

Alan Turing

ALAN TURING YEAR

2012



FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 18 Turingmaschinen

Michael Köhler-Bußmeier

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

- \subseteq Familie der **kontextfreien Mengen**
- \subseteq Familie der **kontextsensitiven Mengen**
- \subseteq Familie der **entscheidbaren Mengen**
- \subseteq Familie der **aufzählbaren Mengen**
- \subseteq Familie der **abzählbaren Mengen**
- \neq Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

\subsetneq Familie der **kontextfreien Mengen**

\subsetneq Familie der **kontextsensitiven Mengen**

\subsetneq Familie der **entscheidbaren Mengen**

\subsetneq Familie der **aufzählbaren Mengen**

\subsetneq Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

- Familie der **kontextfreien Mengen**
- Familie der **kontextsensitiven Mengen**
- Familie der **entscheidbaren Mengen**
- Familie der **aufzählbaren Mengen**
- Familie der **abzählbaren Mengen**
- Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

\subseteq Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

\subseteq Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

\subseteq Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

\subseteq Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

Alan Mathison Turing (1912 – 1954)

Turing formulierte 1937
die nach ihm benannten
Maschinen.



1931



1946



1946



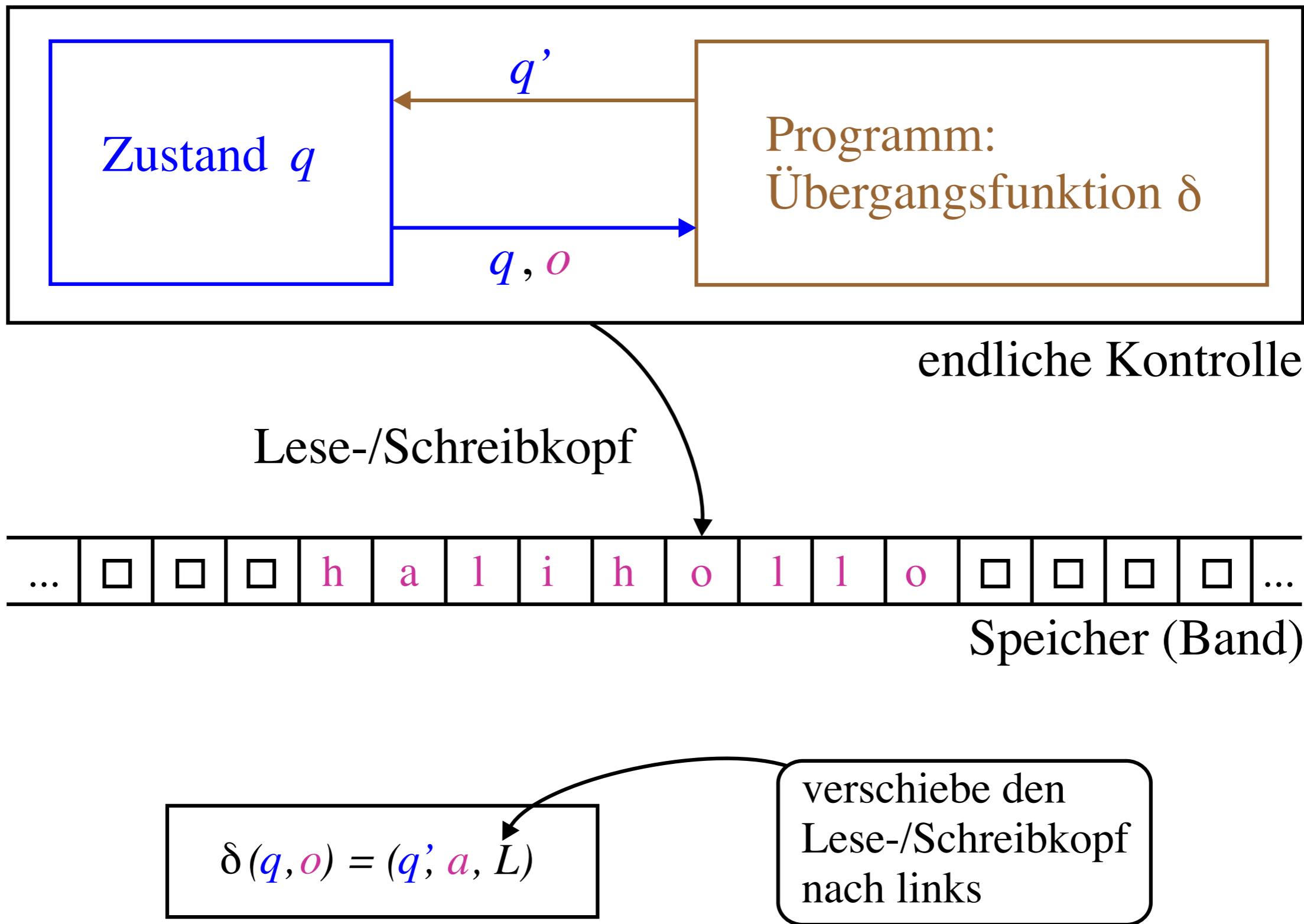
1952



1948

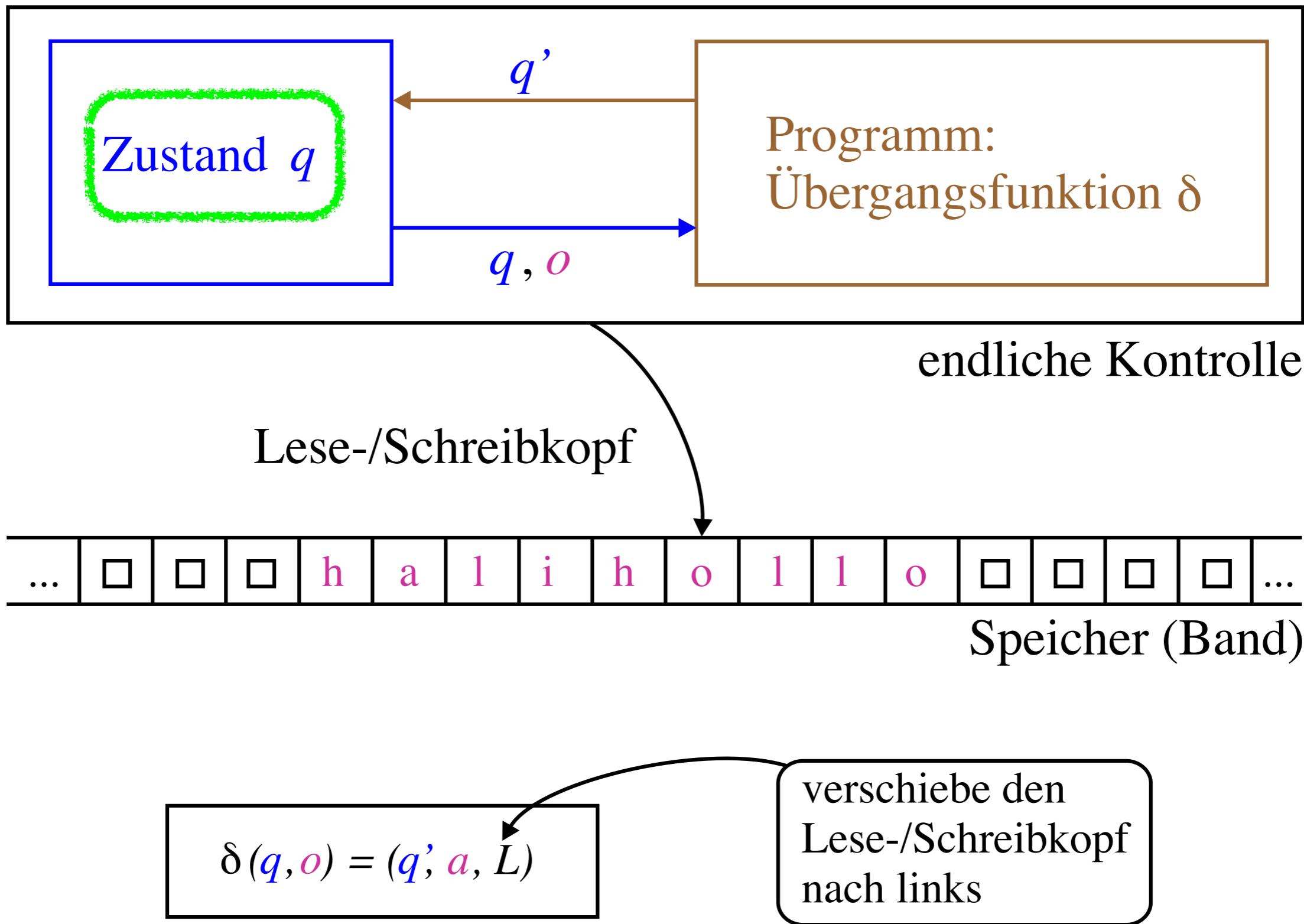
Turingmaschine / Turingautomat

Turing Maschine



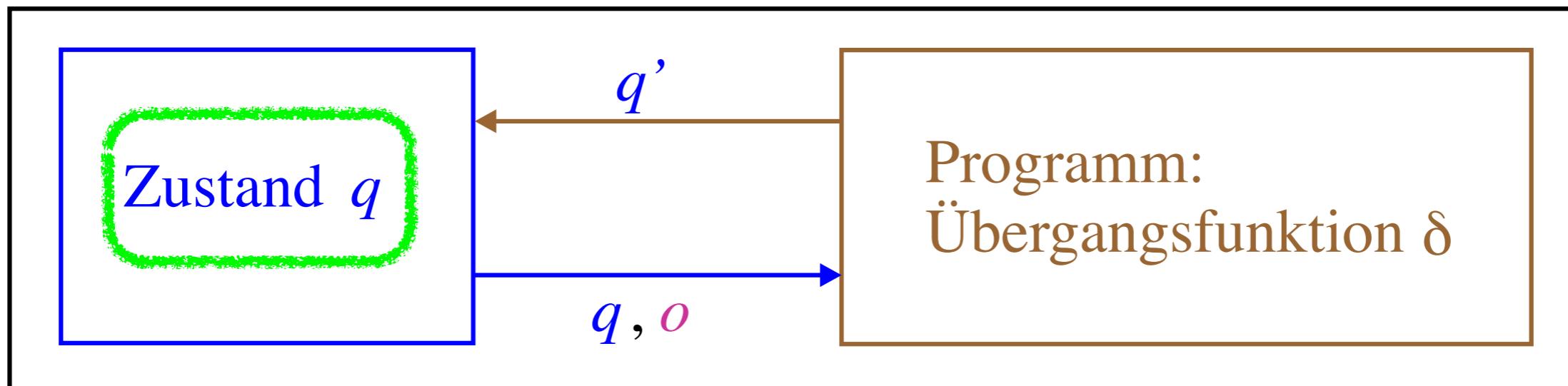
Turingmaschine / Turingautomat

Turing Maschine

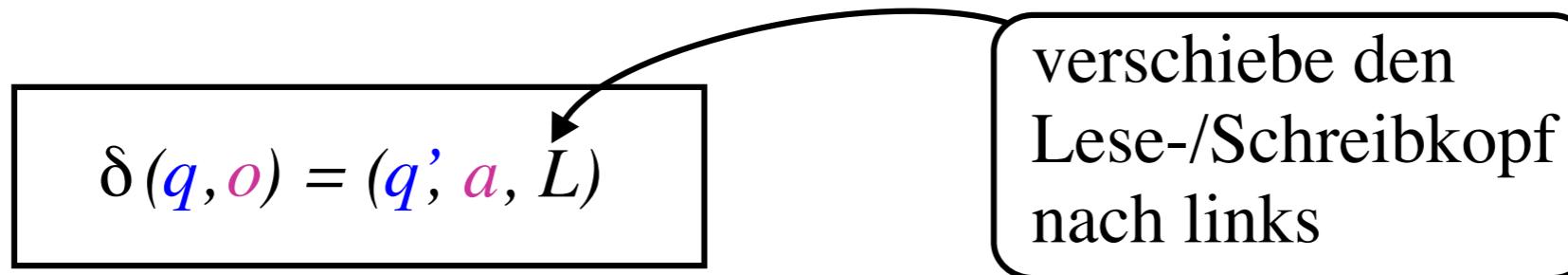
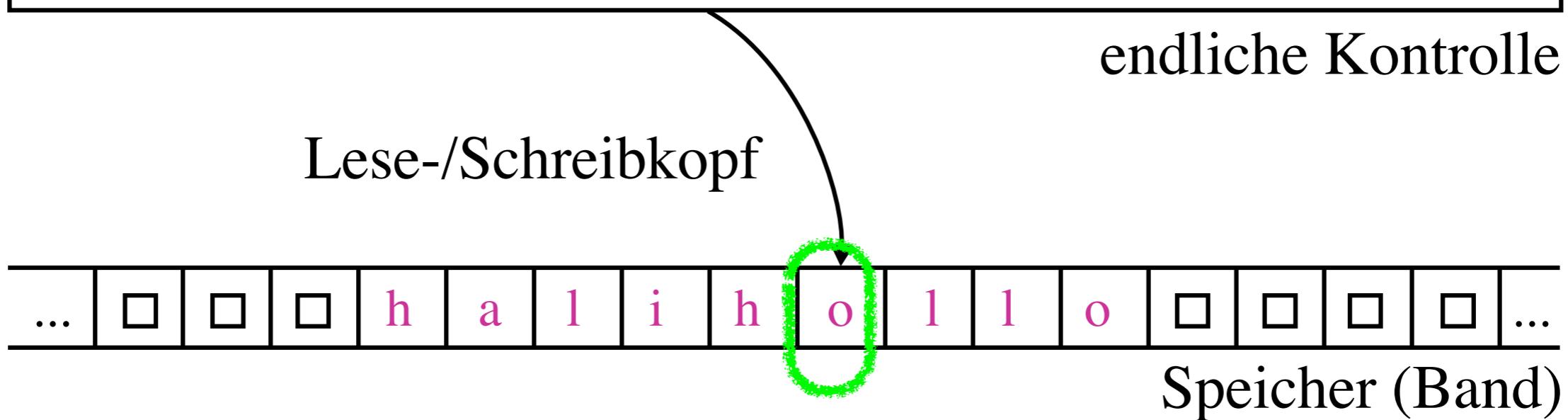


Turingmaschine / Turingautomat

Turing Maschine

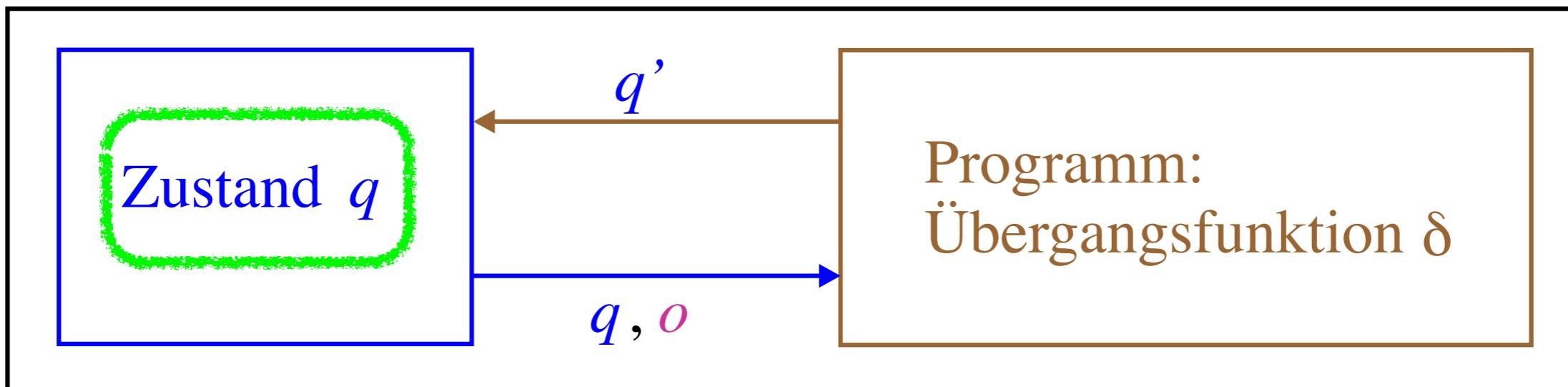


endliche Kontrolle

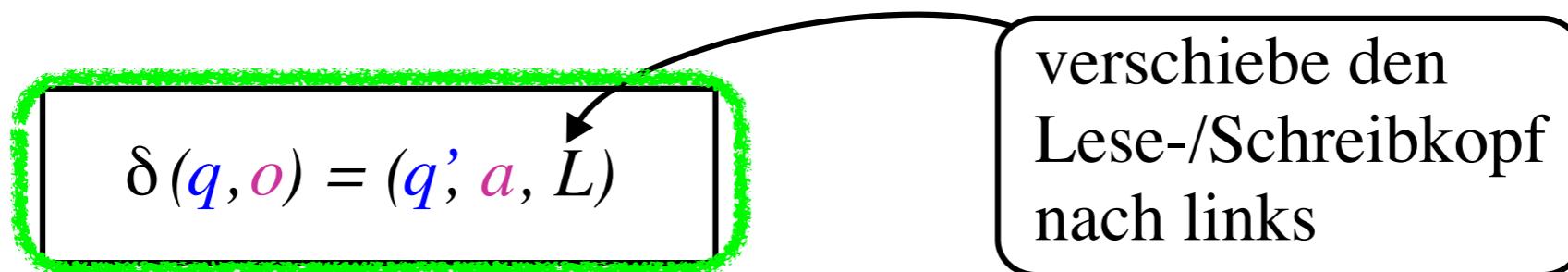
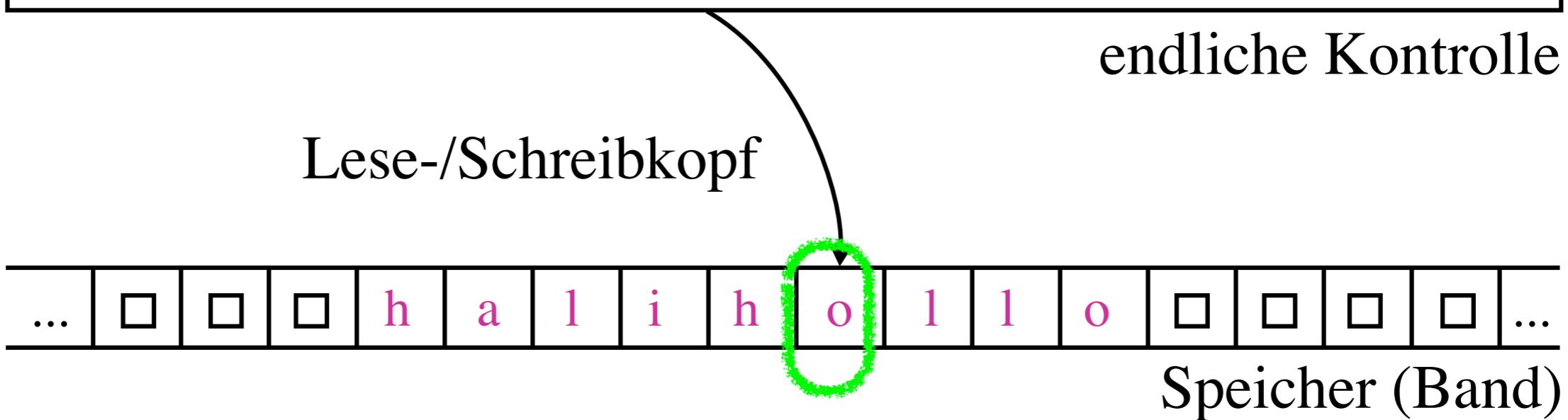


Turingmaschine / Turingautomat

Turing Maschine

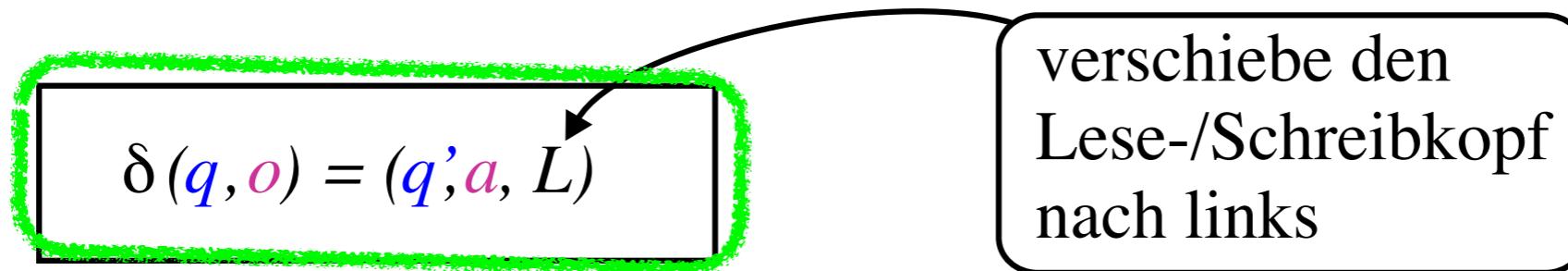
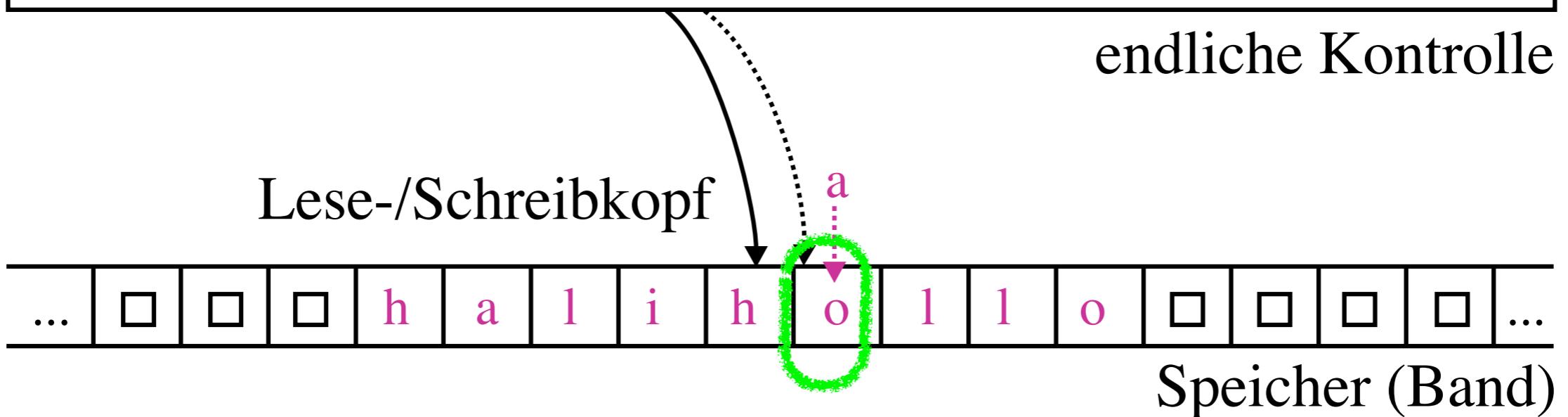
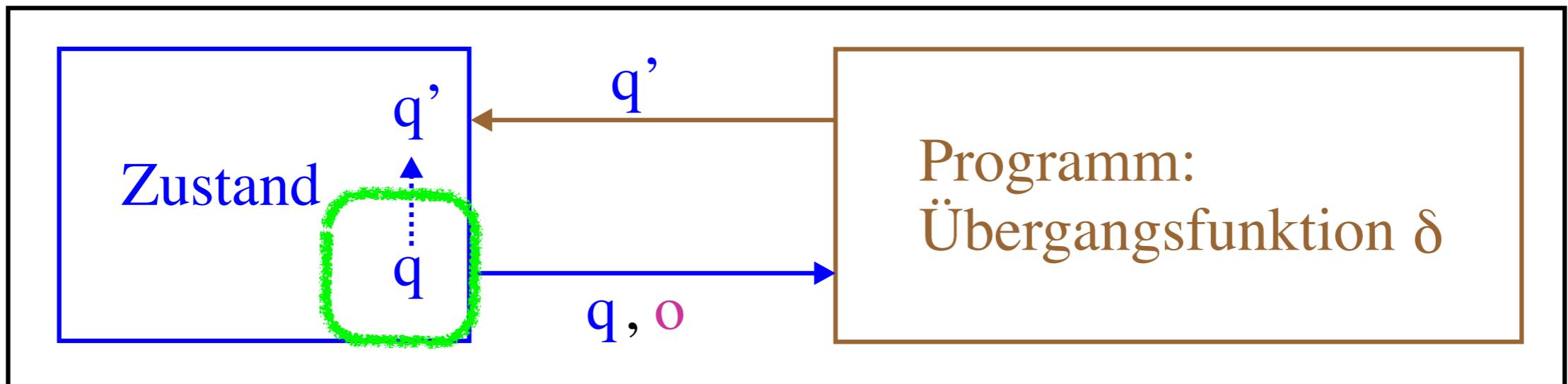


endliche Kontrolle



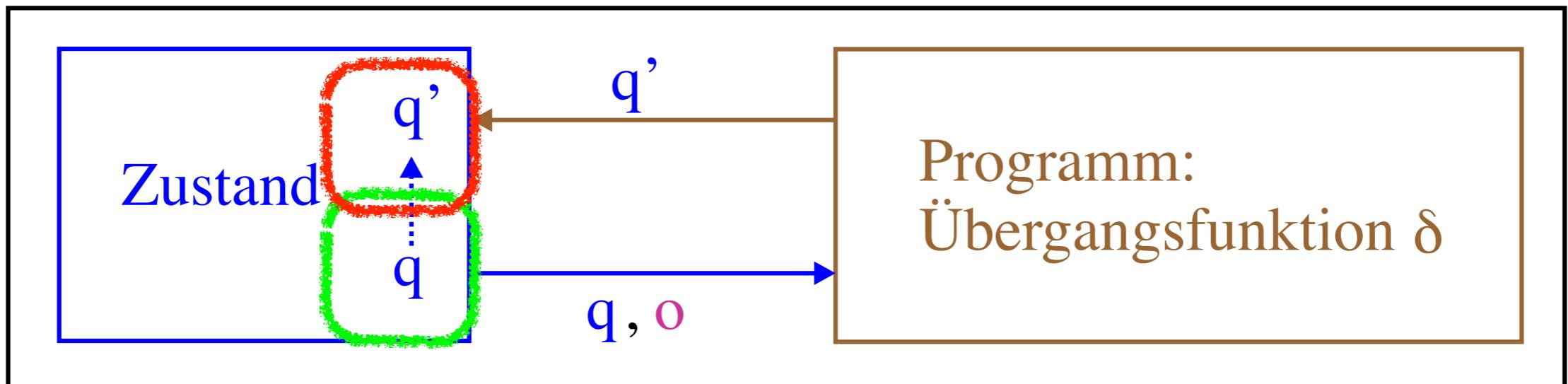
Turingmaschine: Verhalten

Turing Maschine

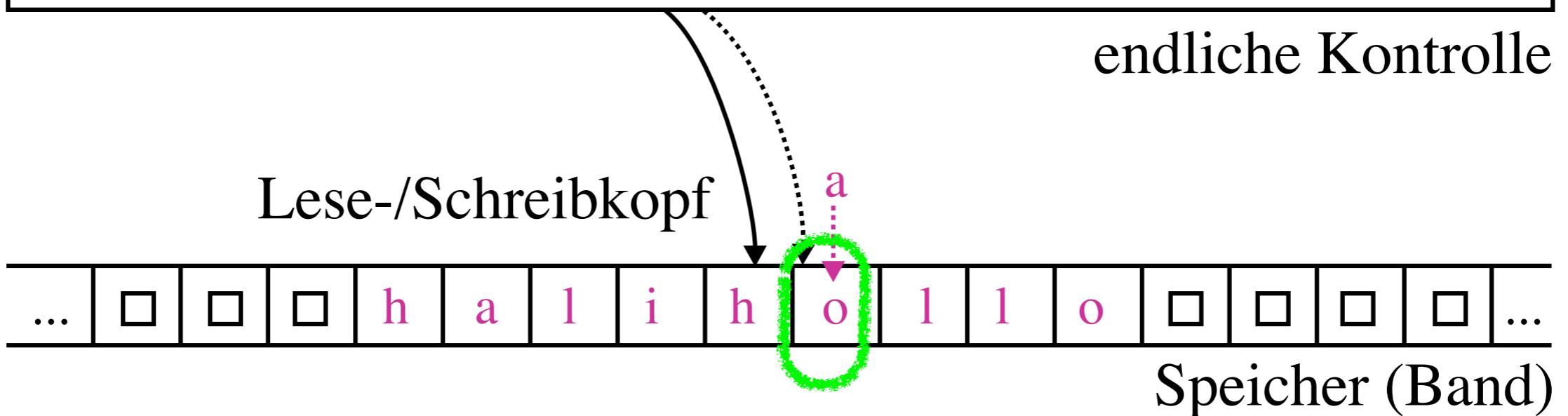


Turingmaschine: Verhalten

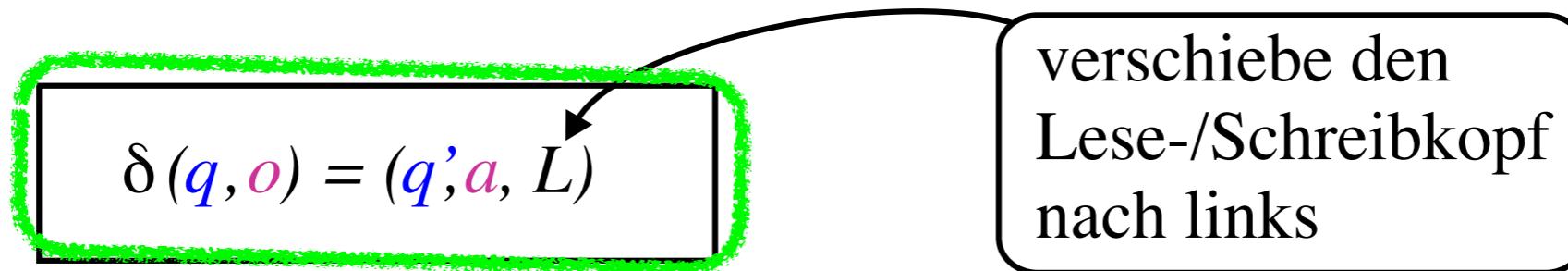
Turing Maschine



endliche Kontrolle

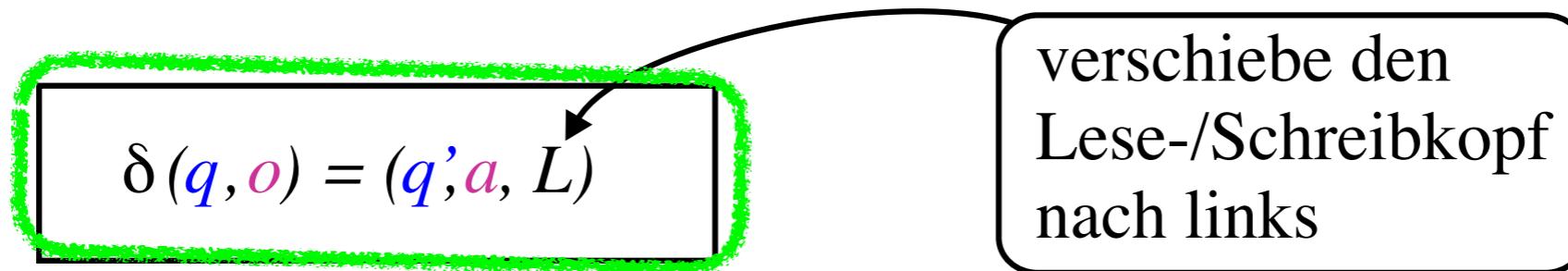
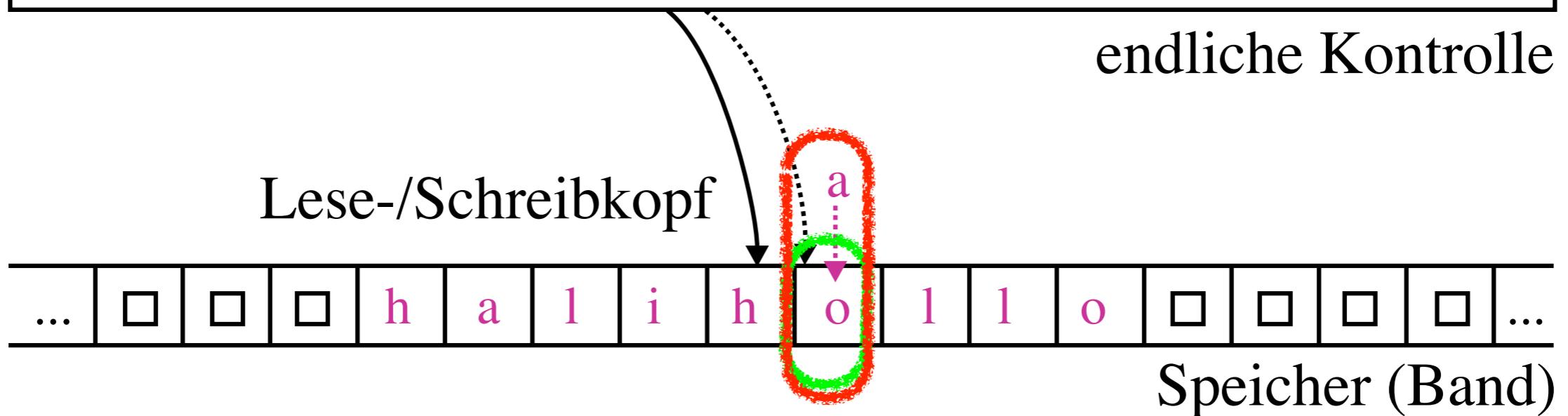
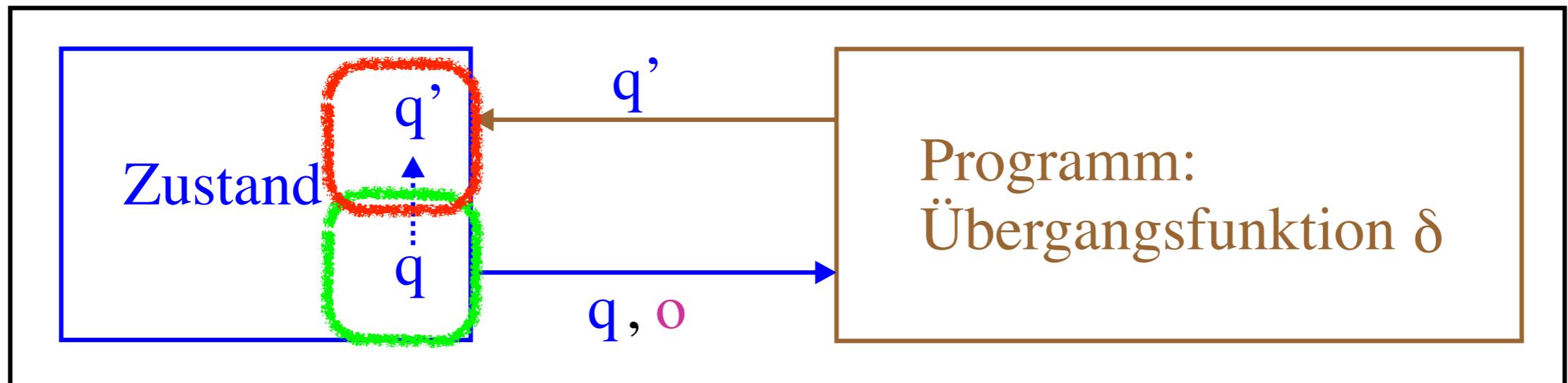


Speicher (Band)



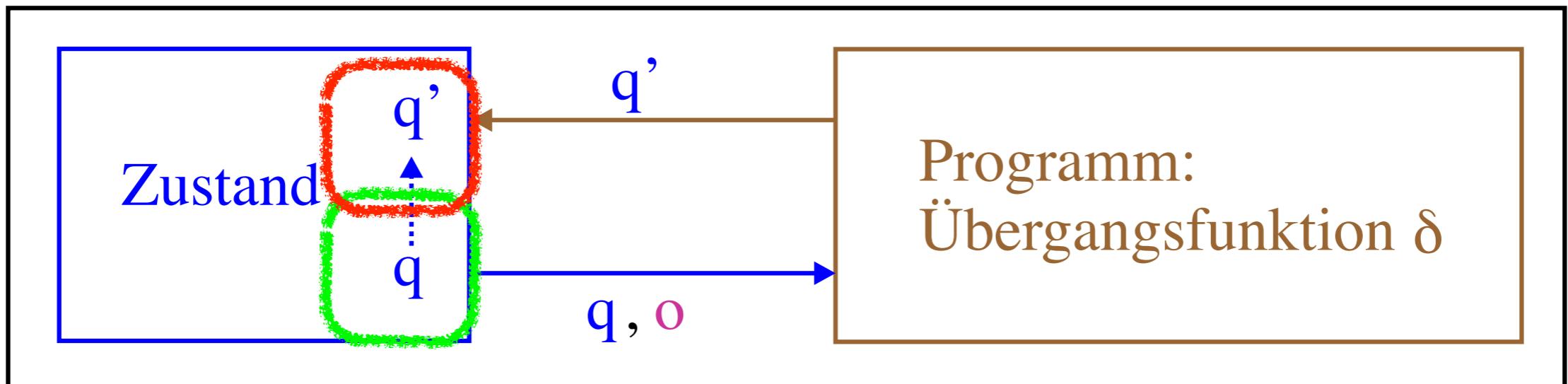
Turingmaschine: Verhalten

Turing Maschine

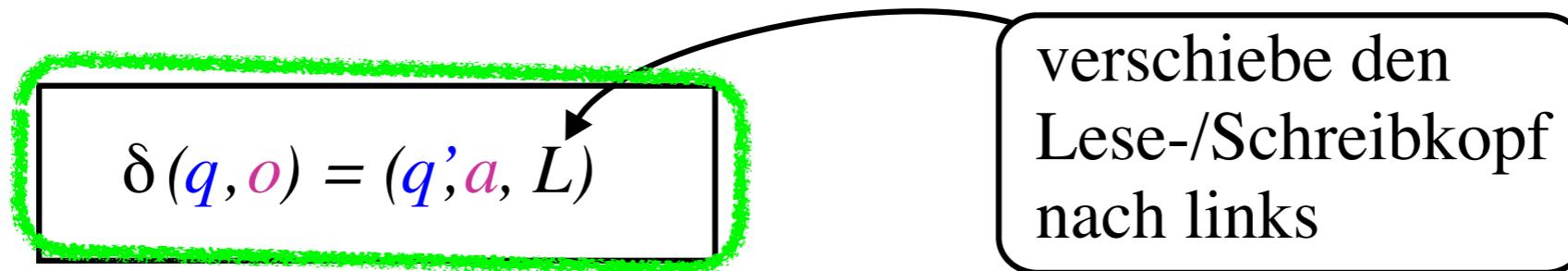
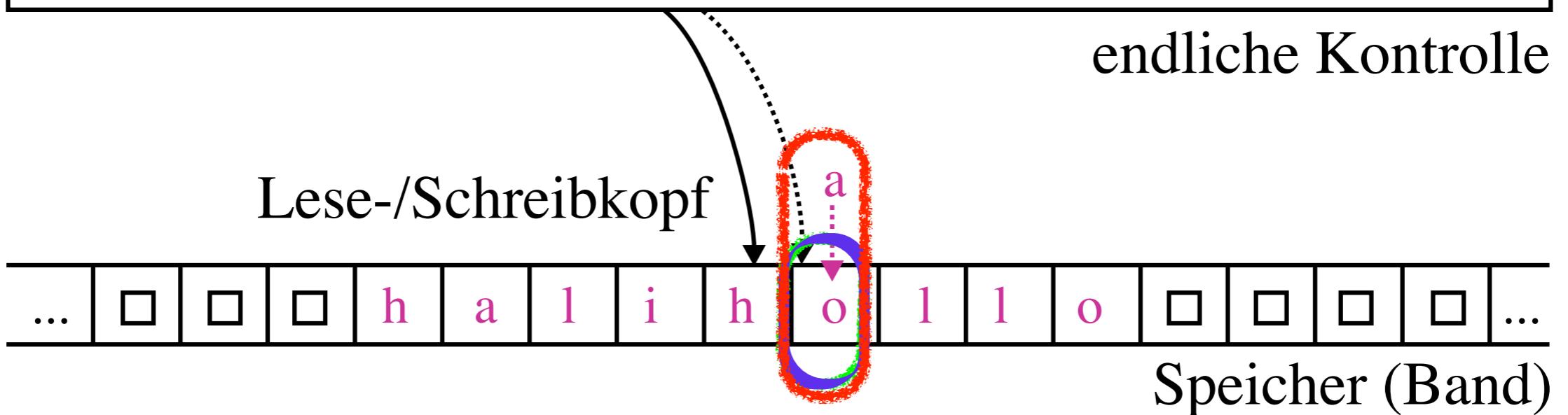


Turingmaschine: Verhalten

Turing Maschine

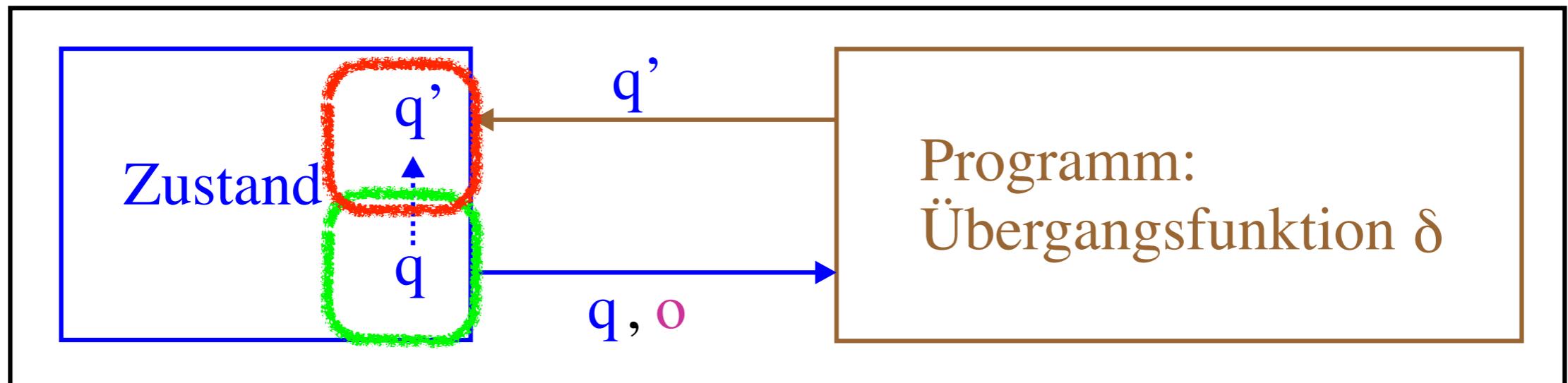


endliche Kontrolle

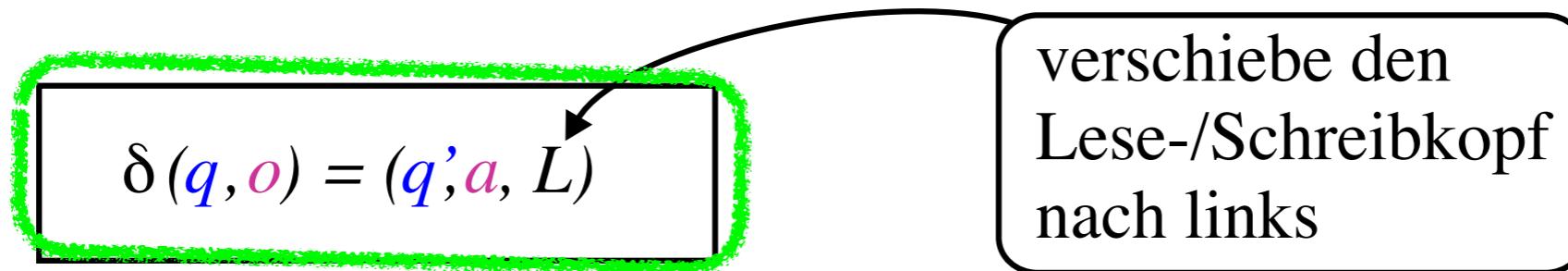
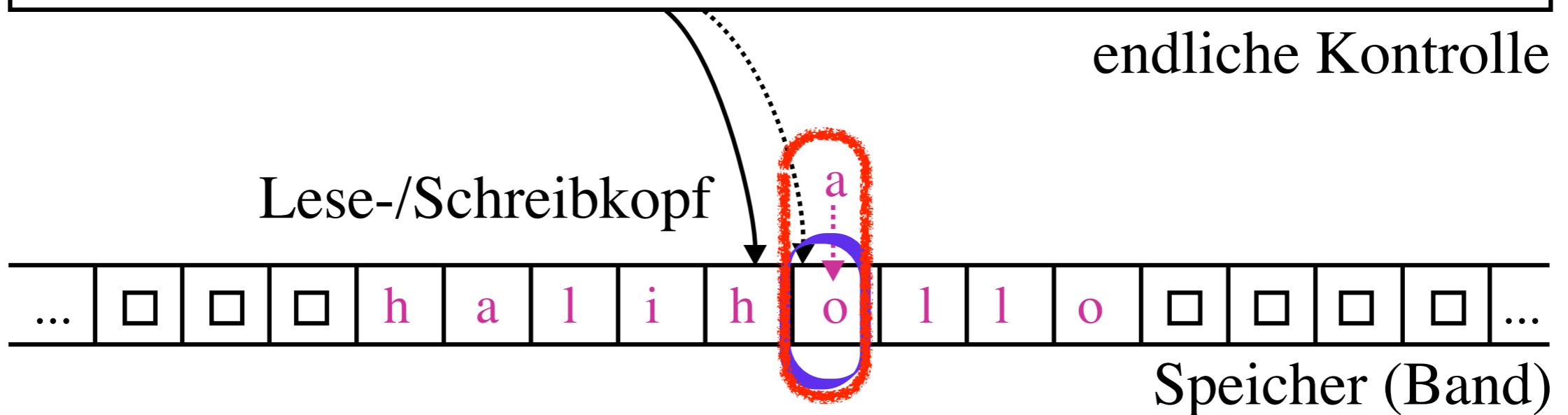


Turingmaschine: Verhalten

Turing Maschine

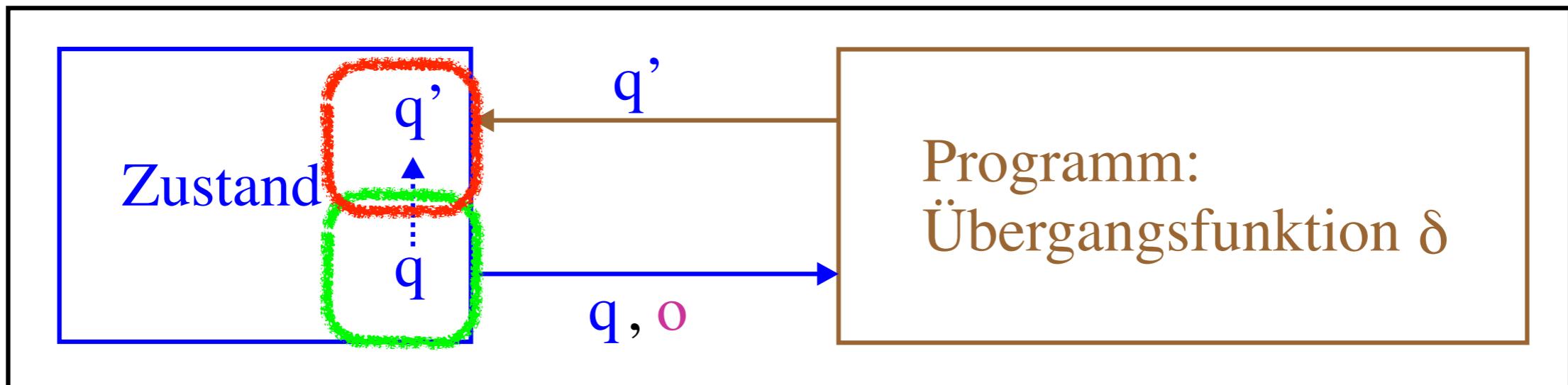


endliche Kontrolle

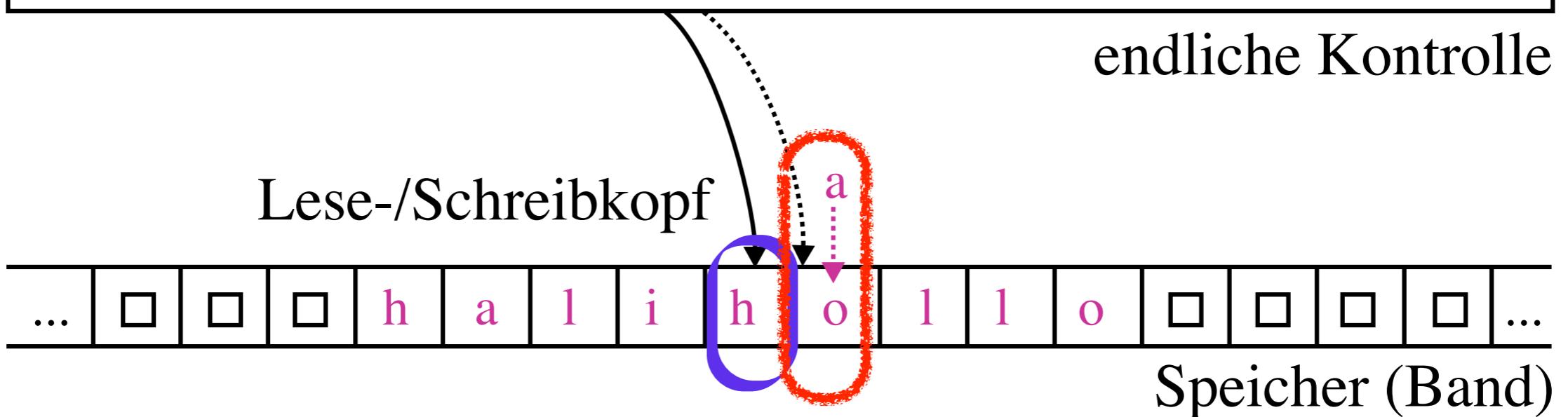


Turingmaschine: Verhalten

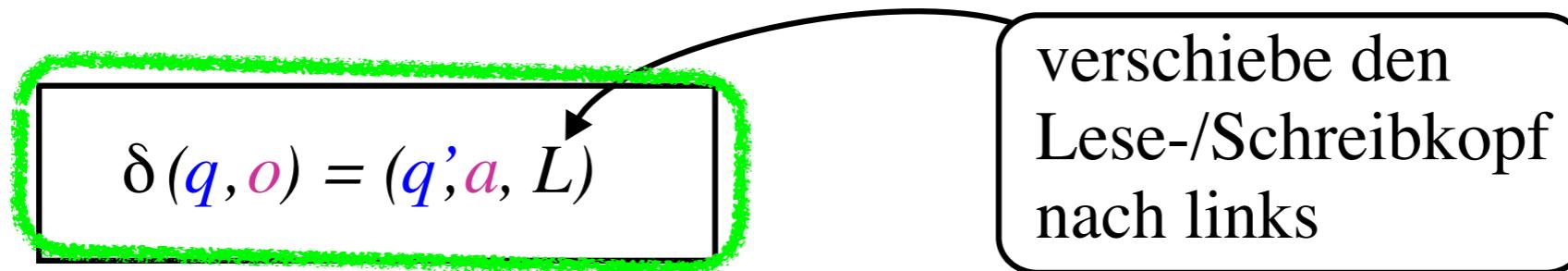
Turing Maschine



endliche Kontrolle



Speicher (Band)



Turingmaschinen

Turingmaschinen können angesehen werden:

1. als deterministisches **Berechnungsmodell** für Funktionen,
2. als deterministische **Algorithmenspezifikation** oder
3. als **akzeptierende Automaten**:
 - 3.1. *deterministisch*
 - 3.2. *nichtdeterministisch*

Wir werden alle Aspekte und
deren Zusammenhänge betrachten,
z.B. gegenseitige Simulation.

Deterministische Turingmaschine (DTM)

Definition 18.4:

Eine **deterministische Turing-Maschine** (DTM) [(TA) bei Vossen/Witt] wird beschrieben durch $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, wobei:

- Q endliche Menge *(Zustände),*
- Γ endliche Menge *(Bandalphabet),*
- Σ endliche Menge *(Eingabealphabet)*
mit $\Sigma \not\subseteq \Gamma$, und $\Gamma \cap Q = \emptyset$,
- $\# \in \Gamma \setminus \Sigma$ *(Symbol für das unbeschriebene Feld),*
- $q_0 \in Q$ *(Anfangs- oder Startzustand),*
- $F \subseteq Q$ *(Menge der Endzustände),*
- $\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{l, r, -\})$ *(Übergangsfunktion.)*

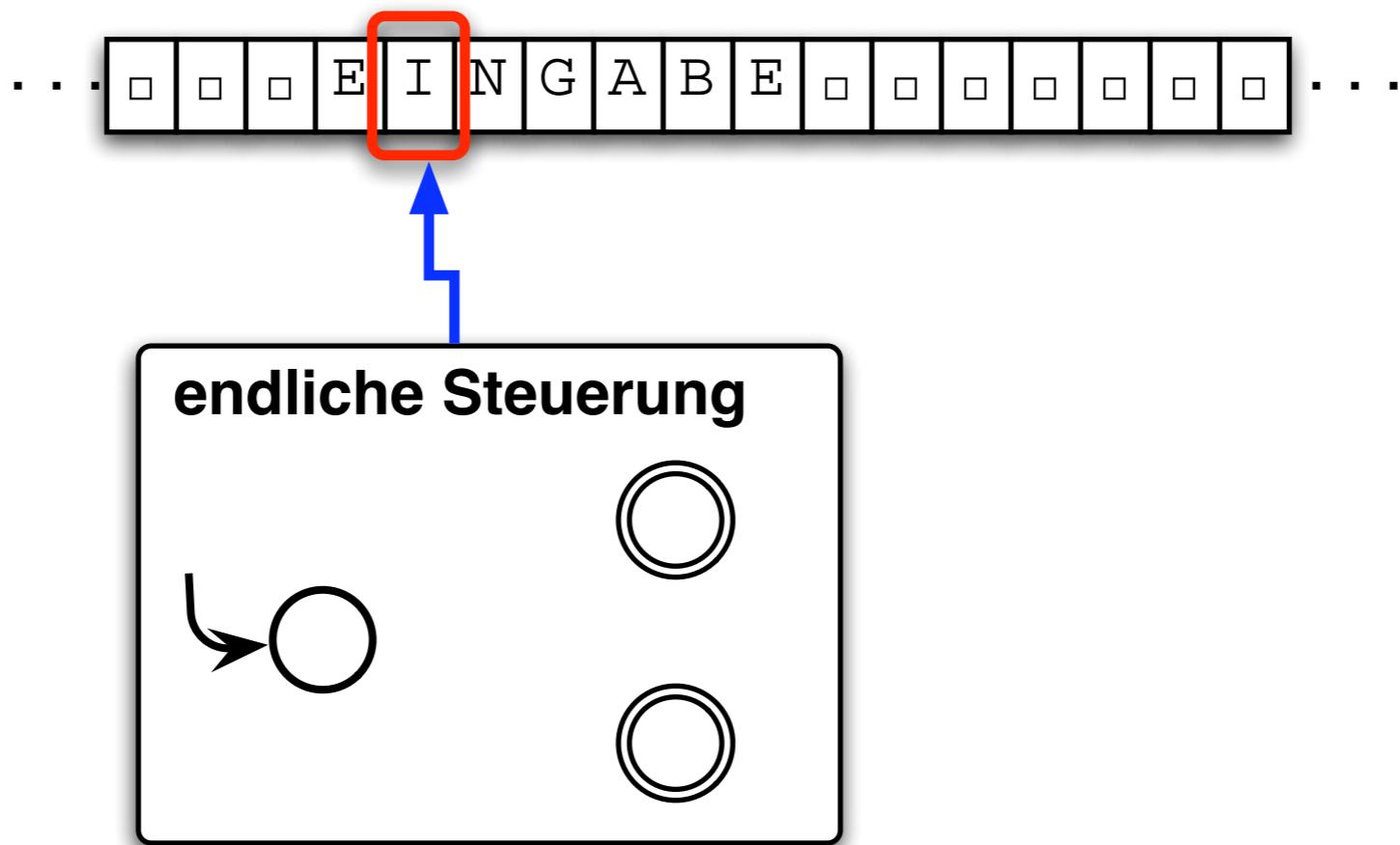
Eigenschaften einer DTM und Varianten

- Die Übergangsfunktion darf partiell sein
(d.h. sie darf Definitionslücken enthalten)!
- Folgezustände/Folgekonfigurationen sind eindeutig bestimmt.
- Es gibt genau einen Startzustand.

Es gibt Varianten, die sich bzgl. der Berechenbarkeit als äquivalent erweisen:

- einseitig unendliches Band
- beidseitig unendliches Band
- mehrere Arbeitsbänder

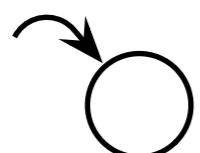
ein Beispiel: mit Funktionstabelle . . .



