Berechenbarkeit und Komplexität: Einführung

Prof. Dr. Berthold Vöcking
Lehrstuhl Informatik 1
Algorithmen und Komplexität

30. Oktober 2007

Was ist ein *Problem*?

Informelle Umschreibung des Begriffes *Problem*:

Für gegebene Eingaben soll der Computer ...
... bestimmte Ausgaben produzieren.

Wir benötigen eine präzisere Definition ...

Alphabete und Wörter

- Ein- und Ausgaben sind Wörter über einem Alphabet Σ.
- Typischerweise $\Sigma = \{0, 1\}$.
- Σ^k ist die Menge aller Wörter der Länge k, z.B.

$$\{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

- Das sogenannte *leere Wort*, also das Wort der Länge 0, bezeichnen wir mit ϵ, d.h. $\Sigma^0 = \{\epsilon\}$.
- $\Sigma^* = \bigcup_{k \in \mathbb{N}_0} \Sigma^k$ ist der sogenannte *Kleenesche Abschluss* von Σ und enthält alle Wörter über Σ, die wir z.B. der Länge nach aufzählen können

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots$$

Problem als Relation

- Im Allgemeinen entspricht ein Problem einer *Relation* $R \subseteq \Sigma^* \times \Sigma^*$.
- Ein Paar (x, y) liegt in R, wenn y eine zulässige Ausgabe zur Eingabe x ist.

Beispiel: Primfaktorbestimmung

Zu einer natürlichen Zahl $q \geq 2$ suchen wir einen Primfaktor.

Wir einigen uns darauf Zahlen binär zu kodieren. Die Binärkodierung einer natürlichen Zahl i bezeichnen wir mit $bin(i)$.

Die entsprechende Relation ist

$$R = \{(x, y) \in \{0,1\}^* \times \{0,1\}^* \mid x = bin(q), y = bin(p), \\ p, q \in \mathbb{N}, q \geq 2, p \text{ prim}, p \text{ teilt } q\} \ .$$

Problem als Funktion

- Häufig gibt es zu jeder Eingabe eine eindeutige Ausgabe.
- Dann können wir das Problem als Funktion $f : \Sigma^* \rightarrow \Sigma^*$ beschreiben.
- Die zur Eingabe $x \in \Sigma^*$ gesuchte Ausgabe ist $f(x) \in \Sigma^*$.

Beispiel: Multiplikation

Zu zwei natürlichen Zahlen $i_1, i_2 \in \mathbb{N}$ suchen wir das Produkt.

Um die Zahlen i_1 und i_2 in der Eingabe voneinander trennen zu können, erweitern wir das Alphabet um ein Trennsymbol #, d.h. $\Sigma = \{0, 1, \#\}$.

Die entsprechende Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist

$$f(bin(i_1)\#bin(i_2)) \;=\; bin(i_1 \cdot i_2) \;.$$

Entscheidungsprobleme als Sprachen

- Viele Probleme lassen sich als Ja-Nein-Fragen formulieren.
- Derartige *Entscheidungsprobleme* sind von der Form $f : \Sigma^* \rightarrow \{0, 1\}$, wobei wir 0 als „Nein“ und 1 als „Ja“ interpretieren.
- Sei $L = f^{-1}(1) \subseteq \Sigma^*$ die Menge derjenigen Eingaben, die mit „Ja“ beantwortet werden.
- L ist eine Teilmenge der Wörter über dem Alphabet Σ. Eine solche Teilmenge wird allgemein als *Sprache* bezeichet.

Entscheidungsprobleme als Sprachen

Beispiel: Graphzusammenhang

Problemstellung: Für einen gegebenen Graphen G soll bestimmt werden, ob G zusammenhängend ist.

Der Graph G liege dabei in einer geeignete Kodierung $code(G) \in \Sigma^*$ vor, z.B. als binär kodierte Adjazenzmatrix.

Die zu diesem Entscheidungsproblem gehörende Sprache ist

$$L = \{ w \in \Sigma^* \mid \exists \text{ Graph } G\text{: } w = code(G) \text{ und } G \text{ ist zusammenhängend} \} .$$

Zentrale Fragestellung

Welche Funktionen sind durch einen Computer berechenbar?

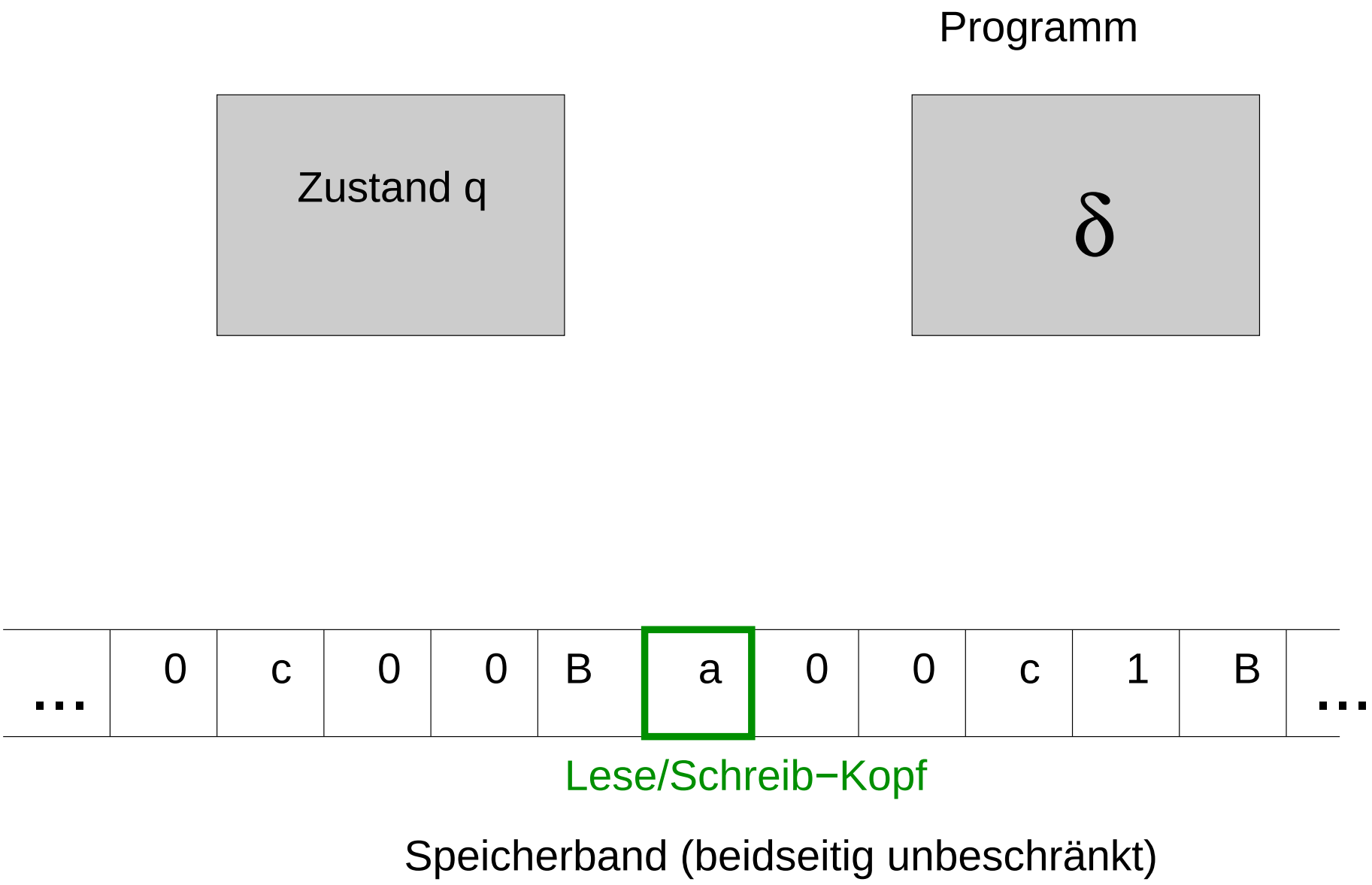
bzw.

Welche Sprachen kann eine Computer entscheiden?

