

DTM für $L = \{0^{2^n}\}, n \geq 0\}$

$Q = \{q_0, \dots, q_6\}, s = q_0, q_{\text{accept}} = q_5, q_{\text{reject}} = q_6, \Sigma = \{0\}, P = \{0, x, \sqcup, \triangleright\}$

δ	0	x	\sqcup	\triangleright
q_0	$(q_1, 0, R)$	(q_6, \sqcup, R)	(q_6, \sqcup, R)	(q_0, \triangleright, R)
q_1	(q_2, x, R)	(q_1, x, R)	(q_5, \sqcup, L)	(q_6, \triangleright, R)
q_2	$(q_3, 0, R)$	(q_2, x, R)	(q_4, \sqcup, L)	(q_6, \triangleright, R)
q_3	(q_2, x, R)	(q_3, x, R)	(q_6, \sqcup, R)	(q_6, \triangleright, R)
q_4	$(q_4, 0, L)$	(q_4, x, L)	(q_6, \sqcup, R)	(q_0, \triangleright, R)

Programmiertechniken: Zustand fungiert als „endlicher Speicher“



Beispiel: Wir wollen testen, ob bei Eingabe $w_1 \dots w_n \in \Sigma^+$ der Buchstabe w_1 in $w_2 \dots w_n$ vorkommt. Wir merken uns w_1 im Zustand. Sei $\Gamma = \Sigma \cup \{\triangleright, \sqcup\}$. $Q = (\{q_0\} \times \Sigma) \cup \{q_0, q_1, q_2\}$, Startzustand: q_0 , $q_{\text{accept}} = q_1$, $q_{\text{reject}} = q_2$

- 1) $\delta(q_0, \triangleright) = (q_0, \triangleright, R)$
- 2) $\delta(q_0, a) = ([q_0, a], a, R) \quad \forall a \in \Sigma$
- 3) $\delta([q_0, a], a) = (q_1, a, L) \quad \forall a \in \Sigma$
- 4) $\delta([q_0, a], b) = ([q_0, a], b, R) \quad \forall a, b \in \Sigma, a \neq b$
- 5) $\delta([q_0, a], \sqcup) = (q_2, \sqcup, L)$
- 6) $\delta([q_0, a], \triangleright) = (q_2, \triangleright, R)$

„Zeichen markieren“

Um $a \in \Gamma$ zu „markieren“, füge neuen Buchstaben \hat{a} zu Γ hinzu. \hat{a} steht für die markierte Version von a .

Beispiel: Animation aus <http://i10www.ira.uka.de/arbeiten/info3-animationen/Animationen>

Mehrband-Turingmaschinen

Eine k -Band Turingmaschine (k -DTM) hat nicht nur ein Band und einen Lesekopf, sondern k Bänder mit je einem Lesekopf. Die Übergangsfunktion ist dann von der Form

$$\delta : Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R, N, L\}^k.$$

Zu Beginn steht die Eingabe auf Band 1, auf allen anderen Bändern steht links das Startsymbol \triangleright gefolgt von Blanks. Die Arbeitsweise einer k -DTM ist wie die Arbeitsweise einer 1-Band-DTM definiert. Köpfe haben die Möglichkeit sich nicht zu bewegen (N).



Wird die Sprache $L \subseteq \Sigma^*$ von einer k-Band Turingmaschine M entschieden (akzeptiert), so gibt es auch eine 1-Band Turingmaschine, die L entscheidet (akzeptiert).