Zustand	0	1	X	Y	#
q_0	(X, q_1, r)	—	—	(Y, q_3, r)	—
q_1	$(0, q_1, r)$	(Y, q_2, l)	—	(Y, q_1, r)	—
q_2	$(0, q_2, l)$	—	(X, q_0, r)	(Y, q_2, l)	—
q_3	—	—	—	(Y, q_3, r)	$(\#, q_4, r)$
q_4	—	—	—	—	—

... in graphischer Darstellung

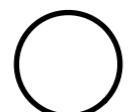
Startzustand:



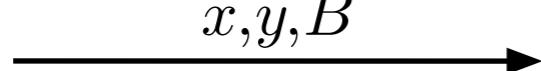
Endzustände:



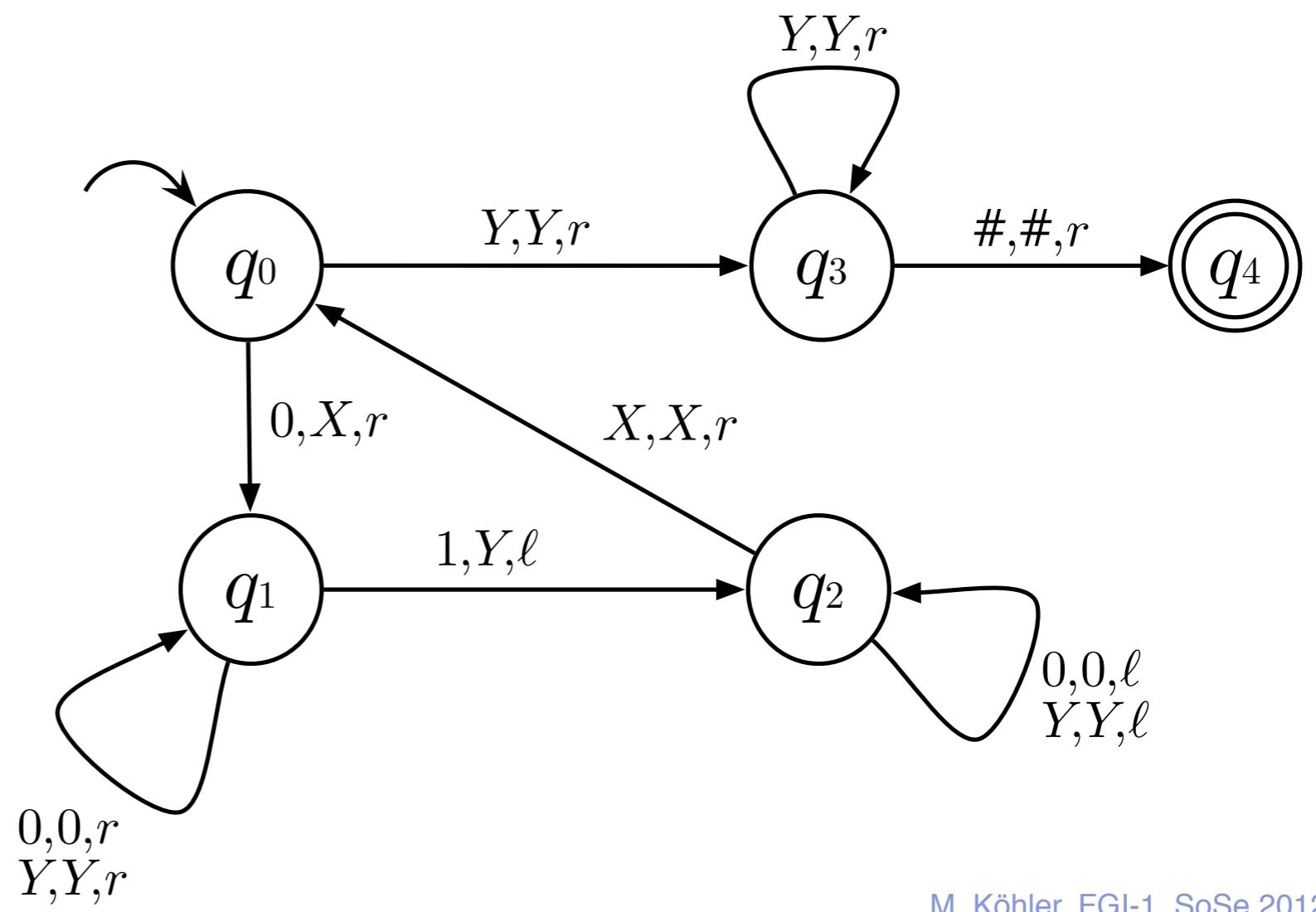
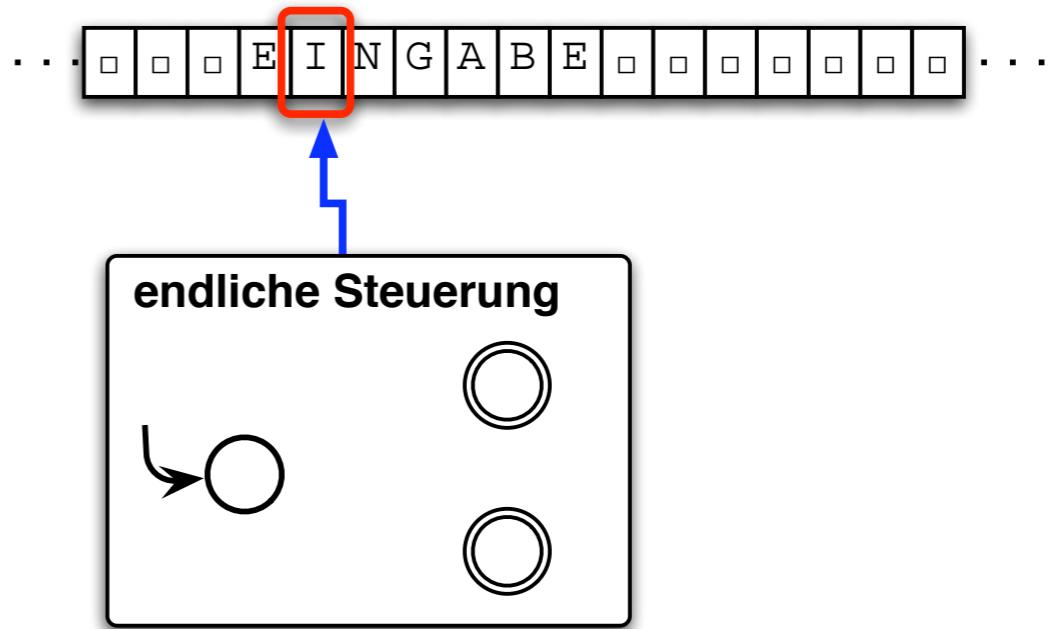
Zustände:



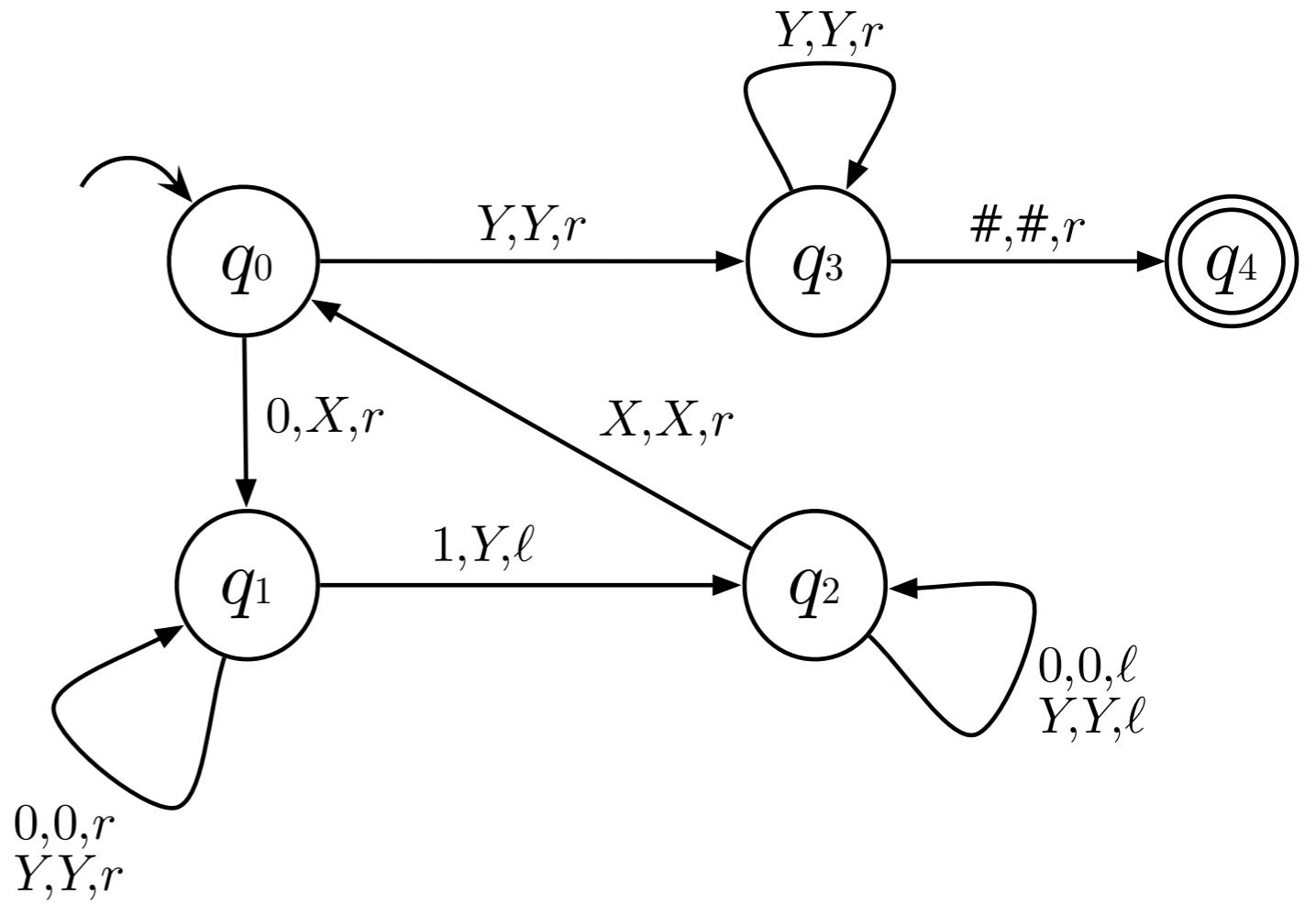
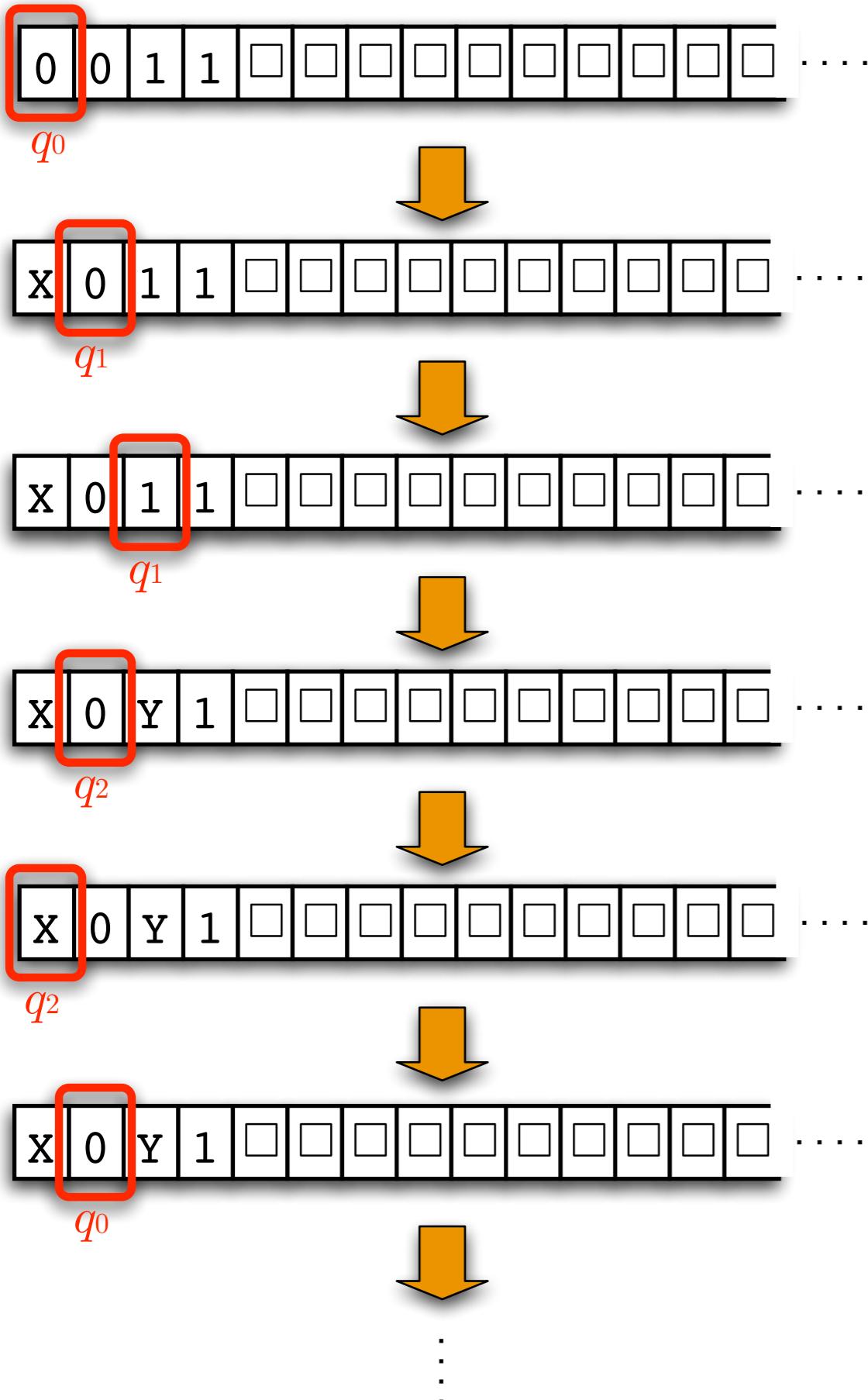
Kanten:



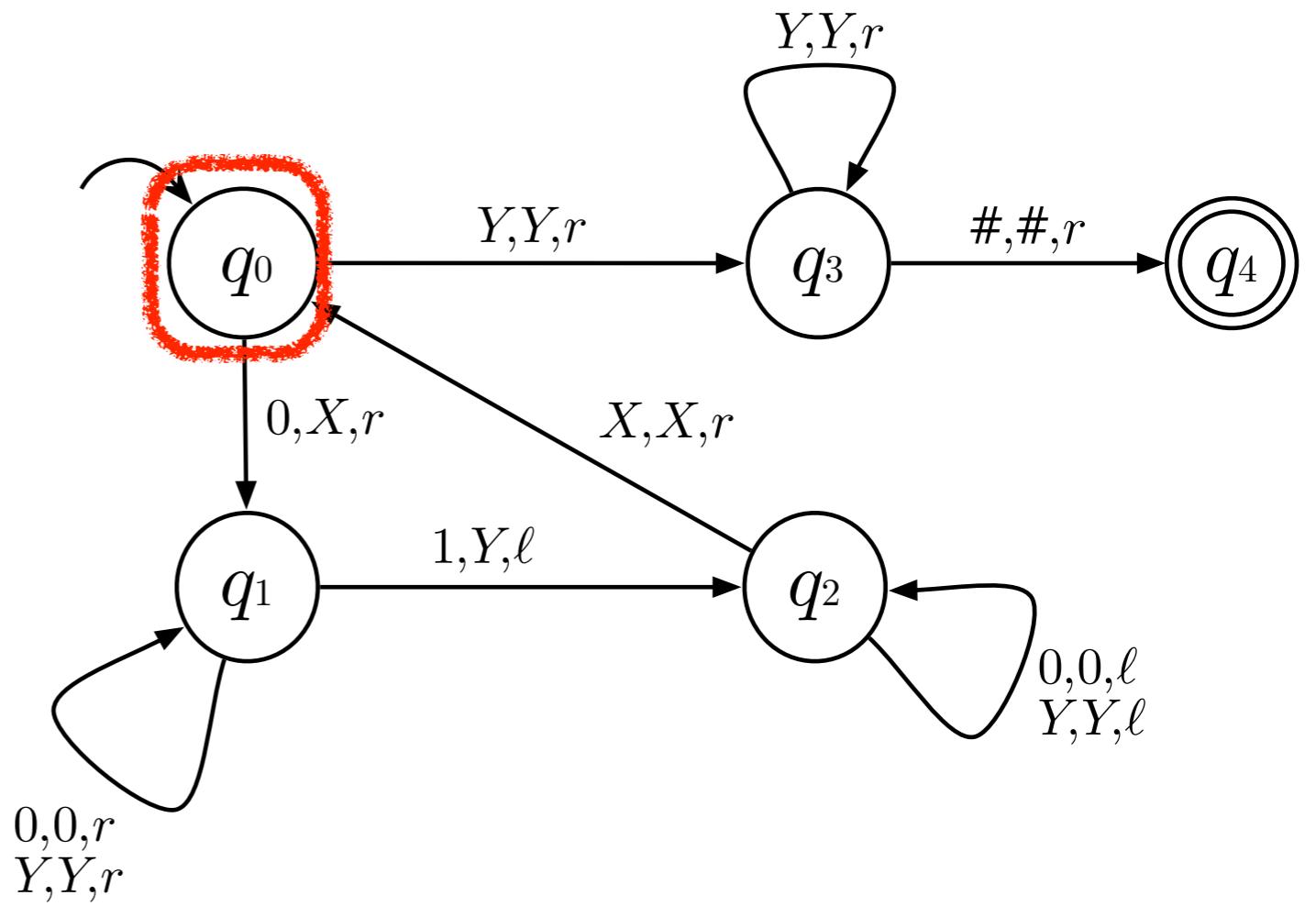
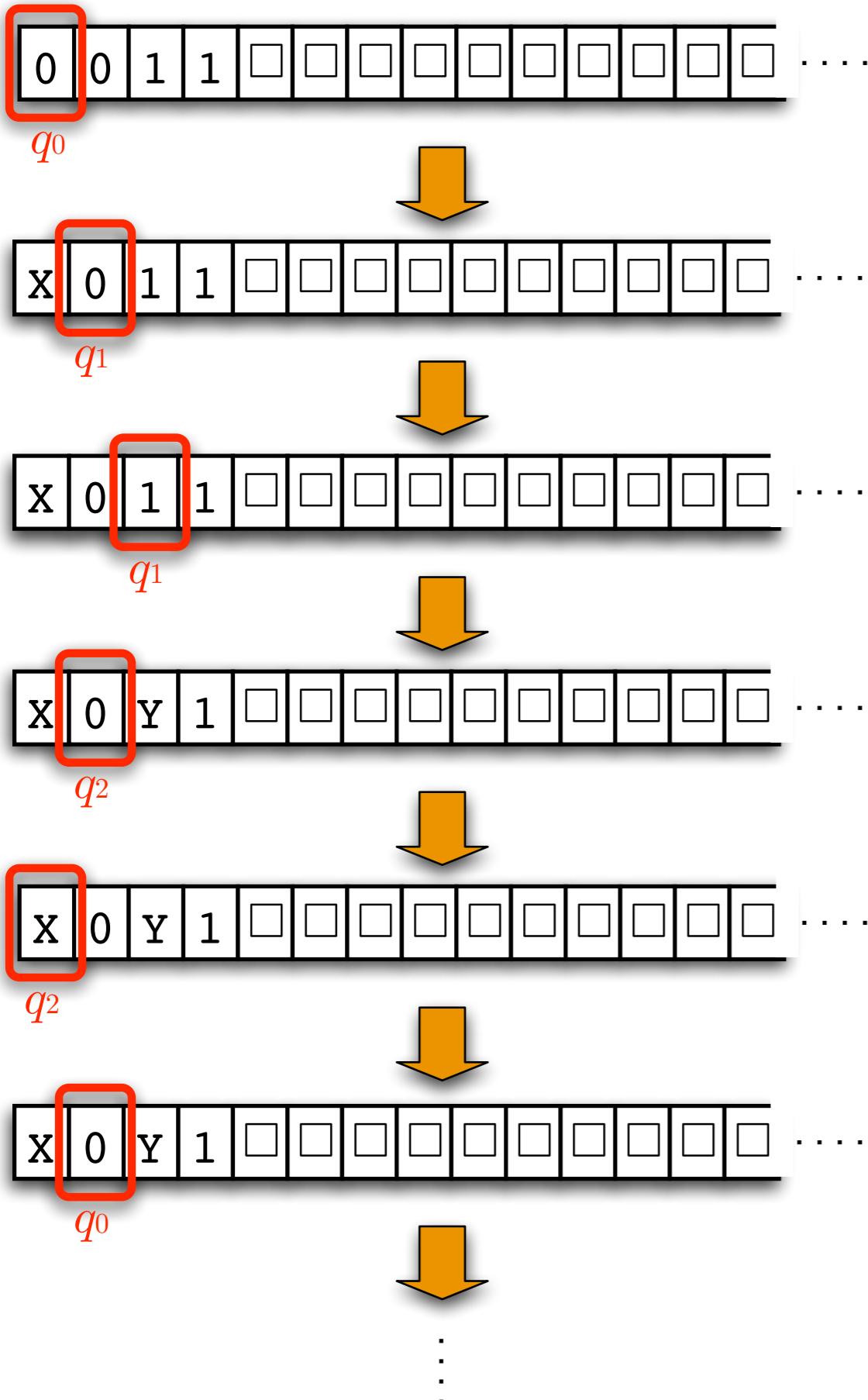
mit $x, y \in \Gamma$ und $B \in \{r, \ell, -\}$



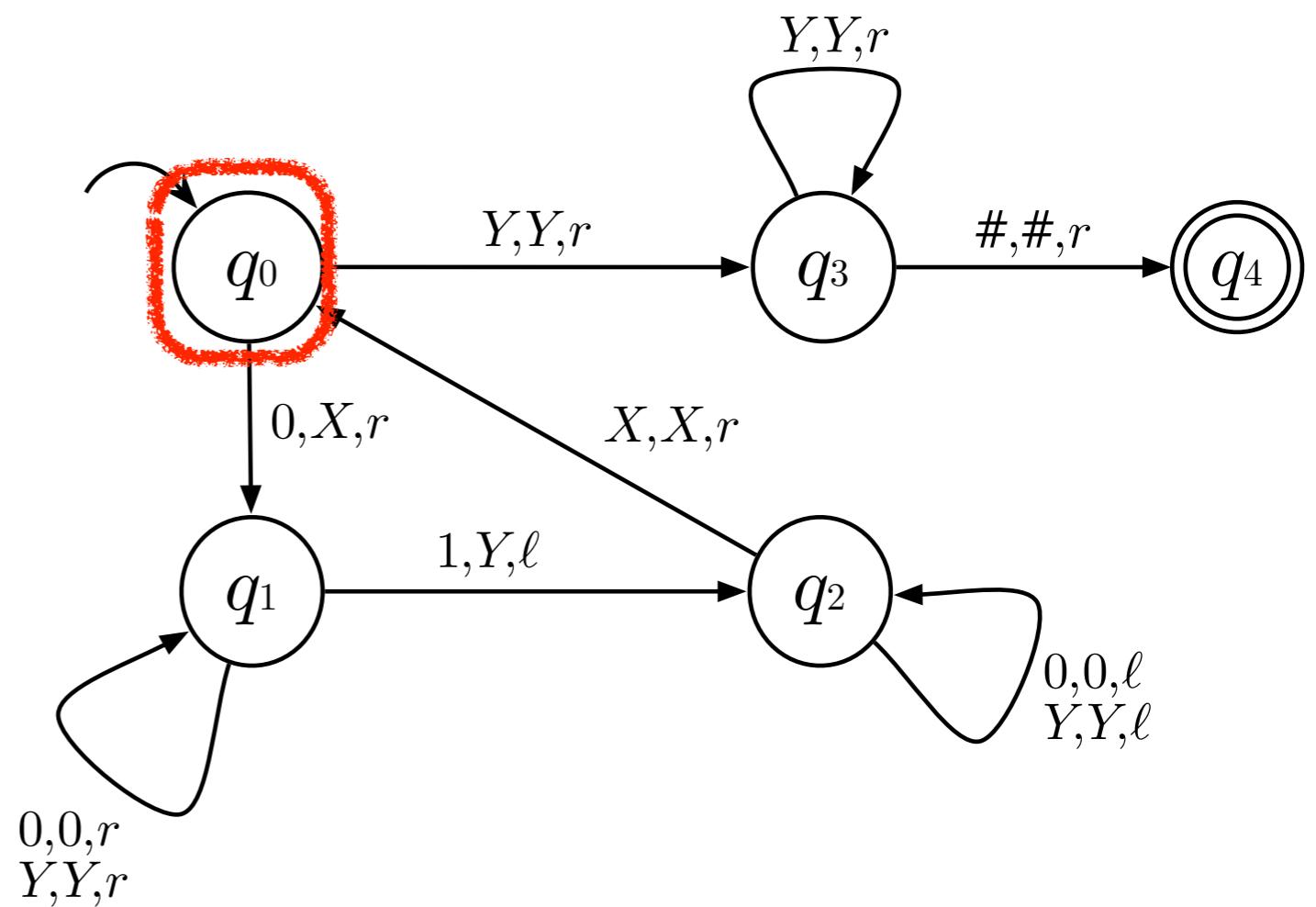
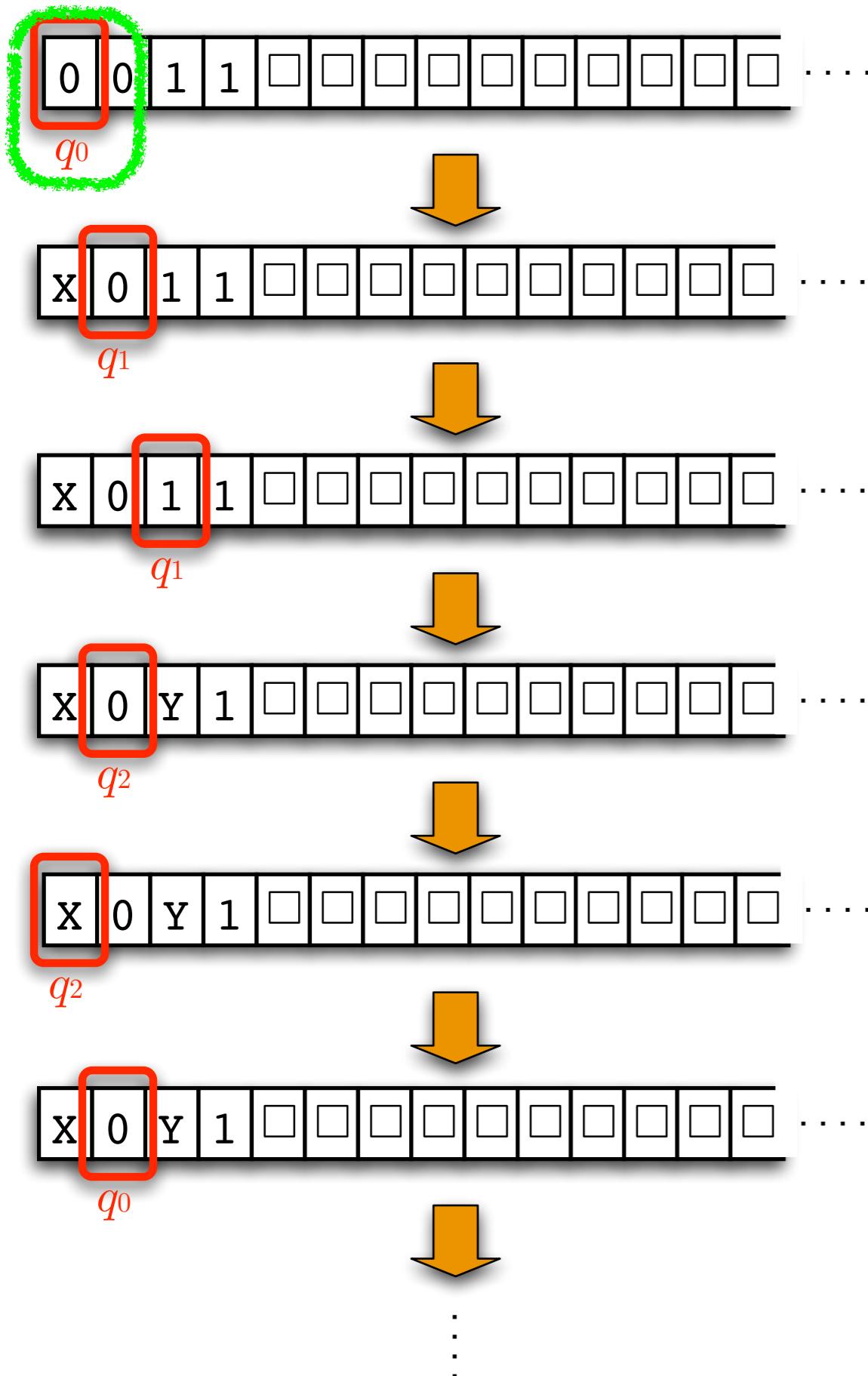
wie arbeitet diese DTM?



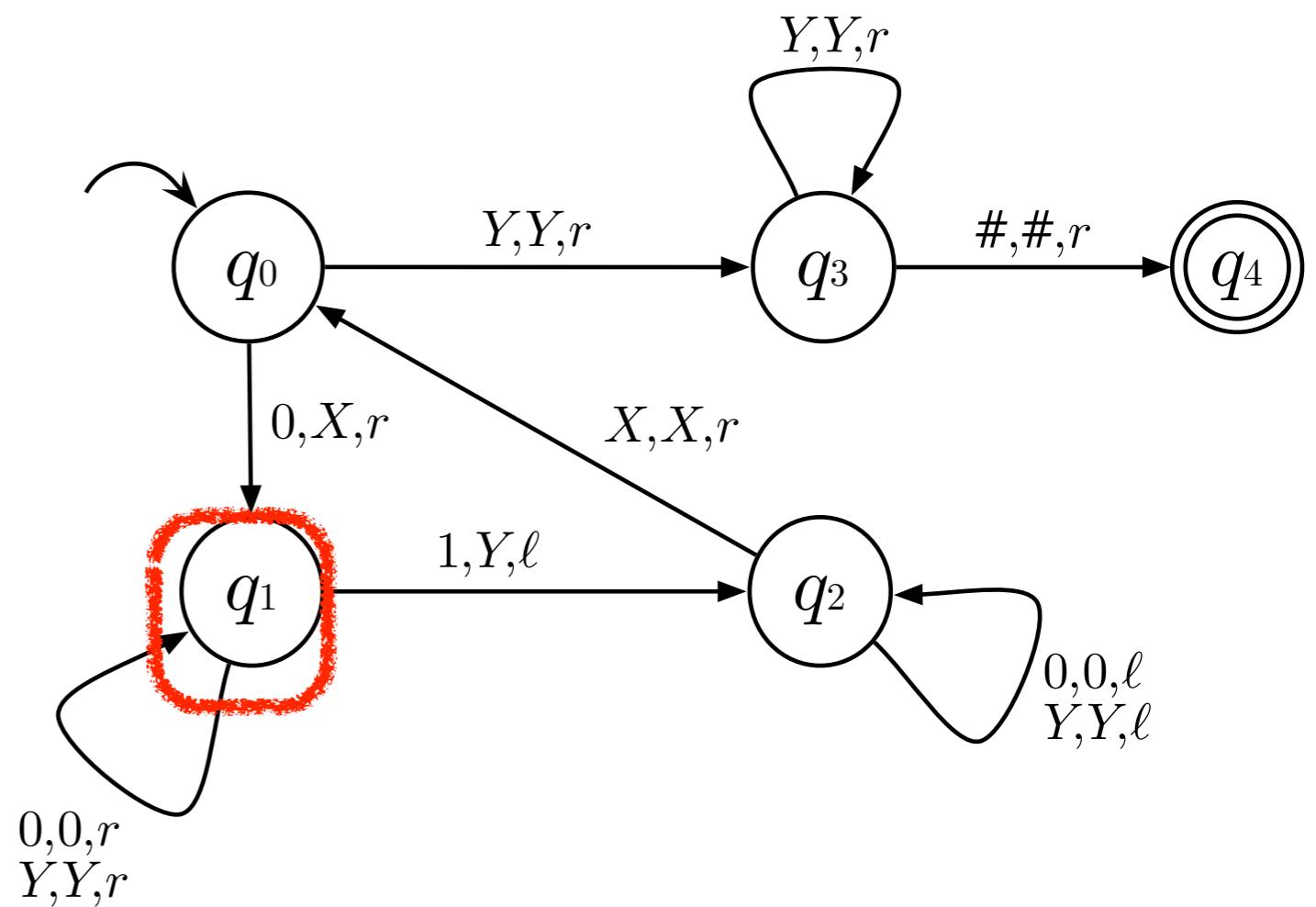
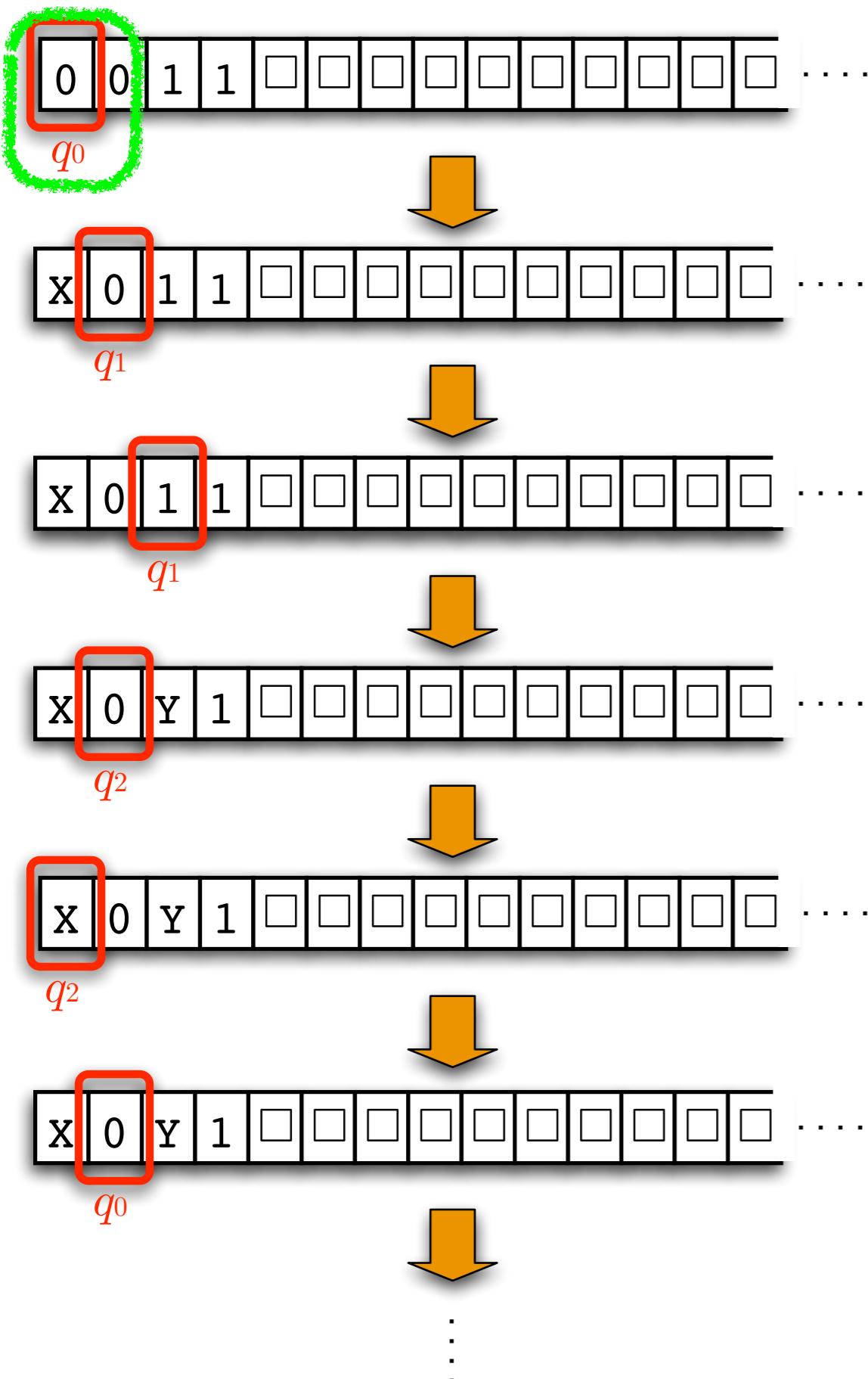
wie arbeitet diese DTM?



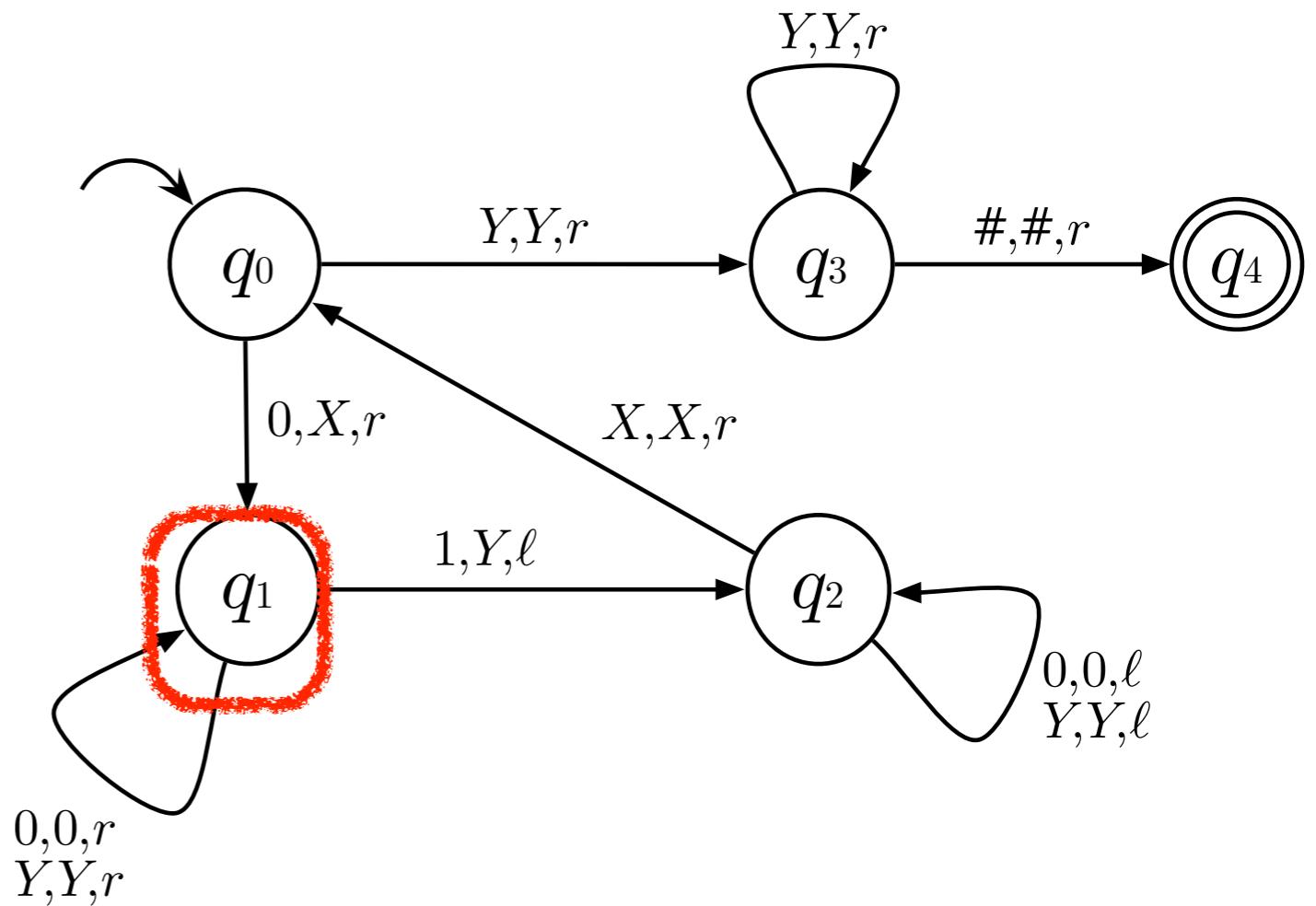
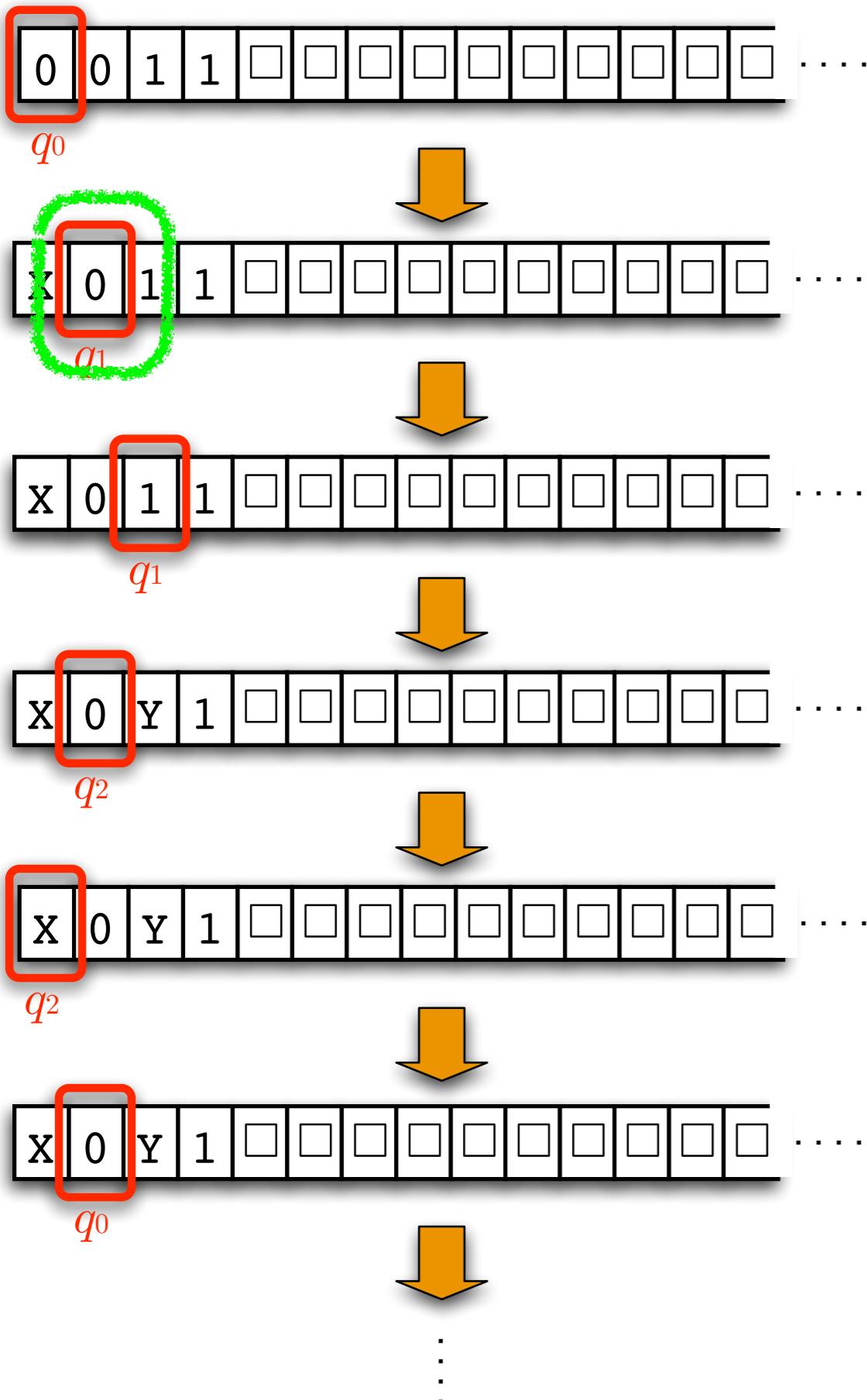
wie arbeitet diese DTM?



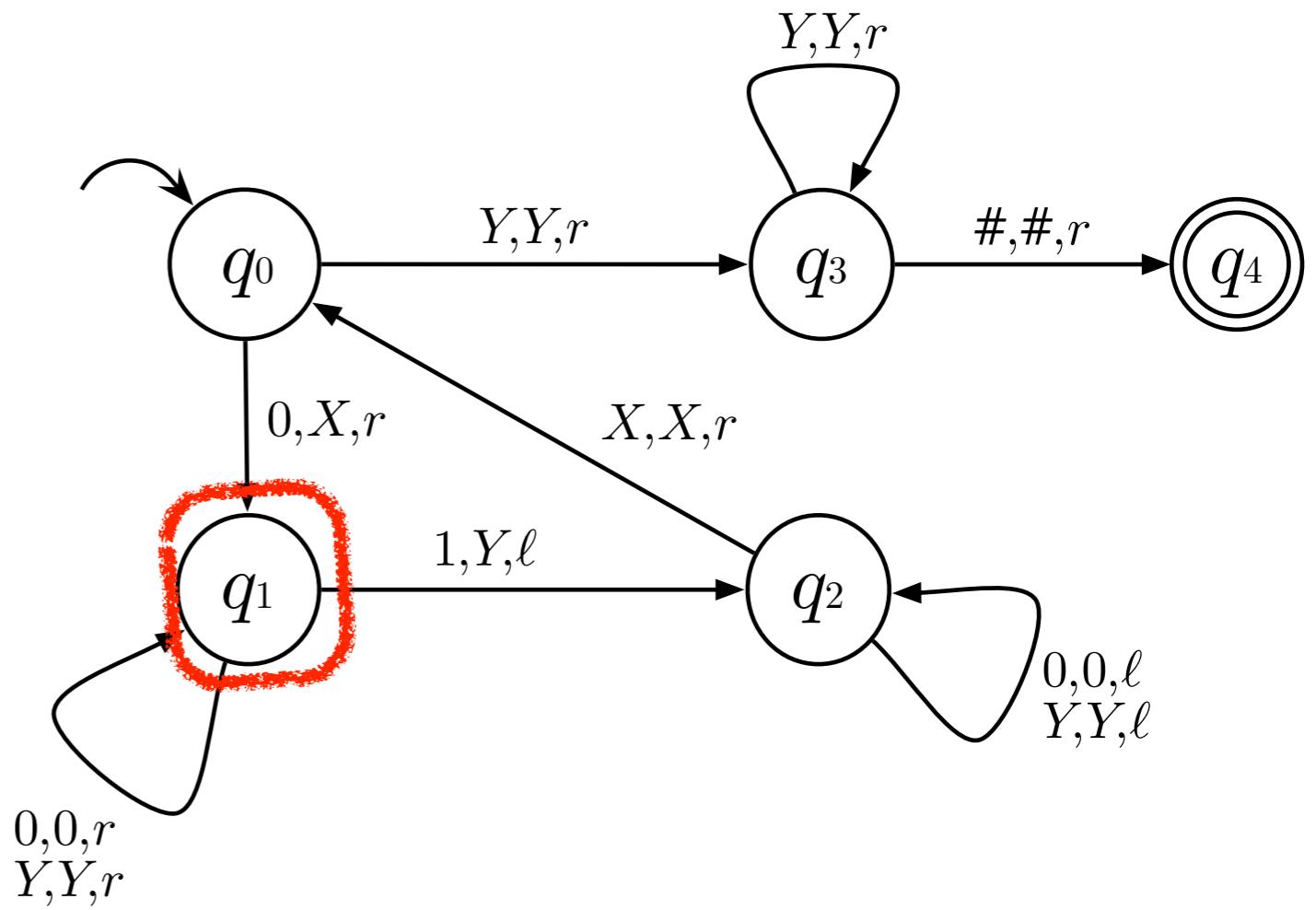
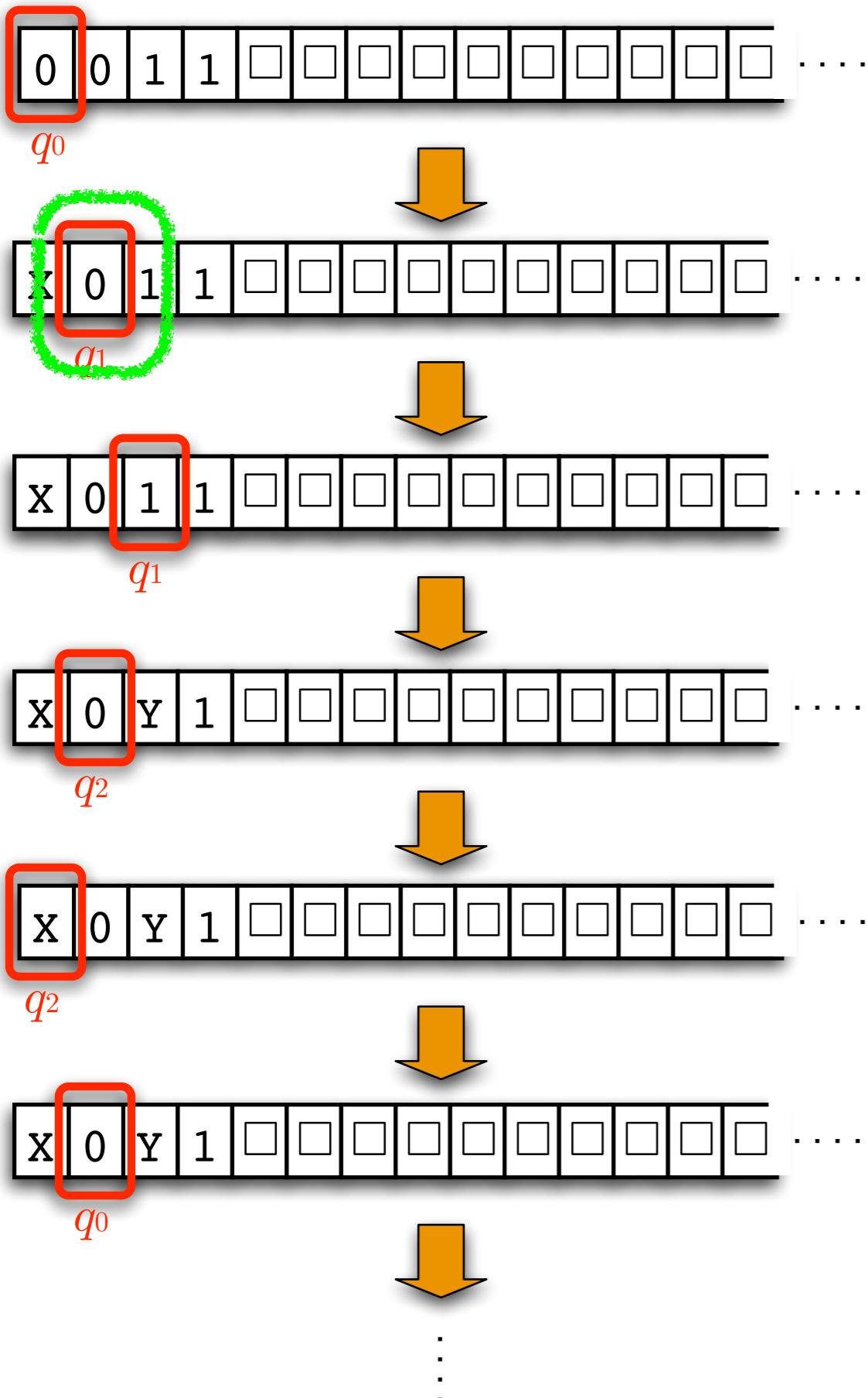
wie arbeitet diese DTM?



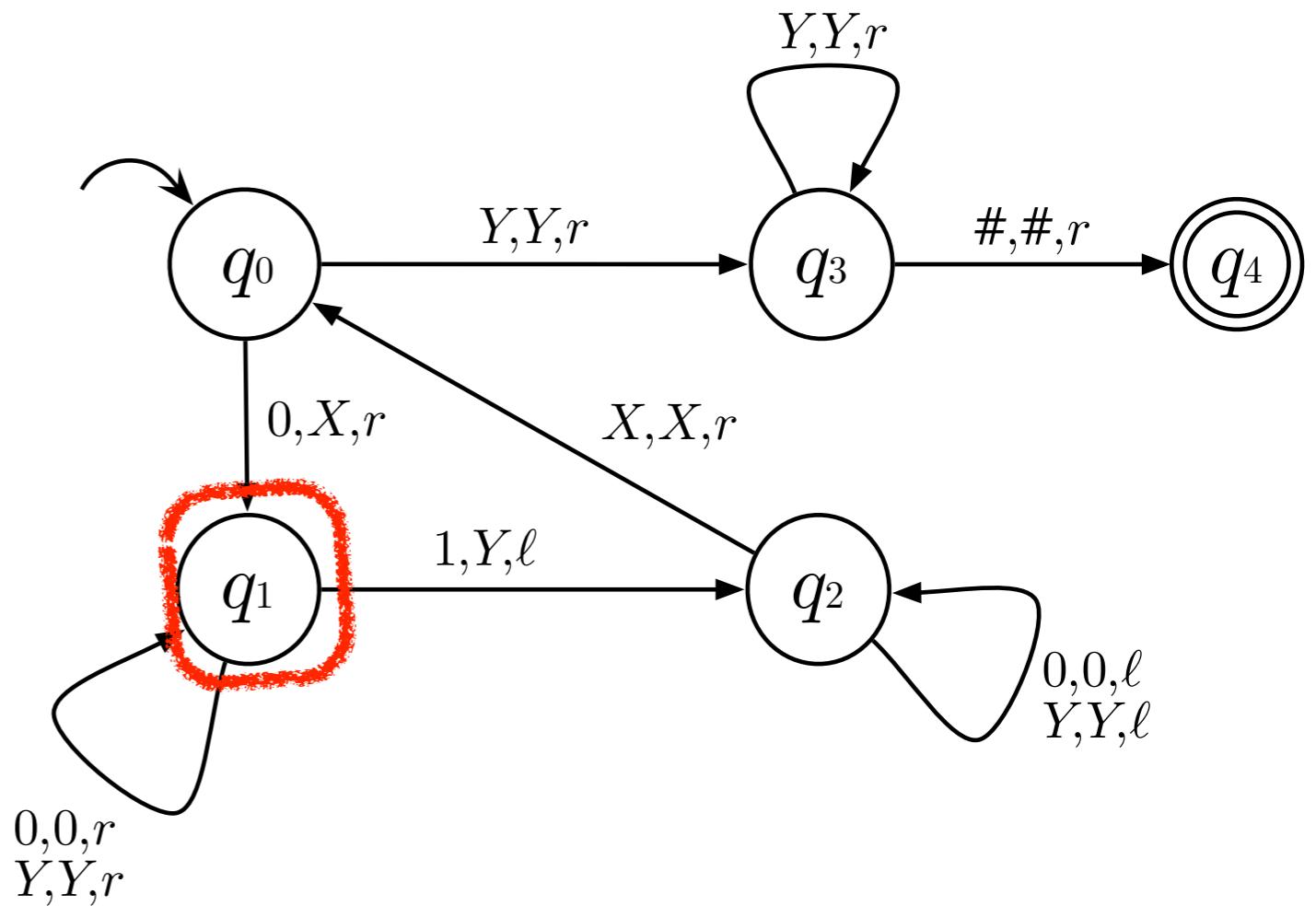
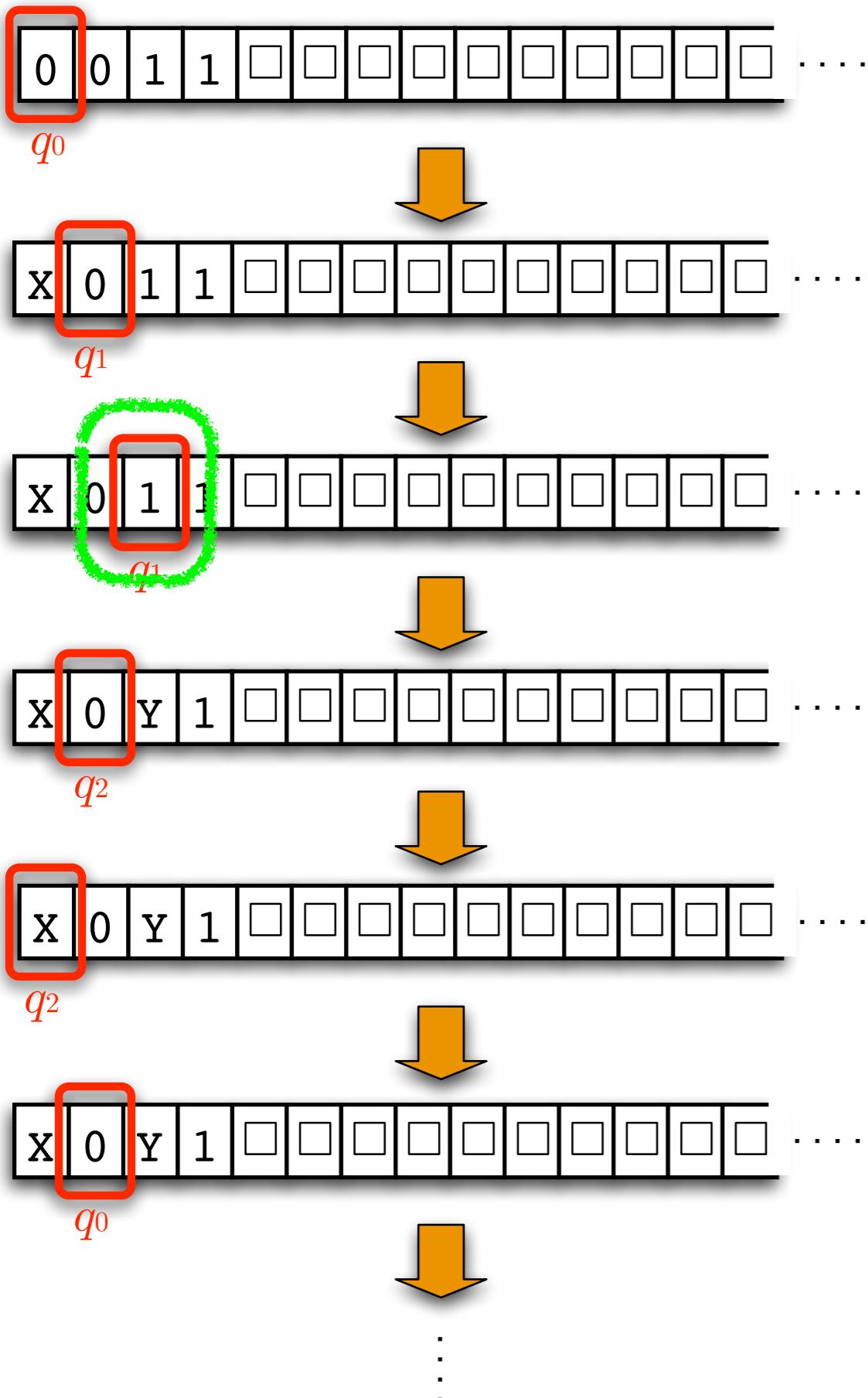
wie arbeitet diese DTM?



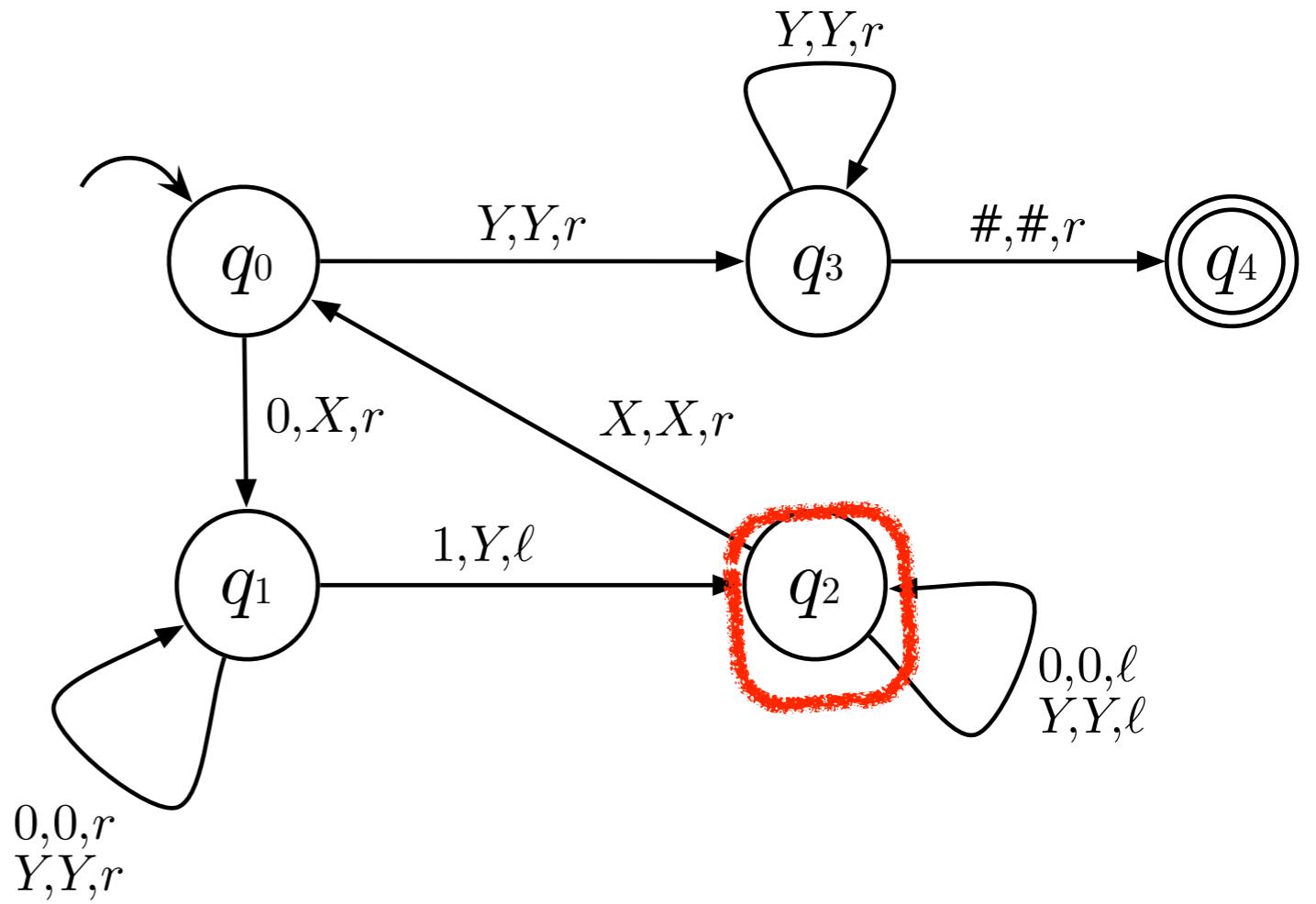
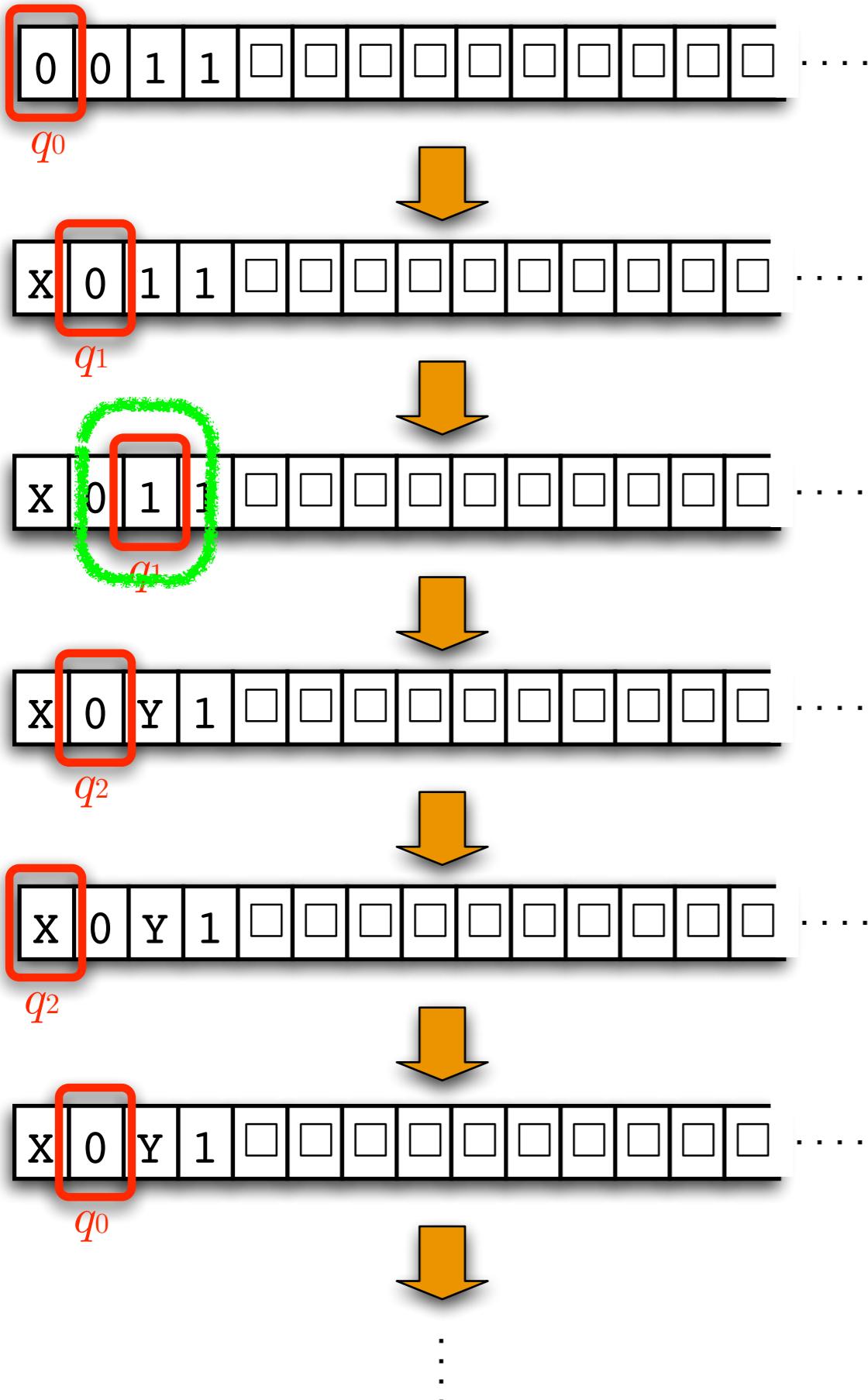
wie arbeitet diese DTM?



