

4. Varianten des Turingmaschinen-Konzeptes

Varianten der Programmstruktur

Varianten der Speicherstruktur

Wir demonstrieren die Robustheit des Turingmaschinen-Konzepts, indem wir eine Reihe von Varianten angeben, die alle zu dem Grundkonzept äquivalent sind, d.h. dieselbe Klasse von (partiell) berechenbaren Funktionen definieren.

*Sprechweise: Wir sagen, dass zwei Maschinen **äquivalent** sind, wenn sie dieselbe (partielle) Funktion berechnen, und wir sagen, dass zwei Maschinenkonzepte **äquivalent** sind, wenn es zu jeder Maschine des einen Konzepts eine äquivalente Maschine des anderen Konzepts gibt.*

Während die im Folgenden betrachtete Variante des Programmformats vor allem einer komfortableren Beschreibung dient, haben die Varianten des Speicherformats einen wesentlichen Einfluss auf die Effizienz (Rechenzeit).

4.1. VARIANTEN DER PROGRAMMSTRUKTUR:

SCHLEIFEN

Die Programme von Turingmaschinen benutzen die Konzepte der maschinen-nahen Programmierung.

Kontrollstrukturen sind

- *Sprünge*
- *Fallunterscheidungen*

zusammengefasst in *bedingten Anweisungen*.

Die der Programmzeile $\delta(z, a) = (a', B, z')$ entsprechende bedingte Instruktion (z, a, a', B, z') entspricht folgendem bedingtem Sprung/Anweisung:

```
z :   if a auf AF then
        begin
            schreibe  $a'$  auf AF;
            verlege AF gemäß B;
            goto  $z'$ 
        end;
```

Die Programmzustände dienen also als Sprungmarken.

In höheren Programmiersprachen werden Sprünge (und Fallunterscheidungen) durch Schleifen ersetzt. Im Folgenden führen wir solch ein höhere Programmiersprache für das Turingmaschinenkonzept ein (die sog. *Turingoperatoren*), die auf *while*-Schleifen basiert.

Wir erinnern hierzu zunächst an das Schleifenkonzept.

ITERATIVE ANWEISUNGEN I: *for*-SCHLEIFEN

n-fache Iteration $\text{Iter}(f, n) = f^n$ einer Operation $f : D \rightarrow D$:

$$\text{Iter}(f, 0)(s) = f^0(s) = s$$

$$\text{Iter}(f, n + 1)(s) = f^{n+1}(s) = f(f^n(s)) = f(\text{Iter}(f, n)(s)),$$

D.h. es wird zu Beginn bereits festgelegt, wie oft f ausgeführt wird (klassische *for*-Schleife):

$$\text{Iter}(f, n)(s) \hat{=} \textit{for } i = 1 \textit{ to } n \textit{ do } s := f(s)$$

ITERATIVE ANWEISUNGEN II: *while*-SCHLEIFEN

Iteration von f nach einem Test $t : D \rightarrow \{0, 1\}$:

$$\text{Iter}_t(f)(s) = f^{n_s}(s),$$

wobei n_s das kleinste n mit $t(f^n(s)) = 1$ ist, falls solch ein n existiert, und $\text{Iter}_t(f)(s) \uparrow$ andernfalls.

D.h. f wird so oft ausgeführt, bis t erstmals positiv ist. Dabei wird *vor* jeder Ausführung von f getestet, ob t gilt; ist t bereits zu Beginn positiv, wird also f nie ausgeführt (*while*-Schleife):

$$\text{Iter}_t(f)(s) \hat{=} \textit{while } t(s) = 0 \textit{ do } s := f(s)$$

TURINGOPERATOREN (TOs)

Auf Schleifen basierende Turingmaschinen-Programme definieren wir mit Hilfe sog. Turingoperatoren.

Solch ein Operator ist eine partielle TM-Speichertransformation basierend auf Verkettung und Iteration der elementaren TM-Operationen (Druck- und Bewegungsbefehle), wobei die elementaren TM-Tests (*Steht a auf dem Arbeitsfeld?*) als Stoppkriterien bei den Iterationen dienen.

Da die Speicherstruktur einer TM nur von deren Bandalphabet Γ abhängt, sprechen wir von *Turingoperatoren P über Γ* . Jeder solcher Operator P ist eine partielle Funktion

$$P : \text{TB}_{\Gamma} \rightarrow \text{TB}_{\Gamma},$$

wobei $\text{TB}_{\Gamma} = \text{BI}_{\Gamma} \times \mathbb{Z}$ die Menge der Turingbänder über dem Bandalphabet Γ sei.

Die Turingoperatoren (TO) P über Γ sind induktiv wie folgt definiert.

1. R, L, S und alle $a \in \Gamma$ sind Turingoperatoren (über Γ), wobei

$$R(f, z) = (f, z + 1) \ \& \ L(f, z) = (f, z - 1) \ \& \ S(f, z) = (f, z)$$

$$a(f, z) = (f_{(a,z)}, z) \quad (\text{wobei } f_{(a,z)}(z) = a \text{ und } f_{(a,z)}(z') = f(z') \text{ f\u00fcr } z' \neq z)$$

Die Turingoperatoren dieser Gruppe sind also gerade die elementaren TM-Operationen.

