine Darstellung $F_M = \alpha_0 \prod_{i=1}^{\infty}[\alpha_1(\beta_1)^i\alpha_2\cdots(\beta_n)^i\alpha_{n+1}]$ mit $\alpha_{j+1}[1] \neq \beta_j[1]$ oder $\alpha_{j+1}[1]$ ist ein Extremstellen-Eintrag. Es wird nun angenommen, daß F_0, F_1 und F_2 Eintragnummernfolgen für ein i, $i+1$ und $i+2$ sind, also

$$\begin{aligned}
F_0 &= \alpha_1 \underbrace{\beta_1\cdots\beta_1}_{\text{i-mal}} \alpha_2 \underbrace{\beta_2\cdots\beta_2}_{\text{i-mal}} \alpha_3\cdots\underbrace{\beta_n\cdots\beta_n}_{\text{i-mal}} \alpha_{n+1}\\
F_1 &= \alpha_1 \underbrace{\beta_1\cdots\beta_1}_{\text{i-mal}} \beta_1\alpha_2 \underbrace{\beta_2\cdots\beta_2}_{\text{i-mal}} \beta_2\alpha_3\cdots\underbrace{\beta_n\cdots\beta_n}_{\text{i-mal}} \beta_n\alpha_{n+1}\\
F_2 &= \alpha_1 \underbrace{\beta_1\cdots\beta_1}_{\text{i-mal}} \beta_1\beta_1\alpha_2 \underbrace{\beta_2\cdots\beta_2}_{\text{i-mal}} \beta_2\beta_2\alpha_3\cdots\underbrace{\beta_n\cdots\beta_n}_{\text{i-mal}} \beta_n\beta_n\alpha_{n+1}
\end{aligned}$$

gilt.

F_1 soll bei erfolgreicher Termination des Algorithmus die allgemeine Hauptschleife darstellen. Daher ist es nur notwendig, die Positionen der Klammerungen festzustellen.

F_0 und F_1 werden jetzt von vorne her bis zum ersten Unterschied (Markierungen sind relevant) verglichen, um das β_1 zu lokalisieren. Dieser Vergleich liefert die Position der öffnenden Klammer vor dem letzten β_1 in F_1, denn es gilt $\alpha_2[1] \neq \beta_1[1]$ (oder bei $\alpha_2[1]$ wird eine Extremstelle erreicht). Vergleicht man F_1 mit F_2, so erhält man analog die schließende Klammer hinter dem letzten β_1 in F_1, und es ist zu prüfen, ob β_1 in F_2 noch einmal vorkommt. Die bis jetzt ermittelte Darstellung ist

$$(F_1 =)\ \alpha_1\beta_1\cdots\beta_1(\beta_1)\alpha_2\beta_2\cdots\beta_2\beta_2\alpha_3\cdots\alpha_{n+1}.$$

Startend bei den α_2 in den drei Folgen fährt der Algorithmus mit den Vergleichen und Markierungen fort. Schließich erhält man die gewünschte allgemeine Darstellung der Hauptschleife nach Ergänzung der Exponenten:

$$[\alpha_1\beta_1\cdots\beta_1(\beta_1)^j\alpha_2\beta_2\cdots\beta_2(\beta_2)^j\alpha_3\cdots(\beta_n)^j\alpha_{n+1}].$$

Die vermutete Darstellung der gesamten Laufformel ist demnach

$$F_M = \alpha_0' \prod_{j=1}^{\infty}[\alpha_1 \underbrace{\beta_1\cdots\beta_1}_{\text{i-mal}}(\beta_1)^j\alpha_2 \underbrace{\beta_2\cdots\beta_2}_{\text{i-mal}}(\beta_2)^j\alpha_3\cdots\underbrace{\beta_n\cdots\beta_n}_{\text{i-mal}}(\beta_n)^j\alpha_{n+1}]$$

für ein gewisses α_0'.

Der hier vorgestellte Algorithmus ist in C implementiert worden und im Anhang dokumentiert dargestellt.

Bemerkung 4.13

Wenn M eine XMAS-Turingmaschine ist, so gibt es wegen Satz 4.9 Teilfolgen F_0, F_1 und F_2 von F_M, so daß der Algorithmus die richtige Hauptschleifendarstellung vermutet.

Beispiel 4.14

Anhand von Turingmaschine M3 soll der vorgestellte Algorithmus illustriert werden. Es war

$$F_{M3}^{35} = 0\_2^+10\_352^+1110\_35352^+111110\_3535352^+111111.$$

Wir wählen

$$\begin{aligned}
F_0 &= 2^+10_35\\
F_1 &= 2^+1110_3535\\
F_2 &= 2^+111110_353535.
\end{aligned}$$

Wie gefordert beginnen die F_j mit einer Extremstellen-Eintragnummer, und ihre Länge nimmt linear jeweils um vier zu. Der Algorithmus vergleicht F_0 mit F_1 bis zum ersten Unterschied und markiert dort in F_1 die Position der ersten öffnenden Klammer.

$$\rightarrow\ F_1 = 2^+1(110\_3535$$

Danach wird F_1 mit F_2 verglichen, und man erhält nach Prüfung, ob in F_2 11 noch einmal vorkommt, die Position der schließenden Klammer.

$$\rightarrow\ F_1 = 2^+1(11)0\_3535$$

In den nächsten Schritten vergleicht der Algorithmus wieder bis zu den ersten Unterschieden in den Folgen, um die nächste Klammerung zu ermitteln.

$$\rightarrow\ F_1 = 2^+1(11)0\_35(35)$$

Dieses führt schließlich zu der Vermutung

$$F_M = 0\_2^+10\_35\prod_{i=1}^{\infty}[2^+1(11)^i0\_35(35)^i].$$

4.4 Verifikation der vermuteten Laufformel

Im letzten Abschnitt ist eine Vermutung geäußert worden, die jetzt bewiesen werden soll. Es wird das Verfahren VERIFIKATION vorgestellt, mit dem mittels vollständiger Induktion die Gültigkeit der vermuteten Laufformel

$$\alpha_0 \prod_{i=1}^{\infty} [\alpha_1(\beta_1)^i \alpha_2(\beta_2)^i \cdots (\beta_n)^i \alpha_{n+1}] \;=:\; \alpha_0 \prod_{i=1}^{\infty} [\mathrm{F_i}]$$

beweisen werden kann.

Hierzu kann die Bearbeitung von α_0, an dessen Ende F_0 vorkommt, F_1 und F_2 angenommen werden, weil aus diesen (in einer Simulation abgearbeiteten) Folgen die Vermutung entstanden ist. In zwei Phasen wird nun versucht, die folgende Behauptung für alle $i \in \mathbb{N}$ zu zeigen:

> Wenn $\mathrm{F_i}$ korrekt bearbeitet wird (d.h. alle Einträge in $\mathrm{F_i}$ werden abgearbeitet und Extremstellen werden genau bei den Extremstellen-Einträgen erreicht), dann wird danach $\mathrm{F_{i+1}}$ korrekt bearbeitet.

In der ersten Phase wird die allgemeine (d.h. von i unabhängige) Bearbeitung von $\mathrm{F_i}$ auf einem symbolischen Band nachvollzogen, auf dem Folgen von Zeichen abgekürzt werden können. Nach der Abarbeitung steht der allgemeine von $\mathrm{F_i}$ hinterlassene Bandinhalt zur Verfügung. Auf diesem wird in einer zweiten Phase die Abarbeitung von $\mathrm{F_{i+1}}$ nachgespielt. Führt die Turingmaschine M genau die Schritte aus, die durch $\mathrm{F_{i+1}}$ gegeben sind, so ist $\mathrm{F_M} = \alpha_0 \prod_{i=1}^{\infty} [\mathrm{F_i}]$ gezeigt. Damit ist M eine XMAS-Turingmaschine und stoppt nicht.

Im Anschluß soll zunächst der neue Bandtyp eingeführt werden. Danach erfolgt die detaillierte Beschreibung der zwei Phasen, deren Korrektheit sich auf die Lemmata am Ende dieses Abschnitts stützt. Das Gesamtergebnis wird schließlich in einem Satz zusammengefaßt und an einem ausführlichen Beispiel illustriert.

Mögliche Inschriften des symbolischen Bandes sind:

- \$ (undefiniert) als Inhalt einer noch nicht bedruckten Bandposition.
- 0 und 1 als Zeichen des Arbeitsalphabetes.
- (u) mit $u \in \{0,1\}^+$ als Abkürzung für das i-fache Vorkommen des Wortes u auf dem originalen Band. Hierbei ist $i \in \mathbb{N}$ beliebig, aber in beiden Phasen gleich.

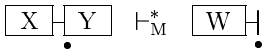
Es wird im folgenden ein endlicher Bandausschnitt betrachtet. Bei Überschreitung einer Grenze innerhalb der beiden Phasen bricht das Verfahren erfolglos ab. Die aktuelle Position ist in der Darstellung des Bandausschnittes durch einen Punkt markiert. Eine typische Situation ist z.B.

$$[\$\$\$\$\dot{\$}1(11)1\$\$\$\$\$].$$

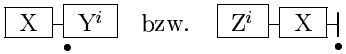
1.Phase Als Ausgangspunkt der zweiten Phase dient der von $\mathrm{F_i}$, dessen korrekte Bearbeitung angenommen wird, hinterlassene Bandinhalt. Durch die allgemeine Ausführung von $\mathrm{F_i}$ auf dem überall undefinierten symbolischen Band soll dieser Bandinhalt ermittelt werden. Hierzu wird $\mathrm{F_i}$ in eine von i unabhängige Form F gebracht, indem die Exponenten entfallen. Alle Klammerungen in F und auf dem Band stehen im folgenden für das i-fache Vorkommen der Inhalte. Die entscheidende Eigenschaft der Eintragnummern-Folgen ist die Rekonstruierbarkeit des von ihnen erwarteten Bandinhaltes. Damit ist es möglich, die Ausführung einer solchen Folge zu verifizieren: Beginnend mit dem überall undefinierten Band wird F Eintrag für Eintrag abgearbeitet. Dabei können die nachstehenden Fälle eintreten:

1. $\mathrm{F}[k] = e, e^+, e_-$
   Gemäß der Definition gilt $e = q \cdot 2 + \mathrm{bit}$, wobei q der aktuelle Zustand ist und bit den aktuellen Bandinhalt unter dem Kopf beschreibt. Damit erwartet der Eintrag e das Zeichen bit auf dem symbolischen Band. Für den vorliegenden Bandinhalt gibt es drei Möglichkeiten:
   - \$ – Hier wird das erwartete Zeichen auf das Band geschrieben, denn es wird angenommen, daß $\mathrm{F_i}$ ausgeführt wird.
   - 0,1 – Der Bandinhalt wird mit dem erwarteten Zeichen verglichen. Das Verfahren bricht erfolglos ab, wenn die Zeichen unterschiedlich sind.
   - (,) – Da von e ein Zeichen aus dem Arbeitsalphabet erwartet wird, muß zunächst die Klammerung auf dem Band verschoben werden. Wenn dieses nicht gelingt, sich also das Zeichen hinter der Klammerung von dem ersten in der Klammerung unterscheidet, dann bricht das Verfahren erfolglos ab.

   Nun kann der zu e gehörige Eintrag bearbeitet werden, d.h. das aktuelle Zeichen wird überschrieben und der Kopf bewegt. Der erreichte Zustand ist durch die nächste Eintragnummer bestimmt. Auf Extremstellen-Markierungen (e^+, e_-) wird in der ersten Phase nicht eingegangen, denn gemäß Voraussetzung werden in $\mathrm{F_i}$ alle Extremstellen genau bei den Extremstellen-Einträgen erreicht.
2. $\mathrm{F}[k] = ($
   In dieser Situation muß geprüft werden, ob die vorliegende Klammerung (β_j) i-mal ausgeführt wird. Dazu stellt der Algorithmus SCHLEIFEN-ANALYSE fest, welchen Bandinhalt die allgemeine Bearbeitung erwartet und welchen sie hinterläßt. Liegt der erwartete Bandinhalt vor, so wird er durch die Ausgabe ersetzt und die Ausführung von F hinter der schließenden Klammer) fortgesetzt. Im anderen Fall terminiert das Verfahren erfolglos.
   Zur Ermittlung der Ein- und Ausgabe von $(\beta_j)^i$ werden zunächst die von β_j bestimmt, indem β_j auf einem überall undefinierten Band bearbeitet wird. Ist eine vorher noch nicht besuchte Bandposition erreicht worden, so merkt man sich das dort erwartete Zeichen auf einem zusätzlichen Band. Wenn man $b_j > 0$ annimmt (d.h. β_j bewegt den Kopf insgesamt nach rechts), dann ergibt sich folgende von β_j durchgeführte Bandveränderung:



Hiebei gibt die linke Punktmarkierung die Position zu Beginn der Bearbeitung von β_j an. Daß der Kopf am Ende rechts von der erwarteten Eingabe steht und $\boxed{W} = \boxed{Z}\boxed{X}$ für eine gewisse Zelle $\boxed{Z}$ gilt, zeigt Lemma 4.16. Weiterhin wird dort bewiesen, daß die allgemeine Eingabe und Ausgabe von $(\beta_j)^i$ durch



gegeben ist. Wenn nun gezeigt werden kann, daß die allgemeine Eingabe (in Klammerform) tatsächlich vorliegt, so kann diese durch die Ausgabe ersetzt werden und die Bearbeitung von F fortfahren. Anderenfalls bricht das Verfahren mit einer Fehlermeldung ab.

2.Phase Nach erfolgreichem Abschluß der ersten Phase liegt die allgemeine Ausgabe von F_i vor. Es muß geprüft werden, ob F_{i+1} darauf korrekt bearbeitet wird. Im Unterschied zur ersten Phase wird jetzt eine Eintragfolge F' ausgeführt, die aus F entsteht, indem jedes β_j einmal explizit vor der Klammerung (β_j) eingefügt wird. Damit beschreibt F' die allgemeine Bearbeitung von F_{i+1}. Bei der Bearbeitung von F' sind nun Extremstellen-Markierungen von Eintragnummern relevant. Undefinierte Bandpositionen müssen genau bei den Extremstellen-Einträgen erreicht werden. Dabei wird der Bandinhalt auf 0 gesetzt, und die Bearbeitung kann fortfahren. Lemma 4.17 rechtfertigt dieses Vorgehen.

Eine Implementierung des vorgestellten Verfahrens ist in den Anhang aufgenommen worden. Zusammenfassend kann folgender Satz zur Verifikation der Laufformel-Vermutung ausgesprochen werden:

Satz 4.15

Sei eine Turingmaschine M und eine vermutete Laufformel

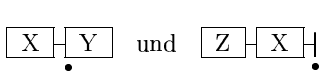
$$F'_M = \alpha_0 \prod_{i=1}^{\infty} [\alpha_1(\beta_1)^i \alpha_2(\beta_2)^i \cdots (\beta_n)^i \alpha_{n+1}] \;=:\; \alpha_0 \prod_{i=1}^{\infty} [F_i]$$

in der Normalform von Satz 4.9 gegeben, mit der Bedingung, daß Extremstellen in der Hauptschleife erreicht werden. Weiterhin soll die Berechnung von M auf ϵ mit α_0, an dessen Ende F_0 vorkommt, F_1 und F_2 beginnen und die Extremstellen dort genau von den Extremstellen-Einträgen erreicht werden. Wenn das vorgestellte Verfahren VERIFIKATION angesetzt auf M und F'_M erfolgreich terminiert, so ist $F_M = F'_M$ und damit bewiesen, daß M nicht stoppt.

Auch hier werden dem Beweis zwei Hilfsaussagen vorangestellt.

Lemma 4.16

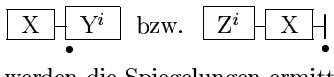
Es seien F_0, F_1 und F_2 von der Turingmaschine M bearbeitet worden. (β_j) sei eine Klammerung der Hauptschleife und hinter dieser komme ein Eintrag $e \neq \beta_j[1]$ oder ein Extremstellen-Eintrag e^+ (e_-) vor. Dann gilt $b_j \neq 0$, und das vorgestellte Verfahren ermittelt für $b_j > 0$



als erwartete Eingabe von β_j bzw. als den von β_j hinterlassenen Bandinhalt für gewisse Zellen

$$\boxed{X}, \boxed{Y} \text{ und } \boxed{Z} \text{ mit } \text{len}\,\boxed{Y} = \text{len}\,\boxed{Z} = |b_j| > 0.$$

Die allgemeine Ein- und Ausgabe von $(\beta_j)^i$ ist für alle $i \in \mathbb{N}$ durch



gegeben. Im Fall $b_j < 0$ werden die Spiegelungen ermittelt.