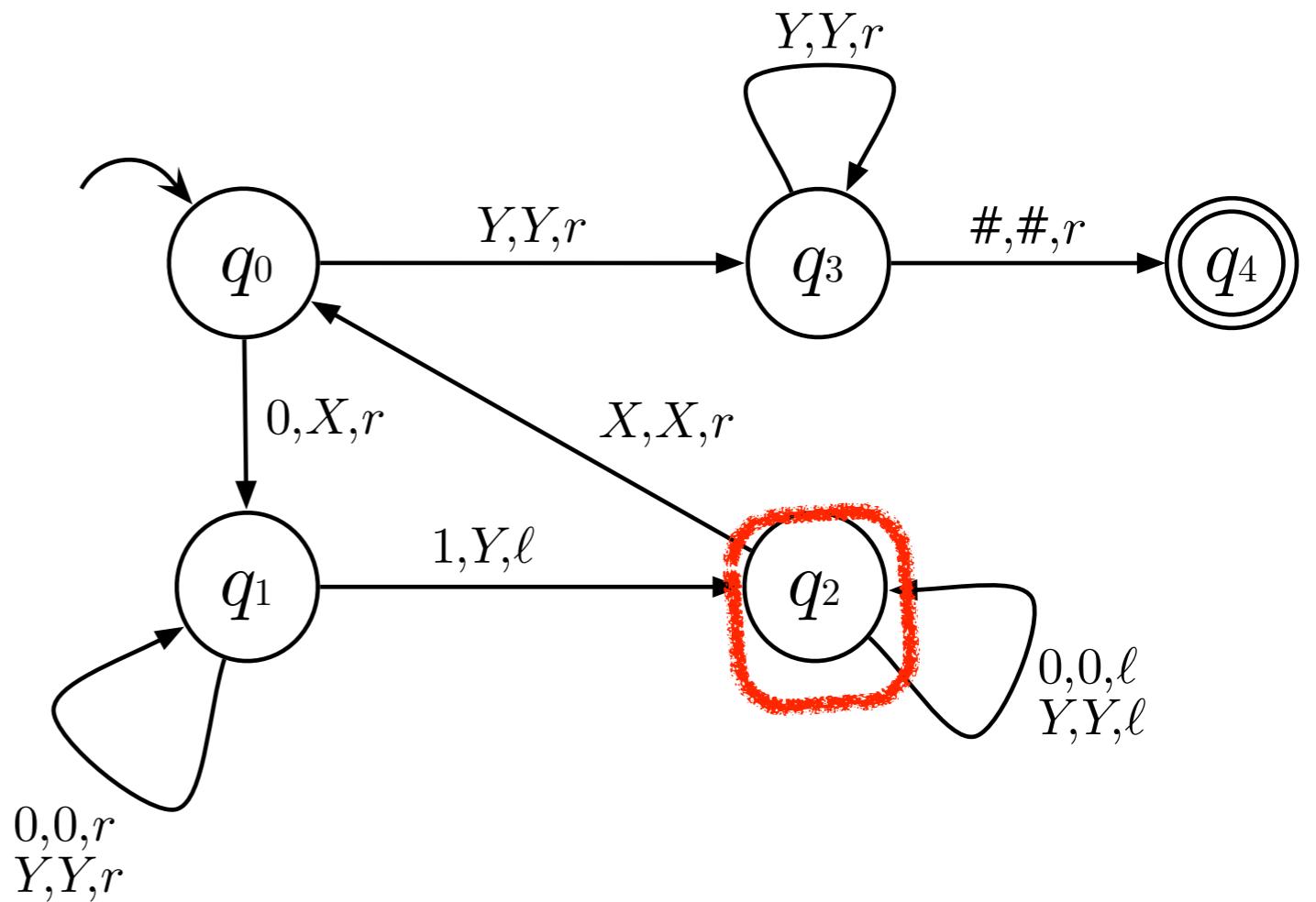
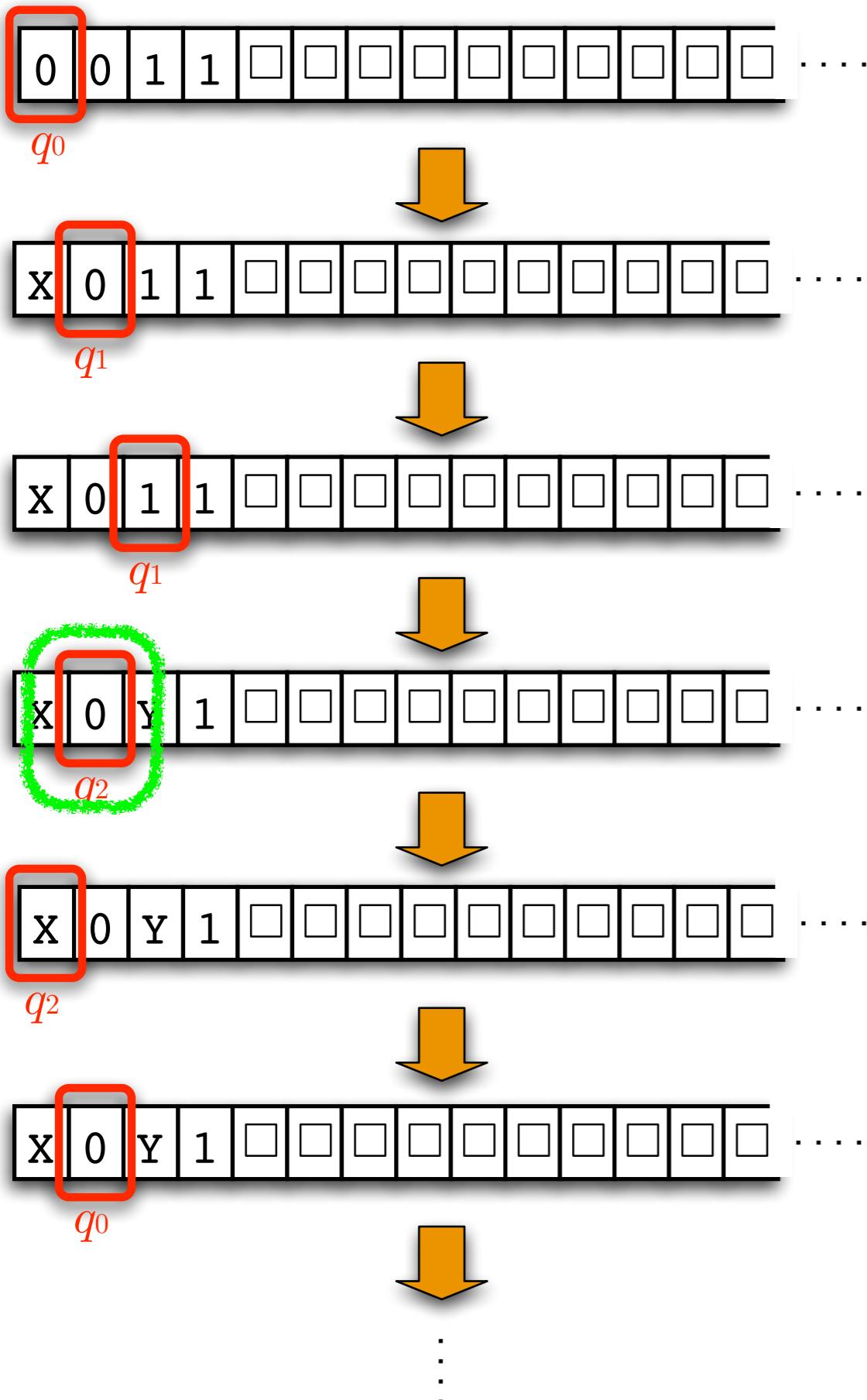
wie arbeitet diese DTM?



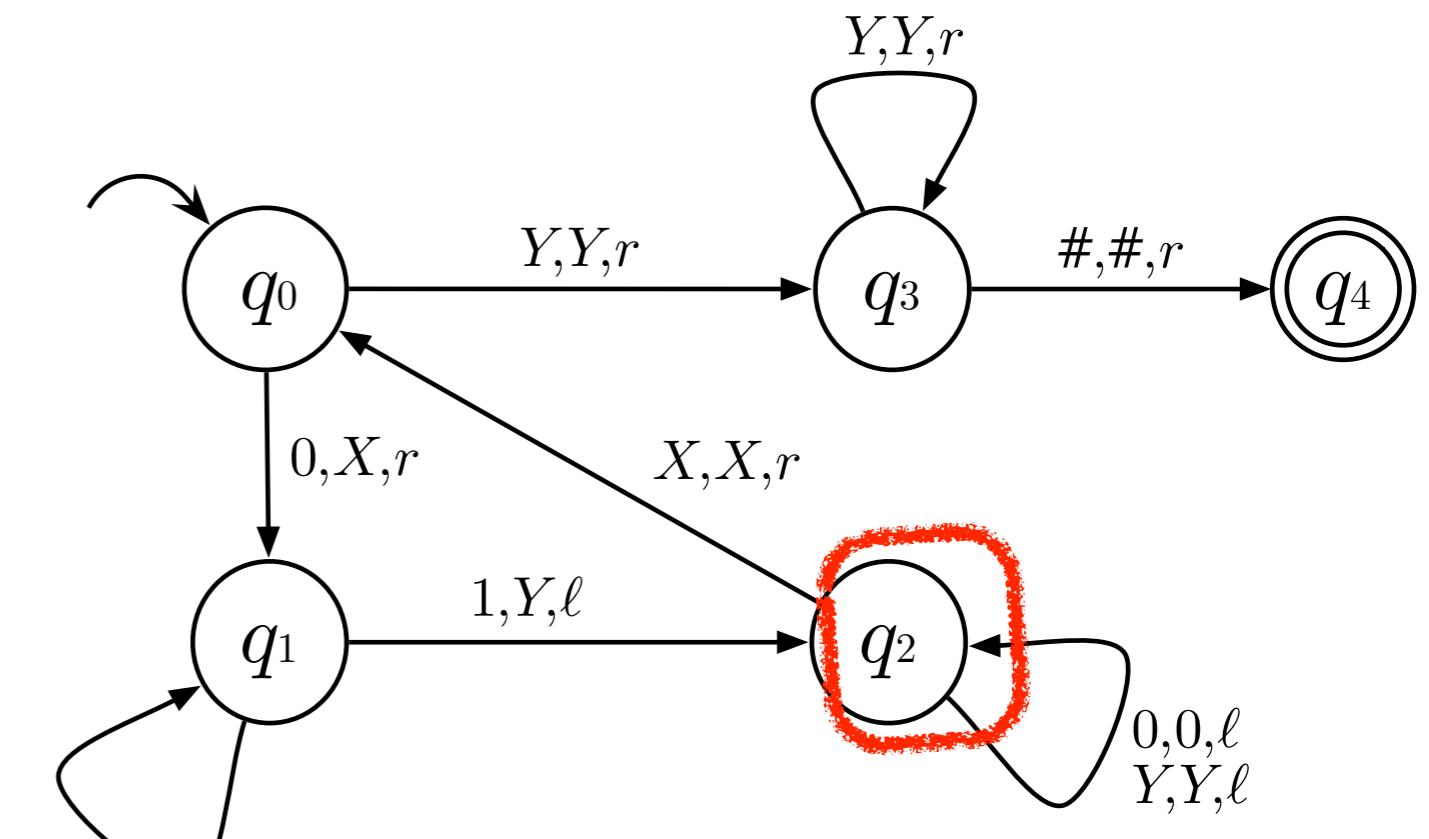
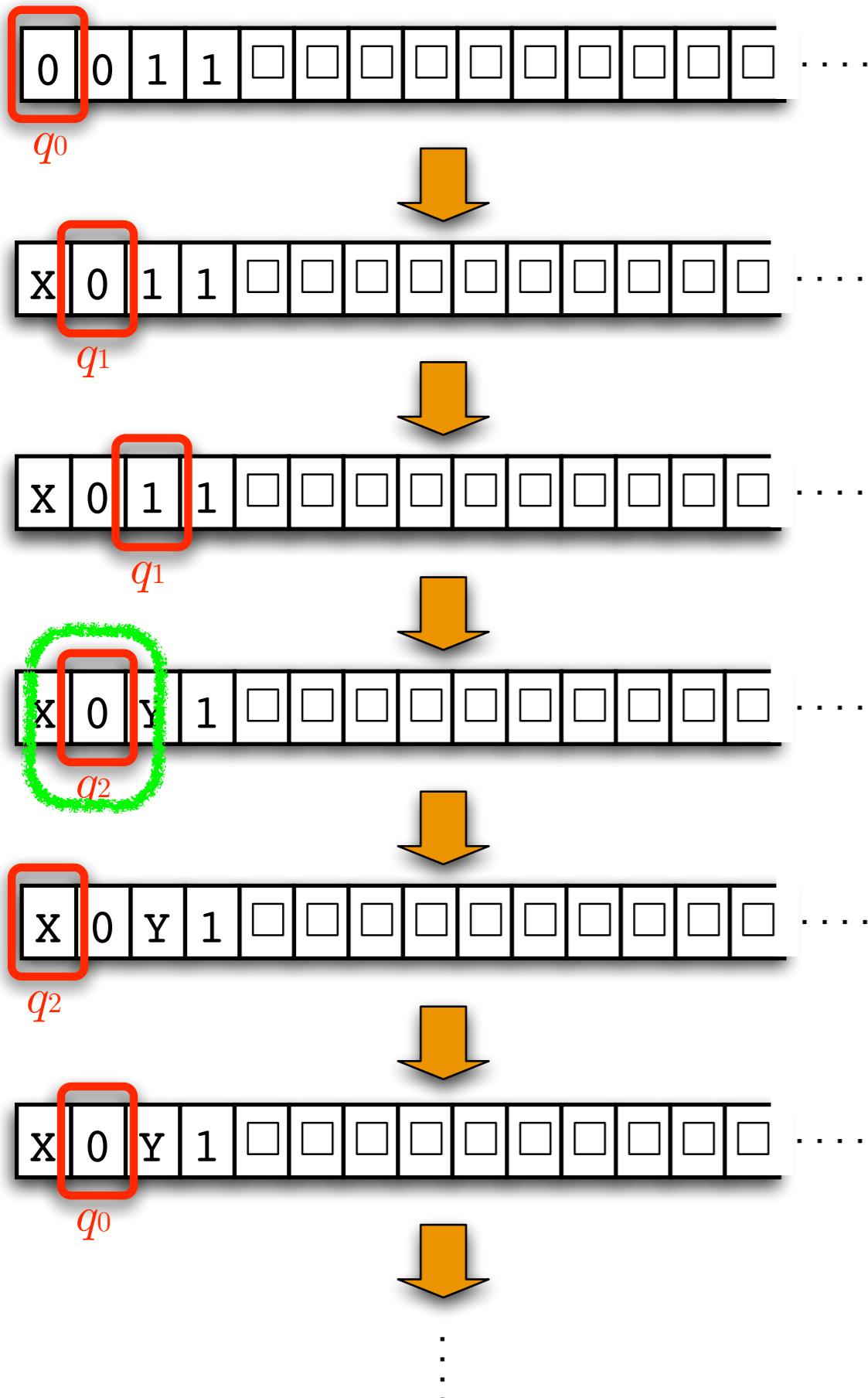
wie arbeitet diese DTM?



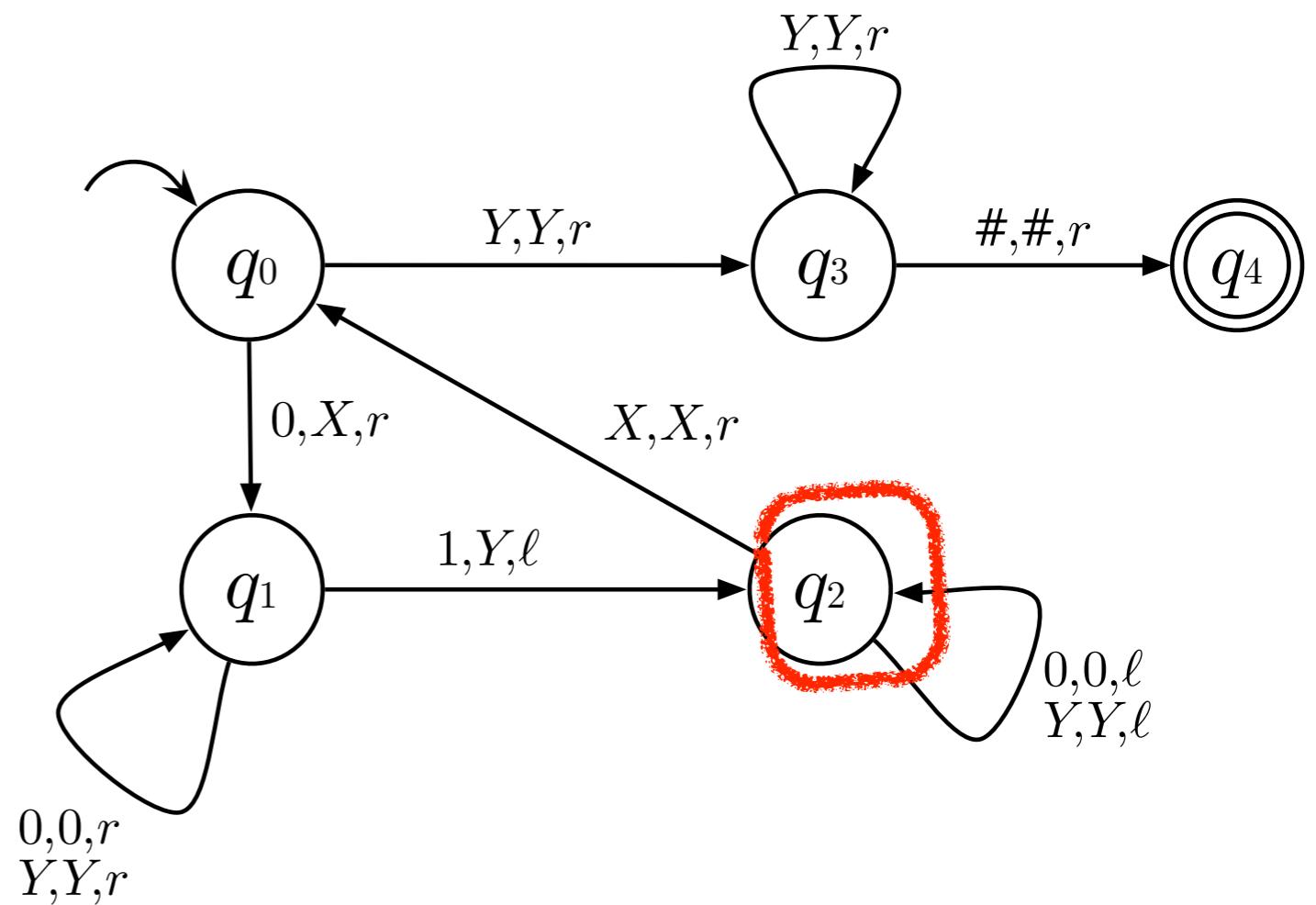
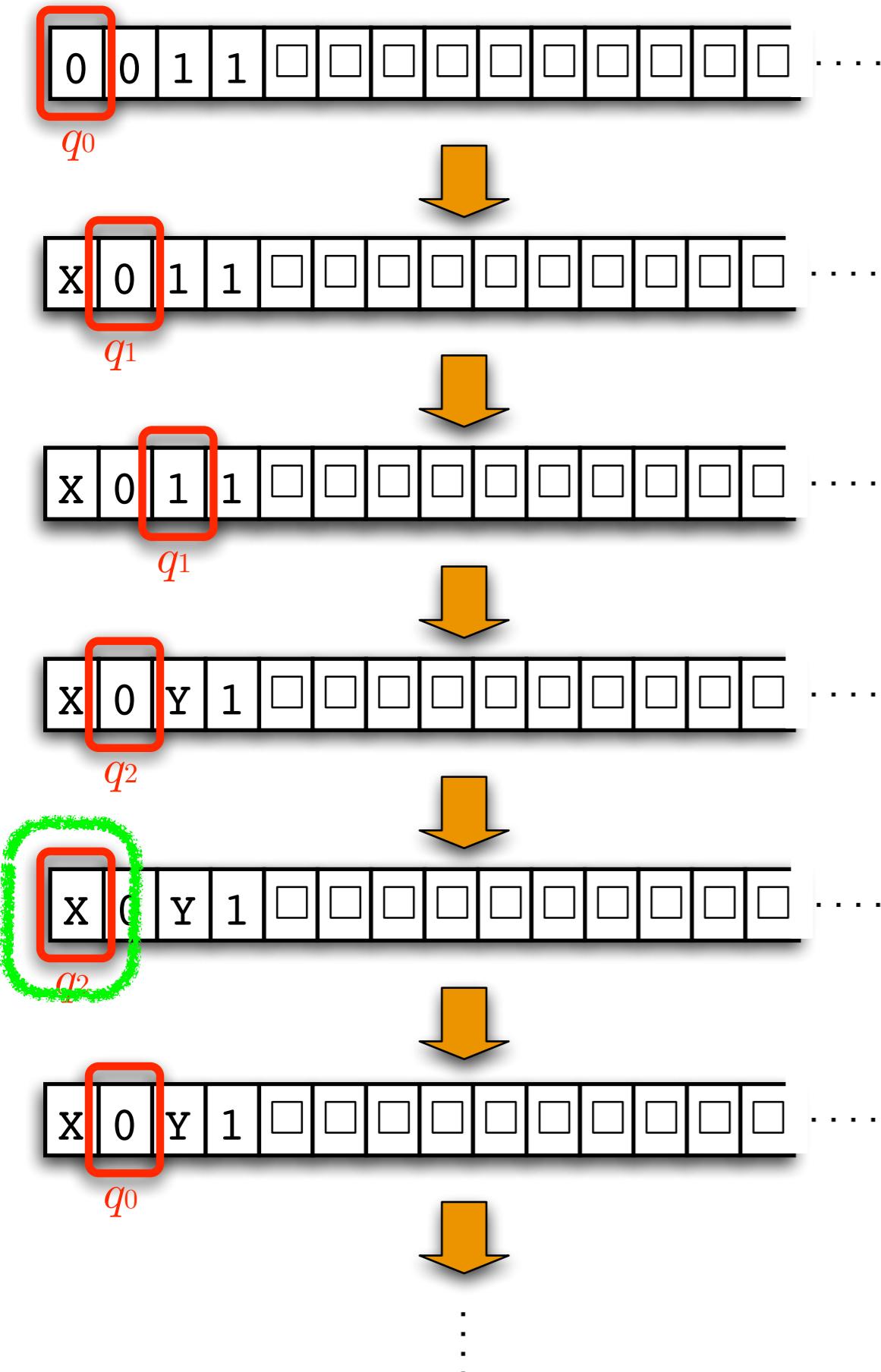
wie arbeitet diese DTM?



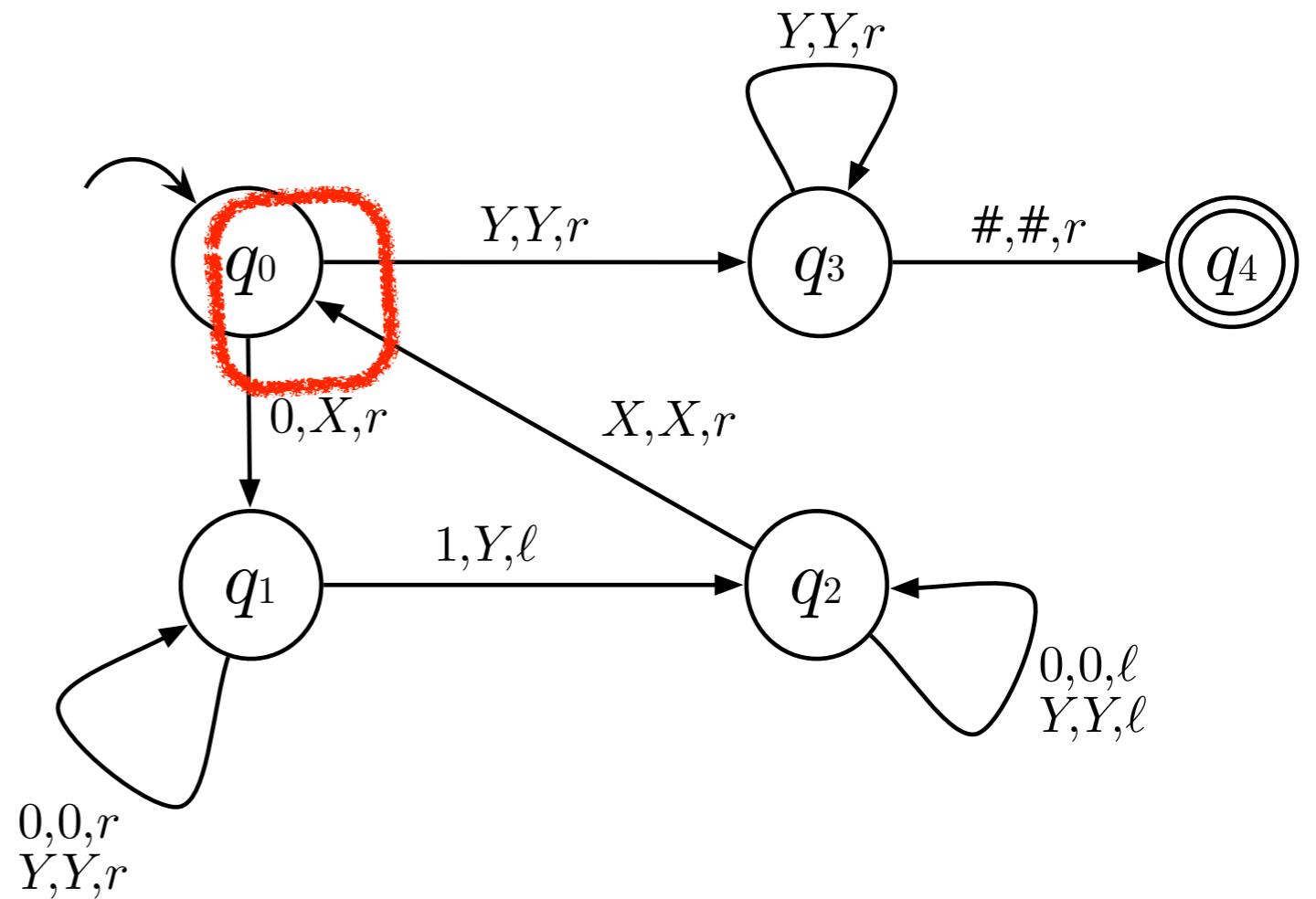
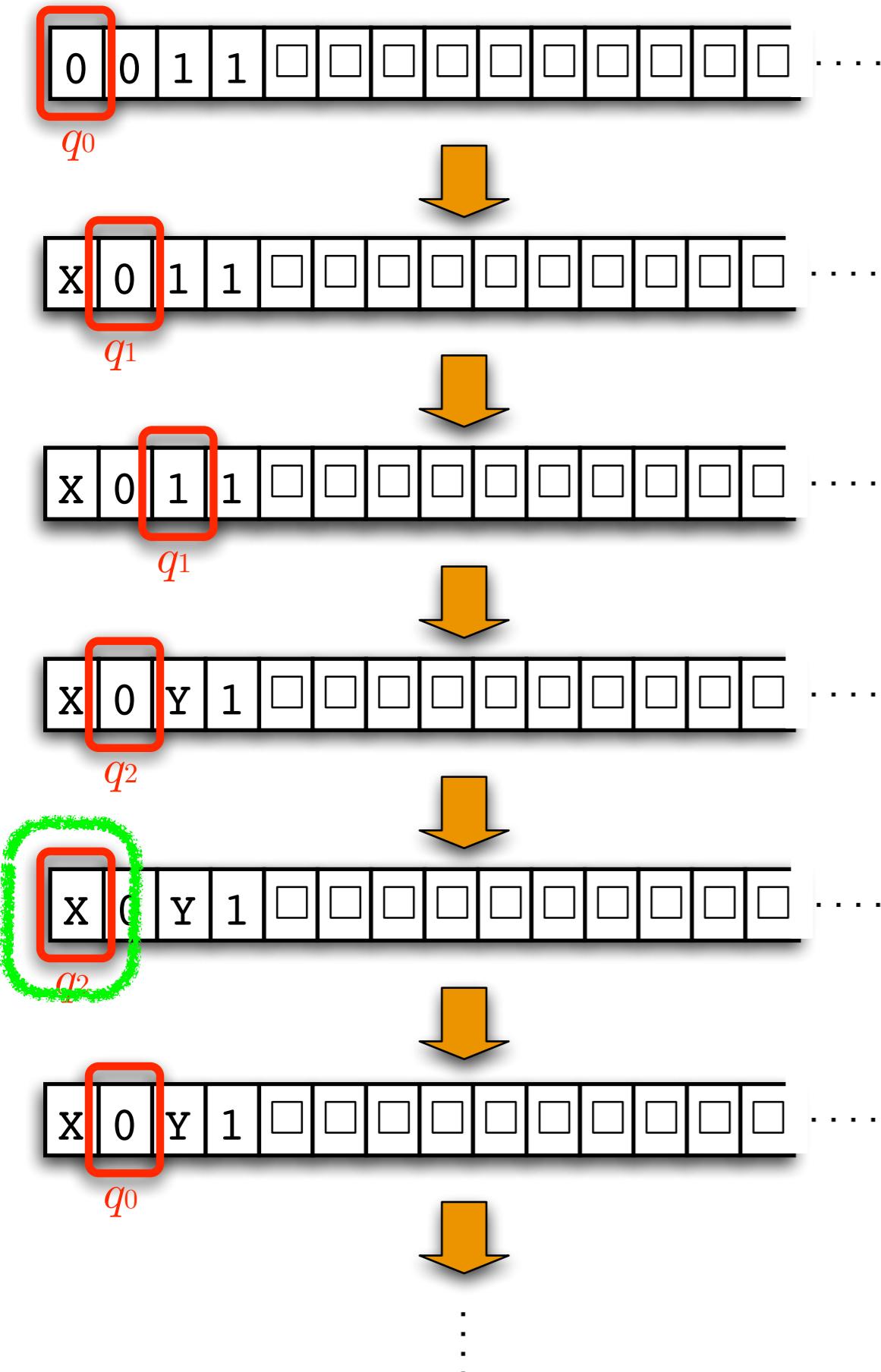
wie arbeitet diese DTM?



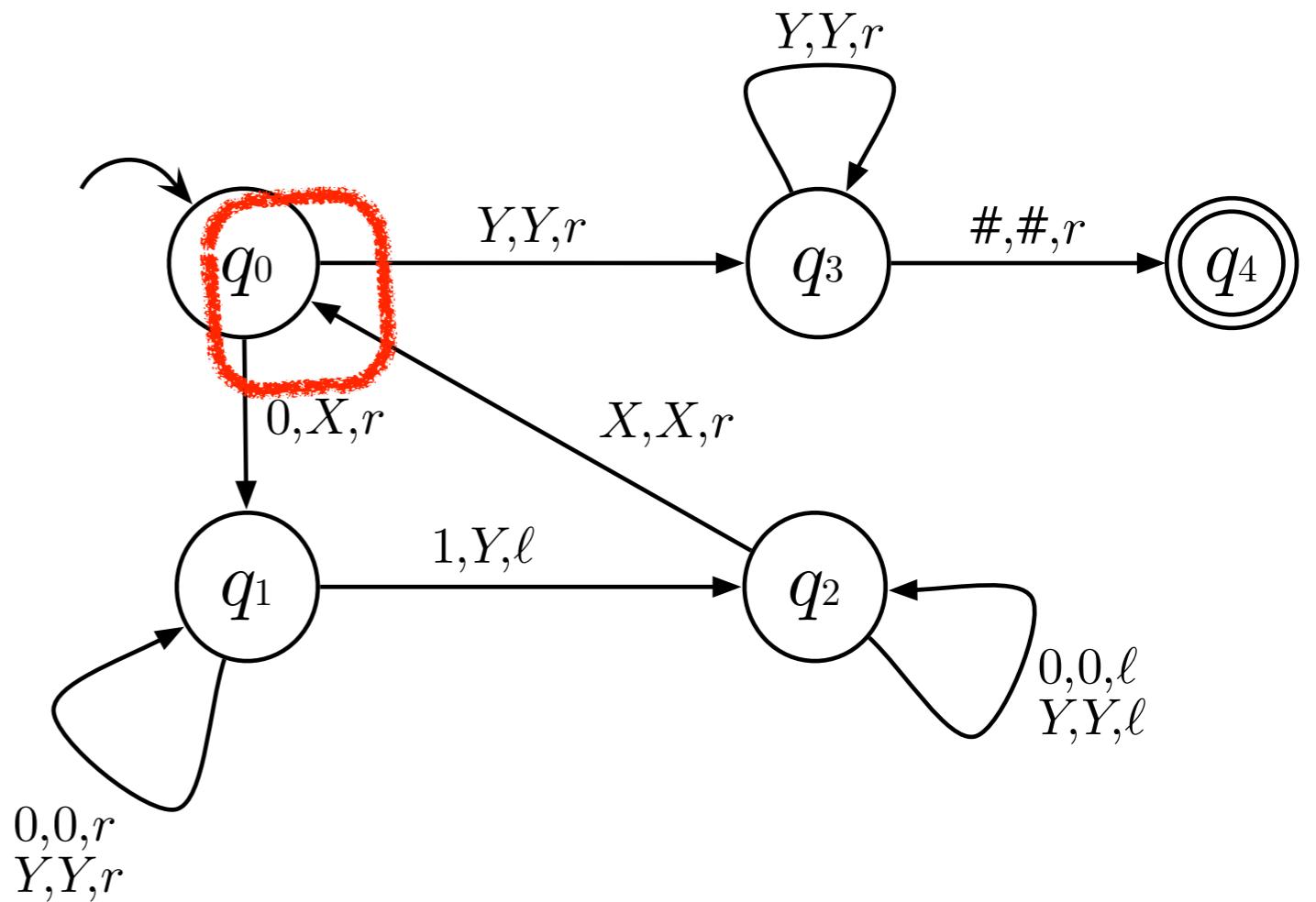
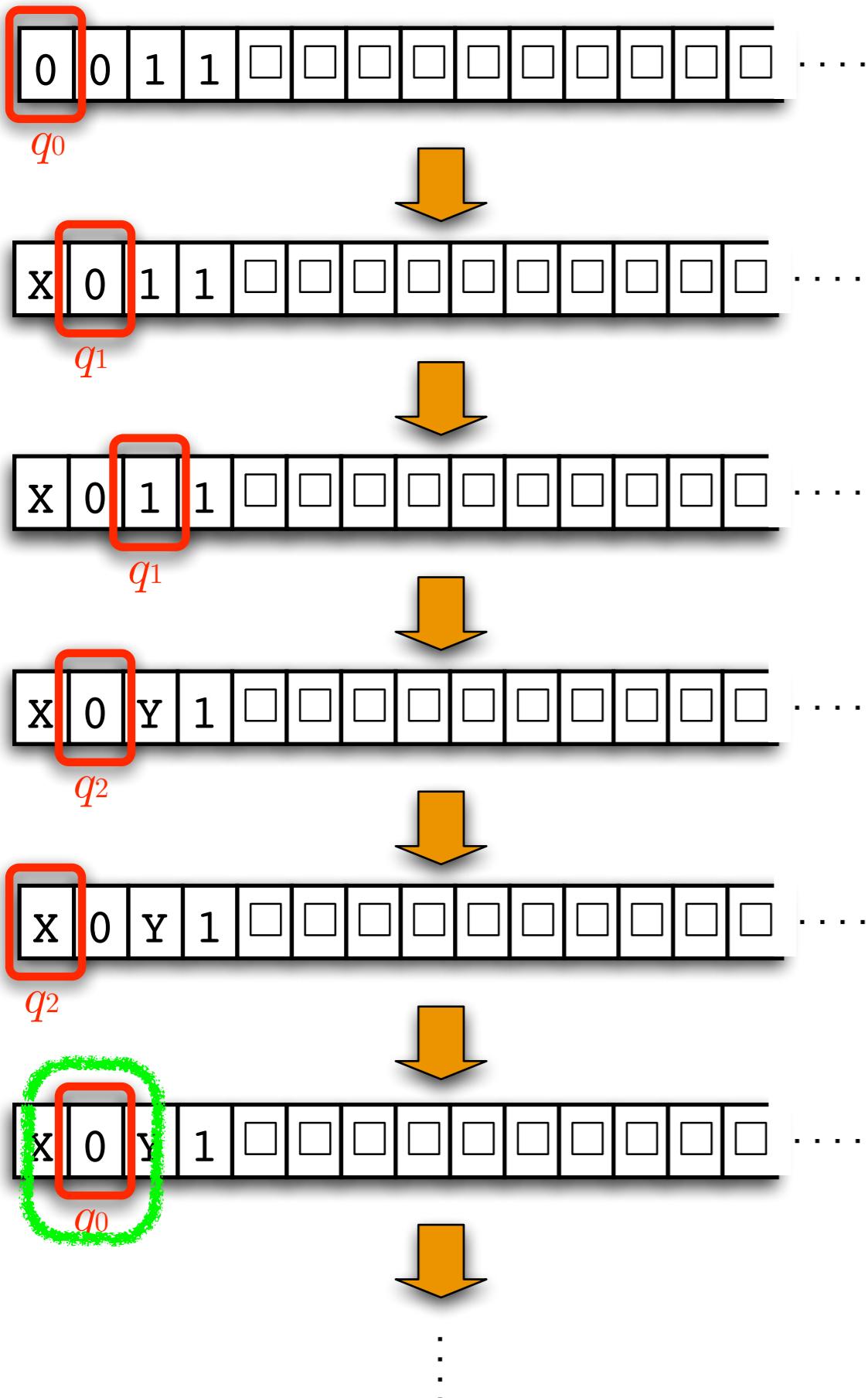
wie arbeitet diese DTM?



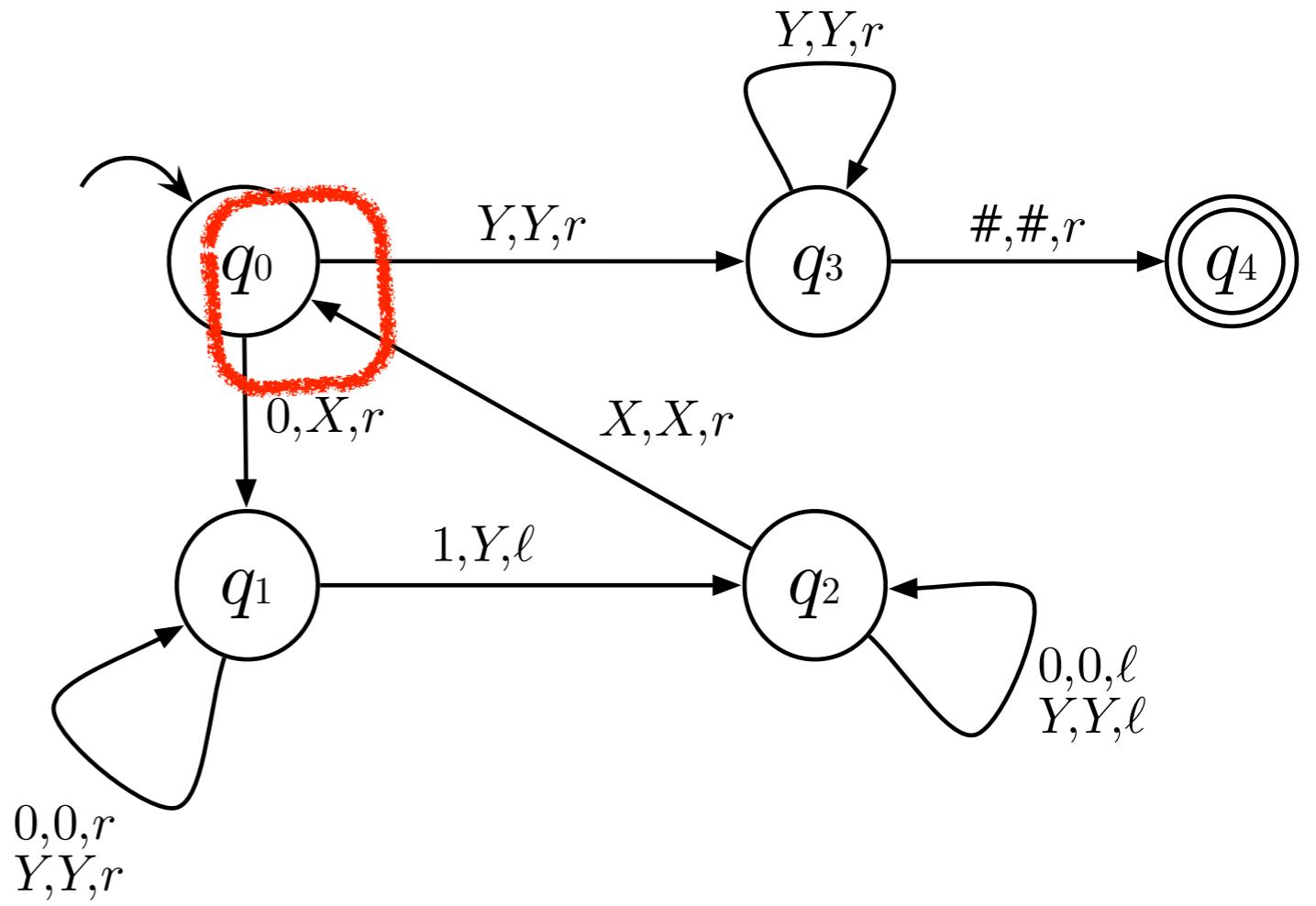
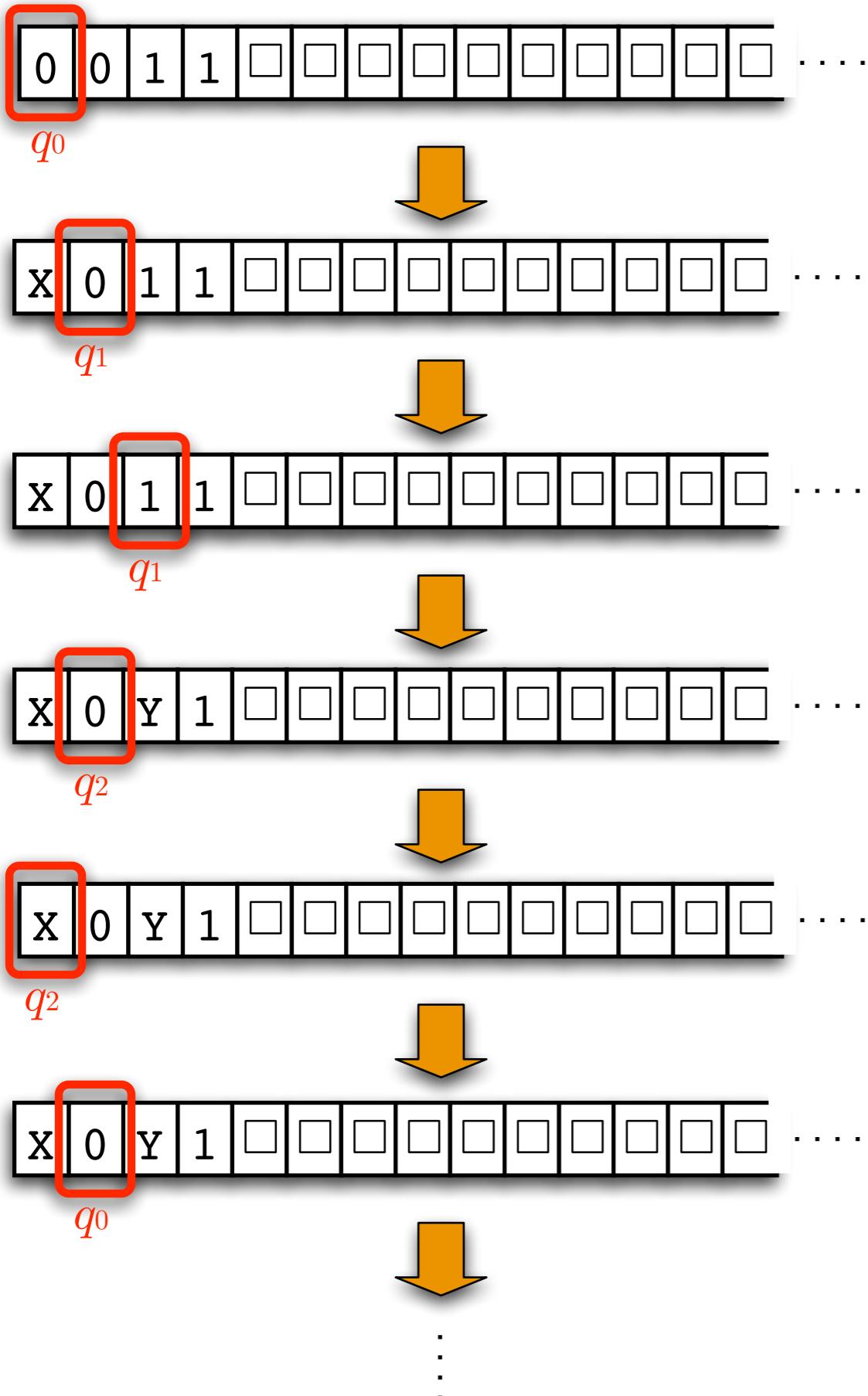
wie arbeitet diese DTM?



wie arbeitet diese DTM?



wie arbeitet diese DTM?



Konfiguration = vollständige Systembeschreibung

Allgemein wird ein Systemzustand vollständig beschrieben durch:

- Spezifikation der Struktur des Systems
(die DTM)
- Angabe des Systemzustandes
(der Zustand der endlichen Steuerung und die Kopfposition)
- Angabe des Speicherinhaltes
(die Bandinschrift)

⇒ Konfiguration einer DTM : $(u, q, v) \in \Gamma^* \times Q \times \Gamma^*$

Statt (u, q, v) schreiben wir uqv .

(Dies ist wegen $Q \cap \Gamma = \emptyset$ unproblematisch.)

Konfigurationsrelation

Definition 18.5:

$w \in \Gamma^* \cdot Q \cdot \Gamma^+$ heißt **Konfiguration** gdw.

- $w = upv$ mit $u \in \Gamma^*$, $v \in \Gamma^+$ und $p \in Q$:
- T befindet sich im Zustand p , die Bandinschrift ist uv und der Kopf steht auf dem ersten Zeichen von v .
- Falls $v \neq \#$, dann ist $v \in \Gamma^* \cdot (\Gamma \setminus \{\#\})$, d.h. ganz rechts in v steht nicht das Symbol $\#$. In jedem Fall sind rechts von v nur noch $\#$'s auf dem Band.
- Entsprechend für u : $(u \neq \epsilon) \implies (u \in (\Gamma \setminus \{\#\})\Gamma^*)$, d.h. u beginnt nicht mit $\#$ und links von u sind nur noch $\#$'s.

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die **Menge aller Konfigurationen** der Turing-Maschine T .
- Für eine $DTM T$ ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqzv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die **Menge aller Konfigurationen** der Turing-Maschine T .
- Für eine $DTM T$ ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

bzw. $uq\#$, falls $yzv \in \{\#\}^*$

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die Menge aller Konfigurationen der Turing-Maschine T .
- Für eine DTM T ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyqzv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

bzw. $uq\#$ falls $yzv \in \{\#\}^*$

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die **Menge aller Konfigurationen** der Turing-Maschine T .
- Für eine $DTM T$ ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

bzw. $uq\#$, falls $yzv \in \{\#\}^*$

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die **Menge aller Konfigurationen** der Turing-Maschine T .
- Für eine $DTM T$ ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqzv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

Konfigurationsrelation

Definition 18.6:

- $CONF(T)$ bezeichnet die Menge aller Konfigurationen der Turing-Maschine T .
- Für eine DTM T ist die **Schrittrelation** $\vdash_T \subseteq CONF(T) \times CONF(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzq\cancel{v}, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

bzw. $uyzq\#$, falls $v = \epsilon$

Konfigurationsrelation

Definition 18.6:

- $\text{CONF}(T)$ bezeichnet die Menge aller Konfigurationen der Turing-Maschine T .
- Für eine DTM T ist die **Schrittrelation** $\vdash_T \subseteq \text{CONF}(T) \times \text{CONF}(T)$ erklärt durch:

$w \vdash_T w'$ gilt genau dann, wenn für $w = uypxv$ mit $u, v \in \Gamma^*$, $x, y, z \in \Gamma$ und $p, q \in Q$, folgendes gilt:

$$w' = \begin{cases} uqyzv, & \text{falls } \delta(p, x) = (q, z, l) \\ uyzqv, & \text{falls } \delta(p, x) = (q, z, r) \\ uyzqzv, & \text{falls } \delta(p, x) = (q, z, -) \end{cases}$$

bzw. $uyzq\#$, falls $v = \epsilon$

Erfolgsrechnung

Definition 18.7:

- Mit \vdash_T^* wird die reflexive, transitive Hülle von \vdash_T bezeichnet.
- Folgen von Konfigurationen $k_1, k_2, k_3, \dots, k_i, k_{i+1}, \dots$, die in der Relation $k_1 \vdash k_2 \vdash \dots \vdash k_i \vdash k_{i+1} \vdash \dots$ stehen, heißen **Rechnungen**.
- Endliche Rechnungen $k_1 \vdash k_2 \vdash \dots \vdash k_t$ heißen **Erfolgsrechnungen**, wenn $k_1 \in \{q_0\} \cdot \Gamma^*$ und $k_t \in \Gamma^* \cdot F \cdot \Gamma^*$ ist.

In einigen Büchern wird das längere Symbol $\overline{\vdash}_T^*, \overline{\vdash}^*, \overline{\vdash}$ benutzt!

Rechnungen

Rechnungen

Rechnungen können:

Rechnungen

Rechnungen können:



endlich sein oder niemals aufhören.

Rechnungen

Rechnungen können:

- endlich sein oder niemals aufhören.
- * Endliche Rechnungen sind **akzeptierend**, d.h letzter erreichter Zustand ist in F, oder

Rechnungen

Rechnungen können:

- endlich sein oder niemals aufhören.
- * Endliche Rechnungen sind **akzeptierend**, d.h letzter erreichter Zustand ist in F, oder
- * nicht akzeptierend, d.h. **verwerfend**, wenn letzter erreichter Zustand nicht in F ist.

Rechnungen

Rechnungen können:

- endlich sein oder niemals aufhören.
- * Endliche Rechnungen sind **akzeptierend**, d.h letzter erreichter Zustand ist in F, oder
- * nicht akzeptierend, d.h. **verwerfend**, wenn letzter erreichter Zustand nicht in F ist.
- * Unendliche Rechnungen können unendlich viele (nicht notwendig verschiedene) Konfigurationen durchlaufen...

Rechnungen

Rechnungen können:

- endlich sein oder niemals aufhören.
- * Endliche Rechnungen sind **akzeptierend**, d.h letzter erreichter Zustand ist in F, oder
- * nicht akzeptierend, d.h. **verwerfend**, wenn letzter erreichter Zustand nicht in F ist.
- * Unendliche Rechnungen können unendlich viele (nicht notwendig verschiedene) Konfigurationen durchlaufen...
- * oder eine letzte Konfiguration nie verlassen und unendlich oft durchlaufen (= Fangzustand).

Rechnungen

Rechnungen können:

- endlich sein oder niemals aufhören.
- * Endliche Rechnungen sind **akzeptierend**, d.h letzter erreichter Zustand ist in F, oder
- * nicht akzeptierend, d.h. **verwerfend**, wenn letzter erreichter Zustand nicht in F ist.
- * Unendliche Rechnungen können unendlich viele (nicht notwendig verschiedene) Konfigurationen durchlaufen...
- * oder eine letzte Konfiguration nie verlassen und unendlich oft durchlaufen (= Fangzustand).

Turing-Berechenbarkeit

Definition 18.12:

Seien Σ und Λ Alphabete.

Eine partielle (Wort-)Funktion $f : \Sigma^* \rightarrow \Lambda^*$ heißt **Turing-berechenbar** oder **partiell rekursiv** genau dann, wenn es eine DTM $T = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ gibt mit:

$$q_0 w \xrightarrow[T]{*} q_e v, \quad \text{für } q_e \in F \text{ und } v \in \Lambda^* \text{ mit } f(w) = v.$$

... der Funktionswert steht *ab der Kopfposition* auf dem Band (abgesehen von nachfolgenden $\#$'s)!

Wenn die DTM T die Eingabe verwirft, oder nie anhält, so soll $f(w) := \perp$ die Undefiniertheit von f anzeigen.

DTM für Addition von 1 (Nachfolgerfunktion)

		$\downarrow q_0$					
...	#	1	3	9	9	#	...
Zellenummer	0	1	2	3	4	5	6

Tabelle gedreht:

Die Nachfolgerfunktion
für natürliche Zahlen
wird beschrieben durch:
 $\text{Succ}(x) := x + 1.$

Für z.B. $x = 1399$ ist
 $\text{Succ}(x) = x + 1 = 1400.$

	q_0	q_1	q_2
0	$q_0, 0, r$	$q_2, 1, -$	$q_2, 0, -$
1	$q_0, 1, r$	$q_2, 2, -$	$q_2, 1, -$
2	$q_0, 2, r$	$q_2, 3, -$	$q_2, 2, -$
3	$q_0, 3, r$	$q_2, 4, -$	$q_2, 3, -$
4	$q_0, 4, r$	$q_2, 5, -$	$q_2, 4, -$
5	$q_0, 5, r$	$q_2, 6, -$	$q_2, 5, -$
6	$q_0, 6, r$	$q_2, 7, -$	$q_2, 6, -$
7	$q_0, 7, r$	$q_2, 8, -$	$q_2, 7, -$
8	$q_0, 8, r$	$q_2, 9, -$	$q_2, 8, -$
9	$q_0, 9, r$	$q_1, 0, l$	$q_2, 9, -$
#	$q_1, \#, l$	$q_2, 1, -$	$q_2, \#, -$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

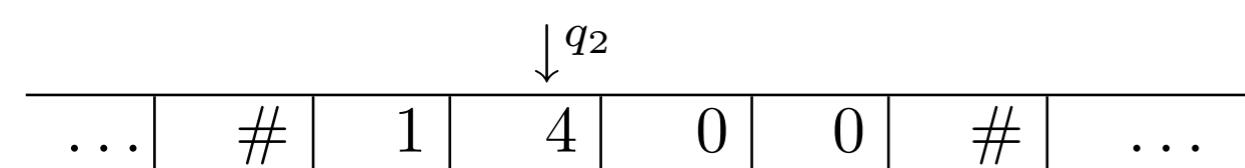
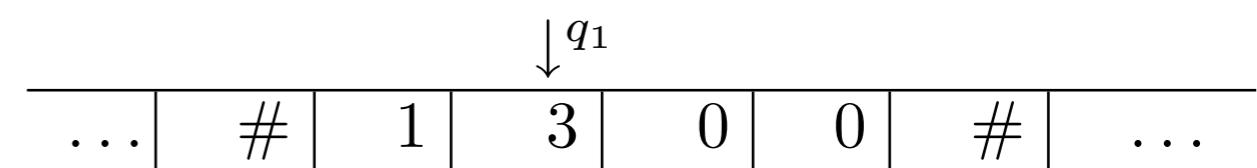
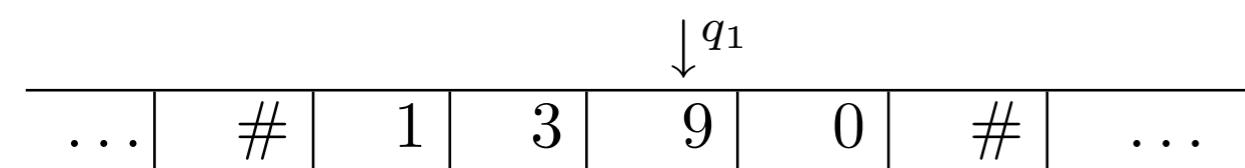
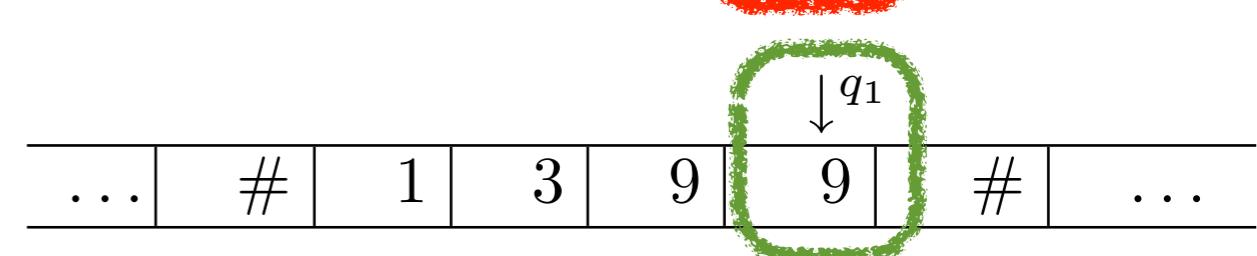
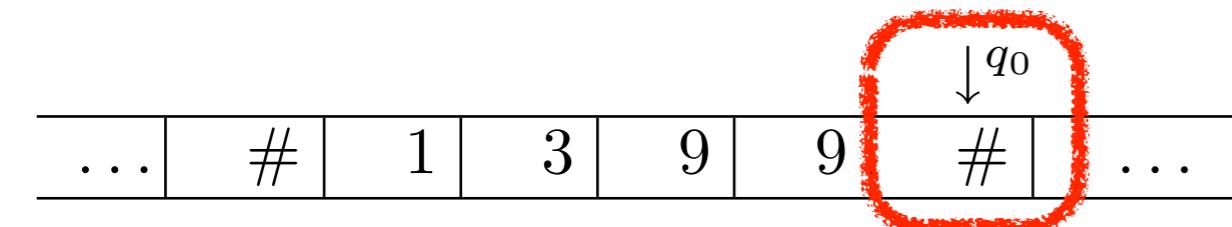
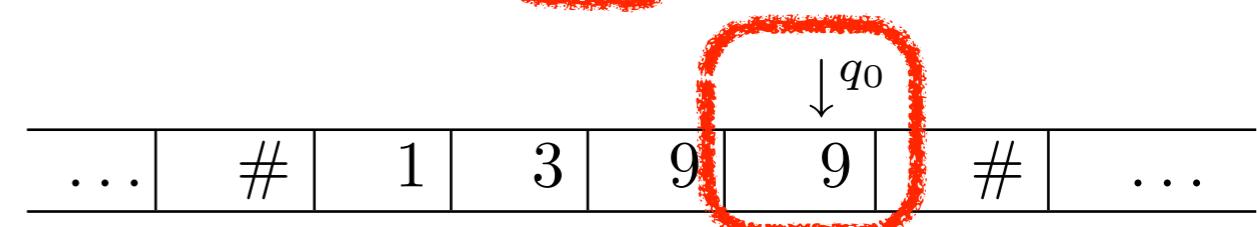
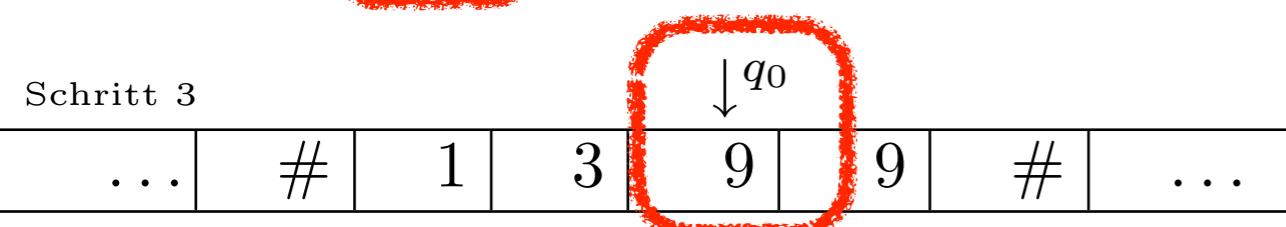
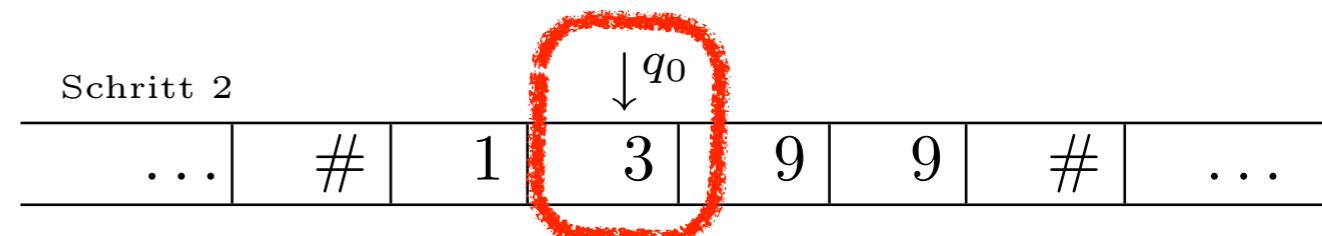
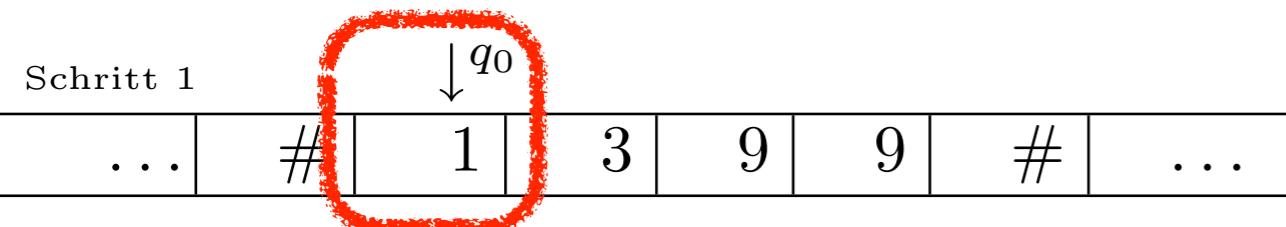
$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

