

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Organisatorisches

Vorlesung:

- Di 11:15 – 12:45 T.01

Proseminar:

- Di 13:00 – 13:45 T.01
- Mi 14:15 – 15:00 T.03
- Beginn: nächste Woche

Organisatorisches

Heimübungen:

- Jede Woche ein Übungsblatt (dienstags)
- 1. Blatt diese Woche
- Abgabe: Dienstag bis 11:00 Uhr
- Erfolgreicher Abschluss (Gruppen zu 2-4 Personen)
 - mind. 50% der erreichbaren Punkte
 - mind. einmal korrekt Vorrechnen
 - zwei Tests während des Semesters (50% der Gesamtbewertung)
- Erste **bewertete** Aufgabe: übernächste Woche
- Vorstellung *einer* Musterlösung im Proseminar

Organisatorisches

Proseminargruppen:

- PLUSonline
 - erreichbar über https://online.uni-salzburg.at/plus_online/webnav.ini
 - ITS-Login und Passwort
 - Bis zu 25 Teilnehmer
 - eine Gruppe
 - Übungsaufgaben und Vorlesungsfolien über die Webseite der Vorlesung: <http://fl.cosy.sbg.ac.at>
- Abmeldung bis zum 25.10.2017, 23:55 Uhr
 - Nach diesem Zeitpunkt ist keine Abmeldung mehr möglich und es folgt eine Bewertung der Leistungen am Ende des Semesters

Organisatorisches

Klausur:

- Eine Korrelation mit den PS-Aufgaben ist zu erwarten
- Es gab in der Vergangenheit einen direkten Zusammenhang zwischen PS-Teilnahme bzw. -abgabe und gutem Abschneiden bei Klausuren

Sprechzeiten:

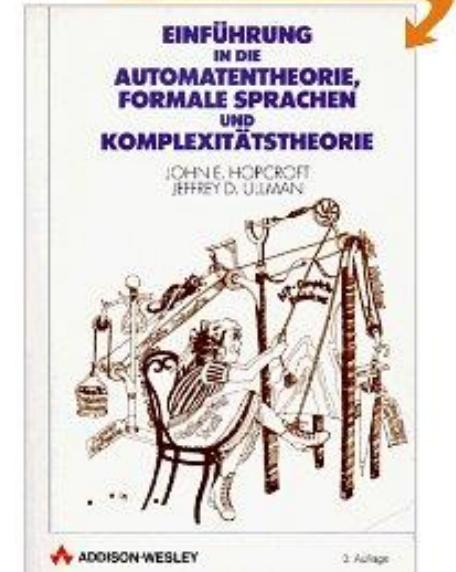
- Di 10:00 – 11:00
- (Raum 2.23, Jakob-Haringer-Straße 2)

Organisatorisches

Literatur:

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*, 3. Auflage, Pearson Studium, 2011
- Die Vorlesungsfolien wurden unter Verwendung der Folien und des Skriptes der Vorlesung „*Einführung in die Berechenbarkeit, Komplexität und Formale Sprachen*“ von Prof. Dr. Johannes Blömer, Prof. Dr. Friedhelm Meyer auf der Heide bzw. Prof. Dr. Christian Scheideler erstellt. Die meisten Bilder, Definitionen und Beschreibungen wurden aus den oben genannten Unterlagen übernommen.
- Die Folien wurden von Eva Lugstein (ehem. Studienassistentin am FB Cowi) überarbeitet.

Hier klicken **Blick ins Buch!**



Quelle: amazon.de

Motivation

- Was lässt sich mit dem Computer lösen?
Wie effizient lassen sich einzelne Probleme lösen?
- Grenzen und Möglichkeiten eines Rechners.
- Eine geeignete Formalisierung ist Voraussetzung
für eine systematische Lösung.
- Als Ausdrucksmittel muss man passende Kalküle
und Notationen anwenden können.

Ziele

- Einen Überblick über grundlegende Formalisierungsmethoden zu bekommen
- Die für die Methoden typische Techniken zu erlernen
- Techniken an typischen Beispielen anzuwenden

Insgesamt soll erlernt werden:

- Aufgaben präzise zu formalisieren und zu analysieren
- berechenbare von unberechenbaren Problemen zu unterscheiden
- festzustellen, ob ein Problem effiziente Lösungen haben kann

Durchführung

Zu jedem Bereich soll(en):

- mit einigen typischen Beispielen motivierend hineingeführt werden
- der konzeptionelle Kern der Methode vorgestellt werden
- Anwendungstechniken an Beispielen gezeigt und in den Proseminaren erfahren werden
- an einem durchgehenden Beispiel größere Zusammenhänge gelernt werden

Inhaltsangabe

- Einleitung, Motivation
 - Turingmaschinen
 - Arbeitstechniken
- } **Einführung**
- Unentscheidbare Probleme
 - Das Halteproblem
 - Reduktionen
- } **Berechenbarkeit**
- Zeitkomplexität
 - Die Klassen P und NP
 - NP-Vollständigkeit
 - NP-vollständige Probleme
- } **Komplexität**
- Formale Sprachen und Automaten
 - Kellerautomaten und kontextfreie Sprachen
 - Kontextsensitive Sprachen
- } **Formale Sprachen**

1. Einführung

ALG(n)

```
1 for  $t = 1$  to  $\infty$ 
2     for  $x = 1$  to  $t$ 
3         for  $y = 1$  to  $t$ 
4             for  $z = 1$  to  $t$ 
5                 if  $x^n + y^n = z^n$ 
6                     return  $(x, y, z)$ 
```

ALG(n) hält bei Eingabe n genau dann, wenn $x^n + y^n = z^n$ für irgendwelche natürlichen Zahlen x, y, z .

(vgl. großen Satz von Fermat)

1. Einführung

Gibt es einen Algorithmus HALTE, der

- als Eingabe einen beliebigen Algorithmus ALG und eine Eingabe w für ALG erhält und
- entscheidet, ob ALG bei Eingabe w hält?

Satz von Turing:

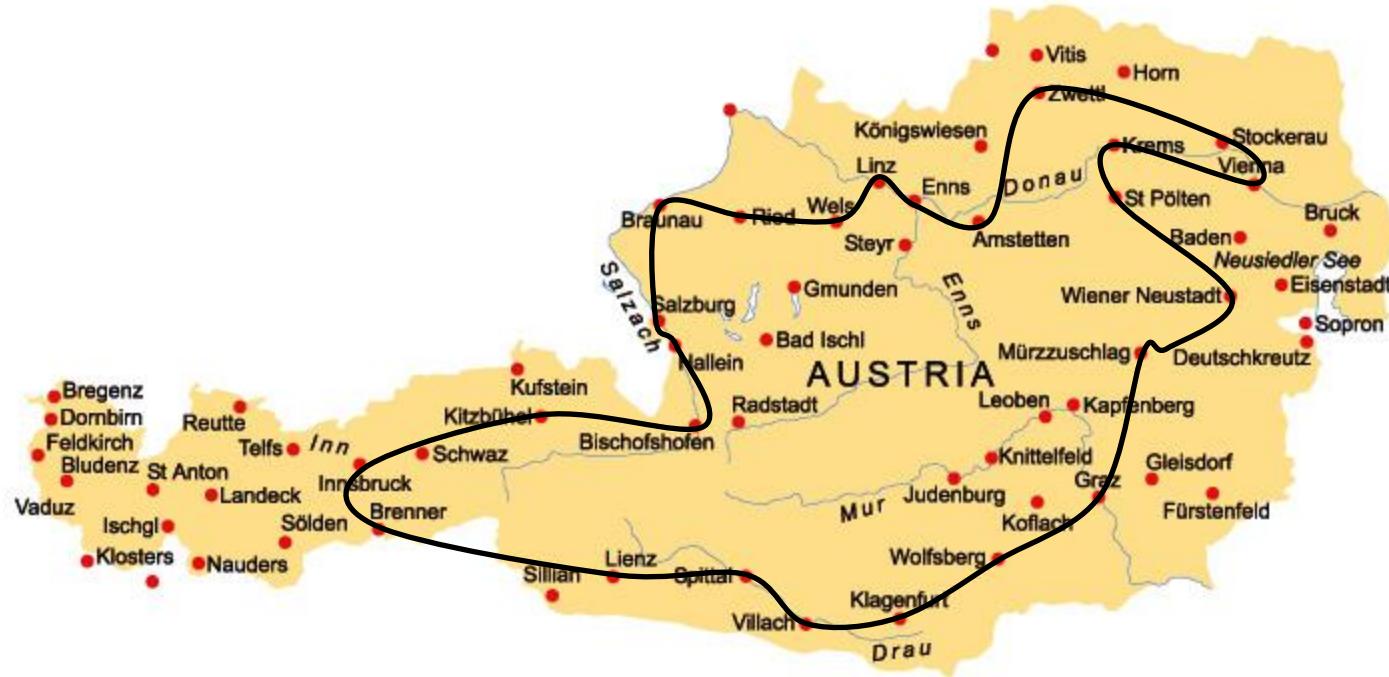
Einen solchen Algorithmus kann es nicht geben.

1. Einführung

- Was sind die wesentlichen Möglichkeiten und Grenzen von Computern?
- **Berechenbarkeit und Komplexität:**
 - Was ist ein Problem?
 - Berechnung einer Funktion, Optimierungsproblem, Entscheiden einer Sprache
 - Multiplikation zweier Matrizen, Kürzeste Wege finden, usw...
 - Wie modelliert man einen Computer?
 - Registermaschinen (RAM), λ -Kalkulus, μ -Rekursion, Turingmaschinen, usw...

1. Einführung

- **Problem des Handlungsreisenden:**
 - Wien, Krems, St. Pölten, Wiener Neustadt, Mürzzuschlag, Graz, Wolfsberg, Klagenfurt, Villach, Spital, Lienz, Brenner, Innsbruck, Kitzbühel, Bischofshofen, Hallein, Salzburg, Braunau, Ried, Wels, Linz, Enns, Amstetten, Zwettl, Stockerau



1. Einführung

- Was sind die wesentlichen Möglichkeiten und Grenzen von Computern?
- **Berechenbarkeit und Komplexität:**
 - Was ist ein Problem?
 - Berechnung einer Funktion, Optimierungsproblem, Entscheiden einer Sprache
 - Multiplikation zweier Matrizen, Kürzeste Wege finden, usw...
 - Wie modelliert man einen Computer?
 - Registermaschinen (RAM), λ -Kalkulus, μ -Rekursion, Turingmaschinen, usw...

1. Einführung

- Alle sinnvollen Rechenmodelle liefern aus unserer Sicht die gleichen Ergebnisse
 - **Churchsche These**
- Es gibt Probleme, die algorithmisch nicht gelöst werden können (z.B. das Halteproblem)
 - **Berechenbarkeit**
- Manche Problem können zwar algorithmisch, aber nicht effizient gelöst werden (z.B. das Problem des Handlungsreisenden)
 - **Komplexität**

1. Einführung

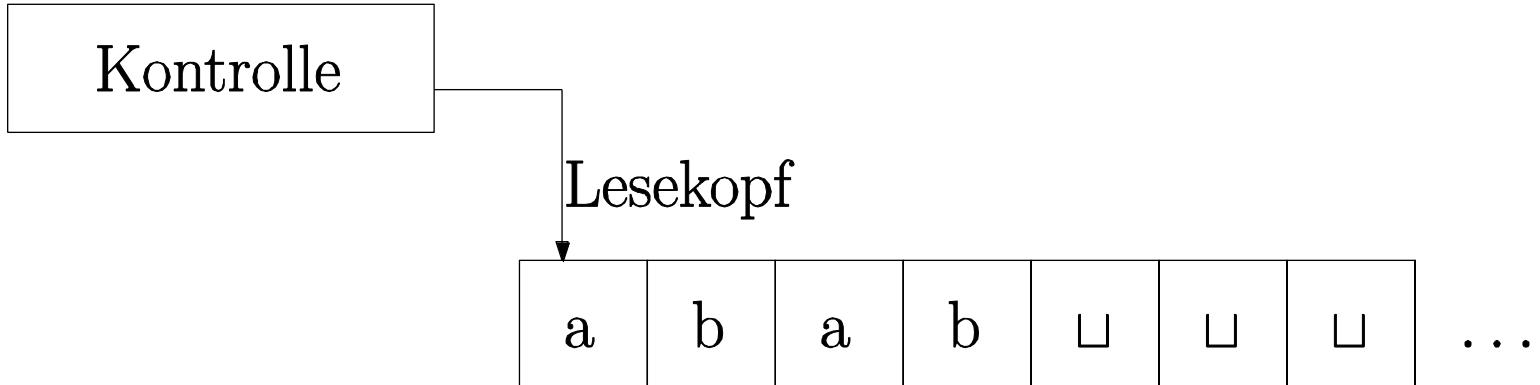
- Wie werden Sprachen beschrieben?
 - Mit Hilfe von Grammatiken – syntaktisch korrekte Java-Programme
- Wie analysiert man Worte?
 - Mit Hilfe der Syntaxanalyse
- Einfachster Fall: Ist x in L ? – Automaten
 - Ist ein Programm syntaktisch korrekt?
- Verschiedene mächtige Grammatiken und Automatentypen:
 - Reguläre Ausdrücke, kontextfreie Grammatiken, kontextsensitive Grammatiken, ...

1. Einführung

- **Turingmaschine**

- Arbeitet auf unbeschränktem Band
- Eingabe steht zu Beginn am Anfang des Bands
- Auf dem Rest des Bandes steht t (Blank)
- Position auf dem Band wird durch den sog. *Lesekopf* beschrieben

Turingmaschine



- Der jeweils nächste Rechenschritt ist eindeutig festgelegt durch den aktuellen Zustand und das aktuell gelesene Zeichen.
- Der Rechenschritt überschreibt das aktuelle Zeichen, bewegt den Kopf nach rechts oder nach links und verändert den Zustand.

Alan Turing



Quelle: rutherfordjournal.org

- Studium in Cambridge
- Entschlüsselung von Enigma-Verschlüsselungen
- Professor in Manchester
- Nach Alan Turing wird der renommierteste Preis in der Informatik benannt
- „The Imitation Game“

1. Einführung

- Teste, ob ein Wort in der folgenden Sprache liegt:
 - $L = \{w\#w \mid w \text{ in } \{0,1\}^*\}$
- **Vorgehensweise:**
 - Von links nach rechts über das Wort laufen
 - Erstes Zeichen links merken und markieren
 - Erstes Zeichen rechts von # vergleichen und markieren
 - Für alle Zeichen wiederholen bis Blank erreicht
 - Rechts dürfen dann nur noch Blanks folgen
 - Falls Zeichen an einer Stelle nicht übereinstimmen ablehnen, sonst am Ende akzeptieren

1. Einführung

0 1 1 0 0 0 # 0 1 1 0 0 0 t ...

x 1 1 0 0 0 # 0 1 1 0 0 0 t ...

x 1 1 0 0 0 # x 1 1 0 0 0 t ...

x 1 1 0 0 0 # x 1 1 0 0 0 t ...

x x 1 0 0 0 # x 1 1 0 0 0 t ...

x x x x x x # x x x x x x t ... accept

1. Einführung

Definition

Eine (*deterministische 1-Band Turingmaschine*) DTM wird beschrieben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Dabei sind Q, Σ, Γ endliche, nichtleere Mengen und es gilt:

- Σ ist Teilmenge von Γ
- t in $\Gamma \setminus \Sigma$ ist das *Blanksymbol* (auch \sqcup)
- Q ist die *Zustandsmenge*
- Σ ist das *Eingabealphabet*
- Γ ist das *Bandalphabet*
- q_0 in Q ist der *Startzustand*
- q_{accept} in Q ist der akzeptierende Endzustand
- q_{reject} in Q ist der ablehnende Endzustand
- $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die (partielle) *Übergangsfunktion*. Sie ist für kein Argument aus $\{q_{accept}, q_{reject}\} \times \Gamma$ definiert.

1. Einführung

- Initial:
 - Eingabe steht links auf dem Band
 - Der Rest des Bands ist leer
 - Kopf befindet sich ganz links
- Berechnungen finden entsprechend der Übergangsfunktion statt
- Wenn der Kopf sich am linken Ende befindet und nach links bewegen soll, bleibt er an seiner Position
- Wenn q_{accept} oder q_{reject} erreicht wird, ist die Bearbeitung beendet

1. Einführung

Momentaufnahme einer Turingmaschine:

- Bei Bandinschrift uv (dabei beginnt u am linken Ende des Bandes und hinter v stehen nur Blanks)
- Zustand q
- Kopf auf erstem Zeichen von v

Konfiguration $C = uqv$

1. Einführung

- Gegeben: Konfigurationen C_1, C_2
- Wir sagen: **Konfiguration C_1 führt zu C_2** , falls die TM von C_1 in einem Schritt zu C_2 übergehen kann

Formal:

- Seien a, b, c in Γ , u, v in Γ^* und Zustände q_i, q_j gegeben
- Wir sagen:
 - $uaq_i bv$ führt zu $uq_j acv$, falls $\delta(q_i, b) = (q_j, c, L)$ und
 - $uaq_i bv$ führt zu $uacq_j v$, falls $\delta(q_i, b) = (q_j, c, R)$

1. Einführung

- Startkonfiguration:
 - $q_0 w$, wobei w die Eingabe ist
- Akzeptierende Konfiguration:
 - Konfigurationen mit Zustand q_{accept}
- Ablehnende Konfiguration:
 - Konfigurationen mit Zustand q_{reject}
- Haltende Konfiguration:
 - akzeptierende oder ablehnende Konfigurationen

1. Einführung

Definition

Eine Turingmaschine M akzeptiert eine Eingabe w , falls es eine Folge von Konfigurationen C_1, C_2, \dots, C_k gibt, sodass

1. C_1 ist die Startkonfiguration von M bei Eingabe w
2. C_i führt zu C_{i+1}
3. C_k ist eine akzeptierende Konfiguration

- Die von M akzeptierten Worte bilden die von M akzeptierte Sprache $L(M)$.
- Eine Turingmaschine M entscheidet die Sprache $L(M)$, wenn jede Eingabe in einer haltenden Konfiguration C_k resultiert.

1. Einführung

Definition

- Eine Sprache L heißt **rekursiv aufzählbar**, falls es eine Turingmaschine M gibt, die L akzeptiert.
- Eine Sprache L heißt **rekursiv** oder **entscheidbar**, falls es eine Turingmaschine M gibt, die L entscheidet.

1. Einführung

Gesucht: Turingmaschine, die $L = \{0^{2^n} \mid n \geq 0\}$ entscheidet

Arbeitsweise:

1. Gehe von links nach rechts über die Eingabe und ersetze jede zweite 0 durch x
2. Wenn nur eine 0 auf dem Band ist, akzeptiere
3. Falls die Anzahl der 0en ungerade ist, lehne ab
4. Bewege den Kopf zurück an das linke Ende

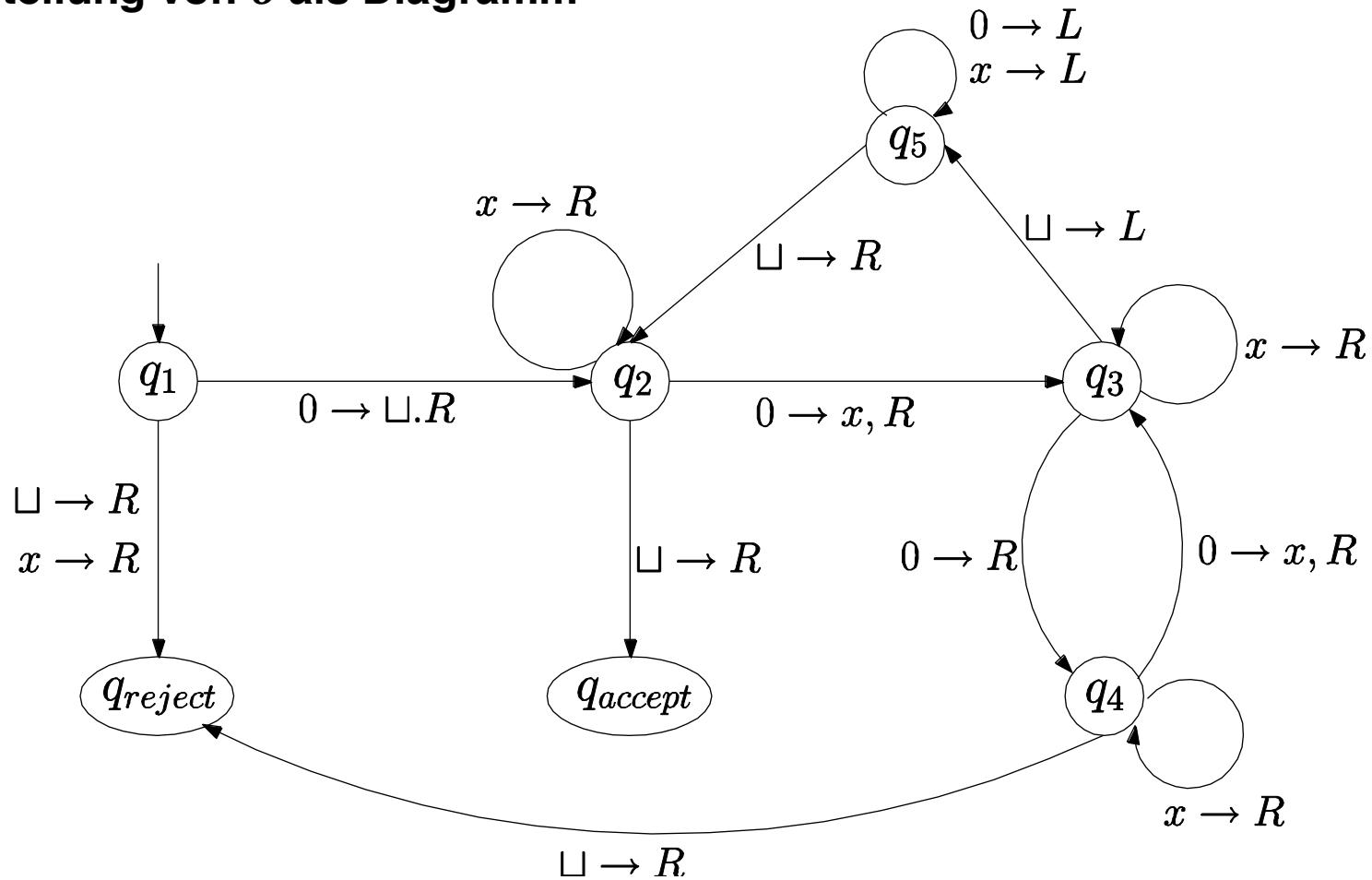
1. Einführung

Definition der Turingmaschine

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, t\}$
- q_1 : Startzustand
- q_{accept} : Akzeptierender Endzustand
- q_{reject} : Ablehnender Endzustand
- δ : Übergangsfunktion

1. Einführung

Darstellung von δ als Diagramm

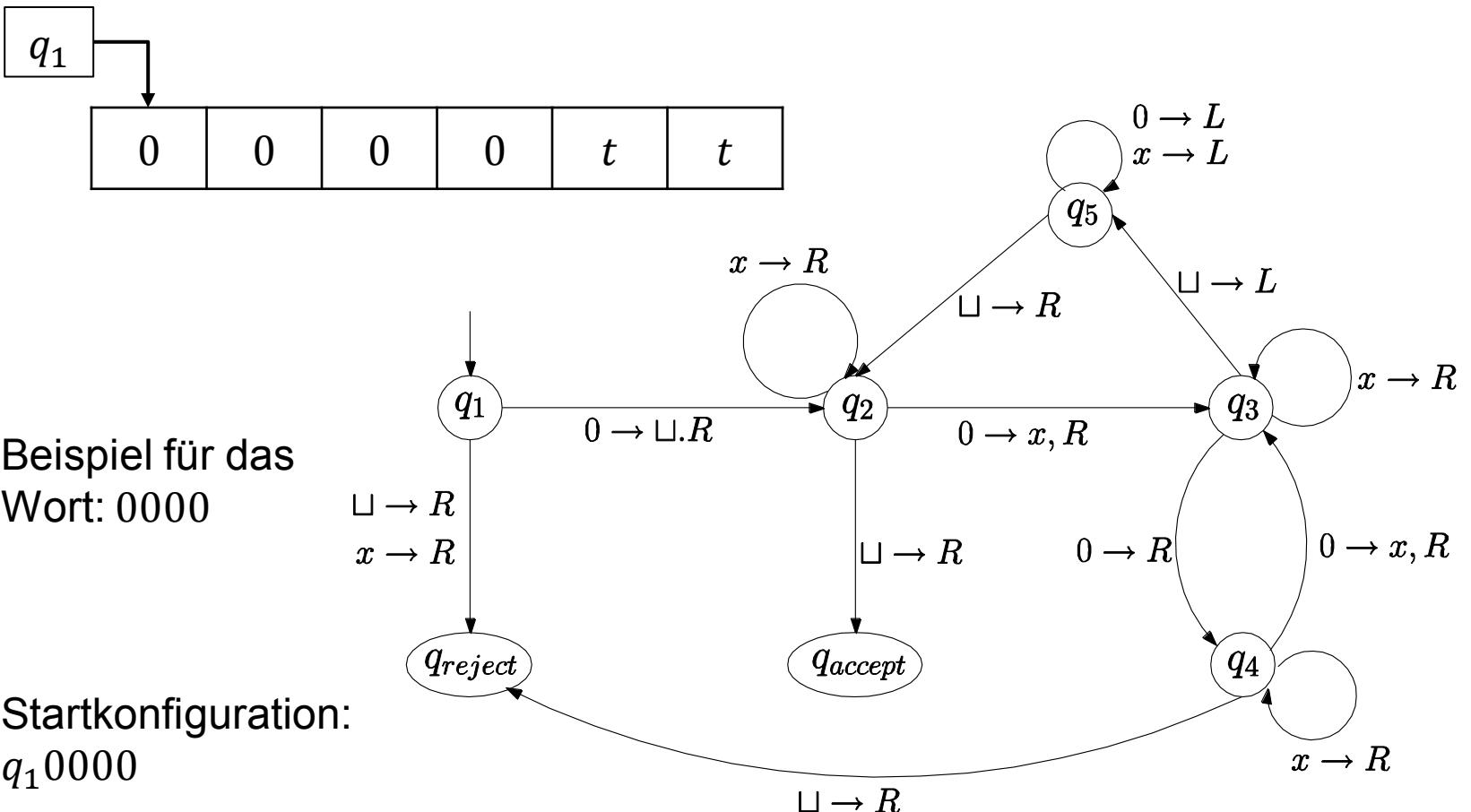


1. Einführung

Darstellung von δ als Tabelle

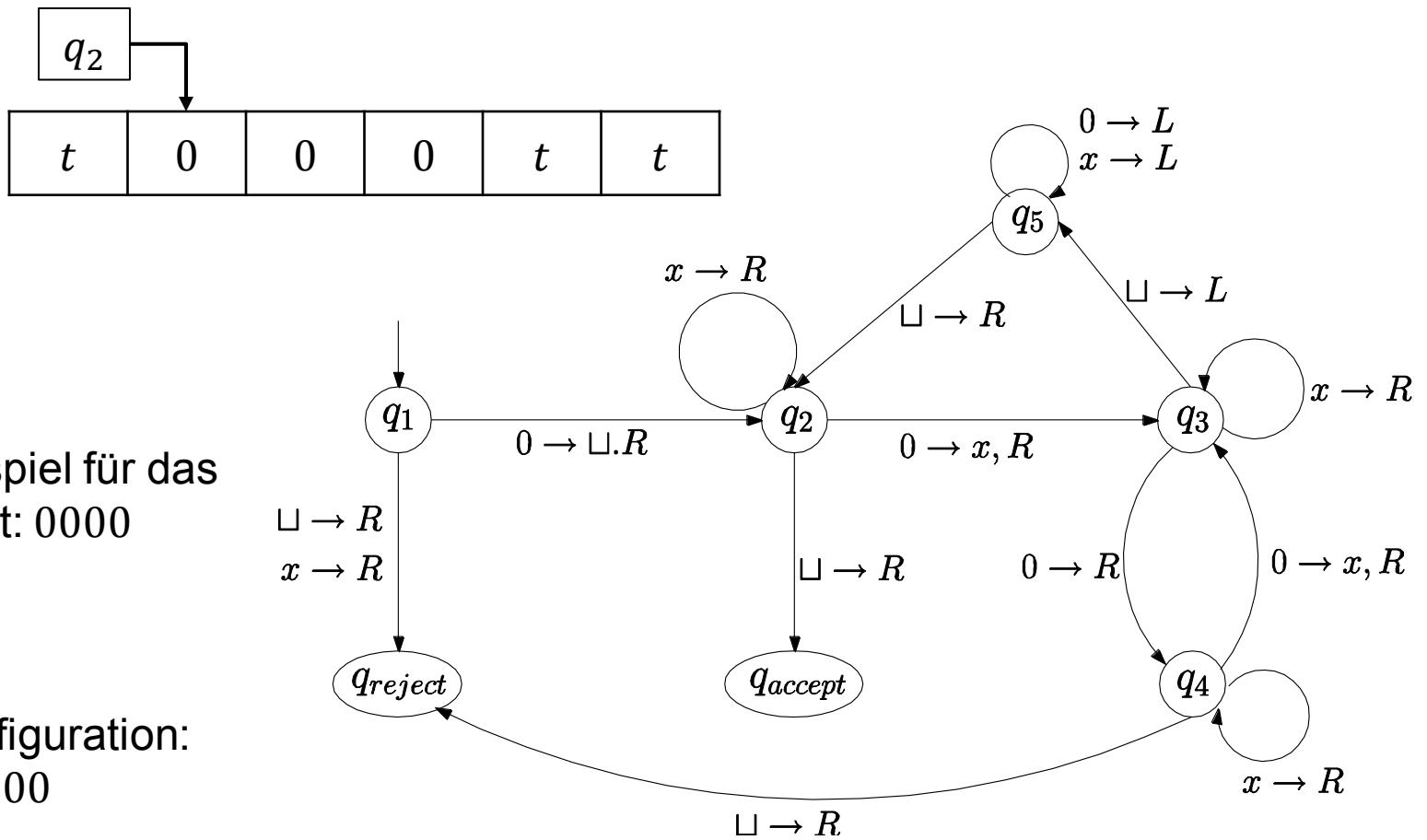
δ	0	x	t
q_1	(q_2, t, R)	(q_{reject}, x, R)	(q_{reject}, t, R)
q_2	(q_3, x, R)	(q_2, x, R)	(q_{accept}, t, R)
q_3	$(q_4, 0, R)$	(q_3, x, R)	(q_5, t, L)
q_4	(q_3, x, R)	(q_4, x, R)	(q_{reject}, t, R)
q_5	$(q_5, 0, L)$	(q_5, x, L)	(q_2, t, R)