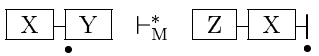
Beweis:

$b_j \neq 0$ folgt für alle j aus der Abarbeitung von F_1 und F_2, denn in F_2 wird $(\beta_j)^2$ für alle j bearbeitet. Aus $b_j = 0$ für ein j folgt dann, daß nach β_j immer β_j bearbeitet wird. Dieses ist aber ein Widerspruch zur Abarbeitung von F_1. Es sei im folgenden $b_j > 0$.
Als nächstes wird gezeigt, daß der Kopf nach Bearbeitung von β_j auf einer Position steht, die währenddessen nicht besucht wurde. Hierbei wird die Normalform der Hauptschleifendarstellung ausgenutzt. Es gibt zwei Fälle:

1. Hinter der Klammerung tritt eine Eintragnummer $e \neq \beta_j[1]$ auf. Der Kopf befindet sich nach Abarbeitung von β_j wegen $b_j > 0$ rechts von der Startposition. Würde er sich auf einer schon bedruckten Position befinden, so müßte hinter der Klammerung der Eintrag $e = \beta_j[1]$ vorkommen, denn der Bandinhalt am Ende der Bearbeitung von β_j wäre festgelegt und wegen der Bearbeitung von $(\beta_j)^2$ in F_2 auch der nun auszuführende Eintrag e. Dieses ist ein Widerspruch zu $e \neq \beta_j[1]$. Somit wurde die schließlich erreichte Position vorher noch nicht besucht.
2. Hinter der Klammerung tritt eine Extremstellen-Eintragnummer e^+ auf. In F_1 (oder F_2, falls $j = n$ und $\alpha_{n+1} = \epsilon$ ist) wurde nach Bearbeitung von $(\beta_j)^1$ e^+ ausgeführt. Da hierbei eine Extremstelle erreicht wurde, ist klar, daß eine in β_j noch nicht bedruckte Position nach der Bearbeitung von β_j betreten wird.

Im nächsten Schritt wird gezeigt, daß sich links von der Endposition auf dem Band die Zelle $\boxed{X}$ befinden. Es kann vorkommen, daß sich der Kopf bei der Ausführung von β_j zeitweise links von der Startposition befindet. Diesen Umstand erkennt man an $\text{len}\,\boxed{X} \neq 0$. In F_2 wird jedes β_j zweimal bearbeitet. Hierzu muß gewährleitet sein,

daß nach Ausführung des ersten β_j links von der aktuellen Position $\boxed{\text{X}}$ auf dem Band steht, da dieser Bandinhalt von dem zweiten β_j erwartet wird.
Insgesamt erfolgt also durch β_j die Transformation



für gewisse Zellen

$$\boxed{\text{X}}\,,\ \boxed{\text{Y}}\ \text{und}\ \boxed{\text{Z}}\ \text{ mit }\ \text{len}\,\boxed{\text{Y}} = \text{len}\,\boxed{\text{Z}} = b_j.$$

Durch vollständige Induktion kann man leicht die allgemeine Transformation zeigen, und die Aussagen im Fall $b_j < 0$ beweist man analog.

◇

Lemma 4.17

Von der Turingmaschine M seien F_0, F_1, F_2 und F_i korrekt bearbeitet worden, d.h. alle Eintragfolgen sind abgearbeitet worden, und Extremstellen wurden genau bei den Extremstellen-Einträgen erreicht. Zusätzlich sei $\text{F}_{\text{i}+1}$ bis zu einem Extremstellen-Eintrag e^+ (e_-) korrekt bearbeitet worden. Dann wird bei e^+ (e_-) eine Extremstelle erreicht.

Beweis:

Zunächst wird $\sum_{j=1}^{n} b_j = 0$ gezeigt, um die Argumentation von i unabhängig zu machen:
Gemäß der Normalformdarstellung aus Satz 4.9 beginnen die F_i mit einem Extremstellen-Eintrag $f^+(f_-)$. Betrachtet man die Bandpositionen pos_0 und pos_1 bei Erreichen dieses Eintrages in F_0 bzw. F_1, so gilt $\text{pos}_1 - \text{pos}_0 = \Delta_{\max}$ ($\Delta_{\min}$) := Anzahl der Maximum(Minimum)-Extremstellen-Einträge in der Hauptschleife. Untersucht man die entsprechenden Positionen in F_1 und F_2, so ergibt sich analog $\text{pos}_2 - \text{pos}_1 = \Delta_{\max}$ ($\Delta_{\min}$). In F_1 werden alle Einträge aus F_0 abgearbeitet, aber zusätzlich alle Klammerungen einmal mehr. Somit gilt auch

$$\text{pos}_2 - \text{pos}_1 = \text{pos}_1 - \text{pos}_0 + \sum_{j=1}^{n} b_j.$$

Hieraus folgt $\sum_{j=1}^{n} b_j = 0$.
Im nächsten Schritt wird hiermit gezeigt, daß bei e^+ eine Extremstelle vorliegt. Die aktuelle Bandposition sei mit pos_{i+1} und die bei Erreichen des entsprechenden Eintrages e^+ in F_i mit pos_i bezeichnet. Für diese beiden Positionen gilt wegen der korrekten Bearbeitung von F_i und $\text{F}_{\text{i}+1}$ bis hin zu e^+

$$\text{pos}_{i+1} = \text{pos}_i + \Delta_{\max} + i \cdot \sum_{j=1}^{n} b_j = \text{pos}_i + \Delta_{\max}\ .$$

Es muß gezeigt werden, daß diese Position nicht schon früher betreten wurde. Nach der Bearbeitung von e^+ in F_i sind genau $\Delta_{\max} - 1$ Maximum-Extremstellen aufgetreten. Damit sind alle bis zum aktuellen e^+ in $\text{F}_{\text{i}+1}$ erreichten Bandpositionen kleiner als $\text{pos}_i + \Delta_{\max}$. Somit wird bei e^+ eine Extremstelle erreicht. Analog beweist man die Aussage für e_-.

◇

Beweis von Satz 4.15:

Wenn das Verfahren VERIFIKATION erfolgreich terminiert, so ist mittels Lemma 4.17 und Lemma 4.16 folgende Aussage für alle $i \in \mathbb{N}$ gezeigt:

> Wenn F_i (nach $\text{F}_{\text{i-1}}$) korrekt bearbeitet wird, dann wird $\text{F}_{\text{i}+1}$ nach F_i korrekt bearbeitet.

Gemäß der Voraussetzung wird F_1 nach F_0 korrekt ausgeführt. Damit ist für alle $i \in \mathbb{N}$

> Nach F_i wird $\text{F}_{\text{i}+1}$ korrekt bearbeitet

gezeigt. Hieraus folgt mittels vollständiger Induktion die Gültigkeit der Laufformel-Vermutung. M ist demnach eine XMAS-Turingmaschine und stoppt daher angesetzt auf ϵ nicht.

◇

Beispiel 4.18

Am Beispiel der Turingmaschine M3 soll die Arbeitsweise des vorgestellten Beweisverfahrens VERIFIKATION ausführlich demonstriert werden. M3 ist durch die Turingtafel

	0	1
0	1R1	1L0
1	1L0	1R2
2	1RH	1R1

mit Eintragnummern

	0	1
0	0	1
1	2	3
2	4	5

gegeben. In Beispiel 4.14 war

$$\text{F}_{\text{M3}} = 0^+_-2^+10\_35\prod_{i=1}^{\infty}[2^+1(11)^i0\_35(35)^i] \ =:\ 0^+_-2^+10\_35\prod_{i=1}^{\infty}[\text{F}_\text{i}]$$

mittels der Bearbeitung der Eintragfolgen F_0, F_1 und F_2 vermutet worden. VERIFIKATION zeigt die Gültigkeit der Vermutung in zwei Phasen.

In der ersten wird die allgemeine Bearbeitung von F_i auf dem überall undefinierten, symbolischen Band simuliert. Der aktuelle Stand der Simulation ist gegeben durch die mit den aktuellen Positionen markierten Hauptschleife und Band. Zunächst sind Extremstellen-Markierungen nicht relevant. Die Ausführung beginnt mit

$$[\dot{2}1(11)035(35)]\ ,\ [\$\$\$\$\$\$\$\$\$\dot{\$}].$$

Die Bearbeitung des Eintrages mit Nummer 2 erwartet eine 0 auf dem Band – es liegt aber ein undefinierter Bandinhalt vor. In der ersten Phase wird in dieser Situation \$ durch das erwartete Zeichen ersetzt.

$$\rightarrow [\dot{2}1(11)035(35)]\ ,\ [\$\$\$\$\$\$\dot{0}\$\$\$]$$

Eintrag 2 schreibt eine 1 auf das Band und bewegt den Kopf nach links.

$$\rightarrow [2\dot{1}(11)035(35)]\ ,\ [\$\$\$\$\$\$\dot{1}\$\$\$]$$

Anschließend wird der nächste Eintrag bearbeitet. Die folgenden Schritte sind:

$$\rightarrow [2\dot{1}(11)035(35)]\ ,\ [\$\$\$\$\$\dot{1}1\$\$\$]$$
$$\rightarrow [21\dot{(}11)035(35)]\ ,\ [\$\$\$\$\$\dot{1}1\$\$\$].$$

Das Verfahren SCHLEIFEN-ANALYSE ermittelt für die Klammerung $(11)^i$ den erwarteten Bandinhalt $(11)^i_{\cdot}$, der auf dem Band durch $(11)_{\cdot}$ abgekürzt wird. Auf dem Band sind die zu prüfenden Positionen undefiniert. Damit kann der von der Klammerung hinterlassene Bandinhalt $(_{\cdot}(11)^i$ abgekürzt durch $_{\cdot}(11))$ auf das Band kopiert werden.

$$\rightarrow [21(11)\dot{0}35(35)]\ ,\ [\$\$\$\dot{\$}(11)11\$\$\$]$$

Die Bearbeitung der ersten Klammerung ist hiermit beendet. Als nächstes wird Eintrag 0 behandelt: Er erwartet eine 0 auf dem Band, schreibt eine 1 und bewegt den Kopf nach rechts.

$$\rightarrow [21(11)\dot{0}35(35)]\ ,\ [\$\$\dot{0}(11)11\$\$\$]$$
$$\rightarrow [21(11)0\dot{3}5(35)]\ ,\ [\$\$1\dot{(}11)11\$\$\$]$$

Eintrag 3 erwartet eine 1, schreibt eine 1 und bewegt den Kopf nach rechts. Da aber auf der aktuellen Bandposition eine Klammer steht, muß diese zunächst verschoben werden, weil ein Zeichen des Arbeitsalphabetes erwartet wird. Dieses gelingt, da das erste Zeichen hinter der schließenden Klammer mit dem ersten Zeichen hinter der öffnenden Klammer übereinstimmt.

$$\rightarrow [21(11)0\dot{3}5(35)]\ ,\ [\$\$\dot{1}1(11)1\$\$\$]$$
$$\rightarrow [21(11)03\dot{5}(35)]\ ,\ [\$\$\dot{1}1\dot{(}11)1\$\$\$]$$

Wie bei dem vorherigen Eintrag muß die Klammerung auf dem Band zunächst verschoben werden.

$$\rightarrow [21(11)03\dot{5}(35)]\ ,\ [\$\$\dot{1}11(11)\$\$\$]$$
$$\rightarrow [21(11)035\dot{(}35)]\ ,\ [\$\$\dot{1}11\dot{(}11)\$\$\$]$$

Als erwarteten bzw. hinterlassenen Bandinhalt von $(35)^i$ bestimmt SCHLEIFEN-ANALYSE $(11)^i$ bzw. $(11)^i_{\cdot}$ (abgekürzt durch (11) und $(11)_{\cdot}$). Der benötigte Bandinhalt ist vorhanden und wird durch die Ausgabe ersetzt.

$$\rightarrow [21(11)035(35)]\dot{\ }\ ,\ [\$\$\dot{1}11(11)\dot{\$}\$\$]$$

Hiermit ist die erste Phase beendet, und der Bandinhalt nach Abarbeitung von F_i steht fest.

In der zweiten Phase muß nun geprüft werden, ob F_{i+1} wie gewünscht auf der Ausgabe von F_i arbeitet. Alle Klammerungen werden einmal mehr ausgeführt, und Extremstellen-Markierungen sind relevant. Die Ausgangssituation ist demnach

$$\rightarrow [\dot{2}^+111(11)0\_3535(35)]\ ,\ [\$\$111(11)\dot{\$}\$\$].$$

Mit 2^+ liegt ein Extremstellen-Eintrag vor, und gemäß Lemma 4.17 wird eine Extremstelle erreicht. Damit ist der Bandinhalt 0, und der Algorithmus kann fortfahren.

$$\rightarrow [\dot{2}^+111(11)0\_3535(35)]\ ,\ [\$\$\dot{1}11(11)\dot{0}\$\$]$$
$$\rightarrow [2^+\dot{1}11(11)0\_3535(35)]\ ,\ [\$\$\dot{1}11(11)\dot{1}\$\$]$$

Bis zur ersten Klammerung ergeben sich die folgenden Veränderungen:

$$\rightarrow [2^+\dot{1}11(11)0\_3535(35)]\ ,\ [\$\$\dot{1}1(11)\dot{1}1\$\$]$$
$$\rightarrow [2^+1\dot{1}1(11)0\_3535(35)]\ ,\ [\$\$\dot{1}1(11)\dot{1}1\$\$]$$
$$\rightarrow [2^+1\dot{1}1(11)0\_3535(35)]\ ,\ [\$\$\dot{1}1(11)\dot{1}11\$\$]$$
$$\rightarrow [2^+11\dot{1}(11)0\_3535(35)]\ ,\ [\$\$\dot{1}1(11)\dot{1}111\$\$]$$
$$\rightarrow [2^+11\dot{1}(11)0\_3535(35)]\ ,\ [\$\$\dot{1}1(11)\dot{1}111\$\$]$$
$$\rightarrow [2^+111\dot{(}11)0\_3535(35)]\ ,\ [\$\$\dot{1}1\dot{(}11)1111\$\$].$$

Wie in der ersten Phase erhält man für $(11)^i$ die Eingabe $(11)^i_{\cdot}$ und Ausgabe $_{\cdot}(11)^i$. Der verlangte Bandinhalt liegt vor und kann demnach durch die Ausgabe ersetzt werden.

$$\rightarrow [2^+111(11)\dot{0}\_3535(35)]\ ,\ [\$\$\dot{\$}(11)1111\$\$]$$

Mit dem Erreichen einer undefinierten Bandposition muß geprüft werden, ob ein Extremstellen-Eintrag erreicht ist. Dieses ist der Fall, und daher liegt hier eine Extremstelle vor – der Bandinhalt ist 0.

$$\rightarrow [2^+111(11)\dot{0}\_3535(35)]\ ,\ [\$\$\dot{0}(11)1111\$\$]$$
$$\rightarrow [2^+111(11)0\_\dot{3}535(35)]\ ,\ [\$\$1\dot{(}11)1111\$\$]$$

Bis zur nächsten Klammerungen erhält man

$$\rightarrow [2^+111(11)0\_\dot{3}535(35)]\ ,\ [\$\$1\dot{1}(11)111\$\$]$$
$$\rightarrow [2^+111(11)0\_3\dot{5}35(35)]\ ,\ [\$\$11\dot{(}11)111\$\$]$$
$$\rightarrow [2^+111(11)0\_35\dot{3}5(35)]\ ,\ [\$\$111\dot{1}(11)11\$\$]$$
$$\rightarrow [2^+111(11)0\_353\dot{5}(35)]\ ,\ [\$\$111\dot{(}11)11\$\$]$$
$$\rightarrow [2^+111(11)0\_3535(35)]\ ,\ [\$\$1111\dot{1}(11)1\$\$]$$
$$\rightarrow [2^+111(11)0\_3535\dot{(}35)]\ ,\ [\$\$1111\dot{(}11)1\$\$]$$
$$\rightarrow [2^+111(11)0\_3535\dot{5}(35)]\ ,\ [\$\$11111\dot{1}(11)\$\$]$$
$$\rightarrow [2^+111(11)0\_3535(35)]\ ,\ [\$\$11111\dot{(}11)\$\$].$$

Die Klammerung $(35)^i$ erwartet $\dot{(}11)^i$ und hinterläßt $(11)^i\dot{.}$.

$$\rightarrow [2^+111(11)0\_3535(35)\dot{]}\ ,\ [\$\$11111(11)\dot{\$}\$]$$

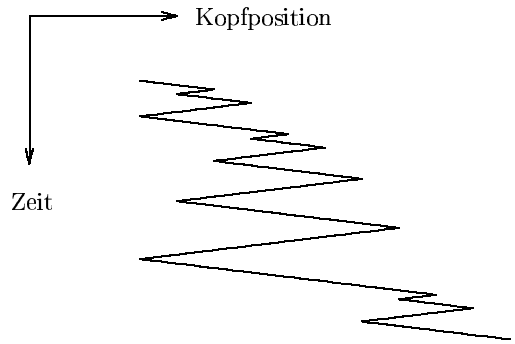
Somit wird F_{i+1} korrekt bearbeitet, und die Gültigkeit der Vermutung ist mit Satz 4.15 bewiesen. M3 ist demnach eine XMAS-Turingmaschine und stoppt nicht.

