
6. Varianten des Turingmaschinen-Konzeptes II: Varianten der Speicherstruktur

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 k -Kopf TM (oder allgemeiner k -Kopf- k^l -Band-TM) als Übung.

6.1 DEFINITION. Seien $\Sigma, T, \Gamma = \{b = a_0, \dots, a_m\}$ Alphabete mit $\Sigma \cup T \subseteq \Gamma - \{b\}$ und sei $k, n \geq 1$. Die k -Band-Turing-Basismaschine mit Bandalphabet Γ zur Berechnung n -stelliger partieller Funktionen von Σ^* nach T^* ist die mathematische Basismaschine

$$TB_k(\Sigma, T, \Gamma, n) = (I, O, S, \text{in}, \text{out}, \text{OPER}, \text{TEST})$$

wobei

- $I = (\Sigma^*)^n$
- $O = T^*$
- $S = (BI \times \mathbb{Z})^k$
Für $s = (f_1, z_1, \dots, f_k, z_k)$ heißt f_l die *Bandinschrift* und z_l die *Position des Arbeitsfeldes des l -ten Bandes* ($1 \leq l \leq k$) der Speicherbelegung s .
- $\text{in} : (\Sigma^*)^n \rightarrow S$ ist gegeben durch

$$\text{in}(w_1, \dots, w_n) = (f_{w_1, \dots, w_n}, 0, f_b, 0, \dots, f_b, 0),$$

d.h. die n Eingaben werden durch Blanks getrennt hinter das Arbeitsfeld auf das erste Band geschrieben. Die übrigen Bänder sind leer.

- $\text{out} : S \rightarrow T^*$ ist gegeben durch

$$\text{out}(f_1, z_1, \dots, f_k, z_k) = f_k(z_k + 1) \dots f_k(z_k + l - 1),$$

wobei l die kleinste Zahl ≥ 1 mit $f_k(z_k + l) \notin T$ ist. D.h. die Ausgabe befindet sich auf dem letzten Band rechts des Arbeitsfeldes.

- $\text{OPER} = \{f_l : f \in \bar{\Gamma} \cup \{R, L, S\} \ \& \ 1 \leq l \leq k\}$, wobei für $a \in \Gamma$

$$\bar{a}_l(f_1, z_1, \dots, f_k, z_k) = (f_1, z_1, \dots, f_{l-1}, z_{l-1}, (f_l)_{(a, z_l)}, z_l, f_{l+1}, z_{l+1}, \dots, f_k, z_k)$$

und für $B \in \{R, L, S\}$

$$B_l(f_1, z_1, \dots, f_k, z_k) = (f_1, z_1, \dots, f_{l-1}, z_{l-1}, f_l, z_l', f_{l+1}, z_{l+1}, \dots, f_k, z_k)$$

wobei

$$z_l' = \begin{cases} z_l - 1 & \text{falls } B = L \\ z_l & \text{falls } B = S \\ z_l + 1 & \text{falls } B = R \end{cases}$$

- $\text{TEST} = \{t_{a,l} : a \in \Gamma \ \& \ 1 \leq l \leq k\}$, wobei $t_{a,l}(f_1, z_1, \dots, f_k, z_k) = 1$ g.d.w. $f_l(z_l) = a$.

Hierbei sind B_l , f_{w_1, \dots, w_n} und $f_{(a,z)}$ wie in Definition 4.1 definiert.

Bei einer k -Band-TM wird also die Eingabe auf das 1. Band geschrieben und die Ausgabe dem k -ten Band entnommen. In jedem Schritt wird ein Band ausgewählt und auf diesem eine elementare Operation oder ein elementarer Test ausgeführt.

Bei Programmen mit bedingten Anweisungen führt man in einem Schritt für jedes Band erst einen Druckbefehl, dann eine Kopfbewegung aus, wobei die Auswahl dieser Operationen neben dem aktuellen Programmzustand von den Inschriften aller Arbeitsfelder abhängt.

Eine bedingte Anweisung für $TB_k(\Sigma, T, \Gamma, n)$ hat also die Gestalt

$$(i, b_1, \dots, b_k, b_1', B_1, \dots, b_k', B_k, j) \in \mathbb{N} \times \Gamma^k \times (\Gamma \times \text{Bew})^k \times \mathbb{N}$$

(für $\text{Bew} = \{L, S, R\}$) und bedeutet, dass, falls im Programmzustand i die Arbeitsfelder mit b_1, \dots, b_k beschriftet sind, dann wird für $1 \leq l \leq k$ auf dem l -ten Band zuerst das Arbeitsfeld mit b_l' neu beschriftet und dann die Kopfbewegung B_l ausgeführt sowie der neue Zustand j angenommen.