2. Sind P_1 und P_2 TOs, so auch P_1P_2 , wobei

$$P_1P_2(f, z) = P_2(P_1(f, z))$$

Verkettung (= Hintereinanderausf\u00fchrung) von P_1 und P_2
Sprechweise: „erst P_1 , dann P_2 “

3. Ist P ein TO und $a \in \Gamma$, so ist auch $[P]_a$ ein TO, wobei

$$[P]_a(f, z) = \text{Iter}_{a?}(P)(f, z).$$

Iteration von P nach $a?$ (wobei $a?(f, z) = 1 \Leftrightarrow f(z) = a$)

Sprechweise: „iteriere P so lange, bis a erstmals auf dem Arbeitsfeld steht“

BEISPIELE von TOs über dem Bandalphabet $\Gamma = \{b, 0, 1\}$

Rechts- und Linksoperatoren ($a \in \Gamma$):

$$R_a := [R]_a \text{ und } L_a := [L]_a$$

Verlege das AF so lange nach rechts (links), bis es erstmals den Buchstaben a enthält! (Steht a bereits auf dem AF, so wird das AF nicht verlegt!)

Bandendeoperatoren:

$$R_{bb} := R_b R [R_b R]_b L \text{ und } L_{bb} := L_b L [L_b L]_b R$$

Verlege das AF so lange nach rechts (links), bis erstmals das AF und sein rechtes (linkes) Nachbarfeld beide leer sind!

Stellt man sicher, dass im relevanten Bandteil niemals zwei Blanks direkt nebeneinander stehen, so wird das AF an das rechte (linke) Ende des relevanten Bandteils verlegt.

Kopieroperator: $K := R1R_{bb}R0L_10R[1R_{bb}0L_10R]_b$

Liegt das AF direkt vor einer endlichen Folge von Unärzahlen, so wird die erste dieser Zahlen an das rechte Bandende kopiert und das AF wird hinter die kopierte Zahl verlegt. Der Bandteil links der ursprünglichen Position des Arbeitsfeldes wird dabei nicht verändert.

$$K(\dots v \underline{b} \underline{n_1} b \underline{n_2} b \dots \underline{n_k} b \dots) = \dots v b \underline{n_1} \underline{b} \underline{n_2} b \dots \underline{n_k} b \underline{n_1} b \dots$$

Löschoperator: $E := R[bR]_b$

Liegt das AF direkt vor einer Unärzahl, so wird diese gelöscht und das AF wird hinter die gelöschte Zahl verlegt. Der Rest des Bandes wird nicht verändert.

$$E(\dots v \underline{b} \underline{n} b w \dots) = \dots v b b^{n+1} \underline{b} w \dots$$

Durch das Löschen entsteht eine Lücke im relevanten Bandteil. Liegt diese zwischen den letzten beiden Unärzahlen auf dem Band, so kann man die Lücke mit Hilfe des folgenden Transportoperators schließen.

Transportoperator: $T := R1L_0RR0R_1bR[1L_0R0R_1bR]_bL_0L_b$

$$T(\dots v \underline{m} b^{k+1} \underline{b} \underline{n} b \dots) = \dots v \underline{m} \underline{b} \underline{n} b \dots$$

VON EINEM TO P BERECHNETE (PARTIELLE) FUNKTION

Die von einem TO P über Γ beschriebene Speichertransformation kann man als Rechnung auffassen. Fügt man den üblichen Ein- und Ausgabemechanismus von Turingmaschinen hinzu, so erhält man die von P berechnete Funktion(en).

Seien $\Sigma, \mathbb{T} \subseteq \Gamma - \{b\}$ und $m \geq 0$. Die *von dem TO P über Γ berechnete partielle Funktion*

$$\varphi_{P,\Sigma,\mathbb{T},m} : (\Sigma^*)^m \rightarrow \mathbb{T}^*$$

ist durch

$$\varphi_{P,\Sigma,\mathbb{T},m}(\vec{w}) = out(P(in(\vec{w})))$$

gegeben, wobei in und out wie die Ein- und Ausgabefunktion einer Turingmaschine $M = (\Sigma, m, \mathbb{T}, \Gamma, Z, z_0, \delta)$ über dem Bandalphabet Γ zur Berechnung von Funktionen vom Typ $(\Sigma^*)^m \rightarrow \mathbb{T}^*$ definiert sind (wobei bei der Ausgabe (von in) bzw. bei der Eingabe (von out) der Zustand unterdrückt wird).

BEISPIELE:

1. Die in einem früheren Beispiel beschriebene Vorgehensweise zur Berechnung der Summe zweier Zahlen lässt sich mit einem TO wie folgt beschreiben:

$$P_+ = R[R]_b 0[L]_b RbRb = RR_b 0L_b RbRb$$

2. Die Funktion $f(n) = 2n$ wird von den folgenden Operatoren (jeweils auf unterschiedliche Weise) berechnet:

$$P_f = R_{bb} R 0 L_{bb} RbR [R_{bb} 0 R 0 L_{bb} RbR]_b$$

$$P'_f = K 0 L_{bb} RbRb$$

$$P''_f = K L_{bb} P_+$$

Bei TOs gehen wir (wie bei TMs) davon aus, dass Zahlen in der modifizierten Unärdarstellung $\underline{n} = 0^{n+1}$ gegeben sind.