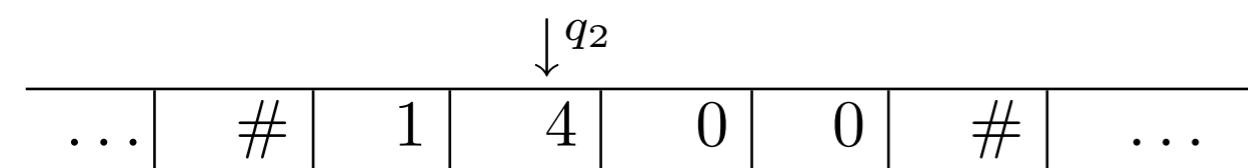
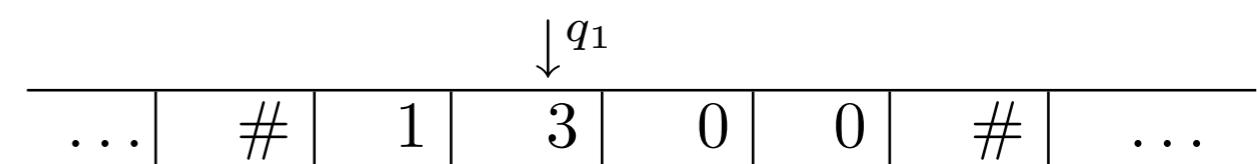
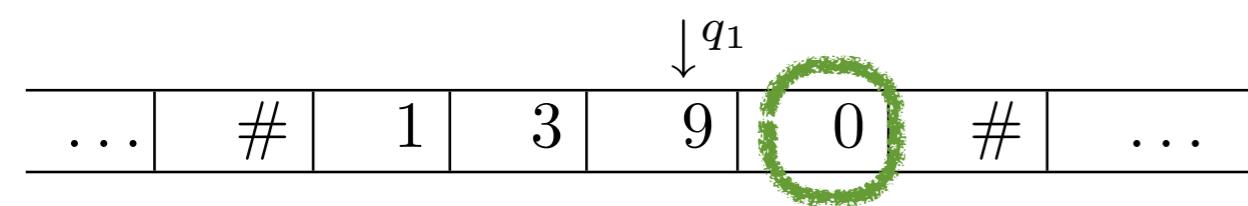
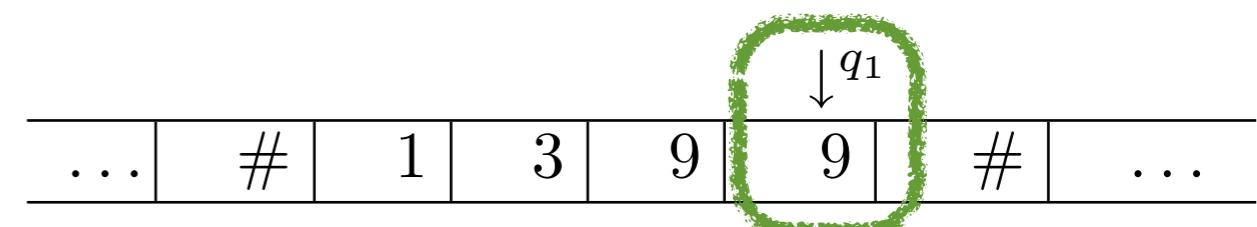
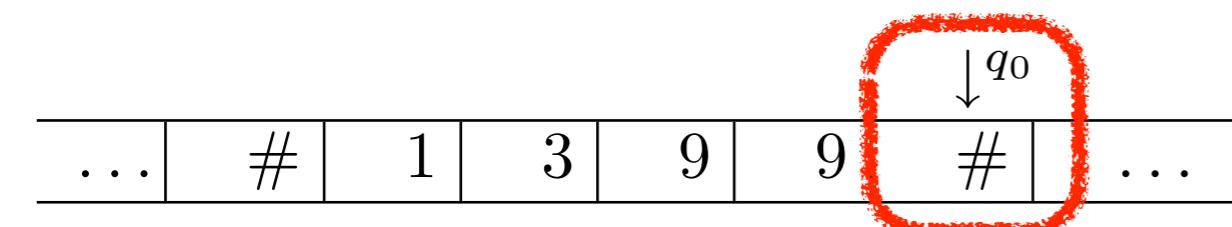
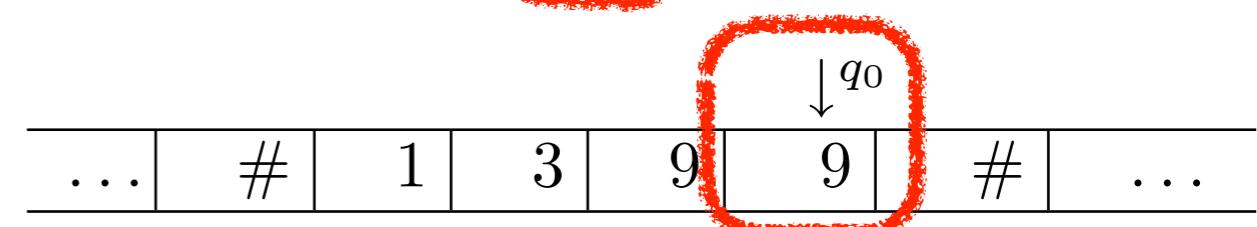
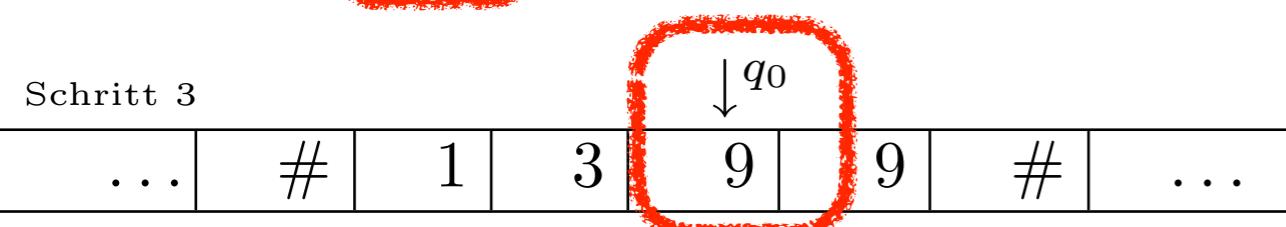
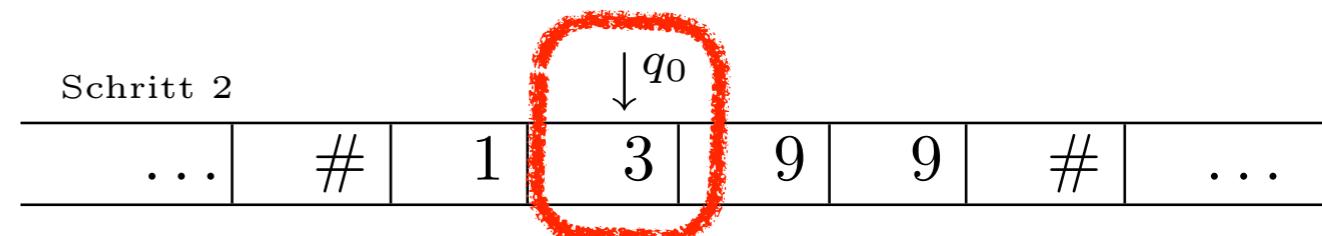
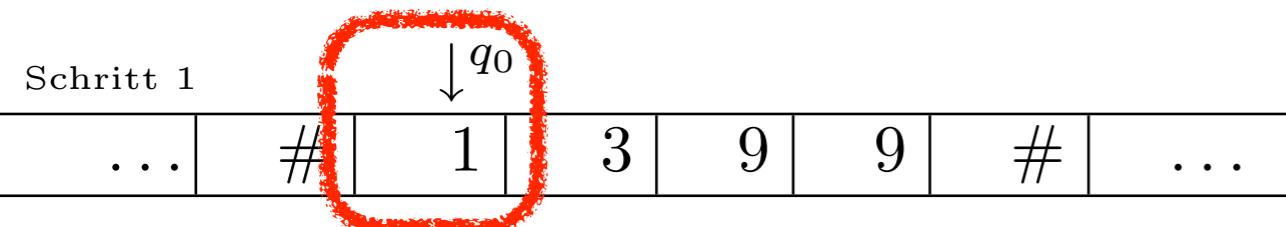
$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

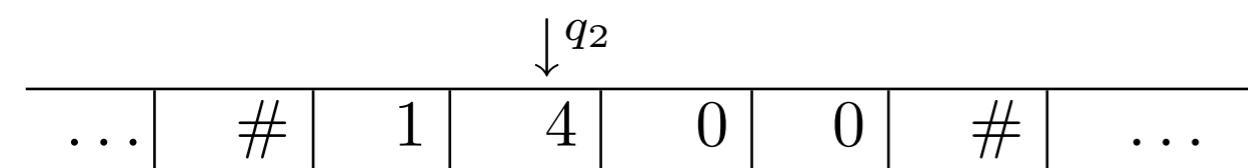
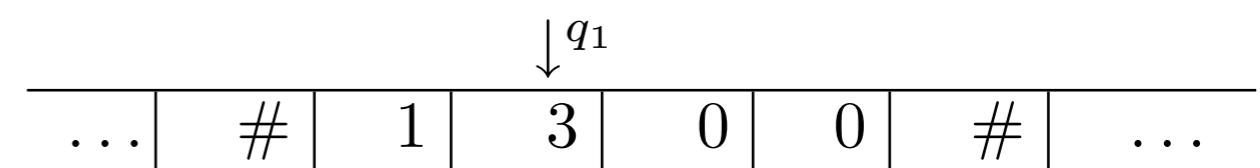
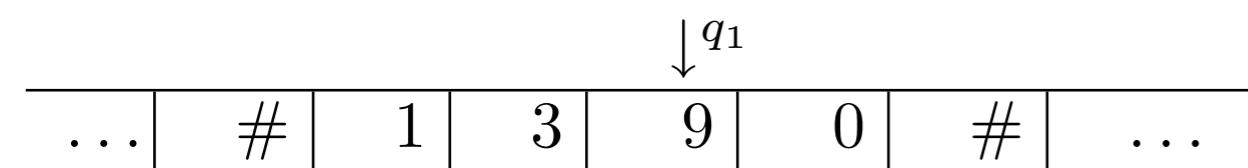
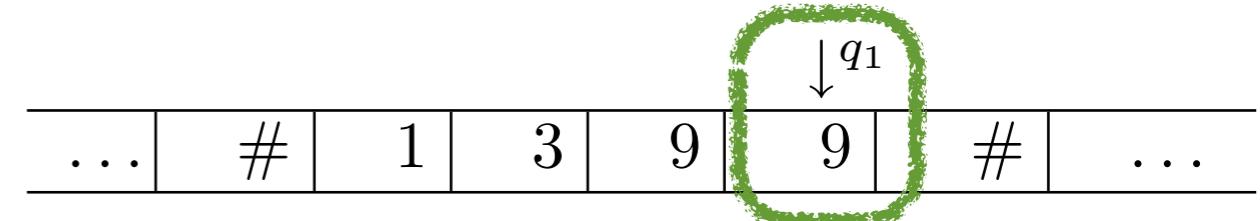
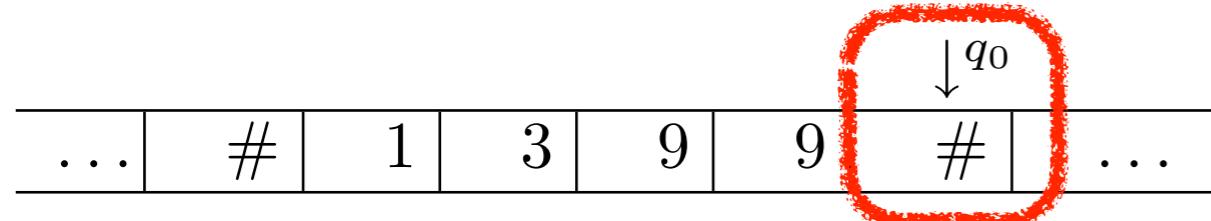
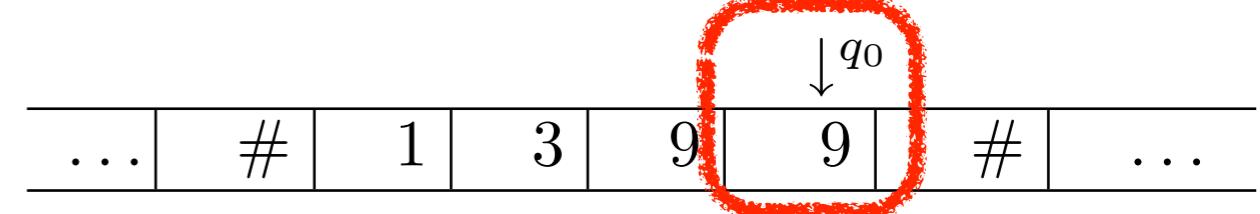
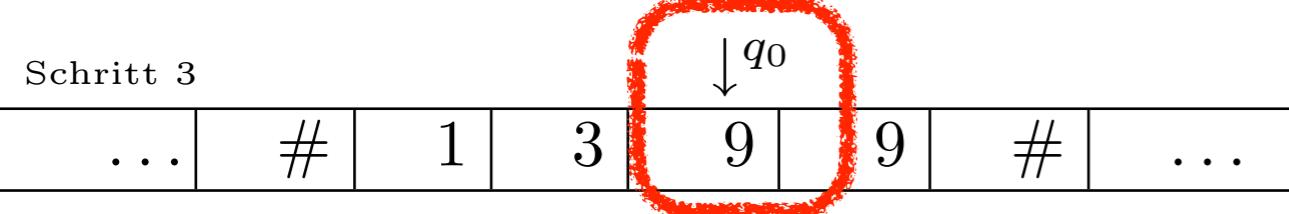
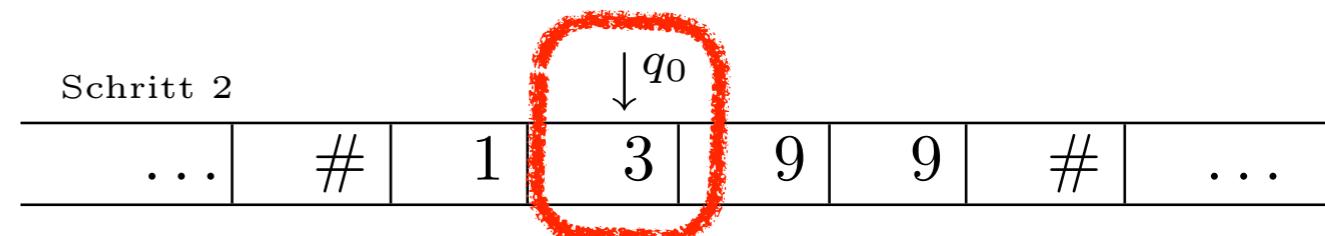
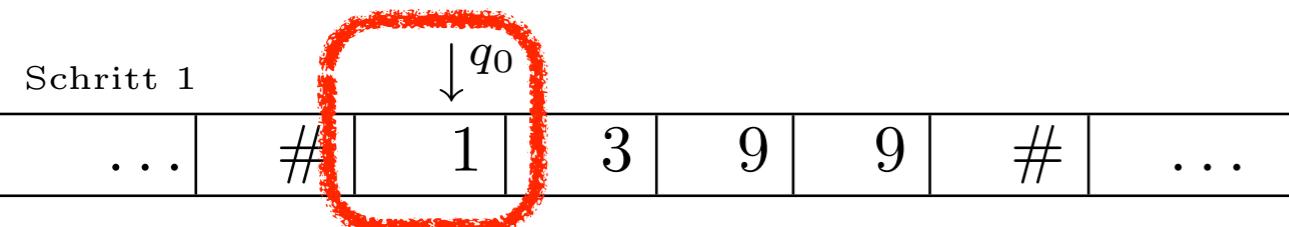
Arbeitsweise von $Succ(1399)$



Arbeitsweise von $\text{Succ}(1399)$



Arbeitsweise von $Succ(1399)$



Arbeitsweise von $Succ(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

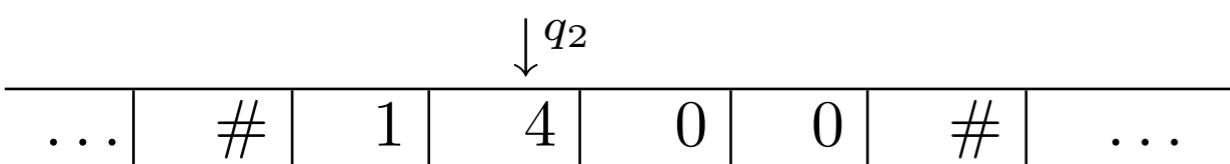
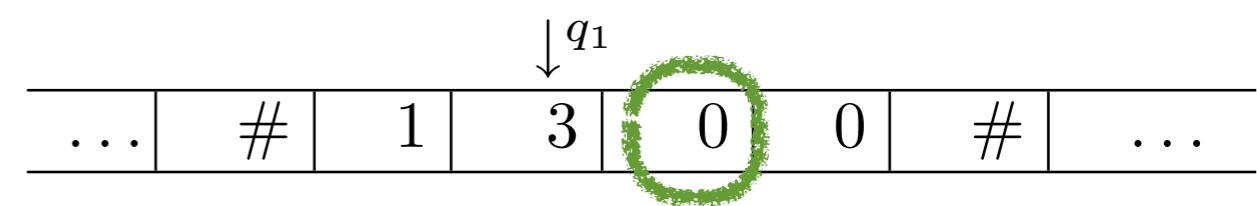
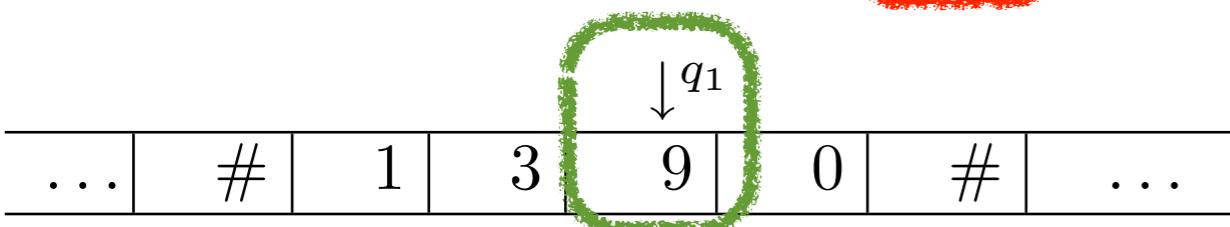
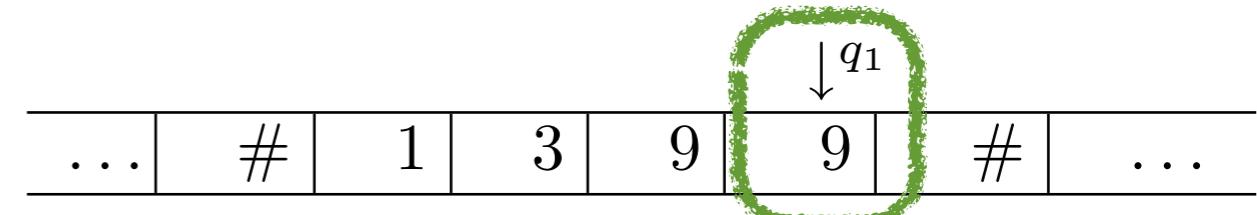
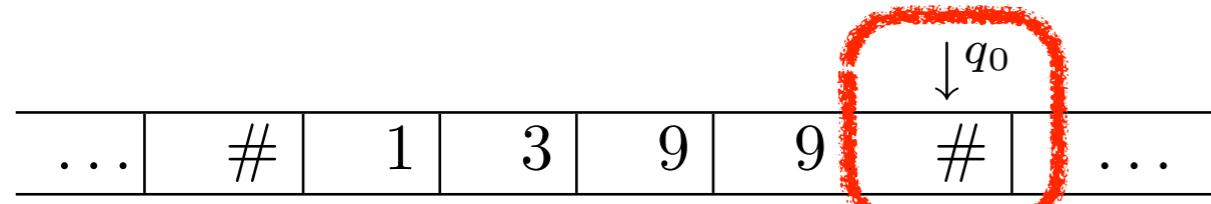
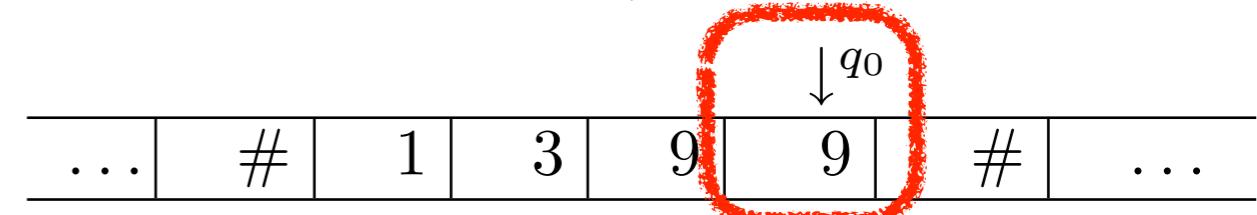
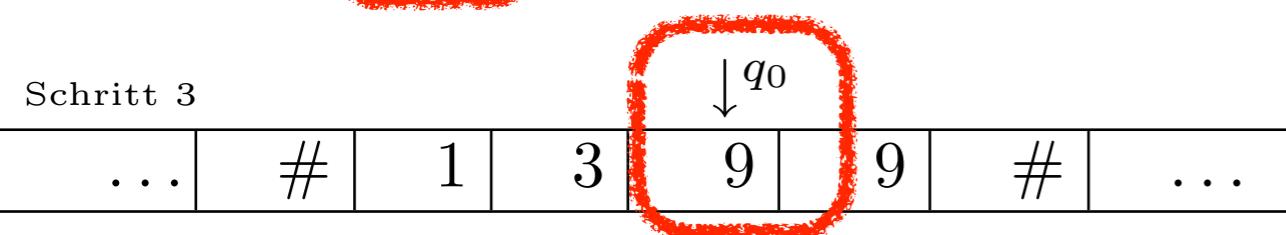
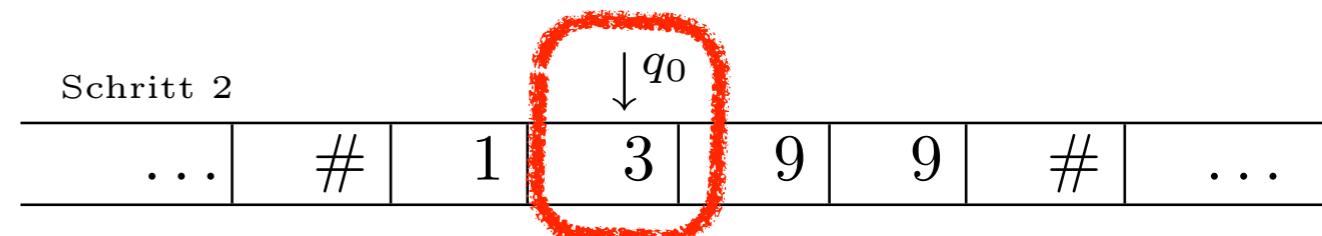
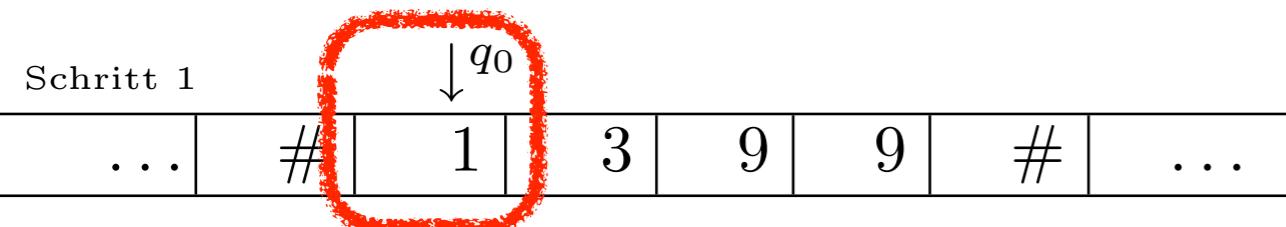
$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

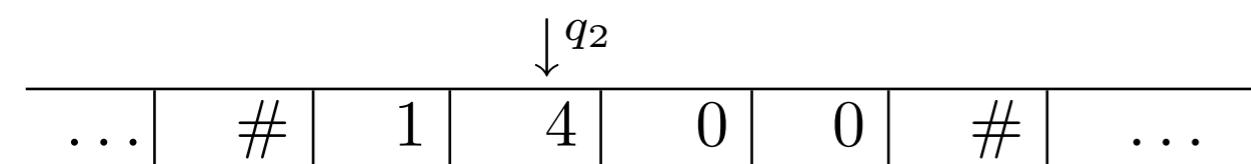
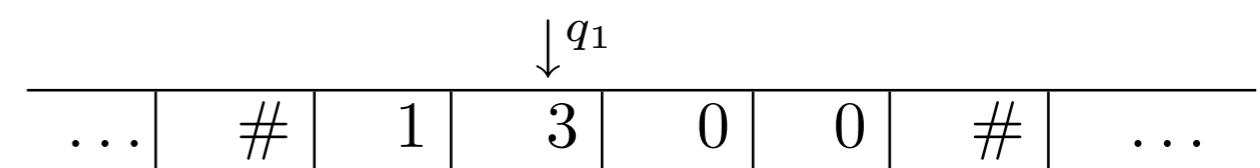
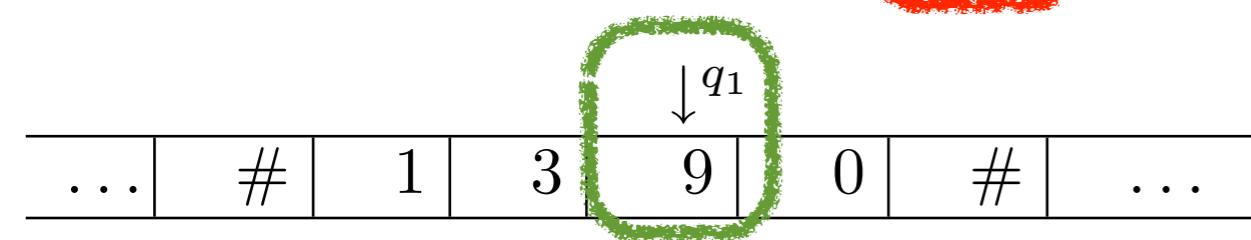
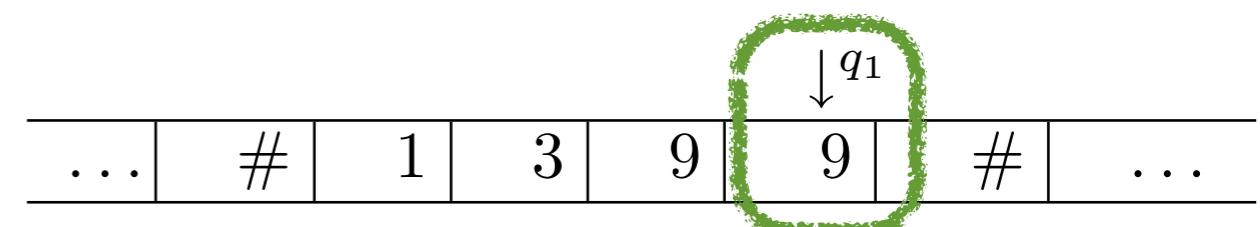
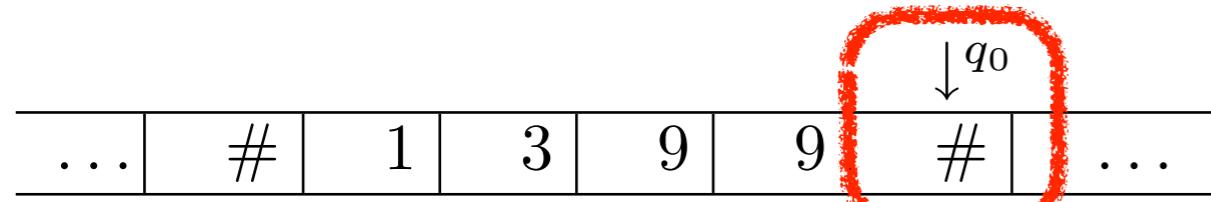
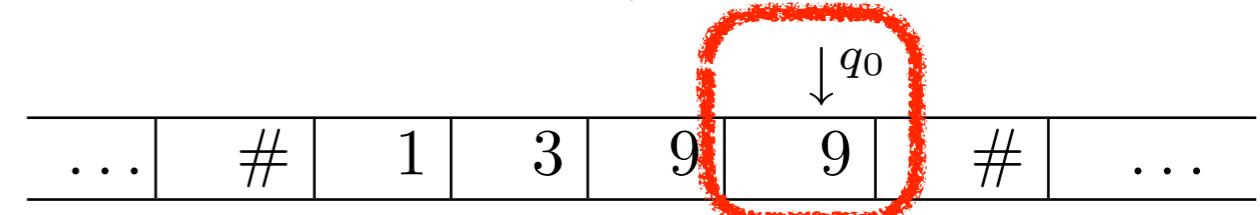
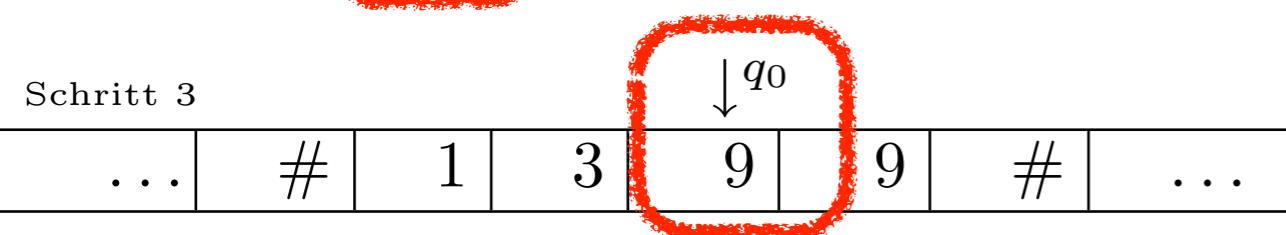
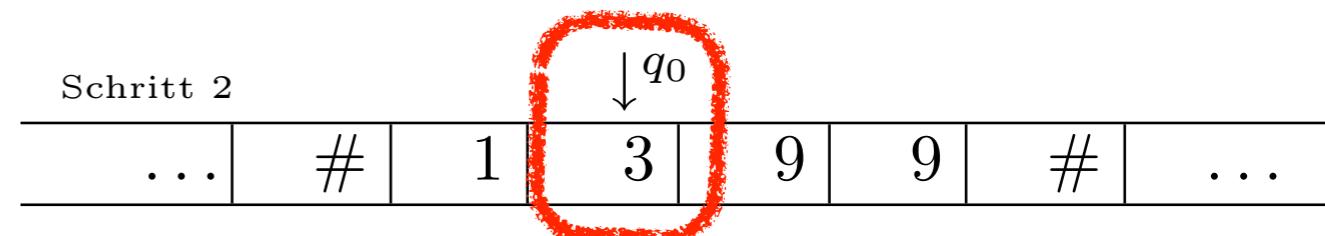
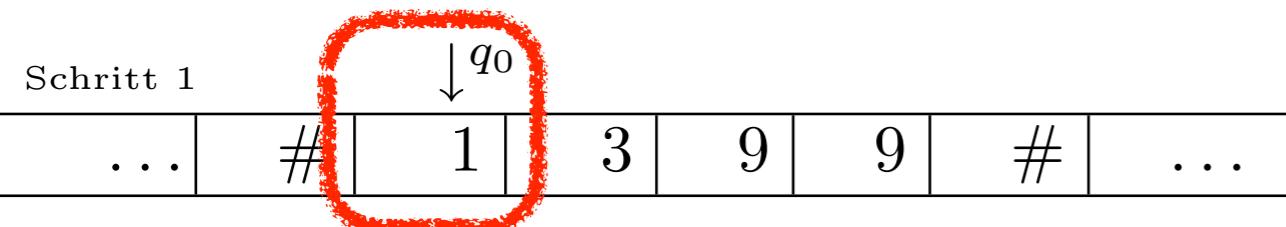
$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $\text{Succ}(1399)$



Arbeitsweise von $\text{Succ}(1399)$



Arbeitsweise von $Succ(1399)$

Schritt 1

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 2

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

Schritt 3

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_0$
$\dots \# 1 3 9 9 \# \dots$

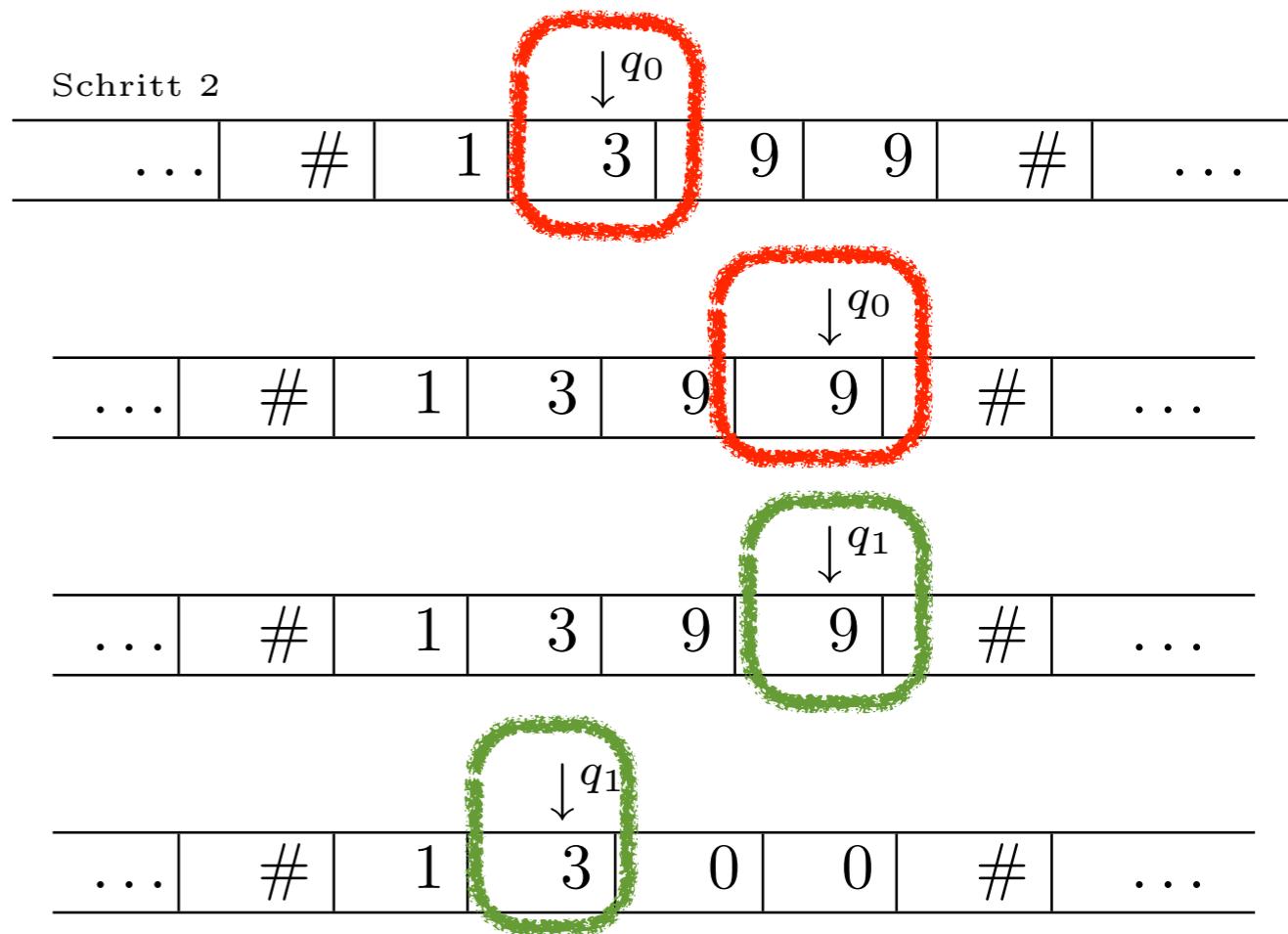
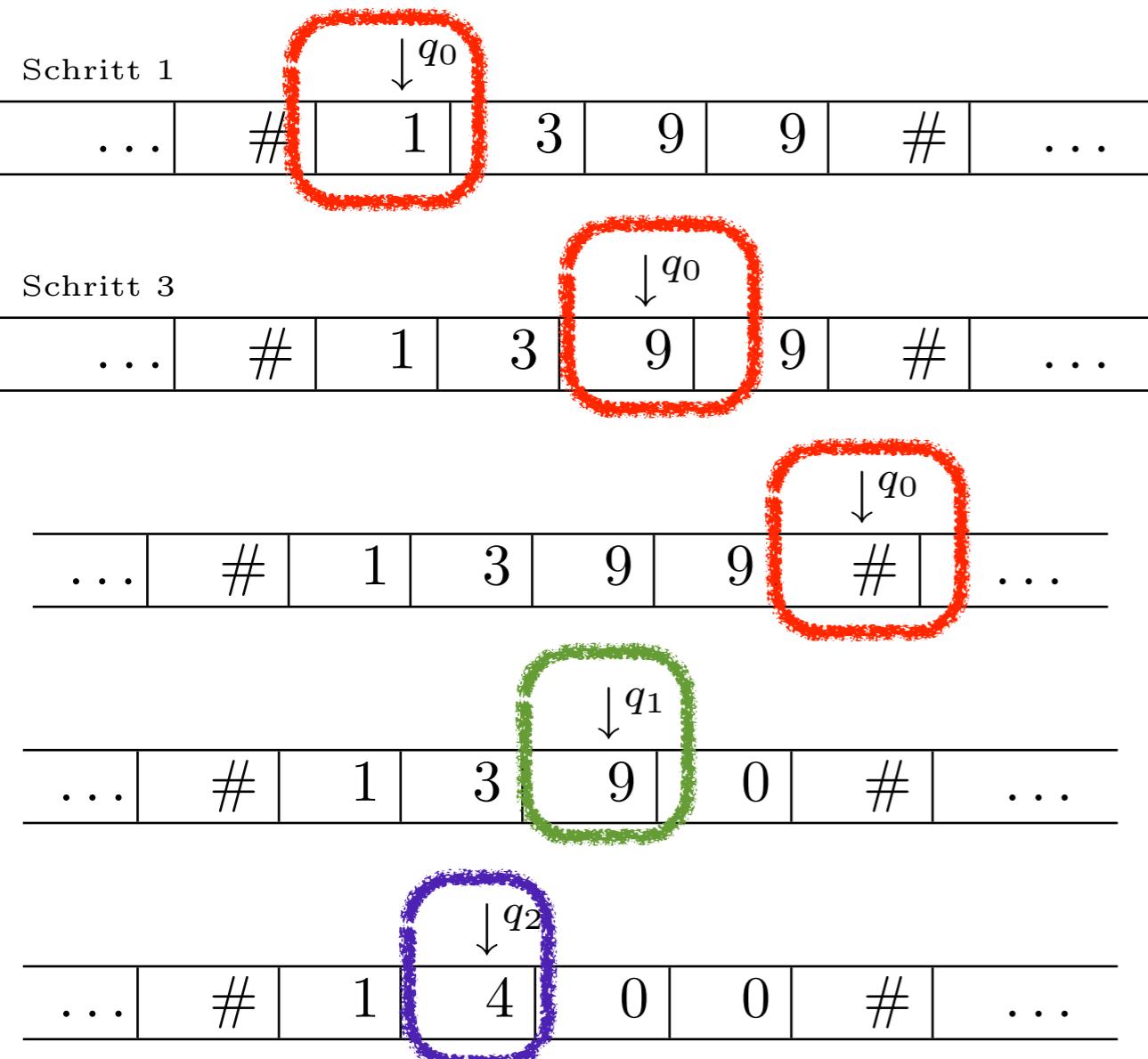
$\downarrow q_1$
$\dots \# 1 3 9 9 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 9 0 \# \dots$

$\downarrow q_1$
$\dots \# 1 3 0 0 \# \dots$

$\downarrow q_2$
$\dots \# 1 4 0 0 \# \dots$

Arbeitsweise von $Succ(1399)$

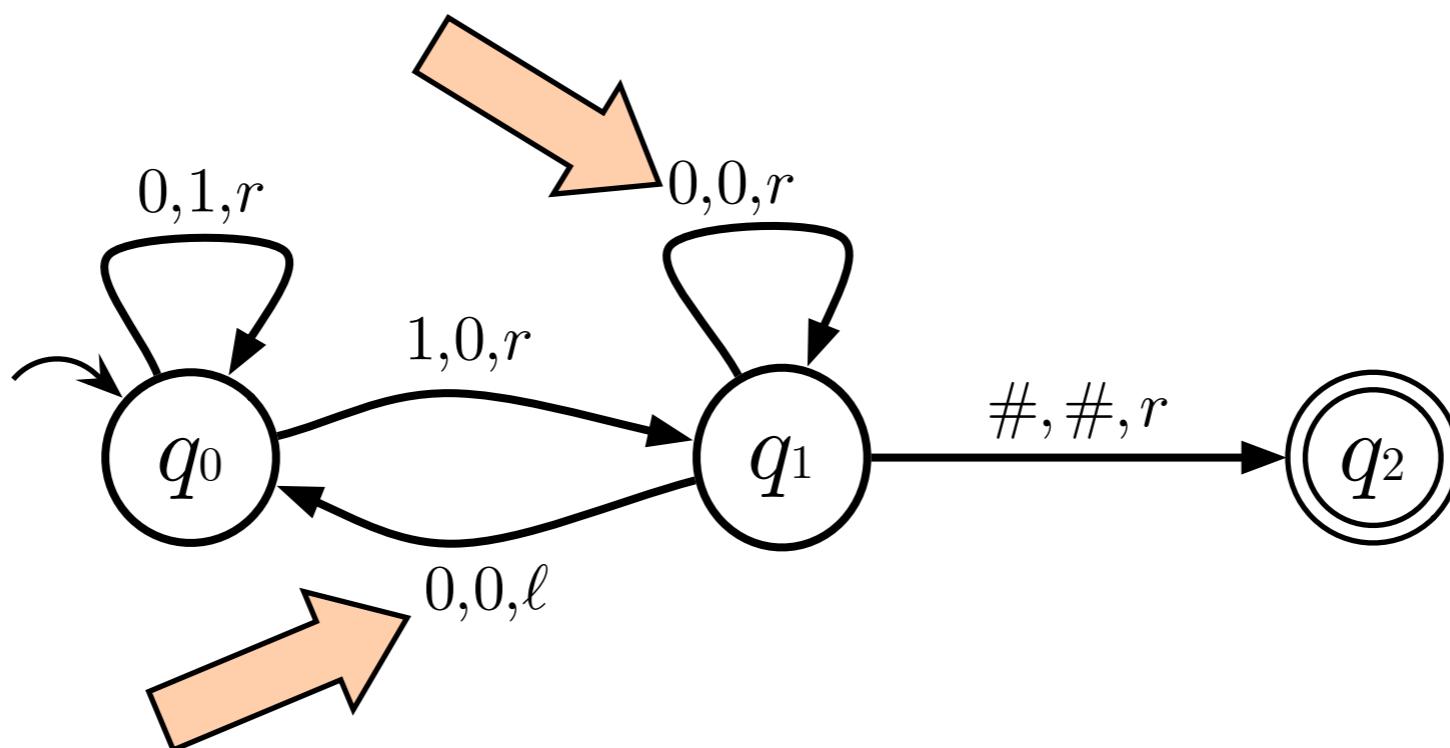


Nichtdeterminismus bei TM

... zu einem Zustand q und einem Symbol x unter dem Kopf kann es mehrere verschiedene Möglichkeiten für die Folgekonfiguration geben.

... zum Beispiel: $\delta(q, A) = \{(q', A, R), (q'', \#, H)\}$ führt zu verschiedenen erlaubten Folgekonfigurationen, die wie bei DTM definiert sind!

$$M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, \#\}, \delta, q_0, \#, \{q_2\})$$



Nichtdeterministische Turingmaschine (NTM)

Definition 18.8:

$M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ definiert eine **nichtdeterministische Turingmaschine (NTM)**, wenn zu jedem Paar $(p, y) \in Q \times \Gamma$ eine endliche Zahl von Übergängen möglich ist:

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{l, r, -\}}$$

oder als Relation notiert;

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{l, r, -\}$$

Alles andere, wie bei der *DTM*.

(Wie üblich bezeichnet 2^A die Potenzmenge einer Menge A .)

Jede DTM ist spezielle NTM, aber i.a. nicht umgekehrt.

Nichtdeterministische Turingmaschine (NTM)

Definition 18.8:

$M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ definiert eine **nichtdeterministische Turingmaschine (NTM)**, wenn zu jedem Paar $(p, y) \in Q \times \Gamma$ eine endliche Zahl von Übergängen möglich ist:

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{l, r, -\}}$$

oder als Relation notiert;

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{l, r, -\}$$

Alles andere, wie bei der *DTM*.

(Wie üblich bezeichnet 2^A die Potenzmenge einer Menge A .)

Jede DTM ist spezielle NTM, aber i.a. nicht umgekehrt.

von NTM akzeptierte Sprache

Definition 18.9:

1. Die von der $NTM T$ akzeptierte Sprache ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

2. Eine von der $NTM T$ akzeptierte Sprache wird von T **entschieden**, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

Die Familie der von NTM 's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine } NTM \text{ ist.}\}$$

von NTM akzeptierte Sprache

Definition 18.9

1. Die von der

Dazu muss noch nicht mal die ganze Eingabe gelesen worden sein.

ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

2. Eine von der NTM T akzeptierte Sprache wird von T entschieden, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

Die Familie der von NTM's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine NTM ist.}\}$$

von NTM akzeptierte Sprache

Definition 18.9:

1. Die von der $NTM T$ akzeptierte Sprache ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

2. Eine von der $NTM T$ akzeptierte Sprache wird von T **entschieden**, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

Die Familie der von NTM 's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine } NTM \text{ ist.}\}$$

von NTM akzeptierte Sprache

Definition 18.9:

1. Die von der NTM T akzeptierte Sprache ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

Termination
garantiert

2. Eine von der NTM T akzeptierte Sprache wird von T entschieden, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

Die Familie der von NTM's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine NTM ist.}\}$$

von NTM akzeptierte Sprache

Definition 18.9:

1. Die von der $NTM T$ akzeptierte Sprache ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

2. Eine von der $NTM T$ akzeptierte Sprache wird von T **entschieden**, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

Die Familie der von NTM 's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine } NTM \text{ ist.}\}$$

von NTM akzeptierte Sprache

Definition 18.9:

1. Die von der $NTM T$ akzeptierte Sprache ist:

$$L(T) := \{w \in \Sigma^* \mid \exists p \in F \exists u, v \in \Gamma^* : q_0 w \xrightarrow[T]{*} upv\}$$

2. Eine von der $NTM T$ akzeptierte Sprache wird von T entschieden, wenn alle Rechnungen der NTM endlich sind.

Definition 18.10:

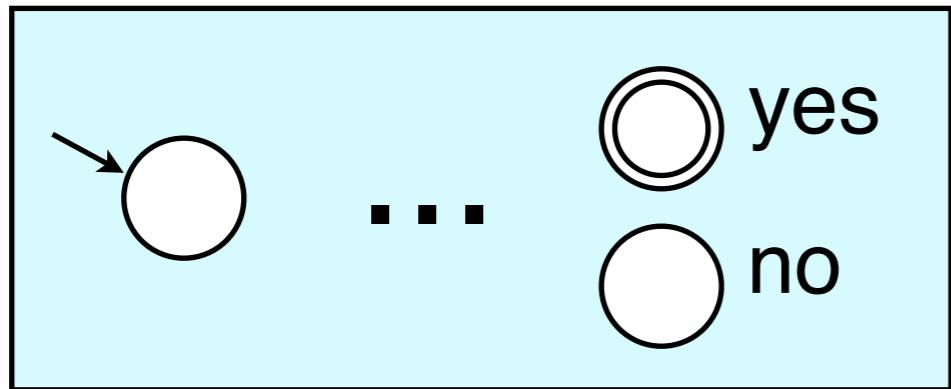
... auch: Familie der aufzählbaren Sprachen

Die Familie der von NTM 's akzeptierten Sprachen aus Σ^* wird mit TA_Σ bezeichnet. D.h.,

$$TA_\Sigma := \{L \subseteq \Sigma^* \mid L = L(T), \text{ wobei } T \text{ eine } NTM \text{ ist.}\}$$

NTM als Entscheider

Eine stets terminierende NTM wird folgendermaßen skizziert:



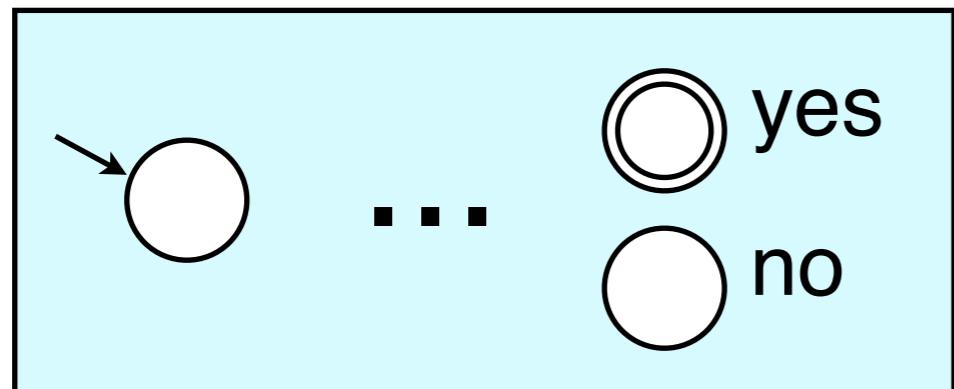
= halten und akzeptieren

= halten und ablehnen

Wir können "Akzeptieren" mit "Halten" gleichsetzen:

NTM als Entscheider

Eine stets terminierende NTM wird folgendermaßen skizziert:



= halten und akzeptieren

= halten und ablehnen

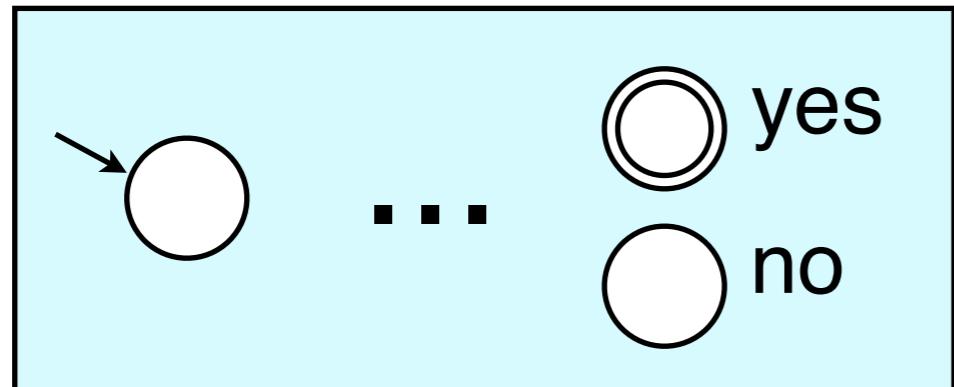
Wir können “Akzeptieren” mit “Halten” gleichsetzen:

Satz 18.1 Sei die NTM A gegeben. Die NTM B entsteht, indem man in A alle Kanten, die aus Endzuständen führen, streicht. Dann gilt $L(A) = L(B)$.

Beweis: Da eine NTM das Eingabewort akzeptiert, sobald eine Rechnung in einen Endzustand führt, sind alle weiteren Kopfbewegungen danach irrelevant.

NTM als Entscheider

Eine stets terminierende NTM wird folgendermaßen skizziert:



= halten und akzeptieren

= halten und ablehnen

Wir können "Akzeptieren" mit "Halten" gleichsetzen:

Satz 18.1 Sei die NTM A gegeben. Die NTM B entsteht, indem man in A alle Kanten, die aus Endzuständen führen, streicht. Dann gilt $L(A) = L(B)$.

Beweis: Da eine NTM das Eingabewort akzeptiert, sobald eine Rechnung in einen Endzustand führt, sind alle weiteren Kopfbewegungen danach irrelevant.

Also können wir aus der NTM A **eine stets terminierende NTM B konstruieren, wenn A auf allen $w \notin L(A)$ hält.**

Eine Hierarchie von Sprachfamilien

Folgende Beziehungen wollen wir zeigen:

- Family $\mathcal{R}eg$ der **regulären Mengen**
- \subsetneq Family $\mathcal{C}f$ der **kontextfreien Mengen**
- \subsetneq Family $\mathcal{C}s$ der **kontextsensitiven Mengen**
- \subsetneq Family $\mathcal{R}ec$ der **entscheidbaren Mengen**
- \subsetneq Family $\mathcal{R}e$ der **aufzählbaren Mengen**
- \subsetneq Family der **abzählbaren Mengen**
- \neq Family der **überabzählbaren Mengen**

Existenz überabzählbarer Mengen ist bekannt.
(z.B. aus *Diagonalbeweis*).

Dies ist die sog. Chomsky-Hierarchie.

NTM durch DTM simulieren

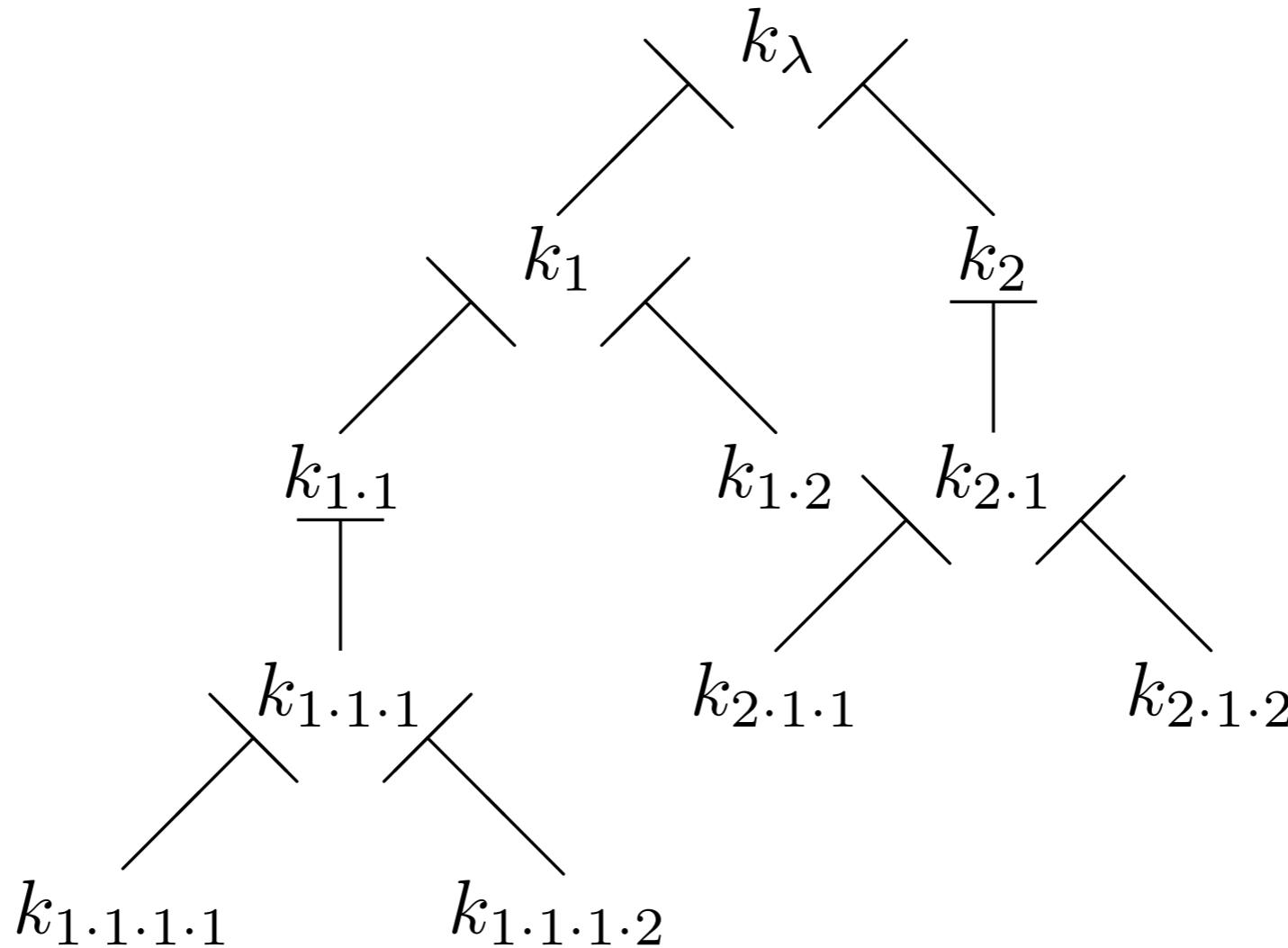
Satz 21.3:

Deterministische und nichtdeterministische Turingmaschinen sind äquivalent, kurz:

$$NTM \equiv DTM$$

Da jede DTM auch eine NTM ist, bleibt nur noch zu zeigen, wie eine beliebige NTM $A := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ durch eine DTM $B := (Q', \Sigma, \Gamma', \delta_B, q'_0, \#, F')$ simuliert werden kann.

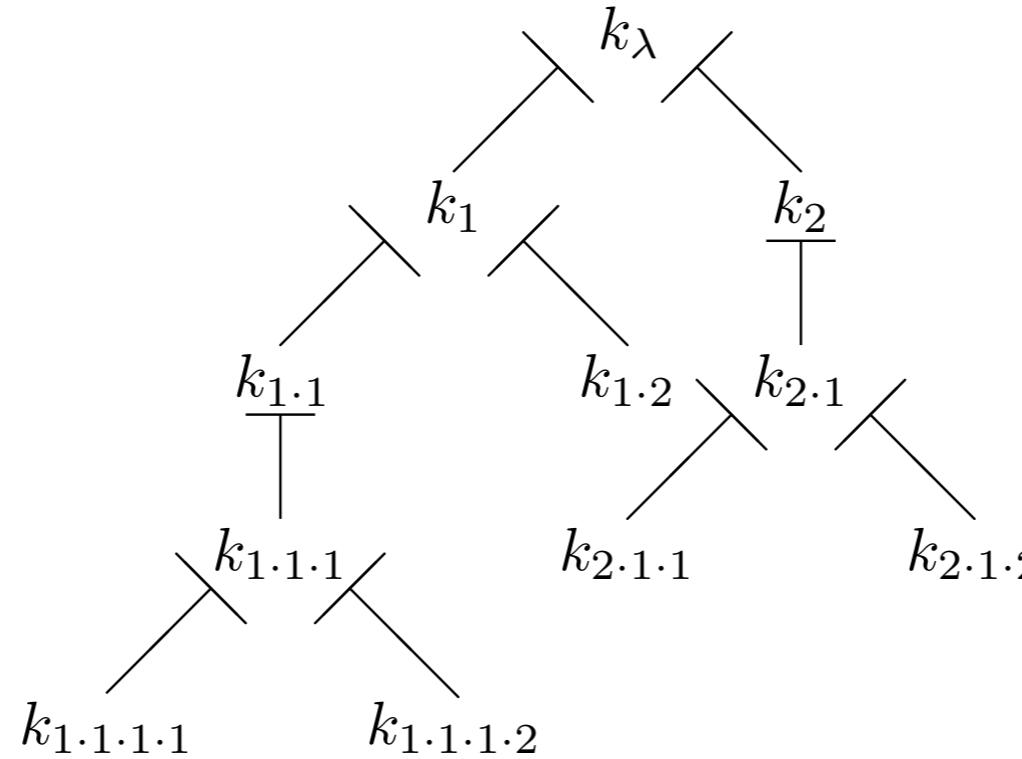
Der Konfigurationsbaum



Die DTM B hat drei Arbeitsbänder, wovon das erste die Eingabe von A enthält und auf dem zweiten Band Wörter $w \in \mathbb{N}^i$ (in Punktschreibweise) in der lexikalischen Ordnung auf $\bigcup_{i \geq 1} \mathbb{N}^i$ generiert werden.