5 Nichtterminationsbeweis durch Bandbetrachtung

In den folgenden beiden Abschnitten kommt eine Methode zur Anwendung, die schon in [Bra83] benutzt wurde, um die Nichttermination bei einfachen XMAS- und COUNTER-Turingmaschinen zu beweisen. Dabei wird das Band in Abschnitte (Zellen) eingeteilt und untersucht, wie diese durch die vorgelegte Turingmaschine verändert werden. Lassen sich diese Veränderungen in Regeln zusammenfassen und sich zusätzlich gewährleisten, daß die Turingmaschine, die diesen Regeln genügt, nicht stoppt, so ist ein neues Verfahren zum Beweis der Nichttermination entstanden. Es muß "nur" noch geprüft werden, ob eine zu untersuchende Turingmaschine das Band regelgerecht verändert.

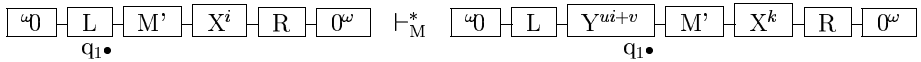
5.1 DRAGON-Turingmaschinen

Die Arbeitsweise der sogenannten DRAGON-Turingmaschinen läßt sich am besten anhand eines Positions-Zeit-Diagrammes darstellen:



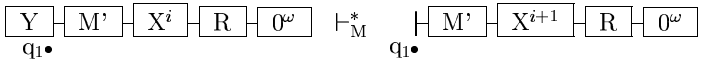
Nachstehend wird ein Nichtterminationsbeweis für einfache DRAGON-Turingmaschinen entwickelt. Das Verhalten der betrachteten Turingmaschinen kann durch folgende beiden Regeln charakterisiert werden:

Regel 1: $\forall i \in \mathbb{N}$



mit $u \in \mathbb{N},\ v, k \in \mathbb{N}_0$ fest und $u \cdot \text{len}\boxed{\text{Y}} = \text{len}\boxed{\text{X}}$.

Regel 2: $\forall i \in \mathbb{N}$



Hierbei wird durch Regel 1 die Aktion nach Erreichen des linken Randes beschrieben. Im wesentlichen wird dabei eine Kette von X-Zellen durch eine Kette von Y-Zellen ersetzt. Im Anschluß daran kann Regel 2 angewandt werden, die beschreibt, wie durch eine Zick-Zack-Bewegung eine Y-Zelle verbraucht und eine X-Zelle erzeugt wird. Der folgende Satz klärt die Stoppeigenschaft der so arbeitenden Turgingmaschinen.

Satz 5.1

Erfüllt M die Regeln 1 und 2 für bestimmte Zellen $\boxed{\text{L}}$, $\boxed{\text{M'}}$, $\boxed{\text{X}}$, $\boxed{\text{Y}}$, $\boxed{\text{R}}$ und erreicht M während der Berechnung eine Konfiguration

$$\boxed{\text{“}0}\boxed{\text{L}}\boxed{\text{M'}}\boxed{\text{X}^i}\boxed{\text{R}}\boxed{0^\omega} \quad \text{mit } i \in \mathbb{N},$$
$$q_1\bullet$$

so stoppt M nicht.

Beweis:

Ausgehend von der gegebenen Konfiguration wird Regel 1 angewandt. M erreicht die Konfiguration

$$\boxed{\text{“}0}\boxed{\text{L}}\boxed{\text{Y}^{ui+v}}\boxed{\text{M'}}\boxed{\text{X}^k}\boxed{\text{R}}\boxed{0^\omega}.$$
$$q_1\bullet$$

Nun kann $(ui+v)$-mal Regel 2 angewandt werden. Damit wird schließlich

$$\boxed{\text{“}0}\boxed{\text{L}}\boxed{\text{M'}}\boxed{\text{X}^{ui+v+k}}\boxed{\text{R}}\boxed{0^\omega}$$
$$q_1\bullet$$

erreicht.
Nach Bearbeitung der vorgegebenen Konfiguration mittels der Regeln 1 und 2 ist also wieder eine Konfiguration vom geforderten Typ entstanden – die Turingmaschine stoppt nicht. Denn nimmt man ihr Halten an, so gibt es in der Berechnung eine letzte Konfiguration von obigem Typ. Dieses steht aber im Widerspruch zu der gerade dargelegten Tatsache, daß nach Auftreten einer derartigen Konfigurationen stets eine weitere erreicht wird.

$\diamond$

Mittels der aus einer Simulation von M gemutmaßten Zellen $\boxed{\text{L}}$ bis $\boxed{\text{R}}$ und des Zustandes q_1 wird nun ein Weg aufgezeigt, wie die beiden Regeln verifiziert werden können. Dazu werden zwei Hilfsaussagen benötigt.

Lemma 5.2

Seien $\boxed{\text{S}}$, $\boxed{\text{T}}$, $\boxed{\text{X}}$, $\boxed{\text{Y}}$ Zellen und $\boxed{\text{X}} \neq \boxed{\epsilon}$. Dann gilt:

$$\boxed{\text{S}}\boxed{\text{X}^i}\boxed{\text{T}} \in \{\, \boxed{\text{Y}} \,\}^* \quad \forall i \in \mathbb{N}_0$$
$$\Leftrightarrow$$
$$\text{len}\ \boxed{\text{X}} = u \cdot \text{len}\ \boxed{\text{Y}} \quad \text{für ein } u \in \mathbb{N}$$
$$\text{und es existiert eine Zerlegung } \boxed{\text{X}} = \boxed{\text{A}}\boxed{\text{B}} \text{ mit}$$
$$\boxed{\text{B}}\boxed{\text{A}}, \boxed{\text{S}}\boxed{\text{A}}, \boxed{\text{B}}\boxed{\text{T}} \in \{\, \boxed{\text{Y}} \,\}^*$$

Beweis:

“$\Rightarrow$” Aus $\boxed{\text{S}}\boxed{\text{X}^i}\boxed{\text{T}} \in \{\, \boxed{\text{Y}} \,\}^* \ \forall i \in \mathbb{N}_0$ folgt direkt $\text{len}\ \boxed{\text{X}} = u \cdot \text{len}\ \boxed{\text{Y}}$ für ein $u \in \mathbb{N}$.

1. Fall $\quad \text{len}\ \boxed{\text{Y}} \mid \text{len}\ \boxed{\text{S}}$

   $\Rightarrow \text{len}\ \boxed{\text{Y}} \mid \text{len}\ \boxed{\text{T}}$

   Aus der Voraussetzung ergibt sich dann mit $i = 0$

   $\boxed{\text{S}}, \boxed{\text{T}} \in \{\, \boxed{\text{Y}} \,\}^*$

   und mit $i = 1$ schließlich

   $\boxed{\text{X}} \in \{\, \boxed{\text{Y}} \,\}^*$.

   Die Zerlegung $\boxed{\text{A}} = \boxed{\epsilon}$, $\boxed{\text{B}} = \boxed{\text{X}}$ leistet das Gewünschte.

2. Fall $\quad \text{len}\ \boxed{\text{Y}} \nmid \text{len}\ \boxed{\text{S}}$

   $\Rightarrow \text{len}\ \boxed{\text{Y}} \nmid \text{len}\ \boxed{\text{T}}$

   $\Rightarrow \boxed{\text{S}} = \boxed{\text{Y}^k}\boxed{\text{C}}$ (k maximal und $\boxed{\text{C}} \neq \boxed{\epsilon}$)

   $\boxed{\text{T}} = \boxed{\text{D}}\boxed{\text{Y}^l}$ (l maximal und $\boxed{\text{D}} \neq \boxed{\epsilon}$)

   Hieraus erhält man mit der Voraussetzung

   $\boxed{\text{C}}\boxed{\text{D}} = \boxed{\text{Y}}$ (wegen $\boxed{\text{S}}\boxed{\text{T}} \in \{\, \boxed{\text{Y}} \,\}^*$)

   und

   $\boxed{\text{C}}\boxed{\text{X}}\boxed{\text{D}} \in \{\, \boxed{\text{Y}} \,\}^*$

   und damit

   $\boxed{\text{X}} = \boxed{\text{D}}\boxed{\text{Y}^{u-1}}\boxed{\text{C}}$.

   Die Zerlegung $\boxed{\text{A}} = \boxed{\text{D}}$ und $\boxed{\text{B}} = \boxed{\text{Y}^{u-1}}\boxed{\text{C}}$ hat dann die geforderten Eigenschaften.

"⇐" Sei $i \geq 1$, dann ist

$$\boxed{S}\boxed{X^i}\boxed{T} = \boxed{S}\boxed{A}\boxed{C^{i-1}}\boxed{B}\boxed{T} \text{ mit } \boxed{C} = \boxed{B}\boxed{A}$$

ein Wort aus $\{ \boxed{Y} \}^*$, denn nach Voraussetzung gilt

$$\boxed{C}, \boxed{S}\boxed{A}, \boxed{B}\boxed{T} \in \{ \boxed{Y} \}^* .$$

Im Fall $i = 0$ schließt man folgendermaßen:

$$\boxed{S}\boxed{X}\boxed{T} = \boxed{S}\boxed{A}\boxed{B}\boxed{T}$$

ist gemäß Voraussetzung aus $\{ \boxed{Y} \}^*$.

Weiterhin gilt $\text{len}\boxed{X} \mid \text{len}\boxed{Y}$. Somit ist $\boxed{S}\boxed{T} \in \{ \boxed{Y} \}^*$.

◇

Lemma 5.3

Seien $\boxed{S}, \boxed{T}, \boxed{X}, \boxed{Y}$ Zellen mit $\text{len}\boxed{S} + \text{len}\boxed{T} = \text{len}\boxed{X} = \text{len}\boxed{Y}$.
Dann gilt

$$\boxed{S}\boxed{X^i}\boxed{T} \in \{ \boxed{Y} \}^* \quad \forall i \in \mathbb{N}_0$$

$$\Leftrightarrow$$

$$\boxed{X} = \boxed{T}\boxed{S} \text{ und } \boxed{Y} = \boxed{S}\boxed{T}$$

Beweis:

"⇒" Laut Voraussetzung ist $\boxed{S}\boxed{T} = \boxed{Y}$ und $\boxed{S}\boxed{X}\boxed{T} = \boxed{Y}\boxed{Y}$. Somit ist
$\boxed{S}\boxed{X}\boxed{T} = \boxed{S}\boxed{T}\boxed{S}\boxed{T}$. Hieraus folgt direkt $\boxed{X} = \boxed{T}\boxed{S}$.

"⇐" Für $i = 0$ folgt die Behauptung direkt. Ist $i \geq 1$, so gilt
$\boxed{S}\boxed{X^i}\boxed{T} = \boxed{Y^{i+1}}$ wegen $\boxed{X} = \boxed{T}\boxed{S}$.

◇

Zur Verifikation der beiden Regeln werden diese in einzelne Bearbeitungsphasen eingeteilt.

zu Regel 1

zunächst wird die Gültigkeit der folgenden drei Regeln geprüft

a) $\boxed{\omega 0}\boxed{L}\boxed{M'}\boxed{X} \vdash_M^* \boxed{\omega 0}\boxed{L}\boxed{S}\boxed{H}$ (L: q_1•, H: •q_r)

b) $\boxed{H}\boxed{X} \vdash_M^* \boxed{X'}\boxed{H}$ (X: •q_r, H: •q_r)

c) $\boxed{H}\boxed{R}\boxed{0^\omega} \vdash_M^* \boxed{T}\boxed{M'}\boxed{X^k}\boxed{R}\boxed{0^\omega}$ (H: •q_r, T: q_1•)

Erfüllt M diese, so ergibt sich insgesamt für alle $i \in \mathbb{N}$

$$\boxed{\omega 0}\boxed{L}\boxed{M'}\boxed{X^i}\boxed{R}\boxed{0^\omega} \vdash_M^*$$

$$\boxed{\omega 0}\boxed{L}\boxed{S}\boxed{X'^{(i-1)}}\boxed{T}\boxed{M'}\boxed{X^k}\boxed{R}\boxed{0^\omega} .$$

(q_1• unter L, q_1• unter T)

Es bleibt jetzt noch zu zeigen, daß $\boxed{S}\boxed{X'^{(i-1)}}\boxed{T}$ für alle $i \in \mathbb{N}$ ein Wort aus $\{ \boxed{Y} \}^*$ ist. Dabei hilft Lemma 5.2. Wenn eine Zerlegung von $\boxed{X'}$ mit den geforderten Eigenschaften existiert und M die Regeln a)–c) erfüllt, dann ist Regel 1 gültig.

zu Regel 2

hierzu werden folgende Regeln geprüft:

d) $\boxed{Y}\boxed{M'}\boxed{X} \vdash_M^* \boxed{M}\boxed{U}\boxed{I}$ (Y: q_1•, I: •q_2),

wobei $\text{len}\boxed{I} = \text{len}\boxed{X}$ und $\text{len}\boxed{M} = \text{len}\boxed{M'}$ gelten soll.

e) $\boxed{I}\boxed{X} \vdash_M^* \boxed{X''}\boxed{I}$ (X: •q_2, I: •q_2)

f) $\boxed{I}\boxed{R}\boxed{0^\omega} \vdash_M^* \boxed{V}\boxed{X}\boxed{R}\boxed{0^\omega}$ (I: •q_2, V: q_3•)

Falls in c) $k = 0$ war, ist noch die folgende Regel zu testen:

g) $\boxed{Y}\boxed{M'}\boxed{R}\boxed{0^\omega} \vdash_M^* \boxed{M'}\boxed{X}\boxed{R}\boxed{0^\omega}$ (Y: q_1•, M': q_1•).

Die Gültigkeit der Regeln d)–g) impliziert für alle $i \in \mathbb{N}$ die Gültigkeit von

$\boxed{Y}\boxed{M'}\boxed{X^i}\boxed{R}\boxed{0^\omega} \underset{M}{\vdash^*} \boxed{M}\boxed{U}\boxed{X''^{(i-1)}}\boxed{V}\boxed{X}\boxed{R}\boxed{0^\omega}$.
$q_1\bullet$ $\quad$ $q_3\bullet$

Nun wird angenommen, daß $\boxed{U}\boxed{X''^{(i-1)}}\boxed{V}$ für alle $i \in \mathbb{N}$ gleich $\boxed{Z^i}$ ist. Hierbei ist $\boxed{Z}$ eine neue Zelle mit len $\boxed{Z}$ = len $\boxed{X''}$ (= len $\boxed{U}$ + len $\boxed{V}$).