6.2 BEISPIEL. Wir geben eine 2-Band-TM $M = (B, P)$ an, die feststellt, ob das Eingabewort $w \in \Sigma_2^*$ ein Palindrom ist, d.h. ob $w = w^R$ gilt, wobei w^R das Spiegelwort von w ist ($\lambda^R = \lambda$, $(i_0 \dots i_n)^R = i_n \dots i_0$). Als Basismaschine dient uns hierzu $B = TB_2(\Sigma_2, \Sigma_2, \Sigma_2 \cup \{b\}, 1)$. Das Programm P bewirkt folgende Schritte: Zunächst wird das Spiegelwort w^R von w auf Band 2 geschrieben. Hierzu wird w auf Band 1 von links nach rechts durchlaufen und dabei gleichzeitig die gelesenen Buchstaben von rechts nach links auf Band 2 kopiert. Der Kopf auf Band 1 wird dann vor w zurückgesetzt. Schließlich werden w auf Band 1 und w^R auf Band 2 gleichzeitig von links nach rechts Buchstabe für Buchstabe verglichen. Findet man eine Abweichung, so gibt man 0 aus, sonst 1.

Formal sieht P wie folgt aus, wobei α und ω der Start- bzw. Stoppzustand von P sind.

$$\begin{aligned}
 &(\alpha, R_1, 0) \\
 &(0, t_{b,1}, 1, 6) \\
 &(1, t_{0,1}, 2, 3) \\
 &(2, 1_2, 4) \\
 &(3, 0_2, 4) \\
 &(4, R_1, 5) \\
 &(5, L_2, 0) \\
 &(6, L_1, 7) \\
 &(7, t_{b,1}, 6, 8) \\
 &(8, R_1, 9) \\
 &(9, R_2, 10) \\
 &(10, t_{b,1}, 11, 14) \\
 &(11, t_{0,1}, 12, 13) \\
 &(12, t_{0,2}, 8, 15) \\
 &(13, t_{1,2}, 8, 15) \\
 &(14, 1_2, 18) \\
 &(15, b_2, 16) \\
 &(16, L_2, 17) \\
 &(17, 0_2, 18) \\
 &(18, L_2, \omega)
 \end{aligned}$$

Bei Eingabe $w = i_0 \dots i_n$ ($n \geq -1$) haben die Zustände von P folgende Bedeutung. In Zustand 0 wird w gespiegelt auf Band 2 kopiert, wobei hierzu für jeden Buchstaben die Zustandsfolge 1–5 durchlaufen wird. Zustand 6 signalisiert, dass der Kopiervorgang beendet ist. Der Kopf auf dem 2. Band steht dann bereits vor w^R , während das Zurücksetzen des Kopfes auf dem 1. Band vor die Eingabe w im Zustand 8 abgeschlossen ist. In diesem Zustand beginnt der Vergleich von w auf Band 1 und w^R auf Band 2 Buchstabe für Buchstabe. Bei Gleichheit $w = w^R$ endet der Vergleich im Zustand 14, bei Ungleichheit $w \neq w^R$ im Zustand 15. Ausgehend von diesen Zuständen wird dann die Ausgabe 1 bzw. 0 auf Band 2 produziert.

Bei Verwendung bedingter Anweisungen erhält man folgendes Programm mit Startzustand 0 (wobei $i = 0, 1$): Hier wird im Zustand 1 w^R auf Band 2 geschrieben, im Zustand 2 der Kopf auf Band 1 vor die Eingabe w zurückgesetzt. Im Zustand 3 werden w auf Band 1 und w^R auf Band 2 verglichen. Zustand 4 signalisiert, dass eine Abweichung gefunden wurde, 5 ist der Stoppzustand.

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		1		L		5
4		b		i/b		b		S		0		L		5

Analysiert man die Rechenzeit der Maschine $M = (B, P)$ in vorstehendem Beispiel, so sieht man, dass diese linear in der Eingabelänge ist, d.h. $\text{time}_M(w) \in O(|w|)$. 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 (siehe Übungen). Dies erfordert je Vergleich $O(|w|)$ Schritte, da das Wort w 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. Wie der nächste Satz zeigt, sind jedoch Mehrband-TMs nicht prinzipiell mächtiger als 1-Band-TMs.