ÄQUIVALENZ VON TURINGMASCHINEN UND TURINGOPERATOREN

Turingmaschinen und Turingoperatoren berechnen dieselben Zahlfunktionen.
D.h. für

- $F(\text{TM})^{(n)} = \{\psi : \mathbb{N}^n \rightarrow \mathbb{N} : \psi \text{ partiell TM-berechenbar}\}$
 $F(\text{TM}) = \bigcup_{n \geq 0} F(\text{TM})^{(n)}$ und $F_{\text{tot}}(\text{TM}) = \{f \in F(\text{TM}) : f \text{ total}\}$
- $F(\text{TO})^{(n)} = \{\psi : \mathbb{N}^n \rightarrow \mathbb{N} : \psi \text{ partiell TO-berechenbar}\}$
 $F(\text{TO}) = \bigcup_{n \geq 0} F(\text{TO})^{(n)}$ und $F_{\text{tot}}(\text{TO}) = \{f \in F(\text{TO}) : f \text{ total}\}$

gilt:

SATZ. $F_{(\text{tot})}(\text{TO}) = F_{(\text{tot})}(\text{TM})$.

Wir zeigen zunächst nur die Inklusion $F_{(\text{tot})}(\text{TO}) \subseteq F_{(\text{tot})}(\text{TM})$. Die andere Inklusion werden wir später zeigen.

Da Ein/Ausgabe bei TOs und TMs gleich definiert ist, genügt es folgendes Lemma zu zeigen.

LEMMA. Zu jedem TO P über Γ gibt es eine Turingmaschine M über dem Bandalphabet Γ mit Startzustand α_M und ausgezeichnetem (Stopp-)Zustand ω_M , sodass für alle Bänder (f, z) über Γ gilt:

(i) Ist $P(f, z)$ definiert, so ist die maximale mit $(\alpha_M, (f, z))$ beginnende Konfigurationenfolge endlich und endet mit der Stoppkonfiguration $(\omega_M, P(f, z))$.

(ii) Ist $P(f, z)$ undefiniert, so ist die maximale mit $(\alpha_M, (f, z))$ beginnende Konfigurationenfolge unendlich.

BEWEIS DES LEMMAS:

Der Beweis ist durch Induktion nach Aufbau (= Länge) des TO P .

1. $P = a \in \Gamma$ oder $P = B \in \text{Bew}$. Dann besteht das Programm δ von M aus den Instruktionen $(\alpha_M, a', a, S, \omega_M)$ bzw. $(\alpha_M, a', a', B, \omega_M)$ für alle $a' \in \Gamma$.

2. $P = P_1P_2$. Nach I.V. gibt es dann die die TOs P_i simulierenden Maschinen M_i mit Zustandsmengen Z_i und ausgezeichneten Zuständen α_{M_i} und ω_{M_i} ($i = 1, 2$). Durch eventuelles Umbenennen der Zustände können wir erreichen, dass $Z_1 \cap Z_2 = \{\omega_{M_1}\}$ und $\omega_{M_1} = \alpha_{M_2}$ gilt. Das Programm von M ist dann die Vereinigung der Programme von M_1 und M_2 , $\alpha_M = \alpha_{M_1}$ und $\omega_M = \omega_{M_2}$.

3. $P = [P_1]_a$ ($a \in \Gamma$). Nach I.V. gibt es dann eine den TO P_1 simulierende Maschine M_1 mit ausgezeichneten Zuständen α_{M_1} und ω_{M_1} . Das Programm der Maschine M erhält man aus dem Programm von M_1 durch Hinzufügen der Instruktionen $(\alpha_M, a, a, S, \omega_M)$ und $(\alpha_M, a', a', S, \alpha_{M_1})$ (für alle $a' \neq a$), wobei $\alpha_M = \omega_{M_1}$ während ω_M neu ist.

4.2 VARIANTEN DER SPEICHERSTRUKTUR:

MEHRBAND-TURINGMASCHINEN UND
HALBBAND-TURINGMASCHINEN

MEHRBAND-TURINGMASCHINEN

Der Speicherzugriff bei Turingmaschinen ist recht umständlich. Man erhält effizientere Speicherstrukturen, wenn man mehrere Köpfe oder mehrere Bänder (mit je einem Kopf) zulässt (oder beides). Die Anzahl der Köpfe bzw. Bänder ist dabei fest, die Zugriffe der verschiedenen Köpfe bzw. auf die verschiedenen Bänder voneinander unabhängig.

Wir formalisieren hier das Konzept der k -Band-TM ($k \geq 1$) und überlassen die Formalisierung der (weniger populären) k -Kopf TM (oder allgemeiner k -Kopf- k' -Band-TM) als Übung. Unser bisheriges Konzept wird gerade der Spezialfall der 1-Band-TM sein.

k -BAND-TURINGMASCHINEN

Seien Σ, T, Γ Alphabete mit $\Sigma \cup T \subseteq \Gamma \setminus \{b\}$ und seien $k, m \geq 1$.