Gemäß Lemma 5.3 ist die Behauptung mit $\boxed{Z} = \boxed{U}\boxed{V}$ äquivalent zu $\boxed{X''} = \boxed{V}\boxed{U}$. Jetzt muß noch geprüft werden, ob $\boxed{Z^i}$ wie gewünscht verarbeitet wird:

h) $\boxed{Z}\boxed{X} \underset{M}{\vdash^*} \boxed{X}\boxed{X}$
$q_3\bullet$ $\quad$ $q_3\bullet$

Falls die Regel

i) $\boxed{M}\boxed{X} \underset{M}{\vdash^*} \boxed{M'}\boxed{X}$
$q_3\bullet$ $\quad$ $q_1\bullet$

noch verifiziert werden kann, so ist Regel 2 erfüllt.

Beispiel 5.4

Zur Veranschaulichung der vorgestellten Methode wird nun bewiesen, daß die Turingmaschine M, gegeben durch

	0	1
0	1R1	1L0
1	0L0	0R2
2	0R3	1RH
3	0R4	0R3
4	1L4	0L1

,

nicht stoppt.
M erreicht nach ein paar Schritten die Konfiguration

$\boxed{\omega 0}\boxed{0011110}\boxed{0^\omega}$.
$\bullet 1$

Durch Raten der Zellenlängen ergibt sich hieraus

$\boxed{L} = \boxed{M'} = \boxed{0}$, $\boxed{X} = \boxed{1111}$, $\boxed{R} = \boxed{0}$ und $q_1 = 1$.

Bei M handelt es sich um eine DRAGON-Turingmaschine, wenn die Regeln a)–i) gültig sind.

a)

$\boxed{\omega 0}\boxed{0}\boxed{0}\boxed{1111} \underset{M}{\vdash^*} \boxed{\omega 0}\boxed{0}\boxed{1111}\boxed{1000}$
$\bullet 1$ $\quad$ $\bullet 4$

Damit kommt $\boxed{L}$ am linken Rand wieder vor, und man erhält $\boxed{S} = \boxed{1111}$, $\boxed{H} = \boxed{1000}$ und $q_r = 4$.

b)

$\boxed{1000}\boxed{1111} \underset{M}{\vdash^*} \boxed{1111}\boxed{1000}$
$\bullet 4$ $\quad$ $\bullet 4$

Diese Regel ist gültig mit $\boxed{X'} = \boxed{1111}$.

c)

$\boxed{1000}\boxed{0}\boxed{0^\omega} \underset{M}{\vdash^*} \boxed{0}\boxed{1111}\boxed{0}\boxed{0^\omega}$
$\bullet 4$ $\quad$ $1\bullet$

$\boxed{T} = \boxed{\epsilon}$ und $k = 1$ macht die Regel gültig.

Für $i \in \mathbb{N}$ ist bis hierhin die folgende Berechnung von M nachgewiesen:

$\boxed{\omega 0}\boxed{0}\boxed{0}\boxed{[1111]^i}\boxed{0}\boxed{0^\omega} \underset{M}{\vdash^*}$
$\bullet 1$

$\boxed{\omega 0}\boxed{0}\boxed{1111}\boxed{[1111]^{i-1}}\boxed{0}\boxed{1111}\boxed{0}\boxed{0^\omega}$.
$1\bullet$

Als nächstes wird $\boxed{Y}$ ermittelt. Es ist notwendig, daß die Zellenlänge von $\boxed{Y}$ die von $\boxed{X}$ teilt. Mögliche Teiler sind 1, 2 und 4, von denen 2 geraten wird. Für alle $i \in \mathbb{N}$ soll $\boxed{1111}\boxed{[1111]^{i-1}}$ ein Wort aus $\{\ \boxed{Y}\ \}^*$ sein. Notwendig und offensichtlich auch hinreichend ist hierfür $\boxed{Y} = \boxed{11}$.
Das Verfahren zeigt die Hinlänglichkeit durch die Zerlegung $\boxed{A}\boxed{B} = \boxed{11}\boxed{11}$ von $\boxed{X'} = \boxed{1111}$. Gemäß Lemma 5.2 ist zu prüfen, ob $\boxed{B}\boxed{A}$, $\boxed{S}\boxed{A}$ und $\boxed{B}\boxed{T}$ jeweils Wörter aus $\{\ \boxed{Y}\ \}^*$ sind. Dieses ist der Fall.

d)

$\boxed{11}\boxed{0}\boxed{1111} \underset{M}{\vdash^*} \boxed{1}\boxed{00}\boxed{0000}$
$1\bullet$ $\quad$ $\bullet 3$

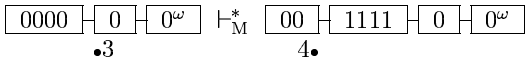
Man erhält hieraus $\boxed{M} = \boxed{1}$, $\boxed{U} = \boxed{00}$, $\boxed{I} = \boxed{0000}$ und $q_2 = 3$.

e)

$\boxed{0000}\boxed{1111} \underset{M}{\vdash^*} \boxed{0000}\boxed{0000}$
$\bullet 3$ $\quad$ $\bullet 3$

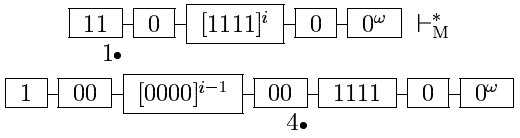
ist gültig mit $\boxed{X''} = \boxed{0000}$.

f)

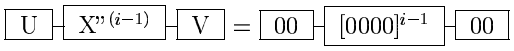


wird erfüllt mit $\boxed{\text{V}} = \boxed{00}$ und $q_3 = 4$.

Mit den Regeln e)–f) ist für alle $i \in \mathbb{N}$ die Gültigkeit der folgenden Regel bewiesen:



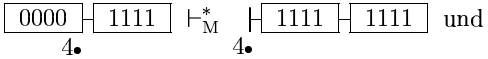
Im nächsten Schritt wird gezeigt, daß



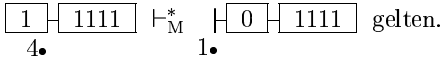
für alle $i \in \mathbb{N}$ ein Wort gebildet aus $\boxed{\text{Z}} = \boxed{\text{U}}\boxed{\text{V}} = \boxed{0000}$ ist. Hierzu ist die Bedingung $\boxed{\text{X"}} = \boxed{\text{V}}\boxed{\text{U}} = \boxed{0000}$ hinreichend, welche auch erfüllt ist.

g) muß nicht geprüft werden, da in c) $k = 1$ war.

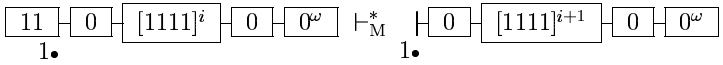
h)



i)



Insgesamt ist mit e)–i) schließlich die Regel



für alle $i \in \mathbb{N}$ gültig. M ist somit eine DRAGON-Turingmaschine und stoppt nicht.

Es muß an dieser Stelle betont werden, daß das vorgestellte Verfahren schon bei Turingmaschinen aus $\mathcal{TM}_5$ an einigen Stellen zu speziell ist. So kann beispielsweise die Zelle am linken Rand verändert werden, und die Verschiebung dieser Zelle nach rechts tritt auch auf. Weiterhin existieren in $\mathcal{TM}_5$ DRAGON-Turingmaschinen, bei denen len $\boxed{\text{Y}}$ len $\boxed{\text{X}}$ nicht teilt. Die Nichttermination kann in diesem Fall wegen Lemma 5.2 mit dem dargestellten Verfahren nicht gezeigt werden. Trotz dieser Einschränkungen konnte mit der Methode für ca. 2/3 aller DRAGON-Turingmaschinen aus $\mathcal{TM}_5$ die Nichttermination bewiesen werden.

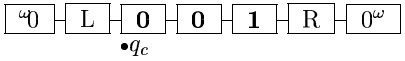
5.2 COUNTER-Turingmaschinen

Die Bandinhalte zu bestimmten Zeiten der Berechnung der hier untersuchten Turingmaschinen ähneln den Ziffernfolgen von Zählern zu einer gewissen Basis $m + 1$.
Zunächst seien die notwendigen Bezeichnungen eingeführt:

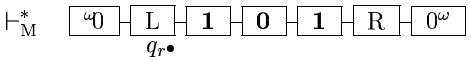
- $\boxed{\mathbf{0}}$,...,$\boxed{\mathbf{m}}$ bezeichnen die Ziffernzellen des Zählers und $\boxed{\text{ü}}$ die Übertragszelle; diese Zellen sollen alle die gleiche Länge l_Z haben.
- $\boxed{\text{L}}$ und $\boxed{\text{R}}$ sind Begrenzungszellen.
- q_c ist der carry-Zustand und q_r der return-Zustand.

Die prinzipielle Arbeitsweise einer COUNTER-Turingmaschine soll an einem Beispiel verdeutlicht werden:

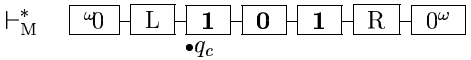
Es sei M eine Binärzähler-Turingmaschine ($m = 1$), und im Laufe der Berechnung werde die Konfiguration



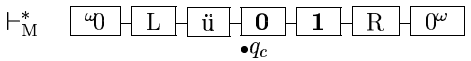
erreicht. Der Bandinhalt entspricht der Dualdarstellung von vier. Die folgenden Schritte bewirken die Erhöhung der linken Ziffer $\boxed{\mathbf{0}}$, da sich M im Übertragszustand q_c befindet. Dieses führt zur Ziffer $\boxed{\mathbf{1}}$ und dem return-Zustand q_r – ein Zeichen dafür, daß alle aufgetretenen Überträge berücksichtigt wurden.

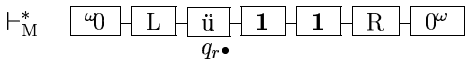


$\boxed{\text{L}}$ wird unverändert im Zustand q_c verlassen.

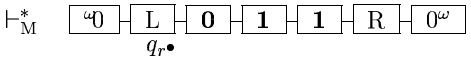


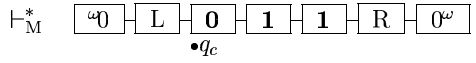
Nun soll die Ziffer $\boxed{\mathbf{1}}$ erhöht werden. Da mit M aber ein Binärzähler vorliegt, muß ein Übertrag auf die nächste Ziffernposition erfolgen. Aus der linken Ziffernzelle $\boxed{\mathbf{1}}$ wird temporär eine Übertragszelle $\boxed{\text{ü}}$.



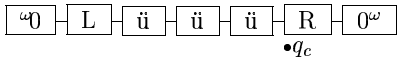


Diese wird beim Zurücklaufen in $\boxed{\mathbf{0}}$ verwandelt.



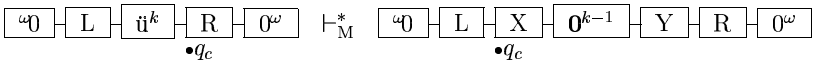


Nach Erreichen der Konfiguration



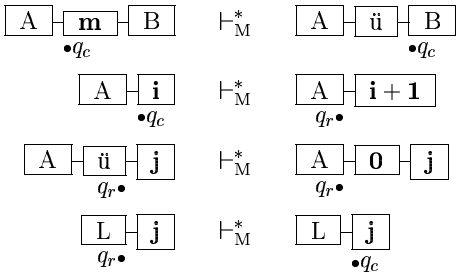
wird die Ziffernanzahl erhöht. Dabei kann es zu einer Linksverschiebung von [L] und/oder zu einer Rechtsverschiebung von [R] kommen. Wie dieses geschehen kann, zeigen die folgenden Regeln, auf deren Gültigkeit sich der Nichtterminationsbeweis stützt:

1. Es existieren Ziffernwörter [X] und [Y] mit Gesamtlänge ≥ 1, so daß für alle $k \geq 2$

   gilt.

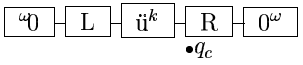
2. Für [A] $\in$ { [L], [ü] }, [B] $\in$ { [R], [**0**], ..., [**m**] }, $i \in \{0, \ldots, m-1\}$ und $j \in \{0, \ldots, m\}$:

Regel 1 beschreibt die Aktion bei Auftreten eines Übertrages bei der höchsten Ziffernposition und die Regeln in 2. den Zählvorgang. Die große Anzahl der zu prüfenden Regeln ergibt sich aus der Hinzunahme von Zellenkontexten, was dazu führt, daß mehr Turingmaschinen als Zähler erkannt werden.

Satz 5.5

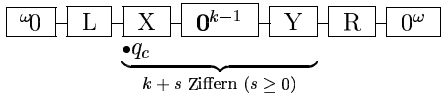
Erfüllt M $\in \mathcal{TM}_n$ die Regeln in 1. und 2. und kommt während der Berechnung die Konfiguration



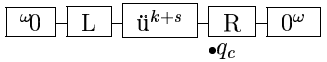
für ein $k \geq 2$ vor, so stoppt M nicht.

Beweis:

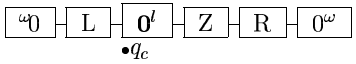
Ist eine obige Konfiguration für ein $k \geq 2$ erreicht worden, so gibt es gemäß der ersten Regel in der folgenden Berechnung die Konfiguration



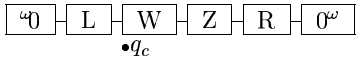
Es bleibt zu zeigen, daß danach die Konfiguration



erreicht wird. Da $k \geq 2$ beliebig war und eine Konfiguration mit $k \geq 2$ schon auftrat, ist die Behauptung des Satzes dann bewiesen. Folgende Aussage impliziert die Teilbehauptung:
Von



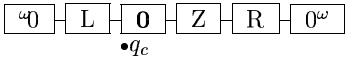
mit $l \geq 1$ und Ziffernwort [Z] aus beginnend werden alle Konfigurationen



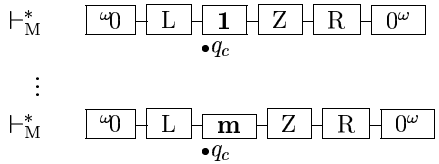
mit [W] $\in$ { [**0**], ..., [**m**] }^{l} erreicht.

Der Beweis erfolgt durch vollständige Induktion in l:

($l = 1$) Nach Auftreten der Konfiguration der Konfiguration

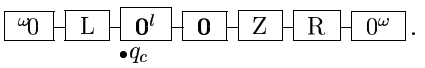


werden gemäß der Regeln in 2. nacheinander die Konfigurationen

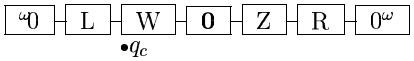


erreicht.

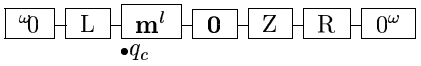
$(l \to l+1)$ Ausgegangen wird von der Konfiguration



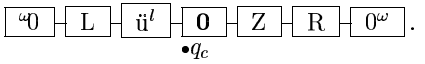
Laut Induktionsvoraussetzung werden alle Konfigurationen



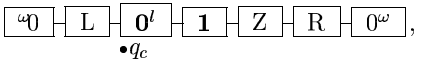
mit $\boxed{\text{W}} \in \{ \boxed{\mathbf{0}}, \ldots, \boxed{\mathbf{m}} \}^l$ erreicht – also auch



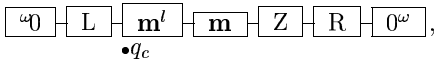
und damit



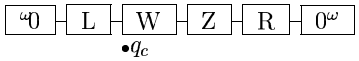
Hier erfolgt ein Übertrag auf die $(l+1)$-te Ziffer, und die Turingmaschine kommt zu



wo erneut die Induktionsvoraussetzung angewandt werden kann, bis schließlich gezeigt ist, daß



erreicht wird. Demnach treten alle Konfigurationen

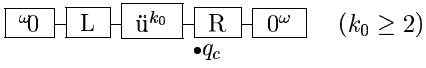


mit $\boxed{\text{W}} \in \{ \boxed{\mathbf{0}}, \ldots, \boxed{\mathbf{m}} \}^{l+1}$ auf.