Breitensuche um Konfigurationsbaum

ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2

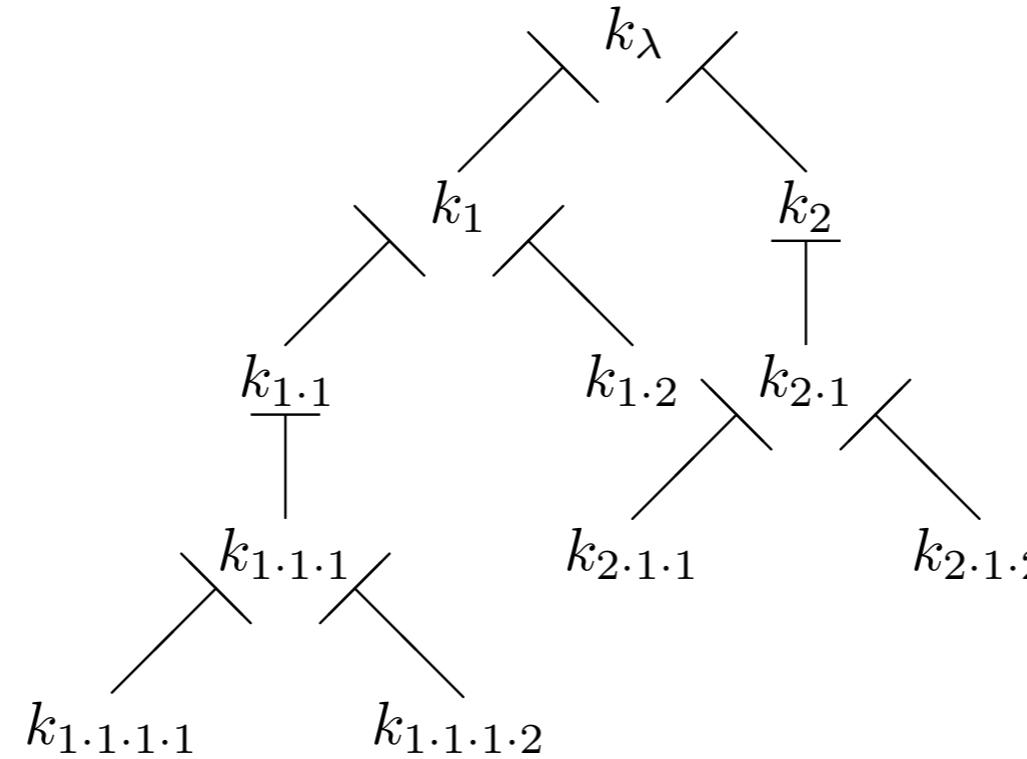


-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ

1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ

1

2

1.1

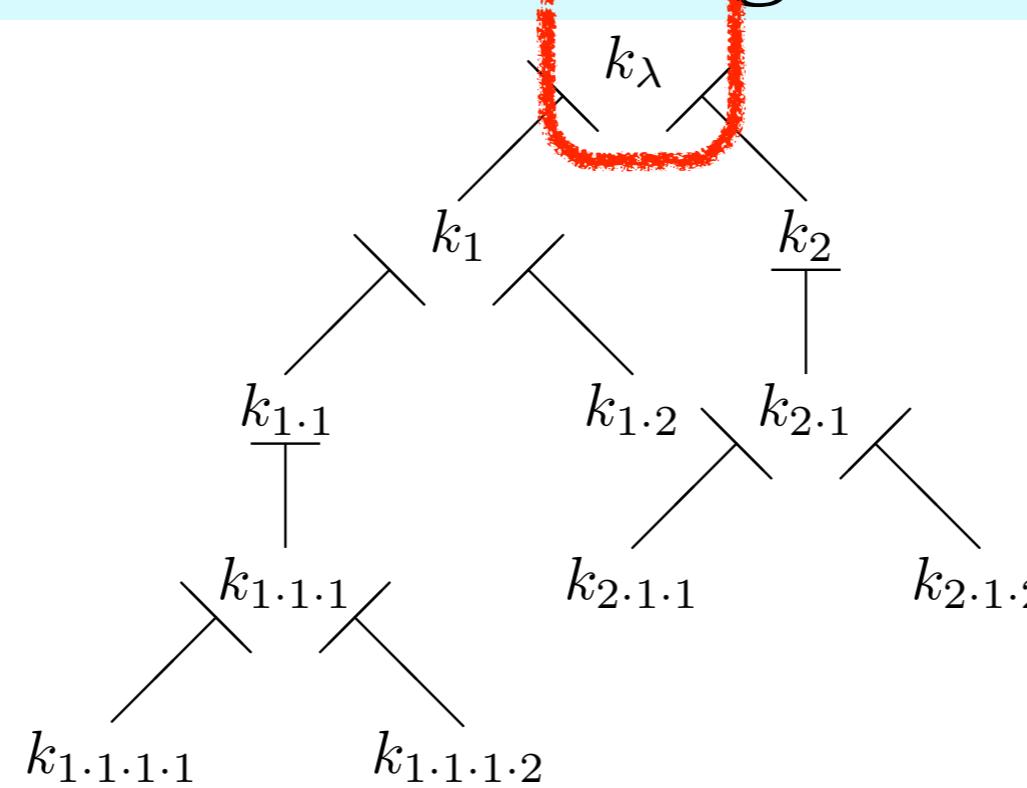
1.2

2.1

2.2

1.1.1

1.1.2

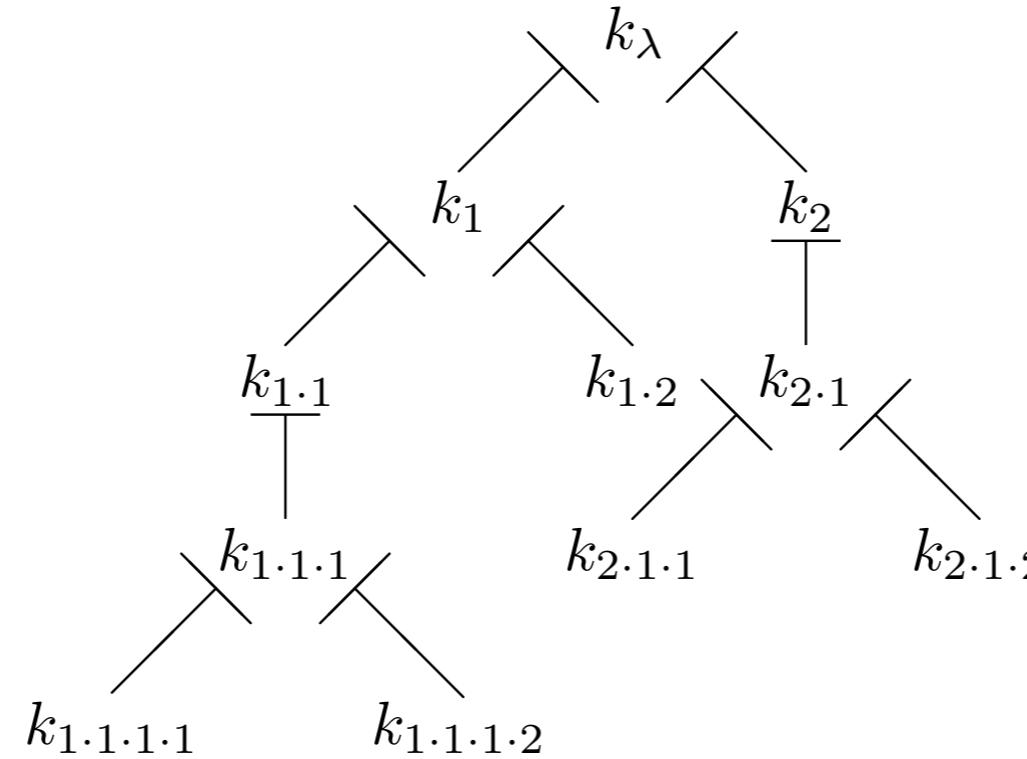


-
- Nach jeder auf dem zweiten Band erzeugten Numerierung
 $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ

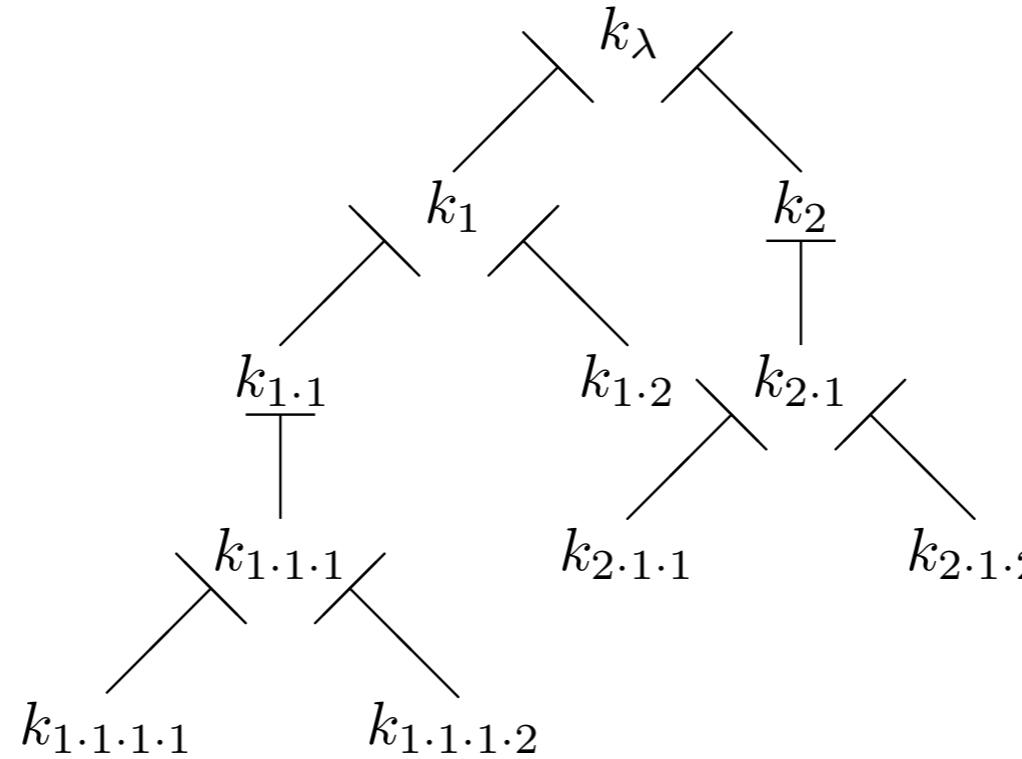
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ
1
2

1.1

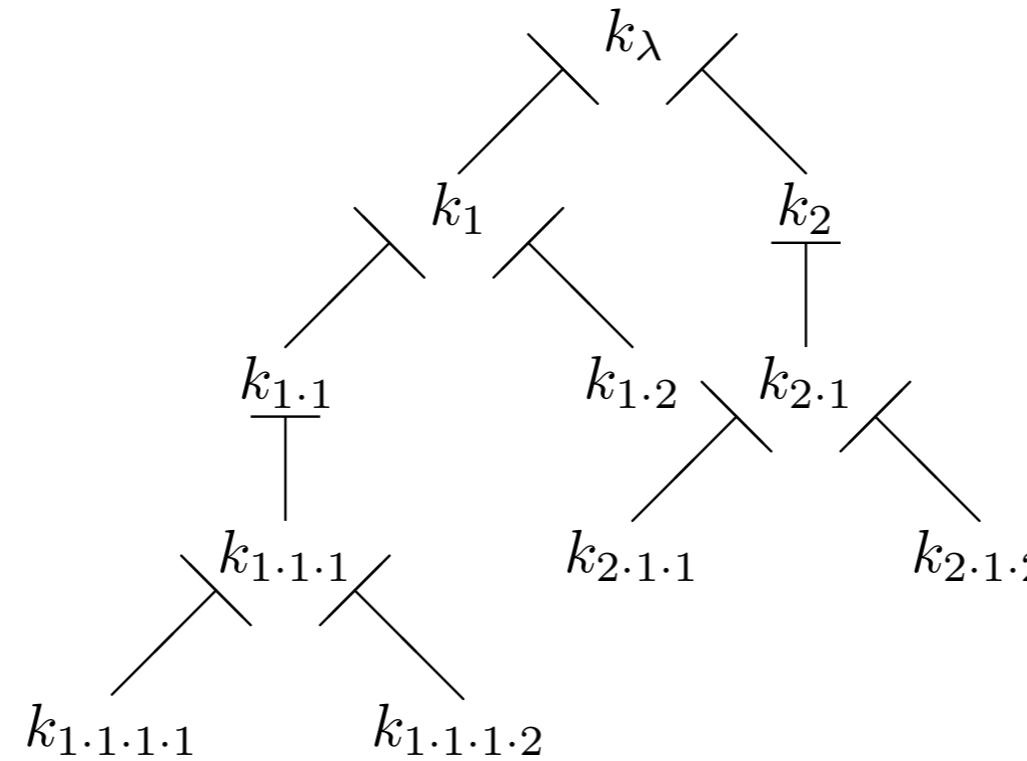
1.2

2.1

2.2

1.1.1

1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ
1
2

1.1

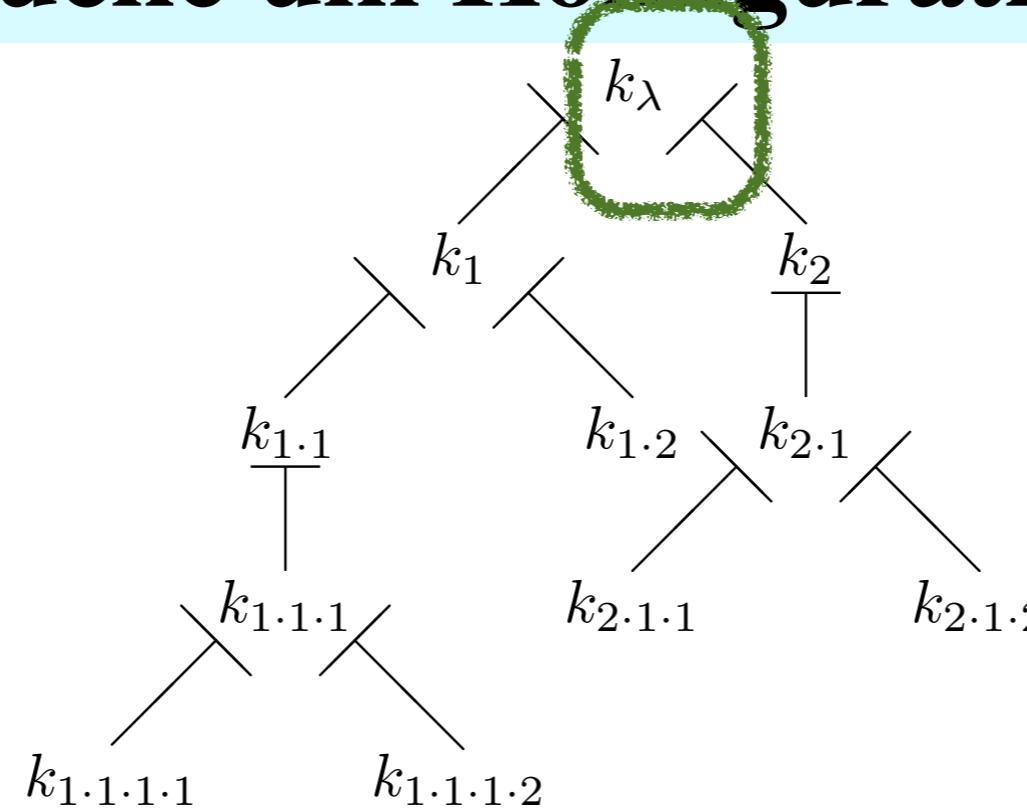
1.2

2.1

2.2

1.1.1

1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ
1
2

1.1

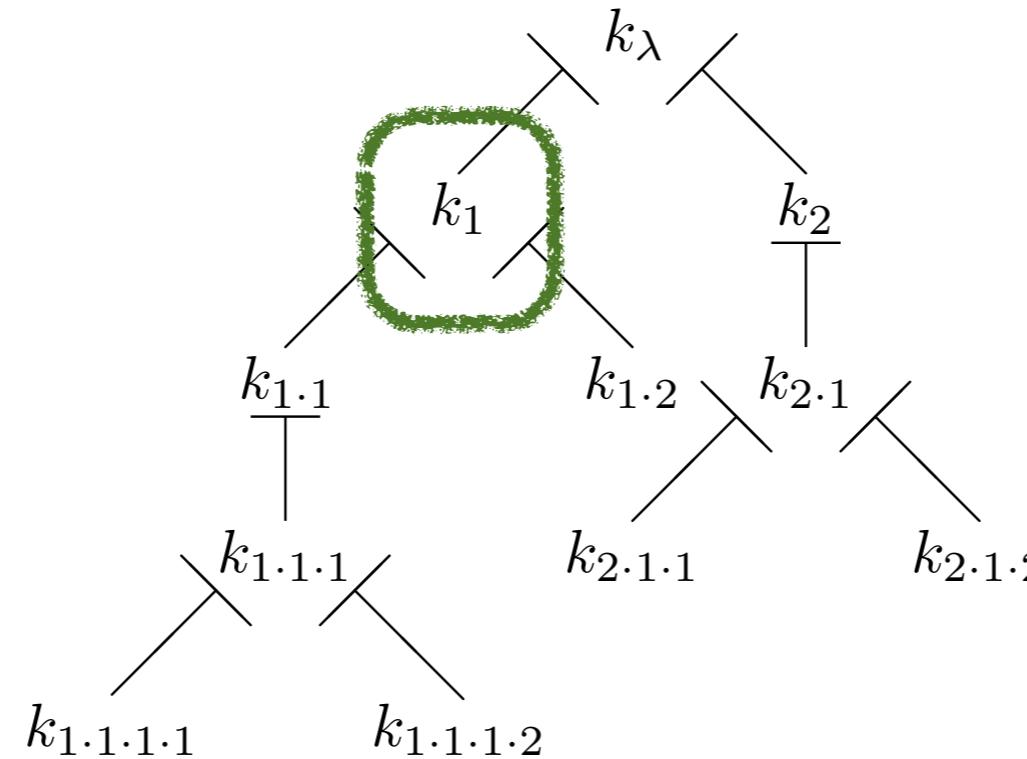
1.2

2.1

2.2

1.1.1

1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ϵ
1
2

1.1

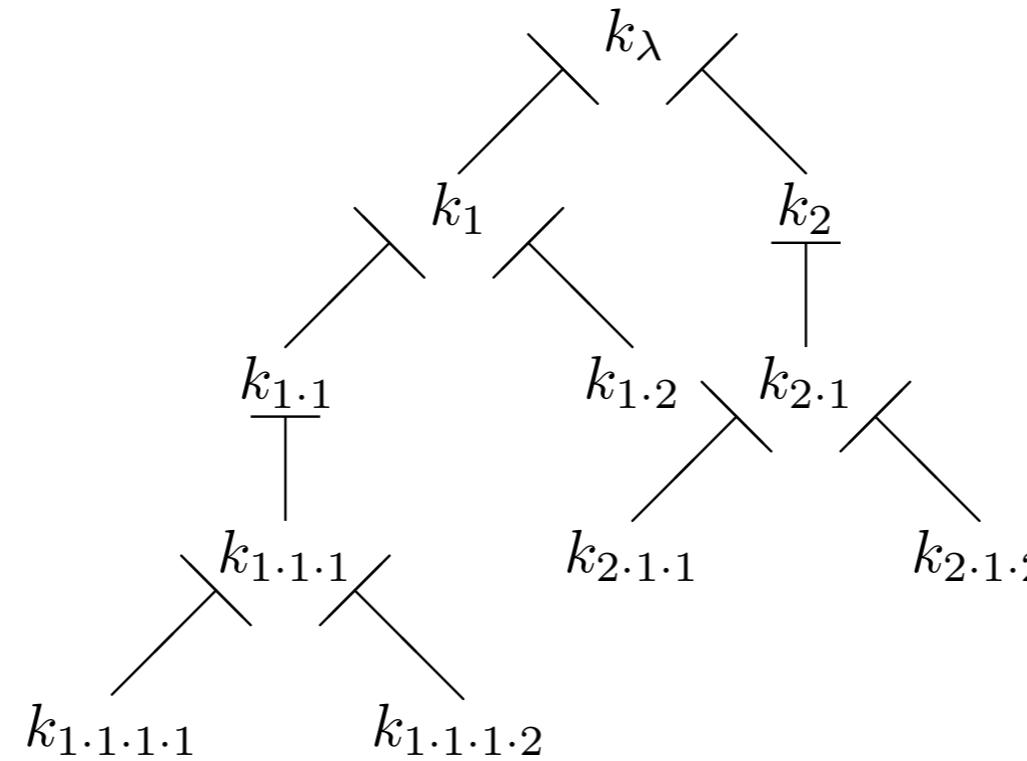
1.2

2.1

2.2

1.1.1

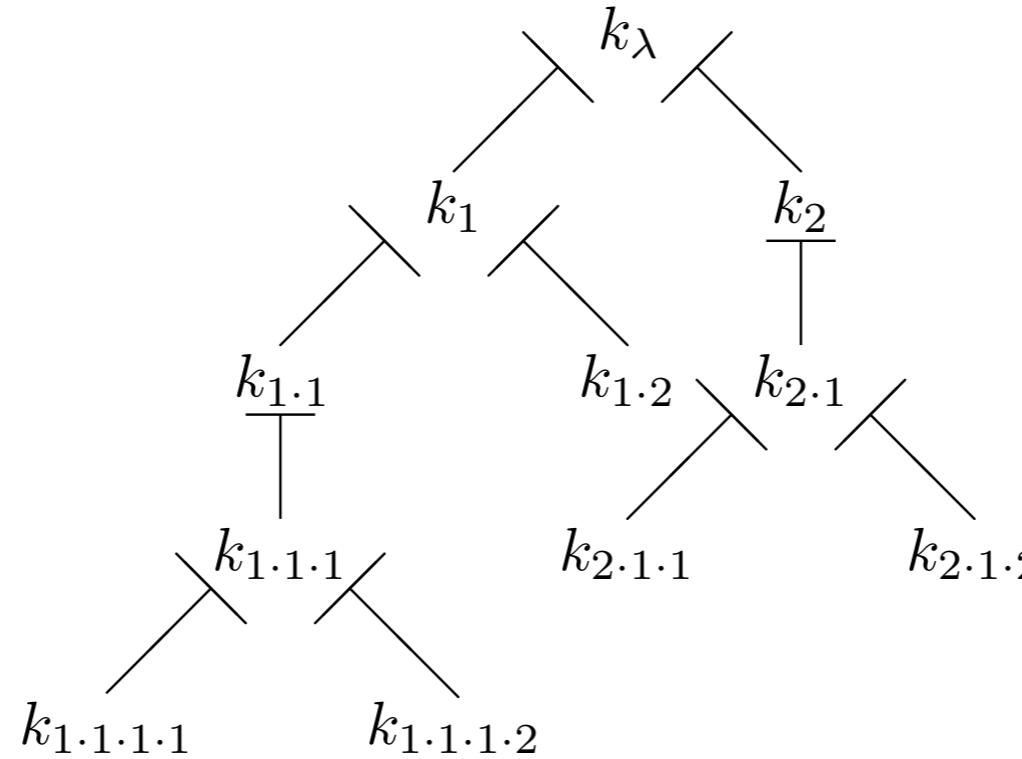
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

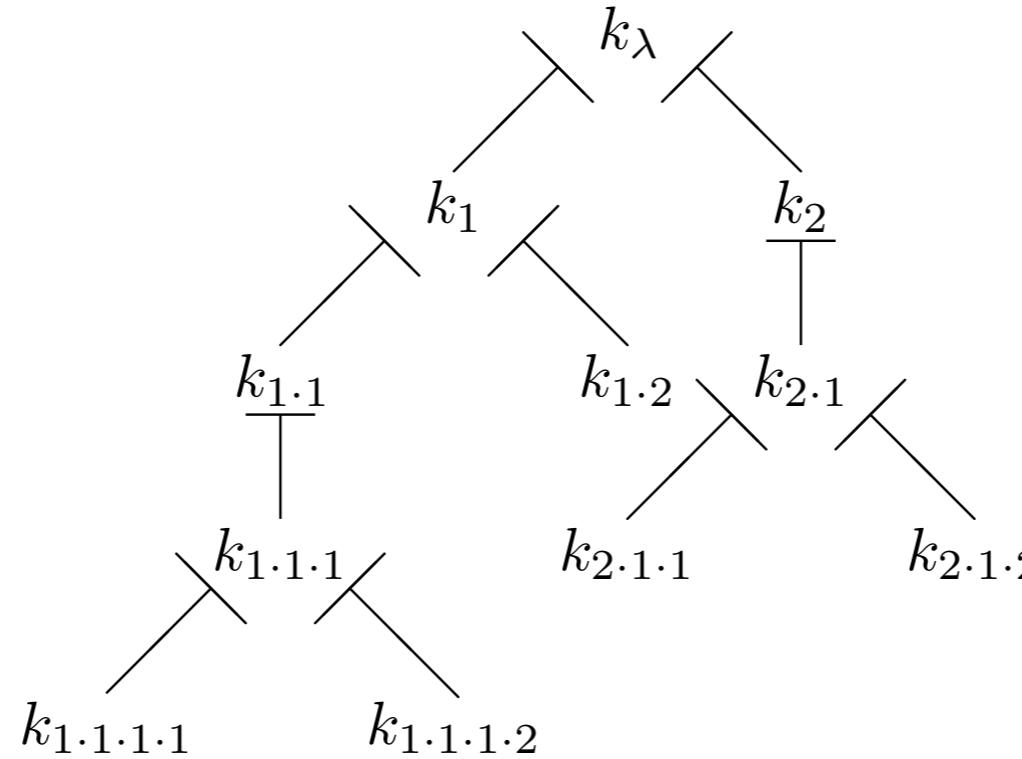
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

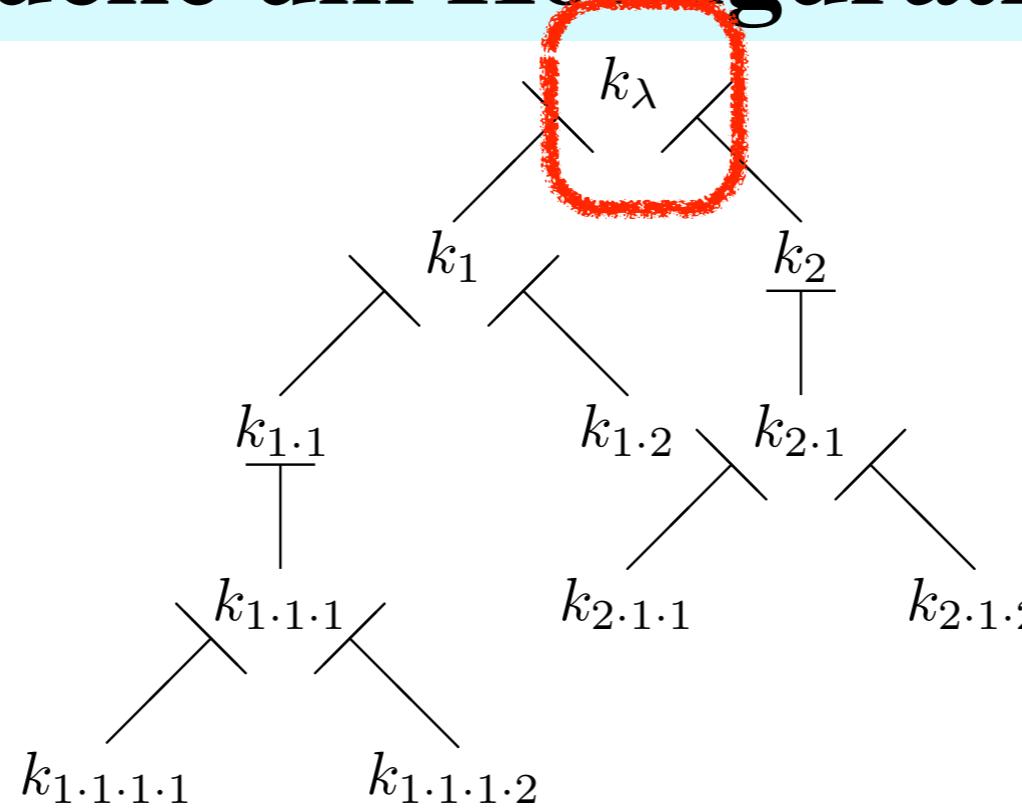
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

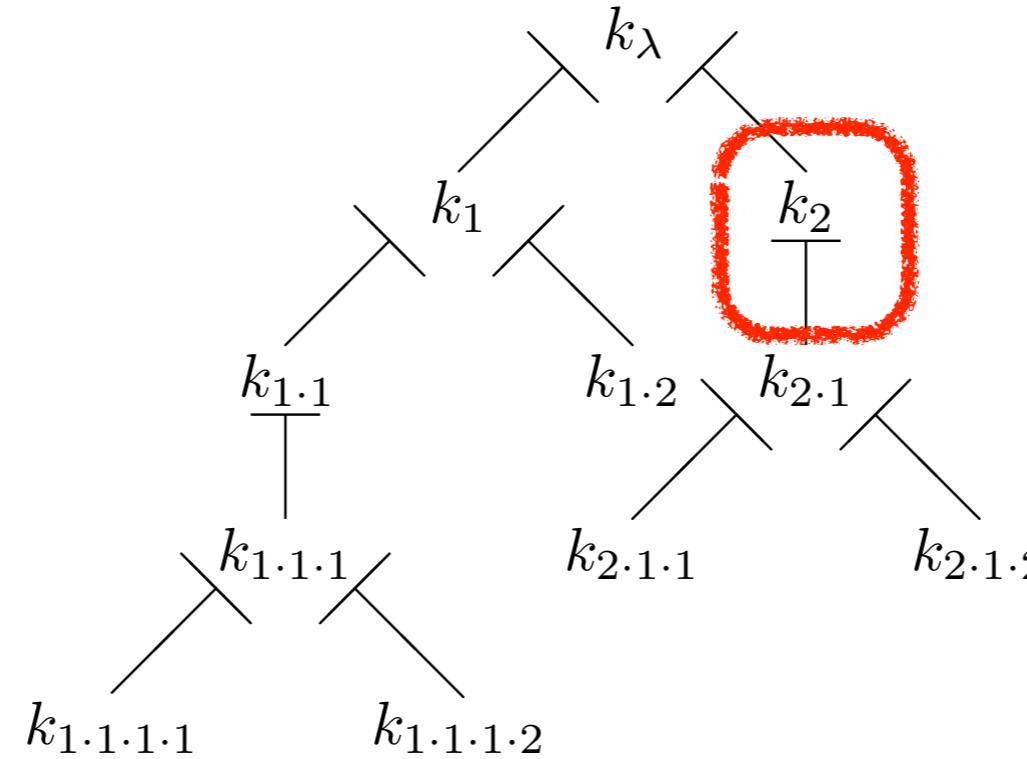
ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

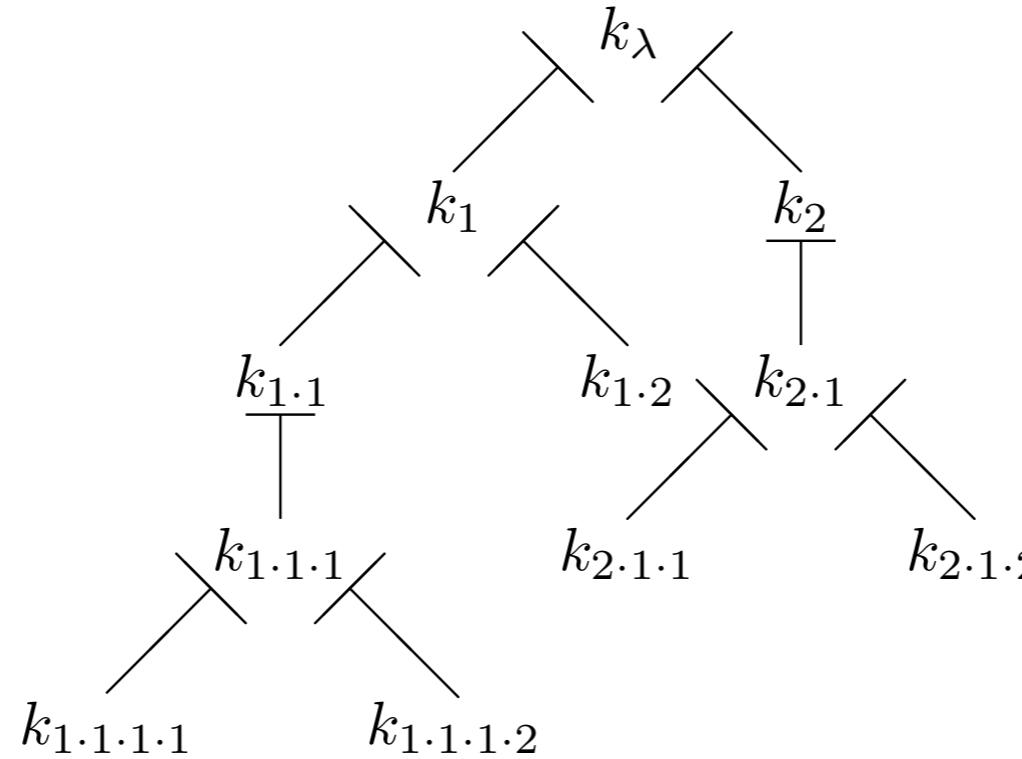
ϵ
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

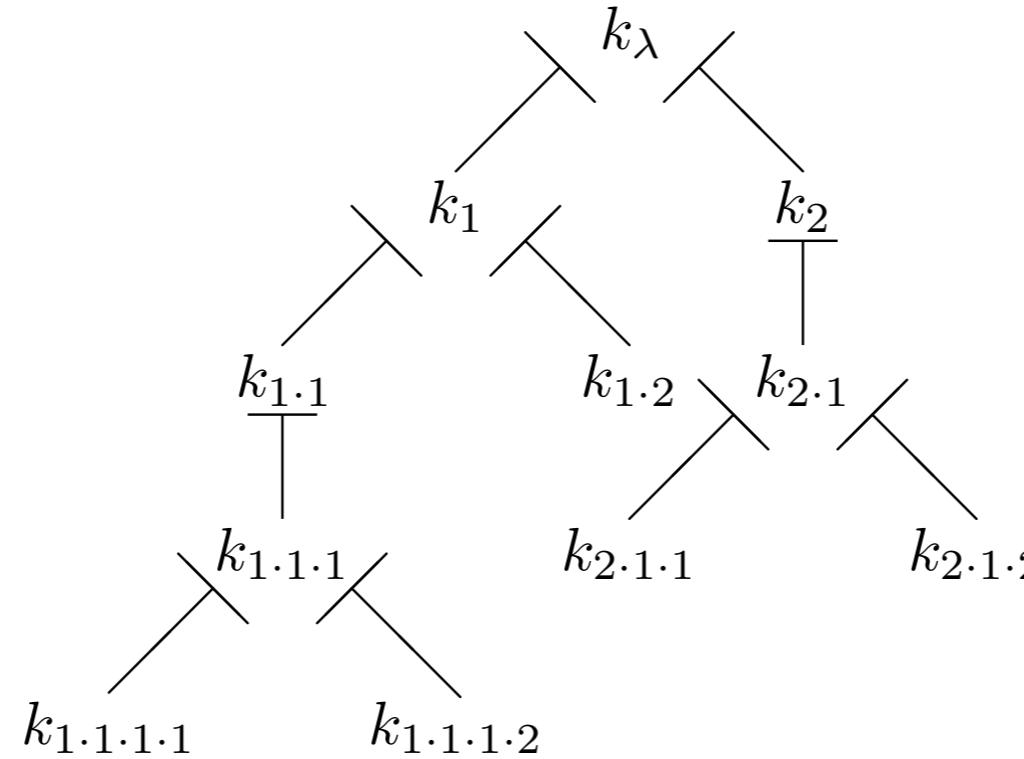
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

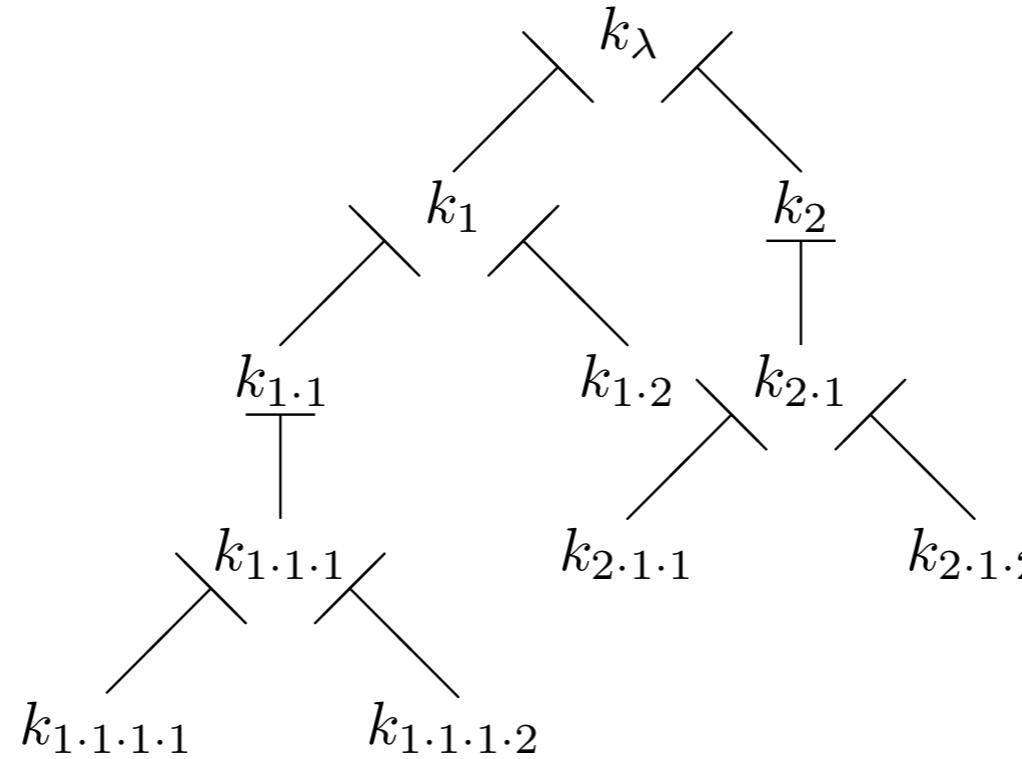
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

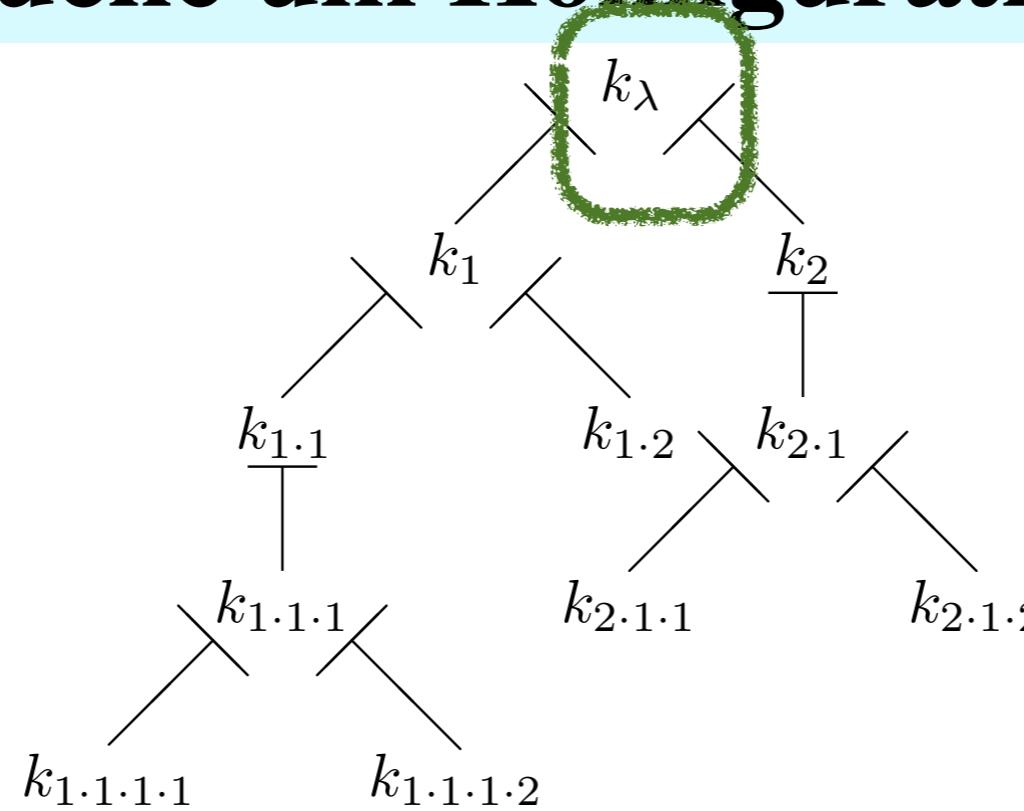
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

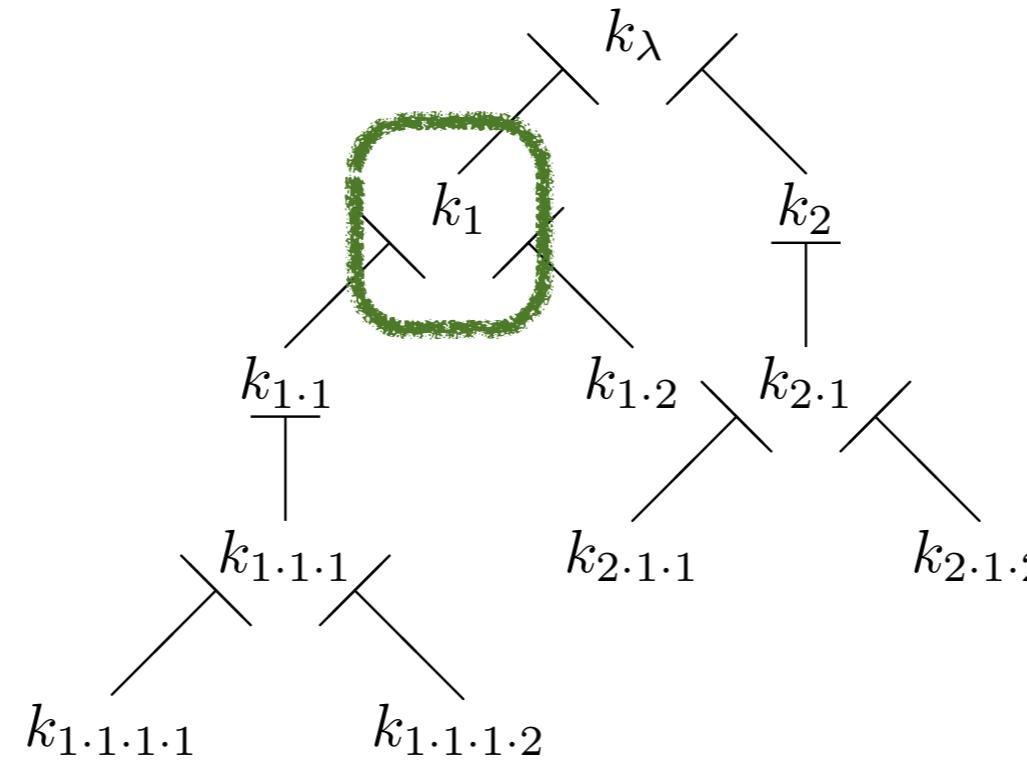
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

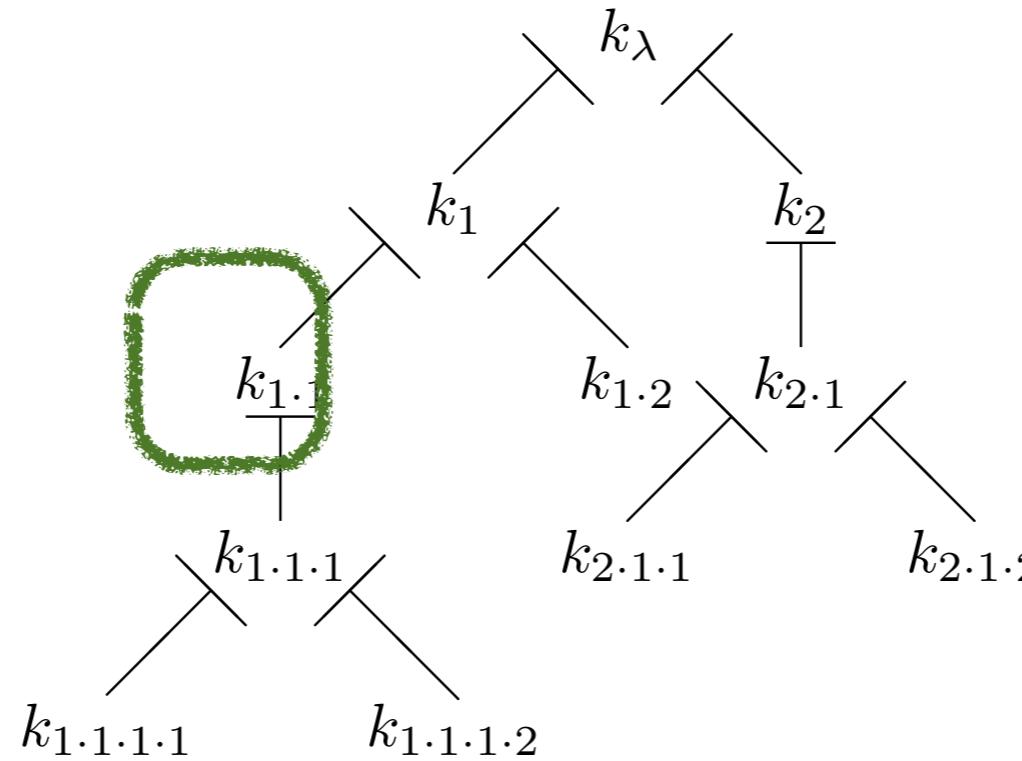
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

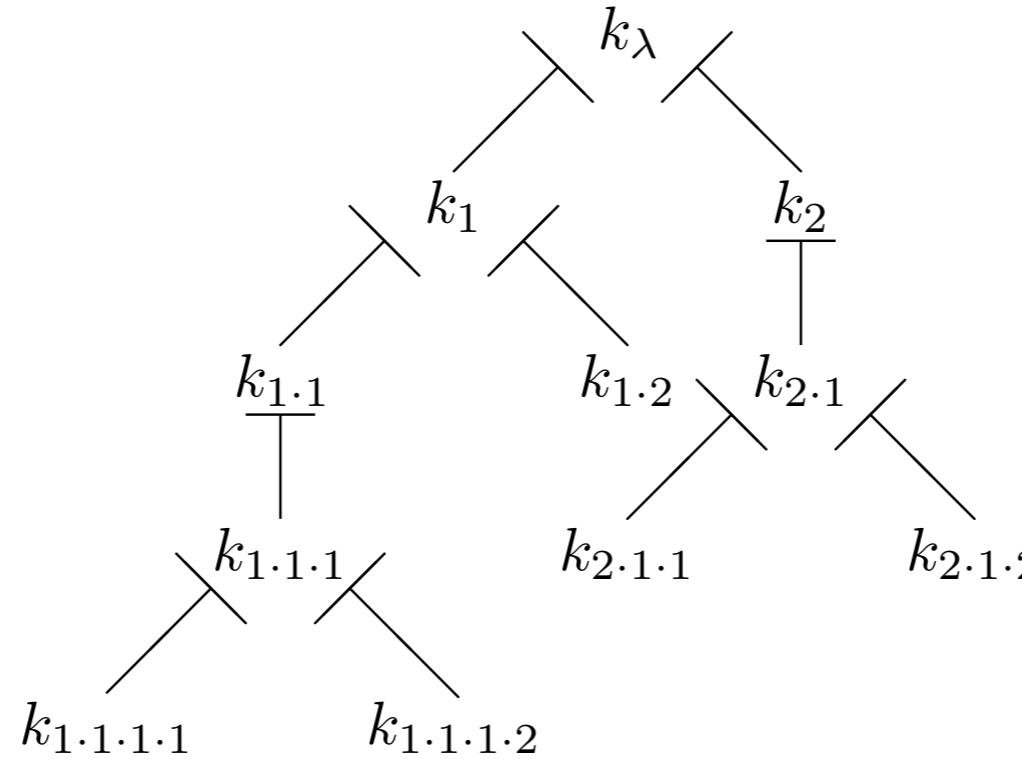
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

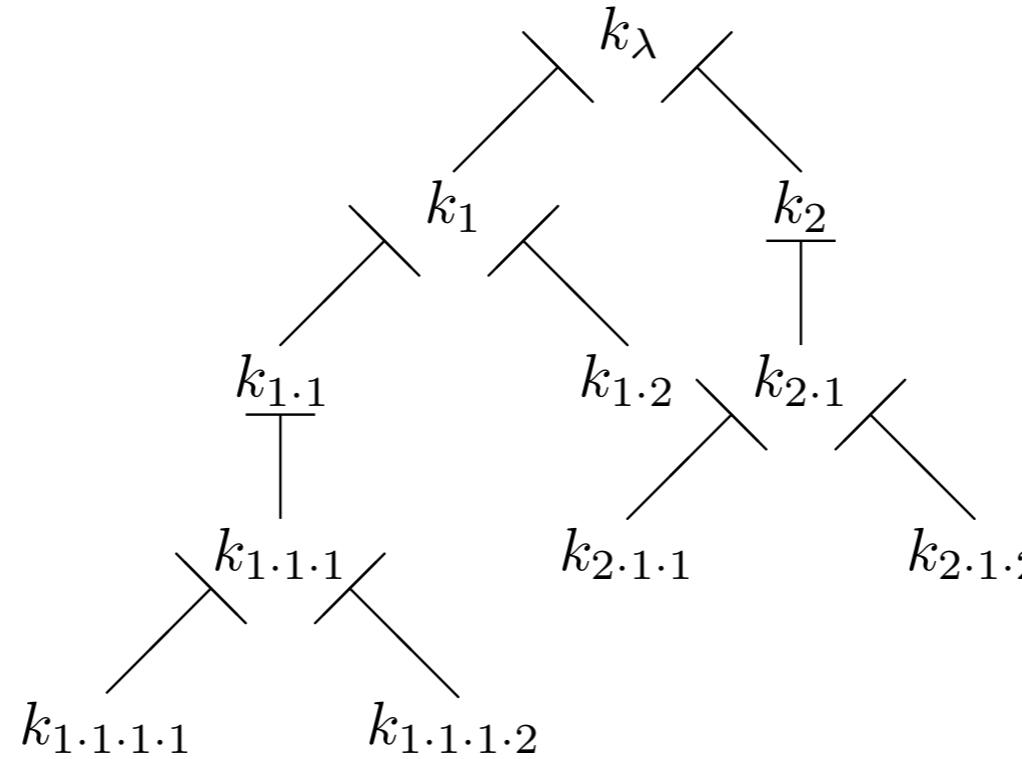
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

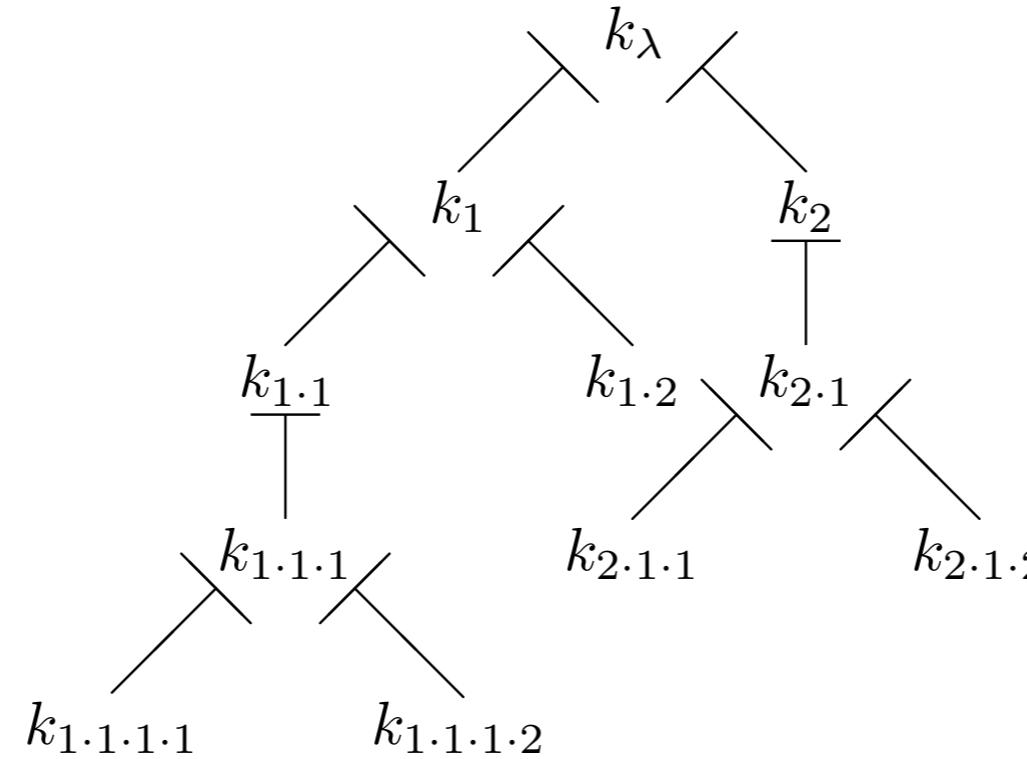
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

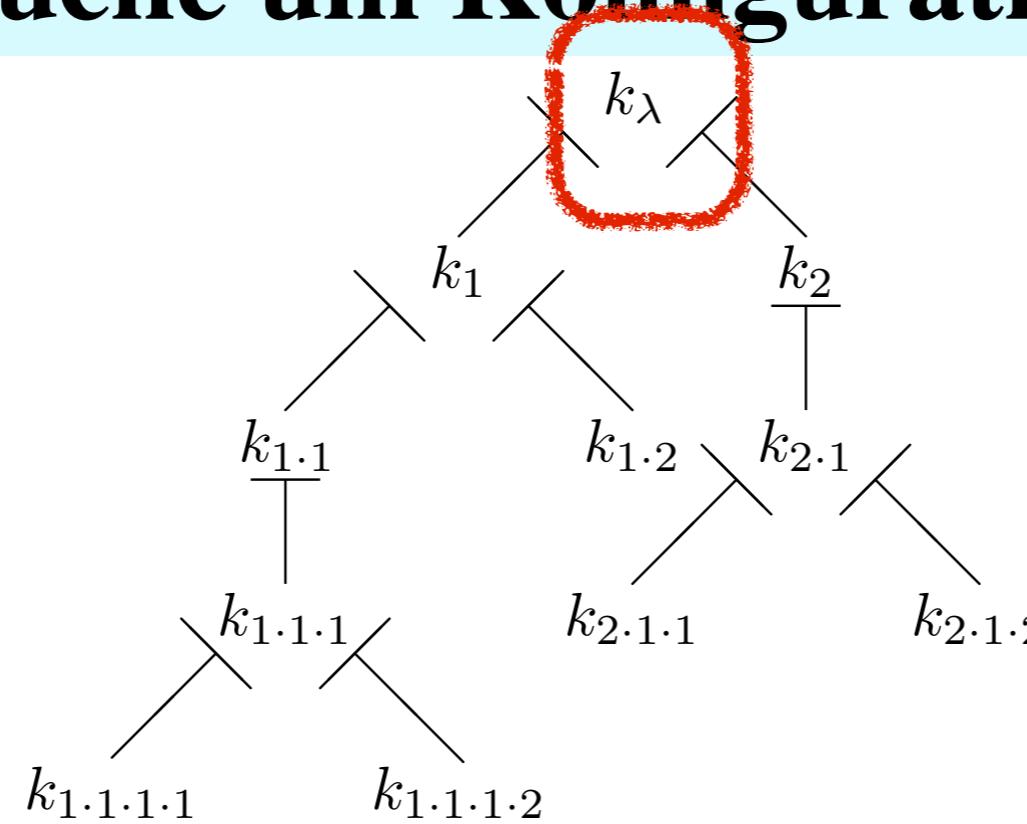
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

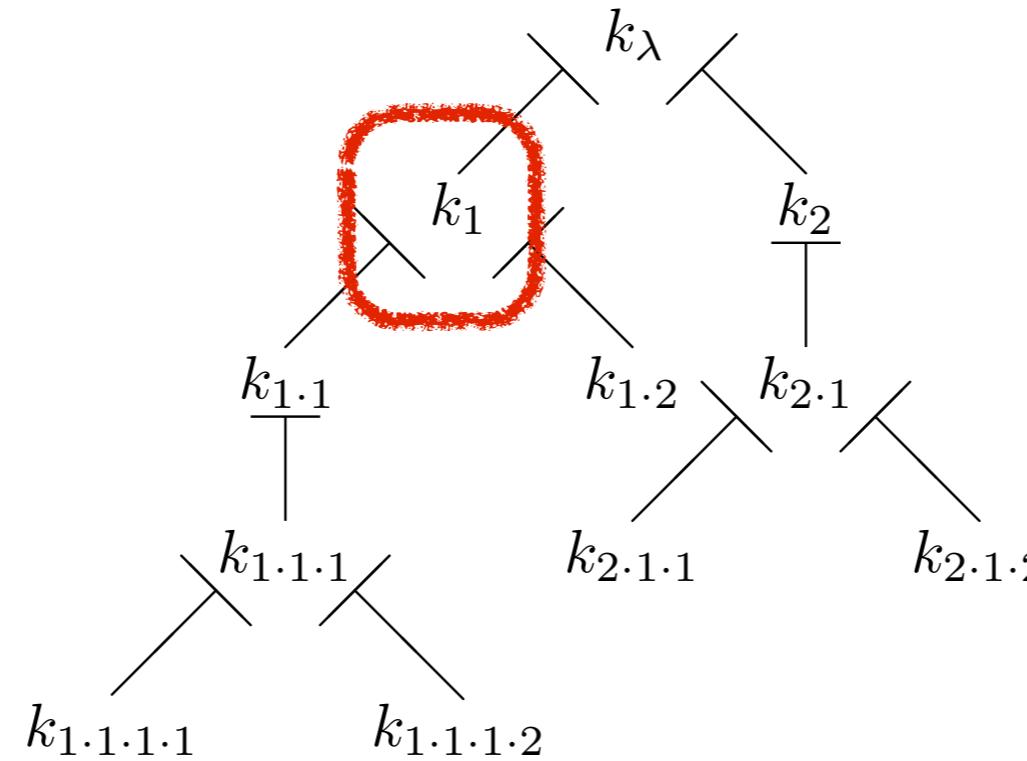
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

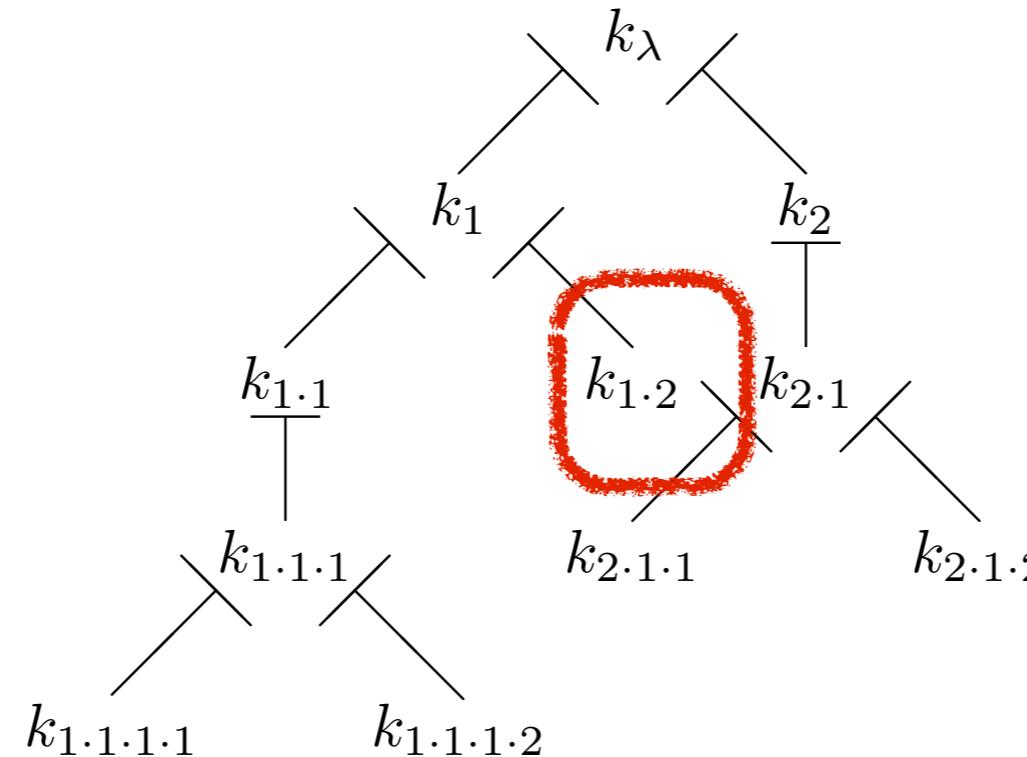
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

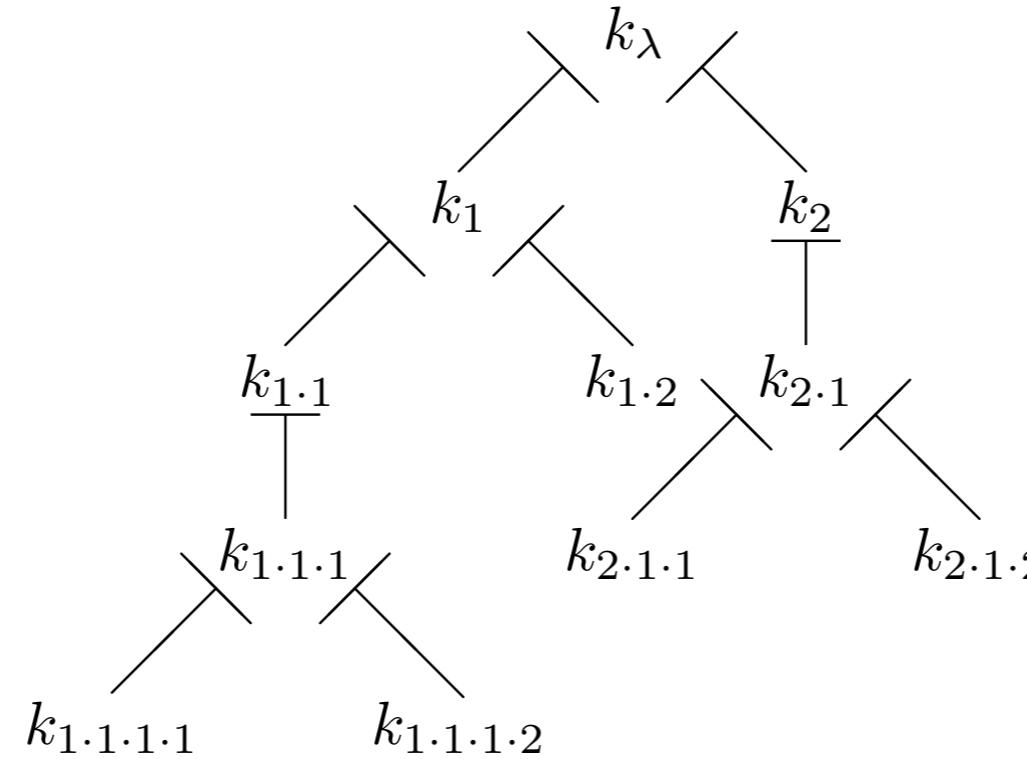
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

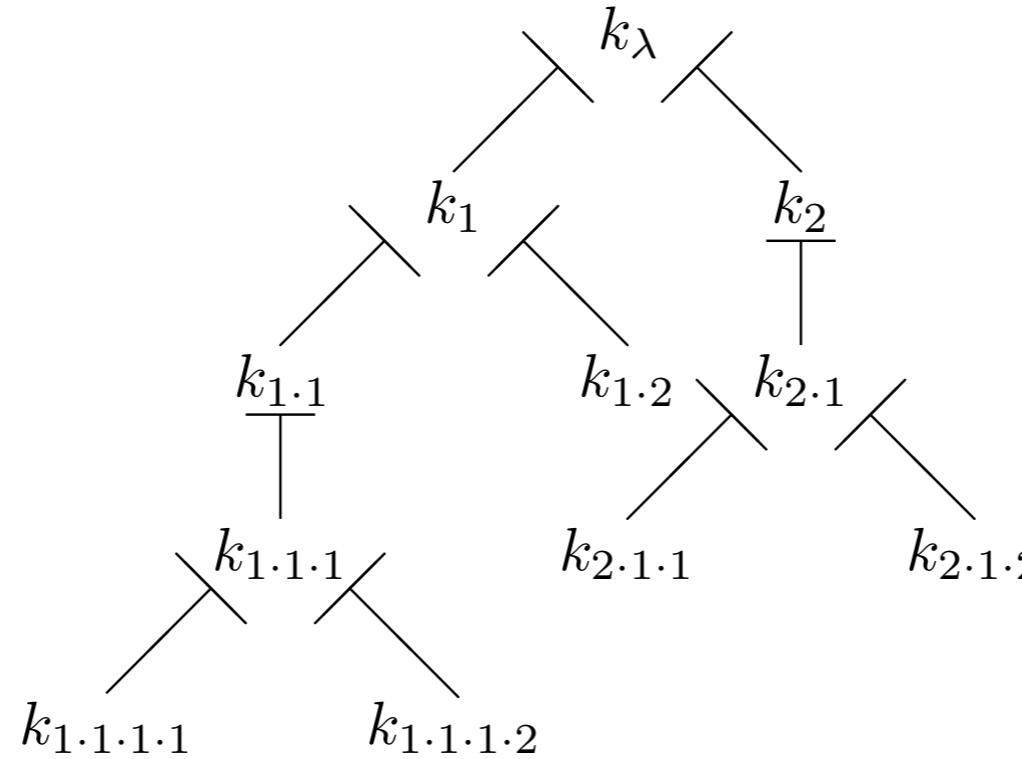
ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

Breitensuche um Konfigurationsbaum

ε
1
2
1.1
1.2
2.1
2.2
1.1.1
1.1.2



-
- Nach jeder auf dem zweiten Band erzeugten Numerierung $w \in \mathbb{N}^i$ simuliert B die Rechnung von A auf dem dritten Band, indem sie die durch w vorgegebenen Überführungen vornimmt, sofern dies in der TM A überhaupt möglich ist. Erreicht B dabei eine Endkonfiguration von A , so akzeptiert B . Da jede endliche Rechnung von A auf diese Weise irgendwann einmal vorkommt, wird von B natürlich genau die Sprache $L(A)$ akzeptiert. *qed.*

einseitiges Turingband

DTM mit linksseitig beschränktem Band

endliche Kontrolle

Kontrollzustände

Programm:
Übergangsfunktion δ

Lese-/Schreibkopf



Arbeitsband

Begrenzungssymbol (\$)

- * kann nicht überschrieben werden
- * kann nicht "nach links" überschritten werden

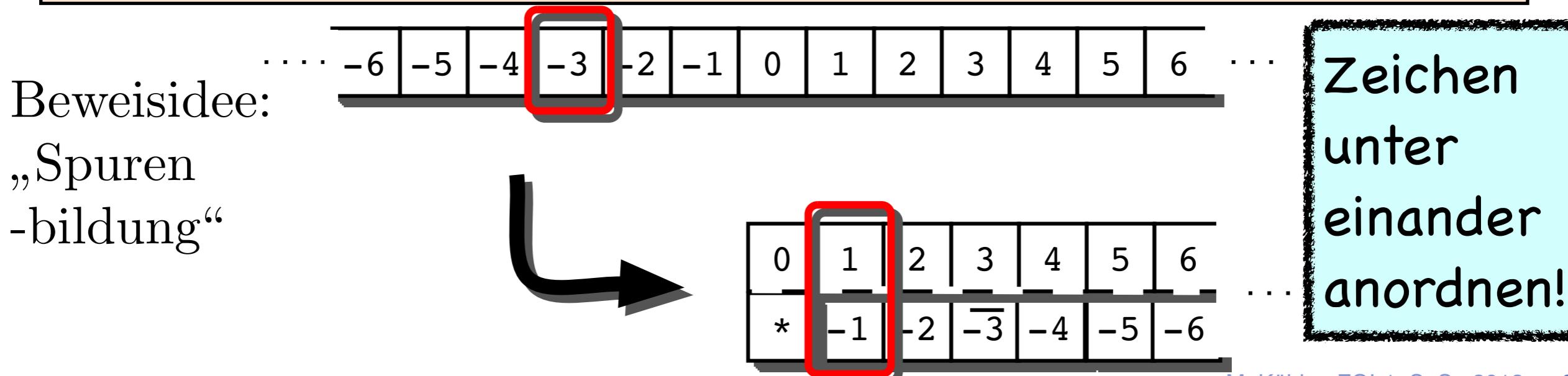
beidseitig / einseitig unendliches Arbeitsband

Definition 21.1:

Zwei Turingmaschinen A und B heißen genau dann **äquivalent**, wenn sie die gleiche Sprache akzeptieren, d.h. $L(A) = L(B)$ gilt.