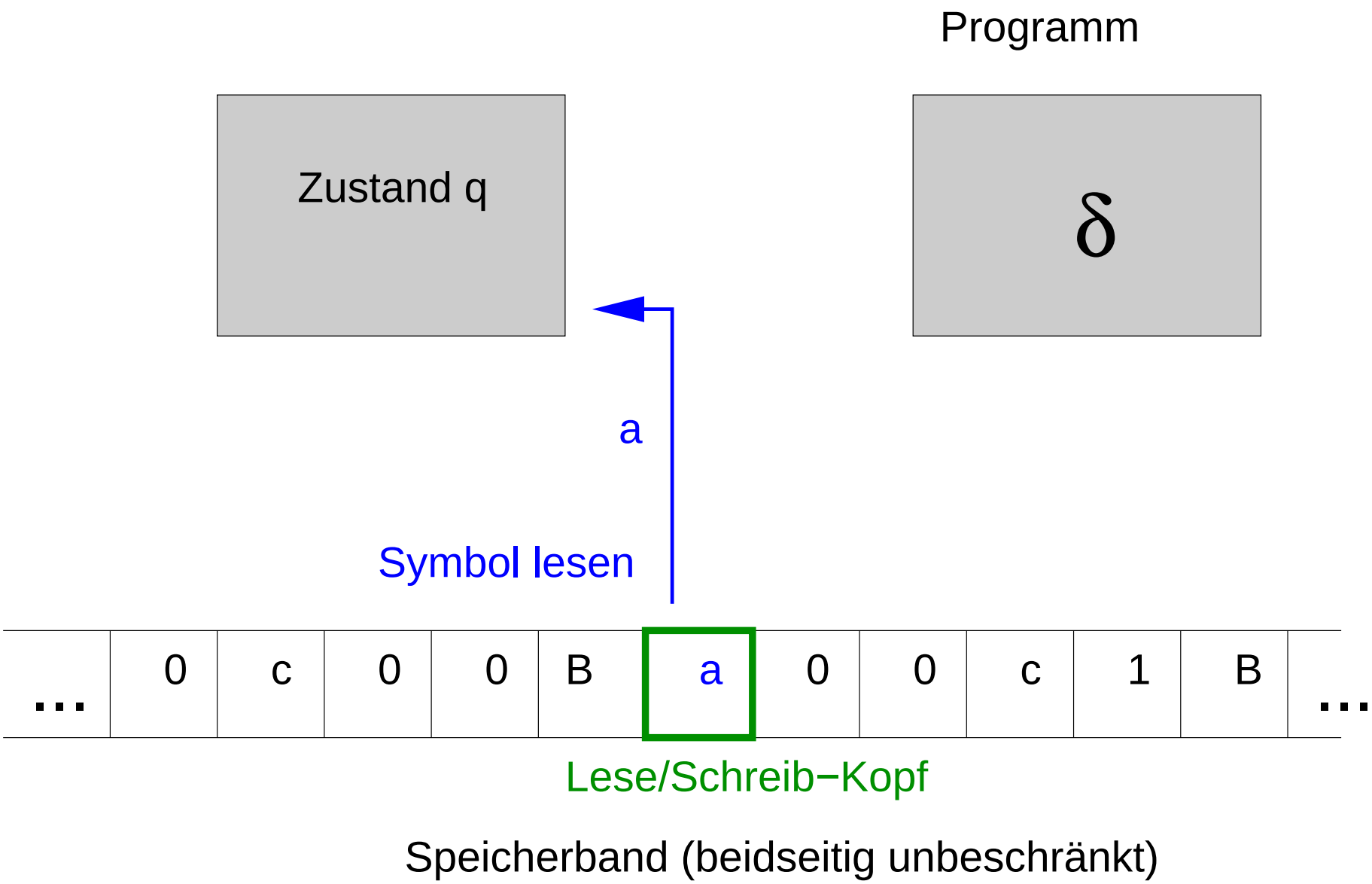
Um diese Fragen in einem mathematisch exakten Sinne klären zu können, müssen wir nöch klären was eigentlich ein Computer ist.

Wir benötigen wir ein mathematisches Rechnermodell.

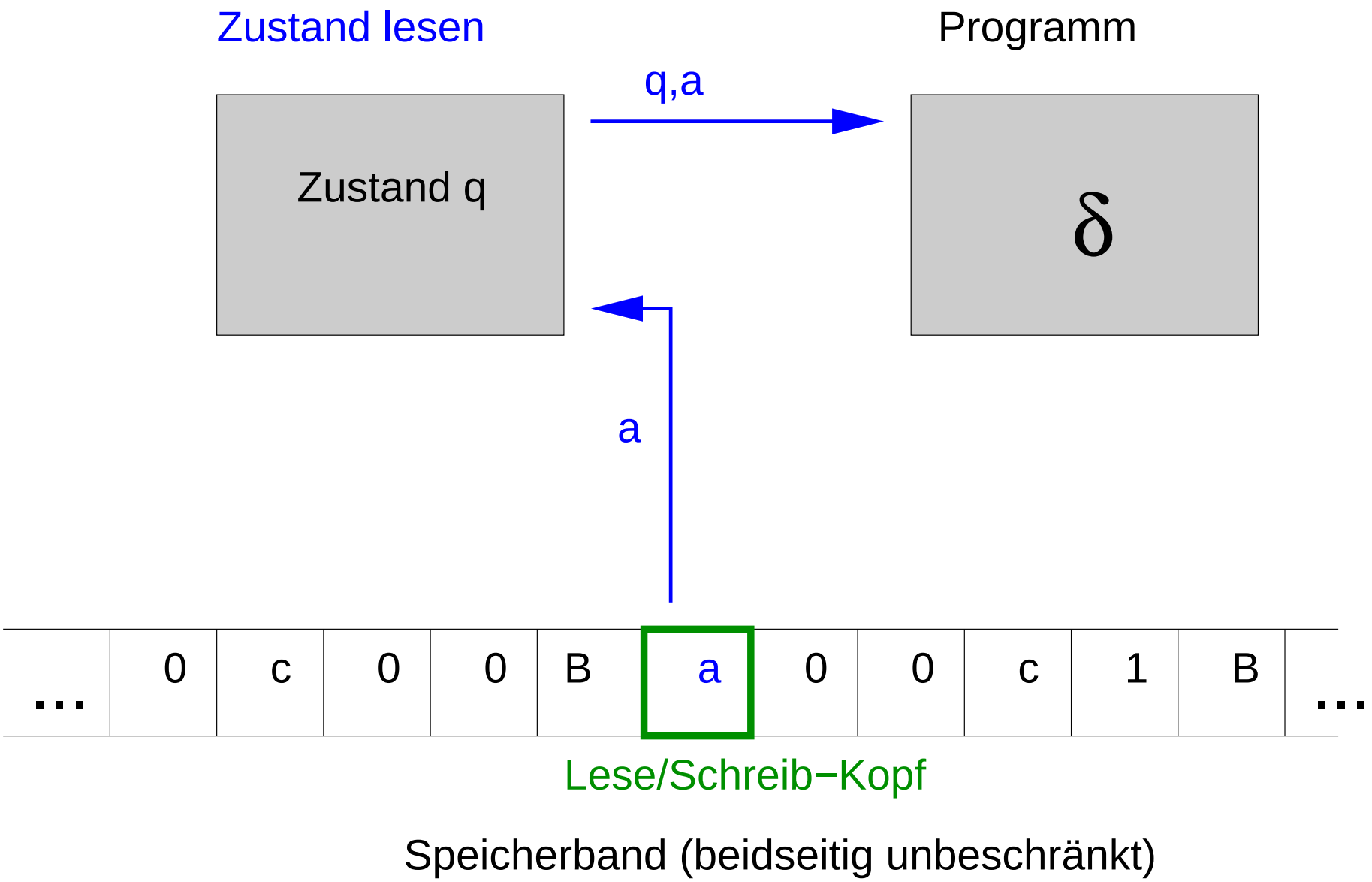
Deterministische Turingmaschinen (TM bzw. DTM)