$\diamond$

Wie zuvor bei den DRAGON-Turingmaschinen liegt die Schwierigkeit bei der Verifikation der Regeln in der Ermittlung der einzelnen Zellen. Im voraus ist zum Beispiel nicht bekannt, welche Länge die Ziffernzellen haben oder zu welcher Basis die (etwaige) COUNTER-Turingmaschine zählt. Falls es nur mit erheblicher Mühe möglich ist, gebrauchte Informationen aus einer Turingmaschinen-Simulation zu erhalten, werden mögliche Alternativen einfach ausprobiert, was natürlich zu einer größeren Laufzeit des Beweisalgorithmus' führt.

Zur Verifikation von Regel 1 wird von einer Konfiguration

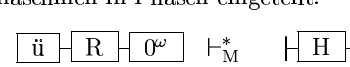


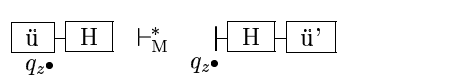
$(k_0 \geq 2)$

ausgegangen, die während einer Simulation erreicht worden sein soll. Eine Möglichkeit zur Ermittlung der Zellen ist das Raten ihrer Längen. Hiermit ist schon die Länge der Ziffernzellen ($l_Z = \text{len}\ \boxed{\text{ü}}$) festgelegt. Der Lauf zum linken Rand wird wie bei den DRAGON-Turingmaschinen in Phasen eingeteilt:

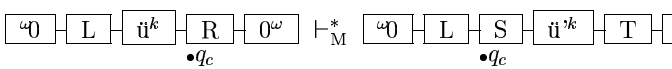
a) 

(len H $= l_Z$)

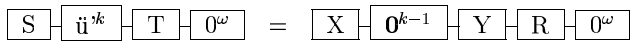
b) 

c) 

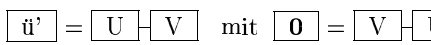
Mit der Gültigkeit dieser Regeln erhält man insgesamt für alle $k \geq 2$



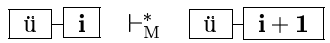
Nun soll die Existenz von Zifferwörtern $\boxed{\text{X}}$ und $\boxed{\text{Y}}$ gezeigt werden, so daß für alle $k \geq 2$



gilt, denn es muß sichergestellt sein, daß ein Ziffernwort zu Beginn des Zählvorganges vorliegt. Mit Lemma 5.3 muß geprüft werden, ob eine Zerlegung



existiert. $\boxed{\text{U}}$ kann durch die Forderung, daß $\boxed{\text{S}}\boxed{\text{U}}$ ein Ziffernwort ist, bestimmt werden. Hierdurch ist $\boxed{\text{V}}$ und damit $\boxed{\mathbf{0}}$ festgelegt. Mit Kenntnis von $\boxed{\mathbf{0}}$ lassen sich nun auch die restlichen Ziffernzellen und q_r bestimmen, indem die Regeln



für $i \in \{0, \ldots, m-1\}$ verifiziert werden.

Es bleibt noch zu zeigen, daß $\boxed{\text{S}}\boxed{\text{U}} =: \boxed{\text{X}}$ ein Ziffernwort ist und $\boxed{\text{V}}\boxed{\text{T}}\boxed{0^\omega} = \boxed{\text{Y}}\boxed{\text{R}}\boxed{0^\omega}$ für ein Ziffernwort $\boxed{\text{Y}}$ gilt. Da jetzt aber alle Ziffernzellen bekannt sind, fällt dieses und die Verifikation der übrigen Regeln in 2. nicht schwer.

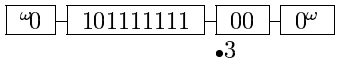
Beispiel 5.6

Gegeben sei die Turingmaschine M durch die Turingtafel

	0	1
0	1R1	1L1
1	0R2	0L1
2	1R3	1RH
3	1R4	1R3
4	0L0	1R2

.

Zu einem Zeitpunkt der Berechnung von M wird die Konfiguration



erreicht. Zur Ermittlung der Zellen L, ü und R werden ihre Längen geraten:

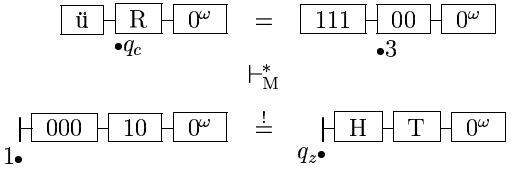
$$\begin{aligned} \text{len}\,\boxed{\text{L}} &= 3, \\ \text{len}\,\boxed{\text{ü}} &= 3 =: l_Z \text{ und} \\ \text{len}\,\boxed{\text{R}} &= 2 \end{aligned}$$

Hieraus ergibt sich aus obiger Konfiguration

$$\begin{aligned} \boxed{\text{L}} &= \boxed{101}, \\ \boxed{\text{ü}} &= \boxed{111}, \\ \boxed{\text{R}} &= \boxed{00} \text{ und} \\ q_c &= 3. \end{aligned}$$

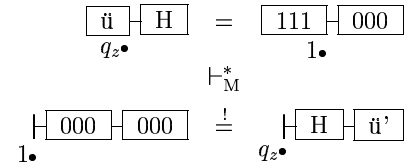
Im nächsten Schritt werden die Regeln a), b) und c) geprüft.

zu a) Die Zellen ü und R und der Übertragszustand q_c sind bekannt. Damit kann die geforderte Zellenveränderung verifiziert werden:



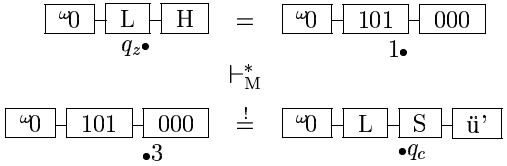
Somit ist $\boxed{\text{H}} = \boxed{000}$, $\boxed{\text{T}} = \boxed{10}$ und $q_z = 1$.

zu b)



$\boxed{\text{ü'}}$ ist also gleich $\boxed{000}$.

zu c)



Zelle S ist demnach leer.

Die Ziffernzelle $\boxed{\mathbf{0}}$ erhält man aus einer Zerlegung $\boxed{\text{ü'}} = \boxed{\text{U}}\boxed{\text{V}}$. Da $\boxed{\text{S}}\boxed{\text{U}}$ ein Ziffernwort sein soll und S leer ist, ergibt sich $\boxed{\text{U}} = \boxed{\text{ü'}}$ und $\boxed{\text{V}} = \boxed{\epsilon}$ und hieraus $\boxed{\mathbf{0}} = \boxed{000}$.
Die Ziffernzellen $\boxed{\mathbf{1}} = \boxed{100}$ und $\boxed{\mathbf{2}} = \boxed{010}$ sowie der Zustand $q_r = 1$ werden durch "Zählen" ermittelt. Weitere Ziffernzellen gibt es nicht, denn beim Inkrementieren der Ziffer $\boxed{\mathbf{2}}$ tritt ein Übertrag auf die nächste Ziffernposition auf.
Daß $\boxed{\text{S}}\boxed{\text{U}} =: \boxed{\text{X}}$ ein Ziffernwort ist, folgt aus der Tatsache, daß S leer und $\boxed{\text{U}} = \boxed{\mathbf{0}}$ ist. Es bleibt also nur noch die Existenz eines Zifferwortes $\boxed{\text{Y}}$ zu zeigen mit

$$\boxed{\text{V}}\boxed{\text{T}}\boxed{0^\omega} = \boxed{\text{Y}}\boxed{\text{R}}\boxed{0^\omega},$$

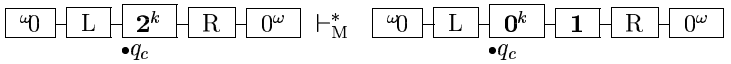
was

$$\boxed{10}\boxed{0^\omega} = \boxed{\text{Y}}\boxed{00}\boxed{0^\omega}$$

entspricht. Wählt man $\boxed{\text{Y}} = \boxed{100} = \boxed{\mathbf{1}}$, so ist die Gleichung erfüllt.
Damit hat man insgesamt für alle $k \geq 2$

$$\boxed{\text{S}}\boxed{\text{ü'}^k}\boxed{\text{T}}\boxed{0^\omega} = \boxed{\mathbf{0}}\boxed{\mathbf{0}^{k-1}}\boxed{\mathbf{1}}\boxed{\text{R}}\boxed{0^\omega}.$$

Da die restlichen Regeln aus 2. auch erfüllt sind, liegt mit M ein Ternärzähler vor, der bei einem Übertrag auf die höherwertigste Ziffernposition für alle $k \geq 2$ gemäß der nachstehenden Regel arbeitet und nicht stoppt.



6 Anwendung der Methoden

Die in den vorangehenden Kapiteln vorgestellten Verfahren zur effizienten Erzeugung von Turingmaschinen und zum Beweis der Nichttermination sollen nun auf die Menge $\mathcal{TM}_5$ angewandt werden. Zunächst werden spezielle Punkte der Implementierung, wie z.B. Maßnahmen zur Erniedrigung des Zeit- und Speicherplatz-Bedarfes behandelt. Danach werden die durchgeführten Reduktionen in der zeitlichen Reihenfolge erläutert und im Anschluß daran die Arbeitsweise der derzeit "besten" Turingmaschine aus $\mathcal{TM}_5$ genau analysiert.

6.1 Aspekte der Implementierung

Da die Arbeitsweise der Algorithmen schon beschrieben wurde, liegt der Schwerpunkt hier auf speziellen Aspekten. Dabei wird zunächst auf die Erzeugung der baumnormalen Turingmaschinen aus $\mathcal{TM}_5$ eingegangen und danach auf die Beweisalgorithmen und die schnelle Simulation von Turingmaschinen.

Bei der Erzeugung baumnormaler Turingmaschinen kommt es auf die schnelle Ausführung eines Turingschrittes an, denn Millionen von Turingmaschinen müssen eine gewisse Anzahl von Schritten simuliert werden, um eine Normalform zu erhalten, in der möglichst nur ein undefinierter Eintrag vorkommt. Nach diesen Schritten werden einfache (schnelle) Tests auf Nichttermination durchgeführt. Passierende Turingmaschinen werden schließlich zur weiteren Untersuchung auf ein Speichermedium abgelegt.

Bei der Simulation ist auch darauf zu achten, daß die Bereitstellung der von den schnellen Tests benötigten Eintragnummern-Folgen samt Extremstellen-Markierung nicht zuviel Zeit beansprucht. Die genannten Bedingungen führten zu einer Implementierung in Assemblersprache, die den effizienten Umgang mit Bits und Bytes zuläßt. Damit ist es möglich, bei der Simulation ein Bit-Band zu verwenden und den gerade bearbeiteten Bandausschnitt in einem Register zu halten. Wird der Auschnitt verlassen, so speichert man diesen ab und holt sich den benachbarten in das Register. Da es sich bei TM-ERZEUGUNG um ein rekursives Verfahren handelt, spart man mit dem Bit-Band auch wesentliche Zeit bei der Parameterübergabe. Als maximale Schrittanzahl ist 150 gewählt worden. Bei Tests wurde festgestellt, daß bei den meisten Turingmaschinen die einfachen Beweisverfahren schon früher greifen. Aus diesem Grund wird die Prüfung erstmals nach 45 Schritten durchgeführt, dann nach 65 und schließlich nach 150 Schritten. Die Turingtafel ist als ein Bytefeld organisiert. Je vier Bytes gehören zu einem Zustand, und ein Eintrag der Turingtafel wird durch zwei Bytes codiert. Ist nach 150 Schritten die Termination einer Turingmaschine nicht entschieden, so wird die dazugehörige Turingtafel in 6 Bytes gepackt (hinreichend sind wegen der Baumnormalform $9 \cdot (2+3) = 45$ Bits, denn der erste Eintrag ist immer 1R1, und ein undefinierter Eintrag kann durch Zielzustand > 4 markiert werden). Eine gewisse Anzahl der so gepackten Turingtafeln wird jeweils mit dem Huffman-Verfahren (siehe z.B. [Knu73]) codiert und auf Diskette abgelegt. Auf diese Art belegten ca. $1,6 \cdot 10^6$ Turingmaschinen rund 4 MByte Speicher, was im Durchschnitt $2,5$ Byte pro Turingmaschine entspricht.

Der Entwicklung der Beweisverfahren ging die Implementierung eines Programmes voraus, das die Berechnung von Turingmaschinen visuell darstellt. Hierbei können die Bandveränderungen Schritt für Schritt verfolgt und somit die Arbeitsweise großer Klassen von Turingmaschinen festgestellt werden, was zu den vorgestellten XMAS-, COUNTER- und DRAGON-Turingmaschinen führte. Die zentralen Teile des XMAS-Beweisalgorithmus' sind im Anhang aufgeführt. Um die Rechenzeit zu verringern, wurden in der ersten Version nacheinander Eintragnummern-Folgen der Länge 300, 1.000 und 3.000 untersucht, denn viele XMAS-Turingmaschinen werden schon frühzeitig erkannt. Für die Implementierung des COUNTER- und DRAGON-Beweisalgorithmus' wurde ein Zellen-Modul entwickelt, in dem nützliche Unterprogramme zur Zellen-Behandlung, wie z.B. den Lauf einer Turingmaschine bis zum Verlassen einer Zelle oder bis zum Erreichen eines gewissen Zustandes, untergebracht wurden. Als Datenstruktur wurde für die Zellen ein Verbundtyp gebildet aus einer Zeichenfolge und deren Länge gewählt.

Um "bessere" Turingmaschinen zu finden, ist ein schneller Simulations-Algorithmus entstanden, bei dem Bandausschnitte der Länge 16 zusammengefaßt und mit einer Cache-Technik früher schon einmal ausgeführte Bandveränderungen festgestellt und dann direkt vorgenommen werden. Damit wurde im Vergleich zu der normalen Implementierung die Geschwindigkeit bis zu Faktor fünf erhöht, was eine $50 \cdot 10^6$-Schritte-Simulation von 800 heuristisch ausgewählten DRAGON-Turingmaschinen in ca. 40 Stunden ermöglichte. Auf diese Weise wurden 19 Turingmaschinen gefunden, deren Werte die derzeit bekannten höchsten unteren Schranken für $\Sigma(5)$ und $S(5)$ überstiegen.

6.2 Reduktions-Chronologie

Die folgende chronologische Auflistung der durchgeführten Reduktionen gibt Aufschluß über die Wirksamkeit der beschriebenen Methoden. Alle angegebenen Rechenzeiten beziehen sich auf eine 7MHz-68000 CPU.

Gemäß Bemerkung 2.2 umfaßt $\mathcal{TM}_5$ ca. $6.3 \cdot 10^{13}$ Turingmaschinen. Der Algorithmus TM-ERZEUGUNG generierte mit SCHRITTMAX = 150 120.546.062 baumnormale Turingmaschinen aus $\mathcal{TM}_5$ in ca. 37 Stunden. Die Anzahl der zu untersuchenden Turingmaschinen schrumpft also im ersten Schritt schon um den Faktor 525.968.

120.546.062

Von diesen Turingmaschinen

- bewegten 62.262.596 (51,7%) den Kopf periodisch in eine Richtung.
- hielten 28.006.987 (23,2%) innerhalb von 150 Schritten.
- erreichten 17.594.824 (14.6%) keinen Halteintrag, da zu einem Zeitpunkt der Simulation kein solcher mehr in der aktuellen Zustands-Zusammenhangskomponente erreichbar war.
- wiederholte sich bei 9.449.929 (7,8%) eine Konfiguration.
- wurde bei 1.589.276 (1,3%) die Nichttermination mittels Rückwärtslauf bewiesen.

Es verblieben 1.642.450 (1,4%) hinsichtlich der Termination unentschiedene Turingmaschinen.

1.642.450

Die ersten Versionen der XMAS- und COUNTER-Beweisprogramme ermittelten in 182 Stunden unter diesen 1.481.510 XMAS- und 58.696 COUNTER-Turingmaschinen. 6.277 Turingmaschinen hielten innerhalb von 3.000 Schritten. Unter diesen waren 36 mit zwei undefinierten Einträgen. Der erreichte undefinierte Eintrag wurde vollständig expandiert, wodurch 720 Turingmaschinen erzeugte wurden. Nach nochmaliger Anwendung der XMAS- und COUNTER-Beweise blieben 329 Turingmaschinen davon übrig.