1. Einführung



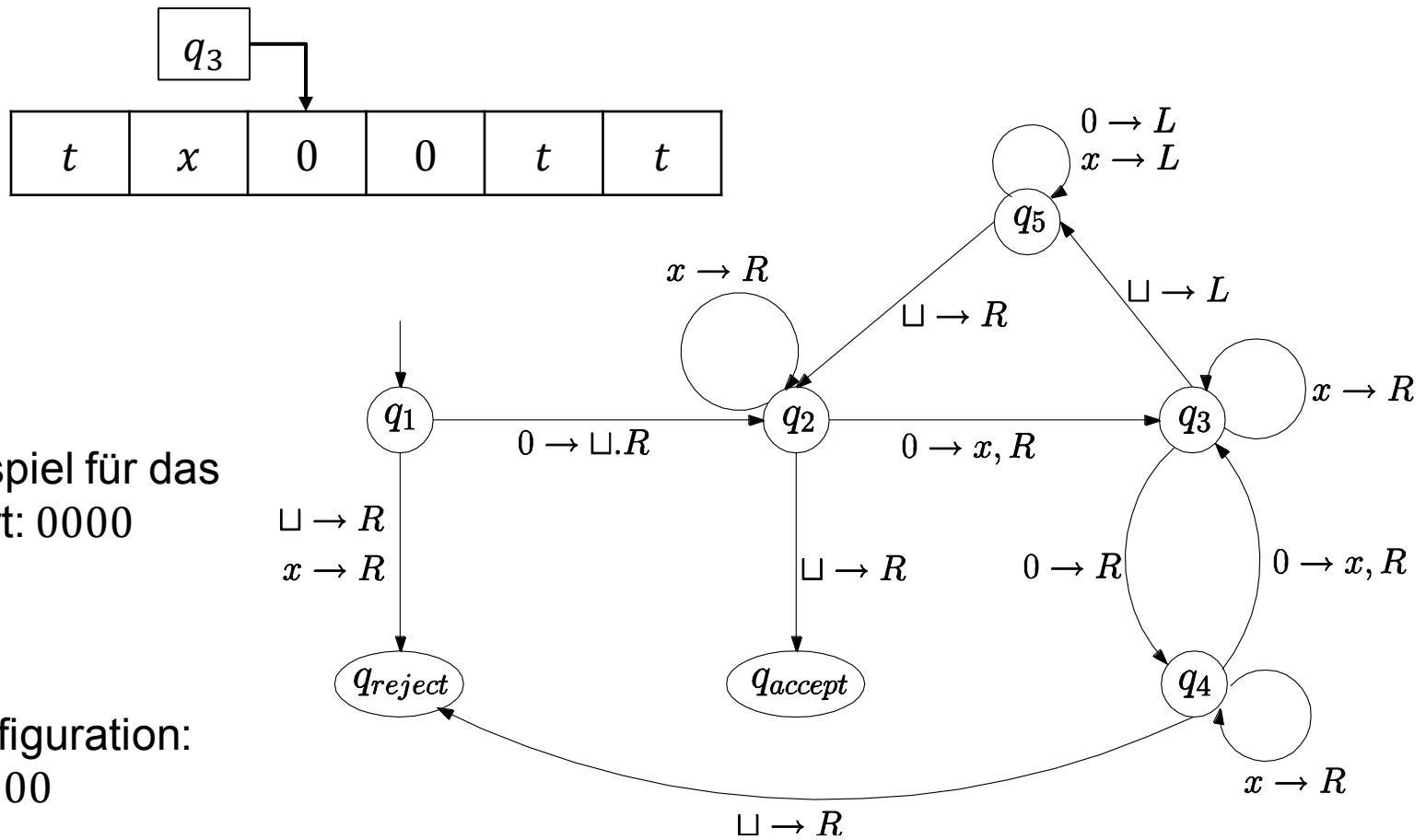
1. Einführung

- Beispiel für das Wort: 0000
- Konfiguration:
 tq_2000



1. Einführung

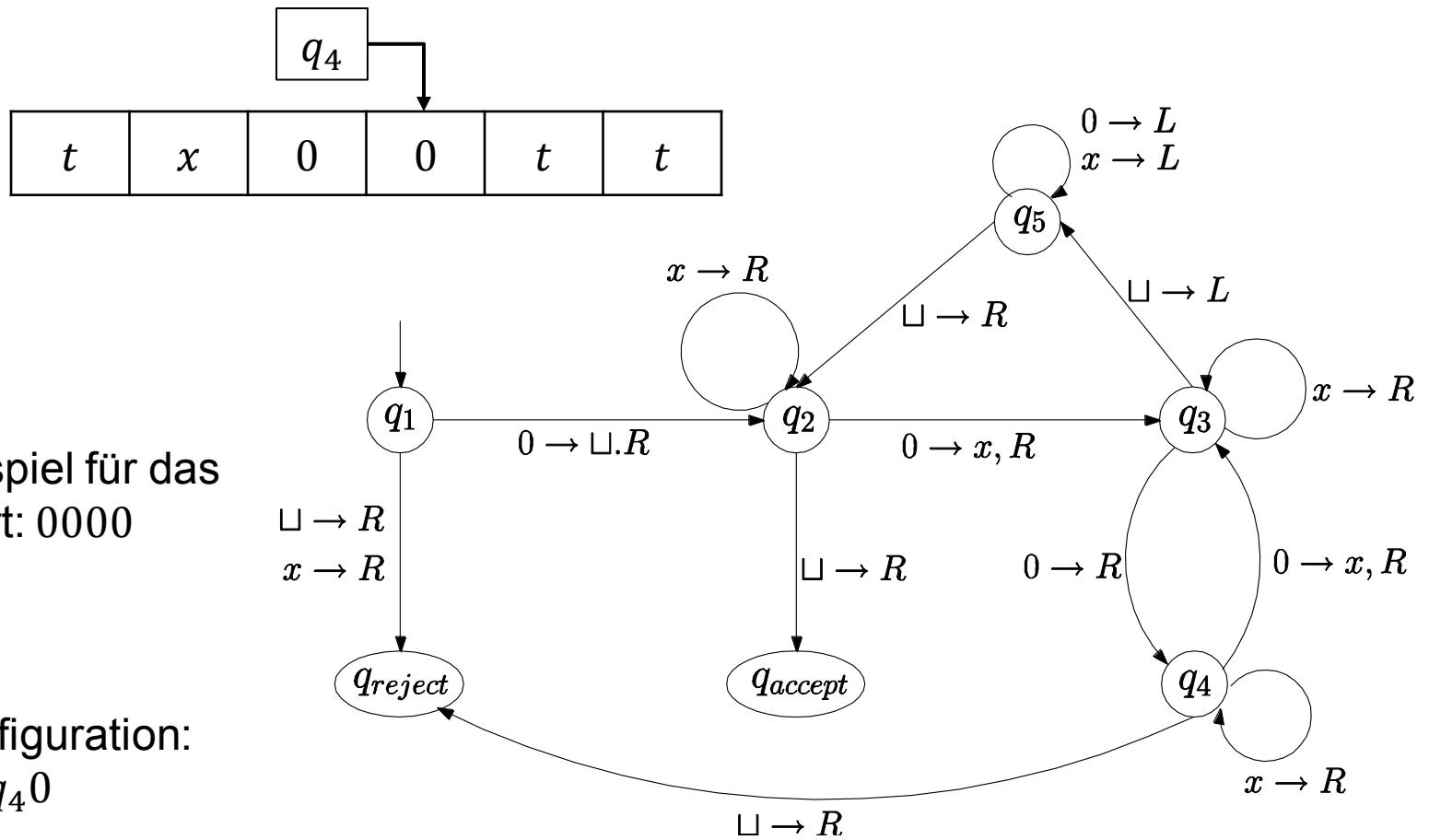
- Beispiel für das Wort: 0000
- Konfiguration:
 txq_300