6.3 SATZ. (BANDREDUKTIONSSATZ) *Das Mehrband-Turingmaschinen-Konzept ist äquivalent zum (1-Band-)Turingmaschinen-Konzept. D.h. zu jeder k -Band-Turingmaschine $M = (B, P)$ mit $k \geq 2$ gibt es eine äquivalente 1-Band-TM $M' = (B', P')$.*

BEWEIS. Sei $M = (B, P)$ eine k -Band-TM ($k \geq 2$). Eine hierzu äquivalente 1-Band-TM $M' = (B', P')$ erhält man durch Schritt-für-Schritt-Simulation von P (auf einer geeigneten Basismaschine B'). D.h. jeder Schritt von M wird durch eine geeignete Schrittfolge von M' simuliert.

Hierzu wählt man das Bandalphabet von B' so, dass (intuitiv) das Band in $2k$ Spuren zerlegt wird, wobei zu jedem Band von B 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.

Das Programm P' besteht dann aus 3 Teilprogrammen $P' = P'_1 \cup P'_2 \cup P'_3$ mit folgenden Funktionen:

P'_1 : Überführung der Startkonfiguration in das (kodierte) Format der Mehrbandmaschine.

P'_2 : Schritt-für-Schritt-Simulation von P : Zur Simulation eines P -Schrittes läuft P'_2 vom linken Ende des relevanten (in Spuren aufgeteilten) Bandteils nach rechts, bis das Arbeitsfeld des Bandes erreicht ist, auf das sich die aktuelle P -Instruktion bezieht. P'_2 simuliert die Instruktion dort lokal, und läuft zum linken Ende des relevanten Bandteils zurück.

P'_3 : Überführung der kodierten Stoppkonfiguration der Mehrband-Maschine in die äquivalente Stoppkonfiguration der 1-Band-Maschine.

Im Folgenden beschreiben wir $M' = (B', P')$ formal, wobei wir uns zur Vereinfachung der Notation auf die Simulation einer 2-Band-TM $M = (B, P)$ beschränken.

Ist $B = TB_2(\Sigma, T, \Gamma, n)$, so ist $B' = TB(\Sigma, T, \Gamma', n)$ mit

$$\Gamma' = \Gamma \cup \{(\sigma_1, b_1, \sigma_2, b_2) : \sigma_1, \sigma_2 \in \{+, -\} \& b_1, b_2 \in \Gamma\} \cup \{[,]\}$$

wobei $(\sigma_1, b_1, \sigma_2, b_2)$ die Spuren von links nach rechts (statt von oben nach unten) beschreibt, d.h. σ_i angibt, ob es sich um das Arbeitsfeld von M auf Band i handelt, und b_i die Feldbeschriftung von Band i ist.

Das Programm P' geben wir als Programm aus bedingten Anweisungen an (dies genügt nach Lemma 5.1), wogegen wir bei P von einem üblichen Programm ausgehen. Ohne Einschränkung können wir annehmen, dass P genau einen Stoppzustand e besitzt und 0 der Startzustand ist.

Die Programmstücke P'_i ($i = 1, 2, 3$) von P' besitzen je eine Startzustand α_i und Stoppzustand ω_i . P entsteht aus diesen Programmteilen durch Umbenennen der Zustände, so dass $\omega_1 = \alpha_2$ und $\omega_2 = \alpha_3$ ist und die Zustandsmengen ansonsten disjunkt sind. Startzustand von P ist α_1 , einziger Stoppzustand ω_3 .

Das Programm P'_1 muss für ein Eingabewort $w = w(0) \dots w(m) \in \Sigma^*(m \geq -1)$ folgende Speichertransformation bewirken (wir schreiben hier anschaulich $(\sigma_1, b_1, \sigma_2, b_2)^T$ statt $(\sigma_1, b_1, \sigma_2, b_2)$):

$$\begin{array}{ccccccc} \dots bw(0) \dots w(n)b \dots & \xrightarrow{*} & \dots [& \begin{array}{ccc} + & - & - \\ b & w(0) & \dots & w(n) \end{array} &] \dots \\ \uparrow & & \uparrow & \begin{array}{ccc} + & - & - \\ b & b & b \end{array} & & & \end{array}$$

Dies wird realisiert durch ($\alpha_1 = 0, \omega_1 = 5$)