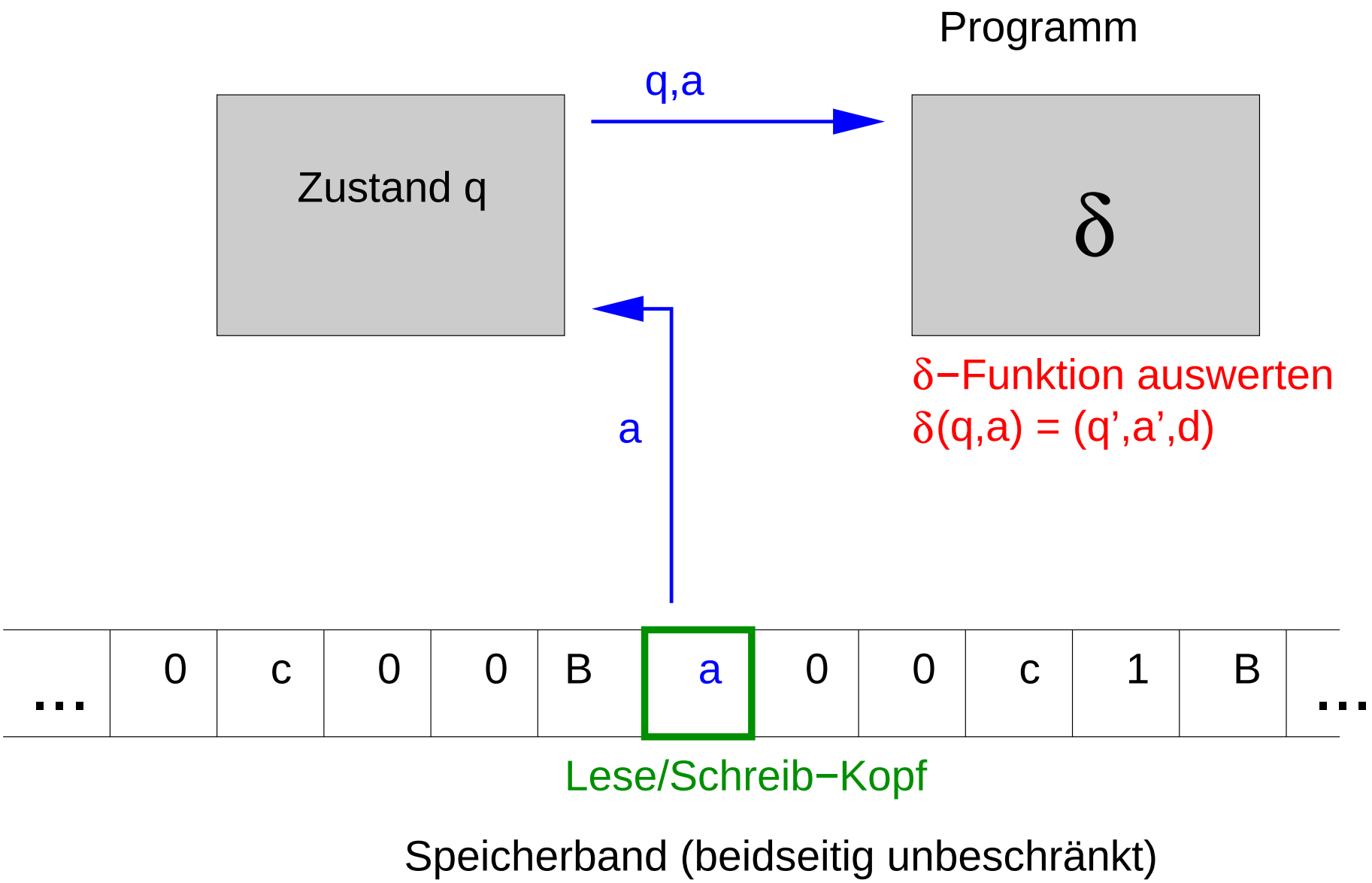
Deterministische Turingmaschinen (TM bzw. DTM)



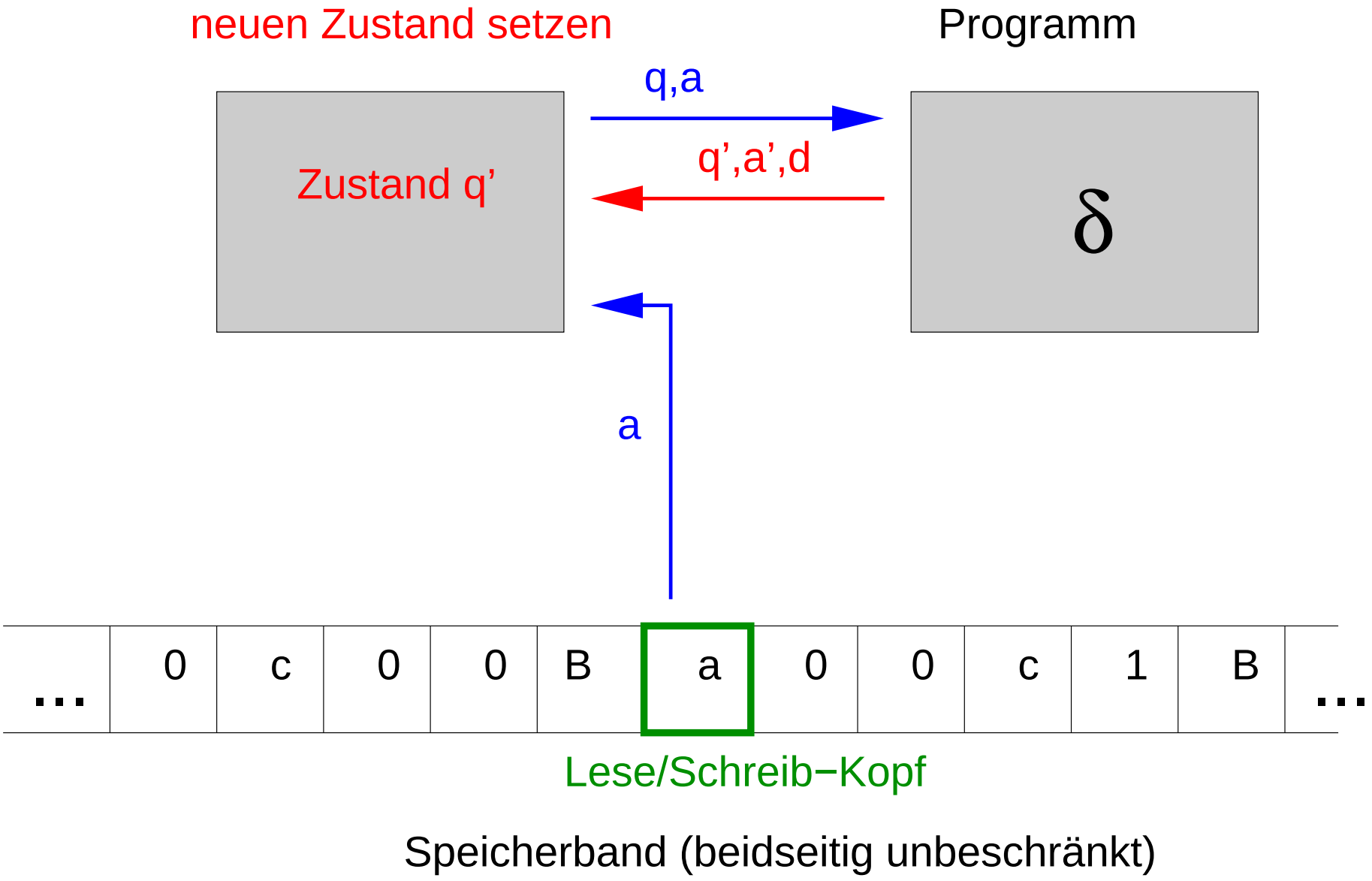
Deterministische Turingmaschinen (TM bzw. DTM)



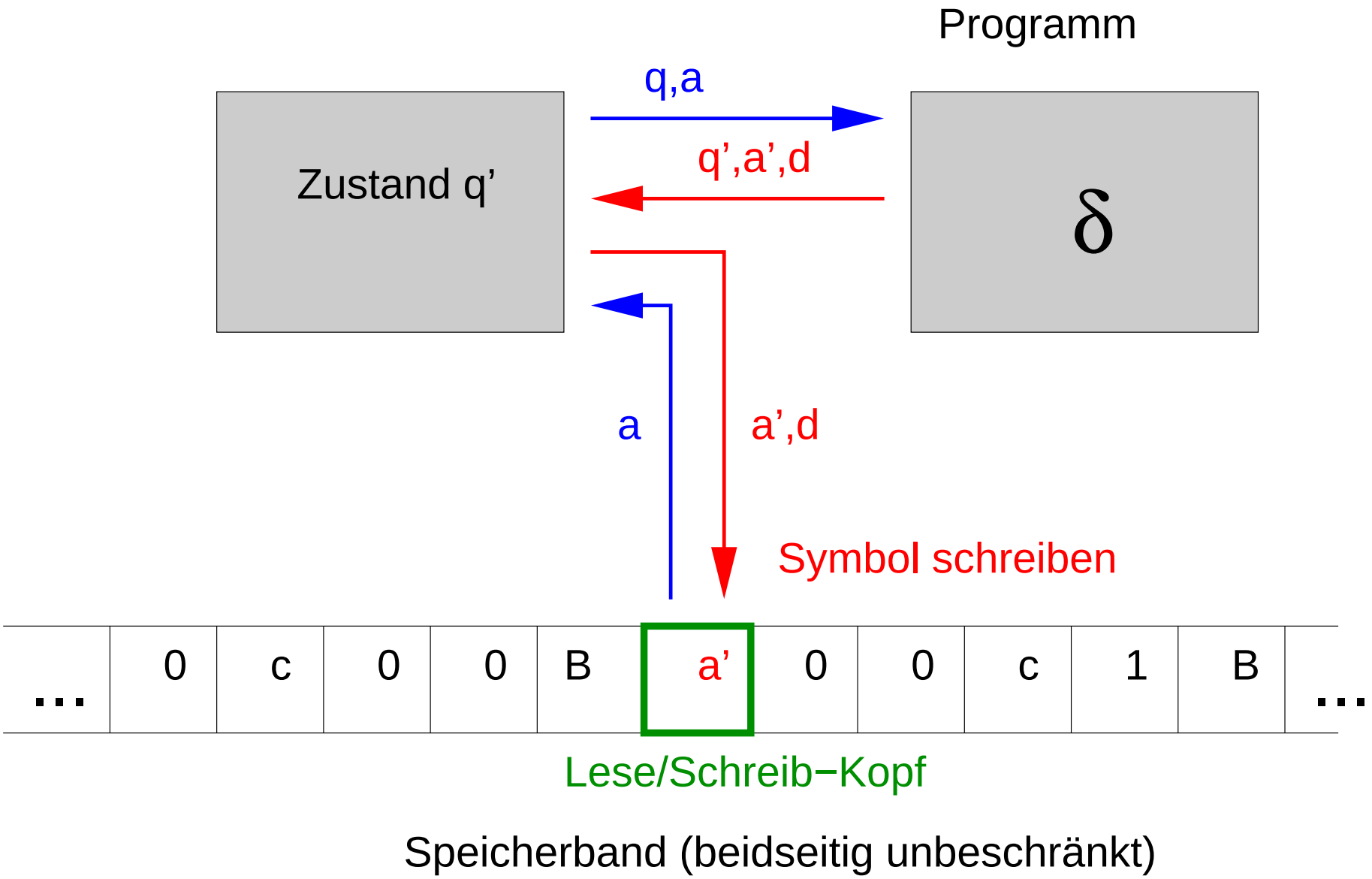
Deterministische Turingmaschinen (TM bzw. DTM)