Eine k -Band-Turing-Basismaschine M mit Bandalphabet Γ zur Berechnung m -stelliger partieller Funktionen von Σ^* nach T^* wird durch ein Tupel

$$M = (k, \Sigma, m, T, \Gamma, Z, z_0, \delta)$$

gegeben, wobei k die Anzahl der Bänder ist und - wie bei den bisher betrachteten (1-Band-)TMs - Σ, T und Γ das Eingabe-, Ausgabe- und Bandalphabet, Z die endliche Menge der Zustände und $z_0 \in Z$ der ausgezeichnete Startzustand sind. Das Programm δ ist nun eine partielle Funktion vom Typ

$$\delta : Z \times \Gamma^k \rightarrow (\Gamma \times \text{Bew})^k \times Z.$$

Eine 'Zeile' $\delta(z, a_1, \dots, a_k) = (a'_1, B_1, \dots, a'_k, B_k, z')$ von δ (in Tabellenform dargestellt) wird als bedingte Anweisung $(z, a_1, \dots, a_k, a'_1, B_1, \dots, a'_k, B_k, z')$ mit Bedingungsteil (z, a_1, \dots, a_k) und Anweisungsteil $(a'_1, B_1, \dots, a'_k, B_k, z')$ gelesen und wie folgt interpretiert: Stehen im Programmzustand z die Buchstaben a_1, \dots, a_k auf den Arbeitsfeldern der k -Bänder von M , so werden diese zunächst mit a'_1, \dots, a'_k neu beschriftet und dann gemäß den Bewegungen B_1, \dots, B_k verlegt. Schließlich wird z' als neuer Zustand angenommen.

ARBEITSWEISE DER k -BAND-TM M

Wir verzichten auf eine formale Beschreibung der Arbeitsweise der k -Band-TM M , da sich diese aus der bereits gegebenen Interpretation der einzelnen Instruktionen entsprechend zum Fall der 1-Band-TM leicht angeben lässt. (Z.B. ist eine Konfiguration nun ein Tupel $(f_1, p_1, \dots, f_k, p_k, z)$, wobei (f_i, p_i) das i -te Band und z der (Programm-)Zustand ist.)

Festlegen müssen wir hierzu lediglich noch den Ein- und Ausgabemechanismus von M :

EINGABE

- Die Eingabe wird rechts des Arbeitsfeldes auf das ansonsten leere *erste* Band geschrieben. Die anderen Bänder sind leer.

AUSGABE

- Die Ausgabe wird dem *letzten* Band rechts des Arbeitsfeldes entnommen.

RECHENZEIT UND PLATZBEDARF DER k -BAND-TM M

Rechenzeit und Platzbedarf sind in Entsprechung zum Fall der Einbandmaschine definiert:

RECHENZEIT

Terminiert M bei Eingabe \vec{x} , so ist die Rechenzeit von M bei Eingabe \vec{x} , $time_M(\vec{x})$, die Länge der Rechnung. Terminiert M nicht, so ist $time_M(\vec{x})$ undefiniert.

PLATZBEDARF

Die Größe einer Konfiguration ist nun die kleinste Zahl s , sodass der relevante Bandteil aller Bänder im Adressintervall $[-s, +s]$ liegt. Hieraus erhält man den Platzbedarf dann wie gehabt: Terminiert M bei Eingabe \vec{x} , so ist der Platzbedarf von M bei Eingabe \vec{x} , $space_M(\vec{x})$, das Maximum der Größen der in der Rechnung vorkommenden Konfigurationen. Terminiert M nicht, so ist $space_M(\vec{x})$ undefiniert.

EIN BEISPIEL FÜR DIE HÖHERE EFFIZIENZ VON MEHRBAND-TMs

Wir betrachten die Sprache der Palindrome über dem binären Alphabet:

$$A = \{w \in \{0, 1\}^* : w = w^R\}$$

Eine 2-Band-TM $M = (2, \{0, 1\}, 1, \{1\}, \{b, 0, 1\}, Z, z_0, \delta)$, die A erkennt (d.h. die charakteristische Funktion c_A von A berechnet) arbeitet anschaulich wie folgt:

- (1) Lese die Eingabe w (von links nach rechts) und schreibe gleichzeitig das Spiegelwort w^R (von rechts nach links) auf Band 2.
- (2) Setze den Kopf auf Band 1 vor die Eingabe w zurück.
(Der Kopf auf Band 2 steht bereits vor w^R !)
- (3) Lese w auf Band 1 und w^R auf Band 2 gleichzeitig (jeweils von links nach rechts). Wird ein Unterschied gefunden, so wird 0 (= 0) ausgegeben, andernfalls 1 (= 00).

Die Rechenzeit von M ist linear, d.h. $time_M(w) = O(|w|)$.