1. Einführung

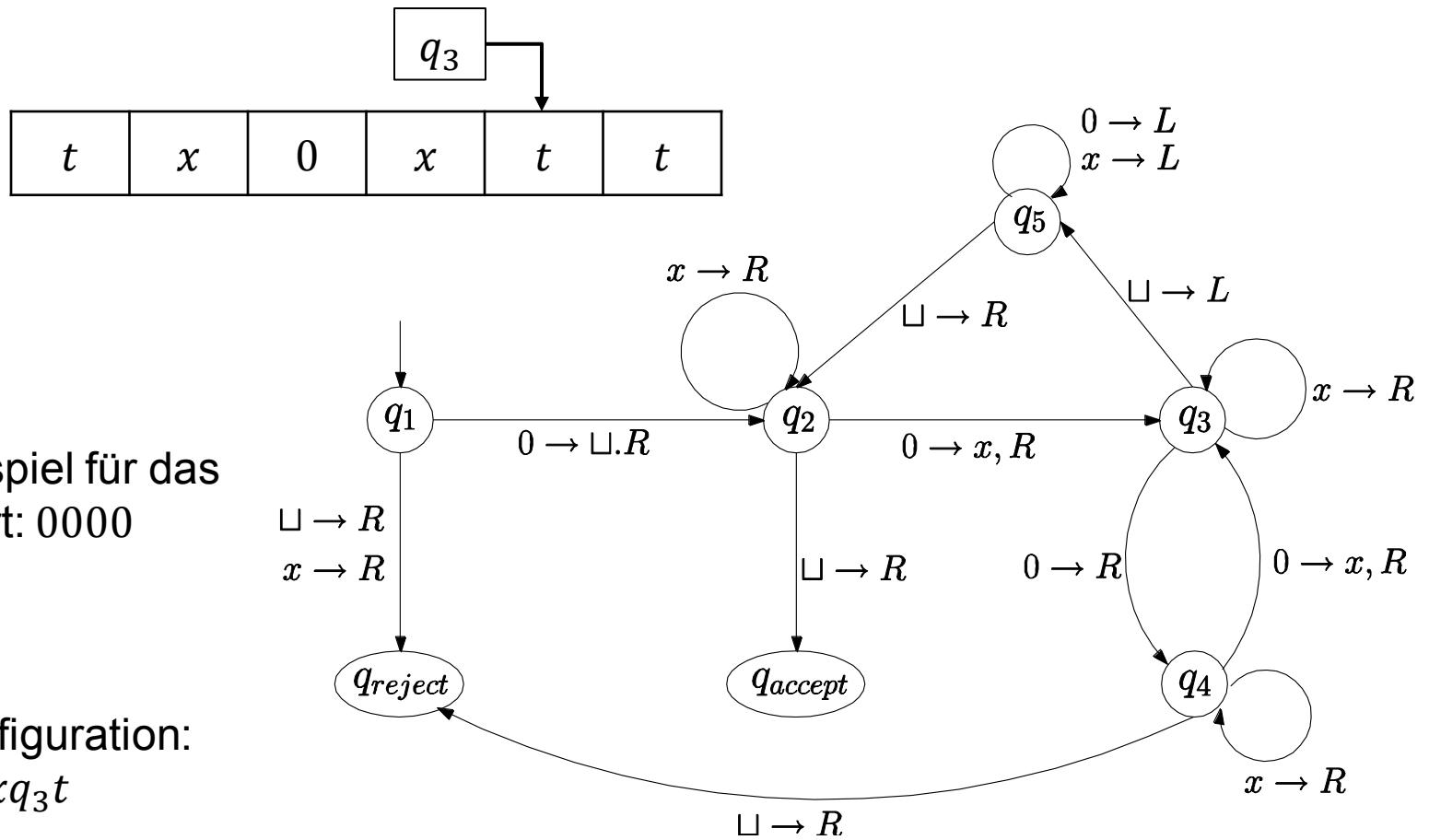
- Beispiel für das Wort: 0000

- Konfiguration:
 $tx0q_40$



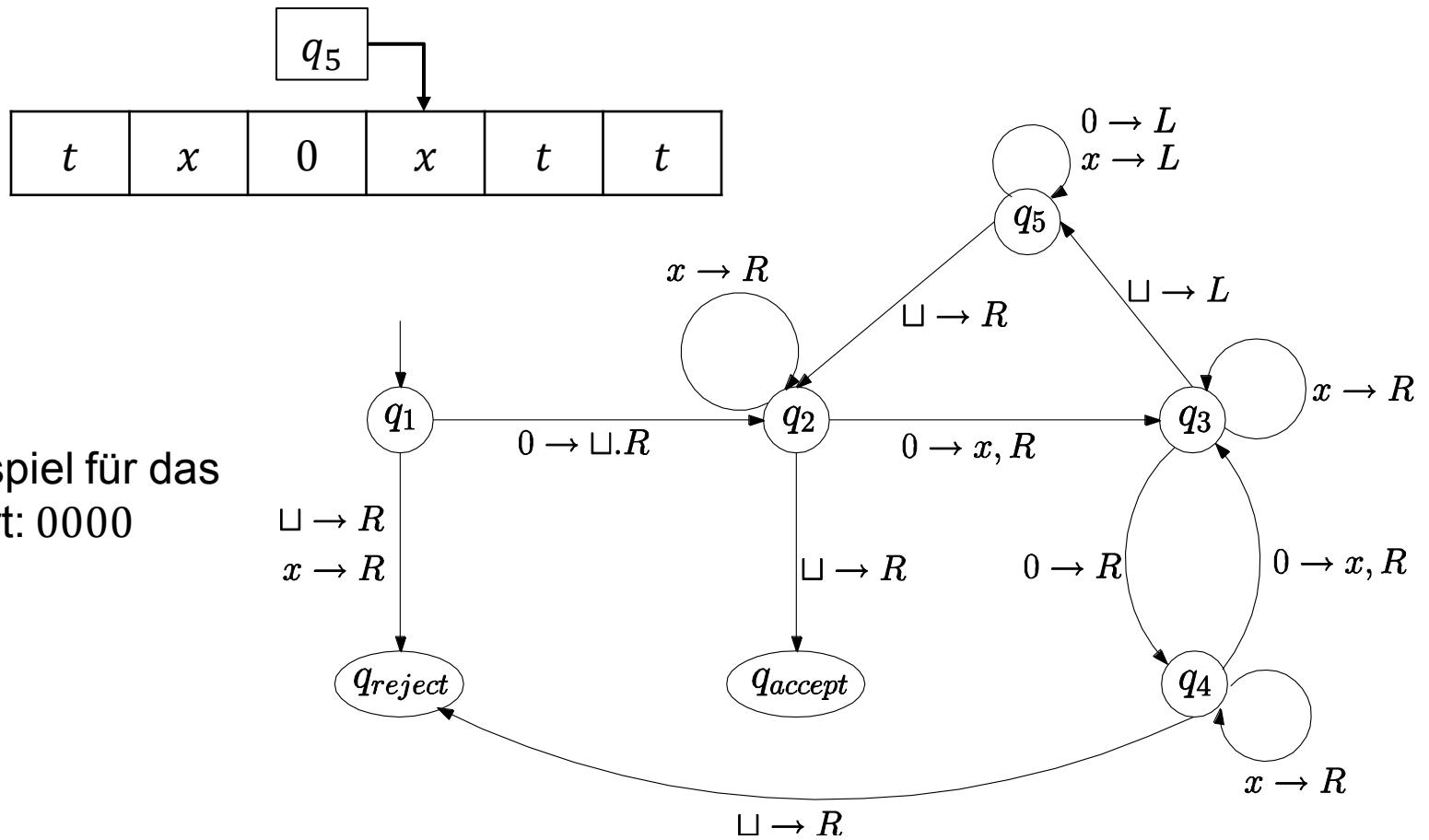
1. Einführung

- Beispiel für das Wort: 0000
- Konfiguration:
 $tx0xq_3t$



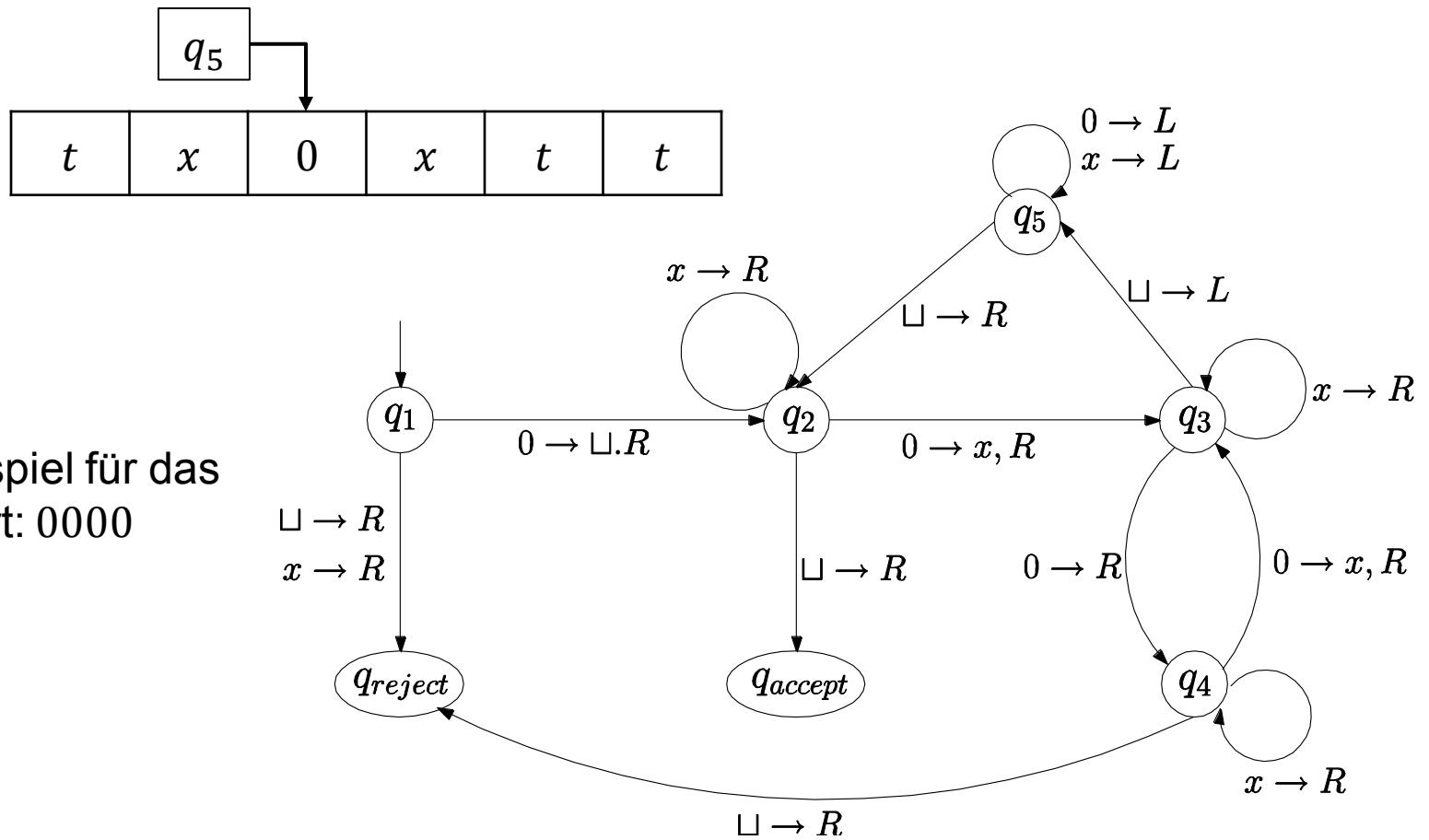
1. Einführung

- Beispiel für das Wort: 0000

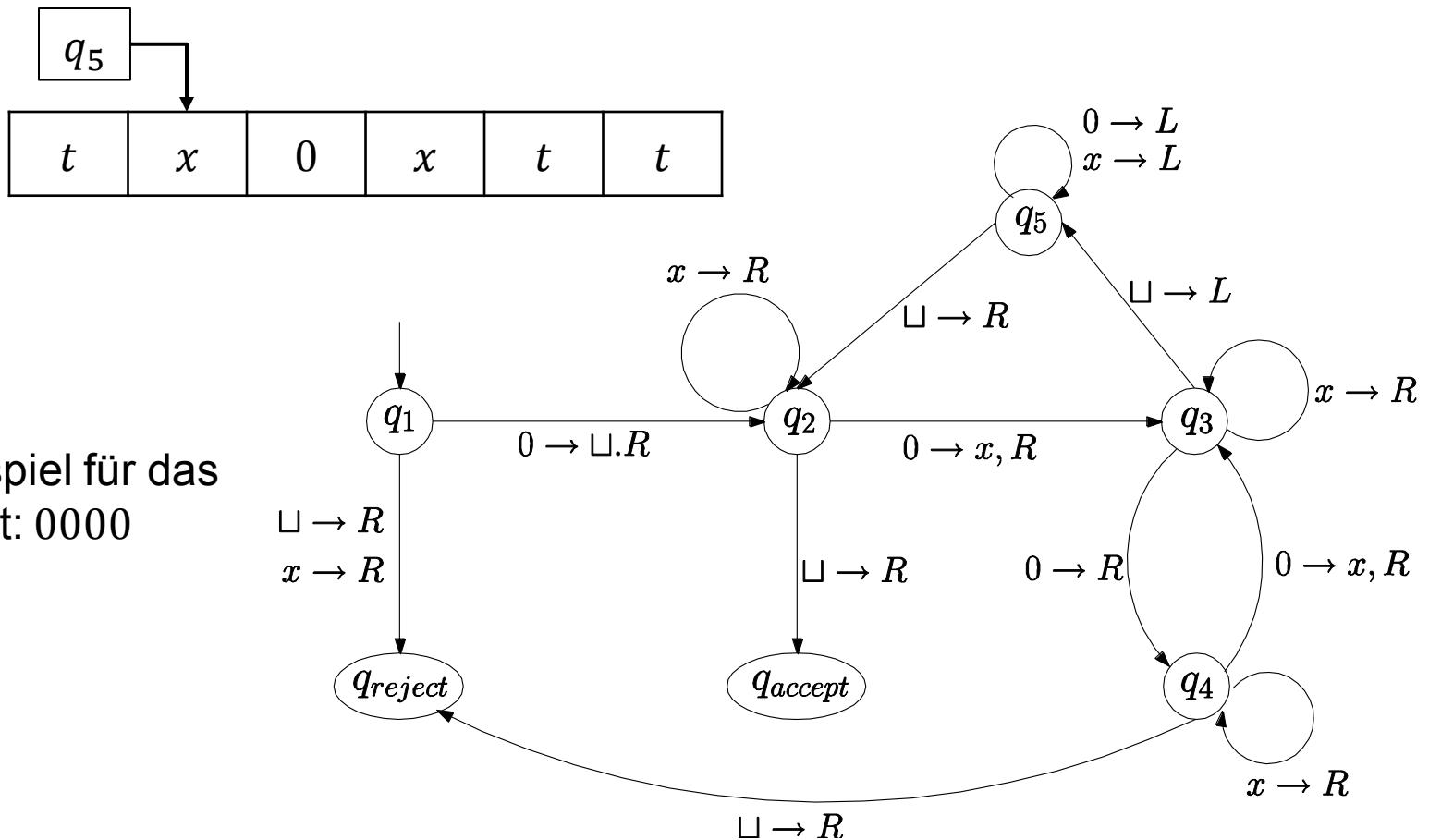


1. Einführung

- Beispiel für das Wort: 0000

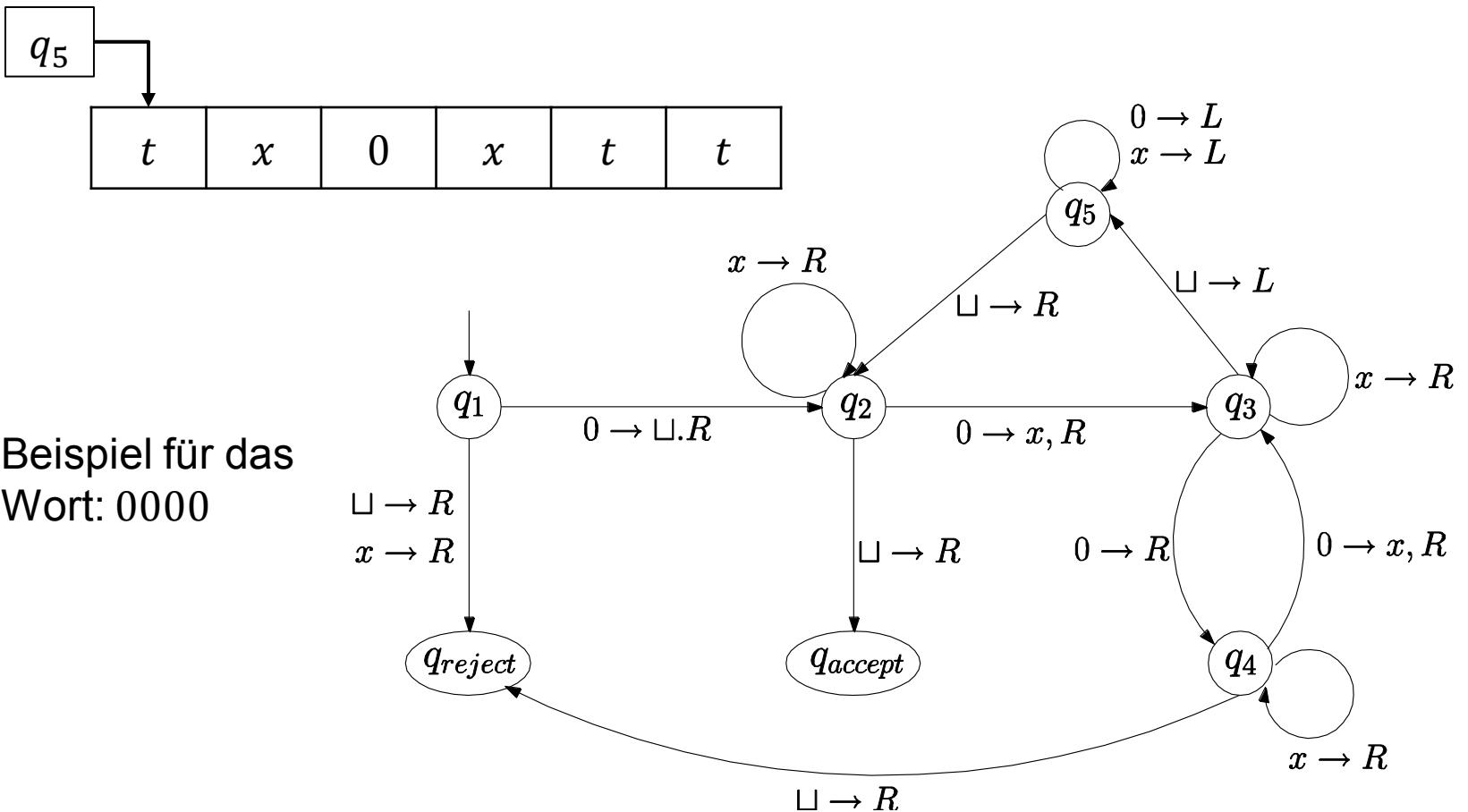


1. Einführung



- Beispiel für das Wort: 0000

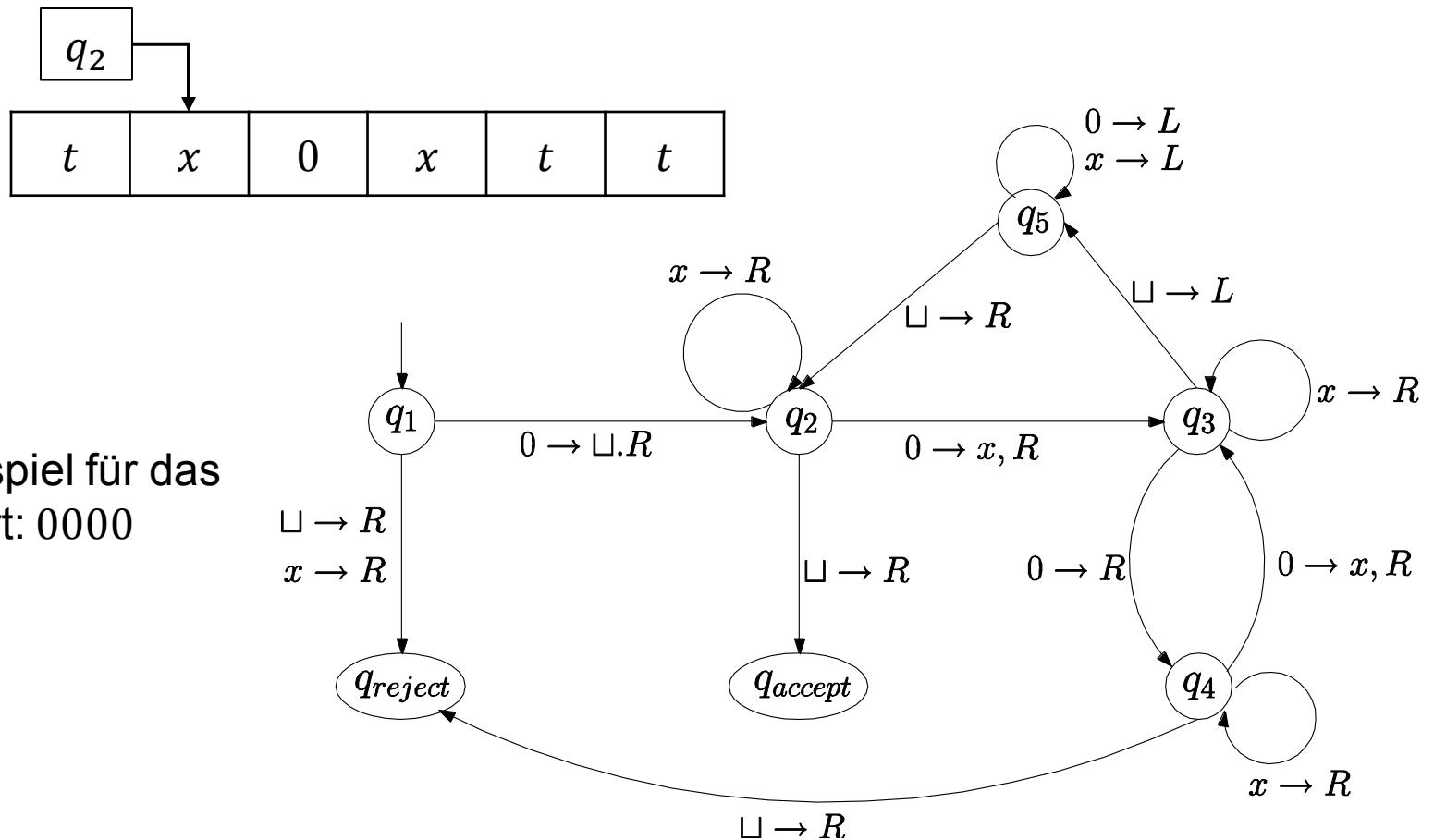
1. Einführung



- Beispiel für das Wort: 0000

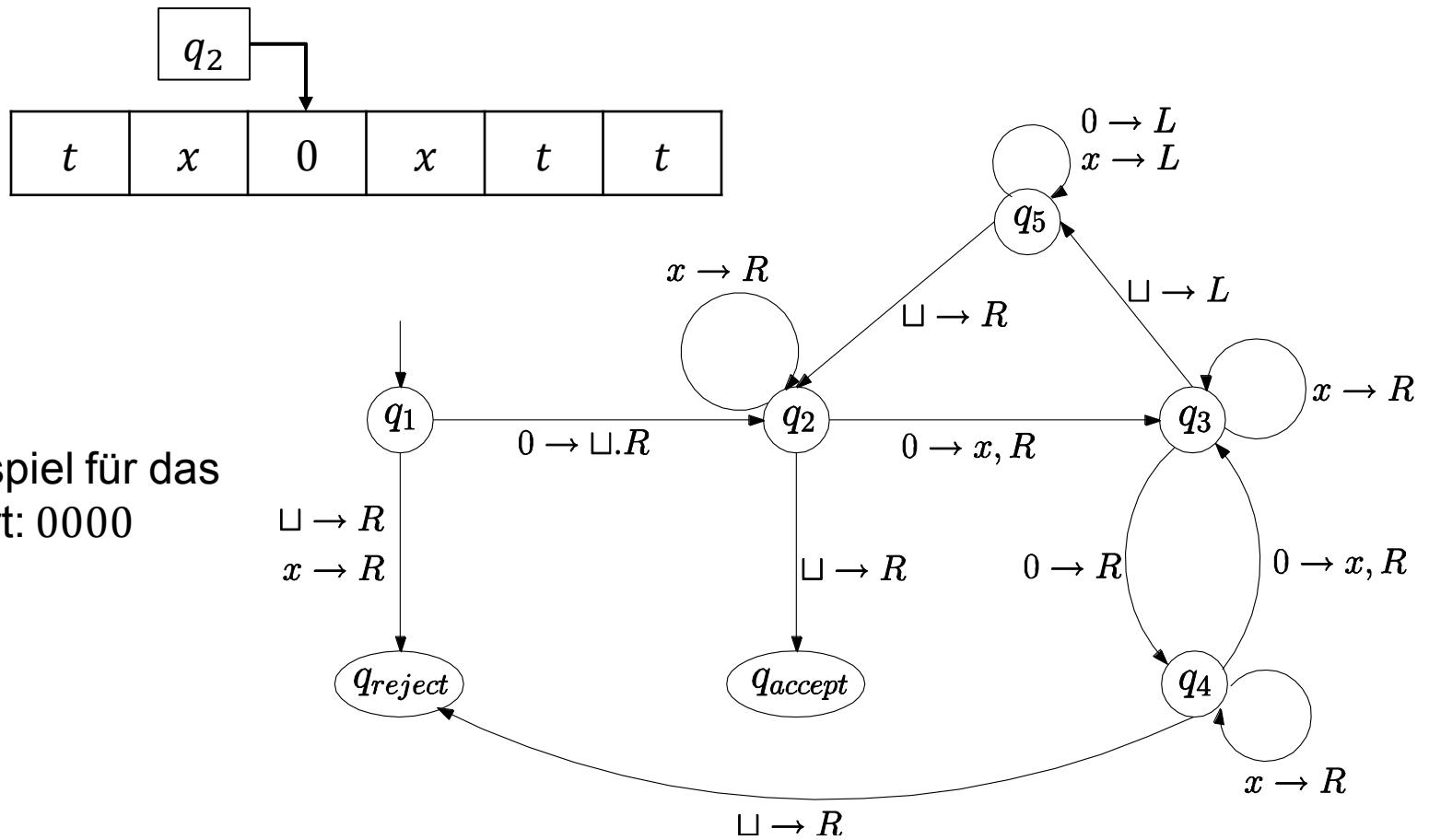
1. Einführung

- Beispiel für das Wort: 0000



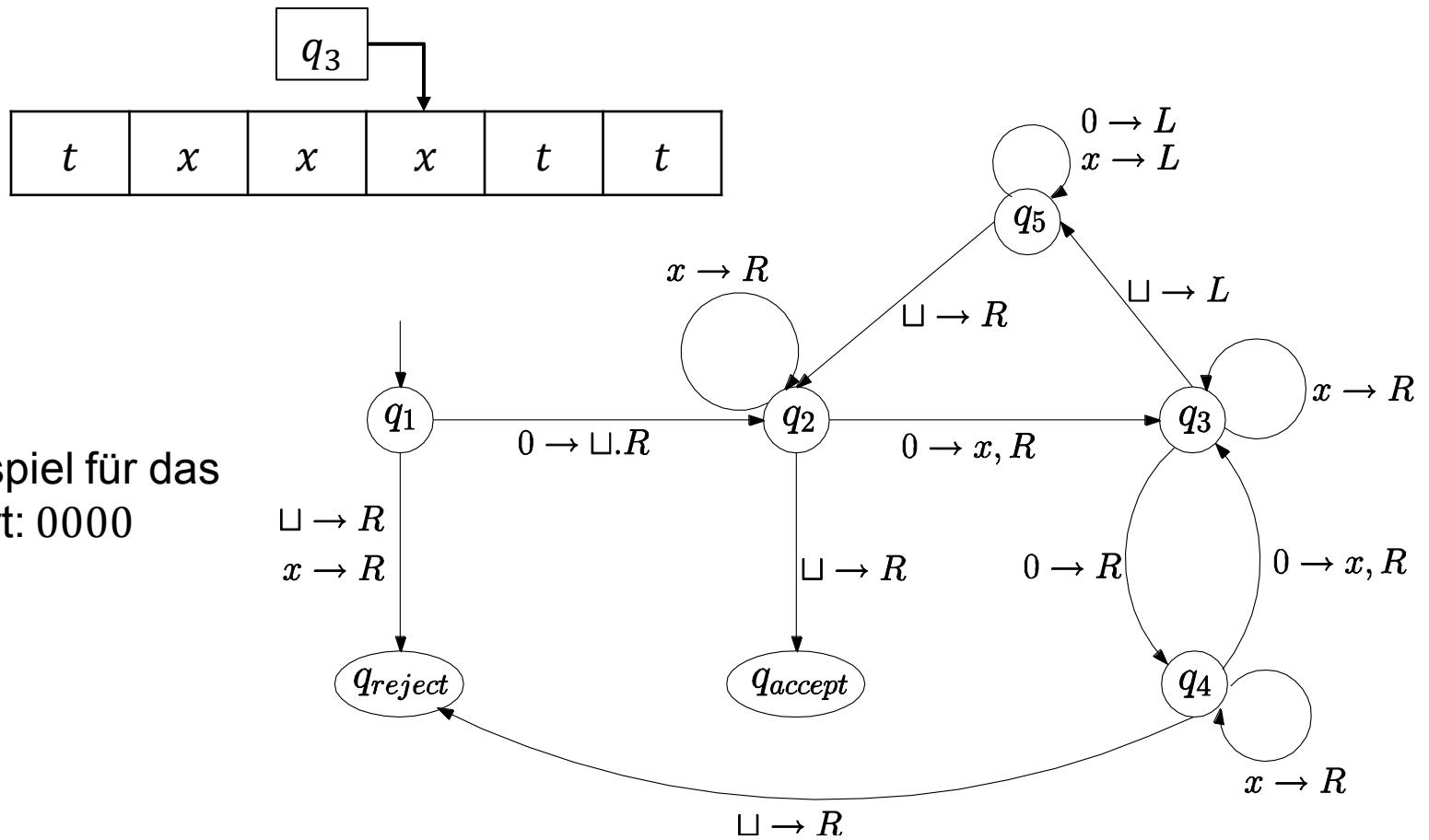
1. Einführung

- Beispiel für das Wort: 0000



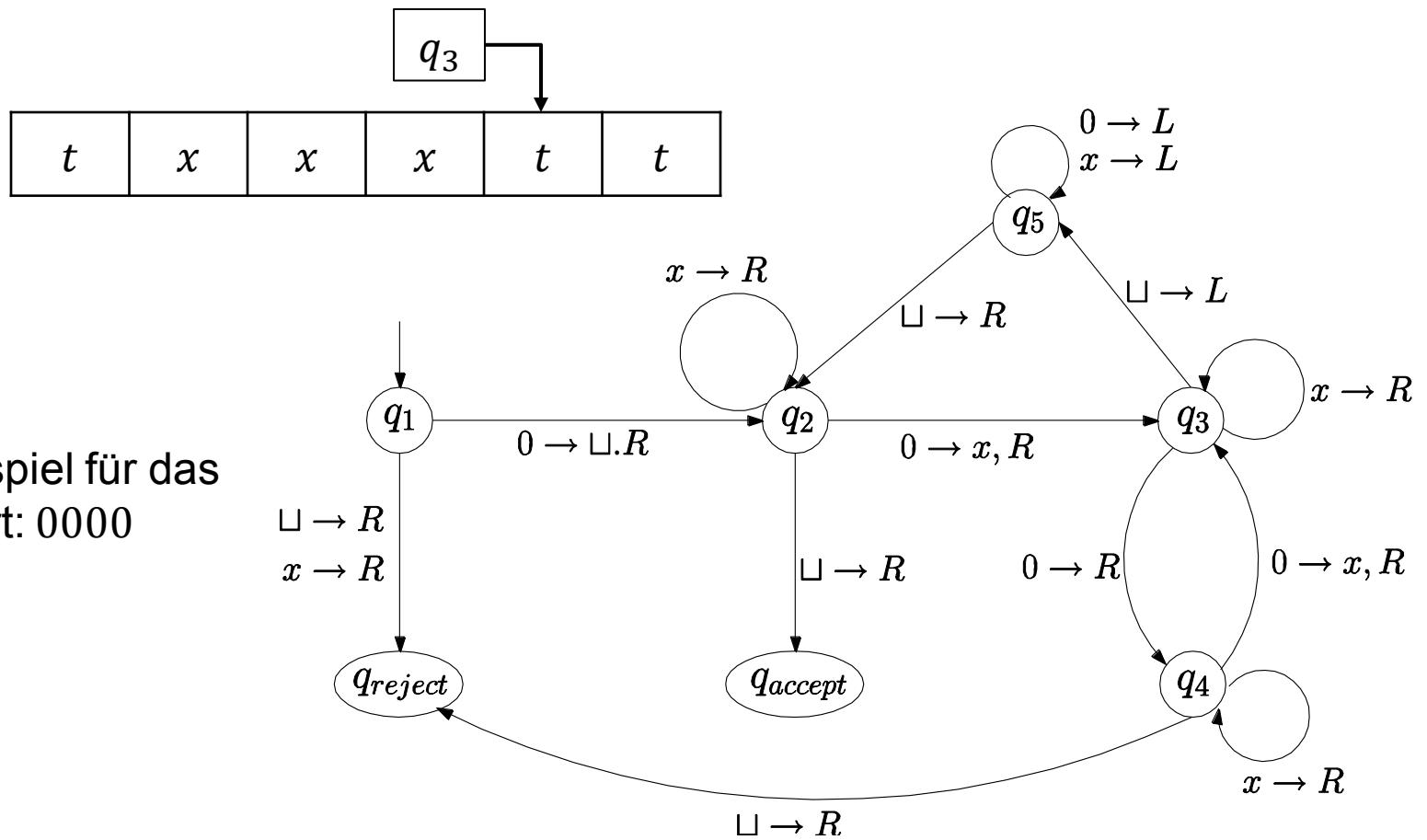
1. Einführung

- Beispiel für das Wort: 0000



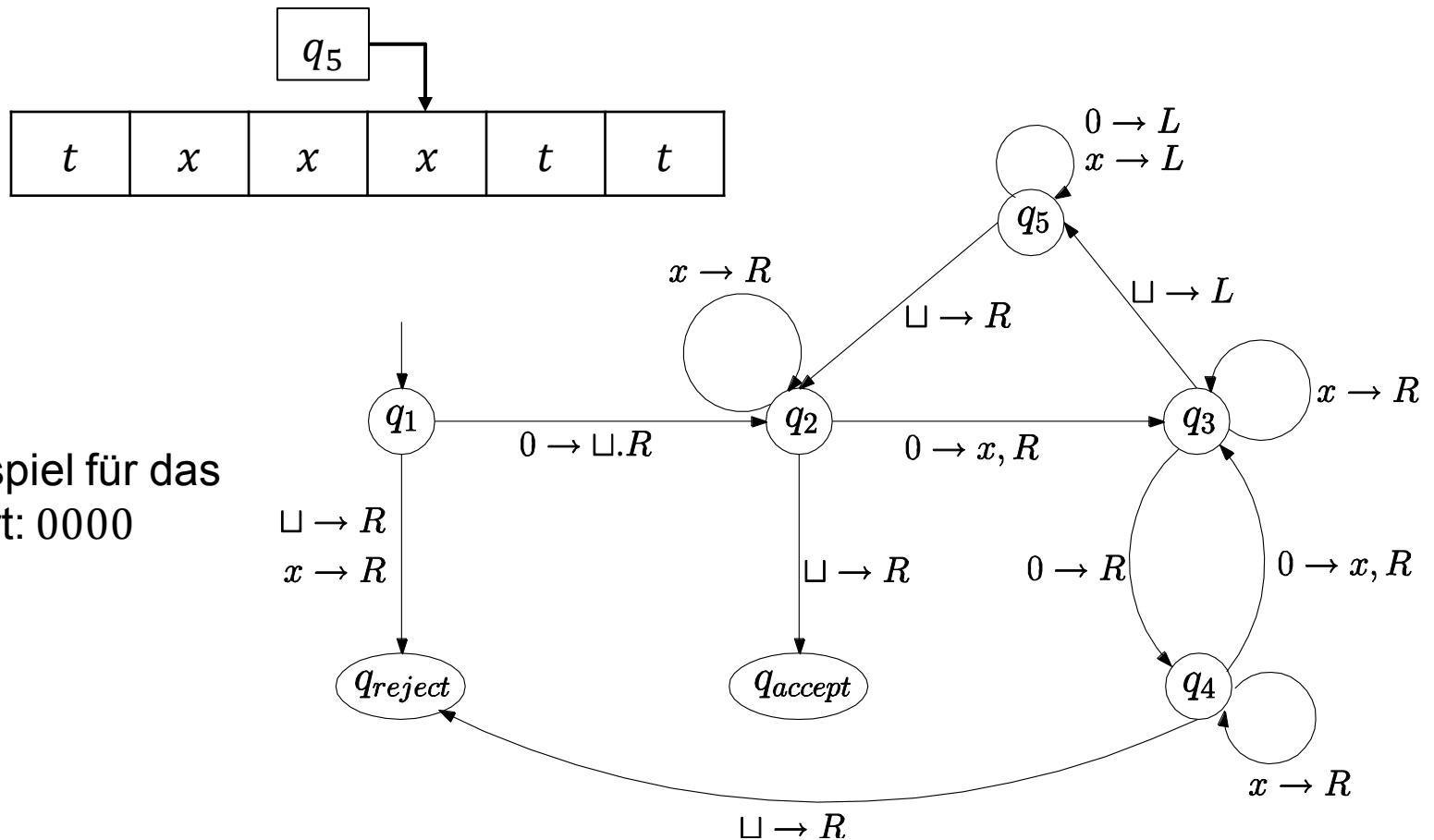
1. Einführung

- Beispiel für das Wort: 0000



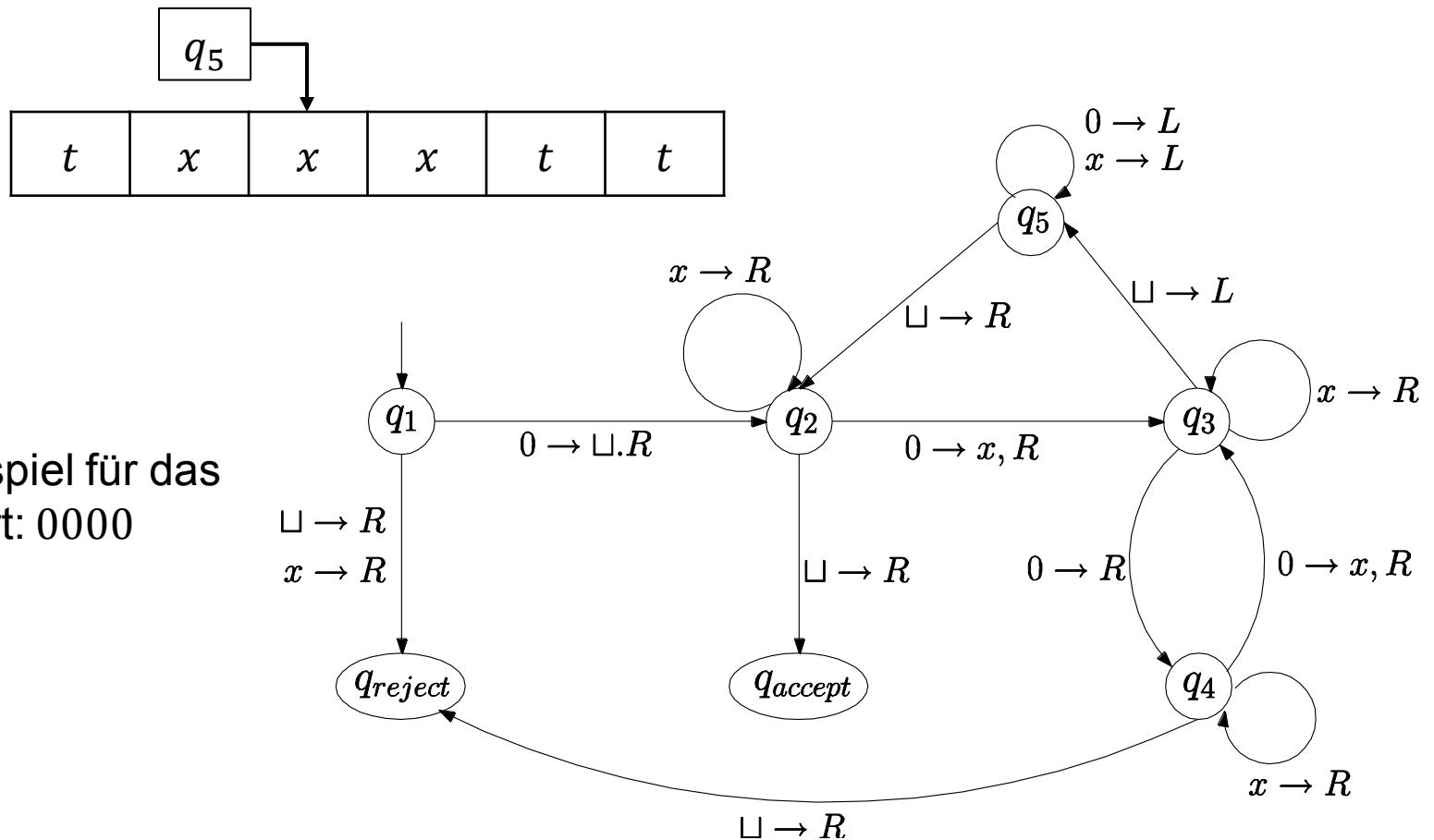
1. Einführung

- Beispiel für das Wort: 0000



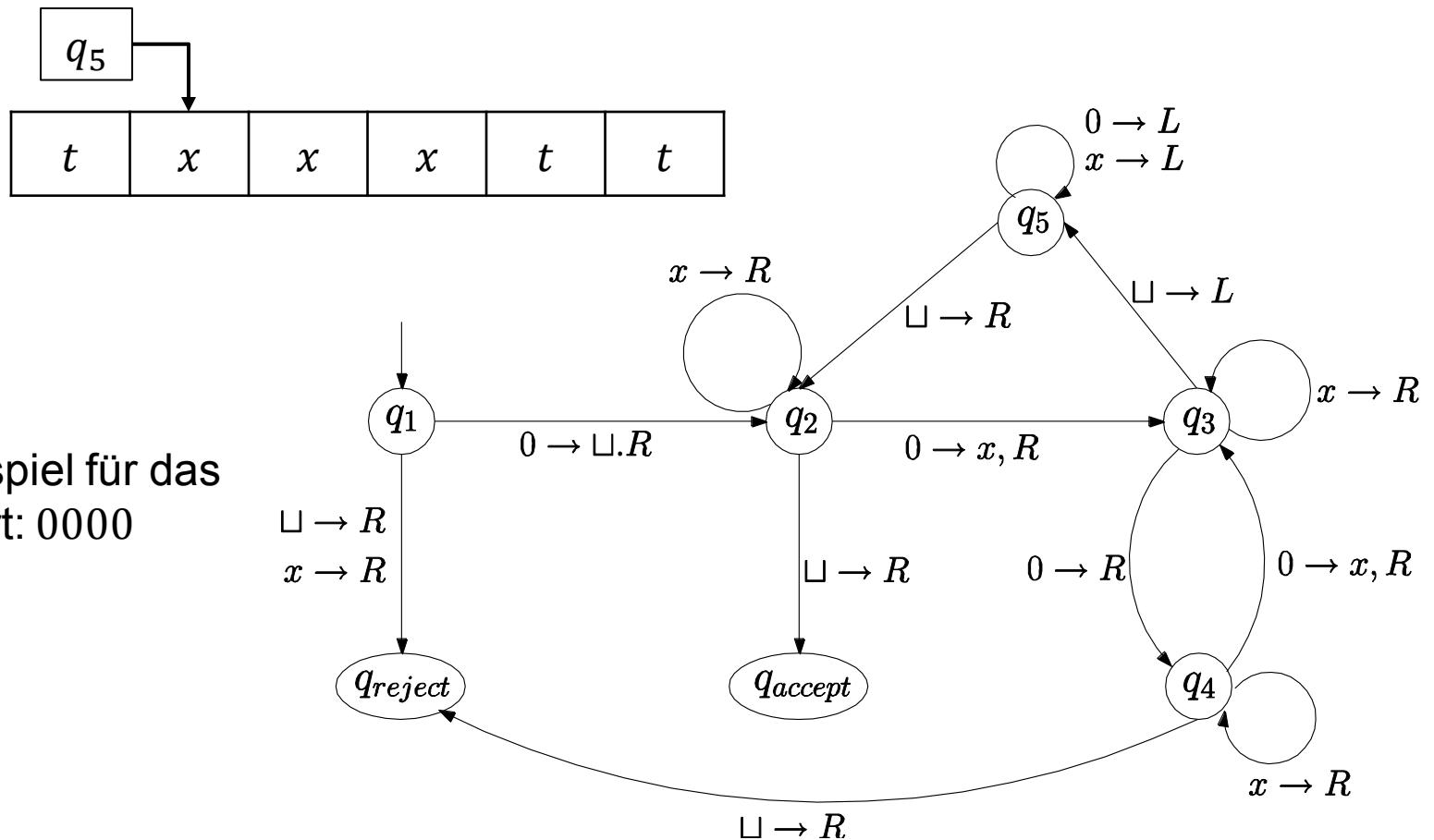
1. Einführung

- Beispiel für das Wort: 0000

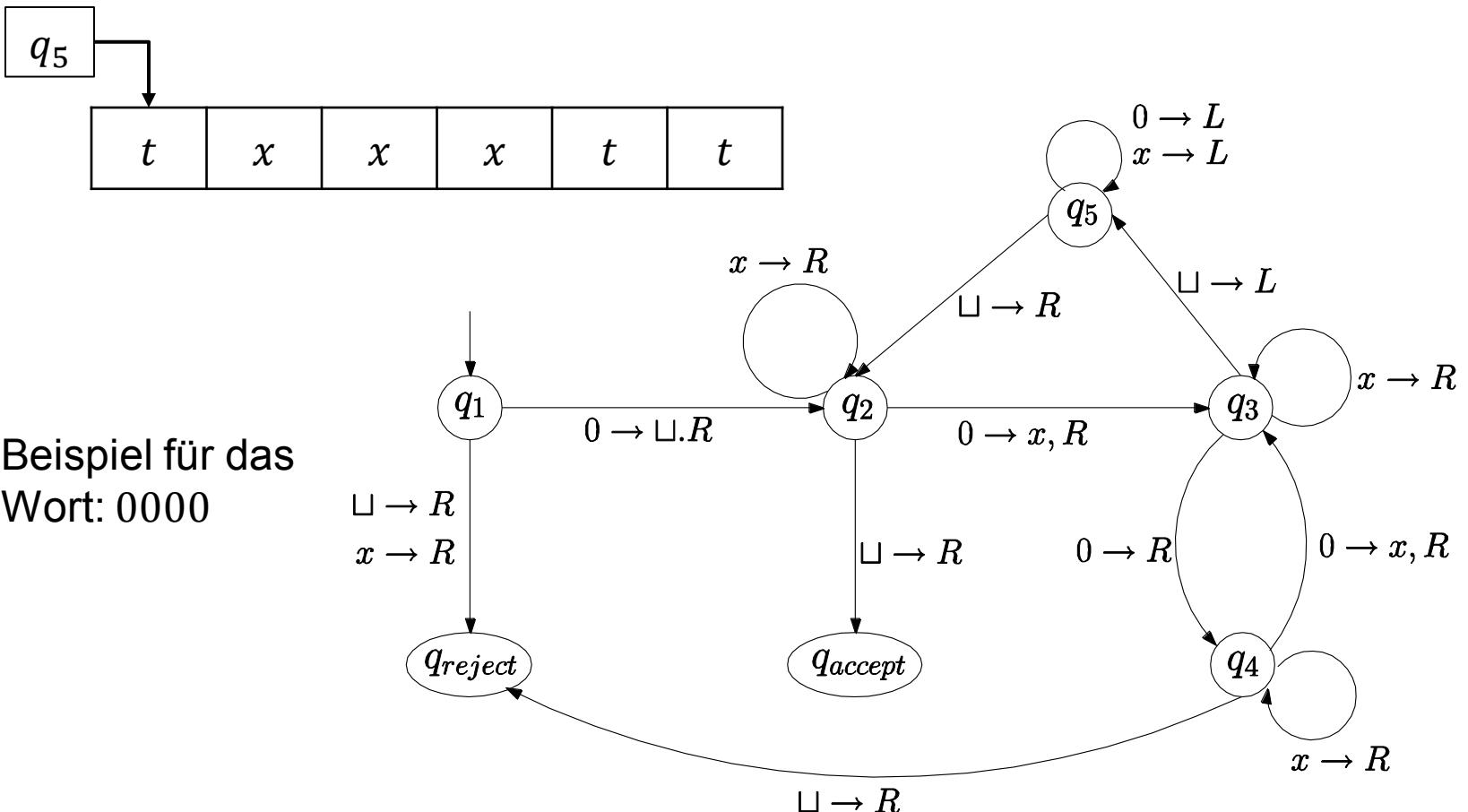


1. Einführung

- Beispiel für das Wort: 0000

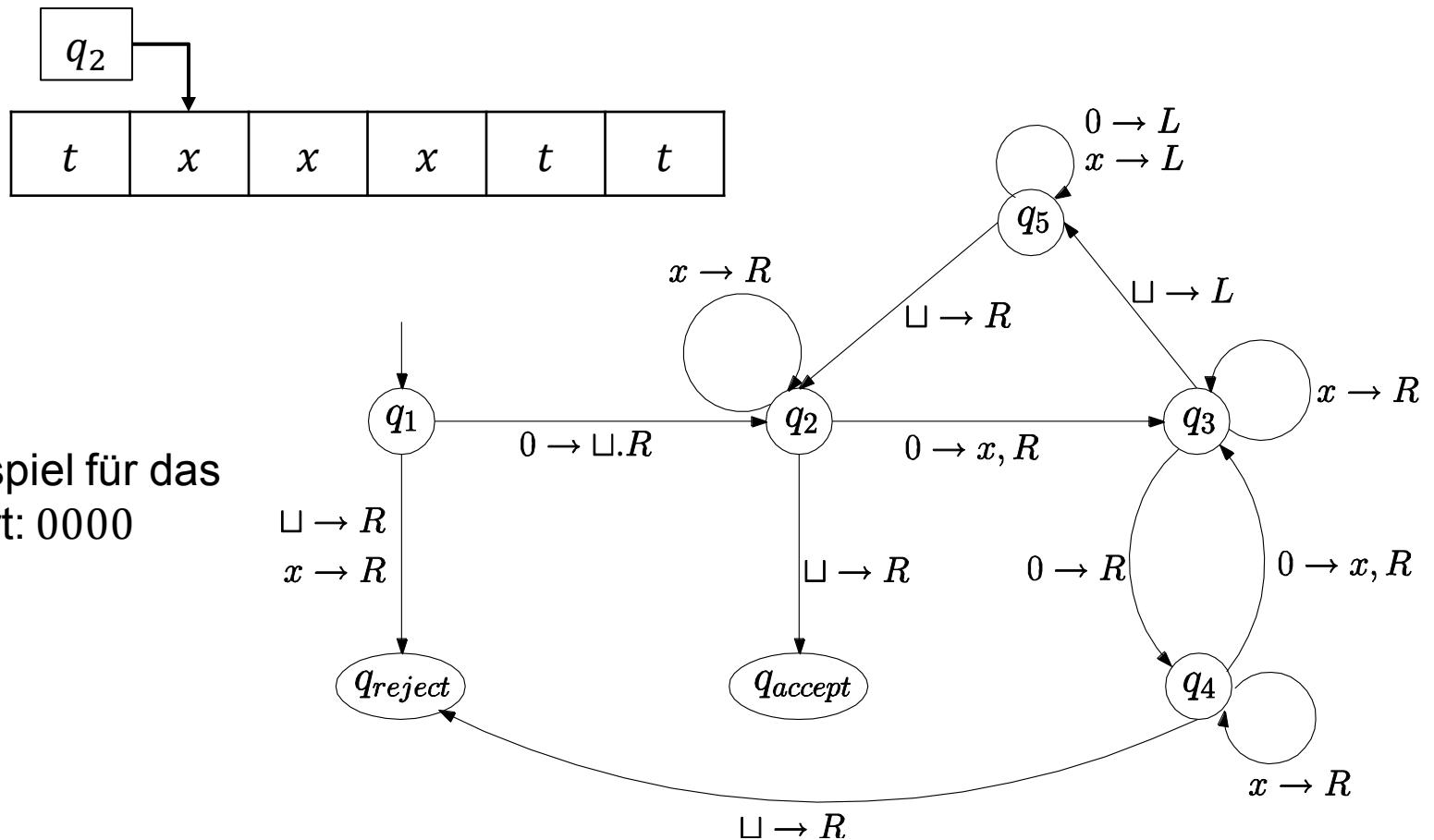


1. Einführung