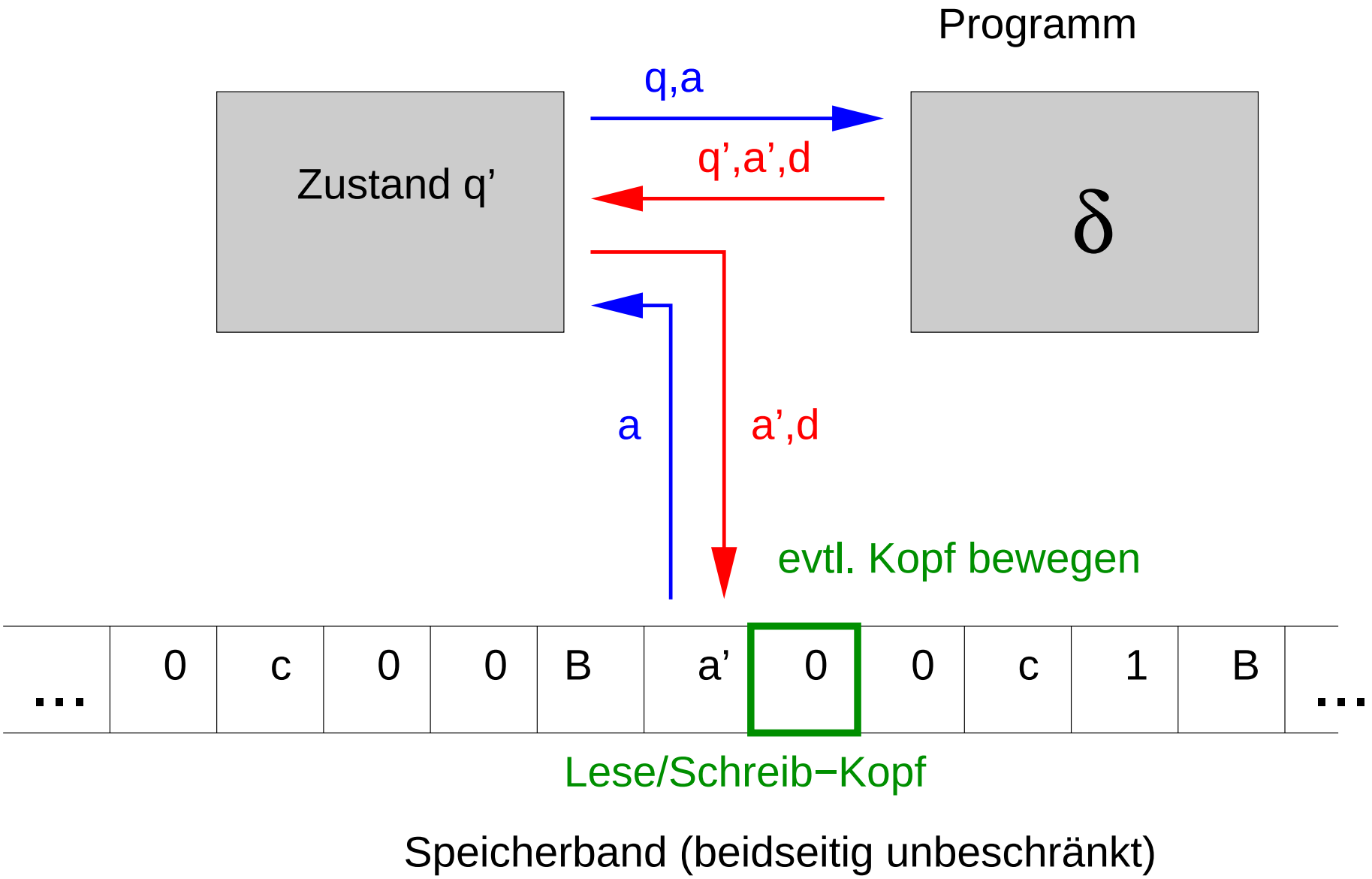
Deterministische Turingmaschinen (TM bzw. DTM)



Deterministische Turingmaschinen (TM bzw. DTM)



Deterministische Turingmaschinen (TM bzw. DTM)



Komponenten der TM

- Q, die endliche Zustandsmenge
- Σ, das endliche Eingabealphabet
- $\Gamma \supset \Sigma$, das endliche Bandalphabet
- $B \in \Gamma \setminus \Sigma$, das Leerzeichen (Blank)
- $q_0 \in Q$, der Anfangszustand
- $\bar{q} \in Q$, der Endzustand
- $\delta : (Q \setminus \bar{q}) \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$, die Zustandsüberführungsfunktion

Eine TM ist definiert durch das 7-Tupel $(Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$.

Funktionsweise der TM

Ausgangssituation

- auf dem Band steht die Eingabe $w \in \Sigma^*$ eingerahmt von Blanks
- der initiale Zustand ist q_0
- der Kopf steht über dem ersten Symbol von w

Nummerierung der Zellen des Bandes

- die initiale Kopfposition wird als Position 0 bezeichnet
- bewegt sich der Kopf einen Schritt „nach rechts“ erhöht sich die Position um 1
- bewegt sich der Kopf um einen Schritt „nach links“ erniedrigt sich die Position um 1

Funktionsweise der TM

Durchführung eines Rechenschrittes

- $a \in \Gamma$ bezeichne das gelesene Symbol
- $q \in Q \setminus \bar{q}$ bezeichne den aktuellen Zustand
- es sei $\delta(q, a) = (q', a', d)$, für $q' \in Q, a' \in \Gamma, d \in \{R, L, N\}$
- dann wird der Zustand auf q' gesetzt
- an der Kopfposition wird das Symbol a' geschrieben
- der Kopf

$$\text{bewegt sich} \begin{cases} \text{um eine Position nach rechts} & \text{falls } d = R \\ \text{um eine Position nach links} & \text{falls } d = L \\ \text{nicht} & \text{falls } d = N \end{cases}$$

Funktionsweise der TM

Ende der Rechnung

- die TM stoppt, wenn sie den Endzustand $\bar{q}$ erreicht
- das Ausgabewort $w' \in \Sigma^*$ kann dann vom Band abgelesen werden: w' beginnt an der Kopfposition und endet unmittelbar vor dem ersten Symbol aus $\Gamma \setminus \Sigma$
- *Spezialfall:* wenn wir es mit Entscheidungsproblemen zu tun haben, wird die Antwort wie folgt als JA oder NEIN interpretiert:
  - die TM *akzeptiert* das Eingabewort, wenn sie terminiert und das Ausgabewort mit einer 1 beginnt
  - die TM *verwirft* das Eingabewort, wenn sie terminiert und das Ausgabewort nicht mit einer 1 beginnt
- Beachte, es gibt die Möglichkeit, dass die Rechnung nicht terminiert.