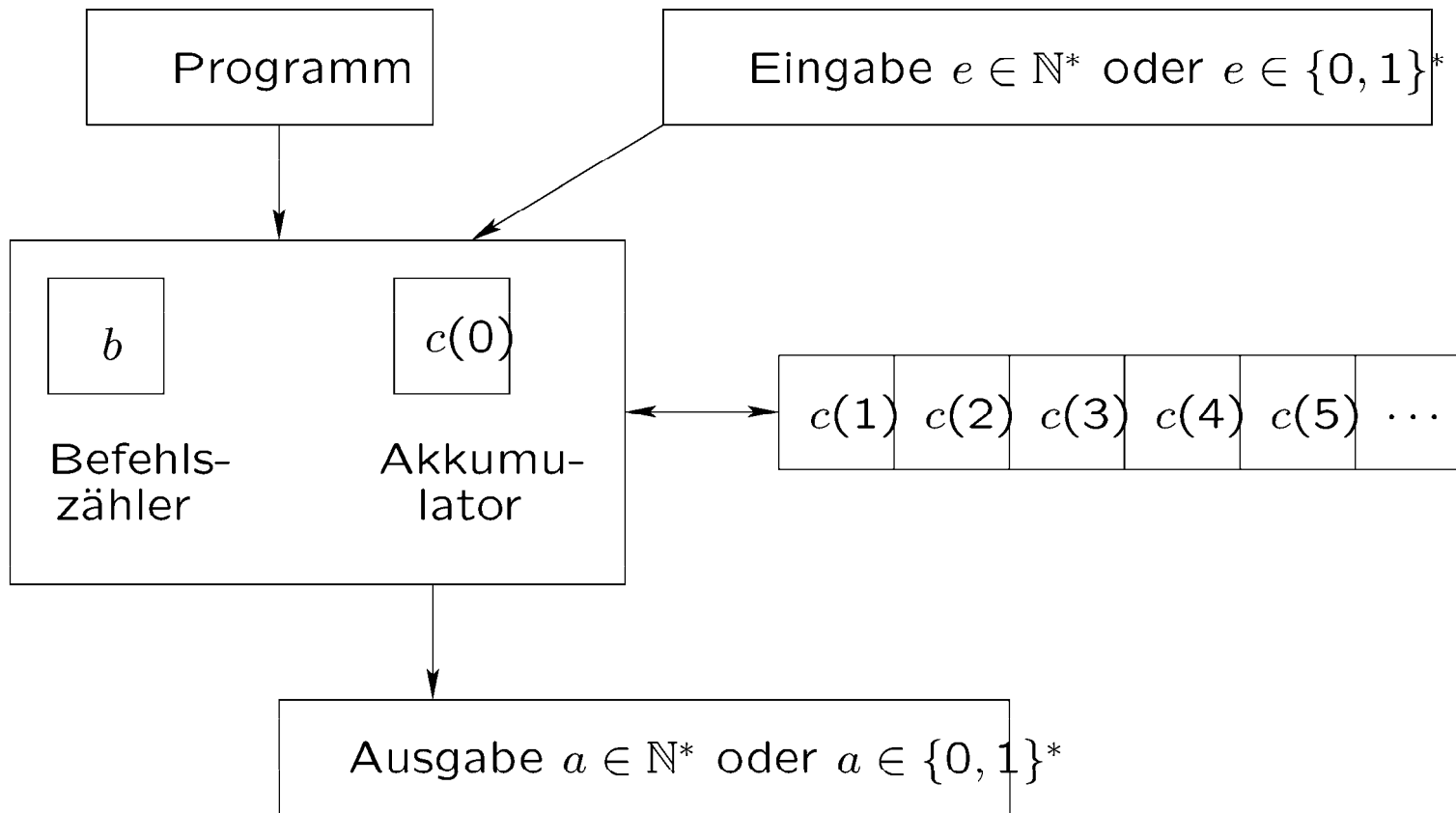
k -Band DTMs berechnen Funktionen

Eine k -Band DTM M berechnet die partielle Funktion $f : \Sigma^* \rightarrow (\Gamma \setminus \{\triangleright, \sqcup\})^*$ mit

- $f(x) = y$, falls nach dem Stoppen der Maschine M im Zustand q_{accept} der Inhalt des k -ten Bandes von M das Wort y ist. Hierbei werden das Startsymbol und alle \sqcup s des Bandes ignoriert.
- $f(x)$ ist nicht definiert, falls M bei der Eingabe x nicht im akzeptierenden Zustand hält.

M berechnet die **totale** Funktion $f : \mathbb{N}^r \rightarrow \mathbb{N}$, falls $\Sigma = \{0, 1, \#\}$ und M für jedes $a_1, \dots, a_r \in \mathbb{N}$ bei Eingabe $\text{bin}(a_1)\#\text{bin}(a_2)\#\dots\#\text{bin}(a_r)$ im Zustand q_{accept} hält und auf dem k -ten Band $\text{bin}(f(a_1, \dots, a_r))$ steht. Wiederum werden das Startsymbol und alle \sqcup s ignoriert.

Der Beweis von Satz 2.2.2 liefert nun auch, dass es für jede Funktion f , die durch eine k -Band DTM Berechnet wird, auch eine 1-Band DTM gibt, die f berechnet.



Schematische Darstellung einer RAM

Ein-/Ausgabe	
- read	$c(0) := \text{head}(e), e := \text{tail}(e), b := b + 1$ falls $e \neq \varepsilon$
- write	$c(0) := EOF, b := b + 1$ falls $e = \varepsilon$
	$a := a \cdot c(0)$
Arithmetik	
- Add i	$c(0) := c(0) + c(i), b := b + 1$
- Sub i	$c(0) := c(0) - c(i), b := b + 1$ falls $c(i) < c(0)$
	$c(0) := 0, b := b + 1$ sonst
- Mult i	$c(0) := c(0) \cdot c(i), b := b + 1$
- Div i	$c(0) := \lfloor \frac{c(0)}{c(i)} \rfloor, b := b + 1$ falls $c(i) \neq 0$
	$c(0) := \text{Fehlerhaft}, b := b + 1$ sonst
- c Add i, c Sub i, ...	$c(0) := c(0) + i, c(0) := c(0) - i, \dots$
(Benutzung von Konstanten)	

Sprünge

- Goto j $b := j$
 - If $c(0) R i$ then goto j $b := \begin{cases} j & \text{falls } c(0) R i \\ b + 1 & \text{sonst} \end{cases}, R \in \{<, >, =, \leq, \geq\}$
 - End Programm hält
-
-

Speicherzugriffe

Direkt

- Load i $c(0) := c(i), b := b + 1$
- Store i $c(i) := c(0), b := b + 1$

Indirekt

- Iload i $c(0) := c(c(i)), b := b + 1$
 - Istore i $c(c(i)) := c(0), b := b + 1$
-

Satz 2.3.1: Jede RAM kann durch DTM simuliert werden

Satz 2.3.2: Jede DTM kann durch RAM simuliert werden

Intuitiv: RAM-“Programmiersprache“ ist einfache, aber
„vollständige“ Assembler-Sprache, also:

Church'sche These (1936). Die im intuitiven Sinne berechenbaren Funktionen sind genau die, die durch Turingmaschinen berechenbar sind.