Programm δ der Maschine M wobei $Z = \{0, \dots, 5\}$, $z_0 = 0$ und $i = 0, 1$:

Z	\times	Γ	\times	Γ	\rightarrow	Γ	\times	Bew	\times	Γ	\times	Bew	\times	Z
0		b		b		b		R		b		S		1
1		i		b		i		R		i		L		1
1		b		b		b		L		b		S		2
2		i		b		i		L		b		S		2
2		b		b		b		R		b		R		3
3		i		i		b		R		b		R		3
3		i		$1 - i$		b		S		b		L		4
3		b		b		b		S		0		L		4
4		b		b		b		S		0		L		5

Das naive Programm zur Lösung des Palindromproblems auf einer 1-Band-TM hat dagegen einen Zeitbedarf der Größenordnung $O(|w|^2)$: Hier vergleicht man den ersten mit dem letzten Buchstaben, streicht diese und führt (bei Gleichheit) das Verfahren rekursiv für das Restwort aus. Dies erfordert je Vergleich $O(|w|)$, $O(|w| - 2)$, $O(|w| - 4)$, ... Schritte, da das Wort w bzw. das aktuelle Restwort hierzu durchlaufen werden muss.

Man kann tatsächlich zeigen, dass *jede* 1-Band-TM zur Lösung des Palindromproblems für unendlich viele Eingaben einen der Größenordnung nach quadratischen Zeitbedarf hat. Dies zeigt, dass Mehrband-TMs im allgemeinen effizienter als 1-Band-TMs sind, weshalb man bei Komplexitätsfragen das Mehrband-TM-Modell zugrundelegt.

Mehrband-TMs sind jedoch nicht prinzipiell mächtiger als 1-Band-TMs.

BANDREDUKTIONSSATZ.

Das Mehrband-Turingmaschinen-Konzept ist äquivalent zum (1-Band-)Turingmaschinen-Konzept. D.h. zu jeder k -Band-Turingmaschine $M = (k, \Sigma, m, \top, \Gamma, Z, z_0, \delta)$ mit $k \geq 1$ gibt es eine äquivalente (1-Band-)TM $M' = (\Sigma, m, \top, \Gamma', Z', z'_0, \delta')$.

Gehen wir davon aus, dass $m = 1$ und $time_M(x) \geq |x|$ stets gilt, so lässt sich Zeit- und Platzbedarf von M' wie folgt abschätzen:

$$time_{M'}(x) \leq O(time_M(x)^2)$$

$$space_{M'}(x) \leq space_M(x) + O(1)$$

D.h. die Einschränkung auf ein Band kann zu einem quadratischen Zeitverlust führen aber zu (praktisch) keinem Platzverlust.

BEWEISIDEE:

Die 1-Band-TM M' simuliert die gegebene k -Band-TM M Schritt-für-Schritt. D.h. jeder Rechenschritt von M wird durch eine geeignete Rechenschrittfolge von M' simuliert.

Hierzu wählt man das Bandalphabet Γ' von M' so, dass (intuitiv) das Band in $2k$ Spuren zerlegt wird, wobei zu jedem Band von M zwei dieser Spuren korrespondieren. In der oberen Spur wird das Arbeitsfeld des k -ten Bandes durch ein $+$ (sonst $-$) markiert, während die untere Spur die Bandinschrift enthält.

In der folgenden Beschreibung der Details der Konstruktion beschränken wir uns auf den Fall $k = 2$ und $m = 1$.

BEWEISIDEE (Fortsetzung):

Das Programm δ' von M' besteht dann aus 3 Teilprogrammen $\delta' = \delta'_1 \cup \delta'_2 \cup \delta'_3$ mit folgenden Funktionen:

- δ'_1 : Überführung der Startkonfiguration in das (kodierte) Format der Mehrbandmaschine.
- δ'_2 : Schritt-für-Schritt-Simulation von M : Beginnend im Zustand $[z, a_1, a_2]$ und dem Arbeitsfeld am linken Rand des relevanten Bandteils, simuliert M' die bedingte M -Instruktion $I = (z, a_1, a_2, a'_1, B_1, a'_2, B_2, z')$, indem es die kodierten M -Arbeitsfelder aufsucht und deren Inschriften und Positionen aktualisiert, dann an den linken Rand des relevanten Bandteils zurückläuft und in den Zustand $[z', a'_1, a'_2]$ geht.
- δ'_3 : Überführung der kodierten Stoppkonfiguration der Mehrband-Maschine in die äquivalente Stoppkonfiguration der 1-Band-Maschine.

BEWEIS (Teil 1): Das Bandalphabet Γ' von M'

Im Folgenden beschreiben wir M' formal, wobei wir uns jedoch auf den Fall einer 2-Band-TM M (d.h. $k = 2$) zur Berechnung einer 1-stelligen Funktion (d.h. $m = 1$) beschränken.

$$\Gamma' = \Gamma \cup \{(\sigma_1, a_1, \sigma_2, a_2)^T : \sigma_1, \sigma_2 \in \{+, -\} \ \& \ a_1, a_2 \in \Gamma\} \cup \{[,]\}$$

- $(\sigma_1, a_1, \sigma_2, a_2)$ beschreibt die Inschriften der 4 Spuren eines Feldes von links nach rechts (d.h. der transponierte Vektor $(\sigma_1, a_1, \sigma_2, a_2)^T$ beschreibt diese von oben nach unten):

σ_i gibt an, ob es sich um das Arbeitsfeld von M auf Band i handelt, und a_i ist die Feldbeschriftung auf Band i ($i = 1, 2$).