96.296

Durch Vergrößerung der Baumtiefe auf 20 und der Schrittanzahl auf 5.000 konnte mit den schon bei der Erzeugung der Turingmaschinen angewandten Methoden weitere 6.792 nicht stoppende Turingmaschinen festgestellt werden. 85 Turingmaschinen hielten.

89.419

Die endgültige Version des COUNTER-Beweises fand 32.154 nicht haltende Turingmaschinen. Diese teilten sich in 84% Binär-, 7% Ternär- und 9% Quartärzähler auf. Zähler zu höheren Basen wurden nicht festgestellt. Die maximale untersuchte Ziffernzellenlänge betrug vier.

57.265

Der Test auf äquivalente Zustände (der zugegebenermaßen schon viel früher angewandt werden sollte, aber noch nicht zur Verfügung stand) reduzierte die Anzahl der Turingmaschinen um 725.

56.540

In 54 Stunden fand der DRAGON-Beweisalgorithmus 14.187 nicht stoppende Turingmaschinen.

42.353

Die Variation des Startzustandes verminderte die Anzahl um 9.319. Auch dieser schnelle Test sollte direkt der Erzeugung folgen, er lag zu dem Zeitpunkt aber noch nicht vor. Nach einer heuristischen Auswahl von 800 Turingmaschinen im Anschluß an eine Analyse der derzeit besten Turingmaschinen wurden "neue" Turingmaschinen gefunden, die mehr als 4.000 Einsen hinterlassen.

33.034

Die endgültige Version des XMAS-Beweises, welche nun Eintragnummern-Folgen bis zur Länge 4.000 untersuchte, erkannte weitere 9.427 XMAS-Turingmaschinen.

23.607

Von den restlichen Maschinen sind etwa 16% DRAGON-Turingmaschinen, bei denen die Teiler-Bedingung nicht erfüllt ist oder das Verhalten geringfügig von den bewiesenen abweicht. Ca. 40% der Turingmaschinen sind "schrumpfende"-DRAGON-Turingmaschinen, bei denen im Gegensatz zu den DRAGON-Turingmaschinen der Bandausschnitt nach jeder Periode kürzer wird, bis er aufgebraucht ist. Danach beginnt der Prozeß von vorne mit einem insgesamt verlängerten Bandausschnitt. Unter den 23.607 Maschinen sind 57 aus $\mathcal{TM}_4$.

Dieses ist der aktuelle Stand.

6.3 Analyse der aktuell besten Turingmaschine aus $\mathcal{TM}_5$

Bei den "neuen" Turingmaschinen handelt es sich um die in Abschnitt 5.1 beschriebenen DRAGON-Turingmaschinen. Die Nichttermination kann bei ihnen durch das vorgestellte Verfahren nicht gezeigt werden, weil die Teilbarkeitsbedingung für die Zellenlängen nicht erfüllt ist. Es stellt sich die Frage, welcher Mechanismus dem Stoppen nach Millionen von Schritten zugrunde liegt. In der sich anschließenden genauen Untersuchung wird sich herausstellen, daß ein einfaches Teilbarkeitskriterium den Schlüssel zu vielen Schritten und hinterlassenen Einsen birgt.

Die zu untersuchende Turingmaschine M ist durch

	0	1
0	1R1	1L2
1	1R2	1R1
2	1R3	0L4
3	1L0	1L3
4	1RH	0L0

gegeben. Sie hinterläßt nach 47.176.870 Schritten 4.098 Einsen auf dem Band.

Im folgenden wird die Konfigurationsfolge

$$(\mathrm{KON}_i)_{i\geq 0} = (\overset{0\bullet}{\boxed{{}^{\omega}0}\boxed{1^{k_i-1}}\boxed{1}\boxed{0^{\omega}}})_{i\geq 0}$$

betrachtet, wobei hier die Namen der Zellen ihren Inhalt beschreiben sollen.
Die erste Konfiguration von obigem Typ –

$$\overset{0\bullet}{\boxed{{}^{\omega}0}\boxed{111}\boxed{1}\boxed{0^{\omega}}}$$

– erreicht M nach vier Schritten; es ist also $k_0 = 4$. Zur Untersuchung der einer solchen Konfiguration folgenden Berechnung erfolgt eine Fallunterscheidung in den k_i.

1.Fall $k_i \equiv 1\ (3)$, $k_i \geq 4$

Wie aus der Turingtafel direkt ersichtlich ist, gilt die Regel $\underset{0\bullet}{\boxed{111}} \vdash_{\mathrm{M}}^{*} \underset{0\bullet}{\boxed{001}}$ und damit $\underset{0\bullet}{\boxed{[111]^k}} \vdash_{\mathrm{M}}^{*} \underset{0\bullet}{\boxed{[001]^k}}$ für $k \in \mathbb{N}$. Hierbei werden die markierten Einträge in

	0	1
0	1R1	**1L2**
1	1R2	1R1
2	1R3	**0L4**
3	1L0	1L3
4	1RH	**0L0**

zyklisch bearbeitet.

Die Berechnung nach Erreichen einer Konfiguration KON_i mit $k_i \equiv 1\ (3)$ setzt M demnach wie folgt fort:

$$\boxed{\omega 0}\boxed{[111]^{(k_i-1)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \quad (0\bullet)$$

$$\boxed{\omega 0}\boxed{0}\boxed{[001]^{(k_i-1)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}} \quad (0\bullet)$$

$$\boxed{\omega 0}\boxed{1}\boxed{[001]^{(k_i-1)/3}}\boxed{1}\boxed{0^\omega} \quad (\bullet 1)$$

Nun wird die Gültigkeit der Regel

$$\boxed{\omega 0}\boxed{1^k}\boxed{001} \vdash_{\mathrm{M}}^{*} \boxed{\omega 0}\boxed{1^{k+5}} \quad \forall k \in \mathbb{N} \qquad (\bullet 1)\ (\bullet 1)$$

gezeigt.

Nach zwei Schritten erreicht M

$$\boxed{\omega 0}\boxed{1^k}\boxed{111}. \qquad (3\bullet)$$

Mit der leicht zu verifizierenden Regel

$$\boxed{1^j} \vdash_{\mathrm{M}}^{*} \boxed{1^j} \quad \forall j \in \mathbb{N} \qquad (3\bullet)\ (3\bullet)$$

wird also

$$\boxed{\omega 0}\boxed{0}\boxed{1^{k+3}} \qquad (3\bullet)$$

erreicht, nach drei weiteren Schritten

$$\boxed{\omega 0}\boxed{11}\boxed{1^{k+3}} \qquad (\bullet 1)$$

und mit

$$\boxed{1^j} \vdash_{\mathrm{M}}^{*} \boxed{1^j} \quad \forall j \in \mathbb{N} \qquad (\bullet 1)\ (\bullet 1)$$

schließlich $\boxed{\omega 0}\boxed{1^{k+5}}$ $(\bullet 1)$, was zu zeigen war.

Damit ergibt sich direkt die Regel

$$\boxed{\omega 0}\boxed{1^k}\boxed{[001]^l} \vdash_{\mathrm{M}}^{*} \boxed{\omega 0}\boxed{1^{k+5l}} \quad \forall k, l \in \mathbb{N}. \qquad (\bullet 1)\ (\bullet 1)$$

Insgesamt gilt jetzt also

$$\boxed{\omega 0}\boxed{1}\boxed{[001]^{(k_i-1)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \boxed{\omega 0}\boxed{1^{1+5(k_i-1)/3}}\boxed{1}\boxed{0^\omega}, \qquad (\bullet 1)\ (\bullet 1)$$

und nach vier Schritten wird

$$\mathrm{KON}_{i+1} = \underset{0\bullet}{\boxed{\omega 0}\boxed{1^{5(k_i-1)/3+4}}\boxed{1}\boxed{0^\omega}}$$

erreicht. Hieraus erhält man

$$k_{i+1} = \frac{5}{3}(k_i - 1) + 4 + 1 = \frac{5}{3}k_i + 4 - \frac{2}{3}.$$

2.Fall $k_i \equiv 2\ (3)$, $k_i \geq 4$

Ähnlich wie im ersten Fall erhält man

$$\boxed{\omega 0}\boxed{1}\boxed{[111]^{(k_i-2)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \quad (0\bullet)$$

$$\boxed{\omega 0}\boxed{1}\boxed{[001]^{(k_i-2)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \quad (0\bullet)$$

$$\boxed{\omega 0}\boxed{1111}\boxed{[001]^{(k_i-2)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \quad (\bullet 0)$$

$$\boxed{\omega 0}\boxed{1^{4+5(k_i-2)/3}}\boxed{1}\boxed{0^\omega} \vdash_{\mathrm{M}}^{*} \quad (\bullet 1)$$

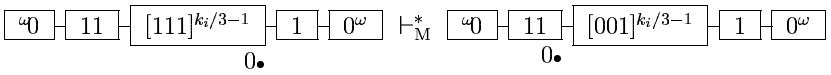
$$\boxed{\omega 0}\boxed{1^{5(k_i-2)/3+7}}\boxed{1}\boxed{0^\omega} \qquad (0\bullet)$$

und hieraus

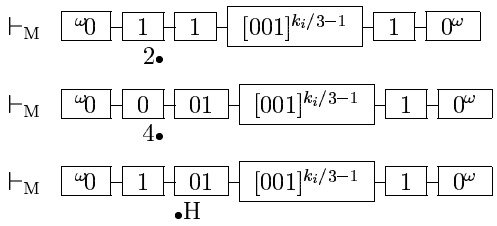
$$k_{i+1} = \frac{5}{3}(k_i - 2) + 7 + 1 = \frac{5}{3}k_i + 4 + \frac{2}{3}.$$

3.Fall $k_i \equiv 0\ (3)$, $k_i \geq 4$

Auch hier sind die ersten Schritte ähnlich:



Die drei folgenden Schritte führen aber zum Halten der Maschine



mit $2 + k_i/3 - 1 + 1 = k_i/3 + 2$ Einsen auf dem Band.

Das Ergebnis läßt sich in einem Iterationsschema zusammenfassen:

k_0	=	4	
k_{i+1}	=	$5/3 \cdot k_i + 4 + 2/3$,	falls $k_i \equiv 1\ (3),\ k_i \geq 4$
k_{i+1}	=	$5/3 \cdot k_i + 4 - 2/3$,	falls $k_i \equiv 2\ (3),\ k_i \geq 4$
Halt		mit $k_i/3 + 2$ Einsen auf dem Band,	falls $k_i \equiv 0\ (3),\ k_i \geq 4$

Nach 14 Schritten ist k_i erstmals durch drei teilbar, wie die folgende Tabelle zeigt.

i	k_i	k_i mod 3	i	k_i	k_i mod 3
0	4	1	8	568	1
1	10	1	9	950	2
2	20	2	10	1.588	1
3	38	2	11	2.650	1
4	68	2	12	4.420	1
5	118	1	13	7.370	2
6	200	2	14	12.288	0
7	338	2			

M stoppt also und hinterläßt 4098 Einsen auf dem Band.

7 Zusammenfassung und Ausblick

Das angestrebte Ziel der vorliegenden Arbeit war die Bestimmung der Werte $\Sigma(5)$ und $S(5)$. Die Anzahl der hierbei zu untersuchenden Turingmaschinen konnte durch die Beschränkung auf eine Normalform drastisch verringert werden. Der Test auf einfache hinreichende Bedingungen zur Nichttermination, die Beachtung von äquivalenten Zuständen und die Variation des Startzustandes verringerte die Anzahl der hinsichtlich der Termination unentschiedenen Turingmaschinen weiter. Die sich anschließenden Untersuchungen gingen auf die drei größten, in der Restmenge vorhandenen Klassen von Turingmaschinen ein. Für jede dieser wurde ein spezielles Beweisverfahren entwickelt, aus dessen erfolgreicher Termination auf die Nichttermination der Berechnung einer vorgelegten Turingmaschine geschlossen werden konnte. Für die XMAS-Turingmaschinen ist mit der Einführung der Laufformel und deren Untersuchung eine neue Möglichkeit vorgestellt worden, die Nichttermination mittels einer Betrachtung der Zustandsfolge zu beweisen. Bei den DRAGON- und COUNTER-Turingmaschinen führte die genaue Untersuchung der Arbeitsweise zu Beweisverfahren unter der Benutzung der von der Turingmaschine vorgenommenen Zellenveränderungen. Die Anwendung all dieser Methoden auf $\mathcal{TM}_5$ führte schließlich zu 23.607 unentschiedenen Turingmaschinen. Damit wurde das gesteckte Ziel nicht erreicht. Als Trost kann das durch die Reduktion ermöglichte Auffinden von 19 Turingmaschinen gelten, deren Werte die alten bekannten unteren Schranken für $\Sigma(5)$ und $S(5)$ beträchtlich übersteigen.

Wie geht es weiter? Ca. 75% der übriggebliebenen Turingmaschinen arbeiten auf eine Weise, die sich durch Zellenveränderungen leicht beschreiben läßt. Für diese können Beweisalgorithmen nach dem Muster der DRAGON- und COUNTER-Verfahren entwickelt werden. Hilfreich wäre hierbei eine komfortable Sprache zur Zellenmanipulation, in der auch die zu verifizierenden Regeln eingegeben werden können und deren Prüfung automatisiert geschieht. Unter Umständen gelingt es auch, den Laufformel-Ansatz auf andere Klassen von Turingmaschinen anzuwenden. Dabei wäre es vorteilhaft, wenn schon aus der Bearbeitung der ersten Hauptschleifen-Durchläufe die Nichttermination folgen würde. Ein anderer Ansatzpunkt liegt in einer Verallgemeinerung des Beweises durch Rückwärtslauf aus Abschnitt 3.4. Dort wurde die Nichttermination durch einen Widerspruch des Bandinhaltes zum auszuführenden Eintrag auf allen Berechnungspfaden gezeigt. Hinreichend ist aber schon die Eigenschaft, daß auf keinem Pfad die Startkonfiguration vorkommt.

In [MarBun90] wird die Ansicht vertreten, daß $\Sigma(6)$ mit einem Netzwerk von 2.000 Workstations "angreifbar" sei. Es wird eine im Vergleich zu $\mathcal{TM}_5$ ca. 500-mal höhere Anzahl zu prüfender Turingmaschinen vorausgesagt, was bei den in [MarBun90] übriggebliebenen ca. 260.000 unentschiedenen Turingmaschinen aus $\mathcal{TM}_5$ auf eine Anzahl von mindestens $130 \cdot 10^6$ Maschinen aus $\mathcal{TM}_6$ hinausläuft. Eine Workstation müßte demnach mindestens 65.000 Turingmaschinen bearbeiten. Das zu parallelisierende Verfahren wird nicht genannt, und dieses ist meiner Meinung nach der Kernpunkt der Kritik. Erst wenn wirksame Methoden bereitgestellt sind, sollte man über deren verteilte Anwendungen nachdenken. Das Interesse sollte nicht wochenlangen, stupiden Simulation gelten, sondern neuen Beweistechniken, um zunächst die Anzahl unentschiedener Maschinen weiter zu verringern und damit die Werte $\Sigma(5)$ und $S(5)$ eines Tages zu bestimmen und nicht nur untere Schranken für diese.

8 Anhang

8.1 Turingtafeln

Die folgende Tabelle zeigt die Turingtafeln der Turingmaschinen aus $\mathcal{TM}_5$ mit den derzeit größten bekannten Werten für Σ und S in aufsteigender Reihenfolge. Alle bis auf die zweite Turingmaschine sind in Baumnormalform angegeben. Diese (unveröffentlichte) Maschine läuft durch die Zustandspermutation einen Schritt länger und war deshalb lange Zeit die beste Turingmaschine aus $\mathcal{TM}_5$ hinsichtlich S. Turingmaschinen, die aus den angegebenen durch Veränderung des Halt-Eintrages entstehen und dadurch u.U. eine andere Anzahl von Einsen hinterlassen, wurden nicht mit aufgeführt. Die erste Maschine ist zuerst von U. Schult [LudSchWan83] entdeckt worden und die dritte Maschine von G. Uhing [Dew85].