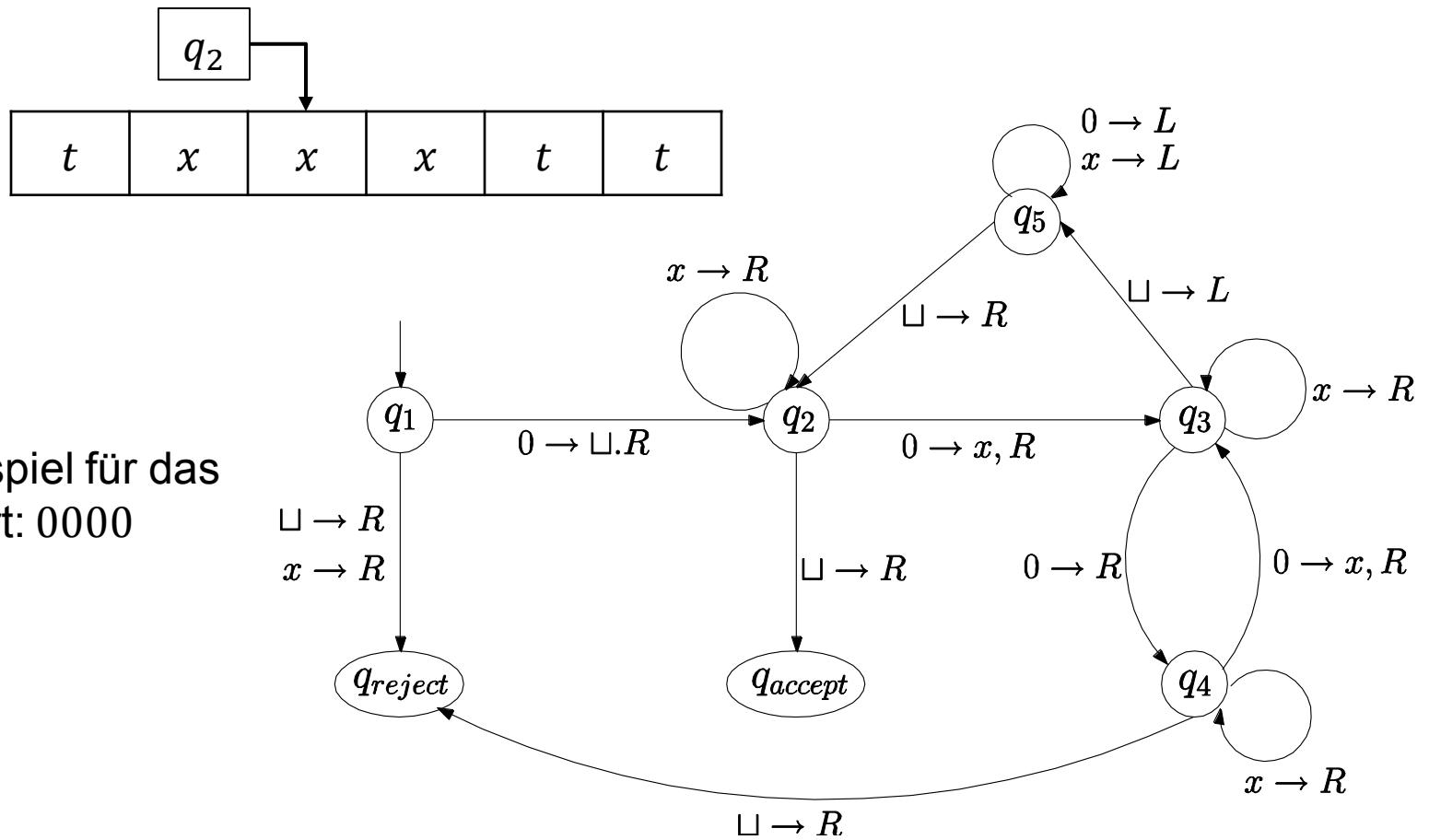
1. Einführung

- Beispiel für das Wort: 0000



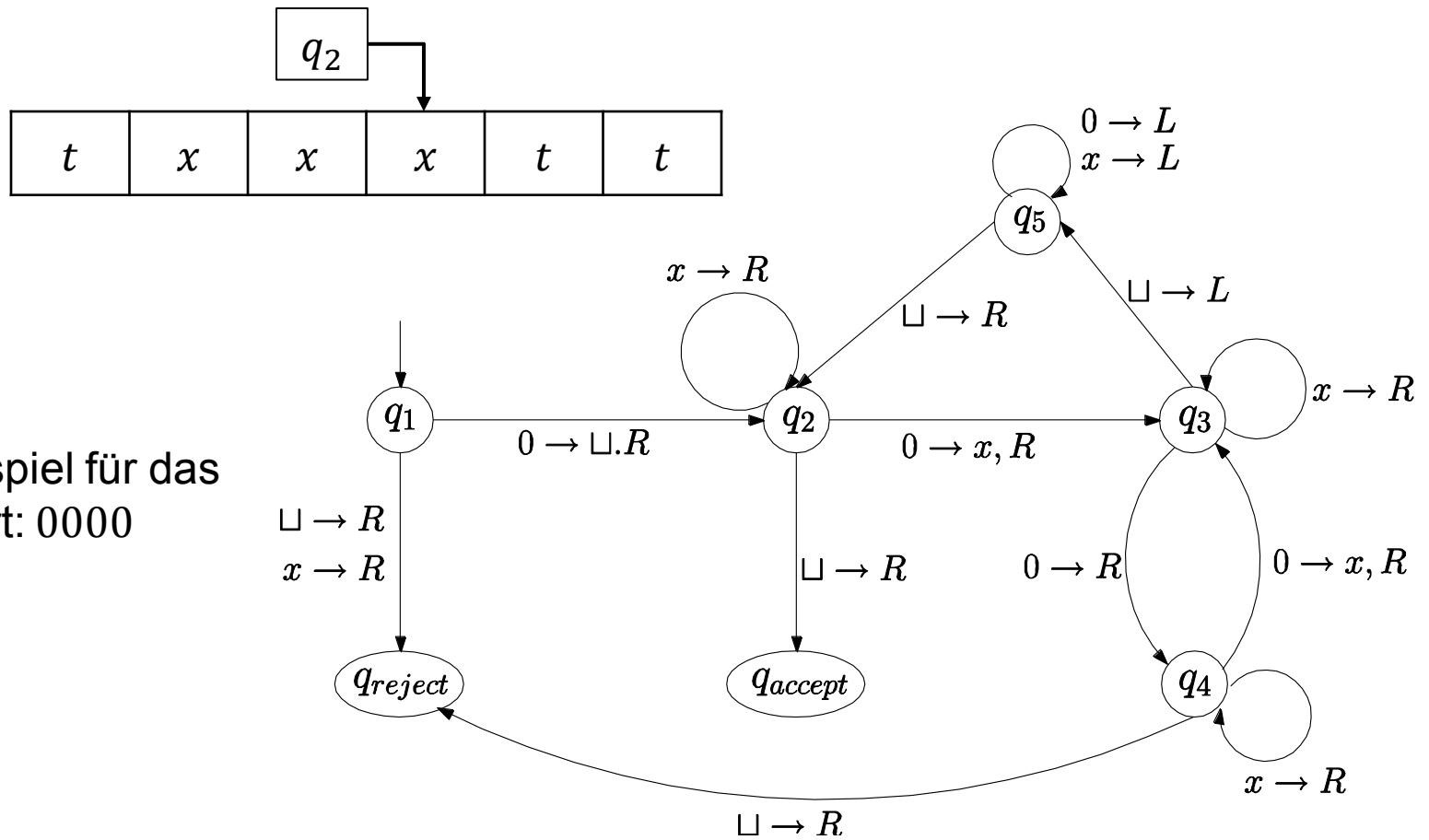
1. Einführung

- Beispiel für das Wort: 0000



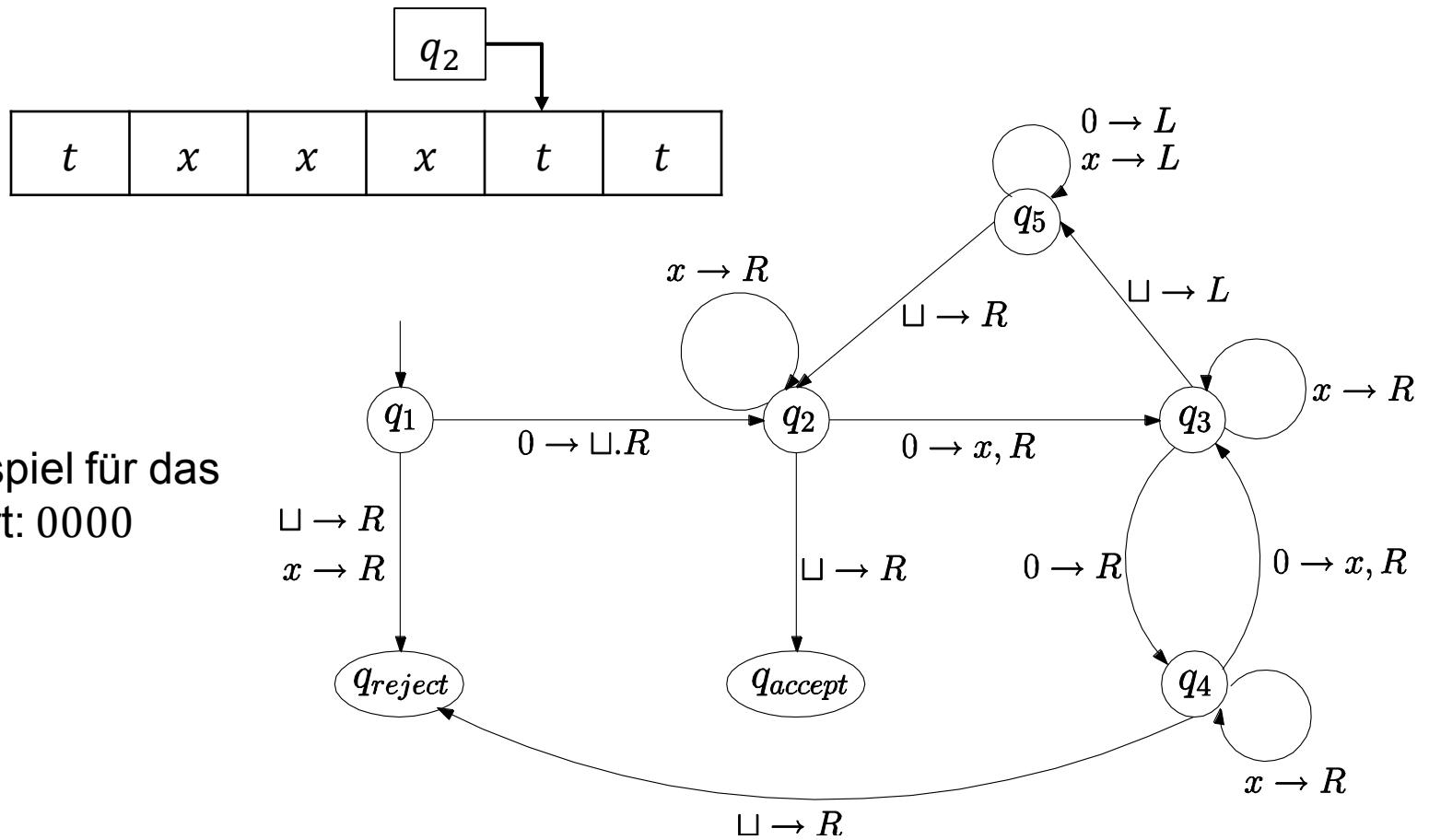
1. Einführung

- Beispiel für das Wort: 0000



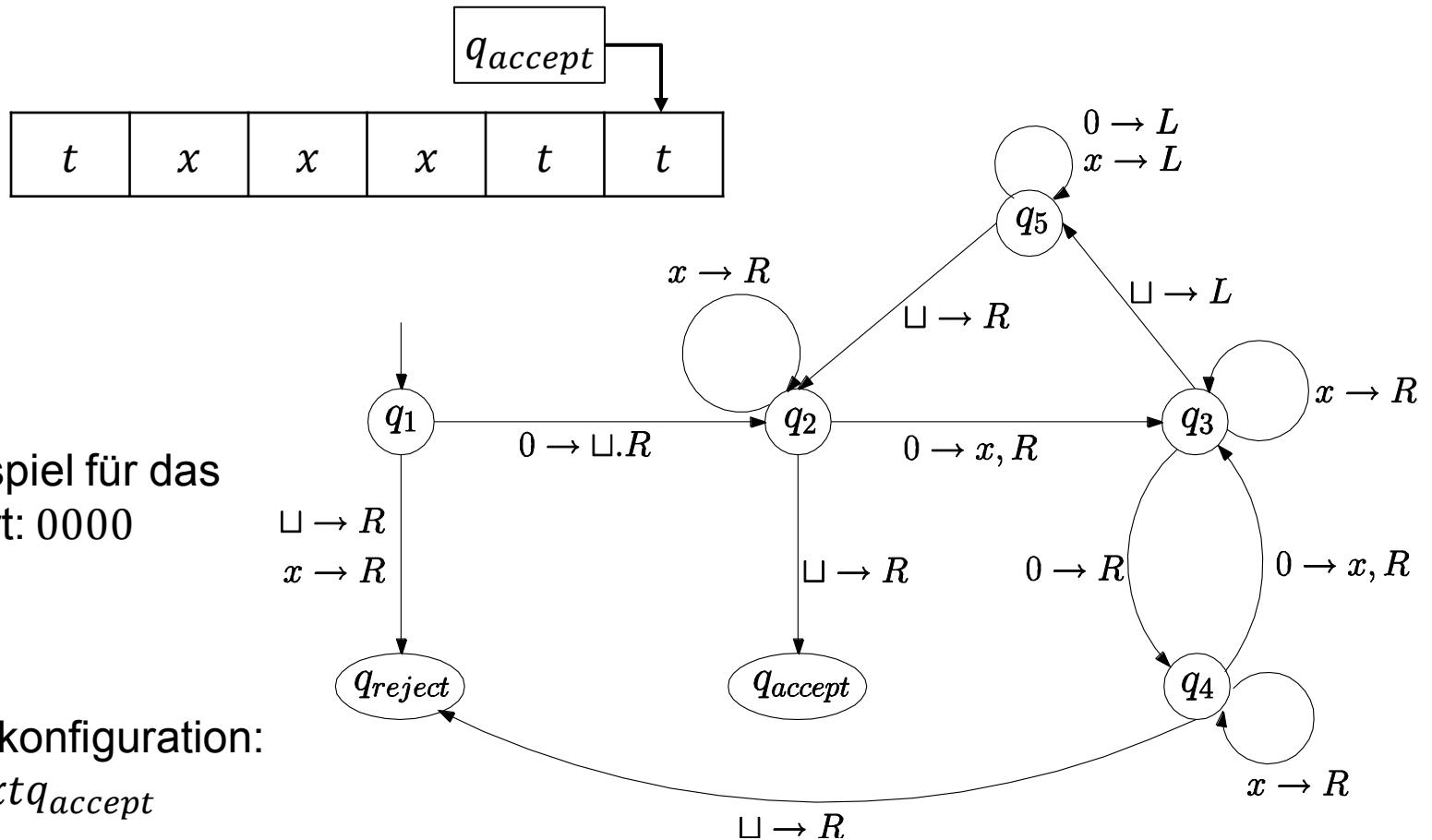
1. Einführung

- Beispiel für das Wort: 0000



1. Einführung

- Beispiel für das Wort: 0000
- Endkonfiguration:
 $txxxtq_{accept}$



1. Einführung

Definition

- Eine Turingmaschine M berechnet die Funktion $f: \Sigma^* \rightarrow \Gamma^* \setminus \{t\}$ falls für alle w aus Σ^* die Berechnung von M mit der Eingabe w in einer akzeptierenden Konfiguration hält und dabei der Bandinhalt $f(w)$ ist.

1. Einführung

- Eine Mehrband- oder k -Band Turingmaschine (k -Band DTM) hat k Bänder mit je einem Kopf.
- Die Übergangsfunktion ist dann von der Form $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Zu Beginn steht die Eingabe auf Band 1, sonst stehen überall Blanks. Die Arbeitsweise ist analog zu 1-Band-DTMs definiert.

1. Einführung

Satz

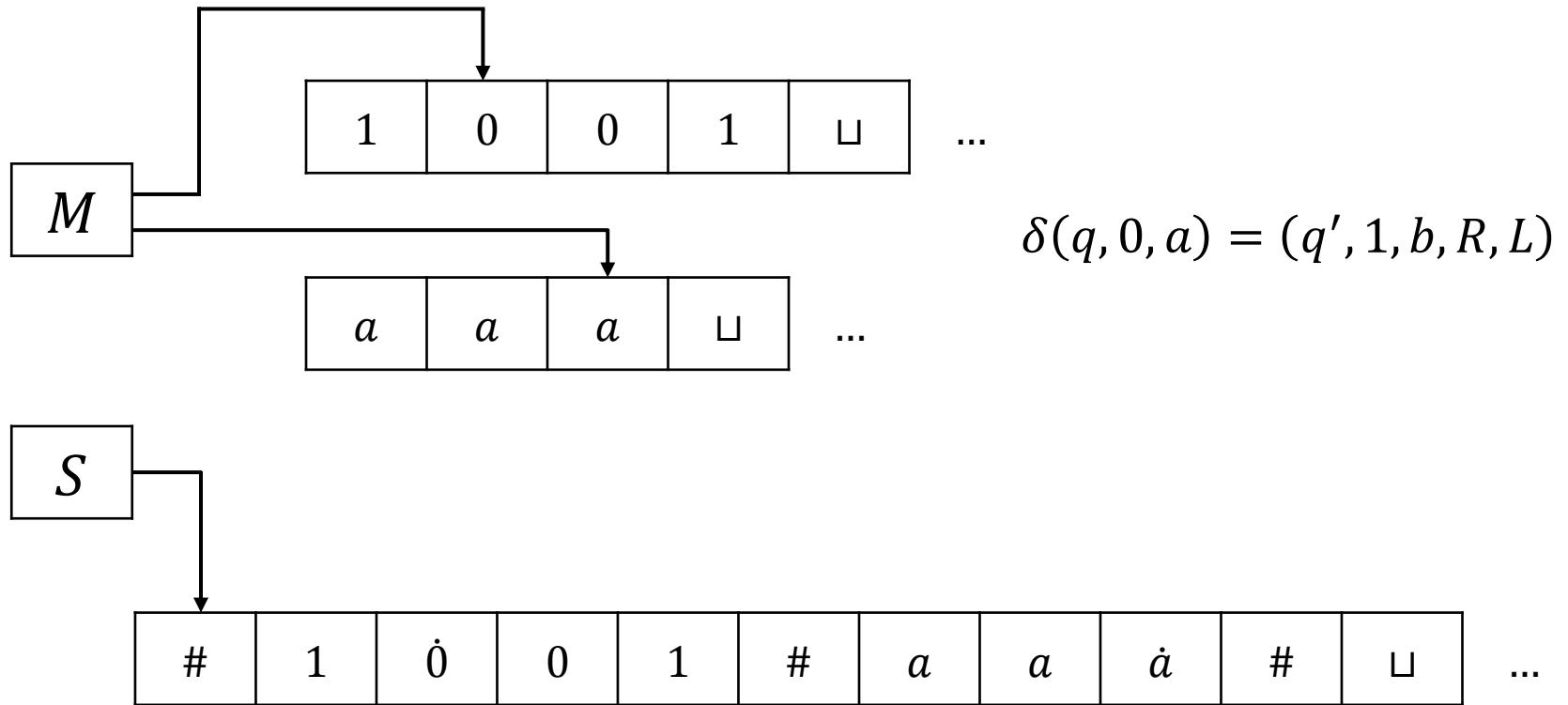
Zu jeder Mehrband-Turingmaschine gibt es eine äquivalente 1-Band-Turingmaschine.

Beweis

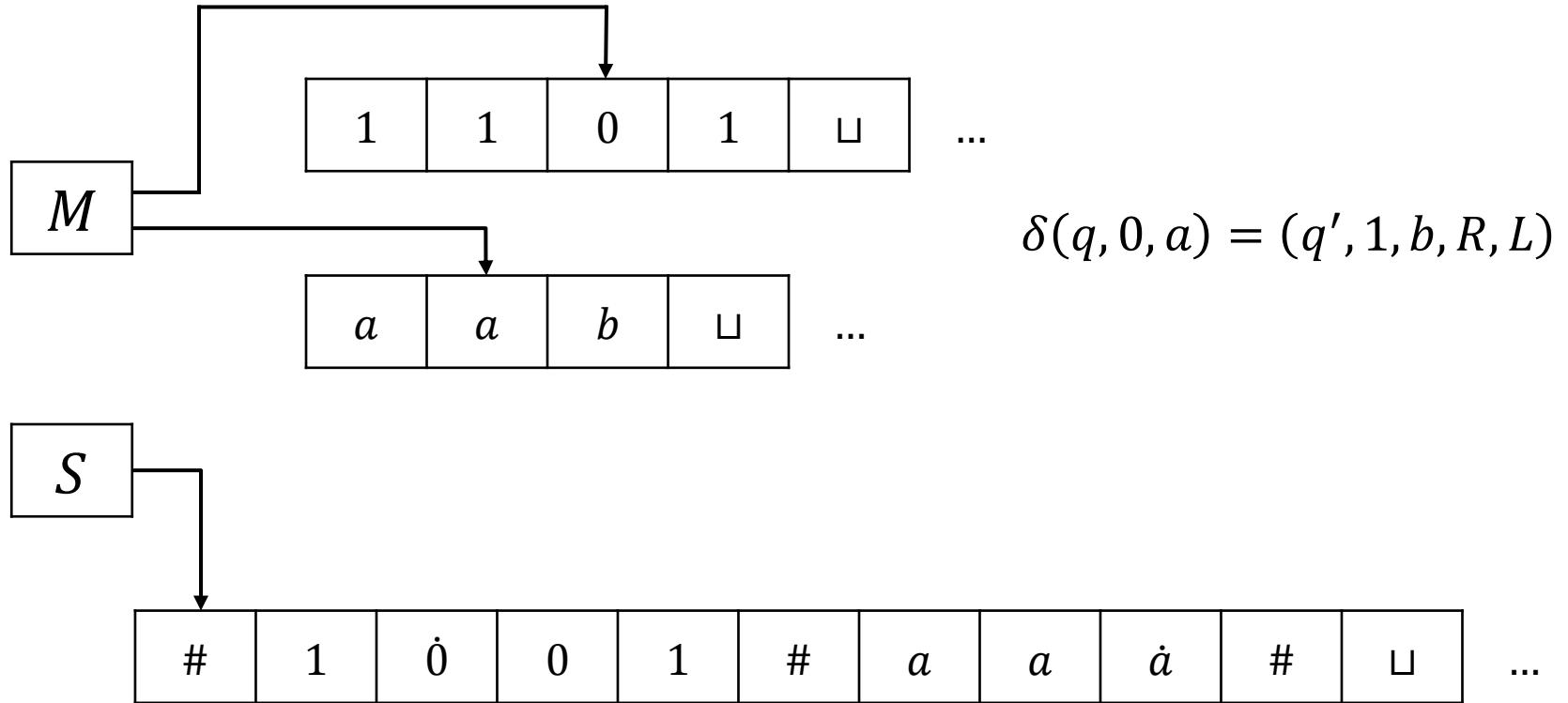
Idee:

Simuliere Mehrband-DTM M auf 1-Band-DTM S .