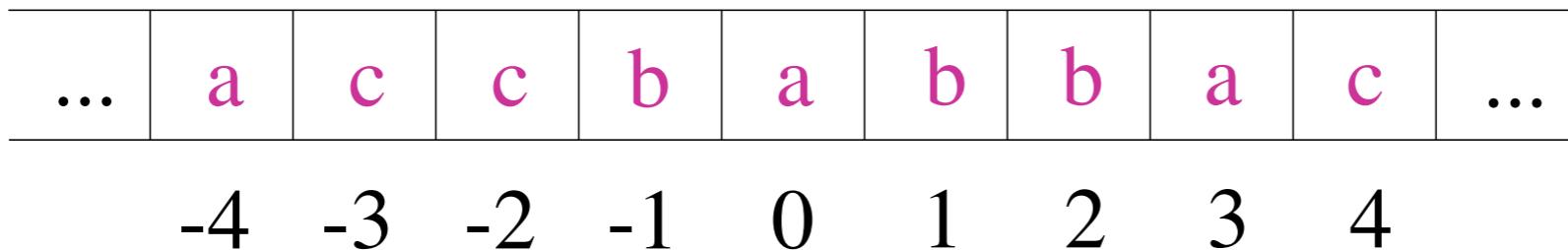
Satz 21.1:

Zu jeder DTM A mit *beidseitig* unendlichem Arbeitsband gibt es eine äquivalente DTM B mit *einseitig* unendlichem Arbeitsband und umgekehrt.

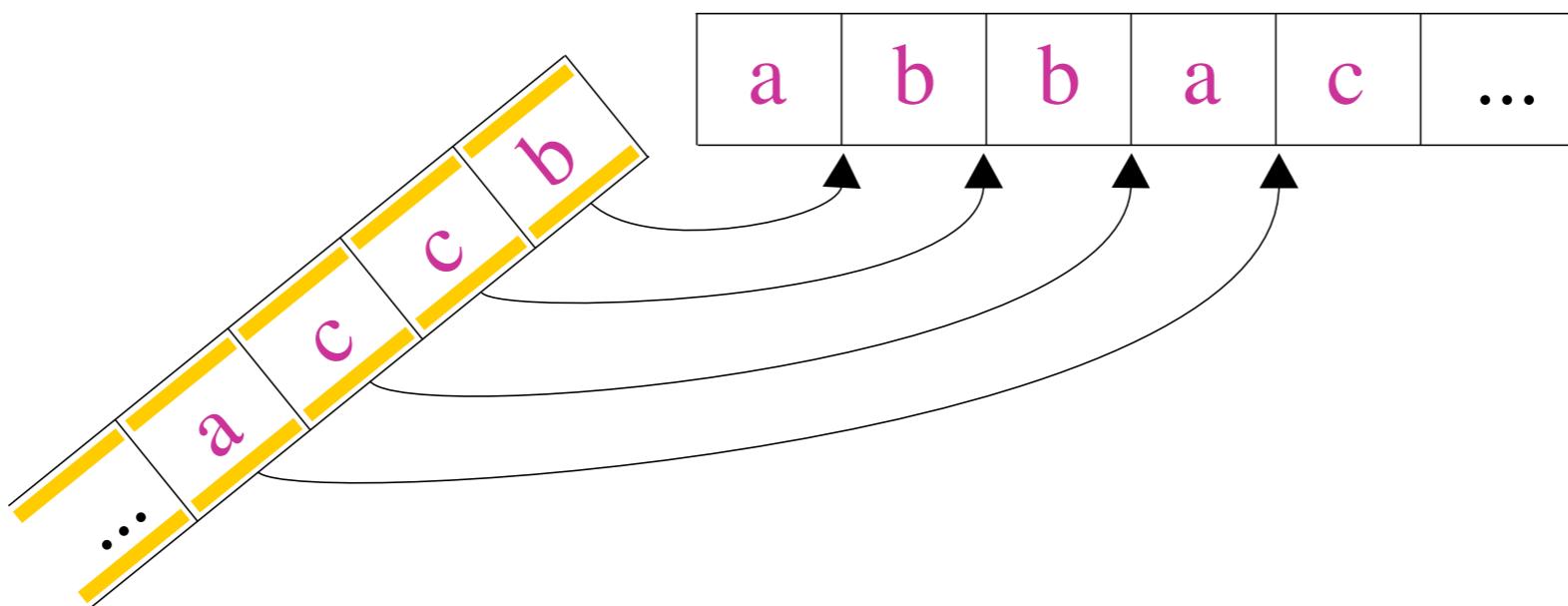


Spurenbildung anschaulicher

Arbeitsband der DTM



wird zerlegt in



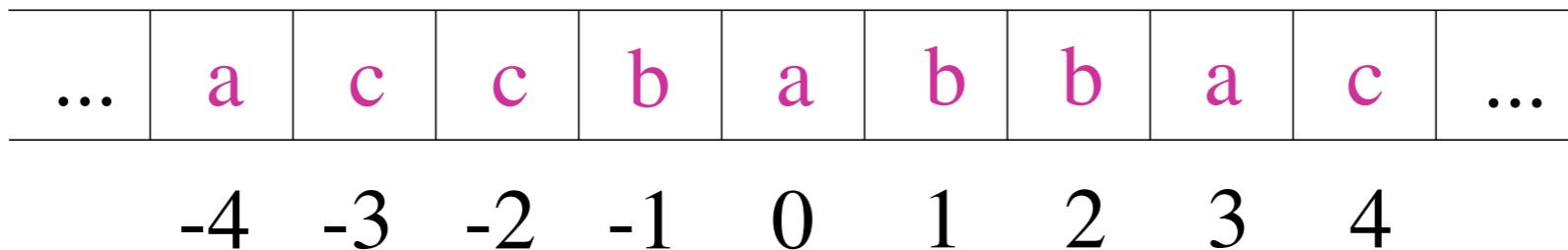
linksseitig beschränktes Arbeitsband

Zeichen
neben
einander
anordnen!

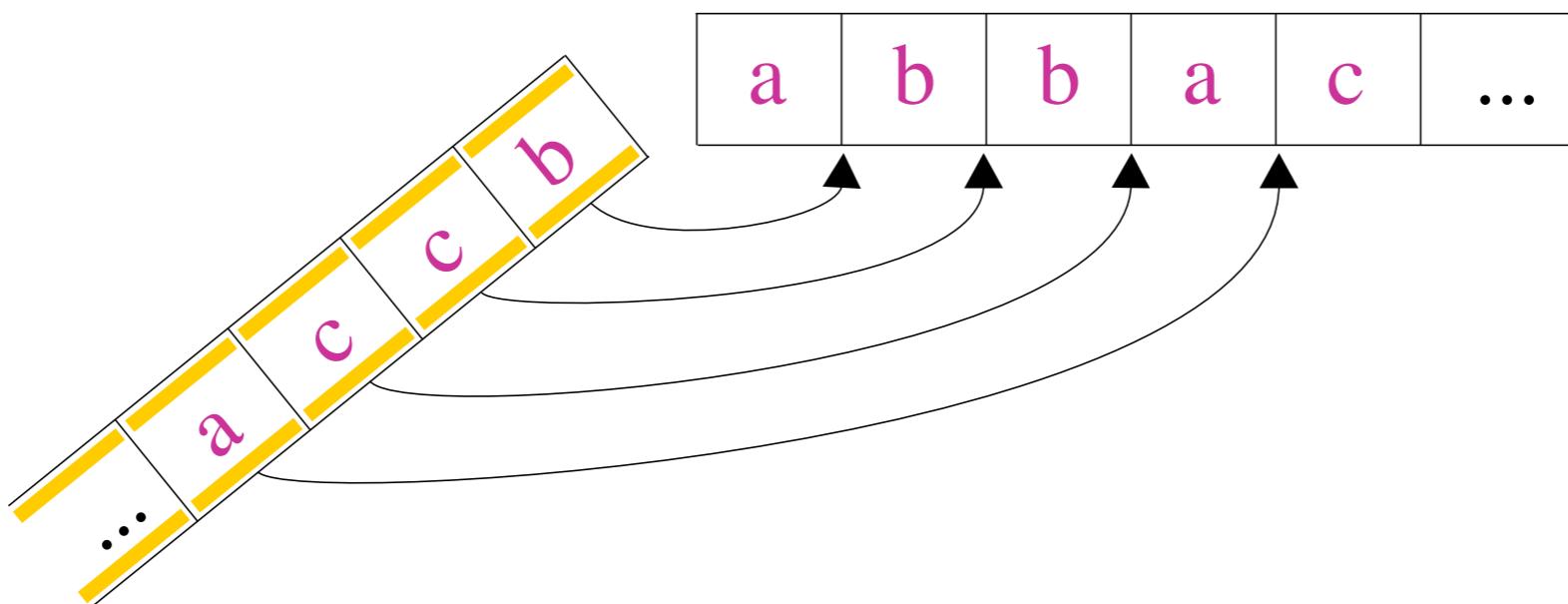
\$	a	b	b	c	b	c	a	a	c	...
----	---	---	---	---	---	---	---	---	---	-----

Spurenbildung anschaulicher

Arbeitsband der DTM

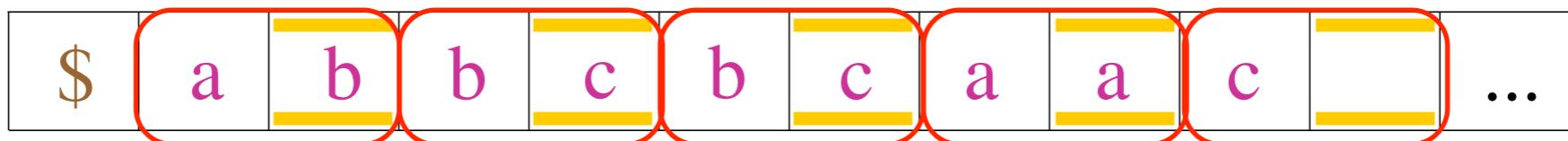


wird zerlegt in

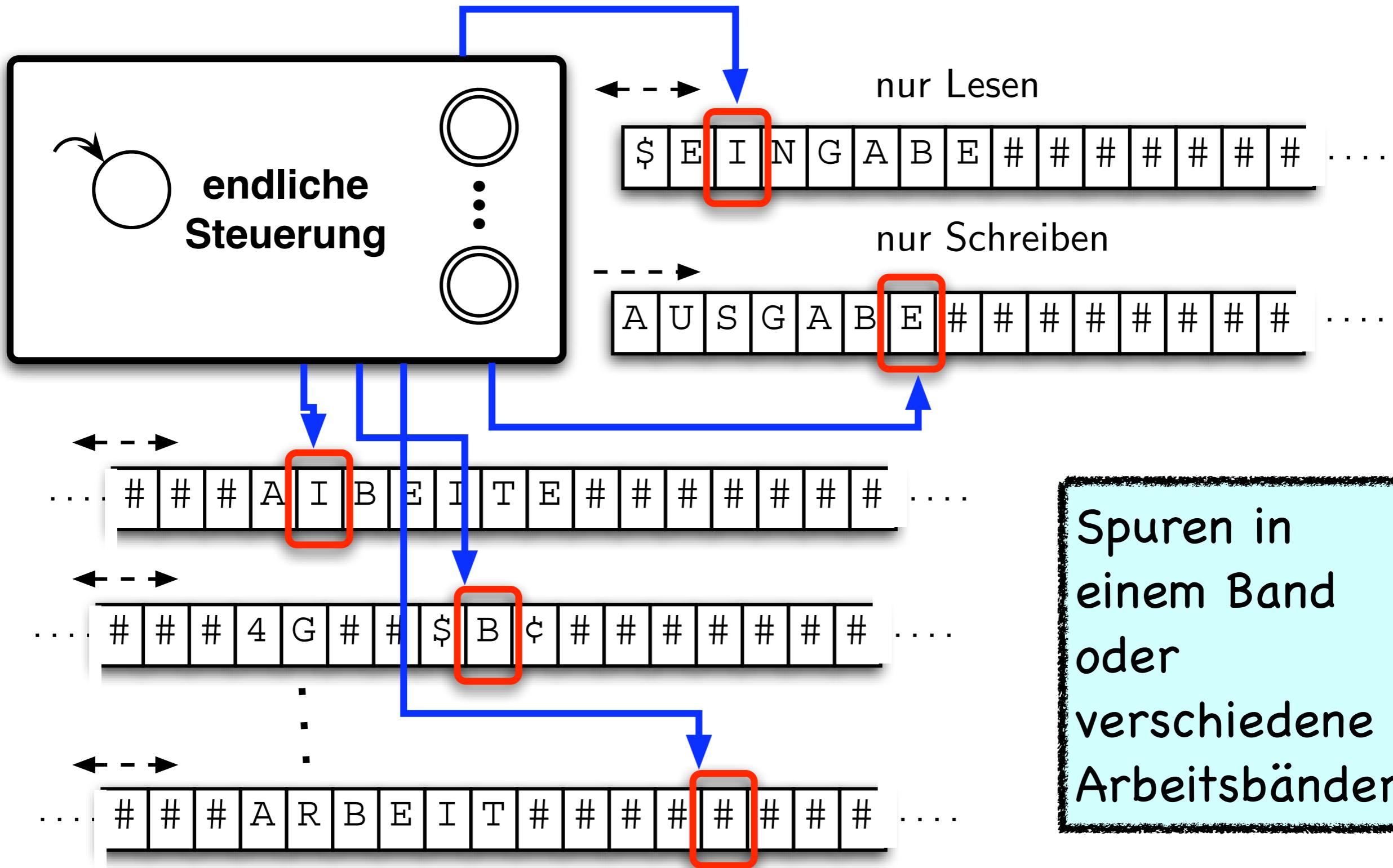


linksseitig beschränktes Arbeitsband

Zeichen
neben
einander
anordnen!



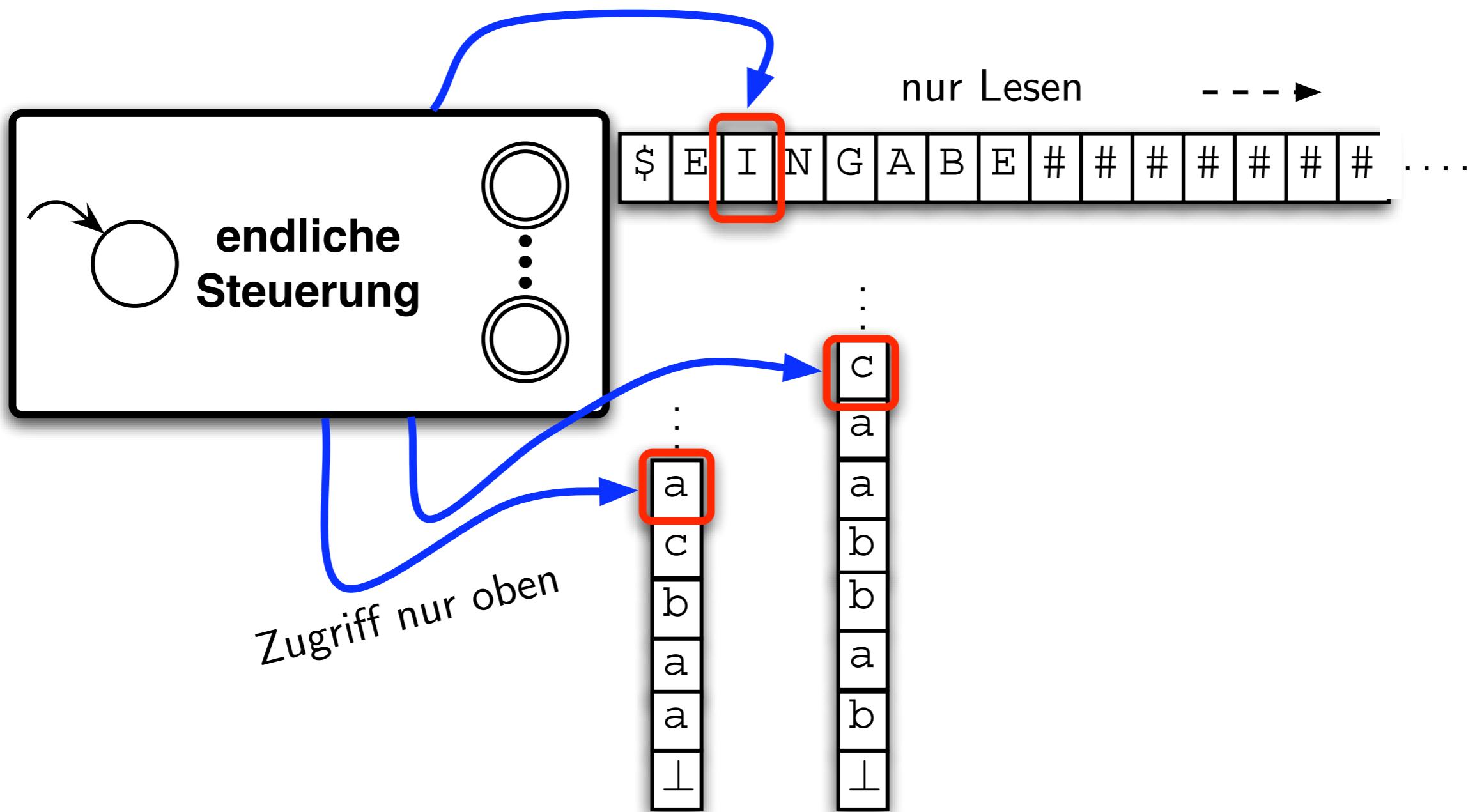
Die k -Band TM



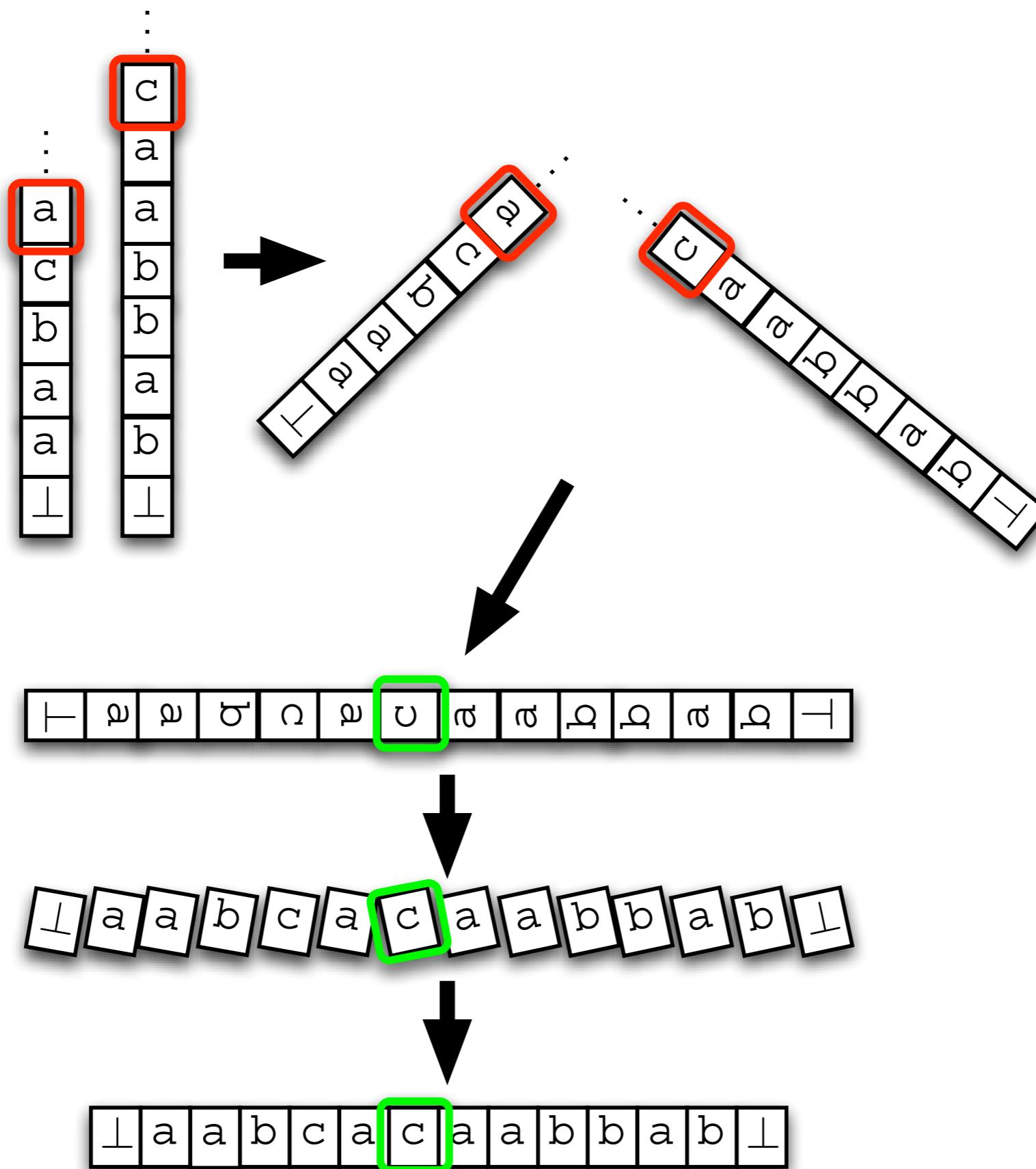
weiteres Modell: 2-Keller-Automat

Ein Kellerautomat akzeptiert nur kontextfreie Sprachen.

Mit **zwei** Kellerspeichern ausgestattet akzeptiert er schon alle aufzählbaren Sprachen!



TM-Simulation mit 2 Kellern



TM-Simulation mit 2 Zählern

Ein Zähler ist ein Keller, der nur ein einziges Kellersymbol enthält, aber dessen Leerheit festgestellt werden kann.

Also ist das Kellerbodenzeichen nur einmal und stets am Kellerboden vorhanden.

Es gibt noch ein zweites Symbol „zum Zählen“.

Es gibt deterministische und nichtdeterministische Zählerautomaten.

Satz 21.2:

Jede Turingmaschine kann durch einen 2-Zählerautomaten simuliert werden.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 18

Turingmaschinen: Aufzählbarkeit

Michael Köhler-Bußmeier

Aufzählung einer Sprache: TM als Generator

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.
- Jetzt wollen wir mit einer TM eine Sprache L generieren.

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.
- Jetzt wollen wir mit einer TM eine Sprache L generieren.
- Dazu wollen wir alle Wörter $w \in L$ in eine Reihenfolge $w_1 w_2 w_3 w_4 \dots$ bringen.

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.
- Jetzt wollen wir mit einer TM eine Sprache L generieren.
- Dazu wollen wir alle Wörter $w \in L$ in eine Reihenfolge $w_1 w_2 w_3 w_4 \dots$ bringen.
- Formal: Zu einem gegebenen n soll die TM das Wort w_n ausrechnen und auf das Band schreiben.

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.
- Jetzt wollen wir mit einer TM eine Sprache L generieren.
- Dazu wollen wir alle Wörter $w \in L$ in eine Reihenfolge $w_1 w_2 w_3 w_4 \dots$ bringen.
- Formal: Zu einem gegebenen n soll die TM das Wort w_n ausrechnen und auf das Band schreiben.
- Wir können dann die Sprache "aufzählen":

Aufzählung einer Sprache: TM als Generator

- Bislang haben wir TMs als Akzeptoren genutzt.
- Jetzt wollen wir mit einer TM eine Sprache L generieren.
- Dazu wollen wir alle Wörter $w \in L$ in eine Reihenfolge $w_1 w_2 w_3 w_4 \dots$ bringen.
- Formal: Zu einem gegebenen n soll die TM das Wort w_n ausrechnen und auf das Band schreiben.
- Wir können dann die Sprache "aufzählen":

```
n := 1
WHILE n>0 DO
    PRINT wn ON TAPE;
    n := n+1
ENDWHILE
```

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale Turing-berechenbare Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $g(\mathbb{N}) = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit $\mathcal{R}e$ (*recursively enumerable sets*) bezeichnet.

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale Turing-berechenbare Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit $\mathcal{R}e$ (*recursively enumerable sets*) bezeichnet.

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale Turing-berechenbare Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit $\mathcal{R}e$ (*recursively enumerable sets*) bezeichnet.

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale Turing-berechenbare Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit $\mathcal{R}e$ (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar**?

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale **Turing-berechenbare** Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit **Re** (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar**?

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (**rekursiv**) **aufzählbar** oder **beweisbar** genau dann, wenn

entweder $L = \emptyset$ ist, oder eine totale **Turing-berechenbare** Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die *Klasse aller aufzählbaren Mengen* wird mit **Re** (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar?**

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (~~rekursiv~~) aufzählbar oder beweisbar genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale ~~Turing-berechenbare~~ Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die Klasse aller aufzählbaren Mengen wird mit $\underline{\mathcal{R}e}$ (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar**?

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (~~rekursiv~~) ~~aufzählbar~~ oder ~~beweisbar~~ genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale ~~Turing-berechenbare~~ Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die Klasse aller aufzählbaren Mengen wird mit Re (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar**?

Aufzählbar versus NTM-akzeptierbar

Definition 18.11:

Eine Menge $L \subseteq \Sigma^*$ heißt (~~rekursiv~~) ~~aufzählbar~~ oder ~~beweisbar~~ genau dann, wenn entweder $L = \emptyset$ ist, oder eine totale ~~Turing-berechenbare~~ Funktion $g : \mathbb{N} \rightarrow \Sigma^*$ existiert, für die $\underline{g(\mathbb{N})} = L$ ist.

Die Klasse aller aufzählbaren Mengen wird mit \mathcal{R}_e (*recursively enumerable sets*) bezeichnet.

Wir kennen **abzählbare** Mengen.
Ist das das Gleiche wie **aufzählbar**?

Also ist jede **aufzählbare** Menge auch **abzählbar**.
Wir werden später sehen, dass die Umkehrung nicht gilt.

Definition Abzählbarkeit

Definition 18.1:

Eine Menge M heißt **abzählbar** genau dann, wenn sie entweder endlich ist oder gleich mächtig zur Menge \mathbb{N} der natürlichen Zahlen.

Definition Abzählbarkeit

Definition 18.1:

Eine Menge M heißt **abzählbar** genau dann, wenn sie entweder endlich ist oder gleich mächtig zur Menge \mathbb{N} der natürlichen Zahlen.

Bemerkung:

Für jede abzählbare Menge M gilt nach dieser Definition entweder

$$M = \emptyset$$

oder

es existiert eine surjektive Abbildung $g : \mathbb{N} \longrightarrow M$ auf die Menge M , kurz $g(\mathbb{N}) = M$.

Wir nennen g dann eine **Abzählung** von M . Bei abzählbaren unendlichen Mengen M kann immer auch eine Bijektion zwischen \mathbb{N} und M angegeben werden.

Beispiele für abzählbare Mengen

- Die natürlichen Zahlen $N = \{0, 1, 2, 3, \dots\}$
- Die geraden Zahlen $\{0, 2, 4, 6, \dots\}$
- Die ganzen Zahlen $Z = \{0, \pm 1, \pm 2, \pm 3, \dots\}$
- Die Paarmenge $N^2 = N \times N$
- Die rationalen Zahlen Q
- uvm...

Die Tupelabzählung

i	j	0	1	2	3	4	5	6	7	...
0		0	1	3	6	10	15	21	28	...
1		2	4	7	11	16	22	29	...	
2		5	8	12	17	23	30	...		
3		9	13	18	24	31	...			
4		14	19	25	32	...				
5		20	26	33	...					
6		27	34	...						
7		35	...							
⋮		...								

Für das k -te Tupel (i, j) gilt die Gleichung:

$$k = i + \sum_{r=0}^{i+j} r$$

Die Tupelabzählung

i	j	0	1	2	3	4	5	6	7	...
0	0	1	3	6	10	15	21	28	...	
1	2	4	7	11	16	22	29	...		
2	5	8	12	17	23	30	...			
3	9	13	18	24	31	...				
4	14	19	25	32	...					
5	20	26	33	...						
6	27	34	...							
7	35	...								
...	...									

Für das k -te Tupel (i, j) gilt die Gleichung:

$$k = i + \sum_{r=0}^{i+j} r$$

Welches Tupel steht wo?

Definition 18.14:

$\text{paar} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ sei definiert durch

$$\text{paar}(i, j) := i + \frac{(i+j)(i+j+1)}{2} = i + \binom{i+j+1}{2}$$

Mit dieser Definition sehen wir, dass auch die Menge
 $\mathbb{N} \times \mathbb{N}$
abzählbar ist.

Das kann nun iteriert werden,
 $\text{paar}(i, j, k) := \text{paar}(\text{paar}(i, j), k)$ etc., so dass

\mathbb{N}^k
für jedes k abzählbar ist!

Bei Gödelisierungen kommen wir darauf wieder zurück!

Diagonalbeweis der Überabzählbarkeit

Nachkommastellen der reellen Zahlen im Intervall $[0, 1]$:

	i_0	i_1	i_2	\dots	i_n	\dots
r_0	$i_{0,0}$	$i_{0,1}$	$i_{0,2}$	\dots	$i_{0,n}$	\dots
r_1	$i_{1,0}$	$i_{1,1}$	$i_{1,2}$	\dots	$i_{1,n}$	\dots
r_2	$i_{2,0}$	$i_{2,1}$	$i_{2,2}$	\dots	$i_{2,n}$	\dots
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
r_n	$i_{n,0}$	$i_{n,1}$	$i_{n,2}$	\dots	$i_{n,n}$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
r_d	$i_{0,0} + 1$	$i_{1,1} + 1$	$i_{2,2} + 1$	\dots	$i_{n,n} + 1$	\dots

Wären alle reellen Zahlen abzählbar, so auch die im Intervall $[0, 1]$. Da die Zahl

$$r_d := 0, (i_{0,0} + 1)(i_{1,1} + 1)(i_{2,2} + 1) \dots (i_{n,n} + 1) \dots$$

reell ist, müsste sie für ein $n \in \mathbb{N}$ in der Aufzählung als eine Zahl r_n vorkommen. In der Diagonale unterscheiden sich diese beiden Zahlen jedoch garantiert. (Addition von 1 geschehe modulo 10!).

Diagonalbeweis der Überabzählbarkeit

Nachkommastellen der reellen Zahlen im Intervall $[0, 1]$:

	i_0	i_1	i_2	\dots	i_n	\dots
r_0	$i_{0,0}$	$i_{0,1}$	$i_{0,2}$	\dots	$i_{0,n}$	\dots
r_1	$i_{1,0}$	$i_{1,1}$	$i_{1,2}$	\dots	$i_{1,n}$	\dots
r_2	$i_{2,0}$	$i_{2,1}$	$i_{2,2}$	\dots	$i_{2,n}$	\dots
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
r_n	$i_{n,0}$	$i_{n,1}$	$i_{n,2}$	\dots	$i_{n,n}$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots
r_d	$i_{0,0} + 1$	$i_{1,1} + 1$	$i_{2,2} + 1$	\dots	$i_{n,n} + 1$	\dots

Wären alle reellen Zahlen abzählbar, so auch die im Intervall $[0, 1)$. Da die Zahl

$$r_d := 0, (i_{0,0} + 1)(i_{1,1} + 1)(i_{2,2} + 1) \dots (i_{n,n} + 1) \dots$$

reell ist, müsste sie für ein $n \in \mathbb{N}$ in der Aufzählung als eine Zahl r_n vorkommen. In der Diagonalen unterscheiden sich diese beiden Zahlen jedoch garantiert. (Addition von 1 geschehe modulo 10!) M. Körber, FG-1, SoSe 2012

abzählbare Folgen von Zahlen sind nicht abzählbar

Satz 18.2:

Die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen aus \mathbb{N} ist nicht abzählbar.

i	$g(i)$	0	1	2	3	\dots	n	\dots
0	$g(0)$	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	\dots	$f_0(n)$	\dots
1	$g(1)$	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	\dots	$f_1(n)$	\dots
2	$g(2)$	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	\dots	$f_2(n)$	\dots
3	$g(3)$	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	\dots	$f_3(n)$	\dots
\vdots								
n	$g(n)$	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	\dots	$f_n(n)$	\dots
\vdots								

Wo ist die abzählbare Folge F_{diagonal} ?

$$F_{\text{diagonal}} := \langle f_0(0) + 1, f_1(1) + 1, f_2(2) + 1, f_3(3) + 1, f_4(4) + 1, \dots \rangle,$$

abzählbare Folgen von Zahlen sind nicht abzählbar

Satz 18.2:

Die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen aus \mathbb{N} ist nicht abzählbar.

i	$g(i)$	0	1	2	3	\dots	n	\dots
0	$g(0)$	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	\dots	$f_0(n)$	\dots
1	$g(1)$	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	\dots	$f_1(n)$	\dots
2	$g(2)$	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	\dots	$f_2(n)$	\dots
3	$g(3)$	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	\dots	$f_3(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\dots
n	$g(n)$	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	\dots	$f_n(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\dots	\vdots	\ddots

Wo ist die abzählbare Folge F_{diagonal} ?

$$F_{\text{diagonal}} := \langle f_0(0) + 1, f_1(1) + 1, f_2(2) + 1, f_3(3) + 1, f_4(4) + 1, \dots \rangle,$$

abzählbare Folgen von Zahlen sind nicht abzählbar

Satz 18.2:

Die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen aus \mathbb{N} ist nicht abzählbar.

i	$g(i)$	0	1	2	3	\dots	n	\dots
0	$g(0)$	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	\dots	$f_0(n)$	\dots
1	$g(1)$	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	\dots	$f_1(n)$	\dots
2	$g(2)$	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	\dots	$f_2(n)$	\dots
3	$g(3)$	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	\dots	$f_3(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\dots
n	$g(n)$	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	\dots	$f_n(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\dots	\vdots	\ddots

Wo ist die abzählbare Folge F_{diagonal} ?

$$F_{\text{diagonal}} := \langle f_0(0) + 1, f_1(1) + 1, f_2(2) + 1, f_3(3) + 1, f_4(4) + 1, \dots \rangle,$$

abzählbare Folgen von Zahlen sind nicht abzählbar

Satz 18.2:

Die Menge aller abzählbar unendlichen Folgen natürlicher Zahlen aus \mathbb{N} ist nicht abzählbar.

i	$g(i)$	0	1	2	3	\dots	n	\dots
0	$g(0)$	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	\dots	$f_0(n)$	\dots
1	$g(1)$	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	\dots	$f_1(n)$	\dots
2	$g(2)$	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	\dots	$f_2(n)$	\dots
3	$g(3)$	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	\dots	$f_3(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\dots
n	$g(n)$	$f_n(0)$	$f_n(1)$	$f_n(2)$	$f_n(3)$	\dots	$f_n(n)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\dots	\vdots	\ddots

Wo ist die abzählbare Folge F_{diagonal} ?

$$F_{\text{diagonal}} := \langle f_0(0) + 1, f_1(1) + 1, f_2(2) + 1, f_3(3) + 1, f_4(4) + 1, \dots \rangle,$$

Nichtabzählbarkeit der Potenzmenge

Satz 18.3:

Die Potenzmenge 2^M einer abzählbaren Menge M ist genau dann abzählbar, wenn M eine endliche Menge ist.

Im Theorem sind zwei Aussagen versteckt, die beide bewiesen werden müssen:

- (a) Wenn M *nicht* endlich ist, so ist die Menge 2^M aller Teilmengen von M nicht abzählbar.
- (b) Wenn M endliche Menge ist, so ist 2^M abzählbar.

Zu (b):

Für eine endliche Menge M hat die Potenzmenge von M gerade $2^{|M|}$ viele Elemente. Somit ist auch die Menge 2^M endlich und damit per Definition abzählbar.

Nichtabzählbarkeit der Potenzmenge

Satz 18.3:

Die Potenzmenge 2^M einer abzählbaren Menge M ist genau dann abzählbar, wenn M eine endliche Menge ist.

Im Theorem sind zwei Aussagen versteckt, die beide bewiesen werden müssen:

- (a) Wenn M *nicht* endlich ist, so ist die Menge 2^M aller Teilmengen von M nicht abzählbar.
- (b) Wenn M endliche Menge ist, so ist 2^M abzählbar.

Zu (b):

Für eine endliche Menge M hat die Potenzmenge von M gerade $2^{|M|}$ viele Elemente. Somit ist auch die Menge 2^M endlich und damit per Definition abzählbar.

Nichtabzählbarkeit der Potenzmenge

Satz 18.3:

Die Potenzmenge 2^M einer abzählbaren Menge M ist genau dann abzählbar, wenn M eine endliche Menge ist.

Im Theorem sind zwei Aussagen versteckt, die beide bewiesen werden müssen:

- (a) Wenn M *nicht* endlich ist, so ist die Menge 2^M aller Teilmengen von M nicht abzählbar.
- (b) Wenn M endliche Menge ist, so ist 2^M abzählbar.

Zu (b):

Für eine endliche Menge M hat die Potenzmenge von M gerade $2^{|M|}$ viele Elemente. Somit ist auch die Menge 2^M endlich und damit per Definition abzählbar.

zum Beweis von (a)

Zu (a):

Sei M eine abzählbare unendliche Menge mit der surjektiven Abbildung $f : \mathbb{N} \longrightarrow M$ zur Abzählung. Für jedes $i \in \mathbb{N}$ bezeichnet also $f(i)$ eine bestimmtes Element von M .

O.B.d.A. dürfen wir annehmen, dass in dieser Abzählung von M keines der Elemente doppelt auftritt, dass also f sogar eine Bijektion ist!

Falls nun die Potenzmenge 2^M ebenfalls abzählbar ist, so gibt es auch für diese Menge eine Abzählung $g : \mathbb{N} \longrightarrow 2^M$.

Wir definieren die Diagonalmenge

$$D := \{f(j) \mid f(j) \notin g(j), j \in \mathbb{N}\}.$$

Offensichtlich ist $D \subseteq M$ und also $D \in 2^M$. Daher muss es eine Zahl $k \in \mathbb{N}$ mit $D = g(k)$ geben. Auf Grund der Definition der Menge D bedeutet dies nun aber, dass $f(k) \in D$ genau dann gilt, wenn $f(k) \notin g(k)$, also genau dann, wenn $f(k) \notin D$ gilt. Also ein Widerspruch und 2^M kann nicht abzählbar sein.

zum Beweis von (a)

Zu (a):

Sei M eine abzählbare unendliche Menge mit der surjektiven Abbildung $f : \mathbb{N} \longrightarrow M$ zur Abzählung. Für jedes $i \in \mathbb{N}$ bezeichnet also $f(i)$ eine bestimmtes Element von M .

O.B.d.A. dürfen wir annehmen, dass in dieser Abzählung von M keines der Elemente doppelt auftritt, dass also f sogar eine Bijektion ist!

Falls nun die Potenzmenge 2^M ebenfalls abzählbar ist, so gibt es auch für diese Menge eine Abzählung $g : \mathbb{N} \longrightarrow 2^M$.

Wir definieren die Diagonalmenge

$$D := \{f(j) \mid f(j) \notin g(j), j \in \mathbb{N}\}.$$

Offensichtlich ist $D \subseteq M$ und also $D \in 2^M$. Daher muss es eine Zahl $k \in \mathbb{N}$ mit $D = g(k)$ geben. Auf Grund der Definition der Menge D bedeutet dies nun aber, dass $f(k) \in D$ genau dann gilt, wenn $f(k) \notin g(k)$, also genau dann, wenn $f(k) \notin D$ gilt. Also ein Widerspruch und 2^M kann nicht abzählbar sein.

zum Beweis von (a)

Zu (a):

Sei M eine abzählbare unendliche Menge mit der surjektiven Abbildung $f : \mathbb{N} \longrightarrow M$ zur Abzählung. Für jedes $i \in \mathbb{N}$ bezeichnet also $f(i)$ eine bestimmtes Element von M .

O.B.d.A. dürfen wir annehmen, dass in dieser Abzählung von M keines der Elemente doppelt auftritt, dass also f sogar eine Bijektion ist!

Falls nun die Potenzmenge 2^M ebenfalls abzählbar ist, so gibt es auch für diese Menge eine Abzählung $g : \mathbb{N} \longrightarrow 2^M$.

Wir definieren die Diagonalmenge

$$D := \{f(j) \mid f(j) \notin g(j), j \in \mathbb{N}\}.$$

Offensichtlich ist $D \subseteq M$ und also $D \in 2^M$. Daher muss es eine Zahl $k \in \mathbb{N}$ mit $D = g(k)$ geben. Auf Grund der Definition der Menge D bedeutet dies nun aber, dass $f(k) \in D$ genau dann gilt, wenn $f(k) \notin g(k)$, also genau dann, wenn $f(k) \notin D$ gilt. Also ein Widerspruch und 2^M kann nicht abzählbar sein.

zum Beweis von (a)

Zu (a):

Sei M eine abzählbare unendliche Menge mit der surjektiven Abbildung $f : \mathbb{N} \longrightarrow M$ zur Abzählung. Für jedes $i \in \mathbb{N}$ bezeichnet also $f(i)$ eine bestimmtes Element von M .

O.B.d.A. dürfen wir annehmen, dass in dieser Abzählung von M keines der Elemente doppelt auftritt, dass also f sogar eine Bijektion ist!

Falls nun die Potenzmenge 2^M ebenfalls abzählbar ist, so gibt es auch für diese Menge eine Abzählung $g : \mathbb{N} \longrightarrow 2^M$.

Wir definieren die Diagonalmenge

$$D := \{f(j) \mid f(j) \notin g(j), j \in \mathbb{N}\}.$$

Offensichtlich ist $D \subseteq M$ und also $D \in 2^M$. Daher muss es eine Zahl $k \in \mathbb{N}$ mit $D = g(k)$ geben. Auf Grund der Definition der Menge D bedeutet dies nun aber, dass $f(k) \in D$ genau dann gilt, wenn $f(k) \notin g(k)$, also genau dann, wenn $f(k) \notin D$ gilt. Also ein Widerspruch und 2^M kann nicht abzählbar sein.

Ordnungsrelation und Abzählung für Σ^*

Definition 18.2:

Sei (Σ, \prec) ein mit \prec linear geordnetes Alphabet, dann ist die **lexikographische Erweiterung** $\prec^{\text{lex}} \subseteq \Sigma^* \times \Sigma^*$ von \prec definiert durch:

1. $\epsilon \prec^{\text{lex}} w$ für alle $w \in \Sigma^+$.
2. $\forall x, y \in \Sigma \forall u, v \in \Sigma^* : (xu \prec^{\text{lex}} yv)$ genau dann, wenn
$$\begin{cases} (x \prec y) \text{ oder} \\ (x = y) \text{ und } (u \prec^{\text{lex}} v) \end{cases}$$

$$\preceq^{\text{lex}} := \prec^{\text{lex}} \cup Id_{\Sigma^*}$$

nennen wir **lexikographische Ordnung** auf Σ^* .

Es gilt unter der üblichen linearen Ordnung der Symbole des lateinischen Alphabets folgende lexikographische Anordnung:

ABBI \prec^{lex} KLAUSUR \prec^{lex} PI \prec^{lex} UNI \prec^{lex} ZOO

Eigenschaften der lexikographischen Ordnung

Die lexikographische Ordnung \prec^{lex} ist total, und besitzt kein Supremum und keine maximalen Elemente!

Die Kette $a \prec^{\text{lex}} aa \prec^{\text{lex}} aaa \prec^{\text{lex}} \dots$ kann unendlich fortgesetzt werden und jedes Element der ebenfalls unendlichen Kette $b \prec^{\text{lex}} bb \prec^{\text{lex}} bbb \prec^{\text{lex}} \dots$ ist bezüglich \prec^{lex} größer als jedes beliebige Element a^i der ersten unendlichen, aufsteigenden Kette. Diese Ordnung eignet sich nicht als Grundlage für eine Abzählung aller Wörter (falls $|\Sigma| \neq 1$)!

Eigenschaften der lexikographischen Ordnung

Die lexikographische Ordnung \prec^{lex} ist total, und besitzt kein Supremum und keine maximalen Elemente!

Die Kette $a \prec^{\text{lex}} aa \prec^{\text{lex}} aaa \prec^{\text{lex}} \dots$ kann unendlich fortgesetzt werden und jedes Element der ebenfalls unendlichen Kette $b \prec^{\text{lex}} bb \prec^{\text{lex}} bbb \prec^{\text{lex}} \dots$ ist bezüglich \prec^{lex} größer als jedes beliebige Element a^i der ersten unendlichen, aufsteigenden Kette. Diese Ordnung eignet sich nicht als Grundlage für eine Abzählung aller Wörter (falls $|\Sigma| \neq 1$)!

Eigenschaften der lexikographischen Ordnung

Die lexikographische Ordnung \prec^{lex} ist total, und besitzt kein Supremum und keine maximalen Elemente!

Die Kette $a \prec^{\text{lex}} aa \prec^{\text{lex}} aaa \prec^{\text{lex}} \dots$ kann unendlich fortgesetzt werden und jedes Element der ebenfalls unendlichen Kette $b \prec^{\text{lex}} bb \prec^{\text{lex}} bbb \prec^{\text{lex}} \dots$ ist bezüglich \prec^{lex} größer als jedes beliebige Element a^i der ersten unendlichen, aufsteigenden Kette. Diese Ordnung eignet sich nicht als Grundlage für eine Abzählung aller Wörter (falls $|\Sigma| \neq 1$)!

Gesucht: eine andere Ordnung, die als Abzählung dienen kann.

Lexikalische (oder kanonische) Ordnung

Definition 18.3:

Sei (Σ, \prec) ein mit \prec linear geordnetes Alphabet, dann ist die **lexikalische** Erweiterung $\preceq^{\text{lg-lex}}$ von \prec definiert durch:

$w_1 \preceq^{\text{lg-lex}} w_2$ gelte für beliebige Wörter $w_1, w_2 \in \Sigma^*$ genau dann, wenn

entweder: $|w_1| < |w_2|$, oder: $|w_1| = |w_2| \wedge w_1 \prec^{\text{lex}} w_2$.

$\preceq^{\text{lg-lex}} := \prec^{\text{lg-lex}} \cup Id_{\Sigma^*}$ heißt **lexikalische Ordnung** auf Σ^* .

Σ^* ist in der lexikalischen Ordnung abzählbar!

Es gilt unter der üblichen linearen Ordnung der Symbole des lateinischen Alphabets folgende lexikographische Anordnung:

$$\text{PI} \prec^{\text{lg-lex}} \text{UNI} \prec^{\text{lg-lex}} \text{ZOO} \prec^{\text{lg-lex}} \text{ABBI} \prec^{\text{lg-lex}} \text{KLAUSUR}$$

Zur Erinnerung:

Definition 12.8:

- Die **b -adische Darstellung** benutzt die Symbole der Ziffernmenge $B := \{1, 2, \dots, b\}$. Ist $b = 2$, so spricht man von einer **dyadischen** Zahlendarstellung.
- Eine natürliche Zahl $n \in \mathbb{N}, n \neq 0$ wird zur Basis $b = |B|$ durch eine Folge $a_k a_{k-1} \cdots a_0$ von Symbolen $a_i \in \{1, \dots, b\}$ notiert, wenn $n = \sum_{i=0}^k a_i \cdot b^i$ gilt.

Wir notieren dann $[a_k a_{k-1} \cdots a_0]_{b\text{-adic}} = n$.

b-adisch = lexikalisch

Für das Alphabet $\Sigma := \{x_1, x_2, \dots, x_b\}$ kann die b -adische Darstellung benutzt werden, indem einem Wort $w = x_{i_1}x_{i_2} \cdots x_{i_m}$ mit $x_{i_j} \in \Sigma$ die Zahl $[i_1 i_2 \cdots i_m]_{b\text{-adic}}$ zugeordnet wird.

Satz 18.1:

Das Wort $w \in \Sigma^*$, welches in der lexikalischen Ordnung $\prec^{\text{lg-lex}}$ zu einem geordneten Alphabet $(\{x_1, x_2, \dots, x_b\}, \prec)$ an n -ter Stelle steht ist auch das n -te Wort in der $|\Sigma|$ -adischen Abzählung von Σ^* !

Also: $u \preceq^{\text{lg-lex}} v \iff [u]_{b\text{-adic}} < [v]_{b\text{-adic}}$

b-adisch = lexikalisch

Für das Alphabet $\Sigma := \{x_1, x_2, \dots, x_b\}$ kann die b -adische Darstellung benutzt werden, indem einem Wort $w = x_{i_1}x_{i_2} \cdots x_{i_m}$ mit $x_{i_j} \in \Sigma$ die Zahl $[i_1 i_2 \cdots i_m]_{b\text{-adic}}$ zugeordnet wird.

Satz 18.1:

Das Wort $w \in \Sigma^*$, welches in der lexikalischen Ordnung $\prec^{\text{lg-lex}}$ zu einem geordneten Alphabet $(\{x_1, x_2, \dots, x_b\}, \prec)$ an n -ter Stelle steht ist auch das n -te Wort in der $|\Sigma|$ -adischen Abzählung von Σ^* !

Also: $u \preceq^{\text{lg-lex}} v \iff [u]_{b\text{-adic}} < [v]_{b\text{-adic}}$

aufzählbar versus NTM

Was verbindet Mengen, die von NTM akzeptiert werden, mit **aufzählbaren Mengen**?

• **rekursiv aufzählbar**

= alle Worte können durchnummeriert werden

• **Turing-akzeptierbar**

= alle Worte können erkannt werden

Satz 18.4:

$$\mathcal{R}e = \bigcup_{\Sigma \text{ endl. Alphabet}} TA_{\Sigma}$$

aufzählbar versus NTM

Was verbindet Mengen, die von NTM akzeptiert werden, mit **aufzählbaren Mengen**?

• **rekursiv aufzählbar**

= alle Worte können durchnummeriert werden

• **Turing-akzeptierbar**

= alle Worte können erkannt werden

Satz 18.4:

$$\mathcal{R}e = \bigcup_{\Sigma \text{ endl. Alphabet}} TA_{\Sigma}$$

aufzählbar versus NTM (Beweisskizze)

Beweisidee: $L = L(M) \Rightarrow \exists \text{ TM } A_f$, zur Berechnung von $f : \mathbb{N} \rightarrow \Sigma^*$ mit $f(i) \in L(M)$ und $f(\mathbb{N}) = L(M)$:

Ist $L = \emptyset$, so ist nichts zu zeigen. Wenn dagegen $L \neq \emptyset$ gilt, dann existiert mindestens ein $w_L \in L$.

- Für die Eingabe n berechnen wir das n -te Paar (i, j) .
 - Sei $w = w_i$ das i -te Wort in der lexikalischen Ordnung.
 - Wir simulieren M genau j Schritte auf w_i .
 - Wird w_i von M im Schritt j akzeptiert, so geben wir w_i aus: $f(n) = w_i$.
 - Wird w_i von M im Schritt j nicht akzeptiert, so geben wir w_L aus: $f(n) = w_L$.
-
- Im Bild von f befindet sich nur Worte aus L , denn nur wenn M akzeptiert, definieren wir $f(n) = w_i$. Also gilt $f(\mathbb{N}) \subseteq L$
 - Da jedes akzeptierte Wort eine kürzeste Rechnung besitzt, existiert ein Paar (i, j) , so dass M im Schritt j das Wort w_i erstmaligst akzeptiert. Also gilt $f(\mathbb{N}) \supseteq L$

aufzählbar versus NTM (Beweisskizze)

Umgekehrt: Wenn L aufzählbar ist, dann ist L auch akzeptierbar.

$\exists \text{ TM } A_f$, mit $f(N) = L \Rightarrow \exists \text{ TM } M : L = L(M)$

- Verwende Zähler i , beginne mit $i = 0$.
- Berechne mit A_f das Wort $f(i) \in L$.
- Kontrolliere, ob $f(i)$ das Eingabewort w ist?
Ja: akzeptiere! Nein: inkrementiere i .

Da jedes akzeptierte Wort w ein Urbild i besitzt,
wird die TM M das Wort daher irgendwann akzeptieren.

aufzählbar versus NTM (Beweisskizze)

Umgekehrt: Wenn L aufzählbar ist, dann ist L auch akzeptierbar.

$\exists \text{ TM } A_f$, mit $f(N) = L \Rightarrow \exists \text{ TM } M : L = L(M)$

- Verwende Zähler i , beginne mit $i = 0$.
- Berechne mit A_f das Wort $f(i) \in L$.
- Kontrolliere, ob $f(i)$ das Eingabewort w ist?
Ja: akzeptiere! Nein: inkrementiere i .

Da jedes akzeptierte Wort w ein Urbild i besitzt,
wird die TM M das Wort daher irgendwann akzeptieren.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 19 Berechenbare Funktionen

Michael Köhler-Bußmeier

Wortfunktion versus ganzahlige Berechnungen

Zur Erinnerung: (Turing-)berechenbare Wortfunktionen:

Definition 19.1:

Seien Σ und Λ Alphabete.

Eine partielle (Wort-)Funktion $f : \Sigma^* \rightarrow \Lambda^*$ heißt **Turing-berechenbar** oder **partiell rekursiv** genau dann, wenn es eine $DTM T = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ gibt mit:

$$q_0 w \xrightarrow[T]{*} q_e v, \quad \text{für } q_e \in F \text{ und } v \in \Lambda^* \text{ mit } f(w) = v.$$

... der Funktionswert steht *ab der Kopfposition* auf dem Band
(abgesehen von nachfolgenden #'s)!

Wenn die $DTM T$ die Eingabe verwirft, oder nie anhält, so soll $f(w) := \perp$ die Undefiniertheit von f anzeigen.

TM-berechenbare Funktionen

Wie definieren Vossen/Witt
(Turing-)berechenbare Zahlenfunktionen?

Definition 19.2:

Seien $r, s \in \mathbb{N}$ und $r, s \geq 1$.

Eine (partielle) Funktion $f : \mathbb{N}^r \rightarrow \mathbb{N}^s$ heißt **Turingberechenbar** oder **partiell rekursiv** gdw. eine DTM $T = (Q, \{|, 0\}, \Gamma, \delta, q_0, \#, F)$ existiert mit

$$q_0 |^{m_1} 0 |^{m_2} 0 \dots |^{m_r} \xrightarrow[T]{*} p |^{n_1} 0 |^{n_2} 0 \dots |^{n_s}$$

mit $p \in F$, sowie

$$f(m_1, m_2, \dots, m_r) = (n_1, n_2, \dots, n_s)$$

gdw. der Funktionswert definiert ist.

einige berechenbare Funktionen

- $f : \mathbb{N} \rightarrow \mathbb{N}$ mit $f(x) := x + 1$ (Nachfolger)
- $f : \{1\}^* \rightarrow \{0, 1\}^*$ mit $f(w) := v$ mit $[w]_1 = [v]_2$ (unär \longrightarrow binär Konversion)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x + y$ (Summe)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x \cdot y$ (Produkt)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x^y$ (Exponentiation)

einige berechenbare Funktionen

- $f : \mathbb{N} \rightarrow \mathbb{N}$ mit $f(x) := x + 1$ (Nachfolger)
- $f : \{1\}^* \rightarrow \{0, 1\}^*$ mit $f(w) := v$ mit $[w]_1 = [v]_2$ (unär \longrightarrow binär Konversion)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x + y$ (Summe)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x \cdot y$ (Produkt)
- $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit $f(x, y) := x^y$ (Exponentiation)

Die Darstellung der Zahlen (binär, unär oder dezimal) ist für die Turingberechenbarkeit nicht von Bedeutung.

Die unäre Darstellung $|^n$ wäre schlecht gewählt, denn sie kann die Zahl $n=0$ nicht eindeutig kodieren!

Besser ist daher die übliche Codierung von n durch $|^{n+1}$.

Diagonalisierungstechnik

Existenz nicht berechenbarer Funktionen:

- DTM als endliche Zeichenkette darstellbar
⇒ Menge aller DTM abzählbar.
- Ist die Menge aller Funktionen $f : \mathbb{N} \rightarrow \{0, 1\}$ abzählbar? (Annahme: JA! ⇒ f_1, f_2, f_3, \dots)
- Definiere $g : \mathbb{N} \rightarrow \{0, 1\}$ durch

$$g(x) := \begin{cases} 0 & \text{falls } f_x(x) = 1 \\ 1 & \text{falls } f_x(x) = 0 \end{cases}$$

- Dann gilt: $\forall n \in \mathbb{N} : g \neq f_n$ **Widerspruch!!!**

Also muss es nicht berechenbare Funktion geben!

Existenz nicht abzählbarer Mengen

Sei $r \in I\!\!R$ mit $0 \leq r \leq 1$ beliebig. Definiere Funktion $f_r : I\!\!N \rightarrow I\!\!N$ durch:

$$f_r(n) := \begin{cases} 1 & \text{falls } n \text{ Präfix des Nachkommateils von } r \text{ ist,} \\ 0 & \text{sonst.} \end{cases}$$

Die Menge $\{f_r \mid r \in]0, 1[, r \text{ irrational}\}$ ist nicht abzählbar.

Die Menge aller Computerprogramme ist jedoch abzählbar, weil jedes Programm in endlichem Text notiert.

Also gibt es unendlich viele Funktionen, die nicht von Turingmaschinen berechnet werden können!

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 19

Turingmaschinen und kontextsensitive Grammatiken

Michael Köhler-Bußmeier

Kostenmaß bei Turingmaschinen

Definition 19.4:

Bei der TM wird die Berechnungskomplexität über ein einfaches Kostenmaß definiert:

- Ein *Schritt* (Konfigurationsübergang) kostet eine *Zeiteinheit*;
- eine *Bandzelle* kostet eine *Platzeinheit*.

Wir werden diese Definition besonders bei der Berechnung und Bestimmung der Komplexität von Algorithmen benötigen, welche durch Turingmaschinen ausgeführt werden!

linear beschränkter Automat

Definition 19.5:

- Ein **linear beschränkter Automat (LBA)** ist eine *NTM*, die bei beliebiger Eingabe w auf dem Arbeitsband höchstens $c \cdot |w|$ Felder bis zur Akzeptierung besucht.
- $c \in \mathbb{R}^+$ ist eine Konstante, die nicht von w abhängt.
- Arbeitet die TM bei gleicher Beschränkung ihres Arbeitsbandes deterministisch, so wird der linear beschränkte Automat mit *DLBA* abgekürzt.

Oft wird nur $c = 1$ betrachtet, aber jeder LBA der allgemeineren Definition lässt sich durch den spezielleren simulieren, siehe 2. Teil des Beweises!

Äquivalenz von LBA und CS

Satz 19.1:

$$\mathcal{LBA} = \mathcal{C}_S$$

Die Familie der von
LBA's bzw. DLBA's
akzeptierten Sprachen
wird mit \mathcal{LBA} bzw.
 \mathcal{DLBA} bezeichnet.

Äquivalenz von LBA und CS

Satz 19.1:

$$\mathcal{LBA} = \mathcal{Cs}$$

Beweis:

1. $\mathcal{Cs} \subseteq \mathcal{LBA}$:

Die Familie der von LBA 's bzw. $DLBA$'s akzeptierten Sprachen wird mit \mathcal{LBA} bzw. \mathcal{DLBA} bezeichnet.