Z	\times	Γ'	\rightarrow	Γ'	\times	Bew	\times	Z
0		b		b		L		1
1		b		[R		2
2		b		$(+, b, +, b)$		R		3
3		$a \in \Sigma$		$(-, a, -, b)$		R		3
3		b]		L		4
4		$a \neq [$		a		L		4
4		[[S		5

Das Programm P'_2 besitzt für jede Instruktion $I \in P$ ein Programmstück P'_I zur Simulation von I . Wir bezeichnen die Zustände von P'_I hier nicht mit Zahlen, sondern geben ihre Funktion veranschaulichende Namen. Die Struktur von P'_I hängt vom Typ von I ab. Wir betrachten die typischen Fälle und überlassen die Ausarbeitung der analogen Fälle als Übung. Es ist $\alpha_2 = 0$ und $\omega_2 = e$.

1. $I = (i, (c)_1, j)$ ist eine Druckinstruktion (für Band 1):

Z	\times	Γ'	\rightarrow	Γ'	\times	Bew	\times	Z
i		[[R		$(i, \rightarrow AF1)$
$(i, \rightarrow AF1)$		$(-, b_1, \sigma_2, b_2)$		$(-, b_1, \sigma_2, b_2)$		R		$(i, \rightarrow AF1)$
$(i, \rightarrow AF1)$		$(+, b_1, \sigma_2, b_2)$		$(+, c, \sigma_2, b_2)$		L		$(j, [\leftarrow)$
$(j, [\leftarrow)$		$a \neq [$		$a \neq [$		L		$(j, [\leftarrow)$
$(j, [\leftarrow)$		[[S		j

2. $I = (i, L_2, j)$ ist eine Linksbewegung (für Band 2):

Z	$\times \Gamma'$	$\rightarrow \Gamma'$	$\times Bew$	$\times Z$
i	[[R	$(i, \rightarrow AF2)$
$(i, \rightarrow AF2)$	$(\sigma_1, b_1, -, b_2)$	$(\sigma_1, b_1, -, b_2)$	R	$(i, \rightarrow AF2)$
$(i, \rightarrow AF2)$	$(\sigma_1, b_1, +, b_2)$	$(\sigma_1, b_1, -, b_2)$	L	$(j, \leftarrow AF2)$
$(j, \leftarrow AF2)$	$(\sigma_1, b_1, -, b_2)$	$(\sigma_1, b_1, +, b_2)$	L	$(j, [\leftarrow)$
$(j, \leftarrow AF2)$	[$(-, b, +, b)$	L	$(j, [\downarrow)$
$(j, [\leftarrow)$	$a \neq [$	a	L	$(j, [\leftarrow)$
$(j, [\leftarrow)$	[[S	j
$(j, [\downarrow)$	b	[S	j

Ist hier $AF2$ auf dem 1. Feld des geklammerten Bandabschnittes, so muss man diesen um ein Feld nach links erweitern, d.h. $b[$ durch $[(-, b, +, b)$ ersetzen. (Entsprechend muss bei Simulation eines Rechtsbefehls bei Bedarf eine Erweiterung am rechten Ende vorgenommen werden.)

3. $I = (i, t_{a,1}, j, k)$ ist ein Test (für Band 1):

Z	$\times \Gamma'$	$\rightarrow \Gamma'$	$\times Bew$	$\times Z$
i	[[R	$(i, \rightarrow AF1)$
$(i, \rightarrow AF1)$	$(-, b_1, \sigma_2, b_2)$	$(-, b_1, \sigma_2, b_2)$	R	$(i, \rightarrow AF1)$
$(i, \rightarrow AF1)$	$(+, a, \sigma_2, b_2)$	$(+, a, \sigma_2, b_2)$	L	$(k, [\leftarrow)$
$(i, \rightarrow AF1)$	$(+, a', \sigma_2, b_2), a' \neq a$	$(+, a', \sigma_2, b_2)$	L	$(j, [\leftarrow)$
$(k, [\leftarrow)$	$c \neq [$	$c \neq [$	L	$(k, [\leftarrow)$
$(k, [\leftarrow)$	[[S	k
$(j, [\leftarrow)$	$c \neq [$	$c \neq [$	L	$(j, [\leftarrow)$
$(j, [\leftarrow)$	[[S	j