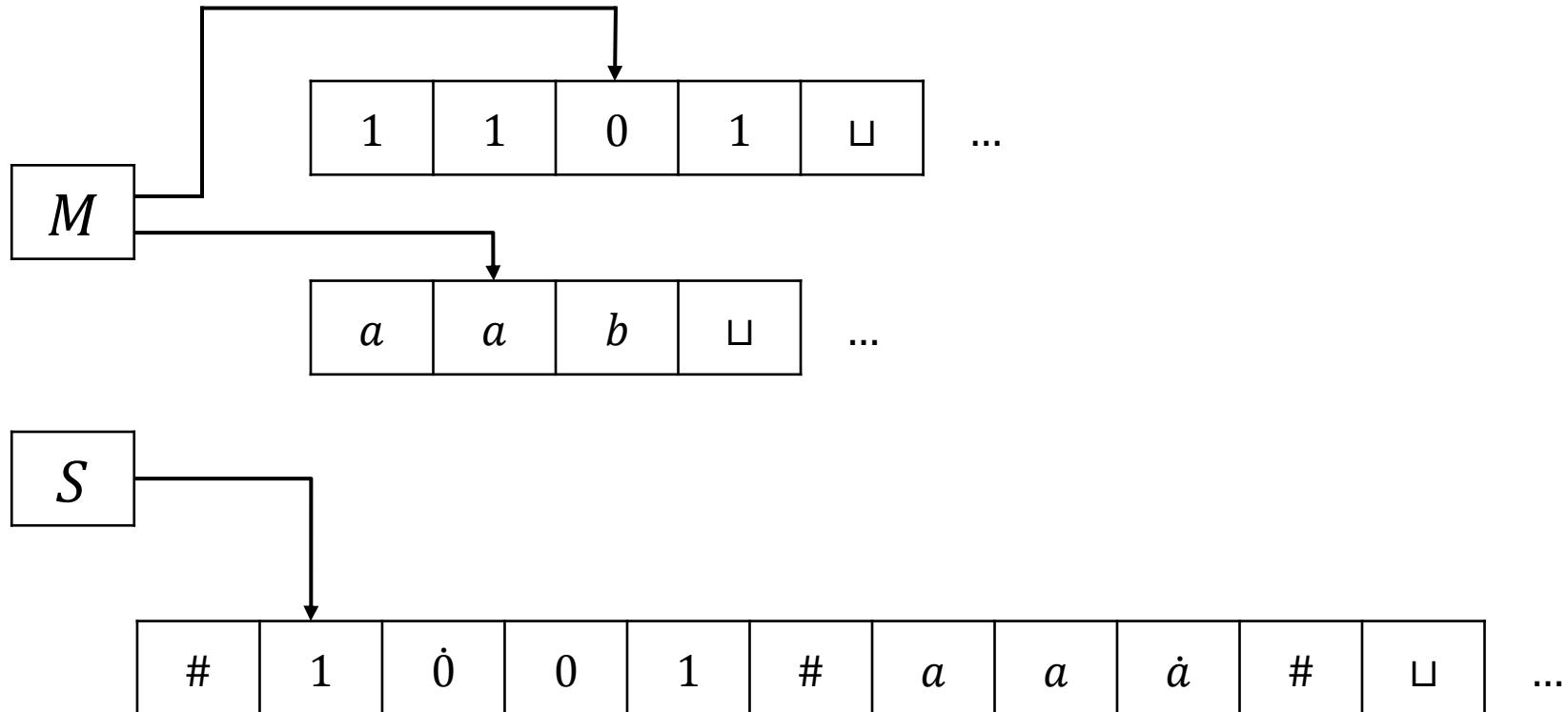
1. Einführung



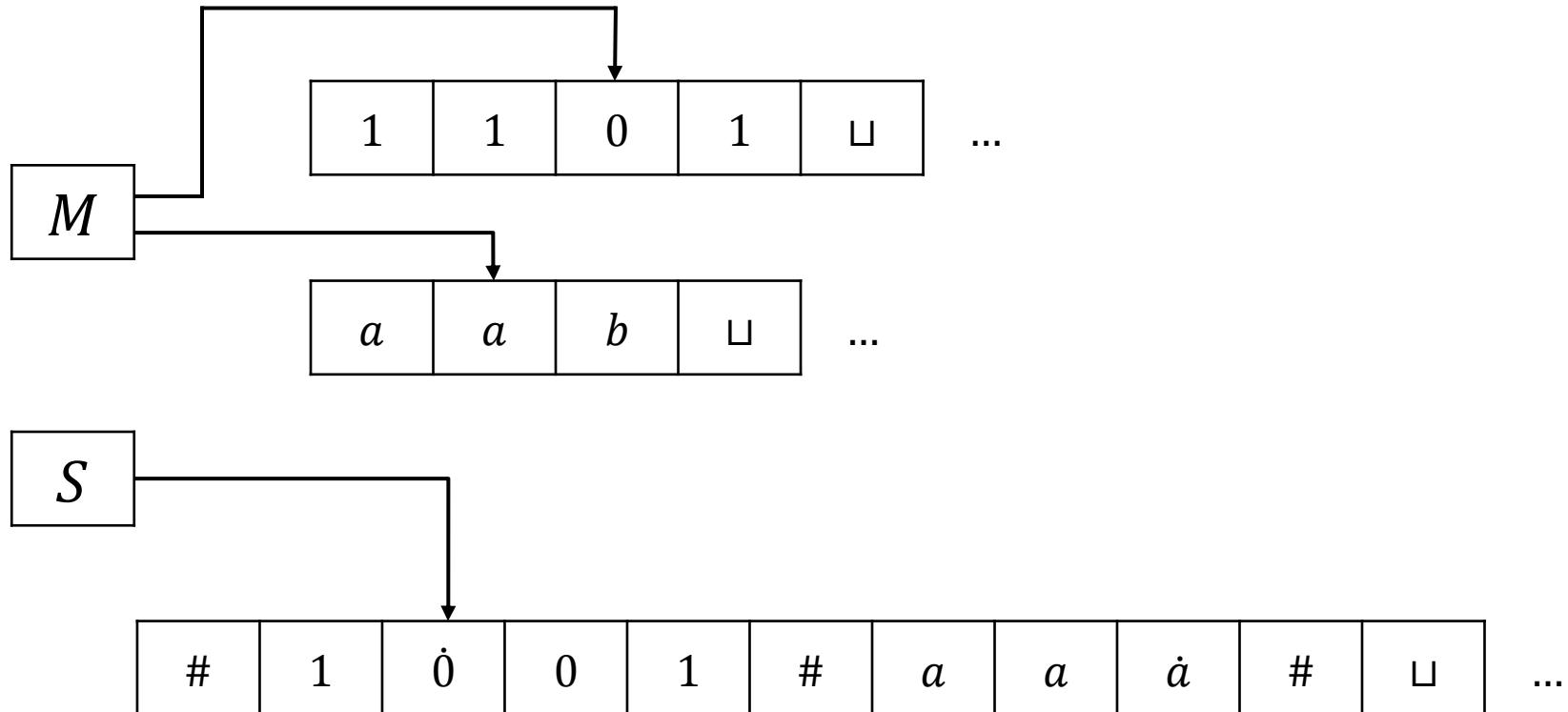
1. Einführung



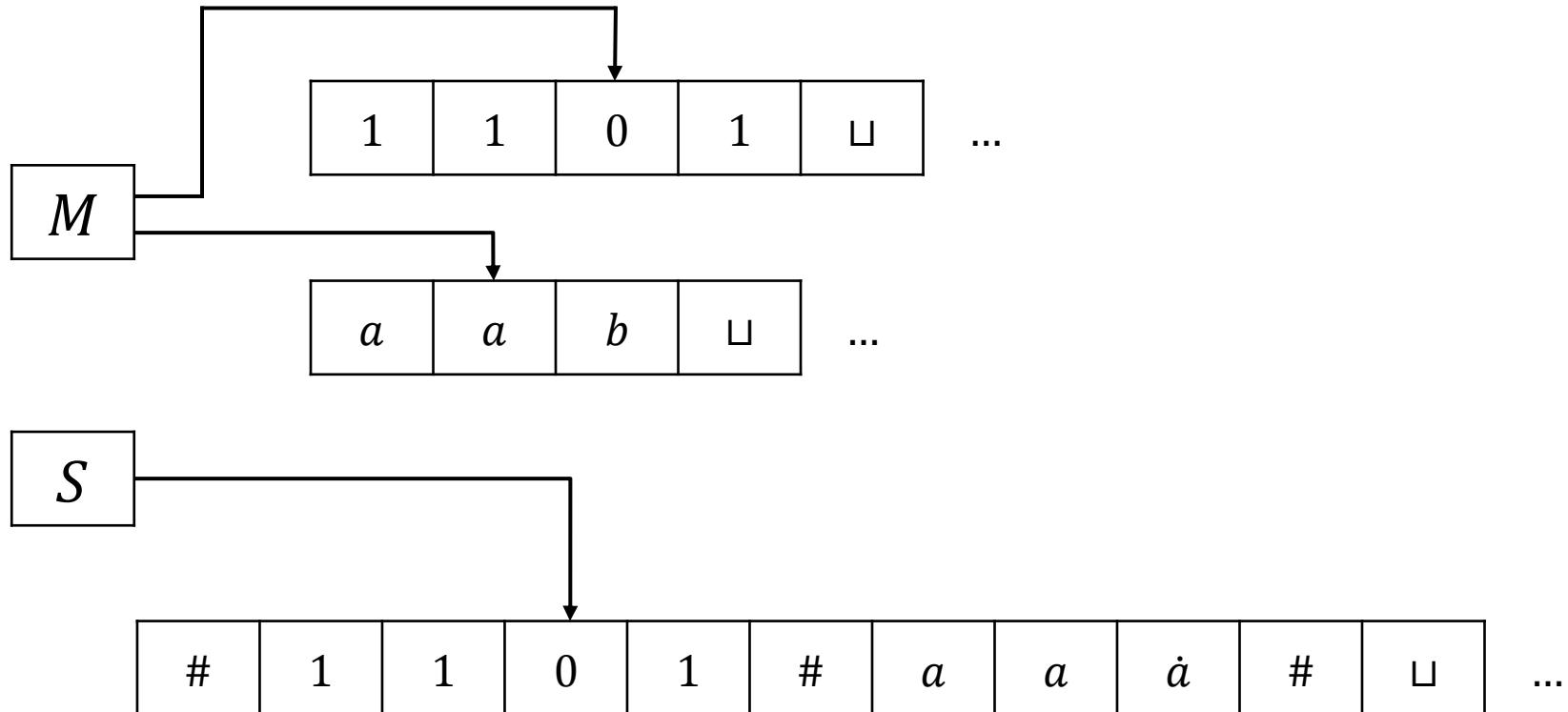
1. Einführung



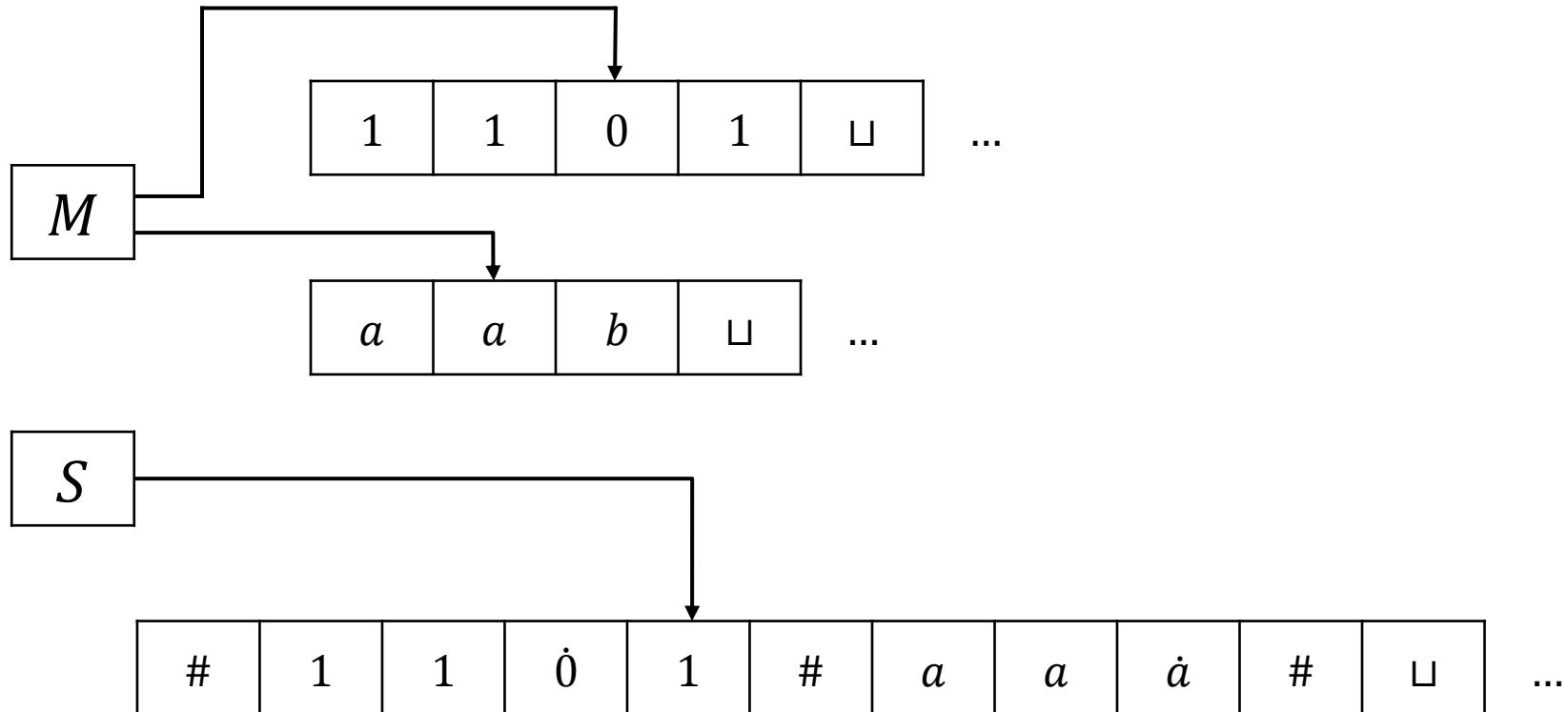
1. Einführung



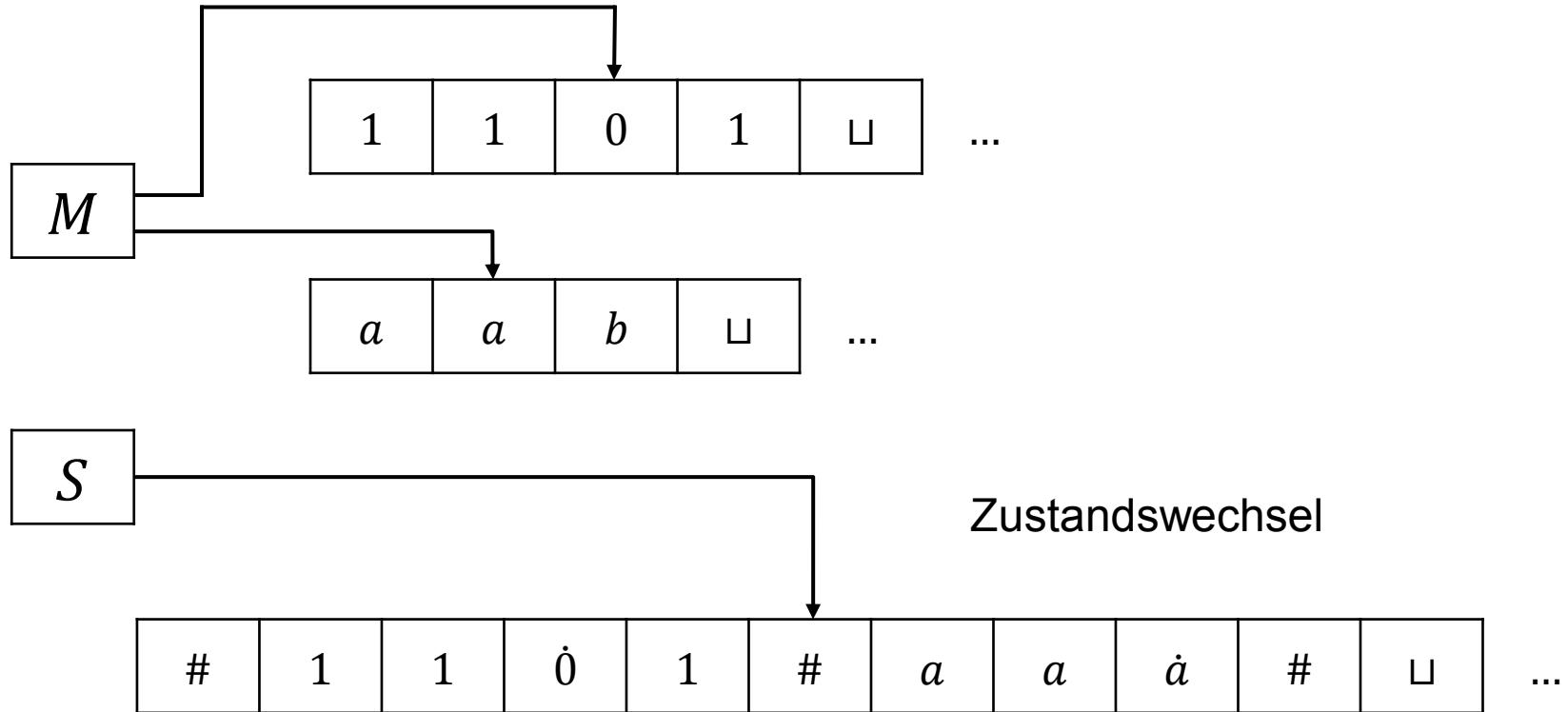
1. Einführung



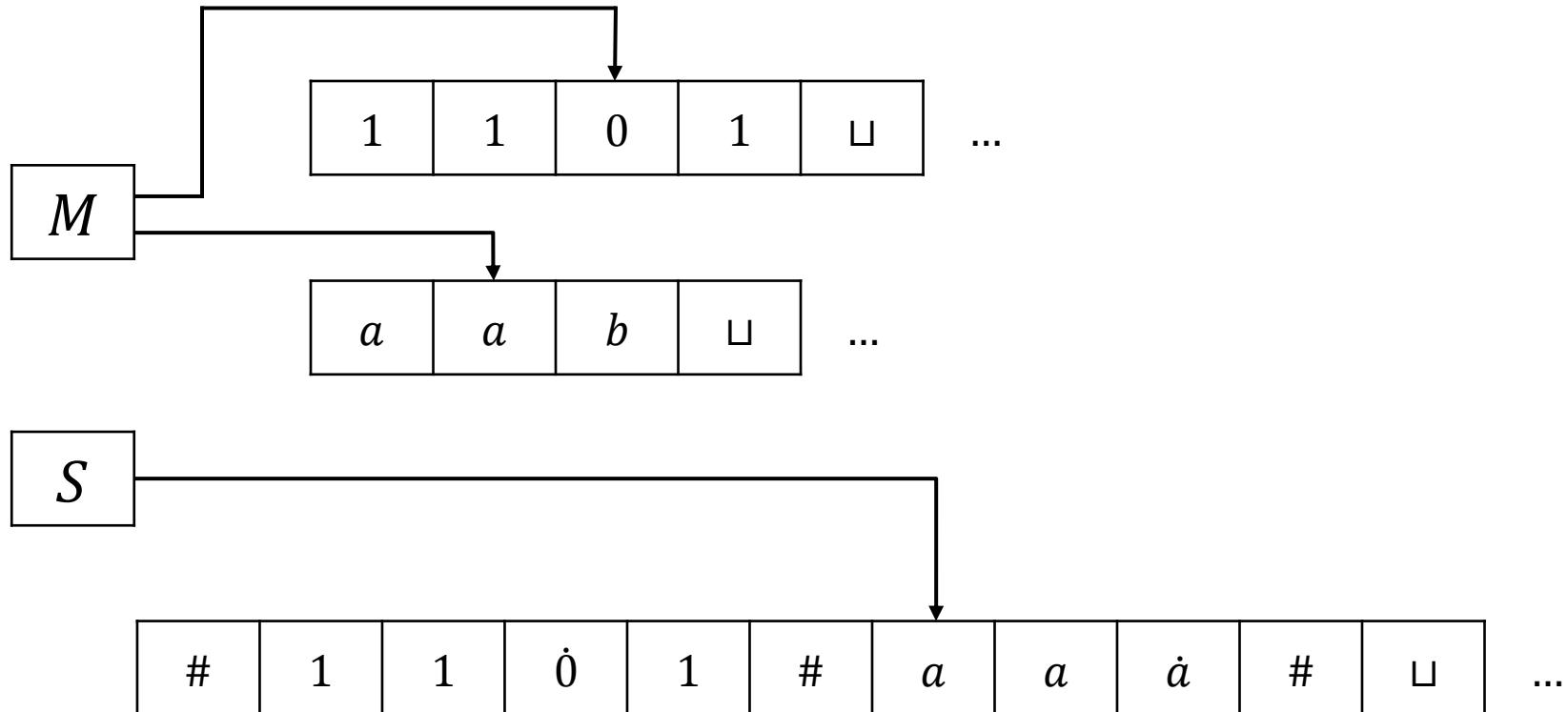
1. Einführung



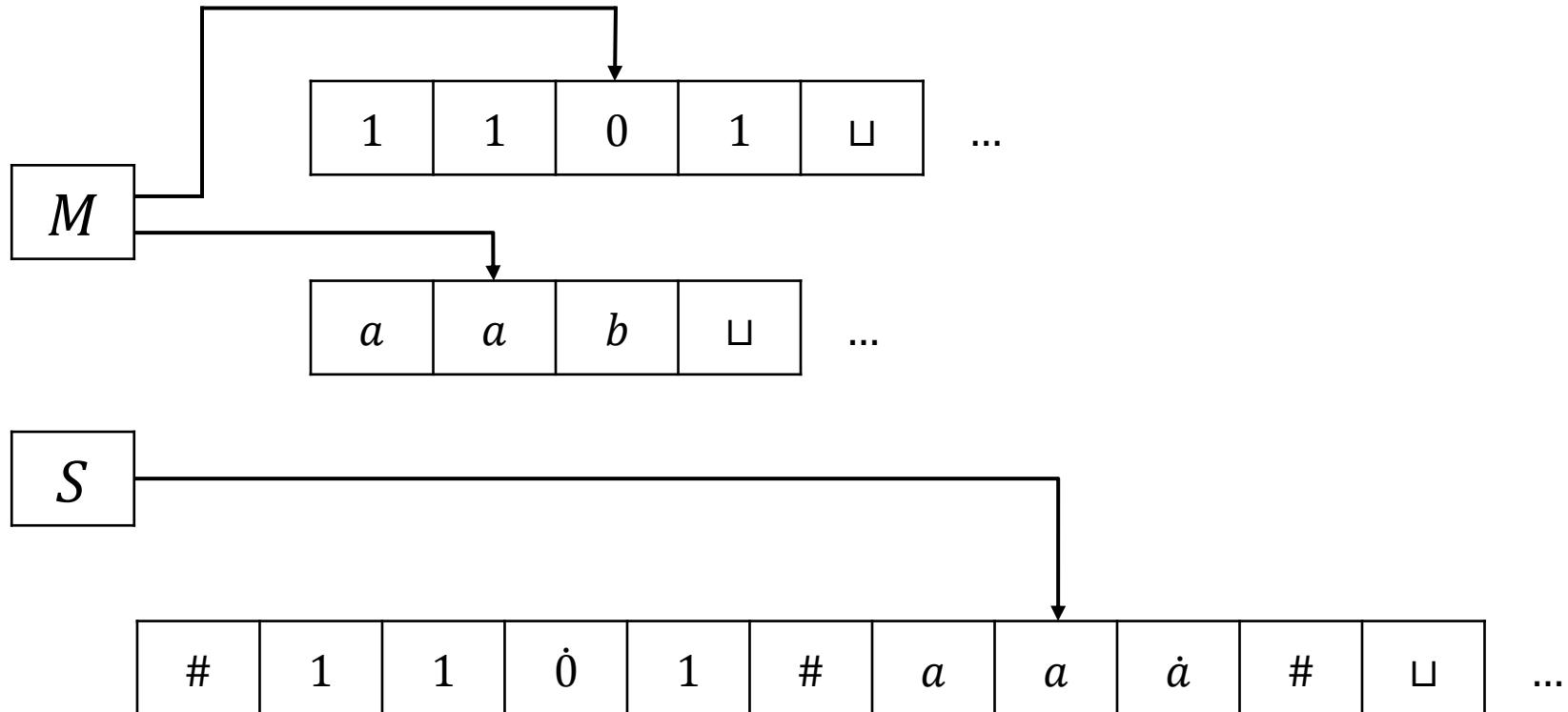
1. Einführung



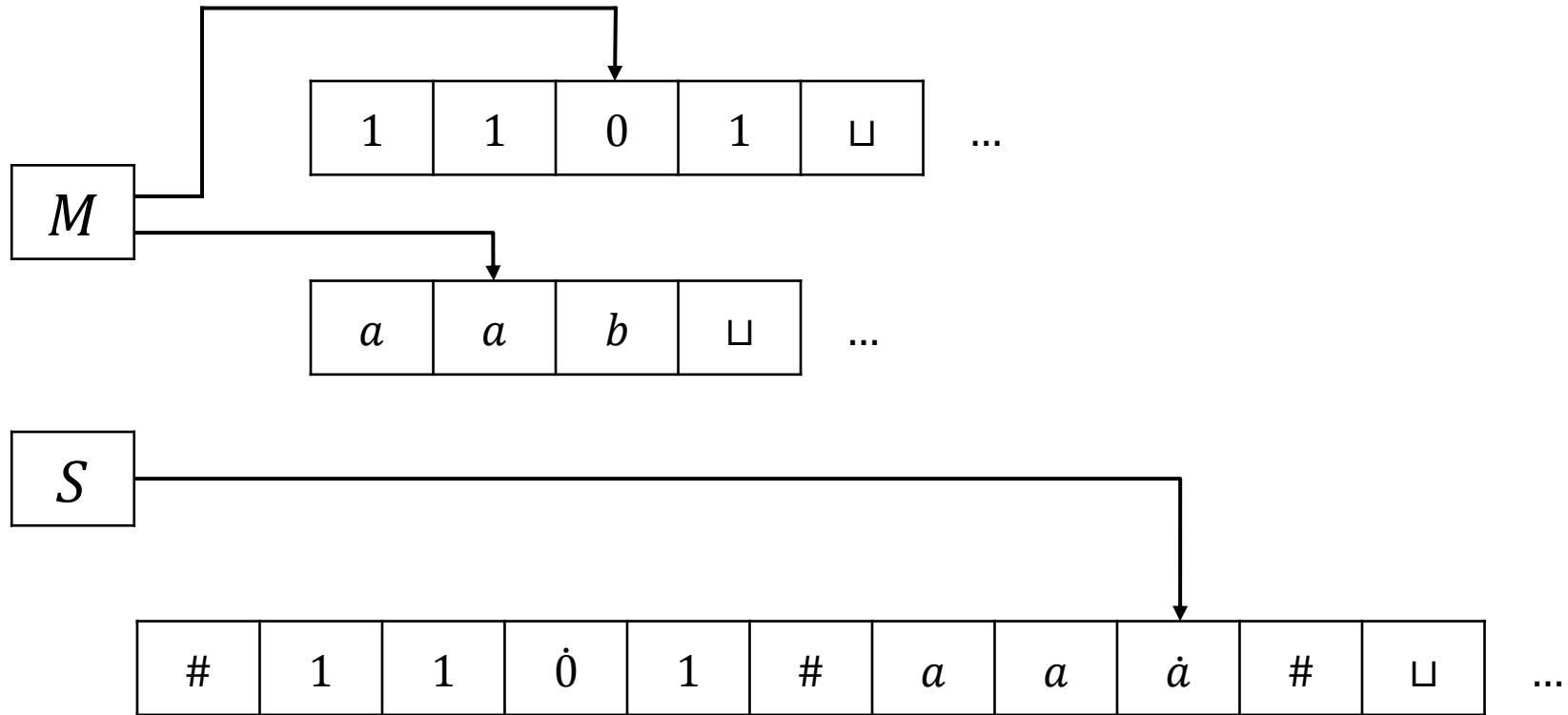
1. Einführung



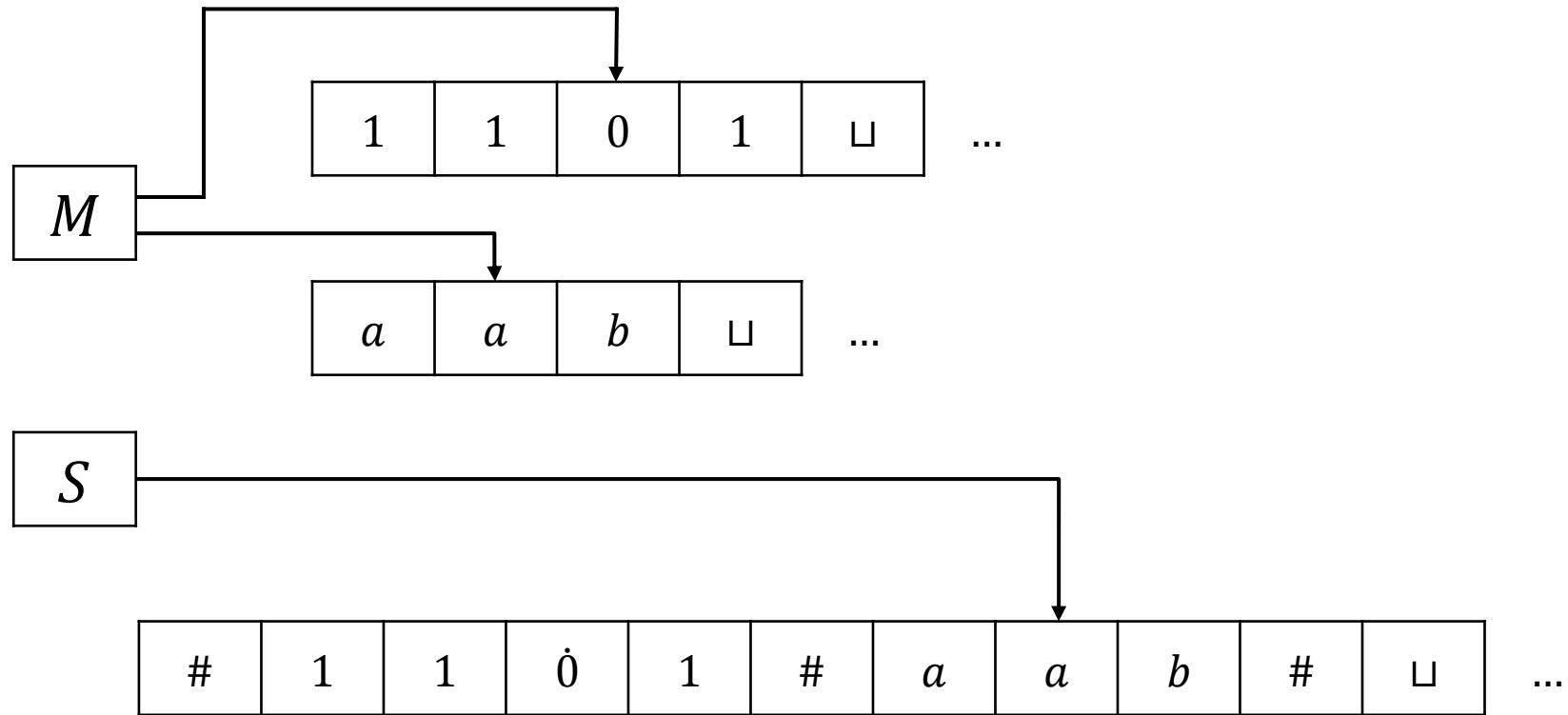
1. Einführung



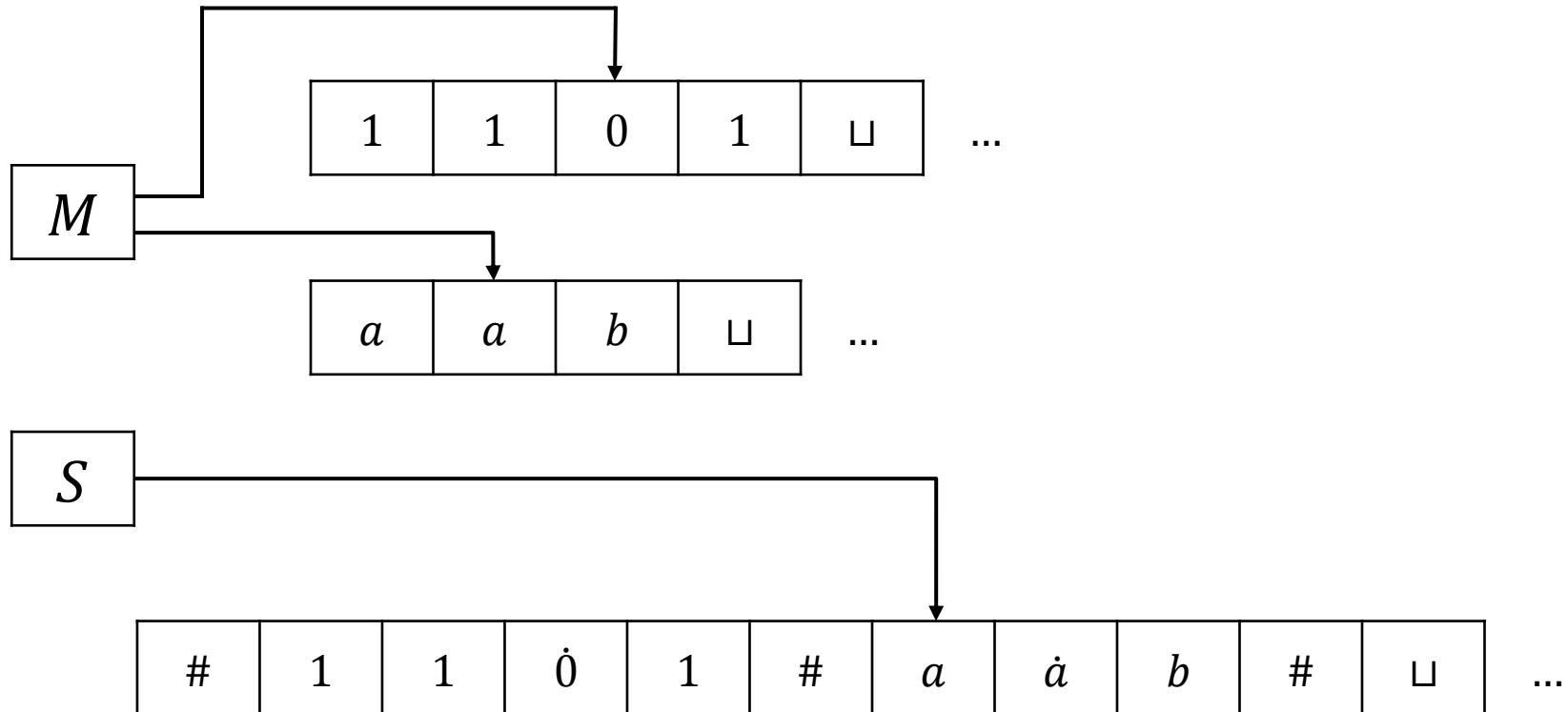
1. Einführung



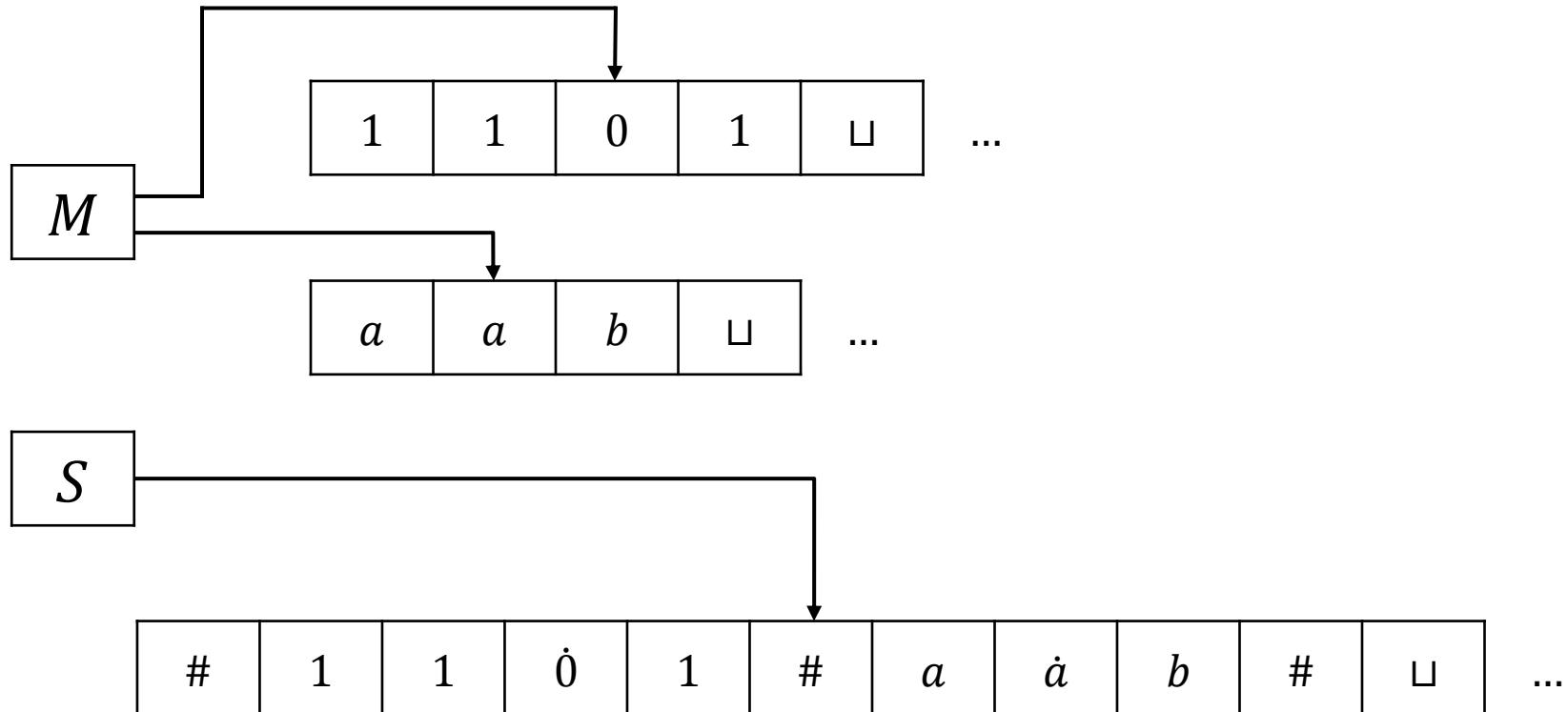
1. Einführung



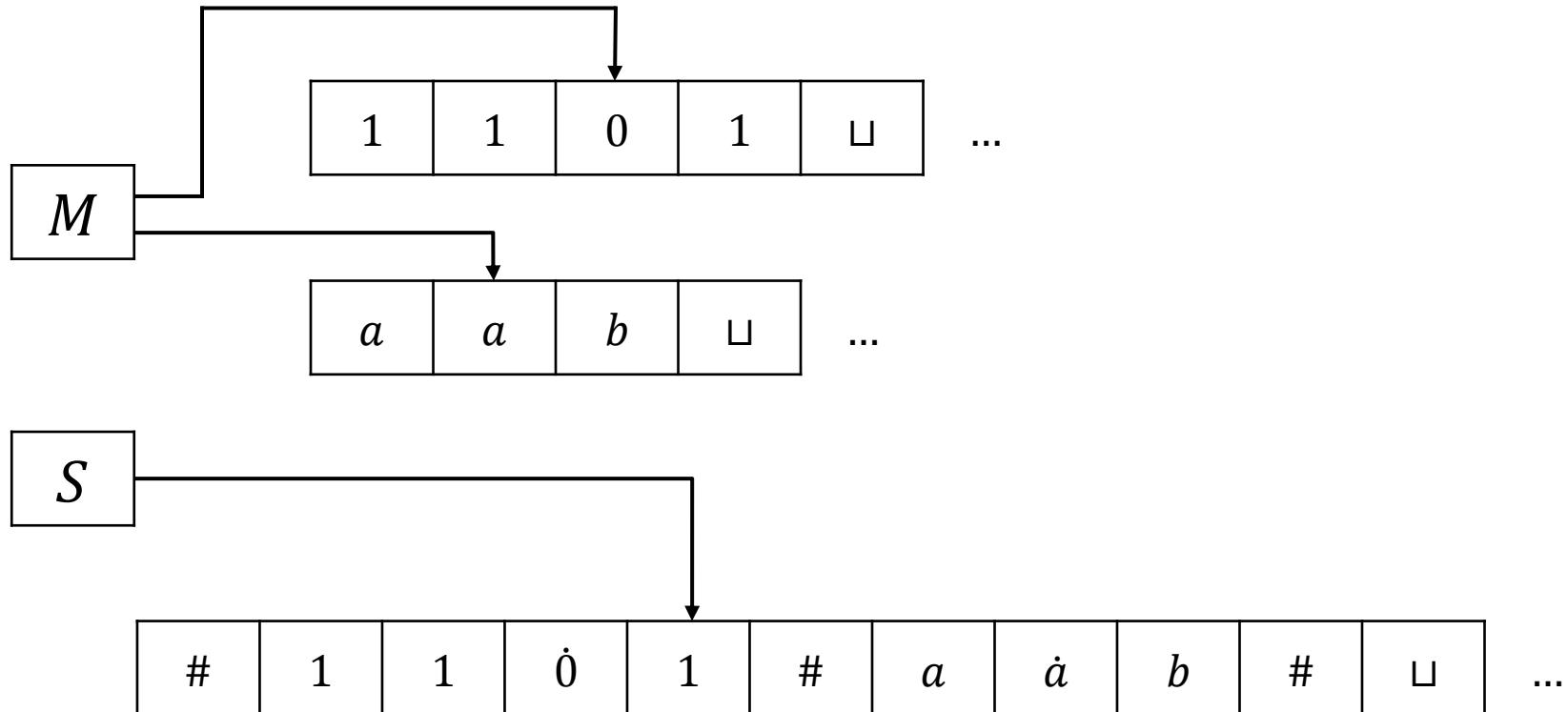
1. Einführung



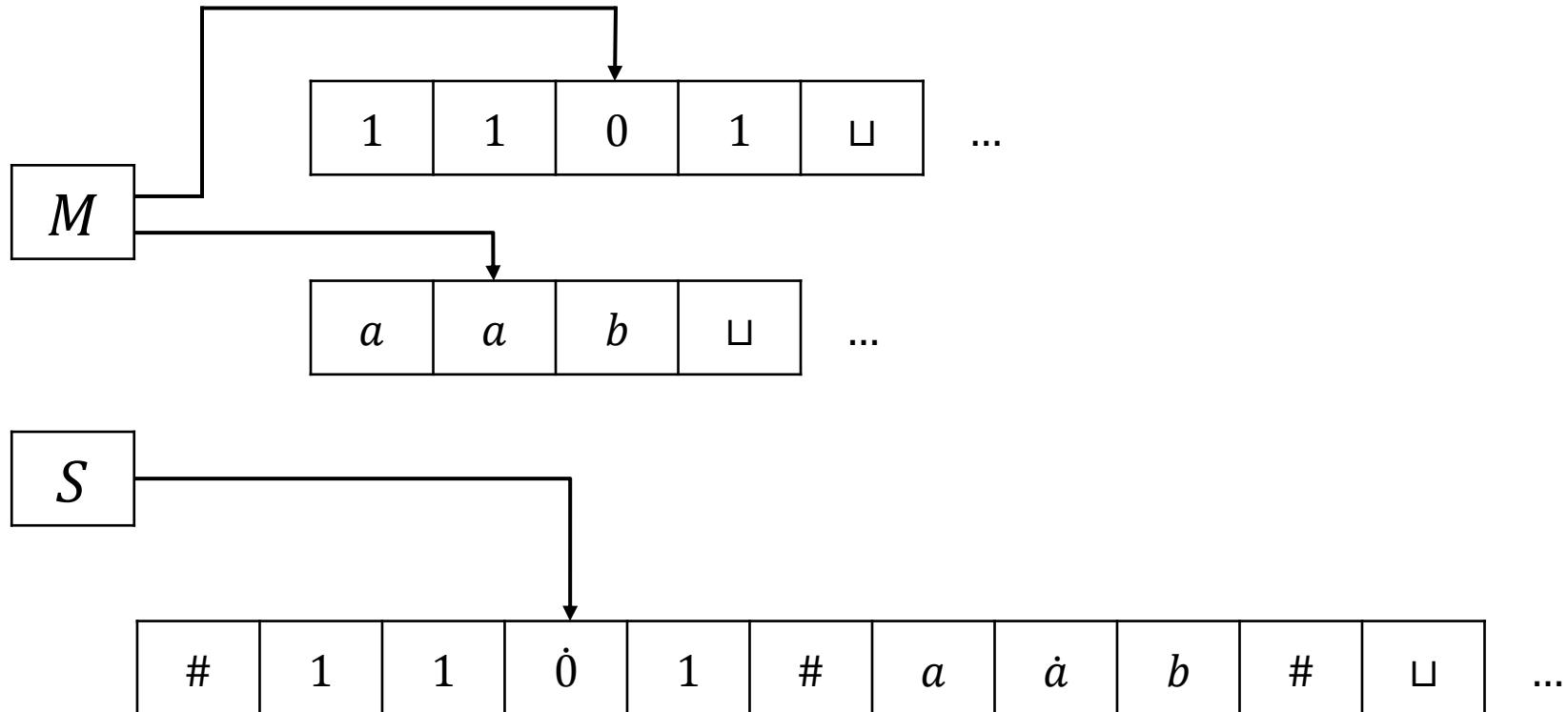
1. Einführung



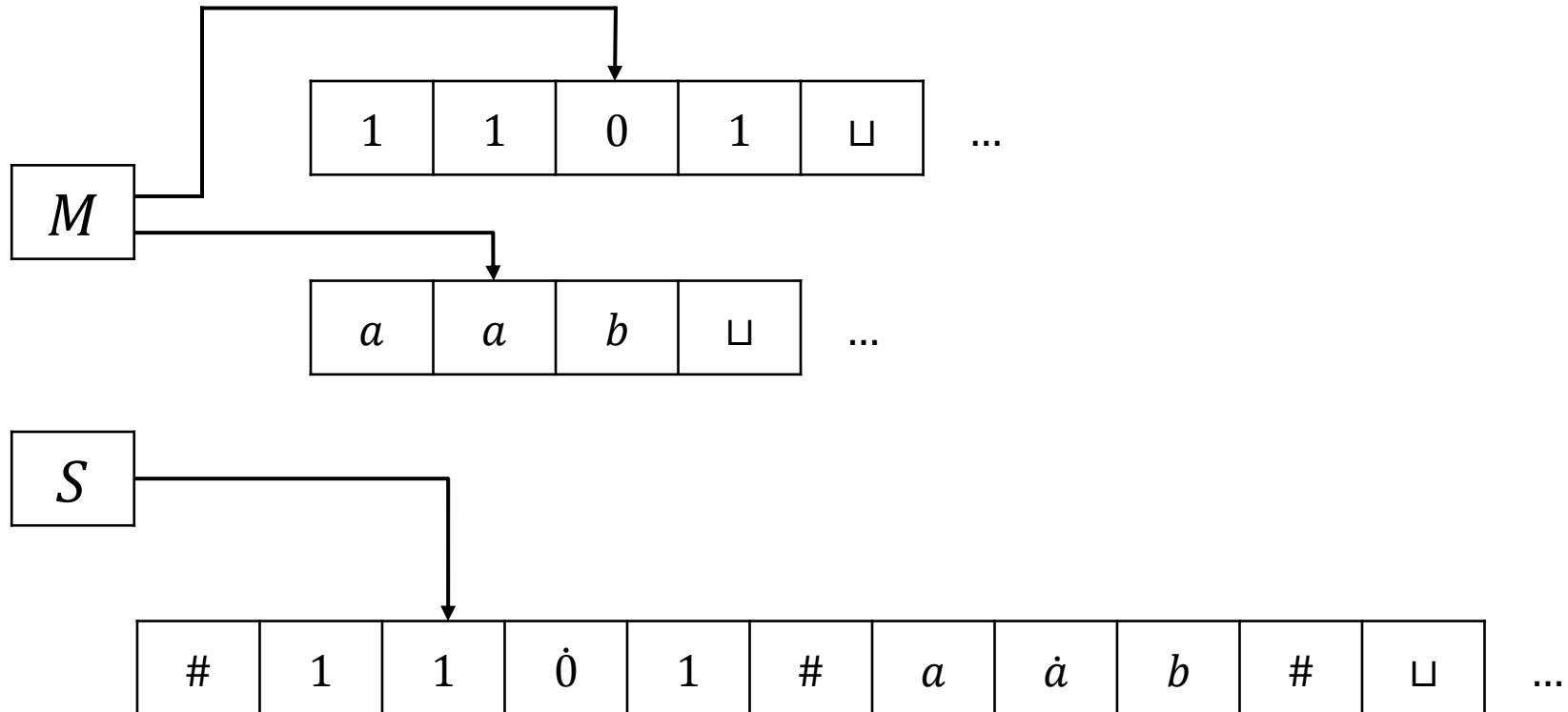
1. Einführung



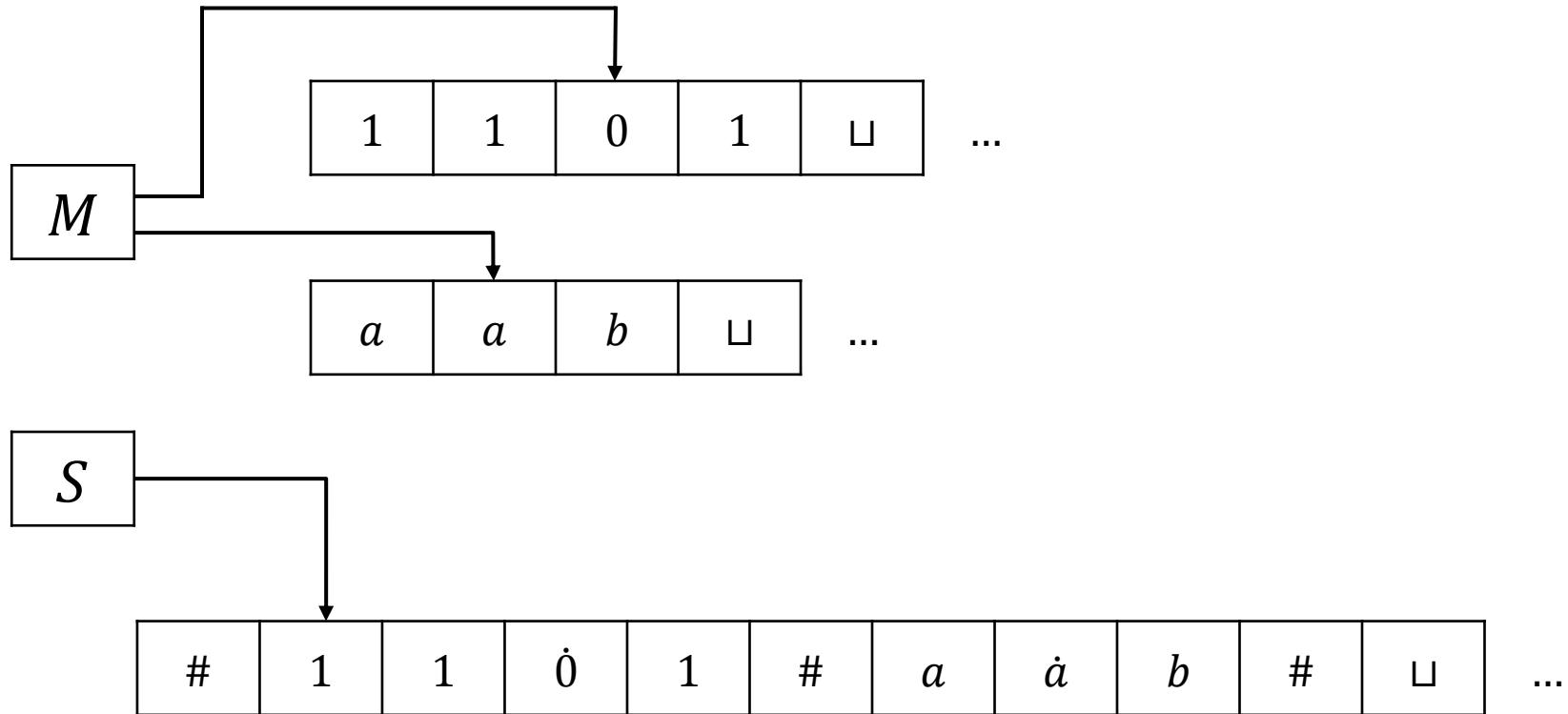
1. Einführung



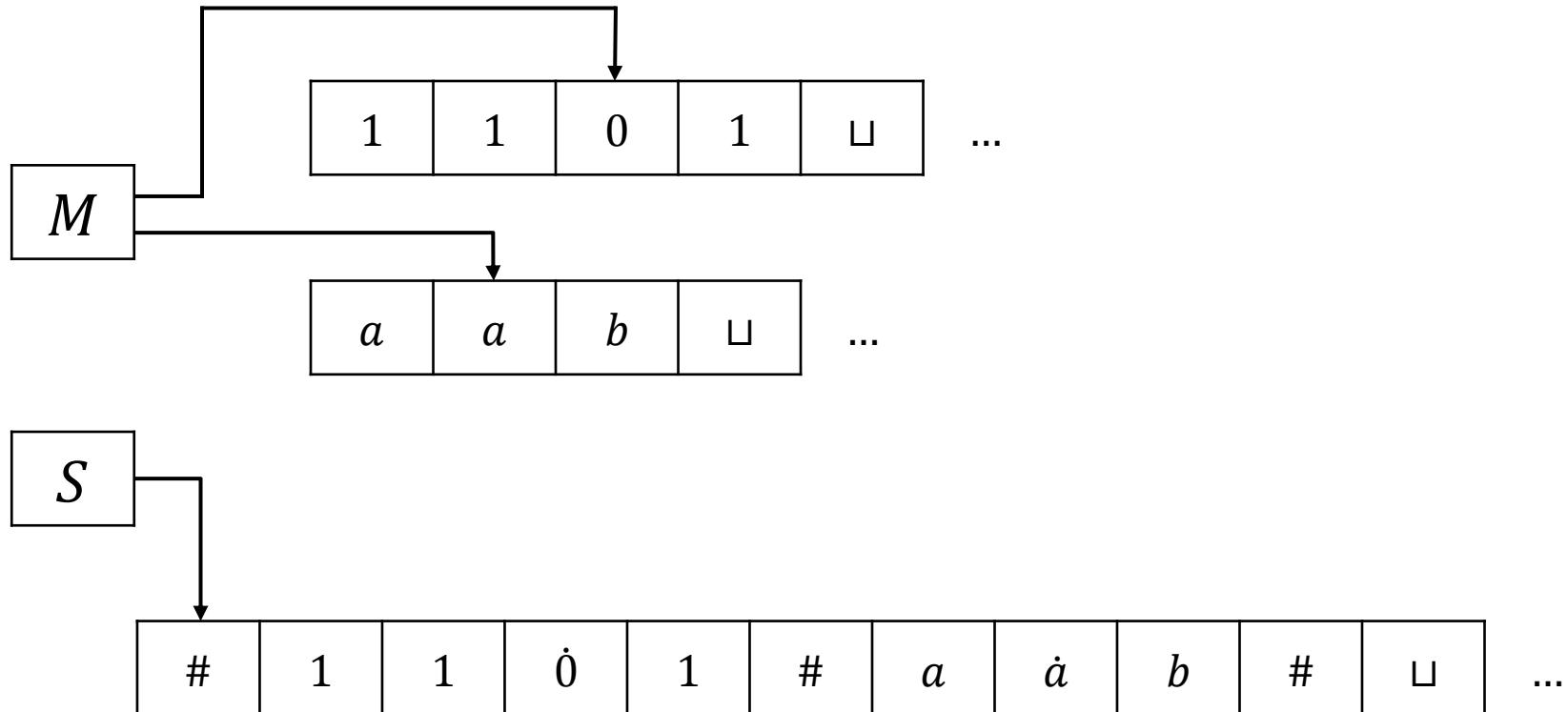
1. Einführung



1. Einführung



1. Einführung



1. Einführung

Korollar

Eine Sprache L ist genau dann rekursiv aufzählbar, wenn es eine Mehrband-Turingmaschine gibt, die L akzeptiert.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Inhaltsangabe

- Einleitung, Motivation
 - Turingmaschinen
 - Arbeitstechniken
- }
- Einführung**
-
- Unentscheidbare Probleme
 - Das Halteproblem
 - Reduktionen
- }
- Berechenbarkeit**
-
- Zeitkomplexität
 - Die Klassen P und NP
 - NP-Vollständigkeit
 - NP-vollständige Probleme
- }
- Komplexität**
-
- Formale Sprachen und Automaten
 - Kellerautomaten und kontextfreie Sprachen
 - Kontextsensitive Sprachen
- }
- Formale Sprachen**

1. Einführung

Gibt es einen Algorithmus HALTE, der

- als Eingabe einen beliebigen Algorithmus ALG und eine Eingabe w für ALG erhält und
- entscheidet, ob ALG bei Eingabe w hält?

Satz von Turing:

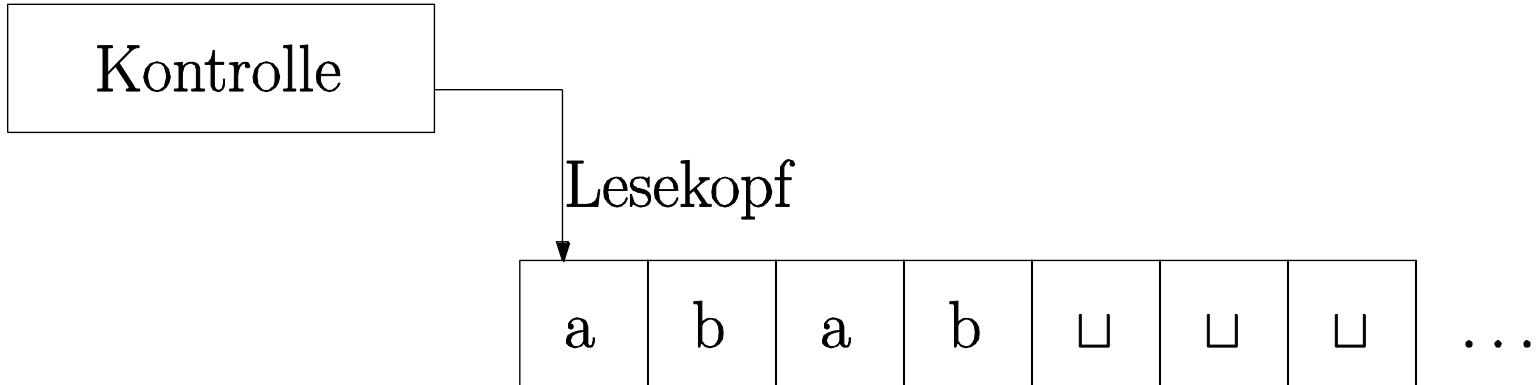
Einen solchen Algorithmus kann es nicht geben.

1. Einführung

- **Turingmaschine**

- Arbeitet auf unbeschränktem Band
- Eingabe steht zu Beginn am Anfang des Bands
- Auf dem Rest des Bandes steht t (Blank)
- Position auf dem Band wird durch den sog. *Lesekopf* beschrieben

Turingmaschine



- Der jeweils nächste Rechenschritt ist eindeutig festgelegt durch den aktuellen Zustand und das aktuell gelesene Zeichen.
- Der Rechenschritt überschreibt das aktuelle Zeichen, bewegt den Kopf nach rechts oder nach links und verändert den Zustand.

1. Einführung

Definition

Eine (*deterministische 1-Band Turingmaschine*) DTM wird beschrieben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Dabei sind Q, Σ, Γ endliche, nichtleere Mengen und es gilt:

- Σ ist Teilmenge von Γ
- t in $\Gamma \setminus \Sigma$ ist das *Blanksymbol* (auch \sqcup)
- Q ist die *Zustandsmenge*
- Σ ist das *Eingabealphabet*
- Γ ist das *Bandalphabet*
- q_0 in Q ist der *Startzustand*
- q_{accept} in Q ist der akzeptierende Endzustand
- q_{reject} in Q ist der ablehnende Endzustand
- $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die (partielle) *Übergangsfunktion*. Sie ist für kein Argument aus $\{q_{accept}, q_{reject}\} \times \Gamma$ definiert.

1. Einführung

- Initial:
 - Eingabe steht links auf dem Band
 - Der Rest des Bands ist leer
 - Kopf befindet sich ganz links
- Berechnungen finden entsprechend der Übergangsfunktion statt
- Wenn der Kopf sich am linken Ende befindet und nach links bewegen soll, bleibt er an seiner Position
- Wenn q_{accept} oder q_{reject} erreicht wird, ist die Bearbeitung beendet

1. Einführung

Momentaufnahme einer Turingmaschine:

- Bei Bandinschrift uv (dabei beginnt u am linken Ende des Bandes und hinter v stehen nur Blanks)