Funktionsweise der TM

Bemerkungen

- Beachte, es gibt die Möglichkeit, dass die TM den Endzustand niemals erreicht. Wir sagen dann, die *Rechnung terminiert nicht*.
- Laufzeit = Anzahl der Zustandsübergänge bis zur Terminierung
- Speicherplatz = Anzahl der Bandzellen, die während der Rechnung besucht werden

Funktionsweise der TM am Beispiel

Sei $L = \{w1 | w \in \{0,1\}^*\}$.

L wird *entschieden* durch die TM $M = (Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$ mit

- $Q = \{q_0, q_1, \bar{q}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B\}$
- δ gemäß Tabelle

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

„accept“ steht als Abkürzung für $\bar{q}, 1, N$.
„reject“ steht als Abkürzung für $\bar{q}, 0, N$.

Die Übergangsfunktion entspricht dem *Programm der TM*.

Funktionsweise der TM am Beispiel

Beschreibung des Programms als Tabelle:

δ	0	1	B
q_0	(q_0, B, R)	(q_1, B, R)	reject
q_1	(q_0, B, R)	(q_1, B, R)	accept

Verbale Beschreibung des Programms:

- Solange ein Symbol aus $\{0, 1\}$ gelesen wird
  - überschreibe das Symbol mit B,
  - bewege den Kopf nach rechts, und
  - gehe in den Zustand q_0, wenn das Symbol eine 0 war, sonst in den Zustand q_1
- Sobald ein Blank gelesen wird, so
  - akzeptiere die Eingabe, falls der aktuelle Zustand q_1 ist, und
  - verwirf die Eingabe ansonsten.

Definition des Begriffes *berechenbar*

Definition

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heißt *rekursiv (berechenbar)*, wenn es eine TM gibt, die aus der Eingabe x den Funktionswert $f(x)$ berechnet.

Definition

Eine Sprache $L \subseteq \Sigma^*$ heißt *rekursiv (entscheidbar)*, wenn es eine TM gibt, die auf allen Eingaben stoppt und die Eingabe w genau dann akzeptiert, wenn $w \in L$ ist.

Programmierung der TM am Beispiel

Beispiel: Wir entwickeln ein TM für die Sprache

$$L = \{0^n 1^n | n \geq 1\} \ .$$

Sei $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, B\}$, $Q = \{q_0, \ldots, q_6, \bar{q}\}$.

Unsere TM arbeitet in zwei Phasen:

- **Phase 1:** Teste, ob das Eingabewort von der Form $0^i 1^j$ für $i \geq 0$ und $j \geq 1$ ist.
- **Phase 2:** Test, ob $i = j$ gilt.

Phase 1 verwendet q_0 und q_1 und wechselt bei Erfolg zu q_2.
Phase 2 verwendet q_2 und q_6 und akzeptiert bei Erfolg.

Programmierung der TM am Beispiel - Phase 1

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	reject
q_1	reject	$(q_1, 1, R)$	(q_2, B, L)

q_0 : Laufe von links nach rechts über die Eingabe bis ein Zeichen ungleich 0 gefunden wird.

- Falls dieses Zeichen eine 1 ist, gehe über in Zustand q_1.
- Sonst ist dieses Zeichen ein Blank. Verwirf die Eingabe.

q_1 : Gehe weiter nach rechts bis zum ersten Zeichen ungleich 1.

- Falls dieses Zeichen eine 0 ist, verwirf die Eingabe.
- Sonst ist das gefundene Zeichen ein Blank. Bewege den Kopf um eine Position nach links auf die letzte gelesene 1. Wechsel in den Zustand q_2, Phase 2 beginnt.

Programmierung der TM am Beispiel - Phase 2

δ	0	1	B
q_2	reject	(q_3, B, L)	reject
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_4, B, R)
q_4	(q_5, B, R)	reject	reject
q_5	$(q_6, 0, R)$	$(q_6, 1, R)$	accept
q_6	$(q_6, 0, R)$	$(q_6, 1, R)$	(q_2, B, L)

q_2 : Kopf steht auf dem letzten Nichtblank. Falls dieses Zeichen eine 1 ist, so lösche es, gehe nach links, und wechsel in den Zustand q_3. Sonst verwirf die Eingabe.

q_3 : Bewege den Kopf auf das erste Nichtblank. Dann q_4.

q_4 : Falls das gelesene Zeichen eine 0 ist, ersetze es durch ein Blank und gehe nach q_5, sonst verwirf die Eingabe.

q_5 : Wir haben jetzt die linkeste 0 und die rechteste 1 gelöscht. Falls Restwort leer, dann q_7 (akzeptiere), sonst q_6.

q_6 : Laufe wieder zum letzten Nichtblank und starte erneut in q_2.

Konfigurationen und (direkte) Nachfolgekonfigurationen

Definition

i) Eine *Konfiguration* einer TM ist ein String $\alpha q \beta$, für $q \in Q$ und $\alpha, \beta \in \Gamma^*$. Bedeutung: auf dem Band steht $\alpha\beta$ eingerahmt von Blanks, der Zustand ist q, und der Kopf steht unter dem ersten Zeichen von β.

ii) $\alpha' q' \beta'$ ist *direkte Nachfolgekonfiguration* von $\alpha q \beta$, falls $\alpha' q' \beta'$ in einem Rechenschritt aus $\alpha q \beta$ ensteht. Wir schreiben $\alpha q \beta \vdash \alpha' q' \beta'$.

iii) $\alpha'' q'' \beta''$ ist *Nachfolgekonfiguration* von $\alpha q \beta$, falls $\alpha'' q'' \beta''$ in endlich vielen Rechenschritten aus $\alpha q \beta$ ensteht. Wir schreiben $\alpha q \beta \vdash^* \alpha'' q'' \beta''$.

Bemerkung: insbesondere gilt $\alpha q \beta \vdash^* \alpha q \beta$.

Beispiel zum Umgang mit Konfigurationen

Die für die Sprache $L = \{0^n 1^n | n \geq 1\}$ beschriebene TM liefert in Phase 1 auf die Eingabe 0011 die folgende Konfigurationsfolge.

Phase 1:

$$q_0 0011 \vdash 0 q_0 011 \vdash 00 q_0 11 \vdash 001 q_1 1 \vdash 0011 q_1 B \vdash 001 q_2 1$$

Beobachtung: abgesehen von Blanks am Anfang und Ende des Strings sind die Konfigurationsbeschreibungen eindeutig.

Trick 1: Speicher im Zustandsraum

Für beliebiges festes $k \in \mathbb{N}$, können wir k Zeichen unseres Bandalphabets im Zustand abspeichern, indem wir den Zustandsraum um den Faktor $|\Gamma|^k$ vergrößern, d.h. wir setzen

$$Q_{neu} := Q \times \Gamma^k \quad .$$

Trick 2: Mehrspurmaschinen

Bei einer k-spurigen TM handelt es sich um eine TM, bei der das Band in k sogenannte Spuren eingeteilt ist, d.h. in jeder Bandzelle stehen k Zeichen, die der Kopf gleichzeitig einlesen kann. Das können wir erreichen, indem wir das Bandalphabet um k-dimensionale Vektoren erweitern, z.B.

$$\Gamma_{neu} := \Sigma \cup \Gamma^k \quad .$$

Beispiel: Addition mittels 3-spuriger TM

Die Verwendung einer mehrspurigen TM erlaubt es häufig, Algorithmen einfacher zu beschreiben.

Wir verdeutlichen dies am Beispiel der Addition. Aus der Eingabe $bin(i_1)\#bin(i_2)$ für $i_1, i_2 \in \mathbb{N}$ soll $bin(i_1 + i_2)$ bererechnet werden. Wir verwenden eine 3-spurige TM mit den Alphabeten $\Sigma = \{0, 1, \#\}$ und

$$\Gamma = \left\{ 0, 1, \#, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, B \right\} .$$

Beispiel: Addition mittels 3-spuriger TM

- *Schritt 1:* Transformation in Spurendarstellung: Schiebe die Eingabe so zusammen, dass die Binärkodierungen von i_1 und i_2 in der ersten und zweiten Spur rechtsbündig übereinander stehen. Aus der Eingabe 0011#0110 wird beispielsweise

$$B^* \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} B^* \ .$$

- *Schritt 2:* Addition nach der Schulmethode indem der Kopf das Band von rechts nach links durchläuft. Überträge werden im Zustand gespeichert. Als Ergebnis auf Spur 3 ergibt sich

$$B^* \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} B^* \ .$$

- *Schritt 3:* Rücktransformation von Spur 3 ins Einspur-Format: Ausgabe 1001.

Techniken zur Programmierung von TMen

Standardtechniken aus der Programmierung können auch auf TMen implementiert werden.