	0	1
0	1R1	0L2
1	1R2	1R3
2	1L0	0R1
3	0R4	1RH
4	1L2	1R0

501
134.467

	0	1
0	0R1	0L2
1	1R2	1R3
2	1L0	0L4
3	1R4	1RH
4	1L0	1R0

1.471
2.358.065

	0	1
0	1R1	1L2
1	0L0	0L3
2	1L0	1RH
3	1L1	1R4
4	0R3	0R1

1.915
2.133.492

	0	1
0	1R1	1R0
1	1L2	0R3
2	1L0	1L2
3	1RH	1R3
4	1L2	1R1

4.096
11.804.896

	0	1
0	1R1	1R0
1	1L2	0R3
2	1L0	1L2
3	1RH	1R4
4	0L4	1R1

4.096
11.804.910

	0	1
0	1R1	1R0
1	0L2	0R2
2	1RH	1R3
3	1L4	1R1
4	1L0	1L4

4.096
11.811.010

	0	1
0	1R1	1R0
1	1L2	0R3
2	1L0	1L2
3	1RH	1R4
4	1L2	0L0

4.096
11.815.076

	0	1
0	1R1	1R0
1	0L2	0R2
2	1RH	1R3
3	1L4	0L0
4	1L0	1L4

4.096
11.821.190

	0	1
0	1R1	1R0
1	1L2	1R3
2	1L0	1L2
3	1RH	1R4
4	1R0	0R1

4.097
11.792.682

	0	1
0	1R1	1R0
1	1L2	1R3
2	1L0	1L2
3	1RH	1R4
4	1L0	0R1

4.097
11.792.696

	0	1
0	1R1	1R0
1	1L2	1R3
2	1L0	1L2
3	1RH	1R4
4	0L4	0R1

4.097
11.792.724

	0	1
0	1R1	1R0
1	1L2	1R3
2	1L0	1L2
3	1RH	0R4
4	1L2	1R1

4.097
11.798.796

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	1L3
3	1R0	0L4
4	1RH	1L2

4.097
11.804.926

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	1L3
3	0R4	0L4
4	1RH	1L2

4.097
11.804.940

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	0L3
3	1R2	1L4
4	1RH	1L2

4.097
11.811.026

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	0L3
3	0R1	1L4
4	1RH	1L2

4.097
11.811.040

	0	1
0	1R1	1R0
1	1L2	1L1
2	0R2	1L3
3	1R0	0L4
4	1RH	1L2

4.097
11.811.040

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	0L3
3	1R2	1L4
4	1RH	0R1

4.097
11.821.220

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	0L3
3	0R1	1L4
4	1RH	0R1

4.097
11.821.234

	0	1
0	1R1	0L3
1	1L2	1R3
2	1L0	1L2
3	1RH	1R4
4	1R0	0R1

4.097
23.554.764

	0	1
0	1R1	1R0
1	1L2	1L1
2	1R0	1L3
3	1R0	1L4
4	1RH	0L2

4.098
11.798.826

	0	1
0	1R1	1L2
1	1R2	1R1
2	1R3	0L4
3	1L0	1L3
4	1RH	0L0

4.098
47.176.870

8.2 Quantitative Ergebnisse für Σ und S

Die bis heute besten bekannten quantitativen Ergebnisse für $\Sigma(k)$ und $S(k)$ bis $k = 6$ sind in der nachstehenden Tabelle zusammengefaßt.

k	$\Sigma(k)$	$S(k)$	Quelle
1	1	1	trivial
2	4	6	leicht
3	6	21	[LinRad65]
4	13	107	[WeiCasFen73], [Bra75]
5	$\geq$ 4.098	$\geq$ 47.176.870	Diese Arbeit und unabhängig [MarBun90]
6	$\geq$ 136.612	$\geq$ 13.122.572.798	[MarBun90]

8.3 Algorithmus zur Erzeugung baumnormaler Turingmaschinen

```
/* Algorithmus TM_ERZEUGUNG zur Erzeugung baumnormaler Turingmaschinen */

#define QANZ            5       /* Anzahl der Zustaende >= 3 */
#define SCHRITTMAX      150     /* maximale Schrittzahl      */
#define FALSE           0
#define TRUE            1
#define LINKS           0
#define RECHTS          1

typedef struct { char def, bit, ri, q; } EINTRAG;

/* Organisation der Turingtafel:                                    */
/*                                                                  */
/*   Jeder Eintrag der Turingtafel besteht aus einer Struktur mit   */
/*   den folgenden Komponenten:                                     */
/*                                                                  */
/*       .def :  != 0  <=>  Eintrag ist definiert                   */
/*       .bit :  zu schreibendes Zeichen (0 oder 1)                 */
/*       .ri  :  Kopfbewegung 0 - links, 1 - rechts                 */
/*       .q   :  Zielzustand aus {0..QANZ-1}                        */

void
Erzeuge(BandAlt, ttafelAlt, q, qnext, pos, Schritt, defanz)
char    *BandAlt;           /* aktuelles Band       */
EINTRAG ttafelAlt[QANZ][2]; /* aktuelle Turingtafel */
short
   q,                       /* aktueller Zustand                        */
   qnext,                   /* naechster noch nicht besuchter Zustand   */
   pos,                     /* aktuelle Bandposition                    */
   Schritt,                 /* Anzahl der noch auszufuehrenden Schritte */
   defanz;                  /* Anzahl der definierten Eintraege         */
{
  char Band[2*SCHRITTMAX];
  short i, bit;
  EINTRAG ttafel[QANZ][2];

/* Arbeitskopien von BandAlt und ttafelAlt anlegen */

  for (i=0; i < 2*SCHRITTMAX; i++) Band[i] = BandAlt[i];
```



```
  for (i=0; i < QANZ; i++) {
    ttafel[i][0] = ttafelAlt[i][0]; ttafel[i][1] = ttafelAlt[i][1];
  }

  while (Schritt > 0) {

    bit = Band[pos];                    /* Zeichen unter dem Kopf      */

    if (!ttafel[q][bit].def) {          /* ist aktueller Eintrag def.? */

      if (defanz == QANZ*2-1) return;   /* nur noch ein Eintrag frei?  */

/* neuen Eintrag mit 0L0 initialisieren */

      ttafel[q][bit].def = TRUE;  ttafel[q][bit].bit = 0;
      ttafel[q][bit].ri  = LINKS; ttafel[q][bit].q   = 0;

/* alle Moeglichkeiten erzeugen */

      do {
        if (ttafel[q][bit].q == qnext && qnext < QANZ-1)

/* der naechste noch nicht besuchte Zustand ist erreicht worden und */
/* dieser ist nicht der hoechste Zustand => qnext wird erhoeht      */

          Erzeuge(Band, ttafel, q, qnext+1, pos, Schritt, defanz+1);

        else

/* sonst bleibt qnext erhalten */

          Erzeuge(Band, ttafel, q, qnext, pos, Schritt, defanz+1);

/* Eintrag erhoehen: zuerst .bit, dann .ri und zuletzt .q */

        if   (!(ttafel[q][bit].bit ^= 1))
          if (!(ttafel[q][bit].ri  ^= 1))
                ttafel[q][bit].q   += 1;

/* wenn maximaler Zustand noch nicht ueberschritten ist, weiter */

      } while (ttafel[q][bit].q <= qnext);
```

```
/* alle Eintragmoeglichkeiten sind betrachtet worden => zurueck */

      return;

    } else {

/* einen Schritt gemaess der Turingtafel ausfuehren */

      Band[pos] = ttafel[q][bit].bit;
      if (ttafel[q][bit].ri == RECHTS) pos++; else pos--;
      q = ttafel[q][bit].q;
      Schritt--;
    }
  }

/* Turingmaschine nach SCHRITTMAX gemachten Schritten merken */

  saveTM(ttafel);
}

main()
{
  char  Band[2*SCHRITTMAX];             /* Turingband  */
  short i;
  EINTRAG ttafel[QANZ][2];              /* Turingtafel */

/* alle Eintraege sind zunaechst undefiniert */

  for (i=0; i < QANZ; i++) ttafel[i][0].def = ttafel[i][1].def = FALSE;

/* im Startzustand beim Lesen einer 0 Aktion 1R1 */

  ttafel[0][0].def = TRUE;   ttafel[0][0].bit = 1;
  ttafel[0][0].ri  = RECHTS; ttafel[0][0].q   = 1;

/* Band loeschen */

  for (i=0; i < 2*SCHRITTMAX; i++) Band[i] = 0;

/* baumnormale Turingmaschinen erzeugen */

  Erzeuge(Band, ttafel, 0, 2, SCHRITTMAX, SCHRITTMAX, 1);
}
```


8.4 Algorithmus zur Vermutung der Laufformel

```
/* Algorithmus FORMEL_ERSTELLEN                    */
/* zur Teilfolgenerkennung im XMAS-Beweis / 2.5.90 */

/* Markierungsflags in F[] */

#define MINI     0x10               /* Minimum-Extremstelle          */
#define MAXI     0x20               /* Maximum-Extremstelle          */
#define KLAUF    0x40               /* Klammer auf                   */
#define KLZU     0x80               /* Klammer zu                    */
#define KLWEG    (~(KLAUF | KLZU))  /* zum Loeschen der Klammerflags */

#define FOREVER  for(;;)
#define FOLEN    500                /* maximale Laenge der Laufformel*/

/* globale Daten: */

char
  FO[FOLEN],          /* Vermutete Gesamt-Laufformel */
  FL[FOLEN];          /* Extremstellenmarkierung     */

extern short F[];     /* Anfangsstueck der Laufformel (Zahlen aus {0..9}) */
                      /* Extremstellen sind mit MINI oder MAXI markiert   */

/* Input:      Startindizes dreier zusammenhaengender Teilfolgen von F[], */
/*             deren Laenge linear zunimmt und in denen Zahlen aus {0..9} */
/*             mit etwaigen Klammermarkierungen (aus vorherigen Versuchen)*/
/*             und Extremstellenmarkierungen vorkommen koennen.           */
/*                                                                        */
/* Output:     return(!= 0) <=> Teilfolgen haben gewuenschte Eigenschaft; */
/*             die vermutete Gesamtdarstellung der Laufformel steht nach  */
/*             return(!= 0) in FO[] (Zeichenkette aus {'0'..'9','{','}'});*/
/*             die dazugehoerigen Extremstellenmarkierungen stehen in FL[]*/
/*             FL[i] != 0  <=>  bei FO[i] Extremstelle; die Zeichenketten */
/*             werden durch 0x00 beendet.                                 */

short
Formel_erstellen(pos1, pos2, pos3)
short pos1, pos2, pos3;
{
  short i, i1, i2, i3, a, pos4, ok;
```

```
  i1 = pos1; i2 = pos2; i3 = pos3;          /* Startindizes auf 3 Folgen */

/* Ende der 3. Folge bestimmen (Laengeneigenschaft benutzen) */

  pos4 = pos3 + (pos3 - pos2) + (pos3 - pos2) - (pos2 - pos1);

/* 1. Schritt:  2. Folge mit Klammerungen markieren */

  FOREVER {

/* Situation:                                  */
/*                                            */
/*     1.Folge    ...XXXX|UUUUUUUU...         */
/*                 i1^=== |ungleich           */
/*     2.Folge    ...XXXX|SSSS|VVVVV...       */
/*                 i2^=== ==== |ungleich      */
/*     3.Folge    ...XXXX|SSSS|SSSS|WWWW...   */
/*                 i3^===                     */

    i = i2;
    while (i1 < pos2 && i < pos3) {     /* 1.& 2. Folge bis zum ersten   */
      F[i1] &= KLWEG; F[i] &= KLWEG;    /* Unterschied vergleichen;      */
                                        /* von vorherigen Versuchen      */
      if (F[i1] != F[i]) break;         /* koennen noch Klammermarkie-   */
      i1++; i++;                        /* rungen in F[] vorhanden sein! */
    }

    if (i == i2) { ok = 0; break; }     /* kein Zeichen gleich => Fehler */

    while (i2 < pos3 && i3 < pos4) {    /* 2.& 3. Folge bis zum ersten  */
      F[i2] &= KLWEG; F[i3] &= KLWEG;   /* Unterschied vergleichen;     */
      if (F[i2] != F[i3]) break;        /* wiederum etwaige Klammermar- */
      i2++; i3++;                       /* kierungen vorher loeschen    */
    }

/* Situation:                                  */
/*                                            */
/*     1.Folge    ...XXXX|UUUUUUUU...         */
/*                   ==== ^i1                 */
/*     2.Folge    ...XXXX|SSSS|VVVVV...       */
/*                   ==== ^i   ^i2            */
/*     3.Folge    ...XXXX|SSSS|SSSS|WWWW...   */
/*                           ^i3              */
```



```
/* Am Ende aller drei Folgen => fertig und ok (alpha am Ende) */

    if (i1 == pos2 && i == pos3 && i2 == pos3 && i3 == pos4) {
      ok = 1; break;
    }

/* SSSS kommt in der 3. Folge nicht einmal vor => fertig und nicht ok */

    if (i2 <= i) { ok = 0; break; }

    F[i] |= KLAUF; F[i2-1] |= KLZU;  /* Klammerpaar in 2. Folge markieren */

/* Test auf doppeltes Vorkommen von SSSS in 3. Folge */

    for (i=i; i < i2; i++)
      if ((F[i] & KLWEG) != (F[i3++] & KLWEG)) break;

    if (i < i2) { ok = 0; break; }    /* Schleife kommt nicht zweimal vor */

    if (i3 > pos4) { ok = 0; break; } /* 3. Folge beim Test verlassen     */

/* bei allen drei Folgen am Ende => fertig und ok (beta am Ende) */

    if (i1 == pos2 && i2 == pos3 && i3 == pos4) { ok = 1; break; }

/* bei mindestens einer Folge am Ende => Fehler */

    if (i1 >= pos2 || i2 >= pos3 || i3 >= pos4) { ok = 0; break; }
  }

/* 2. Schritt:                                                    */
/*   Aus 2. Folge Laufformel und Extremstellenmarkierung erstellen */

  i = 0;

  if (ok)

/* 2. Formel von vorne abarbeiten; bei Klammermarkierung '{' bzw. '}'   */
/* in FL[] einfuegen; sonst den zur Zahl gehoerigen ASCII-Wert anfuegen */
/* und bei Extremstellen entsprechendes FL[i] != 0 setzen               */

    i2 = pos2;
```

```
    while (i2 < pos3) {

      a = F[i2++];                  /* naechstes Zeichen holen */

      if (i >= FOLEN-4) { printf("*** Formel zu lang\n"); return(0); }

      if (a & KLAUF) { FO[i] = '{'; FL[i++] = 0; }
      FO[i]  = (a & 15) + '0';
      FL[i++] = a & (MINI | MAXI);
      if (a & KLZU)  { FO[i] = '}'; FL[i++] = 0; }
    }

  FO[i] = FL[i] = 0;                /* Zeichenketten beenden  */
  return(ok);                       /* und fertig             */
}
```