- Zustand q
- Kopf auf erstem Zeichen von v

Konfiguration $C = uqv$

1. Einführung

- Gegeben: Konfigurationen C_1, C_2
- Wir sagen: **Konfiguration C_1 führt zu C_2** , falls die TM von C_1 in einem Schritt zu C_2 übergehen kann

Formal:

- Seien a, b, c in Γ , u, v in Γ^* und Zustände q_i, q_j gegeben
- Wir sagen:
 - $uaq_i bv$ führt zu $uq_j acv$, falls $\delta(q_i, b) = (q_j, c, L)$ und
 - $uaq_i bv$ führt zu $uacq_j v$, falls $\delta(q_i, b) = (q_j, c, R)$

1. Einführung

- Startkonfiguration:
 - $q_0 w$, wobei w die Eingabe ist
- Akzeptierende Konfiguration:
 - Konfigurationen mit Zustand q_{accept}
- Ablehnende Konfiguration:
 - Konfigurationen mit Zustand q_{reject}
- Haltende Konfiguration:
 - akzeptierende oder ablehnende Konfigurationen

1. Einführung

Definition

Eine Turingmaschine M akzeptiert eine Eingabe w , falls es eine Folge von Konfigurationen C_1, C_2, \dots, C_k gibt, sodass

1. C_1 ist die Startkonfiguration von M bei Eingabe w
2. C_i führt zu C_{i+1}
3. C_k ist eine akzeptierende Konfiguration

- Die von M akzeptierten Worte bilden die von M akzeptierte Sprache $L(M)$.
- Eine Turingmaschine entscheidet eine Sprache, wenn jede Eingabe in einer haltenden Konfiguration C_k resultiert.

1. Einführung

Definition

- Eine Sprache L heißt **rekursiv aufzählbar**, falls es eine Turingmaschine M gibt, die L akzeptiert.
- Eine Sprache L heißt **rekursiv** oder **entscheidbar**, falls es eine Turingmaschine M gibt, die L entscheidet.

1. Einführung

- Eine Mehrband- oder k -Band Turingmaschine (k -Band DTM) hat k Bänder mit je einem Kopf.
- Die Übergangsfunktion ist dann von der Form $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Zu Beginn steht die Eingabe auf Band 1, sonst stehen überall Blanks. Die Arbeitsweise ist analog zu 1-Band-DTMs definiert.

1. Einführung

Satz

Zu jeder Mehrband-Turingmaschine gibt es eine äquivalente 1-Band-Turingmaschine.

Beweis

Idee:

Simuliere Mehrband-DTM M auf 1-Band-DTM S .

2. Berechenbarkeit

Simulationstechniken:

- Merken im Zustand
 - Nutzen Zustände als endlichen Speicher
- Markieren von Symbolen
 - Nutzen Bandalphabet zur Markierung von Positionen des Bandes

2. Berechenbarkeit

Im Zustand merken:

$$L := \{w \mid w = w_1 \dots w_n, \exists i, 2 \leq i \leq n: w_i = w_1\}$$

1. $\delta(q_0, t) = (q_2, t, R)$
2. $\delta(q_0, a) = ([q_0, a], a, R)$ für alle a aus Σ
3. $\delta([q_0, a], a) = (q_1, a, R)$
4. $\delta([q_0, a], b) = ([q_0, a], b, R)$
5. $\delta([q_0, a], t) = (q_2, t, R)$

$$q_{accept} = q_1, q_{reject} = q_2$$

2. Berechenbarkeit

Element-Distinctness:

$L := \{\#w_1\#w_2 \dots \#w_n \mid w_i \text{ aus } \{0,1\}^*, w_i \neq w_j \text{ für alle } i \neq j\}$

Beispiele:

- #011#001#01#00 ist in L
- #011#001#01#00#001 ist nicht in L

2. Berechenbarkeit

Element-Distinctness:

$L := \{\#w_1\#w_2 \dots \#w_n \mid w_i \text{ aus } \{0,1\}^*, w_i \neq w_j \text{ für alle } i \neq j\}$

Beispiele:

- #011#001#01#00 ist in L
- #011#**001**#01#00#**001** ist nicht in L

2. Berechenbarkeit

Turingmaschine für Element-Distinctness:

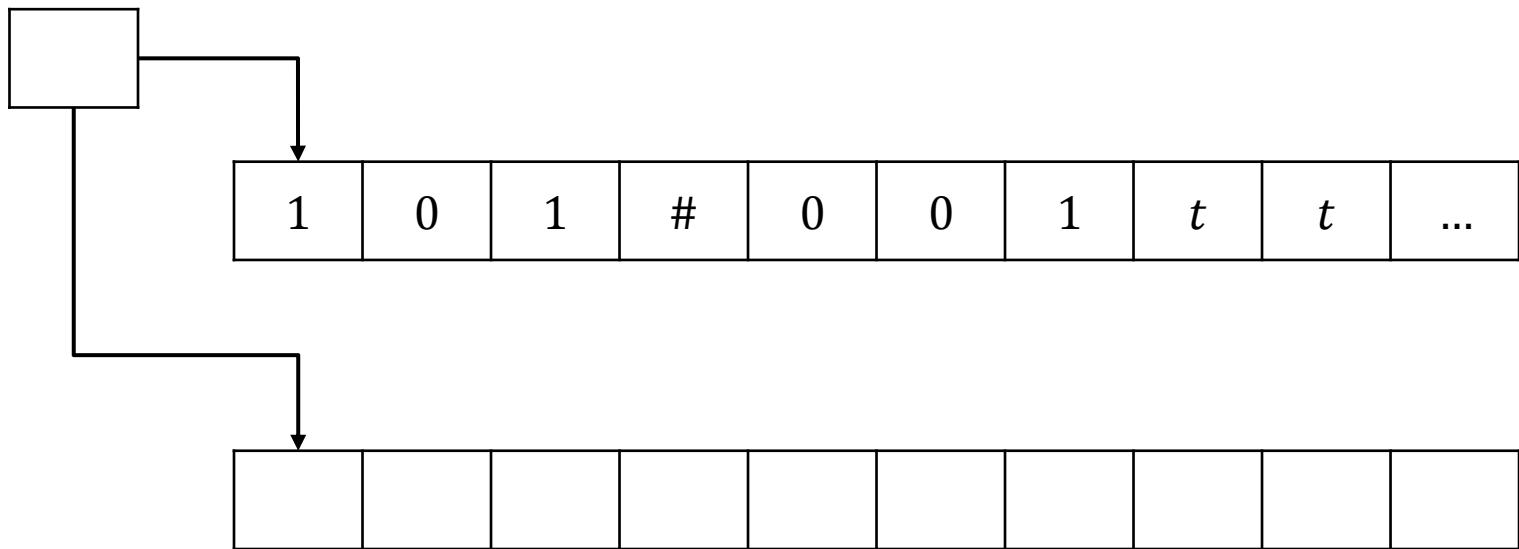
1. Falls das erste Eingabesymbol nicht # ist, lehne ab – sonst ersetze # durch #'.
Wenn kein weiteres # gefunden, akzeptiere.
2. Finde das nächste # und ersetze es durch #'.
Wird kein weiteres # gefunden, akzeptiere.
3. Teste, ob die beiden Folgen w_i, w_j rechts der Symbole #' gleich sind. Wenn ja, lehne ab.
4. Verschiebe Markierungen für den Vergleich des nächsten Paars von Folgen. Falls dieses Paar nicht mehr existiert, akzeptiere. Sonst gehe zu Schritt 3.