- Mit [und] klammern wir den in die Spurendarstellung gebrachten, relevanten Bandteil ein.

BEWEIS (Teil 2): Das Teilprogramm δ'_1

Das Programm δ'_1 muss für ein Eingabewort $w = w(0) \dots w(n) \in \Sigma^*$ ($n \geq -1$) die folgende Konfigurationentransformation leisten (wir markieren das M' -Arbeitsfeld mit \uparrow statt $_$):

$$\begin{array}{ccc}
 \dots bw(0) \dots w(n)b \dots & \xRightarrow{*}_{M'} & \dots [\begin{array}{ccc} + & - & - \\ b & w(0) & \dots & w(n) \end{array}] \dots \\
 \uparrow & & \uparrow \\
 z'_0 & & [z_0, b, b] \quad \begin{array}{ccc} + & - & - \\ b & b & b \end{array}
 \end{array}$$

Dies wird realisiert durch:

Z'	\times	Γ'	\rightarrow	Γ'	\times	Bew	\times	Z'
z'_0		b		b		L		1
1		b		[R		2
2		b		$(+, b, +, b)^T$		R		3
3		$a \in \Sigma$		$(-, a, -, b)^T$		R		3
3		b]		L		4
4		$a \in \Gamma' - \{[\}$		a		L		4
4		[[S		$[z_0, b, b]$

BEWEIS (Teil 3): Das Teilprogramm δ'_2

Das Programm δ'_2 besitzt für jede bedingte Instruktion I von M ein Programmstück δ'_I zur Simulation von I . Die Struktur von δ'_I hängt dabei von den Bewegungsbefehlen in I ab. Von den $9 = 3 \cdot 3$ möglichen Fällen betrachten wir hier nur einen typischen Fall (Linksbewegung auf Band 1; Rechtsbewegung auf Band 2), d.h. wir nehmen an, dass I die Gestalt

$$I = (z, a_1, a_2, a'_1, L, a'_2, R, z')$$

entsprechend der Programmzeile

$$\delta(z, a_1, a_2) = (a'_1, L, a'_2, R, z')$$

hat.

BEWEIS (Teil 4): Das Teilprogramm δ'_I von δ'_2 (Arbeitsweise)

Das Teilprogramm δ'_I zur Simulation von $I = (z, a_1, a_2, a'_1, L, a'_2, R, z')$ muss die Konfiguration

$$\begin{array}{cccccccc}
 & & & - & + & - & & - & - & - & & \\
 \dots & [& \dots & a_{1,l} & a_1 & a_{1,r} & \dots & \widehat{a}_{1,l} & \widehat{a}_1 & \widehat{a}_{1,r} & \dots] & \dots \\
 & \uparrow & & - & - & - & & - & + & - & & \\
 & [z, a_1, a_2] & & \widehat{a}_{2,l} & \widehat{a}_2 & \widehat{a}_{2,r} & & a_{2,l} & a_2 & a_{2,r} & &
 \end{array}$$

in folgende Konfiguration überführen:

$$\begin{array}{cccccccc}
 & & & + & - & - & & - & - & - & & \\
 \dots & [& \dots & a_{1,l} & a'_1 & a_{1,r} & \dots & \widehat{a}_{1,l} & \widehat{a}_1 & \widehat{a}_{1,r} & \dots] & \dots \\
 & \uparrow & & - & - & - & & - & - & + & & \\
 & [z', a'_1, a'_2] & & \widehat{a}_{2,l} & \widehat{a}_2 & \widehat{a}_{2,r} & & a_{2,l} & a'_2 & a_{2,r} & &
 \end{array}$$

Hierbei muss das M-Arbeitsfeld AF1 auf Band 1 nicht links des M-Arbeitsfeldes AF2 auf Band 2 liegen, sondern es kann auch rechts davon oder an der selben Stelle liegen. Weiter muss, falls AF1 direkt rechts von [bzw AF2 direkt links von] liegt, der relevante Bandteil entsprechend erweitert werden, um die erforderliche Links- bzw. Rechtsbewegung durchführen zu können.

Die Idee des folgenden Programmes δ'_I ist, zunächst das 1. Band zu aktualisieren, dann das 2. Band.

BEWEIS (Teil 5): Das Teilprogramm δ'_I von δ'_2 (Definition)