Das Programm P'_3 schließlich leistet für das Ausgabewort $v = v(0) \dots v(n) \in T^*$ ($n \geq -1$) folgende Speichertransformation:

$$\begin{array}{ccccccc}
 \dots & [& \dots & \sigma_{1,0} & \sigma_{1,1} & \dots & \sigma_{1,n+1} \\
 & & & b_{1,0} & b_{1,1} & \dots & b_{1,n+1} \\
 & & & + & - & & - \\
 & \uparrow & & b_2 & v(0) & & v(n) \\
 & & & & & & & \dots & \rightarrow^* & \dots & b v(0) & \dots & v(n) b & \dots \\
 & & & & & & & & & \uparrow & & & &
 \end{array}$$

Hierbei stehen die \dots links und rechts des Ausgabeteiles für beliebige Wörter, die eventuell von P'_3 verändert werden. Dies geschieht durch ($\alpha_3 = 0$ und $\omega_3 = stop$):

Z	\times	Γ'	\rightarrow	Γ'	\times	Bew	\times	Z
0		[b		R		0
0		$(\sigma_1, b_1, -, b_2)$		b		R		0
0		$(\sigma_1, b_1, +, b_2)$		[R		<i>aus</i>
<i>aus</i>		$(\sigma_1, b_1, \sigma_2, b_2), b_2 \in T$		b_2		R		<i>aus</i>
<i>aus</i>		$(\sigma_1, b_1, \sigma_2, b_2), b_2 \notin T$		b		L		[←
<i>aus</i>]		b		L		[←
[←		$a \neq [$		a		L		[←
[←		[b		S		<i>stop</i>

Wir verzichten auf die Verifikation, dass P' das Programm P korrekt simuliert. □

Vergleicht man die Rechenzeiten der Maschinen $M = (B, P)$ und $M' = (B', P')$ oben, so sieht man, dass (unter der Annahme, dass $\text{time}_M(w) \geq |w|$)

$$\text{time}_{M'}(w) \in O(\text{time}_M(w)^2).$$

Da M in jedem Schritt höchstens ein zusätzliches Feld beschriften kann, ist nämlich die Länge des relevanten Bandteiles [...] von M' bei Eingabe w stets durch $2 \cdot \text{time}_M(w) + 3$ beschränkt, weshalb jeder der $\text{time}_M(w)$ M -Schritte durch $O(\text{time}_M(w))$ M' -Schritte simuliert werden kann. Die Dauer der Initial- und Finalphase von M' ist linear in $|w|$, kann also vernachlässigt werden. Wir werden auf diese Beobachtung im Komplexitätstheorie-Teil der Vorlesung zurückkommen.

Weitere Speichervarianten bei Turingmaschinen sind:

- *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, wenn die nichttriviale Richtung im Fall der mehrdimensionalen TMs auch etwas mühselig ist. Bei den Halbband-TMs benutzt man für die nichttriviale Richtung folgende Idee: Um eine (Vollband-) Turingmaschine M durch eine Halbband-TM M' zu simulieren, benutzt man die 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.)

Ähnlich zeigt man die Äquivalenz von Turingmaschinen und 2-Stapel-Maschinen (oder – wie man meist sagt – 2-Kellermaschinen). Der Speicher einer k -Stapel-Maschine über dem Speicheralphabet Γ besteht aus k Stapeln (stacks) über Γ . Die Mächtigkeit von 1-Stapel-Maschinen werden wir im Teil über Formale Sprachen betrachten.

Eine weitere Variante des Speichers erhalten wir durch Normierung des Bandalphabets. Wir sagen, dass die Basismaschine $TB(\Sigma, T, \Gamma, n)$ *Bandalphabet-normiert* ist, wenn $\Gamma = \Sigma \cup T \cup \{b, 0, 1\}$, wobei 0, 1 – nicht aber b – in $\Sigma \cup T$ vorkommen dürfen. Diese Beschränkung des Bandalphabetes schränkt die Mächtigkeit des Konzeptes nicht ein.

6.4 SATZ. *Zu jeder Turingmaschine $M = (B, P)$ über der Basismaschine $TB(\Sigma, T, \Gamma, n)$ gibt es eine äquivalente Turingmaschine $M' = (B', P')$ über der Bandalphabet-normierten Basismaschine $B' = (\Sigma, T, \Gamma', n)$, wobei $\Gamma' = \Sigma \cup T \cup \{b, 0, 1\}$.*

BEWEISIDEE. Im Speicher von B' werden die Buchstaben des Bandalphabetes von Γ binärkodiert. D.h. für $\Gamma = \{a_1, \dots, a_n\}$ wird der Buchstabe a_i durch das Wort $0^i 1^{n-i}$ dargestellt. Buchstaben in S werden also durch „Blöcke“ von Buchstaben der Länge n dargestellt. \square