- *Schleifen* haben wir bereits in Beispiel 2.1.3 gesehen.
- *Variablen* können realisiert werden, indem wir pro Variable eine Spur reservieren.
- *Felder (Arrays)* können ebenfalls auf einer Spur abgespeichert werden.
- *Unterprogramme* können implementiert werden indem wir eine Spur des Bandes als Prozedurstack verwenden.

Basierend auf diesen Techniken können wir auch auf bekannte Algorithmen z.B. zum Sortieren von Daten zurückgreifen.

Turingmaschinen mit mehreren Bändern

k-Band TM

Eine k-Band-TM ist eine Verallgemeinerung der Turingmaschine und verfügt über k Arbeitsbänder mit jeweils einen unabhängigen Kopf. Die Zustandsübergangfunktion ist entsprechend von der Form

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k \quad .$$

Band 1 fungiert als Ein-/Ausgabeband wie bei der (1-Band) TM. Die Bänder $2, \ldots, k$ sind initial mit B^* beschrieben.

Simulation k-Band TM durch 1-Band TM

Satz:

Eine k-Band TM M, die mit Rechenzeit $t(n)$ und Platz $s(n)$ auskommt, kann von einer (1-Band) TM M' mit Zeitbedarf $O(t^2(n))$ und Platzbedarf $O(s(n))$ simuliert werden.

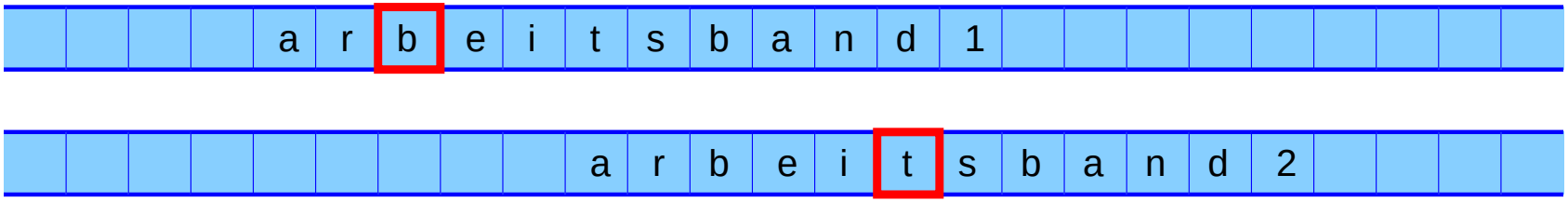
Beweis: Die TM M' verwendet $2k$ Spuren. Nach Simulation des t-ten Schrittes für $0 \leq t \leq t(n)$ gilt

- Die ungeraden Spuren $1, 3, \ldots, 2k-1$ enthalten den Inhalt der Bänder $1, \ldots, k$ von M.
- Auf den geraden Spuren $1, 2, \ldots, 2k$ sind die Kopfposition auf diesen Bändern mit dem Zeichen $\#$ markiert.

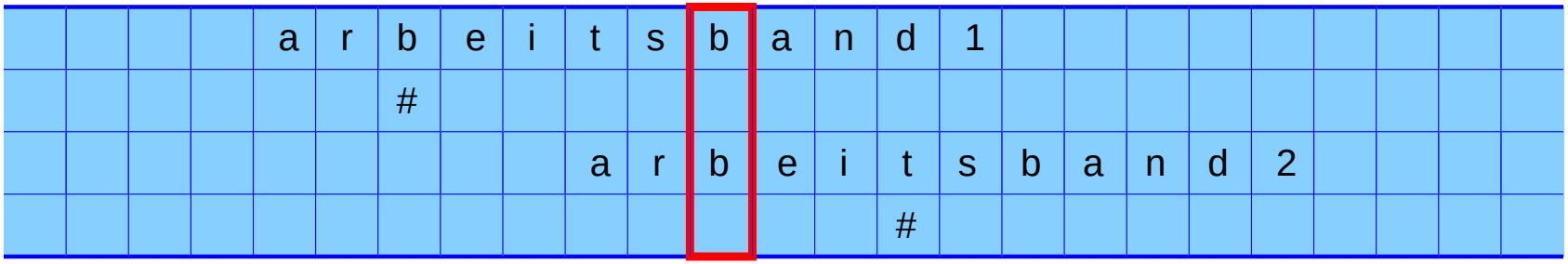
Diese Initialisierung der Spuren ist in Zeit $O(1)$ möglich.

Simulation k-Band TM durch 1-Band-TM – Illustration

simulierte 2−Band−TM M



simulierende 4−spurige TM M' (zu Beginn eines Simulationsschrittes)



Simulation k-Band TM durch 1-Band-TM – Beweis

Jeder Rechenschritt von M wird durch M' wie folgt simuliert.

- Am Anfang stehe der Kopf von M' auf dem linkesten # und M' kenne den Zustand von M.
- Der Kopf von M' läuft nach rechts bis zum rechtesten #, wobei die k Zeichen an den mit # makierten Spurpositionen im Zustand abgespeichert werden.
- Am rechtesten #-Zeichen angekommen kann M' die Übergangsfunktion von M auswerten und kennt den neuen Zustand von M sowie die erforderlichen Übergänge auf den k Bändern.
- Nun läuft der Kopf von M' zurück, verändert dabei die Bandinschriften an den mit # markierten Stellen und verschiebt, falls erforderlich, auch die #-Markierungen um eine Position nach links oder rechts.

Simulation k-Band TM durch 1-Band-TM – Beweis

Laufzeitanalyse:

Wieviele Bandpositionen können zwischen den linkesten und dem rechtesten # liegen?

Nach t Schritten können diese Markierungen höchstens $2t$ Positionen auseinanderliegen.

Also ist der Abstand zwischen diesen Zeichen und somit auch die Laufzeit zur Simulation eines Schrittes durch $O(t(n))$ beschränkt.

Insgesamt ergibt das zur Simulation von $t(n)$ Schritten eine Laufzeitschranke von $O(t(n)^2)$. □

Special versus General Purpose Rechner

- Bisher haben wir für jedes Problem eine eigene TM entworfen, einen *special purpose* Rechner.
- Real existierende Maschinen sind jedoch programmierbare *general purpose* Rechner.
- Wir konstruieren jetzt eine programmierbare Variante der TM, die sogenannte *universelle TM*.

Verhaltensbeschreibung der universellen TM

- Das Programm der universellen TM U ist die Kodierung einer beliebigen TM M.
- Diese Kodierung heißt die *Gödelnummer* von M und wird mit $\langle M \rangle$ bezeichnet.
- Als Eingabe erhält U einen String der Form $\langle M \rangle w$ bestehend aus der Gödelnummer $\langle M \rangle$ und einem beliebigen Wort w.
- Die universelle TM simuliert das Verhalten der TM M auf der Eingabe w.
- Bei inkorrekter Eingabe (d.h. die Eingabe beginnt nicht mit einer Gödelnummer) gibt U eine Fehlermeldung aus.

Gödelnummern

Definition

Mit *Gödelnummern* bezeichnet man die eindeutige prefixfreie Kodierung von TM über einem festen Alphabet.

- O.B.d.A. gehen wir von binären Kodierungen über dem Alphabet $\{0, 1\}$ aus.
- *Prefixfrei* bedeutet, dass keine Gödelnummer Prefix (Anfangsteilwort) einer anderen Gödelnummer sein darf.
- Prefixfreiheit können wir beispielsweise erreichen indem
  - alle Gödelnummern auf 111 enden und ansonsten der Teilstring 111 nicht in der Kodierung vorkommt, oder alternativ
  - alle Gödelnummern mit 111 beginnen und auf 111 enden und ansonsten der Teilstring 111 nicht in der Kodierung vorkommt.

Mögliche Realisierung von Gödelnummern

Wir zeigen, wie eine geeignete Kodierung von TM aussehen könnte.

O.B.d.A. beschänken wir uns auf TM der folgenden Form:

- Sei $Q = \{q_1, \ldots, q_t\}$.
- Der Anfangszustand sei q_1 und der Stoppzustand q_2.
- O.B.d.A. sei $\Gamma = \{0, 1, B\}$. Wir nummerieren das Alphabet durch indem wir $X_1 = 0$, $X_2 = 1$ und $X_3 = B$ setzen.
- Auch die möglichen Kopfbewegungen nummerieren wir indem wir $D_1 = L$, $D_2 = N$ und $D_3 = R$ setzen.

Zur Beschreibung von TM dieser Form müssen wir nur die Übergangsfunktion als Binärstring kodieren.