8.5 Algorithmus zur Verifikation der Laufformel-Vermutung

```
/*********************************************************/
/* Algorithmus VERIFIKATION                              */
/* zum induktiven Beweis der Laufformel-Vermutung / 3.5.90 */
/*********************************************************/

/* nuetzliche Abkuerzungen fuer Positions-Check, Fehlermeldung */
/* und Unendlichschleife                                       */

#define CP if (pos < 4 || pos >= BLEN-4) { printf("*** pos?\n"); exit(1); }
#define ERR(x) return(0);
#define FOREVER for(;;)

/* globale Daten */

extern char
  ttafel[];            /* globale Turingtafel in gepacktem Format */
  CBand_Leer[BLEN],    /* ueberall undefiniertes Band             */
  CBand[BLEN],         /* Arbeitsband                             */
  LBE[SCHLEIFENLEN],   /* allgemeine Eingabe einer Klammerung     */
  LBL[SCHLEIFENLEN];   /* allgemeine Ausgabe einer Klammerung     */


/*********************************************************/
/* Funktion VERIFIKATION                                 */
/*                                                       */
/* Input:  Zeiger auf die vermutete Hauptschleifen-Formel    */
/*         (ohne Exponenten) und die dazugehoerige           */
/*         Extremstellen-Markierung                          */
/*                                                       */
/* Output: return(1) => Vermutung ist korrekt                */
/*********************************************************/

short
VERIFIKATION(FO, FL)
char *FO, *FL;
{
  short i;

  strcpy(CBand, CBand_Leer);            /* 1. Lauf auf ueberall       */
                                        /* undefiniertem Band         */
```

```
  i = LAUF(FO, FL, CBand, BLEN/2, 0);   /* letzte Position zurueck    */

  if (i && LAUF(FO, FL, CBand, i, 1))   /* 2. Lauf auf Ausgabe des 1. */

    return(1);                          /* Vermutung ist korrekt!     */

  else

    return(0);                          /* Fehler aufgetreten         */
}


/*************************************************************************/
/* Funktion LAUF                                                         */
/*                                                                       */
/* Input:  Zeiger auf Laufformel, Extremstellenmarkierung und symbo-     */
/*         lisches Band; Startposition auf dem Band; pass 0 bzw. 1       */
/*                                                                       */
/* Output: return(Endposition); falls diese gleich 0 ist, ist Fehler     */
/*         aufgetreten; falls nicht, so ist die Laufformel korrekt       */
/*         abgearbeit worden; in pass 0 wird das ueberall undefinierte   */
/*         Band erwartet; in pass 1 werden alle Klammerungen einmal      */
/*         zusaetzlich bearbeitet und bei Erreichen einer vorher noch    */
/*         nicht besuchten Position die Extremstellenmarkierung          */
/*         verifiziert                                                   */
/*************************************************************************/

short
LAUF(FO, FL, CBand, pos, pass)
char *FO, *FL;
register char *CBand;
short pos, pass;
{
  register short i, j, k, a, posFO;
  short p, Ri, KlAuf, KlZu, Klammer_extra, KlAuf_pos;

/* in pass 1 Klammerungen einmal mehr bearbeiten */
  Klammer_extra = pass;

  posFO = 0;

  while (a=FO[posFO]) { /* gesamte Formel abarbeiten */
```



```
    if (a == ')') {    /* explizite Bearbeitung der Klammerung zuende */

      posFO = KlAuf_pos;       /* Klammeranfang holen                 */
      Klammer_extra = 0;       /* und Klammer normal bearbeiten       */
      continue;
    }

    if (a == '(') {

      if (Klammer_extra) {     /* zunaechst Klammer einmal explizit   */

        KlAuf_pos = posFO;     /* Klammeranfang merken                */
        posFO++;               /* posFO auf ersten Klammereintrag     */
        continue;
      }

      if (pass) Klammer_extra = 1;

/*************************************************/
/* 1. Ein- und Ausgabe der Klammerung bestimmen */
/*************************************************/

      p = SCHLEIFEN_ANALYSE(FO, posFO, LBL, LBE, CBand_Leer);

/* p gibt Bandposition nach Bearbeitung der Klammerung an */

      while (FO[++posFO] != ')');        /* posFO zeigt auf ')' */

/*********************************************************************/
/* 2. Test, ob Eingabe vorhanden; wenn ja, durch Ausgabe ersetzen */
/*********************************************************************/

/* Endposition p gibt Richtung an */

      if (p > SCHLEIFENLEN/2) {
        Ri = 1; KlAuf = '('; KlZu = ')';
      } else {
        Ri = -1; KlAuf = ')'; KlZu = '(';
      }

      if (CBand[pos] == '0' || CBand[pos] == '1' ||
          CBand[pos] == KlZu) {
```

```
/*********************************************/
/* '0','1' oder KlZu unter Kopf =>           */
/*     naechste Klammerung unter Kopf schieben */
/*********************************************/

        i = pos;

        while ((a=CBand[i]) != '$') {
          if (a == KlZu) break;
          i += Ri;
        }

        if (a == '$') ERR("KLAMMERPAAR NICHT GEFUNDEN")

        while (CBand[pos] != KlAuf) {
          if (VERSCHIEBEN(CBand, i)) break;
          i -= Ri;
        }

        if (CBand[pos] != KlAuf) ERR("SCHIEBE-FEHLER")
      }

      if (CBand[pos] == KlAuf) {

/*********************************************************/
/* KlAuf unter Kopf =>                                   */
/*   Band mit erwarteter Eingabe der Klammerung vergleichen */
/*   und danach die Ausgabe auf das Band schreiben       */
/*********************************************************/

/* Beispiel:                                                    */
/*                                                              */
/*    Eingabe: LBE =  (1)10       Ausgabe: LBL =  01(1)    (Ri=-1) */
/*                      ^S.LEN/2                p^^S.LEN/2      */

        i = SCHLEIFENLEN/2;

        while (LBE[i] != KlZu) i += Ri;  /* KlZu in Eingabe suchen */

        j = i - SCHLEIFENLEN/2 + pos;
        k = p - Ri;
```



```
/* i ist Index auf KlZu in Eingabe, k Index auf entsprechende Stelle */
/* in Ausgabe, j Index auf Bandposition                              */

        while (LBE[i] != '$') {
          if (LBE[i] != CBand[j]) break; /* Bandinhalt stimmt nicht      */
          CBand[j] = LBL[k];             /* Ausgabe auf Band kopieren    */
          i -= Ri; j -= Ri; k -= Ri;     /* naechstes Zeichen betrachten */
        }

        if (LBE[i] != '$')               /* Eingabe steht nicht auf Band */
          ERR("KLAMMER PASST NICHT")

/* neue Position ist hinter der Klammerung */

        pos += p - (SCHLEIFENLEN/2 - Ri - Ri);
CP

      } else {

/*********************************************************/
/*  '$' unter Kopf =>                                    */
/*     nur Praefix der Eingabe vergleichen, die Positionen */
/*     fuer Klammerung muessen undefiniert sein          */
/*********************************************************/

        if (pass) ERR("$ UNTER KOPF BEI (...) IN PASS 1")

        i = SCHLEIFENLEN/2 - Ri; j = pos - Ri;

        if (CBand[j] == '$')      /* undef. Inhalte nur in Richtung */
          ERR("FALSCHE RICHTUNG") /* der Klammerung erlaubt         */

/* PRAEFIX vergleichen */

        while (LBE[i] != '$') {
          if (CBand[j] == KlZu)
            if (VERSCHIEBEN(CBand, j)) ERR("PRAEFIXFEHLER")

          if (LBE[i] != CBand[j]) break;
          i -= Ri; j -= Ri;
        }
```

```
        if (LBE[i] != '$') ERR("PRAEFIX STIMMT NICHT")

/* Ausgabe-Klammerung auf Band kopieren */

        j = p - Ri; k = pos + (p - (SCHLEIFENLEN/2 - Ri - Ri)) - Ri;
        while ((a=LBL[j]) != '$') {
          CBand[k] = a; j -= Ri; k -= Ri;
        }

/* neue Position ist hinter der Klammerung */

        pos += p - (SCHLEIFENLEN/2 - Ri - Ri);
CP
      }

    } else {                      /* Turingschritt ausfuehren      */

      a -= '0';

      if (CBand[pos] == '$') {    /* neue Position betreten?       */

        if (pass) {               /* in pass 1 Extremstelle testen */

          if ((FL[posFO] & (MAXI | MINI)) == 0)
            ERR("EXTREMSTELLE ERREICHT, ABER NICHT MARKIERT")

          CBand[pos] = '0';       /* neue Positionen haben Inhalt '0' */

        } else

          CBand[pos] = (a & 1)+'0';  /* erwartete Eingabe schreiben   */

      } else {                      /* Zeichen != '$' unter Kopf     */

        if (VERSCHIEBEN(CBand, pos)) /* Zeichen des Arbeitsalphabetes */
          ERR("SCHIEBEN UNMOEGLICH") /* unter Kopf forcieren          */

        if (pass && (FL[posFO] & (MAXI | MINI)))     /* Extremstelle? */
          ERR("EXTREMSTELLE MARKIERT, ABER NICHT ERREICHT")
      }

/* Bandinhalt muss hier '0' oder '1' sein! */
```



```
      if (CBand[pos] != (a & 1)+'0') /* erwarteter Bandinhalt nicht da */
        ERR("BANDINHALT FALSCH")

      a += a;                       /* Eintrag in ttafel[] umfasst 2 Bytes */
      a = ttafel[a];                        /* 01 & LR Information       */
      CBand[pos] = (a < 0) ? '1' : '0';     /* Turingschritt             */
      if (a & 0x40) pos++; else pos--;
CP
    }
    posFO++;
  }

  return(pos);
}



/*****************************************************************/
/* Funktion SCHLEIFEN_ANALYSE                                    */
/*                                                               */
/* Input:  Zeiger auf die Hauptschleifen-Vermutung, auf das Ein- */
/*         und Ausgabeband und auf ein ueberall undefiniertes Band, */
/*         Index auf den Beginn der zu untersuchenden Klammerung. */
/*                                                               */
/* Output: return(!= 0) <=> Analyse erfolgreich;                 */
/*         in diesem Fall steht die allgemeine Eingabe der       */
/*         Klammerung in LBE[] und die allgemeine Ausgabe in     */
/*         LBL[]; die Bearbeitung der Klammerung wird bei Position */
/*         SCHLEIFENLEN/2 begonnen und die Endposition wird      */
/*         zurueckgegeben.                                       */
/*****************************************************************/

short
SCHLEIFEN_ANALYSE(FO, FOAnf, LBL, LBE, Band_Leer)
char *FO, *LBL, *LBE, *Band_Leer;
short FOAnf;
{
  short i, j, pos, Ri, a, b;

/* 1. Baender loeschen */

  Band_Leer[SCHLEIFENLEN-1] = 0;
  strcpy(LBE,Band_Leer); strcpy(LBL,Band_Leer);
  Band_Leer[SCHLEIFENLEN-1] = '$';
```

```
/* 2. Lauf durch die Klammerung */

  pos = SCHLEIFENLEN/2;
  while ( (a=FO[++FOAnf]) != ')' ) {

    a -= '0';

    if (LBE[pos] == '$')
      LBE[pos] = (a & 1)+'0';        /* erwartete Eingabe */

    else
      if (LBL[pos] != (a & 1)+'0') {
        printf("*** BANDINHALT FALSCH\n"); exit(1);
      }

    a += a;             /* Eintrag in ttafel[] umfasst 2 Bytes */
    a = ttafel[a];                     /* 01 & LR Information */
    LBL[pos] = (a < 0) ? '1' : '0';    /* Turingschritt       */
    if (a & 0x40) pos++; else pos--;

    if (pos <= 3 || pos >= SCHLEIFENLEN-4) {
      printf("*** BANDUEBERLAUF BEI SCHLEIFENANALYSE\n"); exit(1);
    }
  }

/* Bewegungsrichtung der Klammerung bestimmen */

  if (LBL[pos-1] == '$') { Ri = 1; a = ')'; }
  else { Ri = -1; a = '('; }

  i = SCHLEIFENLEN/2;                  /* Startposition */

/* Klammerpaar in Eingabe einfuegen => allgemeine Eingabe */

  while ( (b=LBE[i]) != '$' ) { LBE[i] = a; a = b; i -= Ri; }
  LBE[i] = a;
  LBE[i-Ri] = (Ri > 0) ? '(' : ')';

/* Klammerpaar in Ausgabe einfuegen => allgemeine Ausgabe */

  i = SCHLEIFENLEN/2; j = pos;
  while (LBL[i] != '$') { i += Ri; j += Ri; }
```



```
  a = (Ri > 0) ? '(' : ')';
  while ( (b=LBL[j]) != '$' ) { LBL[j] = a; a = b; j += Ri; }
  LBL[i] = a;
  LBL[i+Ri] = (Ri > 0) ? ')' : '(';

  return(pos);           /* Endposition zurueck */
}


/*************************************************************/
/* Funktion VERSCHIEBEN                                      */
/*                                                           */
/* Input:  Zeiger auf ein Band, aktuelle Position            */
/*                                                           */
/* Output: return(== 0) <=> Verschiebung korrekt ausgefuehrt;  */
/*         in diesem Fall steht auf der aktuellen Bandposition */
/*         keine Klamme mehr, denn sie ist gemaess folgendem   */
/*         Schema verschoben worden:                         */
/*                                                           */
/* (X...)(X...)()...()X -> X(...X)(...X)()...(...X)          */
/* ^                       ^                                 */
/* bzw.                                                      */
/*                                                           */
/*  X()...()(...X)(...X) -> (X...)...()(X...)(X...)X         */
/*                          ^                          ^     */
/*************************************************************/

short
VERSCHIEBEN(Band, Pos)
register char *Band;
register short Pos;
{
  register char KlAuf, KlZu, c;
  register short Ri;

  if (Band[Pos] == '(')      { Ri = 1;  KlAuf = '('; KlZu = ')'; }
  else if (Band[Pos] == ')') { Ri = -1; KlAuf = ')'; KlZu = '('; }

  else return(0);      /* keine Klammer auf Band[Pos] -> fertig  */

  c = Band[Pos+Ri];    /* Zeichen, das unter Band[Pos] erscheint */
```

```
  FOREVER {

    if (Band[Pos] == KlAuf) {                /* (c  ->  c(  */
                                             /* ^         ^ */

      Band[Pos] = Band[Pos+Ri]; Band[Pos+Ri] = KlAuf;
      if (Band[Pos] != c) return(1); /* Verschiebung nicht moeglich */

      Pos += Ri + Ri;

    } else if (Band[Pos] == KlZu) {

      if (Band[Pos+Ri] == KlAuf) {          /* )(c  ->  c)(  */
                                            /* ^           ^ */

        if ( (Band[Pos] = Band[Pos+Ri+Ri]) != c)
          return(1);     /* Verschiebung ist nicht moeglich */

        Band[Pos+Ri] = KlZu; Band[Pos+Ri+Ri] = KlAuf;
        Pos += Ri + Ri + Ri;

      } else {                               /* )X   ->  X)  */
                                             /* ^          ^ */

        Band[Pos] = Band[Pos+Ri]; Band[Pos+Ri] = KlZu;

        if (Band[Pos] != c)  /* Fehler, falls X != c        */
          return(1);         /* Verschiebung nicht moeglich */
        else break;
      }
    } else Pos += Ri;    /* keine Klammer auf aktueller Position */
  }
  return(0);             /* ok */
}
```


Literatur

[Bra66] A. H. Brady, *The conjectured Highest Scoring Machines for Rado's $\Sigma(k)$ for $k = 4$,* IEEE Transactions on Electronic Computers, vol. EC-15, pp. 802-803.

[Bra75] A. H. Brady, *Solution of the non-computable "Busy Beaver" game for $k = 4$,* Abstracts for ACM Computer Science Conference (Washington D.C. February 1975), p. 27.

[Bra83] A. H. Brady, *The Determination of the Value of Rado's Noncomputable Function $\Sigma(k)$ for Four-State Turing Machines,* Mathematics of computation, vol. 40, 1983, pp. 647-665.

[Dew85] A. K. Dewdney, *Computer recreations,* Scientific American, vol. 252, no. 4, 1985, pp. 12-16.

[Gre64] M. W. Green, *A Lower Bound on Rado's Sigma Function for Binary Turing Machines,* 5th IEEE Symposium on Switching Theory, November 1964, pp. 91-94.

[Knu73] D. E. Knuth, *The Art of Computer Programming, vol. 1: Fundamental Algorithms,* Addison-Wesley, 1973.

[LinRad65] S. Lin, T. Rado, *Computer studies of Turing Machine Problems,* J. ACM, vol. 12, no. 2, April 1965, pp. 196-212.

[LudSchWan83] J. Ludewig, U. Schult, F. Wankmüller, *Chasing the Busy Beaver – Notes and Observations on a Competition to find the 5-state Busy Beaver,* Forschungsberichte des Fachbereiches Informatik Nr. 159, Universität Dortmund 1983.

[MarBun90] H. Marxen, J. Buntrock, *Attacking the Busy Beaver 5,* Bulletin of the EATCS, no. 40, February 1990, pp. 247-251.

[Rad62] T. Rado, *On non-computable functions,* Bell System Tech. J., vol. 41, 1962, pp. 877-884.

[WeiCasFen73] B. Weimann, K. Casper, W. Fenzel, *Untersuchungen über haltende Programme für Turing-Maschinen mit 2 Zeichen und bis zu 5 Befehlen,* Proc. 2. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lecture Notes in Economies and Mathematical Systems, Springer Verlag Berlin, vol. 78, 1973, pp. 72-82.