2. Berechenbarkeit

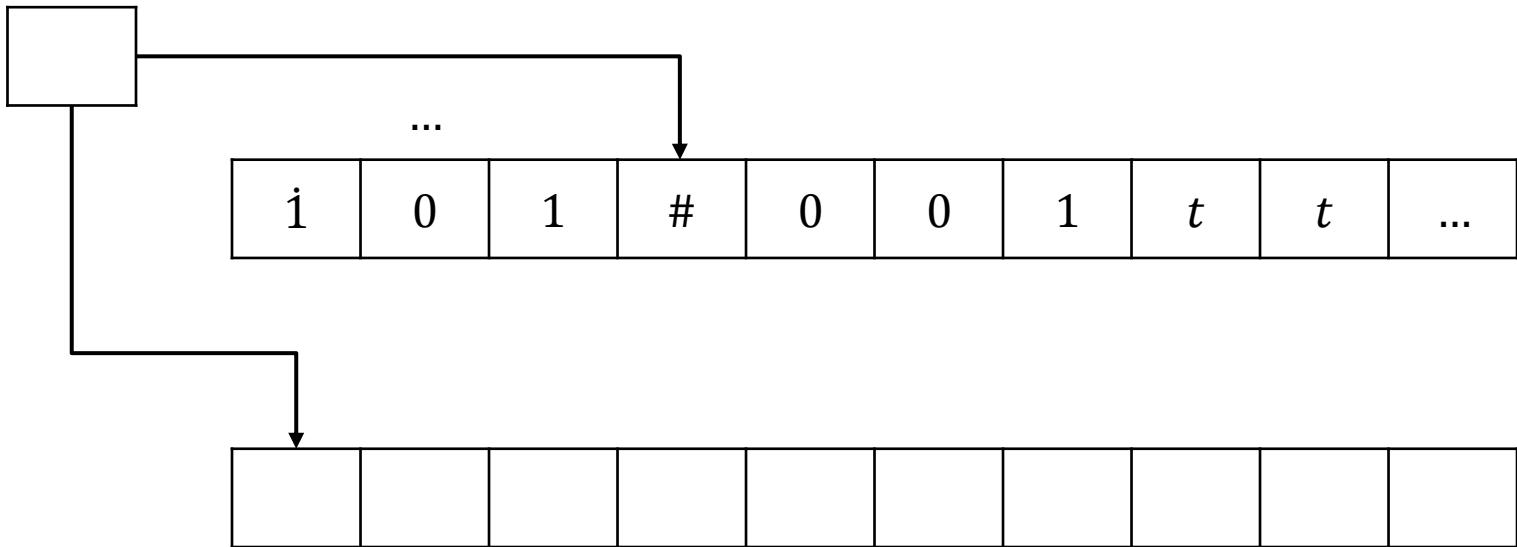
Berechnung, Akzeptieren, Entscheiden, ... k -Band Turingmaschinen

- **Beispiel:** Addition von zwei Binärzahlen
- **Eingabe:** $w_1 \# w_2$ für zwei Binärzahlen w_1 und w_2 (z.B. 100#1000 entspricht den Zahlen 4 und 8).
- **Ausgabe:** das Ergebnis $w_1 + w_2$ auf irgendeinem Band
- **Strategie:**
 - Verwende eine 3-Band Turing-Maschine
 - w_2 wird zunächst auf Band 2 geschrieben
 - w_1 und w_2 werden bitweise auf Band 3 zusammenaddiert.
Zum Schluss geht man in q_{accept} .

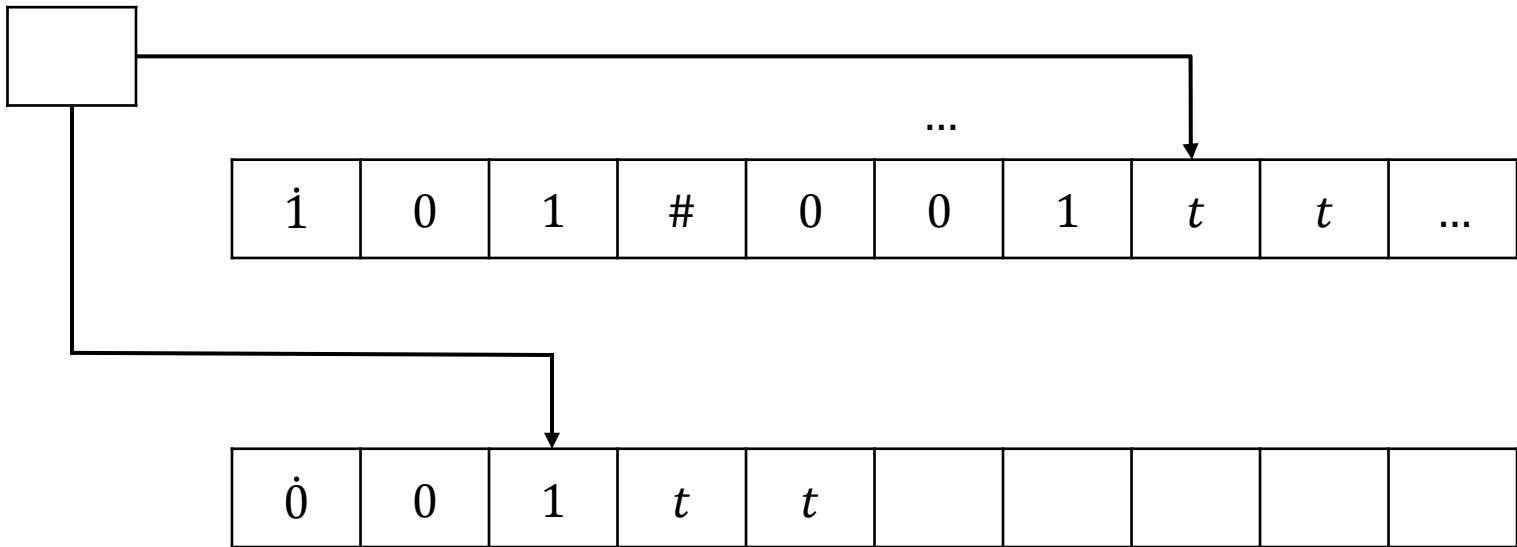
2. Berechenbarkeit



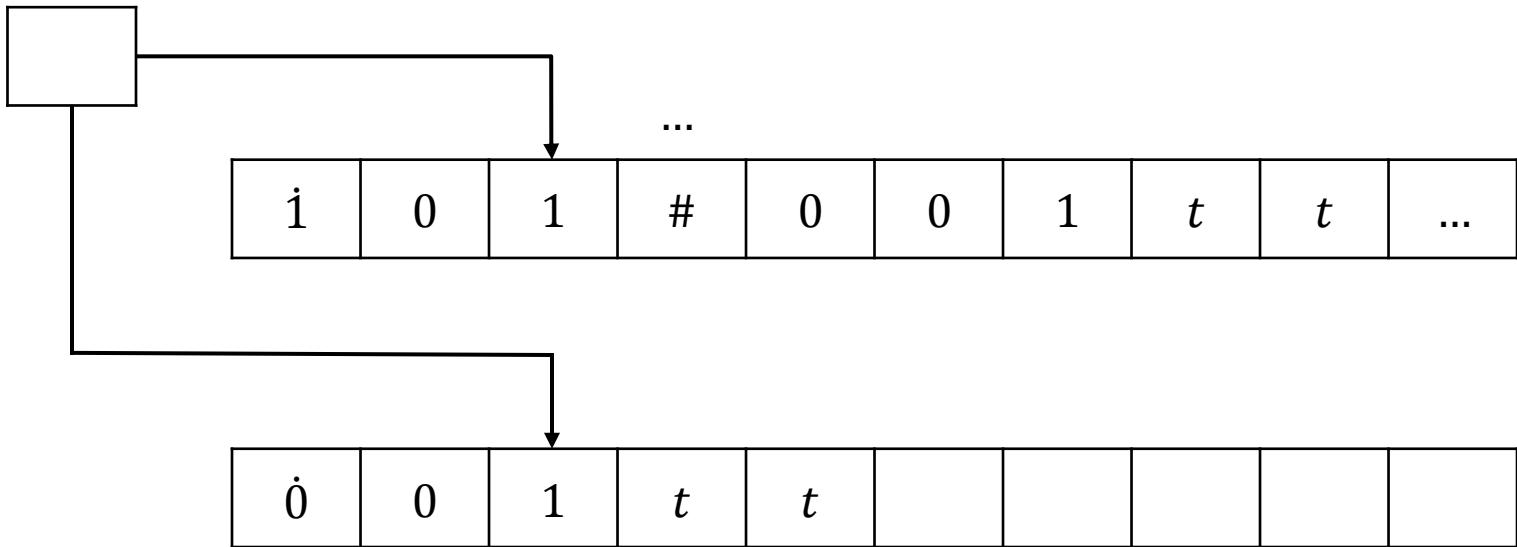
2. Berechenbarkeit



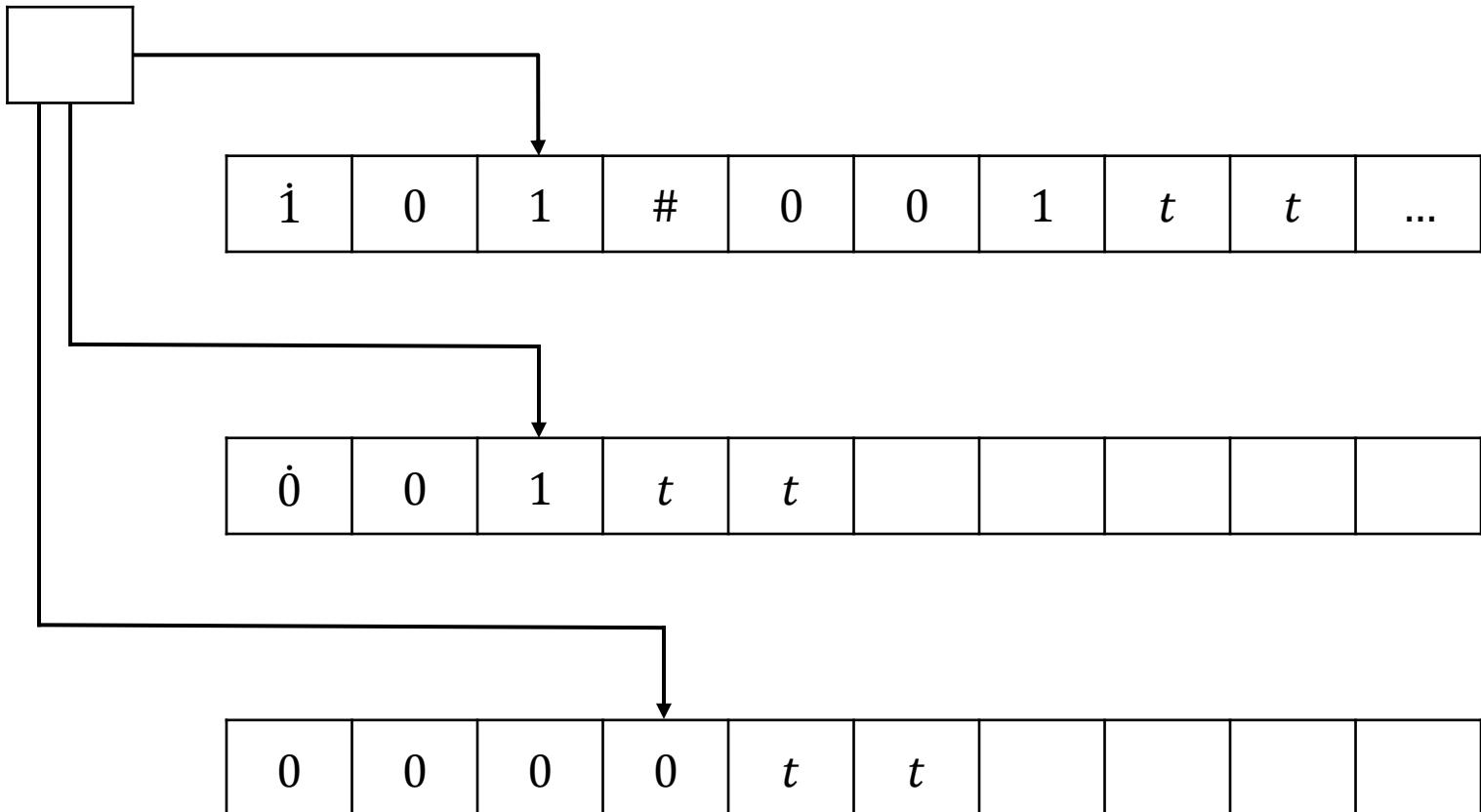
2. Berechenbarkeit



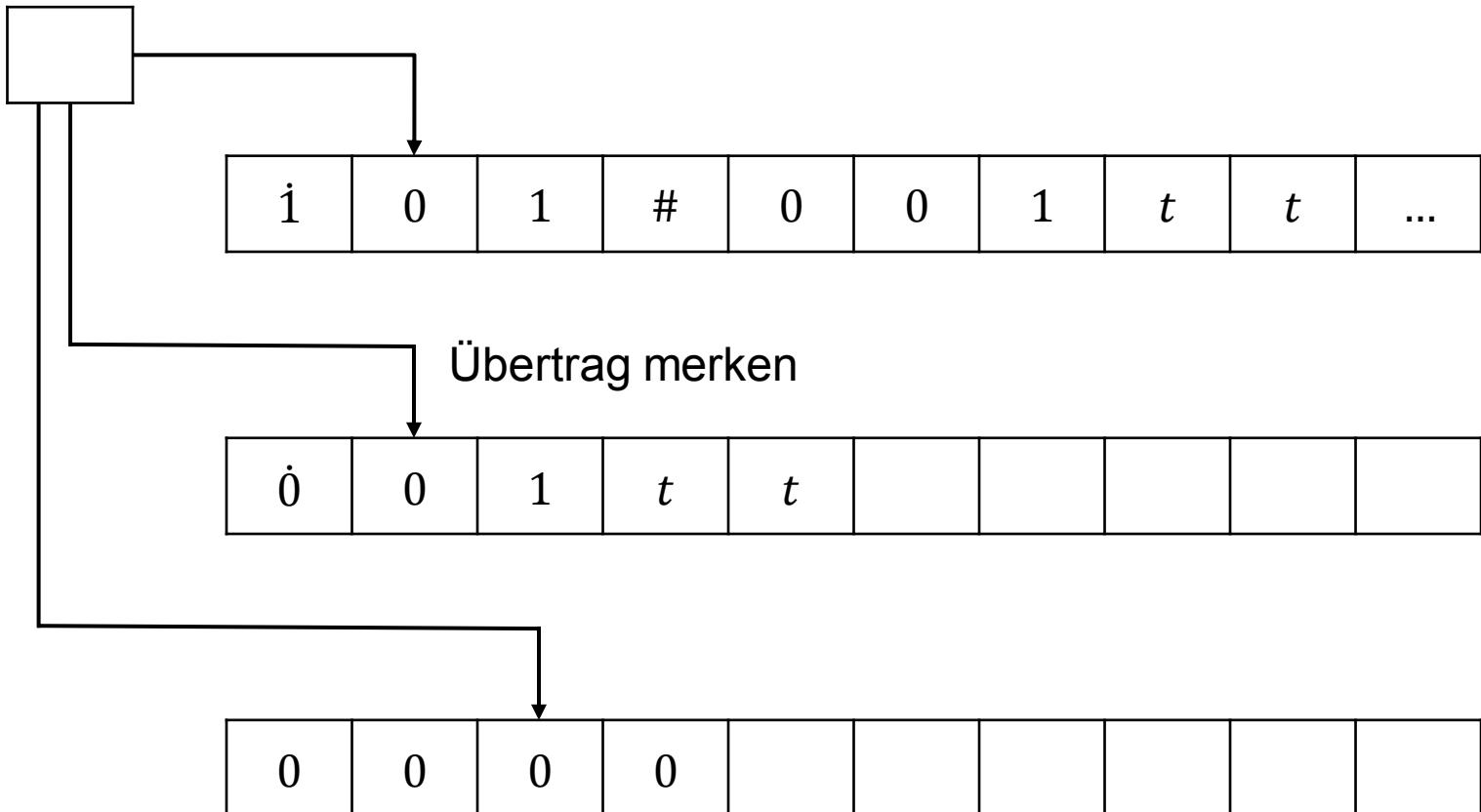
2. Berechenbarkeit



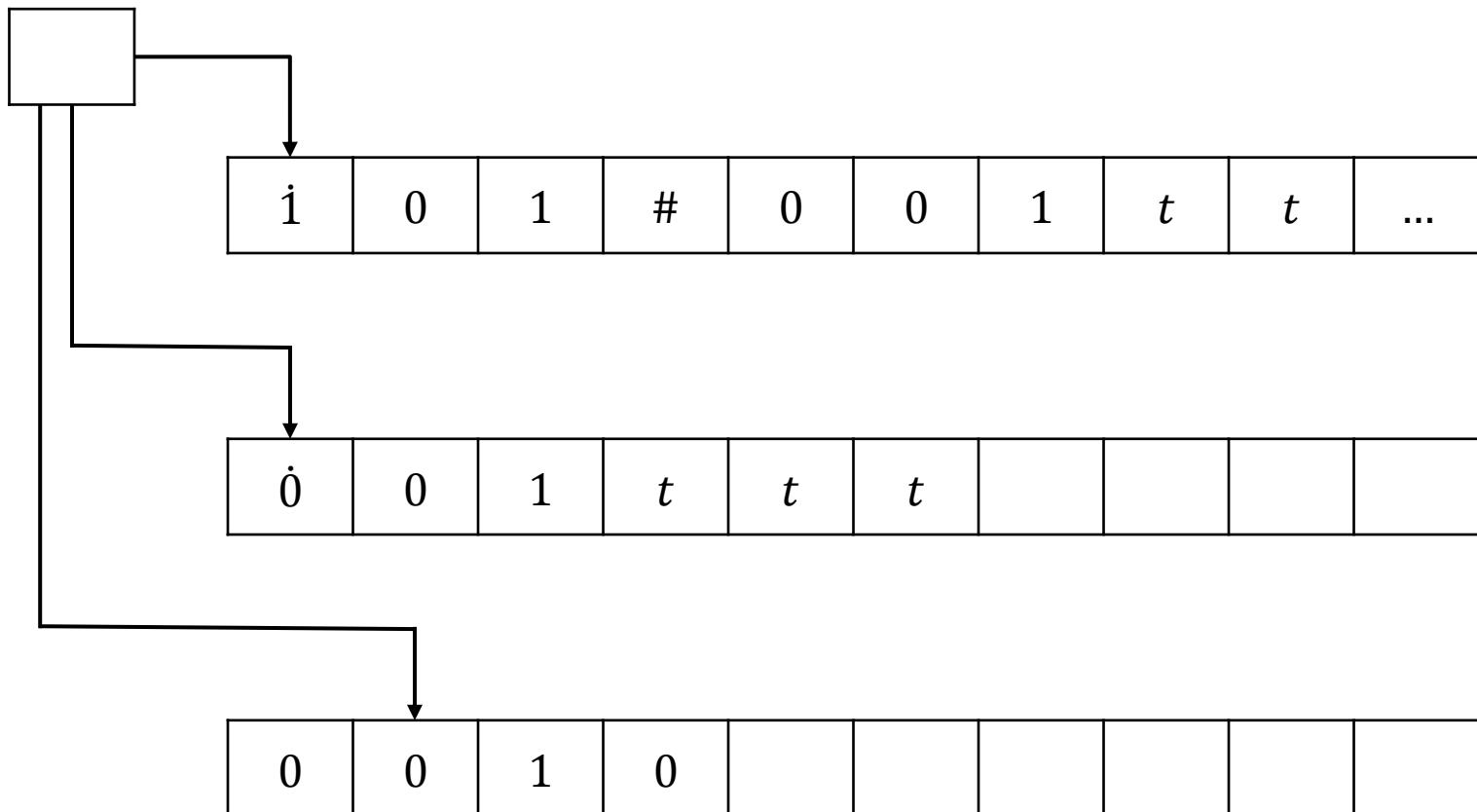
2. Berechenbarkeit



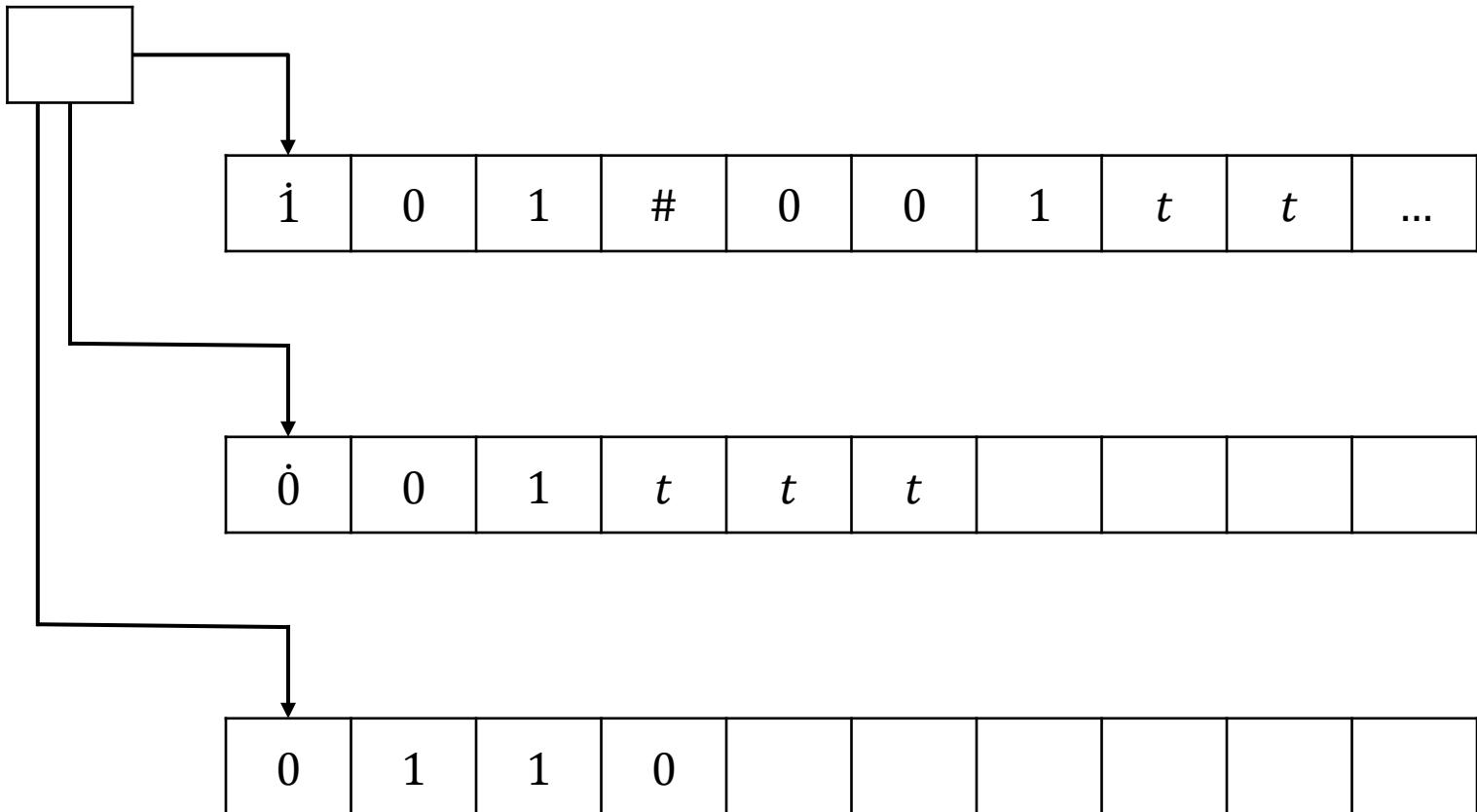
2. Berechenbarkeit



2. Berechenbarkeit



2. Berechenbarkeit



2. Berechenbarkeit

Church'sche These (1936)

Die im intuitiven Sinne berechenbaren Funktionen und Sprachen sind genau die, die durch Turingmaschinen berechenbar sind.

Warum sind Turingmaschinen ein geeignetes Modell?

- Menschliche Wahrnehmung ist endlich.
- Jeder realisierbare Rechner muss endlicher Natur sein und den physikalischen Gesetzen folgen.

2. Berechenbarkeit

Abschlusseigenschaften

\bar{L} : Komplementsprache zu $L - \bar{L} = \Sigma^* \setminus L$

Satz

Seien L_1 und L_2 entscheidbare Sprachen. Dann gilt:

1. $\overline{L_1}$ ist entscheidbar
2. $L_1 \cap L_2$ ist entscheidbar
3. $L_1 \cup L_2$ ist entscheidbar

Satz

Seien L_1 und L_2 rekursiv aufzählbare Sprachen. Dann gilt:

1. $L_1 \cap L_2$ ist rekursiv aufzählbar
2. $L_1 \cup L_2$ ist rekursiv aufzählbar

2. Berechenbarkeit

Abschlusseigenschaften

Satz

Eine Sprache L ist genau dann entscheidbar, wenn L und \bar{L} rekursiv aufzählbar sind.

2. Berechenbarkeit

- **Universelle Turingmaschinen**
 - Bislang *special purpose Computer*: eine Sprache – eine Turing-Maschine
 - Allgemein programmierbare Turing-Maschinen:
universelle Turing-Maschinen
 - Erhalten als Eingabe die Beschreibung einer Turingmaschine und simulieren diese Maschine
 - Benötigen dafür eine einheitliche Beschreibung von Turingmaschinen durch sog. *Gödel-Nummern*

2. Berechenbarkeit

Standardisierungen

- Betrachten nur 1-Band Turing-Maschinen
- Standardalphabet $\Sigma = \{0,1\}$, $\Gamma = \{0,1,t\}$
- andere Alphabete können durch Standardalphabete kodiert werden
- Turingmaschinen mit anderen Alphabeten können durch Turingmaschinen mit Standardalphabeten simuliert werden.

2. Berechenbarkeit

Definition Gödelnummern

Sei M eine 1-Band-Turingmaschine mit

$$Q = \{q_0, \dots, q_n\},$$

$$q_{accept} = q_{n-1},$$

$$q_{reject} = q_n.$$

Sei $X_1 = 0, X_2 = 1, X_3 = t, D_1 = L, D_2 = R$.

Wir kodieren $\delta(q_i, X_j) = (q_k, X_l, D_m)$ durch $0^{i+1}10^j10^{k+1}10^l10^m$.

$Code_r$: Kodierung des r -ten Eintrags für δ , $1 \leq r \leq 4(n - 1)$

Gödelnummer $\langle M \rangle = 111Code_111Code_211 \dots 11Code_g111$

2. Berechenbarkeit

Definition Universelle Turingmaschine

Eine Turingmaschine M_0 heißt universell, falls für jede 1-Band-Turingmaschine M und jedes x aus $\{0,1\}^*$ gilt:

- M_0 gestartet mit $\langle M \rangle x$ hält genau dann, wenn M gestartet mit x hält.
- M_0 akzeptiert $\langle M \rangle x$ genau dann, wenn M das Wort x akzeptiert.

Satz

Es gibt eine universelle 2-Band Turingmaschine.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

1. Einführung

Definition

Eine (*deterministische 1-Band Turingmaschine*) DTM wird beschrieben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Dabei sind Q, Σ, Γ endliche, nichtleere Mengen und es gilt:

- Σ ist Teilmenge von Γ
- t in $\Sigma \cap \Gamma$ ist das *Blanksymbol* (auch \sqcup)
- Q ist die *Zustandsmenge*
- Σ ist das *Eingabealphabet*
- Γ ist das *Bandalphabet*
- q_0 in Q ist der *Startzustand*
- q_{accept} in Q ist der akzeptierende Endzustand
- q_{reject} in Q ist der ablehnende Endzustand
- $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die (partielle) *Übergangsfunktion*. Sie ist für kein Argument aus $\{q_{accept}, q_{reject}\} \times \Gamma$ definiert.

1. Einführung

Momentaufnahme einer Turingmaschine:

- Bei Bandinschrift uv (dabei beginnt u am linken Ende des Bandes und hinter v stehen nur Blanks)
- Zustand q
- Kopf auf erstem Zeichen von v

Konfiguration $C = uqv$

1. Einführung

Definition

- Eine Sprache L heißt **rekursiv aufzählbar**, falls es eine Turingmaschine M gibt, die L akzeptiert.
- Eine Sprache L heißt **rekursiv** oder **entscheidbar**, falls es eine Turingmaschine M gibt, die L entscheidet.

1. Einführung

- Eine Mehrband- oder k -Band Turingmaschine (k -Band DTM) hat k Bänder mit je einem Kopf.
- Die Übergangsfunktion ist dann von der Form $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Zu Beginn steht die Eingabe auf Band 1, sonst stehen überall Blanks. Die Arbeitsweise ist analog zu 1-Band-DTMs definiert.

2. Berechenbarkeit

- **Universelle Turingmaschinen**
 - Bislang *special purpose Computer*: eine Sprache – eine Turing-Maschine
 - Allgemein programmierbare Turing-Maschinen:
universelle Turing-Maschinen
 - Erhalten als Eingabe die Beschreibung einer Turingmaschine und simulieren diese Maschine
 - Benötigen dafür eine einheitliche Beschreibung von Turingmaschinen durch sog. *Gödel-Nummern*

2. Berechenbarkeit

Definition Gödelnummern

Sei M eine 1-Band-Turingmaschine mit

$$Q = \{q_0, \dots, q_n\},$$

$$q_{accept} = q_{n-1},$$

$$q_{reject} = q_n.$$

Sei $X_1 = 0, X_2 = 1, X_3 = t, D_1 = L, D_2 = R$.

Wir kodieren $\delta(q_i, X_j) = (q_k, X_l, D_m)$ durch $0^{i+1}10^j10^{k+1}10^l10^m$.

$Code_r$: Kodierung des r -ten Eintrags für δ , $1 \leq r \leq 4(n - 1)$

Gödelnummer $\langle M \rangle = 111Code_111Code_211 \dots 11Code_g111$

Kurt Gödel



Quelle: www.numbersleuth.org

- Studium an der Universität Wien
- Dozenturen in Wien und Princeton
- Rennomiertester Preis in der theoretischen Informatik wird nach Gödel benannt

2. Berechenbarkeit

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

$$\Sigma = \{0,1\}$$

$$\Gamma = \Sigma \cup \{t\}$$

$$\delta(q_0, 0) = (q_{reject}, 0, R)$$

$$\delta(q_0, 1) = (q_0, 1, R)$$

$$\delta(q_0, t) = (q_{accept}, t, R)$$

$$L = \{1^n \mid n \geq 0\}$$

Gödel-Nummer:

1110101000101001101001010010011010001001000100111

2. Berechenbarkeit

Definition Universelle Turingmaschine

Eine Turingmaschine M_0 heißt **universell**, falls für jede 1-Band-Turingmaschine M und jedes x aus $\{0,1\}^*$ gilt:

- M_0 gestartet mit $\langle M \rangle x$ hält genau dann, wenn M gestartet mit x hält.
- M_0 akzeptiert $\langle M \rangle x$ genau dann, wenn M das Wort x akzeptiert.

Satz

Es gibt eine universelle 2-Band Turingmaschine.

2. Berechenbarkeit

Die Sprache Gödel:

Sprache Gödel := { w aus $\{0,1\}^*$ | w ist die Gödel–Nummer einer DTM}

Lemma

Die Sprache Gödel ist entscheidbar.