Mögliche Realisierung von Gödelnummern

Kodierung der Übergangsfunktion:

- Der Übergang $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ wird kodiert durch den Binärstring
$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m \quad .$$

- Die Kodierung des $j$ten Übergangs bezeichnen wir mit $code(j)$.
- Die Gödelnummer einer TM M mit s vielen Übergängen ist dann

$$\langle M \rangle \;=\; 111\, code(1)\, 11\, code(2)\, 11 \ldots 11\, code(s)\, 111 \quad .$$

Mögliche Implementierung der universellen TM

Als Eingabe erhält die universelle TM U ein Wort der Form $\langle M \rangle w$ für beliebiges $w \in \{0,1\}^*$.

Wir implementieren U zunächst in Form einer 3-Band TM:

- Band 1 von U simuliert das Band der TM M.
- Band 2 von U enthält die Gödelnummer von M.
- Auf Band 3 speichert U den jeweils aktuellen Zustand von M.

Mögliche Implementierung der universellen TM

Initialisierung:

- U überprüft, ob die Eingabe eine korrekte Gödelnummer enthält. Falls nein, Fehlerausgabe.
- U kopiert die Gödelnummer auf Band 2 und schreibt die Binärkodierung des Anfangszustands auf Band 3.
- U bereitet Band 1 so vor, dass es nur das Wort w enthält. Der Kopf steht unter dem ersten Zeichen von w.

Laufzeit? – Die Laufzeit ist $O(1)$, wobei wir die Kodierungslänge von M als Konstante ansehen.

Mögliche Implementierung der universellen TM

Simulierung eines Schritts von M:

U sucht zu dem Zeichen an der Kopfposition aus Band 1 und dem Zustand auf Band 3 die Kodierung des entsprechenden Übergangs von M auf Band 2.

Wie in der Übergangsfunktion beschrieben

- aktualisiert U die Inschrift auf Band 1,
- bewegt U den Kopf auf Band 1, und
- verändert U den auf Band 3 abgespeicherten Zustand von M.

Laufzeit eines Schrittes: $O(1)$.

Das bedeutet U simuliert M mit konstantem Zeitverlust!

Mögliche Implementierung der universellen TM

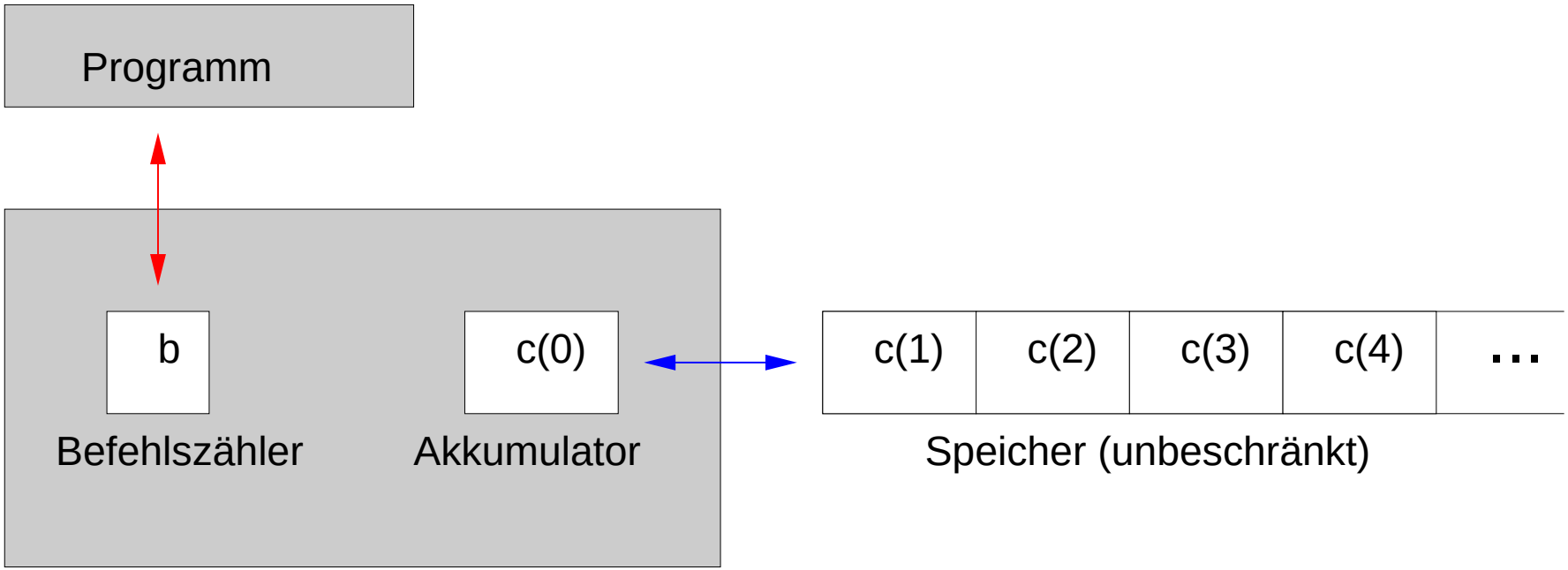
Können wir dieses Ergebnis auch mit einer (1-Band) TM erreichen?

Natürlich können wir die beschriebene 3-Band TM auf der 1-Band TM mit mehreren Spuren simulieren.

Aber bei Verwendung dieser Simulation handeln wir uns einen quadratischen Zeitverlust ein

Wir erhalten eine **universelle 1-Band TM mit konstantem Zeitverlust**, wenn wir die Gödelnunmmer auf Spur 2 und den Zustand auf Spur 3 mit dem Kopf der TM M mitführen.

Registermaschinen (RAM)



Befehlsatz:

LOAD,STORE,ADD,SUB,DIV

INDLOAD,INDSTORE,INDADD,INDSUB,INDDIV

CLOAD, CADD, CSUB, CDIV

GOTO, IF c(0)?x THEN GOTO j (wobei ? aus {=,<,<=,>,>=}), END

Erläuterung einiger ausgewählter RAM-Befehle

- LOAD i: $c(0) := c(i)$, $b := b + 1$;
- INDLOAD i: $c(0) := c(c(i))$, $b := b + 1$;
- CLOAD i: $c(0) := i$, $b := b + 1$;
- STORE i: $c(i) := c(0)$, $b := b + 1$;
- INDSTORE i: $c(c(i)) := c(0)$, $b := b + 1$;
- ADD i: $c(0) := c(0) + c(i)$, $b := b + 1$;
- CADD i: $c(0) := c(0) + i$, $b := b + 1$;
- INDADD i: $c(0) := c(0) + c(c(i))$, $b := b + 1$;

  $\vdots$
- DIV i: $c(0) := \lfloor c(0)/c(i) \rfloor$, $b := b + 1$;
- CDIV i: $c(0) := \lfloor c(0)/i \rfloor$, $b := b + 1$;
- INDIV i: $c(0) := \lfloor c(0)/c(c(i)) \rfloor$, $b := b + 1$;
- GOTO j: $b := j$
- IF $c(0) = x$ GOTO j: $b := j$ falls $c(0) = x$, sonst $b := b + 1$;
- END.

Funktionsweise der RAM

- Der Speicher der RAM ist unbeschränkt und besteht aus den Registern $c(0), c(1), c(2), c(3), \ldots$
- Die Inhalte der Register sind ganze Zahlen, die beliebig groß sein können.
- Die Eingabe sind ebenfalls natürliche Zahlen, die initial in den ersten Registern abgespeichert sind.
- Der Befehlszähler startet mit dem Wert 1. Ausgeführt wird jeweils der Befehl in derjenigen Zeile auf den der Befehlszähler verweist.
- Die Rechnung stoppt sobald der Befehl END erreicht ist.
- Die Ausgabe befindet sich nach dem Stoppen ebenfalls in den ersten Registern.

Beispielprogramm für die RAM: Potenzierung

Eingabe: Zahl $m \in \mathbb{N}$ in Register 1, Zahl $k \in \mathbb{N}$ in Register 2
Ausgabe: Zahl m^k in Register 3 (genannt *erg*)

1:	CLOAD 1	$akku := 1$
2:	STORE 3	$erg := akku$
3:	LOAD 2	$akku := k$
4:	If c(0) = 0 THEN GOTO 11	falls $akku = 0$ dann END
5:	CSUB 1	$akku := akku - 1$
6:	STORE 2	$k := akku$
7:	LOAD 3	$akku := erg$
8:	MULT 1	$akku := akku \cdot m$
9:	STORE 3	$erg := akku$
10:	GOTO 3	zurück zu Zeile 3
11:	END	erg enthält nun m^k

Disclaimer: Wir behaupten nicht, dieses Programm sei effizient.

Bemerkungen zur RAM

Auf einer RAM können wir offensichtlich alle Befehle wie beispielsweise Schleifen und Rekursionen, die wir von höheren Programmiersprachen gewohnt sind, realisieren.