Eine NTM kann auf einem zweiten Arbeitsband (Spur) die Ableitung eines Wortes mit den Produktionen der Grammatik durchführen. Wird die Eingabelänge überschritten: Abbruch.

Äquivalenz von LBA und CS

Satz 19.1:

$$\mathcal{LBA} = \mathcal{Cs}$$

Die Familie der von LBA's bzw. DLBA's akzeptierten Sprachen wird mit \mathcal{LBA} bzw. \mathcal{DLBA} bezeichnet.

Beweis:

1. $\mathcal{Cs} \subseteq \mathcal{LBA}$:

Eine NTM kann auf einem zweiten Arbeitsband (Spur) die Ableitung eines Wortes mit den Produktionen der Grammatik durchführen. Wird die Eingabelänge überschritten: Abbruch.

2. $\mathcal{LBA} \subseteq \mathcal{Cs}$:

Gegeben ist ein LBA, d.h. eine TM, die mit $c \cdot |w|$ Platz auskommt. Zunächst „normieren“ wir den LBA auf maximalen Platzbedarf $|w|$.

Äquivalenz von LBA und CS

Satz 19.1:

$$\mathcal{LBA} = \mathcal{Cs}$$

Beweis:

1. $\mathcal{Cs} \subseteq \mathcal{LBA}$:

Die Familie der von LBA's bzw. DLBA's akzeptierten Sprachen wird mit \mathcal{LBA} bzw. \mathcal{DLBA} bezeichnet.

Eine NTM kann auf einem zweiten Arbeitsband (Spur) die Ableitung eines Wortes mit den Produktionen der Grammatik durchführen. Wird die Eingabelänge überschritten: Abbruch.

2. $\mathcal{LBA} \subseteq \mathcal{Cs}$: dazu: äquivalente Typ-1 Grammatik konstruieren!

Gegeben ist ein LBA, d.h. eine TM, die mit $c \cdot |w|$ Platz auskommt. Zunächst „normieren“ wir den LBA auf maximalen Platzbedarf $|w|$.

Äquivalenz von LBA und CS

Satz 19.1:

$$\mathcal{LBA} = \mathcal{Cs}$$

Beweis:

1. $\mathcal{Cs} \subseteq \mathcal{LBA}$:

Die Familie der von LBA's bzw. DLBA's akzeptierten Sprachen wird mit \mathcal{LBA} bzw. \mathcal{DLBA} bezeichnet.

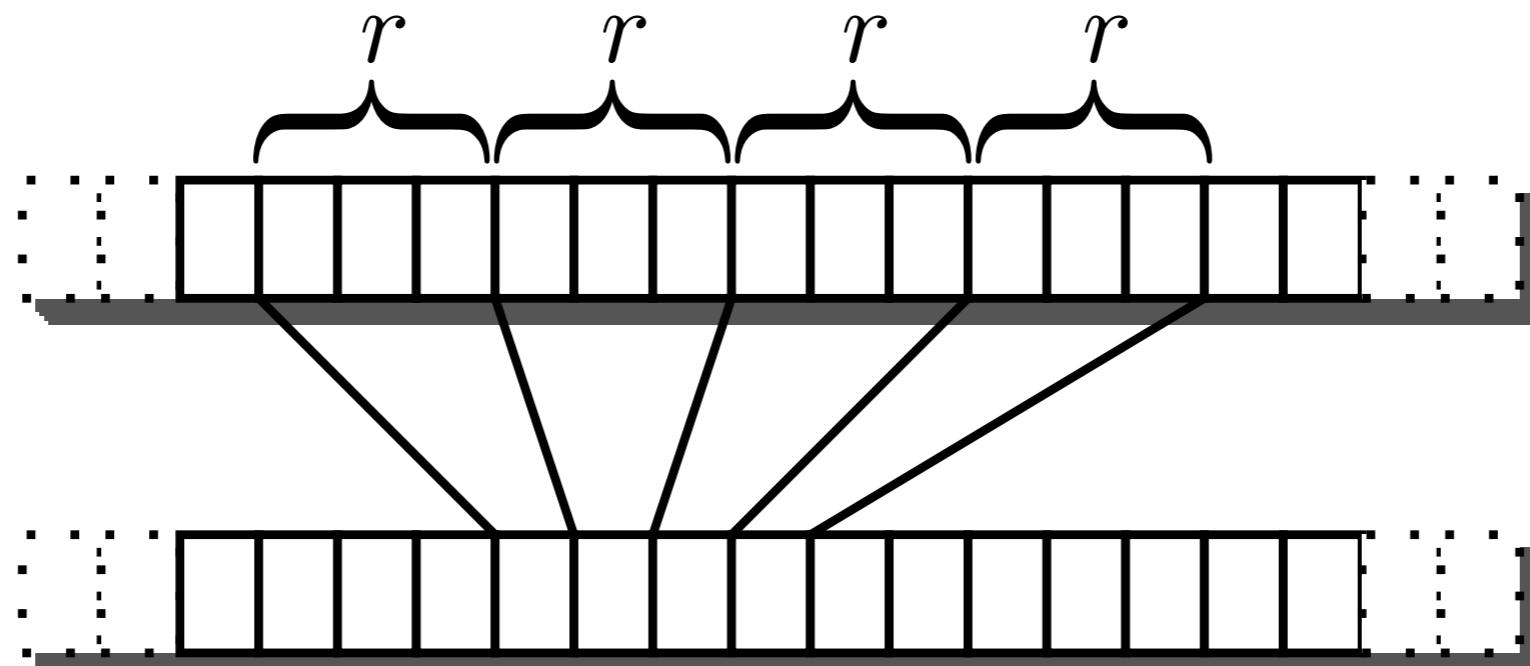
Eine NTM kann auf einem zweiten Arbeitsband (Spur) die Ableitung eines Wortes mit den Produktionen der Grammatik durchführen. Wird die Eingabelänge überschritten: Abbruch.

2. $\mathcal{LBA} \subseteq \mathcal{Cs}$: dazu: äquivalente Typ-1 Grammatik konstruieren!

Gegeben ist ein LBA, d.h. eine TM, die mit $c \cdot |w|$ Platz auskommt. Zunächst „normieren“ wir den LBA auf maximalen Platzbedarf $|w|$.

Bandkompression

- Für $0 \leq c \leq 1$ ist nichts zu tun.
- Ansonsten. Vergrößerung der Bandalphabets und Blockbildung.
 - Blocklänge $r \in \mathbb{N}$;
 - Bandbedarf nur noch $\frac{c}{r} \cdot |w|$;
 - Für $r := \min(n \in \mathbb{N} \mid n \geq c)$ sind das höchstens $|w|$ Felder.



Die Simulation der Rechnung mit Grammatik

Erklärung der verwendeten Nonterminale:

1. Spur	x	Eingabesymbol
2. Spur	y	Bandsymbol
3. Spur	$\$$	Anfang-/Ende-Marker
4. Spur	q	Zustand/Kopfposition

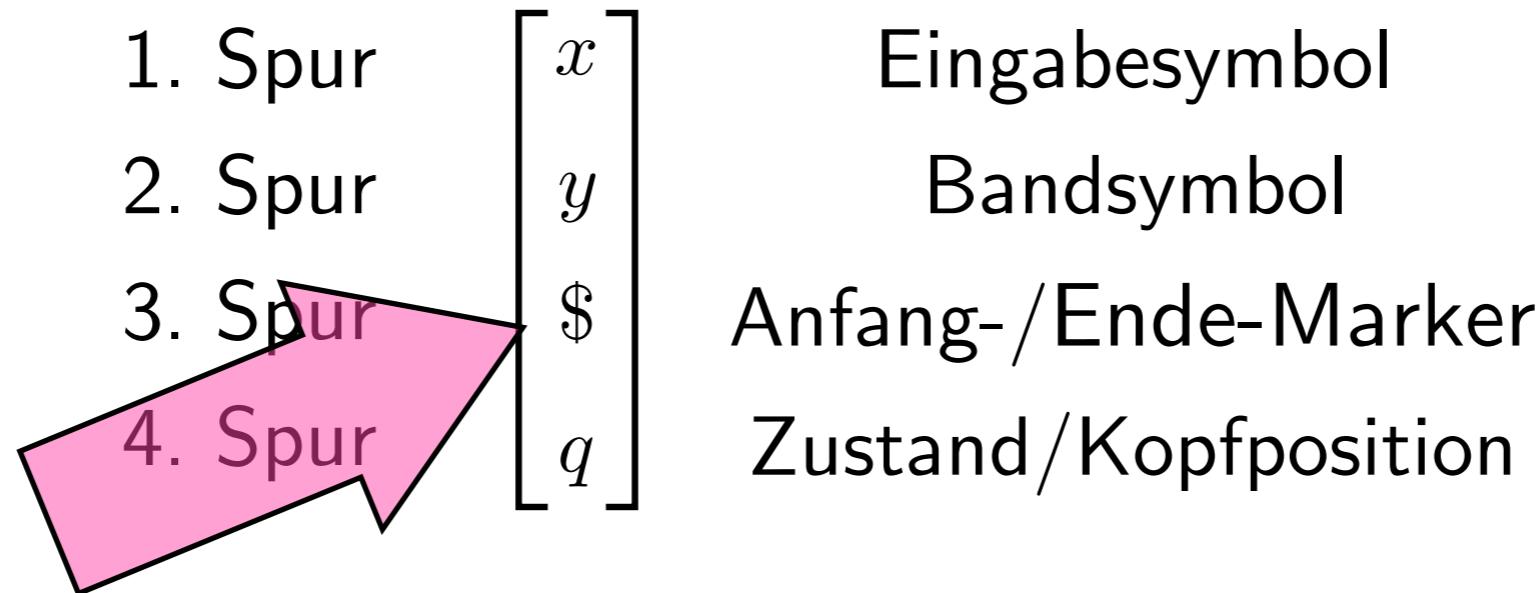
Die *Anfangs-Konfiguration* q_0w für ein beliebiges Wort $w \in \Sigma^*$ wird aus S mit folgenden Regeln erzeugt:

$$S \longrightarrow A \begin{bmatrix} x \\ x \\ \$ \\ \$ \end{bmatrix}, \quad A \longrightarrow A \begin{bmatrix} x \\ x \\ \$ \\ \$ \end{bmatrix} \quad \text{und} \quad A \longrightarrow \begin{bmatrix} x \\ x \\ \$ \\ q_0 \end{bmatrix},$$

für alle $x \in \Sigma$.

Die Simulation der Rechnung mit Grammatik

Erklärung der verwendeten Nonterminale:



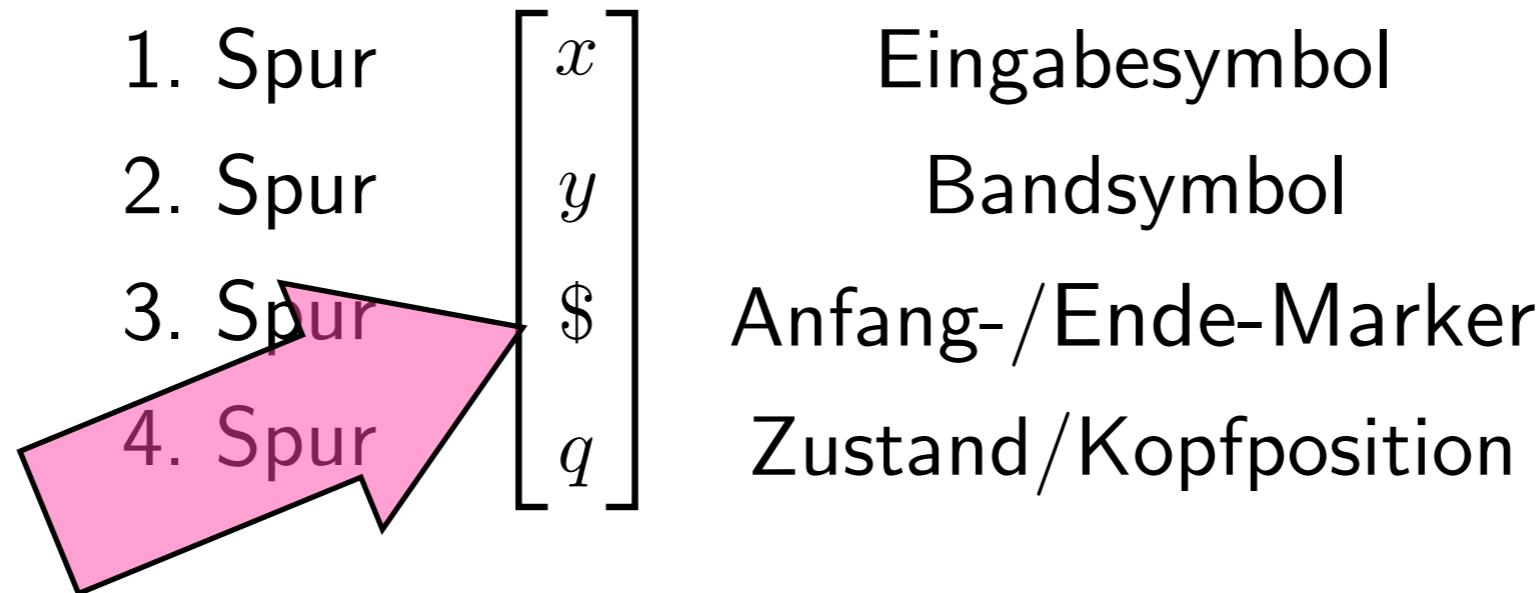
Die *Anfangs-Konfiguration* q_0w für ein beliebiges Wort $w \in \Sigma^*$ wird aus S mit folgenden Regeln erzeugt:

$$S \longrightarrow A \begin{bmatrix} x \\ x \\ \epsilon \\ \# \end{bmatrix}, \quad A \longrightarrow A \begin{bmatrix} x \\ x \\ \# \\ \# \end{bmatrix} \text{ und } A \longrightarrow \begin{bmatrix} x \\ x \\ \epsilon \\ q_0 \end{bmatrix},$$

für alle $x \in \Sigma$.

Die Simulation der Rechnung mit Grammatik

Erklärung der verwendeten Nonterminale:



Die *Anfangs-Konfiguration* q_0w für ein beliebiges Wort $w \in \Sigma^*$ wird aus S mit folgenden Regeln erzeugt:

$$S \longrightarrow A \begin{bmatrix} x \\ x \\ \epsilon \\ \# \end{bmatrix}, \quad A \longrightarrow A \begin{bmatrix} x \\ x \\ \# \\ \# \end{bmatrix} \text{ und } A \xrightarrow{\quad} \begin{bmatrix} x \\ x \\ \epsilon \\ q_0 \end{bmatrix},$$

für alle $x \in \Sigma$.

Simulation eines Linksschritts

Linksschritt: Für $x_1, x_2, x_3 \in \Sigma$, $y_1, y_2 \in \Gamma$ und für $(q, z, l) \in \delta(p, y)$, folgende Regeln in G :

$$\begin{array}{c} \left[\begin{matrix} x_1 \\ y_1 \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_2 \\ y \\ \# \\ p \end{matrix} \right] \longrightarrow \left[\begin{matrix} x_1 \\ y_1 \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_2 \\ z \\ \# \\ q \end{matrix} \right], \quad \left[\begin{matrix} x_1 \\ y_1 \\ \epsilon \\ \# \end{matrix} \right] \left[\begin{matrix} x_2 \\ y \\ \# \\ p \end{matrix} \right] \longrightarrow \left[\begin{matrix} x_1 \\ y_1 \\ \epsilon \\ q \end{matrix} \right] \left[\begin{matrix} x_2 \\ z \\ \# \\ \# \end{matrix} \right] \\ \left[\begin{matrix} x_1 \\ y_1 \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_2 \\ y \\ \epsilon \\ p \end{matrix} \right] \longrightarrow \left[\begin{matrix} x_1 \\ y_1 \\ \# \\ q \end{matrix} \right] \left[\begin{matrix} x_2 \\ z \\ \epsilon \\ \# \end{matrix} \right], \quad \left[\begin{matrix} x_1 \\ y_1 \\ \epsilon \\ \# \end{matrix} \right] \left[\begin{matrix} x_2 \\ y \\ \epsilon \\ p \end{matrix} \right] \longrightarrow \left[\begin{matrix} x_1 \\ y_1 \\ \epsilon \\ q \end{matrix} \right] \left[\begin{matrix} x_2 \\ z \\ \epsilon \\ \# \end{matrix} \right] \end{array}$$

Simulation eines Rechtsschritts

Rechtsschritt: Für $x_1, x_2, x_3 \in \Sigma$, $y_1, y_2 \in \Gamma$ und für $(q, z, r) \in \delta(p, y)$, folgende Regeln in G :

$$\begin{array}{ccc}
 \left[\begin{matrix} x_2 \\ y \\ \# \\ p \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ \# \end{matrix} \right] & \xrightarrow{\quad} & \left[\begin{matrix} x_2 \\ z \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ q \end{matrix} \right], \\
 \left[\begin{matrix} x_2 \\ y \\ \epsilon \\ p \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ \# \end{matrix} \right] & \xrightarrow{\quad} & \left[\begin{matrix} x_2 \\ z \\ \epsilon \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ q \end{matrix} \right], \\
 \left[\begin{matrix} x_2 \\ y_1 \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ \# \end{matrix} \right] & \xrightarrow{\quad} & \left[\begin{matrix} x_2 \\ y_1 \\ \# \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ q \end{matrix} \right], \\
 \left[\begin{matrix} x_2 \\ y_1 \\ \epsilon \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ \# \end{matrix} \right] & \xrightarrow{\quad} & \left[\begin{matrix} x_2 \\ y_1 \\ \epsilon \\ \# \end{matrix} \right] \left[\begin{matrix} x_3 \\ y_2 \\ \# \\ q \end{matrix} \right],
 \end{array}$$

Ende der Simulation

Bei Erreichen eines Endzustands des LBA müssen alle Nonterminale in entsprechende Terminalsymbol überführt werden.

Für alle $x, z \in \Sigma$, $y \in \Gamma$ und $\& \in \{\#, \epsilon, \downarrow\}$ folgende Regeln:

$$\boxed{x} \quad \xrightarrow{\quad \boxed{x} \quad} \text{falls } q \in F, \quad \begin{bmatrix} x \\ y \\ \& \\ \# \end{bmatrix} z \longrightarrow xz, \quad z \begin{bmatrix} x \\ y \\ \& \\ \# \end{bmatrix} \longrightarrow zx$$

Problem/Fehler dieser Simulation

... leider steckt ein Fehler in diesem Satz von Regeln!

- Es können nur Wörter w mit $|w| \geq 2$ generiert werden!
- Warum? $\$$ und ϵ sind keine echten Begrenzer!
- Abhilfe: alle Wörter bis zur Länge 2 daraufhin untersuchen, ob sie vom LBA akzeptiert werden!

Zusätzliche Regeln $S \longrightarrow x$ für diejenigen $x \in \{\epsilon\} \cup \Sigma$, die dazugehören müssen! Warum ist $x \in L(LBA)$ entscheidbar?

- Es gibt nur endlich viele Rechnungen *ohne Schleifen* für ein Wort!

Problem/Fehler dieser Simulation

... leider steckt ein Fehler in diesem Satz von Regeln!

- Es können nur Wörter w mit $|w| \geq 2$ generiert werden!
- Warum? $\$$ und ϵ sind keine echten Begrenzer!
- Abhilfe: alle Wörter bis zur Länge 2 daraufhin untersuchen, ob sie vom LBA akzeptiert werden!

Zusätzliche Regeln $S \longrightarrow x$ für diejenigen $x \in \{\epsilon\} \cup \Sigma$, die dazugehören müssen! Warum ist $x \in L(LBA)$ entscheidbar?

- Es gibt nur endlich viele Rechnungen *ohne Schleifen* für ein Wort!

Vorteile der LBA-Darstellung

Mit den Produktionen kann ein terminales Wort erzeugt werden, gdw. es eine Erfolgsrechnung für w im LBA mit $|w|$ Speicherplatz gibt.

- Mit Hilfe der Charakterisierungen von Sprachfamilien durch Automaten, lassen sich häufig Abschlusseigenschaften leichter beweisen als mit Grammatiken.
- Die Familie $\mathcal{Cs} = \mathcal{LBA}$ ist gegen Durchschnittsbildung abgeschlossen.
- Beweisidee: Spurenbildung und Kombination der beiden LBA 's

Typ-0 Sprachen und Turingmaschinen

Satz 19.2:

$$\mathcal{R}e = \mathcal{L}_0$$

Beweis:

1. $\mathcal{L}_0 \subseteq \mathcal{R}e$:

Eine NTM kann auf einem zweien Arbeitsband (Spur) die Ableitung eines Wortes mit den Produktionen der Grammatik durchführen.

2. $\mathcal{R}e \subseteq \mathcal{L}_0$:

Die Simulation eines *LBA* durch eine kontextsensitive Grammatik kann abgewandelt werden für beliebige Einband *NTM*'s und erfordert Typ-0 Grammatik.

charakteristische Funktion

Diese Funktion gibt Auskunft über das Vorhandensein von Elementen in einer Menge:

Definition 19.6:

Sei $M \subseteq C$, dann ist die **charakteristische Funktion** von M die auf ganz C definierte Funktion $\chi_M : C \rightarrow \{0, 1\}$ mit

$$\chi_M(x) := \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{sonst.} \end{cases}$$

Entscheidbarkeit

Die **Entscheidbarkeit** von Mengen haben wir mit Hilfe stets terminierende TM definiert.
Das folgende ist oft die Basisdefinition:

Definition 19.8:

Eine Menge $L \subseteq \Sigma^*$ heißt (relativ zu Σ^*) **entscheidbar** oder **rekursiv** gdw. ihre charakteristische Funktion $\chi_L : \Sigma^* \rightarrow \{0, 1\}$ berechenbar ist.

Die *Klasse aller entscheidbaren Mengen* wird mit $\mathcal{R}ec$ (*recursive sets*) bezeichnet.

Daraus folgt, dass jede endliche Menge entscheidbar ist:

Die endliche Kontrolle der DTM ist im wesentlichen ein DFA.
Die DTM prüft bei jeder Eingabe, ob diese in der endlichen Menge vorkommt. Wenn ja, so wird 1 ausgegeben, sonst 0.

Rekursivität von CS

Satz 19.3:

$$\mathcal{C}_S \subseteq \mathcal{R}_{\text{ec}}$$

Der Beweis ist am leichtesten mit Grammatiken zu führen:

- Für jedes $L \in \mathcal{C}_S$ und jedes Wort $w \in \Sigma^*$ kann entschieden werden, ob $w \in L$ gilt.
 - Es gibt endlich viele Wörter v mit $|v| \leq |w|$.
 - Es gibt eine kontextsensitive Grammatik $G = (\Sigma, N, P, S)$ mit $L(G) = L$.
 - Keine Regel verkürzt die Satzform!
 - Ohne Satzformwiederholungen gibt es nur endlich viele Ableitungsschritte, bis die Satzform zu lang ist!!

Die Inklusion ist echt:

Satz 19.4:

$$Cs \subsetneq Rec$$

Diagonalbeweis:

- Die Menge der kontextsensitiven Grammatiken ist abzählbar: G_1, G_2, \dots
- Sei $f : \{0, 1\}^* \rightarrow \mathbb{N}$ eine berechenbare Bijektion, mit: $f(w) = i$ gdw. w ist i -tes Wort in der lexikalischen Ordnung auf $\{0, 1\}^*$.
- $L_e := \{w \mid w \notin L(G_{f(w)})\}$ ist entscheidbar!
- ABER: L_e ist nicht kontextsensitiv!

Die Inklusion ist echt:

Satz 19.4:

$$Cs \not\subseteq Rec$$

Diagonalbeweis:

- Die Menge der kontextsensitiven Grammatiken ist abzählbar: G_1, G_2, \dots
- Sei $f : \{0, 1\}^* \rightarrow \mathbb{N}$ eine berechenbare Bijektion, mit: $f(w) = i$ gdw. w ist i -tes Wort in der lexikalischen Ordnung auf $\{0, 1\}^*$.
- $L_e := \{w \mid w \notin L(G_{f(w)})\}$ ist entscheidbar! *„denn das Wortproblem ist für Cs entscheidbar.“*
- ABER: L_e ist nicht kontextsensitiv!

Beweisfortsetzung

Angenommen L_e wäre doch kontextsensitiv.

Annahme: $\exists n$ mit $L(G_n) = L_e$.

Für das n -te Wort u der lexikalischen Aufzählung von $\{0, 1\}^*$ mit $f(u) = n$ ergibt sich ein Widerspruch für beide möglichen Fälle $u \in L_e$ und $u \notin L_e$:

$$u \in L_e \xrightarrow{\text{(Def. von } L_e)} u \notin L(G_n) \xrightarrow{\text{(Annahme)}} u \notin L_e.$$

Andererseits ergibt sich auch:

$$u \notin L_e \xrightarrow{\text{(Def. von } L_e)} u \in L(G_n) \xrightarrow{\text{(Annahme)}} u \in L_e.$$

qed.

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

\subseteq Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

$\subseteq \checkmark$ Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

$\subseteq \checkmark$ Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

$\subseteq \checkmark$ Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 20
Entscheidbarkeit, Gödelisierung,
Halteproblem

Michael Köhler-Bußmeier

Eine Hierarchie von Sprachfamilien

Dies ist die Chomsky-Hierarchie.

	Familie	$\mathcal{R}eg$	der regulären Mengen
\subsetneq	Familie	$\mathcal{C}f$	der kontextfreien Mengen
\subsetneq	Familie	$\mathcal{C}s$	der kontextsensitiven Mengen
\subsetneq	Familie	$\mathcal{R}ec$	der entscheidbaren Mengen
\subsetneq	Familie	$\mathcal{R}e$	der aufzählbaren Mengen
\subsetneq	Familie		der abzählbaren Mengen
\neq	Familie		der überabzählbaren Mengen

Aus der Definition folgen sofort die Inklusionen.

Wir zeigen mit Hilfe der **Diagonalsprache** und der **Haltesprache**, dass die Inklusionen auch **echt** sind.

Eine Hierarchie von Sprachfamilien

Dies ist die Chomsky-Hierarchie.

- Familie $\mathcal{R}eg$ der **regulären Mengen**
- \subsetneq Familie $\mathcal{C}f$ der **kontextfreien Mengen**
- \subsetneq Familie $\mathcal{C}s$ der **kontextsensitiven Mengen**
- \subsetneq Familie $\mathcal{R}ec$ der **entscheidbaren Mengen**
- \subsetneq Familie $\mathcal{R}e$ der **aufzählbaren Mengen**
- \subsetneq Familie der **abzählbaren Mengen**
- \neq Familie der **überabzählbaren Mengen**

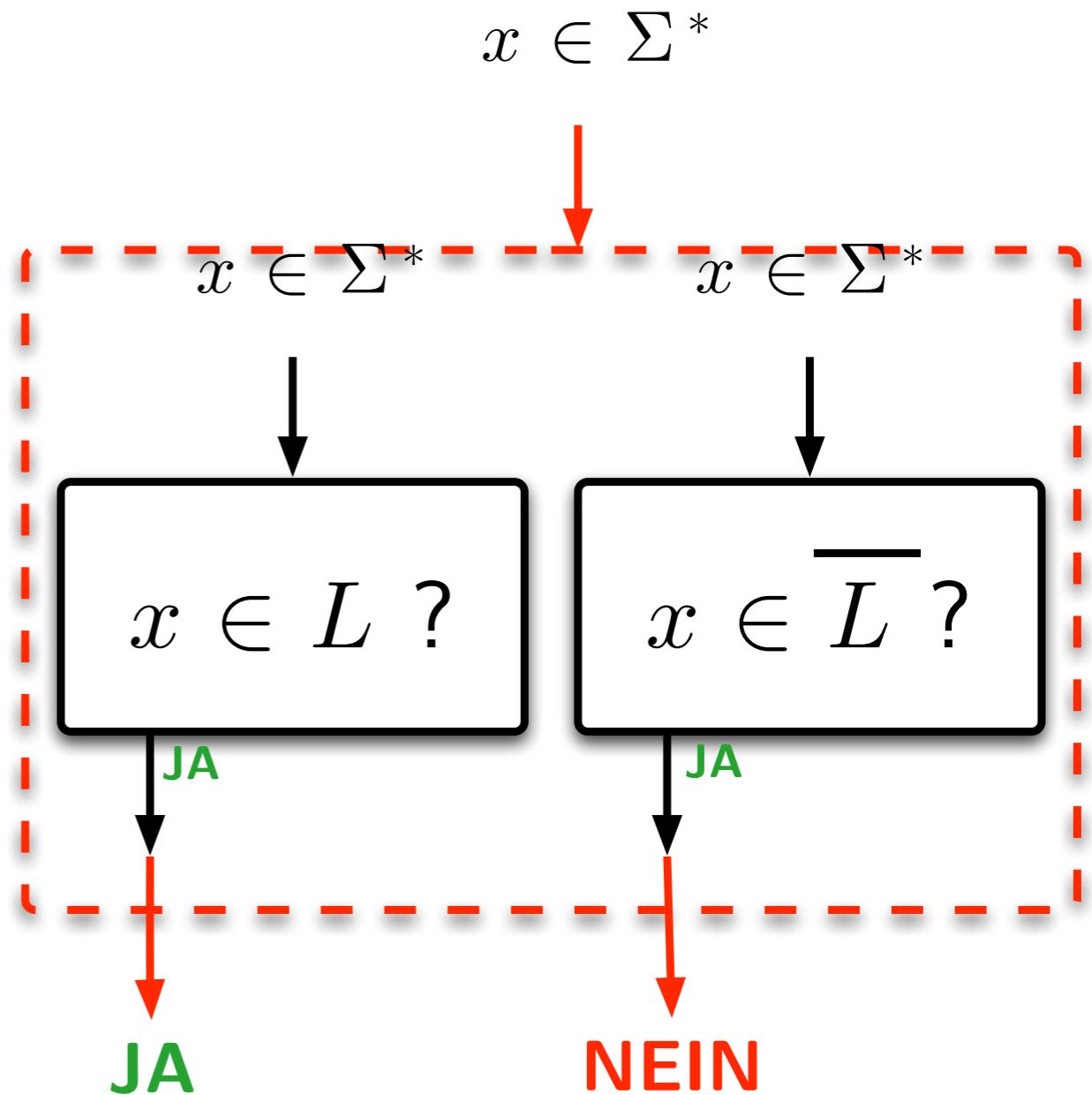
Aus der Definition folgen sofort die Inklusionen.

Wir zeigen mit Hilfe der **Diagonalsprache** und der **Haltesprache**, dass die Inklusionen auch **echt** sind.

ein wichtiger Zusammenhang:

Satz 20.1:

Eine Menge L ist genau dann rekursiv (entscheidbar), wenn L selbst, und auch ihr Komplement \overline{L} rekursiv aufzählbar sind.



⇒ Entscheidbare Mengen sind stets aufzählbar! (Klar?)

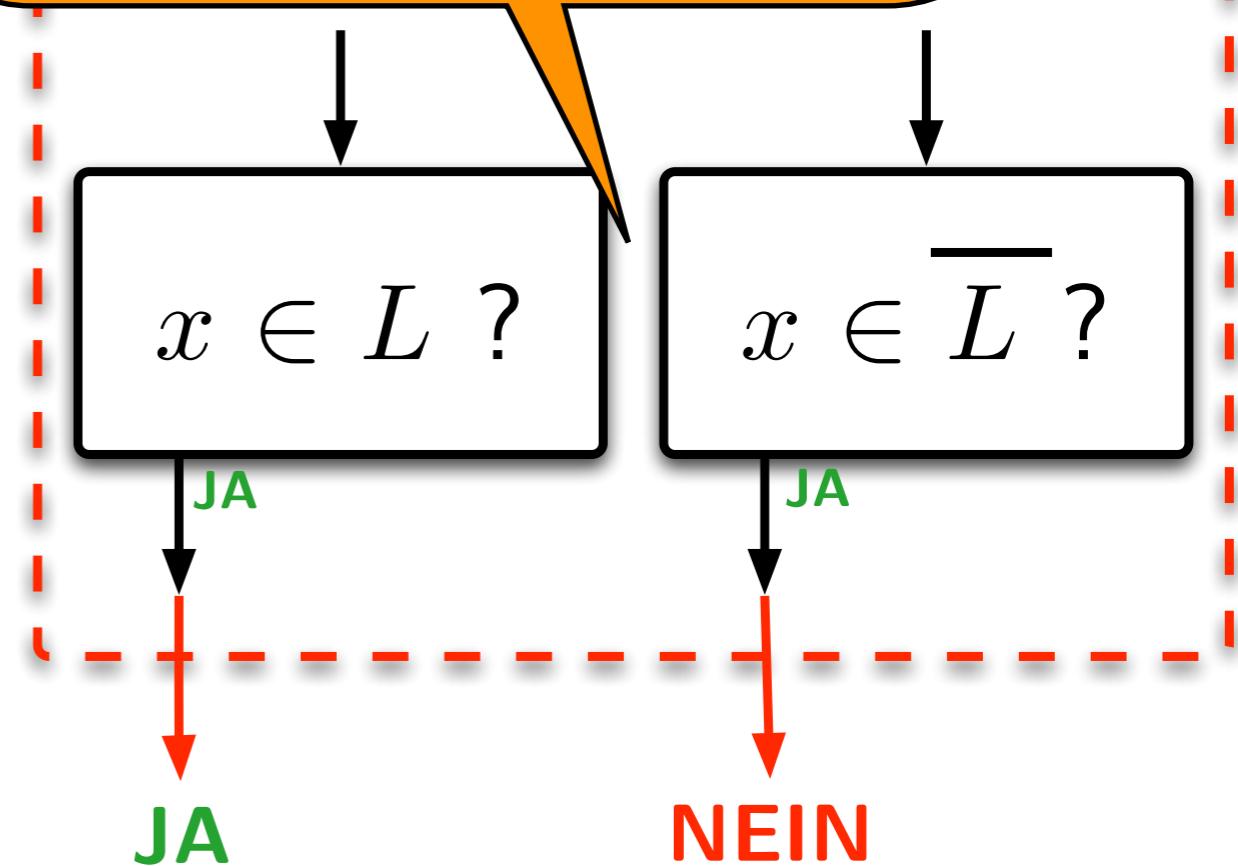
⇐ Dies folgt aus Erklärung zu obigem Bild!

ein wichtiger Zusammenhang:

Satz 20.1:

Eine Menge L ist genau dann rekursiv (entscheidbar), wenn L selbst, und auch ihr Komplement \overline{L} rekursiv aufzählbar sind.

Wir simulieren die beiden TM abwechselnd – jede jeweils einen Schritt.



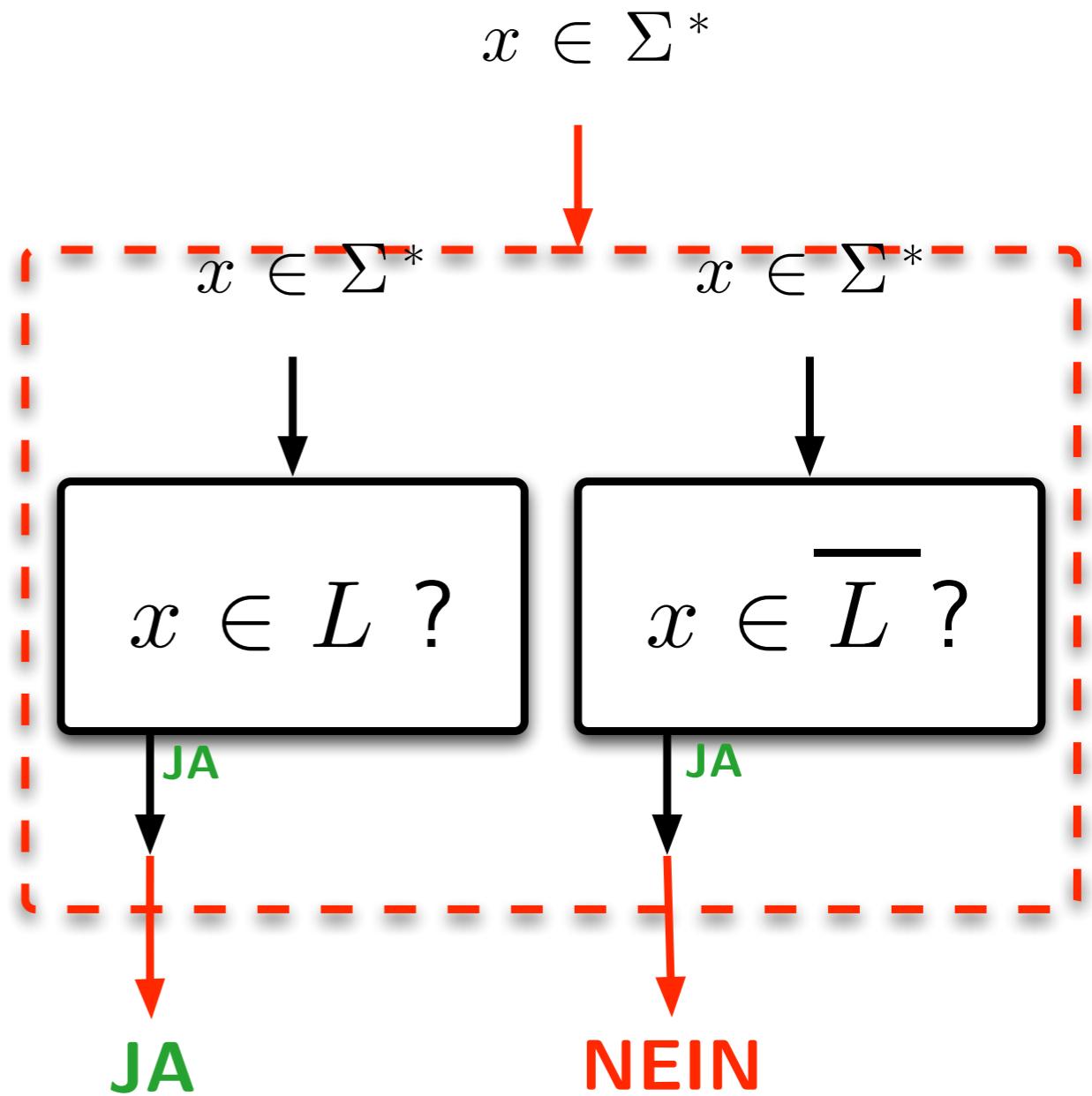
⇒ Entscheidbare Mengen sind stets aufzählbar! (Klar?)

⇐ Dies folgt aus Erklärung zu obigem Bild!

ein wichtiger Zusammenhang:

Satz 20.1:

Eine Menge L ist genau dann rekursiv (entscheidbar), wenn L selbst, und auch ihr Komplement \overline{L} rekursiv aufzählbar sind.



⇒ Entscheidbare Mengen sind stets aufzählbar! (Klar?)

⇐ Dies folgt aus Erklärung zu obigem Bild!

ein wichtiger Zusammenhang:

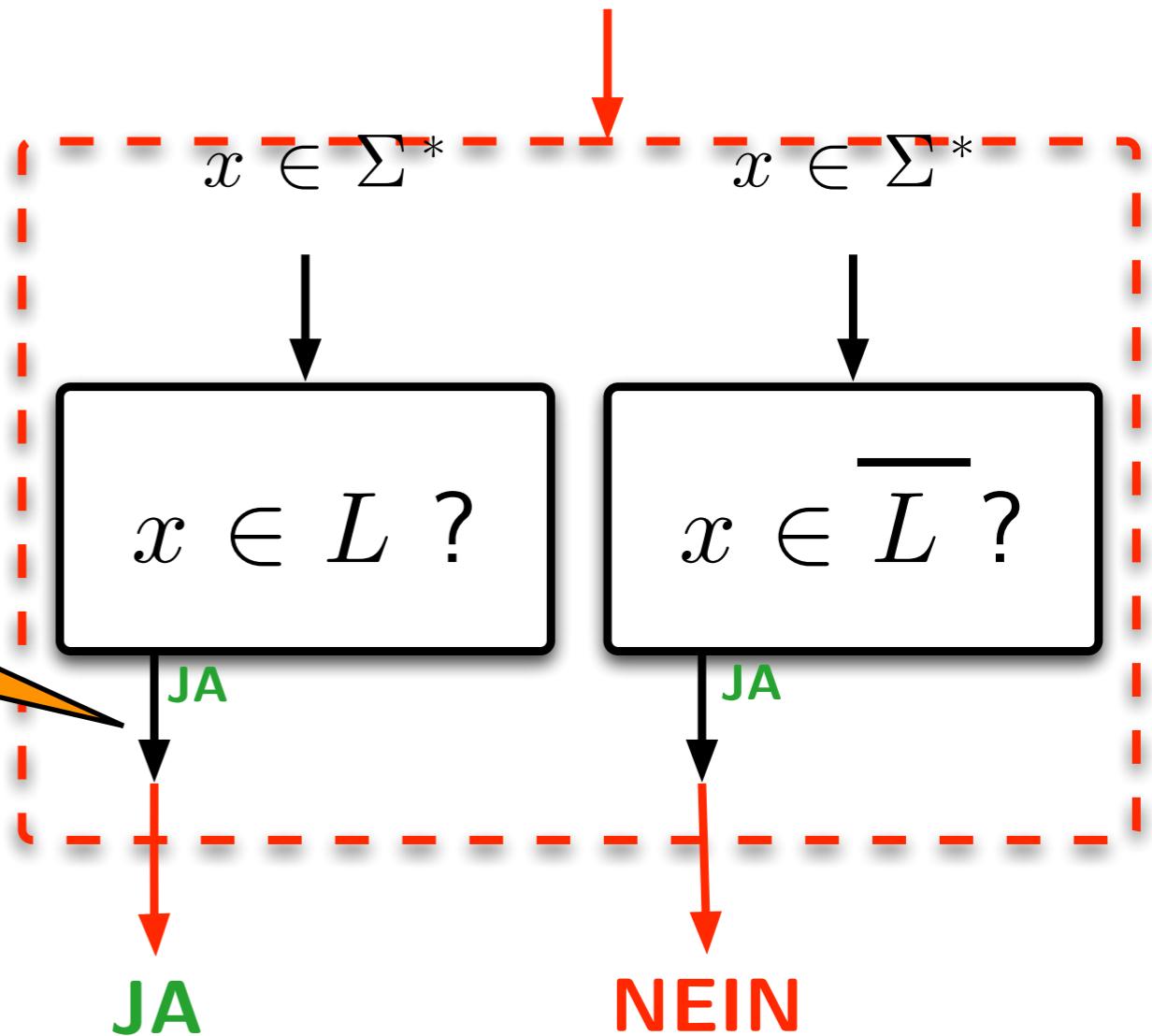
$x \in \Sigma^*$

Satz 20.1:

Eine Menge L ist

Nur eine der beiden TM
kann mit "Ja" antworten.

ihm Komplement \overline{L}
rekursiv aufzählbar
sind.



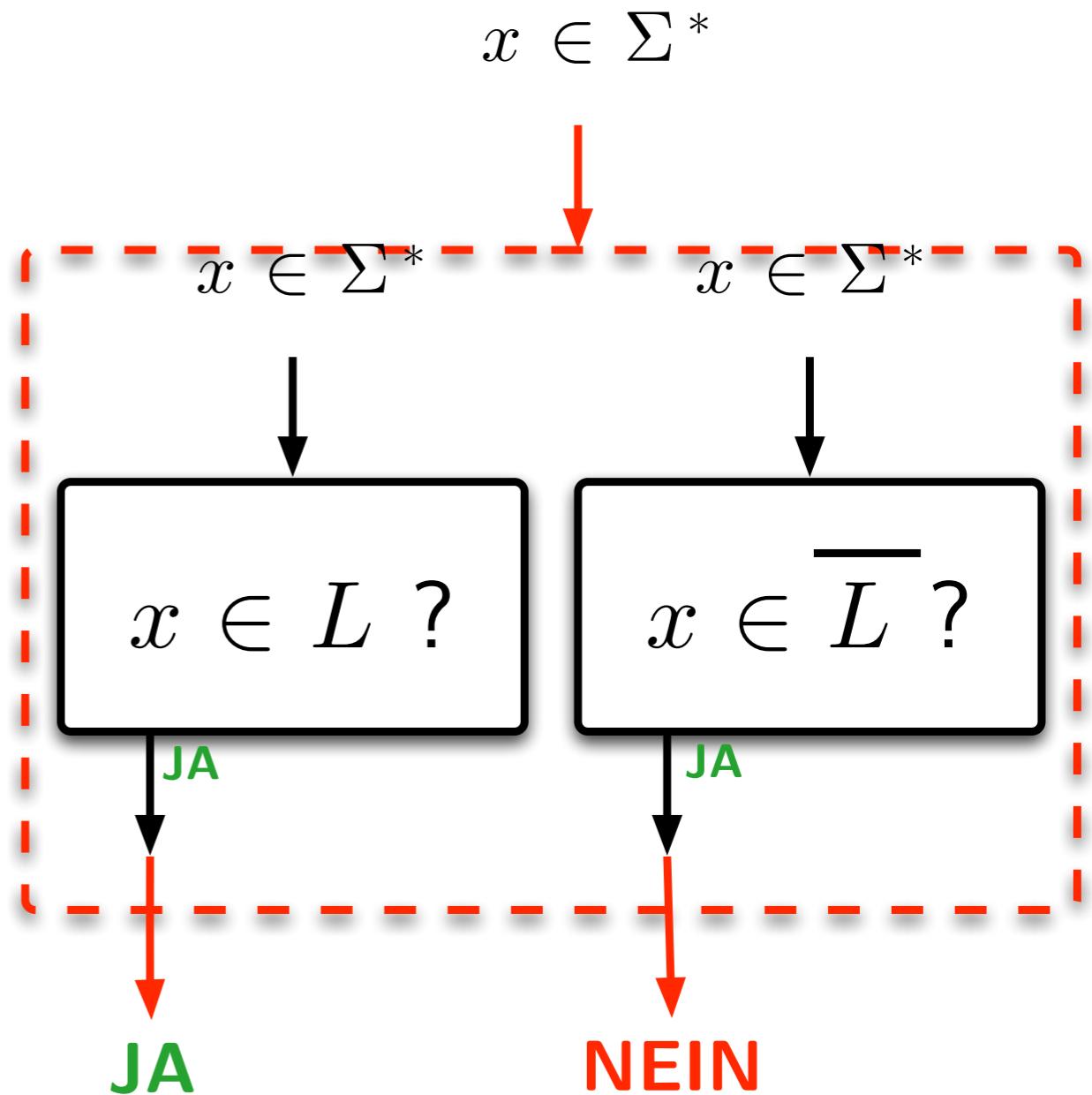
⇒ Entscheidbare Mengen sind stets aufzählbar! (Klar?)

⇐ Dies folgt aus Erklärung zu obigem Bild!

ein wichtiger Zusammenhang:

Satz 20.1:

Eine Menge L ist genau dann rekursiv (entscheidbar), wenn L selbst, und auch ihr Komplement \overline{L} rekursiv aufzählbar sind.



⇒ Entscheidbare Mengen sind stets aufzählbar! (Klar?)

⇐ Dies folgt aus Erklärung zu obigem Bild!

Folgerung

Korollar: Wenn L eine aufzählbare, aber nicht entscheidbare Menge ist, dann ist ihr Komplement nicht mal mehr aufzählbar.

Beweis:

Angenommen das Komplement von L wäre doch aufzählbar, dann wäre nach Satz 20.1 auch L selbst entscheidbar.

Widerspruch zur Annahme.

Also ist das Komplement von L nicht aufzählbar.

Satz 20.1:

Eine Menge L ist genau dann rekursiv (entscheidbar) wenn L selbst, und auch ihr Komplement \bar{L} rekursiv aufzählbar sind.

nichtaufzählbare Sprachen

1. **Frage:** Gibt es formale Sprachen, die von **keiner DTM** akzeptiert werden können?
 2. Wir zeigen, dass es eine Sprache $L \subseteq \{0, 1\}^*$ gibt, die von keiner DTM akzeptiert werden kann! die Sprache L_d
 3. Wir benötigen: Codierung von Automaten als Wörter über einem endlichen Alphabet, d.h. eine *injektive* Funktion
- code : Menge aller DTM → {0, 1}**
4. wir können die lexikalische Ordnung wählen, aber es gibt seit Gödel's Beweisen auch andere!

nichtaufzählbare Sprachen

1. **Frage:** Gibt es formale Sprachen, die von **keiner DTM** akzeptiert werden können?
2. Wir zeigen, dass es eine Sprache $L \subseteq \{0, 1\}^*$ gibt, die von keiner DTM akzeptiert werden kann! **die Sprache L_d**

3. Wir benötigen: Codierung von Automaten als Wörter über einem endlichen Alphabet, d.h. eine *injektive* Funktion

*code : Menge allerDTM → {0, 1}**

4. wir können die lexikalische Ordnung wählen, aber es gibt seit Gödel's Beweisen auch andere!

nichtaufzählbare Sprachen

1. **Frage:** Gibt es formale Sprachen, die von **keiner DTM** akzeptiert werden können?
2. Wir zeigen, dass es eine Sprache $L \subseteq \{0, 1\}^*$ gibt, die von **keiner DTM** akzeptiert werden kann! **die Sprache L_d**

3. Wir benötigen: Codierung von Automaten als Wörter über einem endlichen Alphabet, d.h. eine *injektive* Funktion

*code : Menge allerDTM → {0, 1}**

4. wir können die lexikalische Ordnung wählen, aber es gibt seit Gödel's Beweisen auch andere!

nichtaufzählbare Sprachen

1. **Frage:** Gibt es formale Sprachen, die von **keiner DTM** akzeptiert werden können?
2. Wir zeigen, dass es eine Sprache $L \subseteq \{0, 1\}^*$ gibt, die von **keiner DTM** akzeptiert werden kann! **die Sprache L_d**
3. Wir benötigen: **Codierung von Automaten als Wörter über einem endlichen Alphabet**, d.h. eine *injektive* Funktion

$code : \text{Menge aller } DTM \rightarrow \{0, 1\}^*$

4. wir können die lexikalische Ordnung wählen, aber es gibt seit Gödel's Beweisen auch andere!

nichtaufzählbare Sprachen

1. **Frage:** Gibt es formale Sprachen, die von **keiner DTM** akzeptiert werden können?
2. Wir zeigen, dass es eine Sprache $L \subseteq \{0, 1\}^*$ gibt, die von **keiner DTM** akzeptiert werden kann! **die Sprache L_d**
3. Wir benötigen: **Codierung von Automaten als Wörter über einem endlichen Alphabet**, d.h. eine *injektive* Funktion

$code : \text{Menge aller } DTM \rightarrow \{0, 1\}^*$

4. ... wir können die lexikalische Ordnung wählen, aber es gibt seit Gödel's Beweisen auch andere!

Exkurs: Was sind Gödelnummern?

Definition 20.1:

Eine Funktion $\nu_\Sigma : \Sigma^* \rightarrow \mathbb{N}$ (Σ ein endliches Alphabet) heißt **Gödelisierung**, wenn gilt:

1. $\forall u, v \in \Sigma^* : u \neq v$ impliziert $\nu_\Sigma(u) \neq \nu_\Sigma(v)$, ν_Σ ist also injektiv.
2. Aus jedem Argument $w \in \Sigma^*$ kann $\nu_\Sigma(w)$ in endlich vielen Schritten effektiv bestimmt werden.
3. Für jedes $n \in \mathbb{N}$ kann in endlich vielen Schritten effektiv bestimmt werden, ob es ein $w \in \Sigma^*$ gibt, für das $\nu_\Sigma(w) = n$ gilt.
4. Wenn es ein $w \in \Sigma^*$ mit $\nu_\Sigma(w) = n$ gibt, dann kann w in endlich vielen Schritten effektiv aus n konstruiert werden.

Die Paarfunktion ist Gödelisierung;

zur Erinnerung:

Definition 18.14:

paar : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ sei definiert durch

$$\text{paar}(i, j) := i + \frac{(i+j)(i+j+1)}{2} = i + \binom{i+j+1}{2}$$

...aber wie berechnen wir zu einer Zahl k
das zugehörige Tupel (i,j)?

Formel für $\text{paar}^{-1}(k)$

Dieses hier ohne Beweis...
(Einsetzen in die Berechnungsformel reicht aus):

Satz 20.2:

Für

$$\pi_1(k) := k - \frac{1}{2} \cdot \left\lfloor \sqrt{2 \cdot k + \frac{1}{4}} - \frac{1}{2} \right\rfloor \cdot \left\lfloor \sqrt{2 \cdot k + \frac{1}{4}} + \frac{1}{2} \right\rfloor,$$

$$\pi_2(k) := \left\lfloor \sqrt{2k + \frac{1}{4}} - \frac{1}{2} \right\rfloor - \pi_1(k)$$

gilt:

$$k = \text{paar}(\pi_1(k), \pi_2(k)).$$

Gödelisierung i. A. nicht bijektiv!

Es ist bei Gödelisierungen keine Bijektion zwischen Σ^* und \mathbb{N} gefordert!

Es gibt verschiedene Möglichkeiten spezielle Gödelisierungen anzugeben, von denen wir z.B. die Cantorsche Paarfunktion kennengelernt haben.

Die Paarfunktionen paar, und Ihre Erweiterung auf beliebige Zahlentupel sind Gödelisierungen nur für Zahlentupel über \mathbb{N} mit fester Dimension.

Wenn wir allerdings endliche Zahlenfolgen oder Zeichenketten variabler Länge kodieren möchten, so taugen diese Abbildungen herzlich wenig.

Von Gödel stammt die bijektive Abbildung von Zahlenfolgen/Wörtern auf Zahlen über Primzahlprodukte:

$$\nu_{\mathbb{N}-\text{seq}}(\alpha_1, \alpha_2, \dots, \alpha_m) := \prod_{i=1}^m p_i^{\alpha_i + [i = m]} - 1$$

Gödelisierung i. A. nicht bijektiv!

Es ist bei Gödelisierungen keine Bijektion zwischen Σ^* und \mathbb{N} gefordert!

Es gibt verschiedene Möglichkeiten spezielle Gödelisierungen anzugeben, von denen wir z.B. die Cantorsche Paarfunktion kennengelernt haben.

Die Paarfunktionen paar, und Ihre Erweiterung auf beliebige Zahlentupel sind Gödelisierungen nur für Zahlentupel über \mathbb{N} mit fester Dimension.

Wenn wir allerdings endliche Zahlenfolgen oder Zeichenketten variabler Länge kodieren möchten, so taugen diese Abbildungen herzlich wenig.

Von Gödel stammt die bijektive Abbildung von Zahlenfolgen/Wörtern auf Zahlen über Primzahlprodukte:

$$\nu_{\mathbb{N}-\text{seq}}(\alpha_1, \alpha_2, \dots, \alpha_m) := \prod_{i=1}^m p_i^{\alpha_i + [i = m]} - 1$$

Gödelisierung i. A. nicht bijektiv!

Es ist bei Gödelisierungen keine Bijektion zwischen Σ^* und \mathbb{N} gefordert!

Es gibt verschiedene Möglichkeiten spezielle Gödelisierungen anzugeben, von denen wir z.B. die Cantorsche Paarfunktion kennengelernt haben.

Die Paarfunktionen paar, und Ihre Erweiterung auf beliebige Zahlentupel sind Gödelisierungen nur für Zahlentupel über \mathbb{N} mit fester Dimension.

Wenn wir allerdings endliche Zahlenfolgen oder Zeichenketten variabler Länge kodieren möchten, so taugen diese Abbildungen herzlich wenig.

Von Gödel stammt die bijektive Abbildung von Zahlenfolgen/Wörtern auf Zahlen über Primzahlprodukte:

$$\nu_{\mathbb{N}-\text{seq}}(\alpha_1, \alpha_2, \dots, \alpha_m) := \prod_{i=1}^m p_i^{\alpha_i + [i = m]} - 1$$

Gödelisierung i. A. nicht bijektiv!

Es ist bei Gödelisierungen keine Bijektion zwischen Σ^* und \mathbb{N} gefordert!

Es gibt verschiedene Möglichkeiten spezielle Gödelisierungen anzugeben, von denen wir z.B. die Cantorsche Paarfunktion kennengelernt haben.

Die Paarfunktionen paar, und Ihre Erweiterung auf beliebige Zahlentupel sind Gödelisierungen nur für Zahlentupel über \mathbb{N} mit fester Dimension.

Wenn wir allerdings endliche Zahlenfolgen oder Zeichenketten variabler Länge kodieren möchten, so taugen diese Abbildungen herzlich wenig.

Von Gödel stammt die bijektive Abbildung von Zahlenfolgen/Wörtern auf Zahlen über Primzahlprodukte:

$$\nu_{\mathbb{N}-seq}(\alpha_1, \alpha_2, \dots, \alpha_m) := \prod_{i=1}^m p_i^{\alpha_i + [i = m]} - 1$$

allgemeine Notation für Gödelnummern

Definition 20.2 (Vereinbarung):

Gödelisierungen werden von nun an immer als bijektive Abbildungen angesehen und – sofern das Alphabet G (oder ein anderes) festgelegt ist – durch $\langle \cdot \rangle : G^* \rightarrow \mathbb{N}$ notiert.

Beispiel:

Mit $\langle abbab \rangle$ wird diejenige natürliche Zahl dargestellt, die gemäß der beliebig gewählten, aber eindeutig festgesetzten Gödelisierung, die Gödelnummer von $abbab$ ist.

Wir können **immer** auch die lexikalische Anordnung der Wörter benutzen und benutzen dann i als Gödelnummer des i -ten Wortes in dieser Abzählung!

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) Aufzählung der Wörter $w_i \in G^*$ und $A_j \in G^*$.

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, \}, \{, , \}\}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) Aufzählung der Wörter $w_i \in G^*$ und $A_j \in G^*$.

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, \}, \{, , \}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) Aufzählung der Wörter $w_i \in G^*$ und $A_j \in G^*$.

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, }, \{, , \}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) Aufzählung der Wörter $w_i \in G^*$ und $A_j \in G^*$.

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, \}, \{, , \}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) Aufzählung der Wörter $w_i \in G^*$ und $A_j \in G^*$.

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, \}, \{, , \}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Die Sprache L_d

Satz 20.3:

Sei G ein Alphabet zur Kodierung von Turingmaschinen bzw. Wörtern, sowie

w_i das i -te Wort und

A_j die j -te DTM in der (lexikalischen) $w_i \in G^*$ und $A_j \in G^*$.

...aber abzählbar...

Dann gilt:

Die Menge $L_d := \{\langle w_i \rangle \mid w_i \notin L(A_i)\} \subseteq \mathbb{N}$ ist nicht aufzählbar!

Das Tupel $T := (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, welches eine *DTM* spezifiziert, kann als Zeichenkette über dem Alphabet

$G := Q \uplus \Sigma \uplus \Gamma \uplus \{\), (, \}, \{, , \}$ aufgefasst werden, sofern die Übergangsfunktion $\delta \subseteq Q \times (\Gamma \cup \Sigma) \times (\Gamma \cup \Sigma) \times \{l, r, -\} \times Q$ als Relation notiert wird.

Beweis durch Diagonalisierung:

Beweis:

G^* ist in der lexikalischen Ordnung aufzählbar. (klar)

Die Teilmenge von G^* aller derjenigen Wörter, die Kodierung einer Turingmaschine sind, ist auch aufzählbar.

(Es gibt TM, die für $w \in G^*$ entscheiden kann, ob dieses die zulässige Kodierung einer TM ist.)

Betrachtet man die unendliche Matrix mit den Spalten w_1, w_2, w_3, \dots , den Zeilen A_1, A_2, A_3, \dots und den Einträgen: 1, falls $w_i \in L(A_i)$ bzw. 0, falls $w_i \notin L(A_i)$, so entspricht L_d gerade der Diagonalen, von der nur die Einträge 0 Berücksichtigung fanden.

Beweis durch Diagonalisierung:

Beweis:

G^* ist in der lexikalischen Ordnung aufzählbar. (klar)

Die Teilmenge von G^* aller derjenigen Wörter, die Kodierung einer Turingmaschine sind, ist auch aufzählbar.

(Es gibt TM, die für $w \in G^*$ entscheiden kann, ob dieses die zulässige Kodierung einer TM ist.)

Betrachtet man die unendliche Matrix mit den Spalten w_1, w_2, w_3, \dots , den Zeilen A_1, A_2, A_3, \dots und den Einträgen: 1, falls $w_i \in L(A_i)$ bzw. 0, falls $w_i \notin L(A_i)$, so entspricht L_d gerade der Diagonalen, von der nur die Einträge 0 Berücksichtigung fanden.

Beweis durch Diagonalisierung:

Beweis:

G^* ist in der lexikalischen Ordnung aufzählbar. (klar)

Die Teilmenge von G^* aller derjenigen Wörter, die Kodierung einer Turingmaschine sind, ist auch aufzählbar.

(Es gibt TM, die für $w \in G^*$ entscheiden kann, ob dieses die zulässige Kodierung einer TM ist.)

Betrachtet man die unendliche Matrix mit den Spalten w_1, w_2, w_3, \dots , den Zeilen A_1, A_2, A_3, \dots und den Einträgen: 1, falls $w_i \in L(A_i)$ bzw. 0, falls $w_i \notin L(A_i)$, so entspricht L_d gerade der Diagonalen, von der nur die Einträge 0 Berücksichtigung fanden.