Z'	Γ'	Γ'	Bew	Z'
$[z, a_1, a_2]$	$[$	$[$	R	$[z, a_1, a_2, -, -]$
$[z, a_1, a_2, -, -]$	$(-, a, +/ -, a')^T$	$(-, a, +/ -, a')^T$	R	$[z, a_1, a_2, -, -]$
$[z, a_1, a_2, -, -]$	$(+, a_1, +/ -, a')^T$	$(-, a'_1, +/ -, a')^T$	L	$[z, a_1, a_2, \leftarrow, -]$
$[z, a_1, a_2, \leftarrow, -]$	$(-, a, +/ -, a')^T$	$(+, a, +/ -, a')^T$	L	$[z, a_1, a_2, +, -] \leftarrow$
$[z, a_1, a_2, +, -] \leftarrow$	$(-, a, +/ -, a')^T$	$(-, a, +/ -, a')^T$	L	$[z, a_1, a_2, +, -] \leftarrow$
$[z, a_1, a_2, +, -] \leftarrow$	$[$	$[$	R	$[z, a_1, a_2, +, -]$
$[z, a_1, a_2, \leftarrow, -]$	$[$	$(+, b, -, b)^T$	L	$[z, a_1, a_2, +, -][$
$[z, a_1, a_2, +, -][$	b	$[$	R	$[z, a_1, a_2, +, -]$
$[z, a_1, a_2, +, -]$	$(+/-, a, -, a')^T$	$(+/-, a, -, a')^T$	R	$[z, a_1, a_2, +, -]$
$[z, a_1, a_2, +, -]$	$(+/-, a, +, a_2)^T$	$(+/-, a, -, a'_2)^T$	R	$[z, a_1, a_2, +, \rightarrow]$
$[z, a_1, a_2, +, \rightarrow]$	$(+/-, a, -, a')^T$	$(+/-, a, +, a')^T$	L	$[z, a_1, a_2, +, +] \leftarrow$
$[z, a_1, a_2, +, +] \leftarrow$	$(+/-, a, +/ -, a')^T$	$(+/-, a, +/ -, a')^T$	L	$[z, a_1, a_2, +, +] \leftarrow$
$[z, a_1, a_2, +, +] \leftarrow$	$[$	$[$	S	$[z', a'_1, a'_2]$
$[z, a_1, a_2, +, \rightarrow]$	$]$	$(-, b, +, b)^T$	R	$[z, a_1, a_2, +, +]]$
$[z, a_1, a_2, +, +]]$	b	$]$	L	$[z, a_1, a_2, +, +] \leftarrow$

BEWEIS (Teil 6): Das Teilprogramm δ'_3 (Arbeitsweise)

Die Simulationsphase ist beendet, wenn M' einen Zustand $[z, a_1, a_2]$ erreicht, sodass $\delta(z, a_1, a_2)$ undefiniert ist. Der 3. Programmteil von δ' stellt dann die Ausgabe für das 1-Bandmodell her, indem es im Bereich der Ausgabe die Spurendarstellung auflöst und nur die Inschrift des 2. Bandes (des Ausgabebandes) bewahrt. D.h. für jedes $[z, a_1, a_2]$ mit $\delta(z, a_1, a_2) \uparrow$ wird δ'_3 die Konfiguration

$$\begin{array}{cccccccc}
 & & & \sigma_{1,0} & \sigma_{1,1} & & \sigma_{1,n+1} & \sigma_{1,n+2} \\
 \dots & & [& \dots & a_{1,0} & a_{1,1} & \dots & a_{1,n+1} & a_{1,n+2} & \dots] & \dots \\
 & & \uparrow & & + & - & & - & - & & \\
 & & [z, a_1, a_2] & & a_2 & v(0) & & v(n) & a & &
 \end{array}$$

(wobei $v = v(0) \dots v(n) \in T^*$ ($n \geq -1$) und $a \in \Gamma \setminus T$) in die Konfiguration

$$\begin{array}{ccccccc}
 & & & & & \sigma_{1,n+2} & \\
 \dots & b & v(0) & \dots & v(n) & a_{1,n+2} & \dots] & \dots \\
 & \uparrow & & & & - & & \\
 & z'_e & & & & a & &
 \end{array}$$

überführen (z'_e ist der 'Stoppzustand' von M').

BEWEIS (Teil 7): Das Teilprogramm δ'_3 (Definition)

Z'	\times	Γ'	\rightarrow	Γ'	\times	Bew	\times	Z'
$[z, a_1, a_2]$		[b		R		\rightarrow
\rightarrow		$(+/-, a, -, a')$		b		R		\rightarrow
\rightarrow		$(+/-, a, +, a')$		b		R		<i>convert</i>
<i>convert</i>		$(+/-, a, -, a') (a' \in T)$		a'		R		<i>convert</i>
<i>convert</i>		$(+/-, a, -, a') (a' \notin T)$	$(+/-, a, -, a')$			L		\leftarrow
\leftarrow		$a \in T$		a		L		\leftarrow
\leftarrow		b		b		S		z'_e

Hierbei ist in der ersten Programmzeile $[z, a_1, a_2]$ ein beliebiger M' -Zustand, für den $\delta(z, a_1, a_2) \uparrow$ gilt; $a, a' \in \Gamma$ beliebig.

Hiermit ist δ' und damit M' vollständig beschrieben.

BEWEIS (Teil 8): Komplexität von M'

PLATZBEDARF: Da nur die Randmarken hinzugefügt werden müssen, gilt

$$(1) \text{ space}_{M'}(x) \leq \text{space}_M(x) + O(1)$$

ZEITBEDARF: Die Rechenzeit setzt sich zusammen aus

Initialisierung: Die Zeit ist linear in der Eingabengänge beschränkt, also

$$(2_1) \text{ time}_{in}(x) \leq O(|x|) \leq O(\text{time}_M(x))$$

Simulation: Die Zeit um einen M -Schritt zu simulieren ist linear beschränkt in der Länge der aktuellen M' -Konfiguration, also wegen (1) linear beschränkt in $\text{space}_M(x)$. Da (wegen $\text{time}_M(x) \geq |x|$), $\text{space}_M(x)$ linear beschränkt in $\text{time}_M(x)$ ist, folgt

$$(2_2) \text{ time}_{sim}(x) \leq \text{time}_M(x) \cdot O(\text{time}_M(x))$$

Endphase: Die Zeit ist linear in den Konfigurationenlängen beschränkt, weshalb (wie oben)

$$(2_3) \text{ time}_{out}(x) \leq O(\text{time}_M(x))$$

Insgesamt gilt also:

$$(2) \text{ time}_{M'}(x) = \text{time}_{in}(x) + \text{time}_{sim}(x) + \text{time}_{out}(x) \leq O(\text{time}_M(x)^2)$$