Modelle für die Rechenzeit

- *Uniformes Kostenmaß:* Jeder Schritt zählt eine Zeiteinheit.
- *Logarithmisches Kostenmaß:* Die Laufzeitkosten eines Schrittes sind proportional zur binären Länge der Zahlen in den angesprochenen Registern.

Simulation RAM durch TM

Satz:

Jede im logarithmischen Kostenmaß $t(n)$-zeitbeschränkte RAM kann für ein Polynom q durch eine $O(q(n + t(n)))$-zeitbeschränkte TM simuliert werden.

Simulation RAM durch TM – Vorbemerkung zum Beweis

Im Beweis können wir für die Simulation eine 2-Band-TM statt einer (1-Band) TM verwenden. Warum?

Simulation RAM durch TM – Vorbemerkung zum Beweis

- Seien $\alpha, \beta, \gamma \in \mathbb{N}$ seien geeignet gewählte Konstanten.
- Wir werden zeigen, die Laufzeit der Simulation der RAM mit Laufzeitschranke $t(n)$ durch eine 2-Band TM ist nach oben beschränkt durch $t'(n) = \alpha(n + t(n))^\beta$.
- Die 2-Band TM mit Laufzeitschranke $t'(n)$ kann nun wiederum mit quadratischem Zeitverlust durch eine (1-Band) TM simuliert werden, also mit einer Laufzeitschranke der Form $t''(n) = \gamma(t'(n))^2$ simulieren.
- Für die Simulation der RAM auf der (1-Band) TM ergibt sich somit eine Laufzeitschranke von

$$t''(n) \;=\; \gamma(t'(n))^2 \;=\; \gamma\left(\alpha(n + t(n))^\beta\right)^2 \;=\; \gamma\alpha^2 \cdot (n+t(n))^{2\beta}.$$

- Diese Laufzeitschranke ist polynomiel in $n + t(n)$, weil sowohl der Term $\gamma\alpha^2$ als auch der Term 2β konstant sonst.

Simulation RAM durch TM – Vorbemerkung zum Beweis

Beobachtung:

Die Klasse der Polynome ist gegen Hintereinanderausführung abgeschlossen.

Deshalb können wir eine *konstante Anzahl* von Simulationen, deren Zeitverlust jeweils polynomiell nach oben beschränkt ist, ineinander schachteln und erhalten dadurch wiederum eine Simulation mit polynomiell beschränktem Zeitverlust.

Simulation RAM durch TM – Beweis

Beweis des Satzes:

- Wir verwenden eine 2-Band-TM, die die RAM schrittweise simuliert.
- Das RAM-Programm P bestehe aus p Programmzeilen.
- Für jede Programmzeile schreiben wir ein TM-Unterprogramm. Sei M_i das Unterprogramm für Programmzeile i, $0 \leq i \leq p$.
- Außerdem spezifizieren wir ein Unterprogramm M_0 für die Initialisierung der TM und M_{p+1} für die Aufbereitung der Ausgabe des Ergebnisses.

Simulation RAM durch TM – Beweis

Abspeichern der *Registermaschinenkonfiguration* auf der TM:

- Den Befehlszähler kann die TM im Zustand abspeichern, da die Länge des RAM-Programms konstant ist.
- Die Registerinhalte werden wie folgt auf Band 2 abgespeichert:

$$\#\#0\#bin(c(0))\#\#bin(i_1)\#bin(c(i_1))\#\#\ldots$$

$$\ldots\#\#bin(i_m)\#bin(c(i_m))\#\#\#\ ,$$

wobei $0, i_1, \ldots, i_m$ die Indizes der benutzten Register sind.

Beobachtung:

Der Länge des Speicherinhalts auf Band 2 ist durch $O(n + t(n))$ beschränkt, weil die RAM für jedes neue Bit, das sie erzeugt, mindestens eine Zeiteinheit benötigt.

Simulation RAM durch TM – Beweis

Rechenschritt für Rechenschritt simuliert die TM nun die Konfigurationsveränderungen der RAM.

Dazu ruft die TM das im Programmzähler b angegebene Unterprogramm M_b auf.

Das Unterprogramm M_b

- kopiert den Inhalt der in Programmzeile b angesprochenen Register auf Band 1,
- führt die notwendigen Operationen auf diesen Registerinhalten durch,
- kopiert dann das Ergebnis in das in Zeile b angegebene Register auf Band 2 zurück, und
- aktualisiert zuletzt den Programmzähler b.

Simulation RAM durch TM – Beweis

Laufzeitanalyse:

Die Initialisierung erfordert Zeit $O(n)$.

Alle anderen Unterprogramme haben eine Laufzeit, die polynomiell in der Länge der Bandinschrift auf Band 2 beschränkt ist, also eine polynomielle Laufzeit in $n + t(n)$.

Somit ist auch die Gesamtlaufzeit der Simulation polynomiell in $n + t(n)$ beschränkt. □

Simulation TM durch RAM

Satz:

Jede $t(n)$-zeitbeschränkte TM kann durch eine RAM simuliert werden, die uniform $O(t(n) + n)$ und logarithmisch $O((t(n) + n) \log(t(n) + n))$ zeitbeschränkt ist.

Simulation TM durch RAM – Beweis

Beweis des Satzes:

- O.B.d.A. nehmen wir an, es handelt sich um eine TM mit einseitig beschränktem Band, deren Zellen mit $0, 1, 2, 3, \ldots$ durchnummeriert sind. (vgl. Übung)
- Die Zustände und Zeichen werden ebenfalls durchnummeriert und mit ihren Nummern identifiziert, so dass sie in den Registern abgespeichert werden können.
- Register 1 speichert den Index der Kopfposition.
- Register 2 speichert den aktuellen Zustand.
- Die Register $3, 4, 5, 6, \ldots$ speichern die Inhalte der jemals besuchten Bandpositionen $0, 1, 2, 3, \ldots$

Simulation TM durch RAM – Beweis

Die TM wird nun Schritt für Schritt durch die RAM simuliert.

Auswahl des richtigen TM-Übergangs

Die RAM verwendet eine zwei-stufige *if*-Abfrage:

- Auf einer ersten Stufe von $|Q|$ vielen *if*-Abfragen wird der aktuelle Zustand selektiert.
- Für jeden möglichen Zustand, gibt es dann eine zweite Stufe von $|\Gamma|$ vielen *if*-Abfragen, die das gelesene Zeichen selektieren.

Durchführung des TM-Übergangs

Je nach Ausgang der *if*-Abfragen aktualisiert die RAM

- den TM-Zustand in Register 2,
- die TM-Bandinschrift in Register $c(1)$ und
- die TM-Bandposition in Register 1.

Simulation TM durch RAM – Beweis

Laufzeitanalyse im uniformen Kostenmodell:

- Die Initialisierung kann in Zeit $O(n)$ durchgeführt werden.
- Die Simulation jedes einzelnen TM-Schrittes hat konstante Laufzeit.
- Insgesamt ist die Simulationszeit somit $O(n + t(n))$.

Simulation TM durch RAM – Beweis

- Die in den Registern gespeicherten Zahlen repräsentieren Zustände, Zeichen und Bandpositionen.
- Zustände und Zeichen haben eine konstante Kodierungslänge.
- Die Bandpositionen, die während der Simulation angesprochen werden, sind durch $\max\{n, t(n)\} \leq n + t(n)$ beschränkt. Die Kodierungslänge dieser Positionen ist also $O(\log(t(n) + n))$.
- Damit kann die Simulation jedes einzelnen TM-Schrittes in Zeit $O(\log(t(n) + n))$ durchgeführt werden.
- Insgesamt ergibt sich somit eine Simulationszeit von $O((t(n) + n) \log(t(n) + n)$.

□

Zusammenfassung

- Die Mehrband-TM kann mit quadratischem Zeitverlust durch eine (1-Band) TM simuliert werden.
- TM und RAM (mit logarithmischen Laufzeitkosten) können sich gegenseitig mit polynomiellem Zeitverlust simulieren.
- Wenn es uns also „nur“ um Fragen der Berechenbarkeit von Problemen (oder um ihre Lösbarkeit in polynomieller Zeit) geht, können wir wahlweise auf die TM, die Mehrband-TM oder die RAM zurückgreifen.

Die Church-Turing-These

Kein jemals bisher vorgeschlagenes „vernünftiges“ Rechnermodell hat eine größere Mächtigkeit als die TM.

Diese Einsicht hat Church zur Formulierung der folgenden These veranlasst.

Church-Turing-These

Die Klasse der rekursiven Funktionen (also der durch eine TM-berechenbaren Funktionen) stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.

Da diese These zu großen Teilen auf Turings Forschungsergebnissen beruht, wird sie als *Church-Turing-These* bezeichnet.