Die Matrix der charakteristischen Funktion

$L_d \notin \mathcal{R}e:$

Charakteristische Funktion für $L(M_i)$ und L_d :

	w_0	w_1	w_2	\cdots	w_n	\cdots
M_0	0	1	1	\cdots	0	\cdots
M_1	1	1	0	\cdots	1	\cdots
M_2	1	0	0	\cdots	1	\cdots
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
M_n	1	1	0	\cdots	1	\cdots
\vdots						



Die Matrix der charakteristischen Funktion

$L_d \notin \mathcal{R}e:$

Charakteristische Funktion für $L(M_i)$ und L_d :

	w_0	w_1	w_2	\cdots	w_n	\cdots
M_0	0	1	1	\cdots	0	\cdots
M_1	1	1	0	\cdots	1	\cdots
M_2	1	0	0	\cdots	1	\cdots
\vdots						
M_n	1	1	0	\cdots	1	\cdots
\vdots						



Die Matrix der charakteristischen Funktion

$L_d \notin \mathcal{R}e$:

Charakteristische Funktion für $L(M_i)$ und L_d :

	w_0	w_1	w_2	\cdots	w_n	\cdots
M_0	0	1	1	\cdots	0	\cdots
M_1	1	1	0	\cdots	1	\cdots
M_2	1	0	0	\cdots	1	\cdots
\vdots						
M_n	1	1	0	\cdots	1	\cdots
\vdots						
L_d	1	0	1	\cdots	0	\cdots

Die Matrix der charakteristischen Funktion

$L_d \notin \mathcal{R}e:$

Charakteristische Funktion für $L(M_i)$ und L_d :

	w_0	w_1	w_2	\cdots	w_n	\cdots
M_0	0	1	1	\cdots	0	\cdots
M_1	1	1	0	\cdots	1	\cdots
M_2	1	0	0	\cdots	1	\cdots
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots
M_n	1	1	0	\cdots	1	\cdots
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\ddots
L_d	1	0	1	\cdots	0	\cdots

Beendigung des Beweises

Angenommen, es sei L_d doch aufzählbar, d.h. es gibt eine TM A_j in der Aufzählung aller (Kodierungen von) Turingmaschinen die L_d akzeptiert: $L_d = L(A_j)$. Nun gilt für das Wort w_j entweder $w_j \in L_d$ oder $w_j \notin L_d$. In beiden Fällen ergibt sich ein Widerspruch wie folgt:

$$w_j \in L_d \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \notin L(A_j) \xrightarrow{\text{(Annahme)}} w_j \notin L_d.$$

Andererseits auch:

$$w_j \notin L_d \xrightarrow{\text{(Annahme)}} w_j \notin L(A_j) \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \in L_d.$$

Wegen Satz 20.1 sind weder L_d noch $G^* \setminus L_d$ rekursiv (entscheidbar).

$G^* \setminus L_d = \{w_i \mid w_i \in L(A_i)\}$ ist jedoch eine aufzählbare Menge.

Beendigung des Beweises

Angenommen, es sei L_d doch aufzählbar, d.h. es gibt eine TM A_j in der Aufzählung aller (Kodierungen von) Turingmaschinen die L_d akzeptiert: $L_d = L(A_j)$. Nun gilt für das Wort w_j entweder $w_j \in L_d$ oder $w_j \notin L_d$. In beiden Fällen ergibt sich ein Widerspruch wie folgt:

$$w_j \in L_d \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \notin L(A_j) \xrightarrow{\text{(Annahme)}} w_j \notin L_d.$$

Andererseits auch:

$$w_j \notin L_d \xrightarrow{\text{(Annahme)}} w_j \notin L(A_j) \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \in L_d.$$

Wegen Satz 20.1 sind weder L_d noch $G^* \setminus L_d$ rekursiv (entscheidbar).

$G^* \setminus L_d = \{w_i \mid w_i \in L(A_i)\}$ ist jedoch eine aufzählbare Menge.

Beendigung des Beweises

Angenommen, es sei L_d doch aufzählbar, d.h. es gibt eine TM A_j in der Aufzählung aller (Kodierungen von) Turingmaschinen die L_d akzeptiert: $L_d = L(A_j)$. Nun gilt für das Wort w_j entweder $w_j \in L_d$ oder $w_j \notin L_d$. In beiden Fällen ergibt sich ein Widerspruch wie folgt:

$$w_j \in L_d \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \notin L(A_j) \xrightarrow{\text{(Annahme)}} w_j \notin L_d.$$

Andererseits auch:

$$w_j \notin L_d \xrightarrow{\text{(Annahme)}} w_j \notin L(A_j) \xrightarrow{\text{(Def. von } L_d\text{)}} w_j \in L_d.$$

Wegen Satz 20.1 sind weder L_d noch $G^* \setminus L_d$ rekursiv (entscheidbar).

$G^* \setminus L_d = \{w_i \mid w_i \in L(A_i)\}$ ist jedoch eine aufzählbare Menge.

Beendigung des Beweises

Angenommen, es sei L_d doch aufzählbar, d.h. es gibt eine TM A_j in der Aufzählung aller (Kodierungen von) Turingmaschinen die L_d akzeptiert: $L_d = L(A_j)$. Nun gilt für das Wort w_j entweder $w_j \in L_d$ oder $w_j \notin L_d$. In beiden Fällen ergibt sich ein Widerspruch wie folgt:

$$w_j \in L_d \xrightarrow{(\text{Def. von } L_d)} w_j \notin L(A_j) \xrightarrow{(\text{Annahme})} w_j \notin L_d.$$

Andererseits auch:

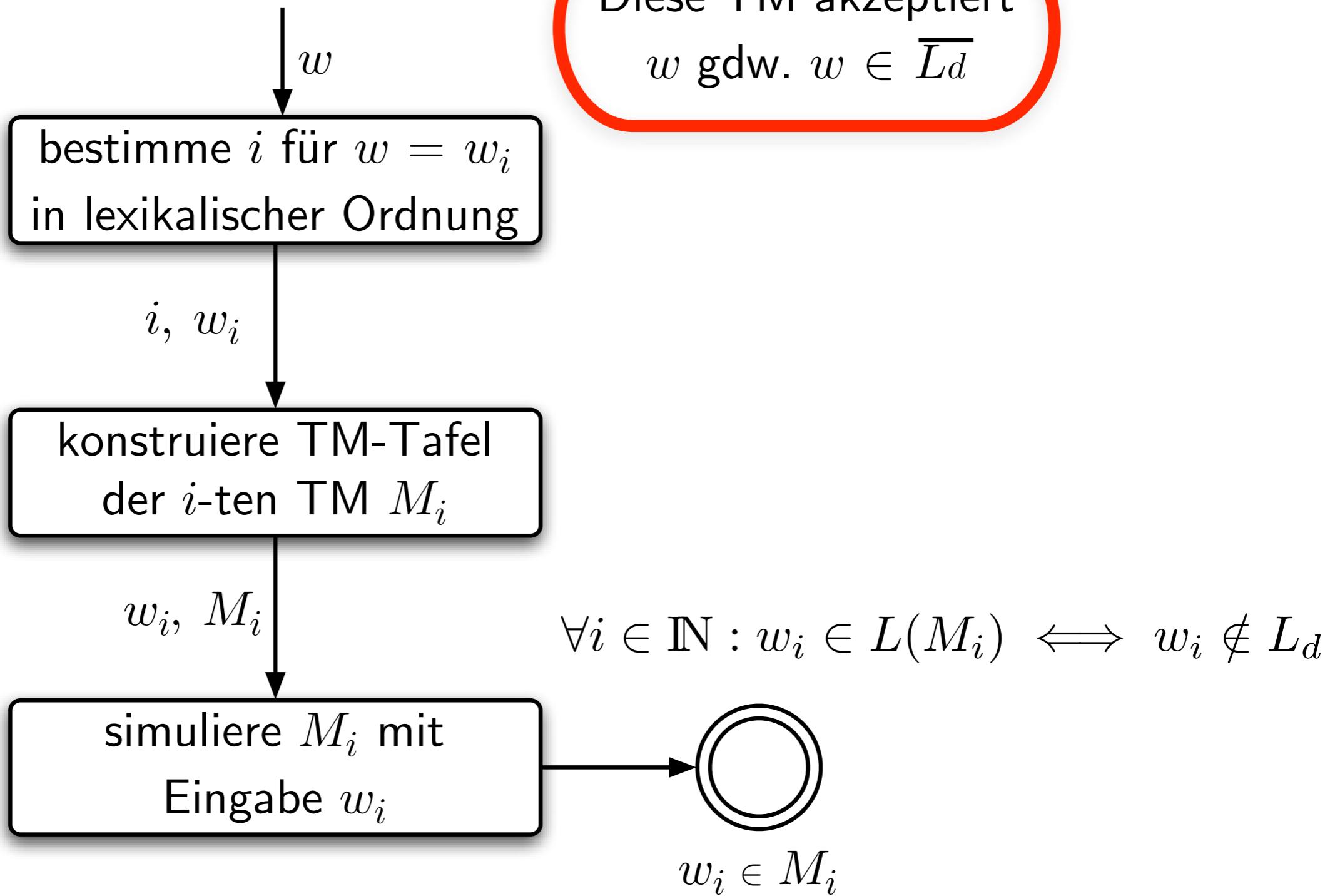
$$w_j \notin L_d \xrightarrow{(\text{Annahme})} w_j \notin L(A_j) \xrightarrow{(\text{Def. von } L_d)} w_j \in L_d.$$

Wegen Satz 20.1 sind weder L_d noch $G^* \setminus L_d$ rekursiv (entscheidbar).

$G^* \setminus L_d = \{w_i \mid w_i \in L(A_i)\}$ ist jedoch eine aufzählbare Menge.

das Komplement von L_d

... ist aufzählbar:



Das Halteproblem

Wichtig zur Erkennung von Endlosschleifen in Programmen.

Definition 19.9:

Sei $H := \left\{ \langle A \rangle \langle w \rangle \in \{0, 1\}^* \mid \begin{array}{l} \text{die TMA } A \text{ hält bei Eingabe von } w \in \Sigma^* \text{ an} \end{array} \right\}$.

Das Halteproblem

Wichtig zur Erkennung von Endlosschleifen in Programmen.

Definition 19.9:

Sei $H := \left\{ \langle A \rangle \langle w \rangle \in \{0, 1\}^* \mid \begin{array}{l} \text{die TMA hält bei Eingabe von } w \in \Sigma^* \text{ an} \end{array} \right\}$.

Programm $P =$ Turingmaschine A

Programmeingabe $x =$ Eingabewort w der TM A

Das Halteproblem

Satz: Die Haltesprache H ist aufzählbar.

Es gilt aber folgendes Negativresultat:

Satz 20.4:

Die Menge H aus Def. 19.9 ist nicht entscheidbar.

Das Halteproblem

Satz: Die Haltesprache H ist aufzählbar.

Beweis:

Es gibt eine TM, die jede TM A simulieren kann: die sogenannte **Universelle Turingmaschine** (UTM).

Die UTM bekommt (A, w) als Eingabe (geeignet kodiert) und simuliert A auf der Eingabe w .

Also akzeptiert die UTM die Menge H .

Es gilt aber folgendes Negativresultat:

Satz 20.4:

Die Menge H aus Def. 19.9 ist nicht entscheidbar.

Das Halteproblem

Satz: Die Haltesprache H ist aufzählbar.

Beweis:

Es gibt eine TM, die jede TM A simulieren kann: die sogenannte **Universelle Turingmaschine** (UTM).

Die UTM bekommt (A, w) als Eingabe (geeignet kodiert) und simuliert A auf der Eingabe w .

Also akzeptiert die UTM die Menge H .

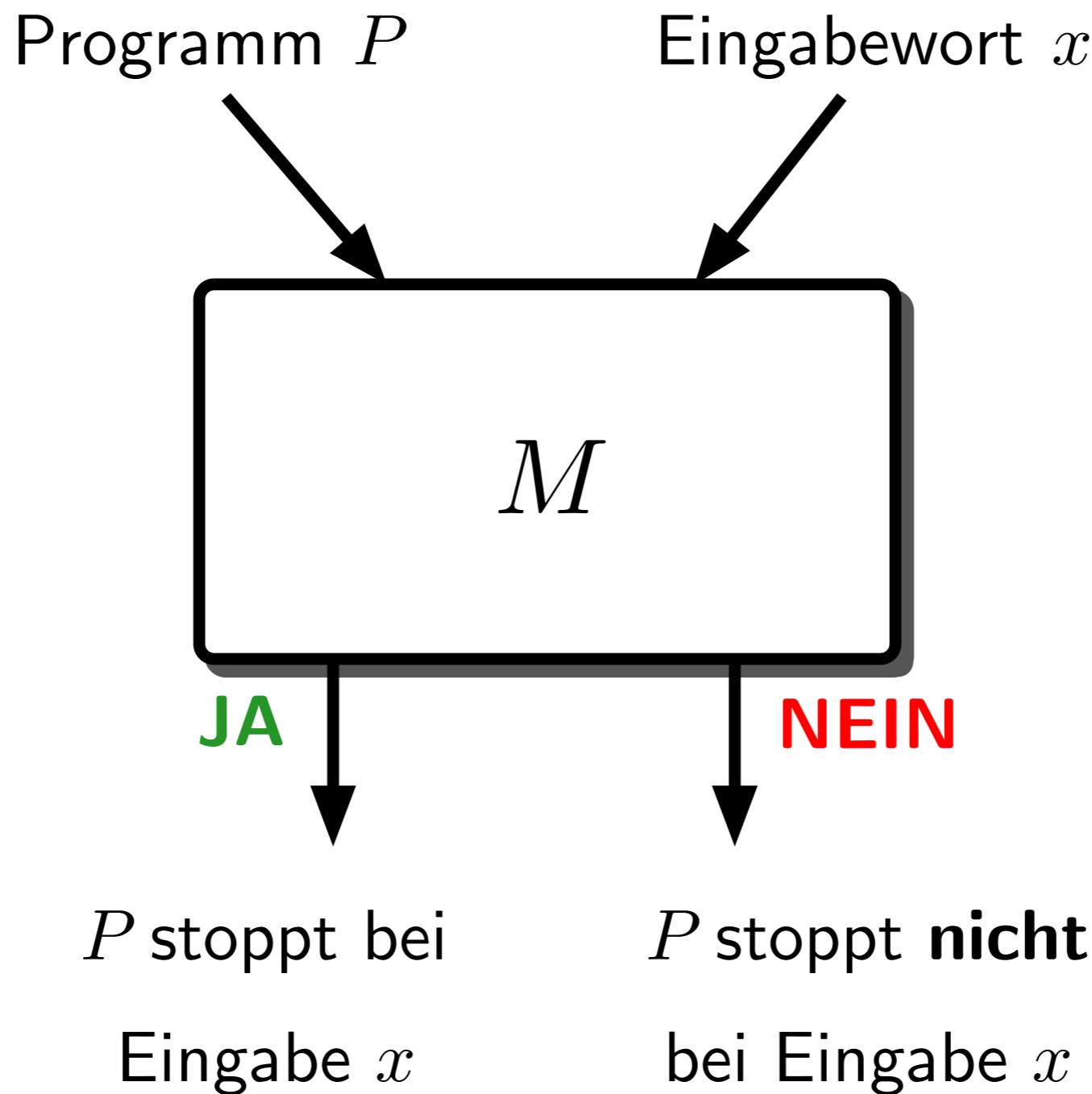
Es gilt aber folgendes Negativresultat:

Satz 20.4:

Die Menge H aus Def. 19.9 ist nicht entscheidbar.

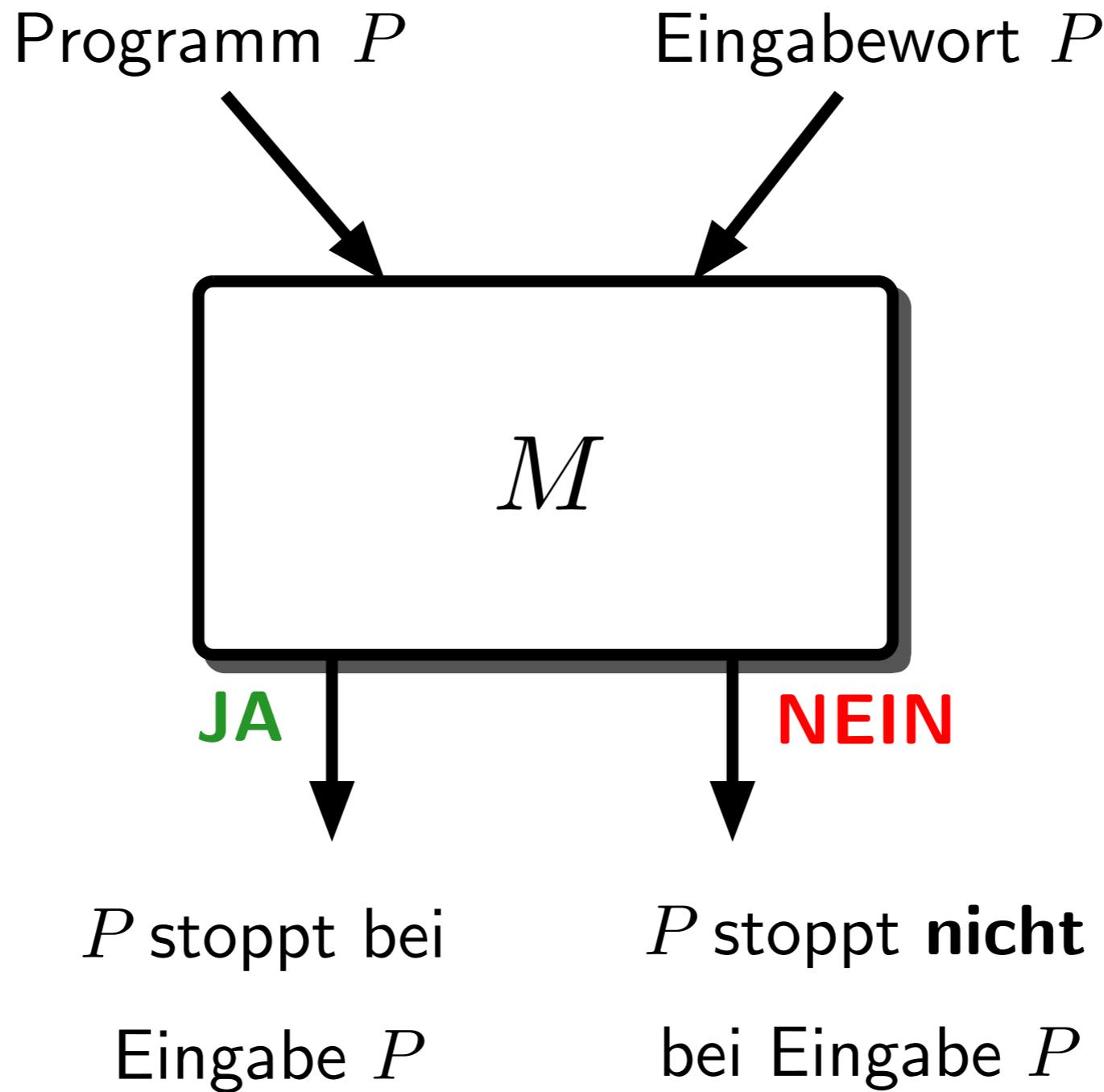
indirekter Beweis durch Selbstanwendung

Angenommen M löst das Halteproblem H :



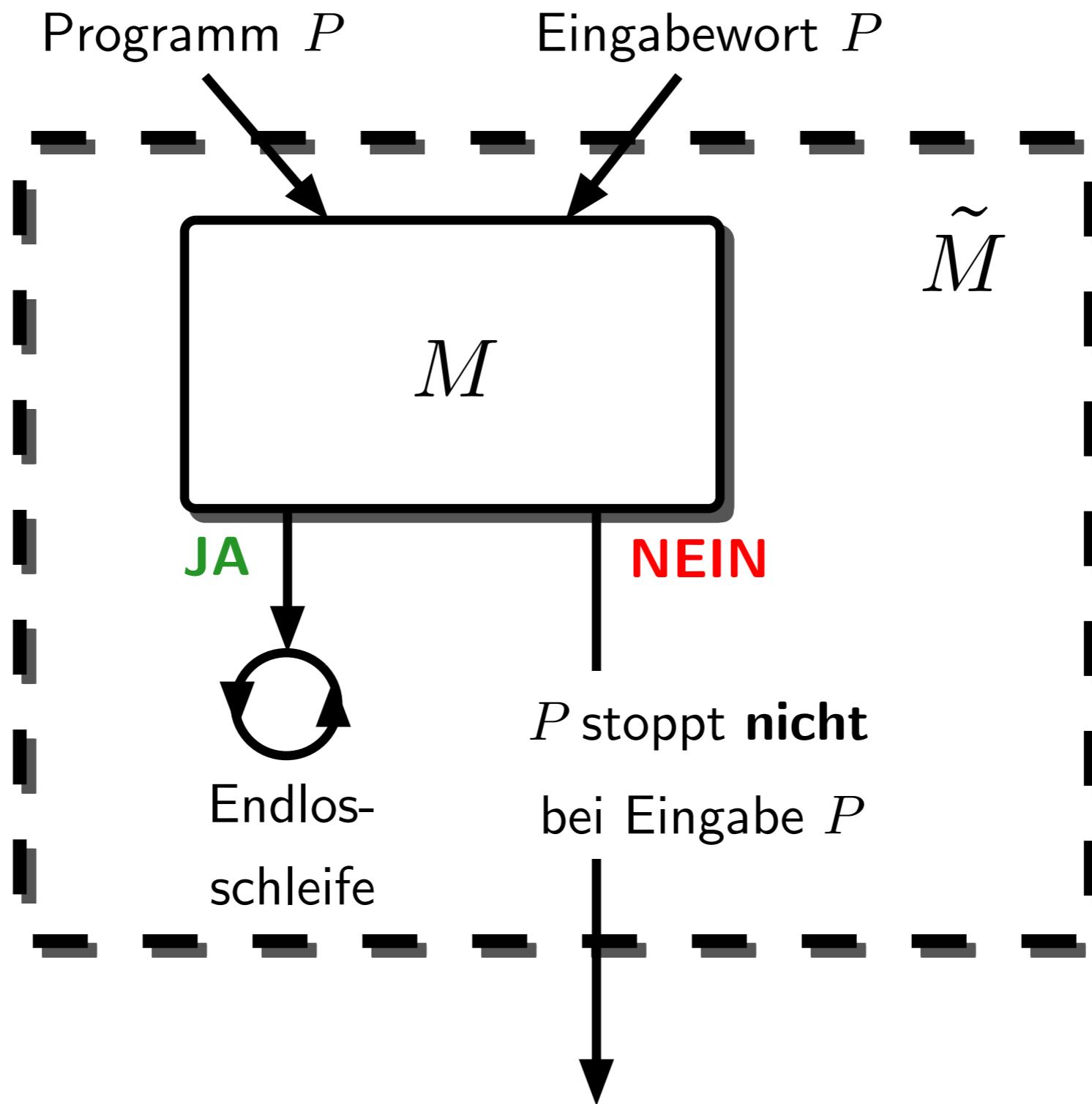
Halteproblem (2)

Eingabewort x ersetzt durch das Programm P :



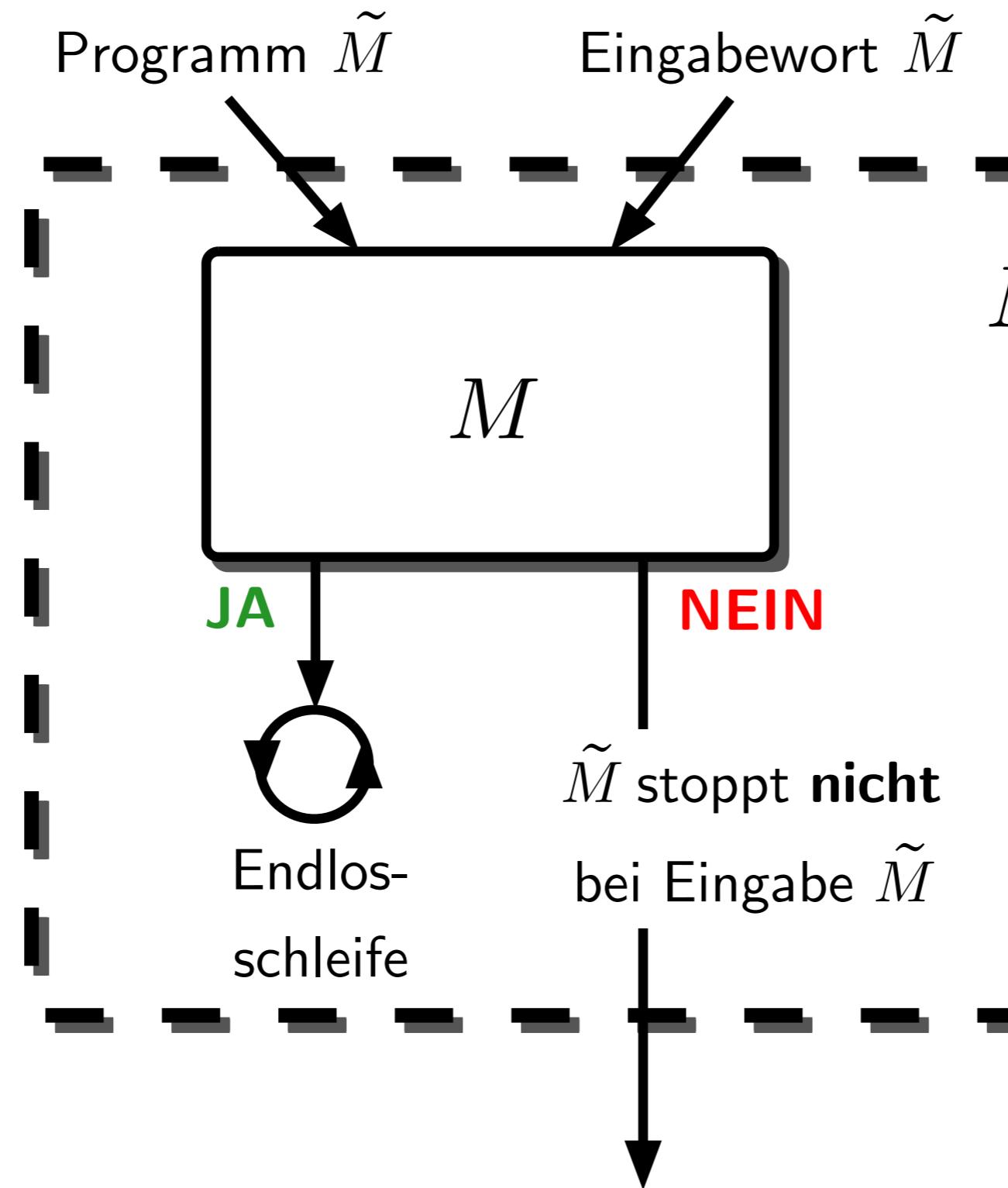
Halteproblem (3)

Neue Maschine \tilde{M} mit Endlosschleife bei JA :



Halteproblem (3): die Selbstanwendung

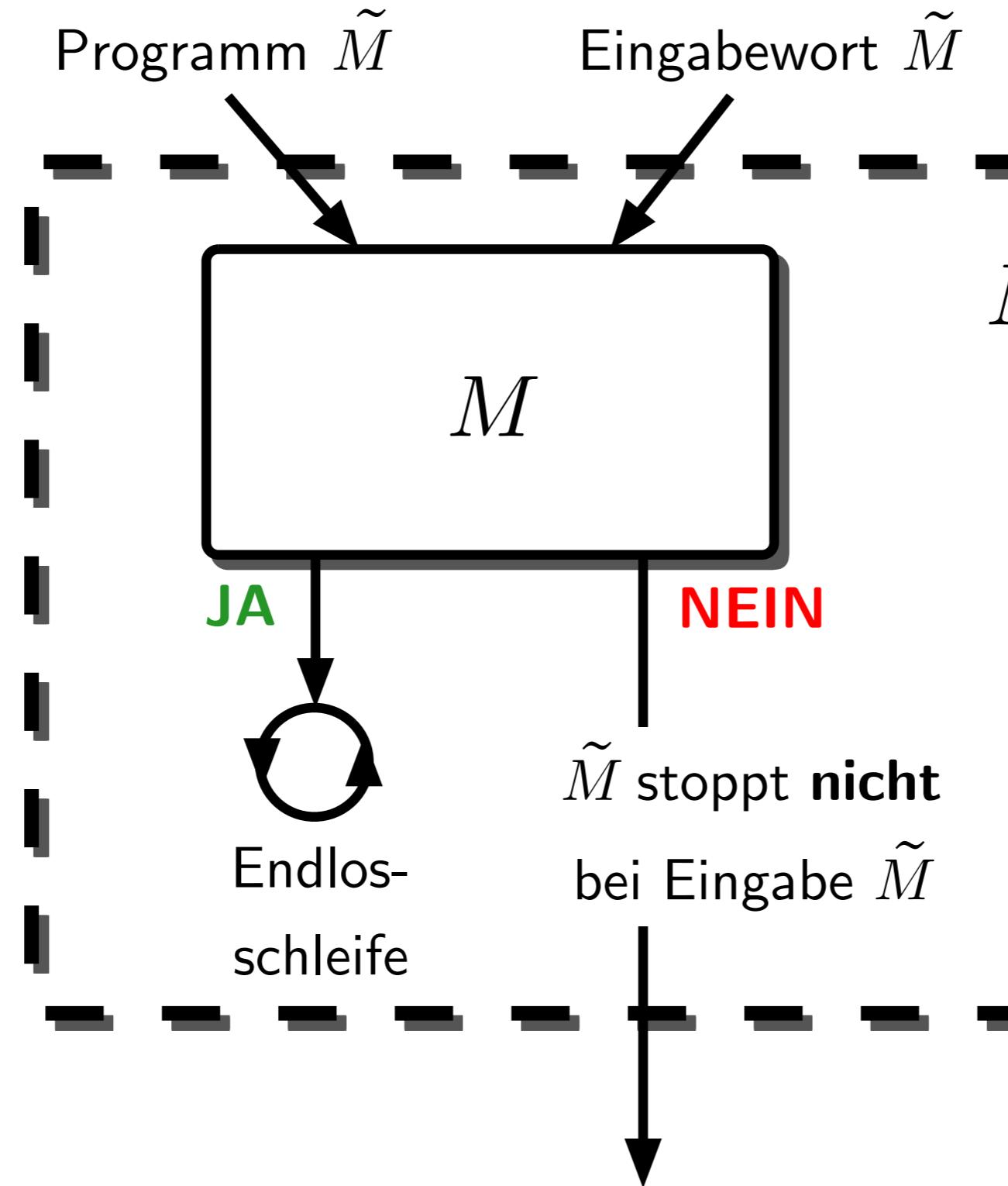
Als Programm P verwenden wir nun \tilde{M} :



dies ist ein
offensichtlicher
Widerspruch,
denn \tilde{M} stoppt!

Halteproblem (3): die Selbstanwendung

Als Programm P verwenden wir nun \tilde{M} :



*dies ist ein
offensichtlicher
Widerspruch,
denn \tilde{M} stoppt!*

Halteproblem und die Selbstanwendung

Als Programm P v



Eine Hierarchie von Sprachfamilien

Dies ist die Chomsky-Hierarchie.

	Familie	$\mathcal{R}eg$	der regulären Mengen
\subsetneq	Familie	$\mathcal{C}f$	der kontextfreien Mengen
\subsetneq	Familie	$\mathcal{C}s$	der kontextsensitiven Mengen
\subsetneq	Familie	$\mathcal{R}ec$	der entscheidbaren Mengen
\subsetneq	Familie	$\mathcal{R}e$	der aufzählbaren Mengen
\subsetneq	Familie		der abzählbaren Mengen
\neq	Familie		der überabzählbaren Mengen

Die *Haltesprache* ist *unentscheidbar*, aber *aufzählbar*.
Die *Diagonalsprache* ist ab-, aber nicht aufzählbar.

Eine Hierarchie von Sprachfamilien

Dies ist die Chomsky-Hierarchie.

- Familie $\mathcal{R}eg$ der **regulären Mengen**
- \subsetneq Familie $\mathcal{C}f$ der **kontextfreien Mengen**
- \subsetneq Familie $\mathcal{C}s$ der **kontextsensitiven Mengen**
- \subsetneq Familie $\mathcal{R}ec$ der **entscheidbaren Mengen**
- \subsetneq Familie $\mathcal{R}e$ der **aufzählbaren Mengen**
- \subsetneq Familie der **abzählbaren Mengen**
- \neq Familie der **überabzählbaren Mengen**

Die *Haltesprache* ist *unentscheidbar*, aber *aufzählbar*.
Die *Diagonalsprache* ist ab-, aber nicht aufzählbar.

Folgerung

Alle Inklusionen dieser Sprachfamilien wurden nun als echte Inklusionen erkannt:
keine zwei Sprachfamilien sind identisch!

Folgende Beziehungen sind nun bewiesen:

- Family der **regulären Mengen**
- \subsetneq Family der **kontextfreien Mengen**
- \subsetneq Family der **kontextsensitiven Mengen**
- \subsetneq Family der **entscheidbaren Mengen**
- \subsetneq Family der **aufzählbaren Mengen**
- \subsetneq Family der **abzählbaren Mengen**
- \neq Family der **überabzählbaren Mengen**

Reduktionen

Definition 20.1 Die Entscheidbarkeit des Problems P ist auf das Problem P' reduzierbar, wenn man aus einer TM A' , die P' entscheidet, effektiv eine TM A konstruieren kann, die P entscheidet.

A A'

kurz: $P \leq P'$

Satz 20.1 (Reduktion) Angenommen L lässt sich auf L' reduzieren und L ist unentscheidbar, dann ist auch L' unentscheidbar.

Beweis: Wenn sich L auf L' reduzieren lässt, dann kann man mit Hilfe der TM A' eine TM A konstruieren, die L entscheidet. Wäre nun L' entscheidbar, dann wäre auch L entscheidbar. Widerspruch.

Reduktionen

Definition 20.1 Die Entscheidbarkeit des Problems P ist auf das Problem P' reduzierbar, wenn man aus einer TM A' , die P' entscheidet, effektiv eine TM A konstruieren kann, die P entscheidet.

A A'

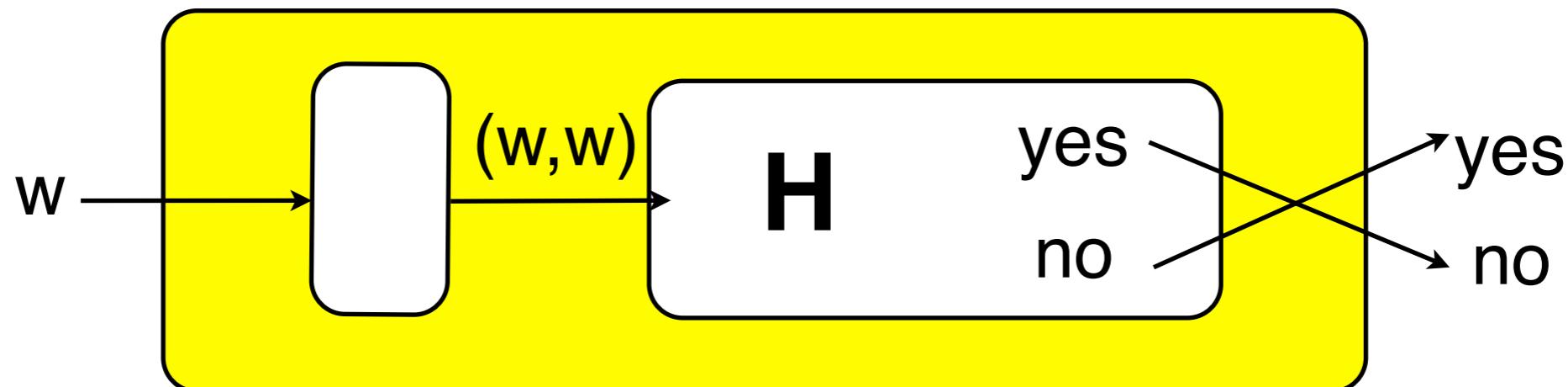
kurz: $P \leq P'$

Satz 20.1 (Reduktion) Angenommen L lässt sich auf L' reduzieren und L ist unentscheidbar, dann ist auch L' unentscheidbar.

Beweis: Wenn sich L auf L' reduzieren lässt, dann kann man mit Hilfe der TM A' eine TM A konstruieren, die L entscheidet. Wäre nun L' entscheidbar, dann wäre auch L entscheidbar. Widerspruch.

Halteproblem und L_d

- Wir kodieren H und L_d über dem Alphabet $\{0,1\}$
- Unentscheidbarkeit von H kann durch Reduktion von L_d auf H gezeigt werden:



$$w \in L_d \iff \langle w \rangle \langle w \rangle \notin H$$

- D.h.: wenn H entscheidbar ist, dann ist auch L_d entscheidbar.
Letzteres ist aber nicht der Fall!

Somit haben wir einen weiteren Beweis für die Unentscheidbarkeit von H .

Wichtige Unentscheidbarkeitsresultate:



Schon als unentscheidbar kennen wir:

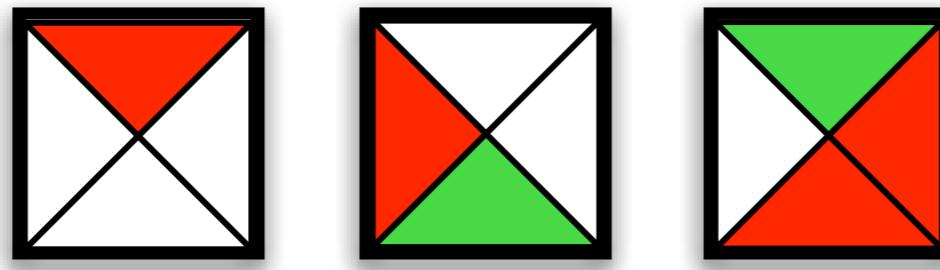
1. Sprache L_d (...ist nicht mal aufzählbar...)
2. Komplement von L_d
3. Halteproblem, d.h., Sprache H



außerdem unentscheidbar:

1. Kachelprobleme
2. 10. Hilbertsches Problem
3. Vorkommen einer 1 als Bild einer Funktion
4. jedes nichttriviale Problem über Turingmaschinen!

Kachel-/Dominoprobleme



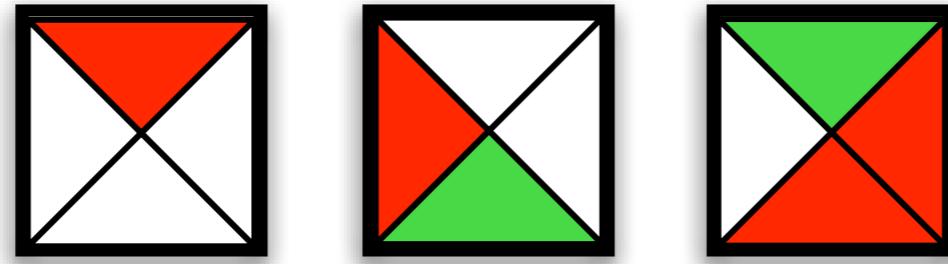
Nur gleichfarbige Seiten dürfen aneinander gelegt werden!
Steine dürfen nicht gedreht werden!

Gegeben: Eine Menge von r Kacheltypen $\mathcal{R} = \{K_1, K_2, \dots, K_r\}$, $n \in \mathbb{N}$ (beliebig viele Kacheln von jedem Typ)

Gesucht: (A) Kann man den ersten Quadranten der euklidischen Ebene korrekt kacheln?
(B) Kann man eine Fläche der Größe $n \times n$ korrekt kacheln?

Antwort: JA oder NEIN

Kachelproblem



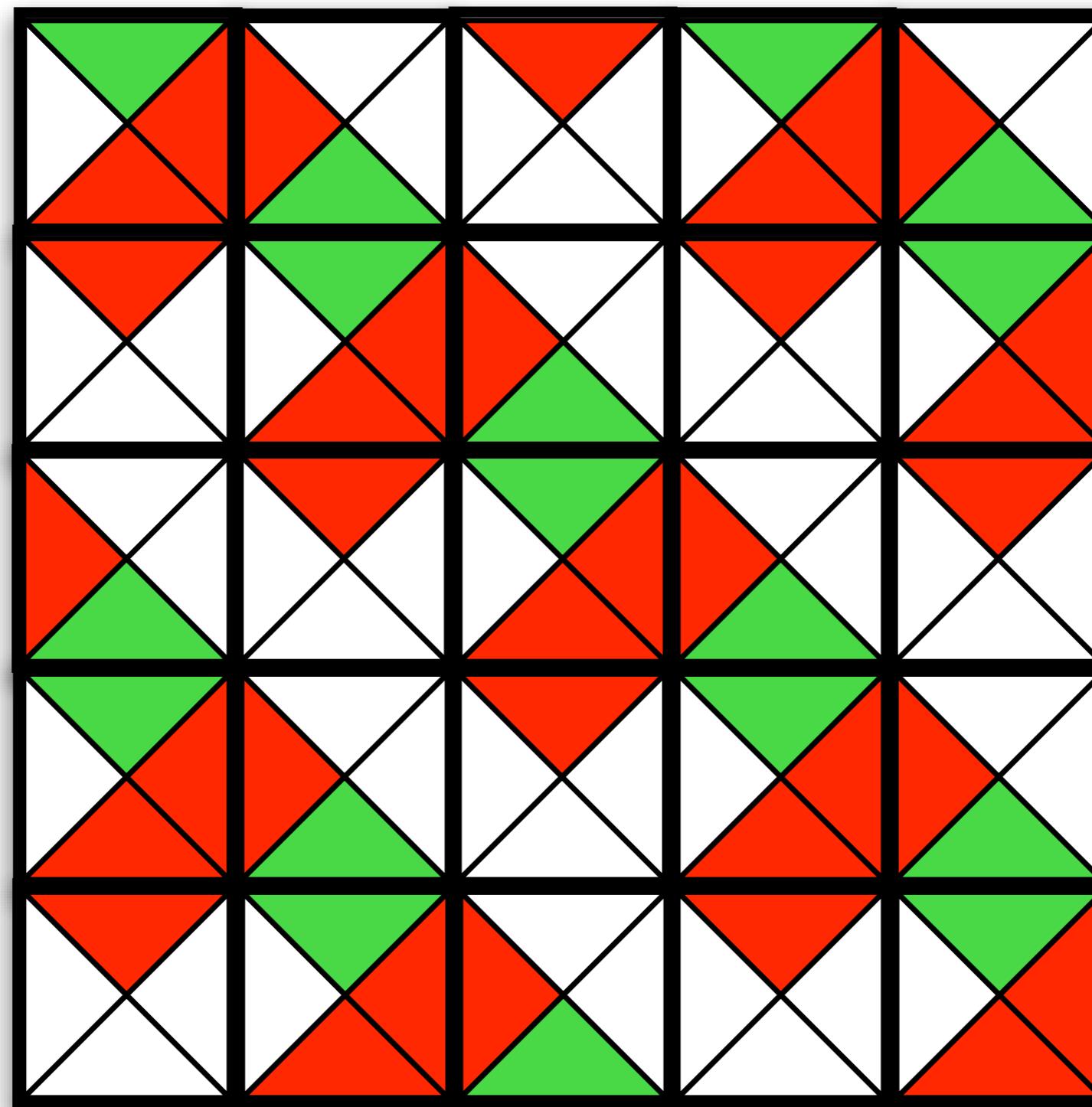
Nur gleichfarbige Seiten dürfen aneinander gelegt werden!
Steine dürfen nicht gedreht werden!

Satz 20.5:

Das Kachelproblem aus A ist unentscheidbar!

Für das endliche Kachelproblem B gibt es zur Zeit deterministische Verfahren nur mit exponentiellem Aufwand,
denn das Problem ist \mathcal{NP} -vollständig.

typische Lösung eines endl. Kachelproblems:



Post'sches Korrespondenzproblem (1947)

nach: Emil Leon Post (* 11. Februar 1897 in Augustów, Polen; † 21. April 1954 in New York)

Definition 20.3:

Sei $P \subseteq \Sigma^* \times \Sigma^*$ eine endliche Menge von Wortpaaren
 $P := \{(u_i, v_i) \mid 1 \leq i \leq |P|, u_i, v_i \in \Sigma^*\}$.

Als Post'sches Korrespondenzproblem (*PCP*) bezeichnet man folgendes Problem:

PCP:

Gegeben:	Eine beliebige, endliche Menge $P = \{(u_i, v_i) \mid 1 \leq i \leq P , u_i, v_i \in \Sigma^*\} \subseteq \Sigma^* \times \Sigma^*$
Gesucht:	Indexfolge $i_1 i_2 \dots i_k$ mit $\forall_{j=1}^k : 1 \leq i_j \leq P $ und $u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}$
Antwort:	JA oder NEIN

PCP-Beispiel

Die Paare: 1: $\begin{bmatrix} a \\ ba \end{bmatrix}$, 2: $\begin{bmatrix} aa \\ a \end{bmatrix}$, 3: $\begin{bmatrix} ab \\ aa \end{bmatrix}$. Gibt es Lösung?

Ja: eine Indexfolge ist 231, mit:

$$\begin{bmatrix} aa \\ a \end{bmatrix} \begin{bmatrix} ab \\ aa \end{bmatrix} \begin{bmatrix} a \\ ba \end{bmatrix}$$

PCP-Beispiel

Die Paare: 1: $\begin{bmatrix} a \\ ba \end{bmatrix}$, 2: $\begin{bmatrix} aa \\ a \end{bmatrix}$, 3: $\begin{bmatrix} ab \\ aa \end{bmatrix}$. Gibt es Lösung?

Ja: eine Indexfolge ist 231, mit:

$$\begin{bmatrix} aa \\ a \end{bmatrix} \begin{bmatrix} ab \\ aa \end{bmatrix} \begin{bmatrix} a \\ ba \end{bmatrix}$$

Unentscheidbarkeit des PCP

Satz 20.6:

Es gibt keinen Algorithmus, der für ein beliebiges *PCP* entscheidet, ob es eine Lösung besitzt!

Wenn das Alphabet nur ein Symbol enthält, dann ist jedes *PCP* entscheidbar.

Es konnte (mit sehr komplizierten Beweis) gezeigt werden, dass jedes *PCP* **mit höchstens zwei Wortpaaren** entscheidbar ist.

Das bisher kleinste, unentscheidbare *PCP* besitzt 7 Wortpaare.

Der Satz von Rice (1951)

von: Henry Gordon Rice (* 1920)

Satz 20.7:

Jede nichttriviale Eigenschaft \mathcal{S} der aufzählbaren Sprachen ist unentscheidbar.

Nichttriviale Eigenschaften zeichnen sich dadurch aus, dass es sowohl Mengen gibt, die sie erfüllen, als auch andere, die diese Eigenschaft nicht erfüllen.

Viele der bisherigen (und noch folgenden) Unentscheidbarkeitsresultate folgen aus dem Satz von Rice.

Kommt 1 als Funktionswert vor?

Definition 20.4:

Mit M_T sei die Menge aller (Kodierungen von) Turing-Maschinen M bezeichnet, die totale Funktionen $f_M : \Sigma^* \rightarrow \{0, 1\}$ berechnen.

Problem:

Gegeben:	Eine beliebige stets haltende Turing-Maschine
Gesucht:	Wird bei allen Eingaben stets die Ausgabe 0 berechnet?
Antwort:	JA oder NEIN

Das Problem, ob eine Funktion irgendwann den Wert 1 als Resultat erhält, ist **UNENTSCHEIDBAR** !

Unentscheidbarkeitsresultate

Satz 20.8:

Es gibt keinen Algorithmus, der für eine beliebige Funktion

$$f_M \in M_T$$

entscheidet, ob

$$f_M(w) = 1 \text{ für mindestens ein } w \in \Sigma^* \text{ gilt.}$$

Korollar:

Es ist unentscheidbar, ob eine beliebige, durch eine TM definierte, entscheidbare Menge M leer ist.

Beweis per Reduktion auf H

Sei $P := \{M \in M_T \mid \exists w \in \Sigma^* : f_M(w) = 1\}$ die Menge aller TM, die für mindesten eine Eingabe w eine 1 ausgeben.

Beweis per Reduktion auf H

Sei $P := \{M \in M_T \mid \exists w \in \Sigma^* : f_M(w) = 1\}$ die Menge aller TM, die für mindesten eine Eingabe w eine 1 ausgeben.

Abgenommen P wäre entscheidbar, dann gäbe es eine TM A_P , die die charakteristische Funktion $\chi_P : M_T \rightarrow \{0, 1\}$ berechnet:

$$\chi_P(M) = \begin{cases} 1, & \text{falls } M \in P \\ 0, & \text{sonst} \end{cases}$$

Beweis per Reduktion auf H

Sei $P := \{M \in M_T \mid \exists w \in \Sigma^* : f_M(w) = 1\}$ die Menge aller TM, die für mindesten eine Eingabe w eine 1 ausgeben.

Abgenommen P wäre entscheidbar, dann gäbe es eine TM A_P , die die charakteristische Funktion $\chi_P : M_T \rightarrow \{0, 1\}$ berechnet:

$$\chi_P(M) = \begin{cases} 1, & \text{falls } M \in P \\ 0, & \text{sonst} \end{cases}$$

Die TM $M_{B,w}$ simuliert bei der Eingabe n genau n Schritte von B auf der Eingabe w .

Wenn B nach genau n Schritten hält, dann gibt $M_{B,w}$ eine 1 aus; andernfalls eine 0.

Beweis per Reduktion auf H

Sei $P := \{M \in M_T \mid \exists w \in \Sigma^* : f_M(w) = 1\}$ die Menge aller TM, die für mindesten eine Eingabe w eine 1 ausgeben.

Abgenommen P wäre entscheidbar, dann gäbe es eine TM A_P , die die charakteristische Funktion $\chi_P : M_T \rightarrow \{0, 1\}$ berechnet:

$$\chi_P(M) = \begin{cases} 1, & \text{falls } M \in P \\ 0, & \text{sonst} \end{cases}$$

$$\chi_P(M_{B,w}) = 1$$

\iff

$$M_{B,w} \in P$$

\iff

$M_{B,w}$ drückt bei Eingabe von n eine 1

\iff

B hält auf w nach n Schritten an

\iff

$$w \in L(B)$$

Die TM $M_{B,w}$ simuliert bei der Eingabe n genau n Schritte von B auf der Eingabe w .

Wenn B nach genau n Schritten hält, dann gibt $M_{B,w}$ eine 1 aus; andernfalls eine 0.

Beweis per Reduktion auf H

Sei $P := \{M \in M_T \mid \exists w \in \Sigma^* : f_M(w) = 1\}$ die Menge aller TM, die für mindesten eine Eingabe w eine 1 ausgeben.

Abgenommen P wäre entscheidbar, dann gäbe es eine TM A_P , die die charakteristische Funktion $\chi_P : M_T \rightarrow \{0, 1\}$ berechnet:

$$\chi_P(M) = \begin{cases} 1, & \text{falls } M \in P \\ 0, & \text{sonst} \end{cases}$$

$$\chi_P(M_{B,w}) = 1$$

 \iff

$$M_{B,w} \in P$$

 \iff

$M_{B,w}$ drückt bei Eingabe von n eine 1

 \iff

B hält auf w nach n Schritten an

 \iff

$$w \in L(B)$$

Die TM $M_{B,w}$ simuliert bei der Eingabe n genau n Schritte von B auf der Eingabe w .

Wenn B nach genau n Schritten hält, dann gibt $M_{B,w}$ eine 1 aus; andernfalls eine 0.

Dann entscheidet A_P aber auch das Halteproblem. Widerspruch.

... noch mehr Unentscheidbares

Folgende Probleme sind unentscheidbar:

Ist L endliche Menge?

Ist L leere Menge?

Ist L unendliche Menge?

Ist L reguläre Menge?

Ist L kontextfreie Sprache?

Ist L entscheidbare Sprache?

Ist L eine Menge mit genau einem Element?

Die Kodierung von L geschieht hierbei stets in Form einer TM.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 21

μ -rekursive Funktionen (Exkurs)

Michael Köhler-Bußmeier

Wortfunktion versus ganzahlige Berechnungen

Zur Erinnerung: (Turing-)berechenbare Wortfunktionen:

Definition 19.1:

Seien Σ und Λ Alphabete.

Eine partielle (Wort-)Funktion $f : \Sigma^* \rightarrow \Lambda^*$ heißt **Turing-berechenbar** oder **partiell rekursiv** genau dann, wenn es eine $DTM T = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ gibt mit:

$$q_0 w \xrightarrow[T]{*} q_e v, \quad \text{für } q_e \in F \text{ und } v \in \Lambda^* \text{ mit } f(w) = v.$$

... der Funktionswert steht *ab der Kopfposition* auf dem Band
(abgesehen von nachfolgenden #'s)!

Wenn die $DTM T$ die Eingabe verwirft, oder nie anhält, so soll $f(w) := \perp$ die Undefiniertheit von f anzeigen.

TM-berechenbare Funktionen

Wie definieren Vossen/Witt
(Turing-)berechenbare Zahlenfunktionen?

Definition 19.2:

Seien $r, s \in \mathbb{N}$ und $r, s \geq 1$.

Eine (partielle) Funktion $f : \mathbb{N}^r \rightarrow \mathbb{N}^s$ heißt **Turingberechenbar** oder **partiell rekursiv** gdw. eine DTM $T = (Q, \{|, 0\}, \Gamma, \delta, q_0, \#, F)$ existiert mit

$$q_0 |^{m_1} 0 |^{m_2} 0 \dots |^{m_r} \xrightarrow[T]{*} p |^{n_1} 0 |^{n_2} 0 \dots |^{n_s}$$

mit $p \in F$, sowie

$$f(m_1, m_2, \dots, m_r) = (n_1, n_2, \dots, n_s)$$

gdw. der Funktionswert definiert ist.

Turing'sche These

Turing'sche These oder Church'sche These

Jede intuitiv berechenbare Funktion kann auch von einer Turingmaschine berechnet werden kann.

Was **intuitiv berechenbar** ist, kann nicht definiert werden.

Daher ist die Turingmaschine das Standardmodell, mit dem

Berechenbarkeit und der Algorithmusbegriff **definiert** wird.

Ein *Algorithmus* ist eine präzise, das heißt in einer festgelegten Sprache abgefasste endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung effektiv ausführbarer, elementarer Verarbeitungsschritte.

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
 - nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
 - einfache Schritte enthalten (**Elementarität**).
-
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
 - sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
 - nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
 - nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
 - einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
 - nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
 - nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
 - einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
 - nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
 - einfache Schritte enthalten (**Elementarität**).
 - auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
 - nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
 - einfache Schritte enthalten (**Elementarität**).
 - auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

Eigenschaften von Algorithmen

Ein Algorithmus soll:

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**)
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**). *Dies ist nicht notwendig, aber oft erwünscht!*

weitere Berechenbarkeitsdefinitionen

Präzisierungen des Begriffes *Berechenbarkeit* gab es, viele verschiedene: λ -Definierbarkeit, berechenbare arithmetische Funktionen, μ -rekursive Funktionen, Markov-Algorithmen, Post'sche Systeme und eben die Turing-Berechenbarkeit.

Wir betrachten nun noch die

primitiv-rekursiven Funktionen

und die

μ -rekursiven Funktionen!

primitiv-rekursive Funktionen (Basis...)

Definition 20.2:

Die Menge \mathcal{PR} der *primitiv-rekursiven Funktionen* besteht aus den folgenden Basisfunktionen und den sich daraus durch primitive Rekursion ergebenden Funktionen:

1. Basisfunktionen:

Basisfunktionen sind die folgenden Funktionen:

- a) Nachfolgerfunktion: $S : \mathbb{N} \rightarrow \mathbb{N} = \{0, 1, 2, \dots\}$ mit
$$S(n) := n + 1.$$

primitiv-rekursive Funktionen (Basis...)

1. b) konstante Funktionen: $C_q^r : \mathbb{N}^r \rightarrow \mathbb{N}$ für $r, q \in \mathbb{N}$

mit:

$$C_q^r(n_1, n_2, \dots, n_r) := q$$

für alle r -Tupel $(n_1, n_2, \dots, n_r) \in \mathbb{N}^r$. Die nullstellige Konstante $q \in \mathbb{N}$ wird nun so ausgedrückt: $C_q^0 := q$.

c) Projektionsfunktionen: $U_i^r : \mathbb{N}^r \rightarrow \mathbb{N}$ mit:

$$U_i^r(n_1, n_2, \dots, n_r) := n_i$$

für alle r -Tupel $(n_1, n_2, \dots, n_r) \in \mathbb{N}^r$.

primitiv-rekursive Funktionen (Substitution)

2. Substitution:

Sind $f : \mathbb{N}^r \rightarrow \mathbb{N}$ und $g_1, g_2, \dots, g_r : \mathbb{N}^m \rightarrow \mathbb{N}$ in \mathcal{PR} ,
so ist auch die Funktion $h : \mathbb{N}^m \rightarrow \mathbb{N}$ mit

$$h(n_1, n_2, \dots, n_m) := f(g_1(n_1, \dots, n_m), \dots, g_r(n_1, \dots, n_m))$$

in \mathcal{PR} .

primitiv-rekursive Funktionen (prim. Rekursion)

3. primitive Rekursion:

Sind $f : \mathbb{N}^r \rightarrow \mathbb{N}$ und $g : \mathbb{N}^{r+2} \rightarrow \mathbb{N}$ in \mathcal{PR} , so ist auch die folgende Funktion $h : \mathbb{N}^{r+1} \rightarrow \mathbb{N}$, die den folgenden

Rekursionsgleichungen genügt:

$$a) \quad h(0, n_1, n_2, \dots, n_r) := f(n_1, \dots, n_r),$$

$$b) \quad h(S(n), n_1, \dots, n_r) := g(n, h(n, n_1, \dots, n_r), n_1, \dots, n_r),$$

in \mathcal{PR} .

Man sagt: “ h entsteht durch primitive Rekursion aus f und g ”.

4. Andere Funktionen sollen nicht zu \mathcal{PR} gehören.

primitiv-rekursive Funktionen = Loop Programme

In Programmiersprachen entspricht die „**primitive Rekursion**“ einer „For-Schleife“ oder „Loop“, deren Endebedingung vor deren Eintritt bekannt ist und während ihres Ablaufs nicht geändert wird („**LOOP-Programme**“).

„**While**“-Programme besitzen zur Schleifentermination ein Prädikat, dessen Wert sich während der Schleifendurchgänge ändern darf.

„**While**“-Programme entsprechen die μ -rekursiven Funktionen, d.h. alle (Turing-)berechenbaren Funktionen!

Beispiele primitiver Rekursion

Beispiele:

1. **Addition in \mathbb{N} :** Die Addition in \mathbb{N} ist durch die folgenden Rekursionsgleichungen definiert: $add : \mathbb{N}^2 \rightarrow \mathbb{N}$ mit

$$add(0, n) := U_1^1(n),$$

$$add(S(m), n) := g(m, add(m, n), n)$$

$$g(x, y, z) := S(U_2^3(x, y, z)).$$

Übliche Kurzform:

$$0 + n := n,$$

$$(m + 1) + n := (m + n) + 1.$$

Beispiele primitiver Rekursion

Beispiele:

2. Multiplikation in \mathbb{N} : Die Multiplikation in \mathbb{N} , $mult : \mathbb{N}^2 \rightarrow \mathbb{N}$, ist definiert durch:

$$mult(0, n) := C_0^1(n),$$

$$mult(S(m), n) := g(m, mult(m, n), n)$$

$$g(x, y, z) := add(U_2^3(x, y, z), U_3^3(x, y, z)).$$

Kurzform:

$$0 \cdot n := 0,$$

$$(m + 1) \cdot n := m \cdot n + n.$$

Beispiele primitiver Rekursion

Beispiele:

3. Exponentialfunktion: $\exp(n) = 2^n$ ist in \mathcal{PR} -Darstellung:

$$\exp(0) := S(C_0^0),$$

$$\exp(S(n)) := g(n, \exp(n)),$$

$$g(x, y) := \text{mult}(C_2^2(x, y), U_2^2(x, y)).$$

oder alternativ auch

$$g(x, y) := \text{mult}(S(S(C_0^0))), U_2^2(x, y)).$$

Etwas einfacher, und durch abkürzende Schreibweisen auf das Wesentliche verkürzt, kann diese primitive Rekursion auch so dargestellt werden:

$$\exp(0) := 1,$$

$$\exp(n + 1) := 2 \cdot \exp(n).$$

Eigenschaften prim.-rek. Funktionen

Jede primitiv-rekursive Funktion $f \in \mathcal{PR}$ ist eine **totale** Funktion, d.h. sie ist überall definiert. Außerdem ist $f(x)$ stets eindeutig bestimmt.

Auch die Ackermannfunktion ist total und (Turing-) berechenbar.
Sie ist jedoch **nicht** mehr primitiv-rekursiv!

Definition 20.3:

Die **Ackermann-Funktion** ist definiert durch:

- a) $f(0, n) := n + 1,$
- b) $f(m + 1, 0) := f(m, 1),$
- c) $f(m + 1, n + 1) := f(m, f(m + 1, n)).$

μ -rekursive Funktionen

Definition 20.4:

Eine Funktion f heißt **μ -rekursiv (partiell-rekursiv)**, wenn f entweder

1. eine primitiv-rekursive Grundfunktion ist oder
2. durch Substitution aus μ -rekursiven Funktionen entsteht oder
3. durch primitive Rekursion aus μ -rekursiven Funktionen entsteht oder
4. durch Anwendung der Minimalisierung aus μ -rekursiven Funktionen entsteht.

der μ -Operator

Sei $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ eine beliebige partielle (im allgemeinen also nicht totale) Funktion und (x_1, x_2, \dots, x_n) ein beliebiges, festes n -Tupel aus \mathbb{N}^n . Dann betrachten wir die Folge von existierenden (!) Funktionswerten:

$$y_0 := f(x_1, x_2, \dots, x_n, 0),$$

$$y_1 := f(x_1, x_2, \dots, x_n, 1),$$

⋮

$$y_k := f(x_1, x_2, \dots, x_n, k),$$

⋮

Wenn dann $f(x_1, x_2, \dots, x_n, i)$ für alle i mit $0 \leq i < k$ definiert ist, dann setzen wir:

$$\psi(x_1, x_2, \dots, x_n) := \mu y [f(x_1, x_2, \dots, x_n, y) = 0] := k$$

der μ -Operator (Forts.)

Die Funktion $\psi(x_1, x_2, \dots, x_n)$ sei dann nicht definiert, wenn in der Folge $f(x_1, x_2, \dots, x_n, i), 0 \leq i$ niemals der Wert 0 auftritt oder wenn zwar $f(x_1, x_2, \dots, x_n, i) = 0$ erstmalig für ein i trifft, aber $f(x_1, x_2, \dots, x_n, k)$ für ein $k < i$ nicht definiert ist.

ψ ist eine arithmetische Funktion mit $\psi : \mathbb{N}^n \rightarrow \mathbb{N}$ und wir sagen: ψ wird **durch den μ -Operator angewandt auf f definiert** oder ψ entsteht aus f durch **Minimalisierung**.

Zusammenfassung

Satz 21.4:

Für eine n -stellige Zahlenfunktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$ sind folgende Aussagen äquivalent:

1. f ist durch eine 1-Band DTM berechenbar,
2. f ist durch eine k -band DTM berechenbar,
3. f ist μ -rekursiv (partiell-rekursiv),
4. f ist durch einen 2-Zählerautomaten berechenbar,
5. f ist durch einen 2-Kellerautomaten berechenbar,
6. f ist durch eine RAM berechenbar, ...

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 21

...

Michael Köhler-Bußmeier