Weitere Speichervarianten bei Turingmaschinen:

- *Mehrdimensionale Turingmaschinen*

Im 2-dimensionalen Fall bestehen hier die Felder aus den „Kästchen“ der Zahlenebene, d.h. die Adressierung der Felder erfolgt hier durch \mathbb{Z}^2 . Bei den Kopfbewegungen kommen entsprechend die Befehle O („nach oben“) und U („nach unten“) hinzu. Den höher dimensional Fall beschreibt man analog.

- *Halbband-Turingmaschinen*

Hier ist das Band (oder die Bänder) nur nach rechts unbeschränkt, d.h. die Adressen sind natürliche statt ganzer Zahlen.

Auch hier lässt sich wiederum die Äquivalenz zum ursprünglichen Konzept zeigen. Wir betrachten hier nur den Fall der Halbband-Maschinen.

Beweisidee für die Simulation einer (Vollband-) Turingmaschine M durch eine Halbband-TM M' mit Hilfe einer Variante der Spurentechnik: Das M' -Halbband besteht aus 2 Spuren, wobei die obere Spur die rechte Bandhälfte von M und die untere Spur die gespiegelte linke Bandhälfte aufnimmt. (D.h. anschaulich, dass man vor Beginn der Rechnung das M -Band am Arbeitsfeld zerschneidet und die linke Bandhälfte unter die rechte Bandhälfte umklappt.)

Sowohl Rechenzeit als auch Platzbedarf von M und M' sind linear korreliert, d.h. Zeit- und Platzkomplexität von (1-Band-)TMs und Halbband-TMs unterscheiden sich nur um einen linearen Faktor.

BEMERKUNG: Ähnlich zeigt man die Äquivalenz von Turingmaschinen und 2-Stapel-Maschinen (oder – wie man meist sagt – 2-Kellermaschinen oder 2-Push-Down-Automaten). Der Speicher einer k -Stapel-Maschine über dem Speicheralphabet Γ besteht aus k Stapeln (stacks) über Γ . Bei der Simulation stellt man die linke Bandhälfte der TM durch den ersten Stapel, die rechte Bandhälfte durch den zweiten Stapel dar (und umgekehrt).

1-Stapel-Maschinen sind dagegen weit weniger mächtig, wie wir im Teil über Formale Sprachen sehen werden.

Eine weitere Variante des Speichers erhalten wir durch Normierung des Bandalphabets:

Eine Turingmaschine $M = (\Sigma, m, T, \Gamma, Z, z_0, \delta)$ ist *Bandalphabet-normiert* wenn

$$\Gamma = \Sigma \cup T \cup \{b, 0, 1\}$$

gilt (wobei $0, 1$ in $\Sigma \cup T$ vorkommen dürfen). Als zusätzliche Hilfszeichen werden im Speicher neben dem Blank also höchstens die Bits 0 und 1 zugelassen.

NORMIERUNGSSATZ. Zu jeder (k -Band-)Turingmaschine M gibt es eine äquivalente Bandalphabet-normierte (k -Band-)Turingmaschine M' . Weiter gilt, dass $time_{M'}(x) \leq O(time_M(x))$ und $space_{M'}(x) \leq O(space_M(x))$.

BEWEISIDEE: Im Speicher von M' werden die Buchstaben a_i des Bandalphabets $\Gamma = \{a_1, \dots, a_n\}$ durch deren Binärkodierung $0^i 1^{n-i}$ dargestellt. Jeder Buchstabe aus Γ wird also durch einen Block der Länge n von Buchstaben über dem normierten Bandalphabet dargestellt, weshalb sich der Platzbedarf um den Faktor n erhöht. Entsprechendes gilt für die Zeit, da das Lesen eines kodierten Buchstabens aus Γ die Zeit $O(n)$ erfordert. Ähnliches gilt für das Schreiben und das Verlegen des M -Arbeitsfeldes.

KONKLUSION:

Das “minimalistische” Turingmaschinenmodell ist äquivalent zu “komfortableren” Modellen, die man durch Veränderungen des Programmformats bzw. der Speicherstruktur erhält. Für Untersuchungen des Berechenbarkeitsbegriffs sowie der Grenzen der Berechenbarkeit ist das Grundmodell daher ausreichend.

Die betrachteten Erweiterungen der Turingmaschine erlauben jedoch z.T. schnellere Verfahren. Für Komplexitätsuntersuchungen ist das Basismodell daher nicht adäquat. Hier werden wir das Mehrband-Turingmaschinenmodell zugrundelegen.