Die Sprache States:

Sprache States := { $(\langle M \rangle, d)$ | M besitzt mindestens d Zustände}

Lemma

Die Sprache States ist entscheidbar.

2. Berechenbarkeit

Das Halteproblem

$H := \{(\langle M \rangle, x) \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

Satz —

Das Halteproblem ist rekursiv aufzählbar.

2. Berechenbarkeit

Die Sprache Useful

$\text{Useful} := \left\{ (\langle M \rangle, q) \mid M \text{ ist DTM mit Zustand } q, \text{ und es gibt eine Eingabe } w, \text{ so dass } M \text{ gestartet mit } w \text{ in den Zustand } q \text{ gerät} \right\}$

Satz

Useful ist rekursiv aufzählbar.

2. Berechenbarkeit

Aufzählung von binären Eingabefolgen:

- für alle natürlichen Zahlen i sei $w_i = w$, falls $\text{bin}(i) = 1w$
- damit werden alle möglichen w aus $\{0,1\}^*$ aufgezählt

Aufzählung von Turingmaschinen:

M_i ist:

- M_{reject} , falls i keine Gödelnummer ist
- M , falls $\text{bin}(i)$ die Gödelnummer der DTM M ist, d.h. $\langle M \rangle = \text{bin}(i)$

2. Berechenbarkeit

Die Sprache Diag

$\text{Diag} := \{w \text{ in } \{0,1\}^* \mid w = w_i \text{ und die DTM } M_i \text{ akzeptiert } w \text{ nicht}\}$

Satz

Die Sprache Diag ist nicht rekursiv aufzählbar.

2. Berechenbarkeit

	M_1	M_2	M_3	M_7	M_i
w_1	na	na	na		na		na	
w_2	na	na	na		na		na	
w_3	na	na	na		na		na	
:								
w_7	na	na	na		na		a	
:								
w_i	na	na	na		na		a	

Diagonale

Tabelle für Akzeptanz/Nichtakzeptanz von DTMs

Quelle: Skript Blömer

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

1. Einführung

Definition

Eine (*deterministische 1-Band Turingmaschine*) DTM wird beschrieben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Dabei sind Q, Σ, Γ endliche, nichtleere Mengen und es gilt:

- Σ ist Teilmenge von Γ
- t in Γ ist das *Blanksymbol* (auch \sqcup)
- Q ist die *Zustandsmenge*
- Σ ist das *Eingabealphabet*
- Γ ist das *Bandalphabet*
- q_0 in Q ist der *Startzustand*
- q_{accept} in Q ist der akzeptierende Endzustand
- q_{reject} in Q ist der ablehnende Endzustand
- $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die (partielle) *Übergangsfunktion*.
Sie ist für kein Argument aus $\{q_{accept}, q_{reject}\} \times \Gamma$ definiert.

1. Einführung

Momentaufnahme einer Turingmaschine:

- Bei Bandinschrift uv (dabei beginnt u am linken Ende des Bandes und hinter v stehen nur Blanks)
- Zustand q
- Kopf auf erstem Zeichen von v

Konfiguration $C = uqv$

1. Einführung

Definition

- Eine Sprache L heißt **rekursiv aufzählbar**, falls es eine Turingmaschine M gibt, die L akzeptiert.
- Eine Sprache L heißt **rekursiv** oder **entscheidbar**, falls es eine Turingmaschine M gibt, die L entscheidet.

1. Einführung

- Eine Mehrband- oder k -Band Turingmaschine (k -Band DTM) hat k Bänder mit je einem Kopf.
- Die Übergangsfunktion ist dann von der Form $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Zu Beginn steht die Eingabe auf Band 1, sonst stehen überall Blanks. Die Arbeitsweise ist analog zu 1-Band-DTMs definiert.

2. Berechenbarkeit

- **Universelle Turingmaschinen**
 - Bislang *special purpose Computer*: eine Sprache – eine Turing-Maschine
 - Allgemein programmierbare Turing-Maschinen:
universelle Turing-Maschinen
 - Erhalten als Eingabe die Beschreibung einer Turingmaschine und simulieren diese Maschine
 - Benötigen dafür eine einheitliche Beschreibung von Turingmaschinen durch sog. *Gödel-Nummern*

2. Berechenbarkeit

Definition Gödelnummern

Sei M eine 1-Band-Turingmaschine mit

$$Q = \{q_0, \dots, q_n\},$$

$$q_{accept} = q_{n-1},$$

$$q_{reject} = q_n.$$

Sei $X_1 = 0, X_2 = 1, X_3 = t, D_1 = L, D_2 = R$.

Wir kodieren $\delta(q_i, X_j) = (q_k, X_l, D_m)$ durch $0^{i+1}10^j10^{k+1}10^l10^m$.

$Code_r$: Kodierung des r -ten Eintrags für δ , $1 \leq r \leq 4(n - 1)$

Gödelnummer $\langle M \rangle = 111Code_111Code_211 \dots 11Code_g111$

2. Berechenbarkeit

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

$$\Sigma = \{0,1\}$$

$$\Gamma = \Sigma \cup \{t\}$$

$$\delta(q_0, 0) = (q_{reject}, 0, R)$$

$$\delta(q_0, 1) = (q_0, 1, R)$$

$$\delta(q_0, t) = (q_{accept}, t, R)$$

$$L = \{1^n \mid n \geq 0\}$$

Gödel-Nummer:

1110101000101001101001010010011010001001000100111

2. Berechenbarkeit

Definition Universelle Turingmaschine

Eine Turingmaschine M_0 heißt **universell**, falls für jede 1-Band-Turingmaschine M und jedes x aus $\{0,1\}^*$ gilt:

- M_0 gestartet mit $\langle M \rangle x$ hält genau dann, wenn M gestartet mit x hält.
- M_0 akzeptiert $\langle M \rangle x$ genau dann, wenn M das Wort x akzeptiert.

Satz

Es gibt eine universelle 2-Band Turingmaschine.

2. Berechenbarkeit

Die Sprache Gödel:

Sprache Gödel := { w aus $\{0,1\}^*$ | w ist die Gödel–Nummer einer DTM}

Lemma —

Die Sprache Gödel ist entscheidbar.

Die Sprache States:

Sprache States := { $(\langle M \rangle, d)$ | M besitzt mindestens d Zustände}

Lemma —

Die Sprache States ist entscheidbar.

2. Berechenbarkeit

Das Halteproblem

$H := \{(\langle M \rangle, x) \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

Satz —

Das Halteproblem ist rekursiv aufzählbar.

2. Berechenbarkeit

Die Sprache Useful

$\text{Useful} := \left\{ (\langle M \rangle, q) \mid M \text{ ist DTM mit Zustand } q, \text{ und es gibt eine Eingabe } w, \right. \\ \left. \text{so dass } M \text{ gestartet mit } w \text{ in den Zustand } q \text{ gerät} \right\}$

Satz

Die Sprache Useful ist rekursiv aufzählbar.

2. Berechenbarkeit

Aufzählung von binären Eingabefolgen:

- für alle natürlichen Zahlen i sei $w_i = w$, falls $\text{bin}(i) = 1w$
- damit werden alle möglichen w aus $\{0,1\}^*$ aufgezählt

Aufzählung von Turingmaschinen:

M_i ist:

- M_{reject} , falls i keine Gödelnummer ist
- M , falls $\text{bin}(i)$ die Gödelnummer der DTM M ist, d.h. $\langle M \rangle = \text{bin}(i)$

2. Berechenbarkeit

Die Sprache Diag

$\text{Diag} := \{w \text{ in } \{0,1\}^* \mid w = w_i \text{ und die DTM } M_i \text{ akzeptiert } w \text{ nicht}\}$

Satz

Die Sprache Diag ist **nicht** rekursiv aufzählbar.

2. Berechenbarkeit

	M_1	M_2	M_3	M_7	M_i
w_1	na	na	na		na		na	
w_2	na	na	na		na		na	
w_3	na	na	na		na		na	
:								
w_7	na	na	na		na		a	
:								
w_i	na	na	na		na		a	

Diagonale

Tabelle für Akzeptanz/Nichtakzeptanz von DTMs

Quelle: Skript Johannes Blömer, Universität Paderborn

2. Berechenbarkeit

Reduktionen

Formalisierung von

- Sprache A ist nicht schwerer als Sprache B

Idee

- Algorithmus/DTM für B kann genutzt werden, um A zu akzeptieren/entscheiden.

2. Berechenbarkeit

Zwei einfache Sprachen

$P := \{w \text{ in } \{0,1\}^* \mid w \text{ ist ein Palindrom}\}$

$XOR := \{(a, b, c) \text{ in } \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \mid a, b, c \text{ haben die gleiche Länge und } a \oplus b = c\}$

$f: \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$

$w \rightarrow (w, w^R, 0^{|w|})$

2. Berechenbarkeit

Von XOR und f zu P

M_P bei Eingabe w

1. Berechne mit M_f das Tripel $f(w) = (w, w^R, 0^{|w|})$.
2. Simuliere M_{XOR} mit Eingabe $f(w)$.
3. Falls M_{XOR} die Eingabe $f(w)$ akzeptiert, akzeptiere w .
4. Falls M_{XOR} die Eingabe $f(w)$ ablehnt, lehne w ab.

M_{XOR} entscheidet XOR , M_f berechnet f .

2. Berechenbarkeit

Definition Reduktionen

L' heißt reduzierbar auf L , falls es eine Funktion $f: \{0,1\}^* \rightarrow \{0,1\}^*$ gibt mit

1. Für alle w aus $\{0,1\}^*$ gilt:
 w ist in L' genau dann, wenn $f(w)$ in L
2. Funktion f ist berechenbar, d.h., es gibt eine DTM M_f , die die Funktion f berechnet.

f heißt Reduktion von L' auf L , geschrieben $L' \leq L$.

2. Berechenbarkeit

Definition

Eine DTM M berechnet die Funktion $f: \Sigma^* \rightarrow \Gamma$, falls für alle w aus Σ^* die Berechnung von M mit Eingabe w in einer akzeptierenden Konfiguration hält und dabei der Bandinhalt $f(w)$ ist.

Hierbei werden \triangleright und alle t ignoriert.

Lemma

Seien L' und L Sprachen mit $L' \leq L$. Dann gilt:

1. Ist L entscheidbar, so ist auch L' entscheidbar.
2. Ist L rekursiv aufzählbar, so ist auch L' rekursiv aufzählbar.

2. Berechenbarkeit

Von L und f zu L'

M' bei Eingabe w

1. Berechne mit M_f die Folge $f(w)$.
2. Simuliere M mit Eingabe $f(w)$.
3. Falls M die Eingabe $f(w)$ akzeptiert, akzeptiere w .
4. Falls M die Eingabe $f(w)$ ablehnt, lehne w ab.

2. Berechenbarkeit

Akzeptanz- und Halteproblem

$H := \{\langle M \rangle x \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

$A := \{\langle M \rangle x \mid M \text{ ist DTM, die die Eingabe } x \text{ akzeptiert}\}$

Lemma

Das Halteproblem kann auf das Akzeptanzproblem reduziert werden.

$$H \leq A$$

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

1. Einführung

Definition

Eine (*deterministische 1-Band Turingmaschine*) DTM wird beschrieben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.

Dabei sind Q, Σ, Γ endliche, nichtleere Mengen und es gilt:

- Σ ist Teilmenge von Γ
- t in $\Sigma \cap \Gamma$ ist das *Blanksymbol* (auch \sqcup)
- Q ist die *Zustandsmenge*
- Σ ist das *Eingabealphabet*
- Γ ist das *Bandalphabet*
- q_0 in Q ist der *Startzustand*
- q_{accept} in Q ist der akzeptierende Endzustand
- q_{reject} in Q ist der ablehnende Endzustand
- $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ist die (partielle) *Übergangsfunktion*. Sie ist für kein Argument aus $\{q_{accept}, q_{reject}\} \times \Gamma$ definiert.

1. Einführung

Definition

- Eine Sprache L heißt **rekursiv aufzählbar**, falls es eine Turingmaschine M gibt, die L akzeptiert.
- Eine Sprache L heißt **rekursiv** oder **entscheidbar**, falls es eine Turingmaschine M gibt, die L entscheidet.

1. Einführung

- Eine Mehrband- oder k -Band Turingmaschine (k -Band DTM) hat k Bänder mit je einem Kopf.
- Die Übergangsfunktion ist dann von der Form $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$
- Zu Beginn steht die Eingabe auf Band 1, sonst stehen überall Blanks. Die Arbeitsweise ist analog zu 1-Band-DTMs definiert.

2. Berechenbarkeit

- **Universelle Turingmaschinen**
 - Bislang *special purpose Computer*: eine Sprache – eine Turing-Maschine
 - Allgemein programmierbare Turing-Maschinen:
universelle Turing-Maschinen
 - Erhalten als Eingabe die Beschreibung einer Turingmaschine und simulieren diese Maschine
 - Benötigen dafür eine einheitliche Beschreibung von Turingmaschinen durch sog. *Gödel-Nummern*

2. Berechenbarkeit

Definition Gödelnummern

Sei M eine 1-Band-Turingmaschine mit

$$Q = \{q_0, \dots, q_n\},$$

$$q_{accept} = q_{n-1},$$

$$q_{reject} = q_n.$$

Sei $X_1 = 0, X_2 = 1, X_3 = t, D_1 = L, D_2 = R$.

Wir kodieren $\delta(q_i, X_j) = (q_k, X_l, D_m)$ durch $0^{i+1}10^j10^{k+1}10^l10^m$.

$Code_r$: Kodierung des r -ten Eintrags für δ , $1 \leq r \leq 4(n - 1)$

Gödelnummer $\langle M \rangle = 111Code_111Code_211 \dots 11Code_g111$

2. Berechenbarkeit

Definition Universelle Turingmaschine

Eine Turingmaschine M_0 heißt **universell**, falls für jede 1-Band-Turingmaschine M und jedes x aus $\{0,1\}^*$ gilt:

- M_0 gestartet mit $\langle M \rangle x$ hält genau dann, wenn M gestartet mit x hält.
- M_0 akzeptiert $\langle M \rangle x$ genau dann, wenn M das Wort x akzeptiert.

Satz

Es gibt eine universelle 2-Band Turingmaschine.

2. Berechenbarkeit

Die Sprache Gödel:

Sprache Gödel := { w aus $\{0,1\}^*$ | w ist die Gödel–Nummer einer DTM}

Lemma —

Die Sprache Gödel ist entscheidbar.

Die Sprache States:

Sprache States := { $(\langle M \rangle, d)$ | M besitzt mindestens d Zustände}

Lemma —

Die Sprache States ist entscheidbar.

2. Berechenbarkeit

Das Halteproblem

$H := \{(\langle M \rangle, x) \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

Satz

Das Halteproblem ist rekursiv aufzählbar.

2. Berechenbarkeit

Die Sprache Useful

$\text{Useful} := \left\{ (\langle M \rangle, q) \mid M \text{ ist DTM mit Zustand } q, \text{ und es gibt eine Eingabe } w, \right. \\ \left. \text{so dass } M \text{ gestartet mit } w \text{ in den Zustand } q \text{ gerät} \right\}$

Satz

Die Sprache Useful ist rekursiv aufzählbar.

2. Berechenbarkeit

Aufzählung von binären Eingabefolgen:

- für alle natürlichen Zahlen i sei $w_i = w$, falls $\text{bin}(i) = 1w$
- damit werden alle möglichen w aus $\{0,1\}^*$ aufgezählt

Aufzählung von Turingmaschinen:

M_i ist:

- M_{reject} , falls i keine Gödelnummer ist
- M , falls $\text{bin}(i)$ die Gödelnummer der DTM M ist, d.h. $\langle M \rangle = \text{bin}(i)$

2. Berechenbarkeit

Die Sprache Diag

$\text{Diag} := \{w \text{ in } \{0,1\}^* \mid w = w_i \text{ und die DTM } M_i \text{ akzeptiert } w \text{ nicht}\}$

Satz

Die Sprache Diag ist **nicht** rekursiv aufzählbar.

2. Berechenbarkeit

Reduktionen

Formalisierung von

- Sprache A ist nicht schwerer als Sprache B

Idee

- Algorithmus/DTM für B kann genutzt werden, um A zu akzeptieren/entscheiden.

2. Berechenbarkeit

Definition Reduktionen

L' heißt reduzierbar auf L , falls es eine Funktion $f: \{0,1\}^* \rightarrow \{0,1\}^*$ gibt mit

1. Für alle w aus $\{0,1\}^*$ gilt:
 w ist in L' genau dann, wenn $f(w)$ in L
2. Funktion f ist berechenbar, d.h., es gibt eine DTM M_f , die die Funktion f berechnet.

f heißt Reduktion von L' auf L , geschrieben $L' \leq L$.

2. Berechenbarkeit

Definition

Eine DTM M berechnet die Funktion $f: \Sigma^* \rightarrow \Gamma$, falls für alle w aus Σ^* die Berechnung von M mit Eingabe w in einer akzeptierenden Konfiguration hält und dabei der Bandinhalt $f(w)$ ist.

Hierbei werden \triangleright und alle t ignoriert.

Lemma

Seien L' und L Sprachen mit $L' \leq L$. Dann gilt:

1. Ist L entscheidbar, so ist auch L' entscheidbar.
2. Ist L rekursiv aufzählbar, so ist auch L' rekursiv aufzählbar.

2. Berechenbarkeit

Lemma

Seien L' und L Sprachen mit $L' \leq L$. Dann gilt:

1. Ist L entscheidbar, so ist auch L' entscheidbar.
2. Ist L rekursiv aufzählbar, so ist auch L' rekursiv aufzählbar.

Korollar

Seien L' und L Sprachen mit $L' \leq L$. Dann gilt:

1. Ist L' nicht entscheidbar, so ist auch L nicht entscheidbar.
2. Ist L' nicht rekursiv aufzählbar, so ist auch L nicht rekursiv aufzählbar.

2. Berechenbarkeit

Von L und f zu L'

M' bei Eingabe w

1. Berechne mit M_f die Folge $f(w)$.
2. Simuliere M mit Eingabe $f(w)$.
3. Falls M die Eingabe $f(w)$ akzeptiert, akzeptiere w .
4. Falls M die Eingabe $f(w)$ ablehnt, lehne w ab.

2. Berechenbarkeit

Akzeptanz- und Halteproblem

$H := \{\langle M \rangle x \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

$A := \{\langle M \rangle x \mid M \text{ ist DTM, die die Eingabe } x \text{ akzeptiert}\}$

Lemma

Das Halteproblem kann auf das Akzeptanzproblem reduziert werden.

$$H \leq A$$

2. Berechenbarkeit

Akzeptanzproblem und die Sprache Useful

$A := \{\langle M \rangle x \mid M \text{ ist DTM, die die Eingabe } x \text{ akzeptiert}\}$

$\text{Useful} := \left\{ (\langle M \rangle, q) \mid M \text{ ist DTM mit Zustand } q, \text{ und es gibt eine Eingabe } w, \right.$
 $\left. \text{so dass } M \text{ gestartet mit } w \text{ in den Zustand } q \text{ gerät} \right\}$

Lemma

Das Akzeptanzproblem kann auf die Sprache Useful reduziert werden.

$$A \leq \text{Useful}$$

2. Berechenbarkeit

Halteproblem

$H := \{\langle M \rangle x \mid M \text{ ist DTM, die gestartet mit Eingabe } x \text{ hält}\}$

■ **Satz** —

Das Halteproblem ist nicht entscheidbar.

2. Berechenbarkeit

Das Komplement des Halteproblems

$$\overline{H} := \left\{ \begin{array}{l} w \text{ aus } \{0,1\}^* \mid w \text{ ist nicht von der Form } \langle M \rangle x \text{ für eine DTM } M, \text{ oder} \\ w = \langle M \rangle x, \text{ wobei } M \text{ gestartet mit Eingabe } x \text{ nicht hält} \end{array} \right\}$$

Korollar

Das Komplement des Halteproblems ist nicht rekursiv aufzählbar.

Korollar

Die Klasse der rekursiv aufzählbaren Sprachen ist von der Klasse der entscheidbaren Sprachen verschieden und nicht gegen Komplementbildung abgeschlossen.

2. Berechenbarkeit

Akzeptanzproblem und die Sprache Useful

$A := \{\langle M \rangle x \mid M \text{ ist DTM, die die Eingabe } x \text{ akzeptiert}\}$

$\text{Useful} := \left\{ (\langle M \rangle, q) \mid M \text{ ist DTM mit Zustand } q, \text{ und es gibt eine Eingabe } w, \right.$
 $\left. \text{so dass } M \text{ gestartet mit } w \text{ in den Zustand } q \text{ gerät} \right\}$

Satz

Das Akzeptanzproblem A und die Sprache Useful sind nicht entscheidbar.

2. Berechenbarkeit

Halteproblem mit leerem Band

$H_0 := \{\langle M \rangle \mid M \text{ ist DTM, die gestartet mit Eingabe } \varepsilon \text{ hält}\}$

Satz

Das Halteproblem mit leerem Band H_0 ist nicht entscheidbar.

2. Berechenbarkeit

Totalitätsproblem

$T_o := \{\langle M \rangle \mid M \text{ hält bei jeder Eingabe}\}$

Endlichkeitsproblem

$E_o := \{\langle M \rangle \mid M \text{ hält für endlich viele Eingaben}\}$

Äquivalenzproblem

$Q_o := \{\langle M \rangle, \langle M' \rangle \mid M \text{ und } M' \text{ akzeptieren die gleiche Sprache}\}$

Satz

Das Äquivalenzproblem und das Totalitätsproblem sind nicht rekursiv aufzählbar.

2. Berechenbarkeit

Der Satz von Rice

Satz

Sei \mathcal{R} die Menge aller berechenbaren Funktionen und sei S eine nicht-triviale Teilmenge von \mathcal{R} . Dann ist die Sprache

$$L(S) := \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } S\}$$

nicht entscheidbar.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Inhaltsangabe

- Einleitung, Motivation
 - Turingmaschinen
 - Arbeitstechniken
- }
- Einführung**
-
- Unentscheidbare Probleme
 - Das Halteproblem
 - Reduktionen
- }
- Berechenbarkeit**
-
- Zeitkomplexität
 - Die Klassen P und NP
 - NP-Vollständigkeit
 - NP-vollständige Probleme
- }
- Komplexität**
-
- Formale Sprachen und Automaten
 - Kellerautomaten und kontextfreie Sprachen
 - Kontextsensitive Sprachen
- }
- Formale Sprachen**

Wiederholung

Gibt es einen Algorithmus HALTE, der

- als Eingabe einen beliebigen Algorithmus ALG und eine Eingabe w für ALG erhält und
- entscheidet, ob ALG bei Eingabe w hält?

Satz von Turing:

Einen solchen Algorithmus kann es nicht geben.

Wiederholung

- Alle sinnvollen Rechenmodelle liefern aus unserer Sicht die gleichen Ergebnisse
 - **Churchsche These**
- Es gibt Probleme, die algorithmisch nicht gelöst werden können (z.B. das Halteproblem)
 - **Berechenbarkeit**
- Manche Problem können zwar algorithmisch, aber nicht effizient gelöst werden (z.B. das Problem des Handlungsreisenden)
 - **Komplexität**

3. Komplexität

Von Berechenbarkeit zu Komplexität

- Berechenbarkeit in Bezug auf Entscheidbarkeit und Aufzählbarkeit betrachtet nur prinzipielle Lösbarkeit von Problemen mit Hilfe von Computern
- Prinzipiell lösbare Probleme können praktisch nicht lösbar sein, weil jeder Algorithmus zur Lösung des Problems zu viel Zeit (und/oder Platz) benötigt.
- Komplexitätstheorie versucht Probleme gemäß des Zeit- und Platzbedarfs des besten Algorithmus zu ihrer Lösung zu klassifizieren.
- Konzentrieren uns auf Zeitbedarf und die Klassen P und NP

3. Komplexität

Beispielprobleme:

- Minimum-Suche
 - Eingabearray $A[1 \dots n]$
 - Ausgabe: Minimum in $A[1 \dots n]$
- Sortierproblem
 - Eingabe: Folge von n Zahlen (a_1, \dots, a_n)
 - Ausgabe: Umordnung der Zahlen in (b_1, \dots, b_n) , sodass $b_1 \leq \dots \leq b_n$

Wiederholung

- **Problem des Handlungsreisenden:**
 - Wien, Krems, St. Pölten, Wiener Neustadt, Mürzzuschlag, Graz, Wolfsberg, Klagenfurt, Villach, Spital, Lienz, Brenner, Innsbruck, Kitzbühel, Bischofshofen, Hallein, Salzburg, Braunau, Ried, Wels, Linz, Enns, Amstetten, Zwettl, Stockerau



O-Notation

Definition

Sei $g: N \rightarrow R^+$ eine Funktion.

Dann bezeichnen wir mit $O(g(n))$ die folgende Menge von Funktionen:

$$O(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c > 0, n_0, \text{ sodass} \\ \text{für alle } n \geq n_0 \text{ gilt } 0 \leq f(n) \leq c g(n) \end{array} \right\}$$

- $O(g(n))$ formalisiert:
Die Funktion $f(n)$ wächst asymptotisch nicht schneller als $g(n)$.
- Statt $f(n)$ in $O(g(n))$ in der Regel $f(n) = O(g(n))$.

Ω -Notation

Definition

Sei $g: N \rightarrow R^+$ eine Funktion.

Dann bezeichnen wir mit $\Omega(g(n))$ die folgende Menge von Funktionen:

$$\Omega(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c > 0, n_0, \text{ sodass} \\ \text{für alle } n \geq n_0 \text{ gilt } 0 \leq c g(n) \leq f(n) \end{array} \right\}$$

- $\Omega(g(n))$ formalisiert:
Die Funktion $f(n)$ wächst asymptotisch mindestens so schnell wie $g(n)$.
- Statt $f(n)$ in $\Omega(g(n))$ in der Regel $f(n) = \Omega(g(n))$.

Θ-Notation

Definition

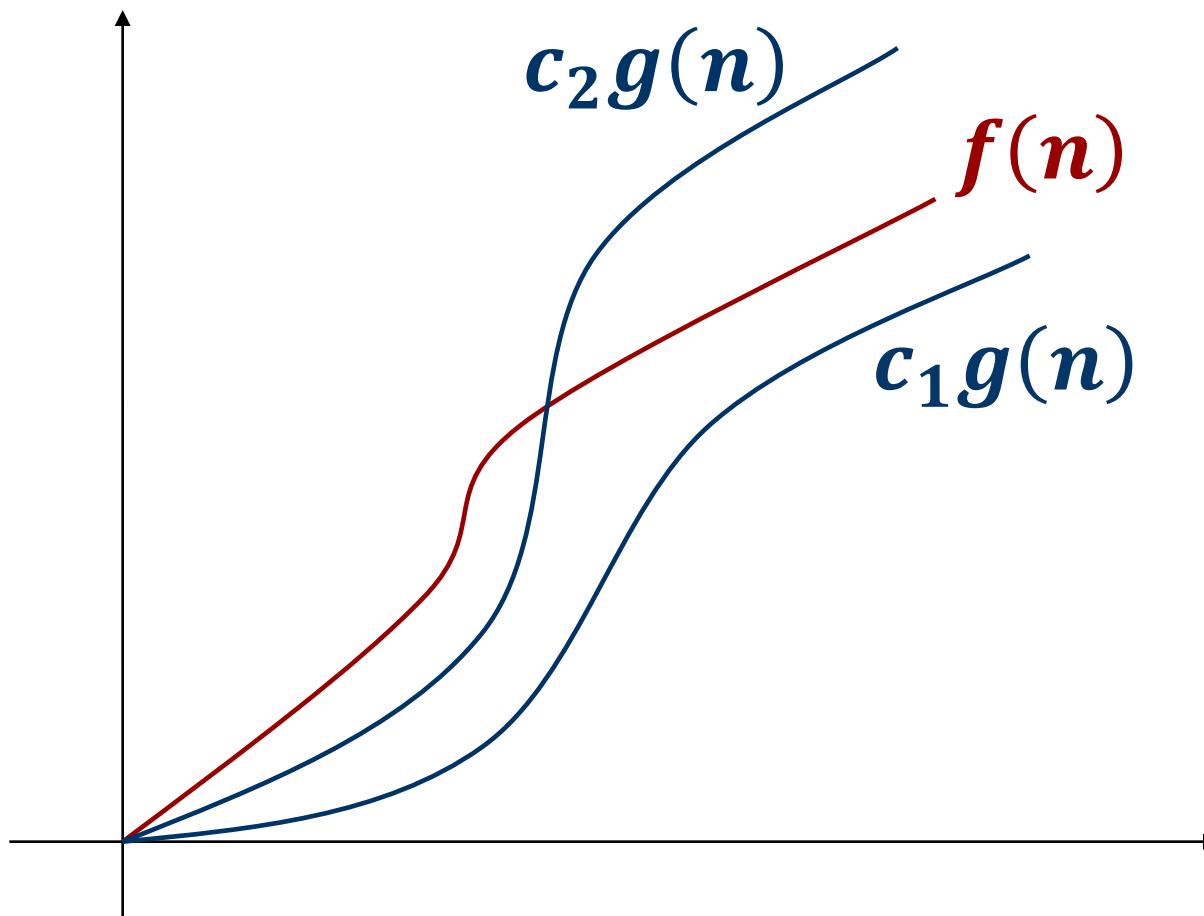
Sei $g: N \rightarrow R^+$ eine Funktion.

Dann bezeichnen wir mit $\Theta(g(n))$ die folgende Menge von Funktionen:

$$\Theta(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c_1 > 0, c_2, n_0, \text{ sodass f\"ur} \\ \text{alle } n \geq n_0 \text{ gilt } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \end{array} \right\}$$

- $\Theta(g(n))$ formalisiert:
Die Funktion $f(n)$ w\"achst asymptotisch genau so schnell wie $g(n)$.
- Statt $f(n)$ in $\Theta(g(n))$ in der Regel $f(n) = \Theta(g(n))$.

Illustration von $\Theta(g(n))$



Regeln für Kalküle - Transitivität

O -, Ω - und Θ -Kalkül sind **transitiv**, d.h.:

- Aus $f(n) = O(g(n))$ und $g(n) = O(h(n))$
folgt $f(n) = O(h(n))$.
- Aus $f(n) = \Omega(g(n))$ und $g(n) = \Omega(h(n))$
folgt $f(n) = \Omega(h(n))$.
- Aus $f(n) = \Theta(g(n))$ und $g(n) = \Theta(h(n))$
folgt $f(n) = \Theta(h(n))$.

Regeln für Kalküle - Reflexivität

- O-, Ω - und Θ -Kalkül sind **reflexiv**, d.h.:

$$f(n) = \mathbf{O}(f(n))$$

$$f(n) = \mathbf{\Omega}(f(n))$$

$$f(n) = \mathbf{\Theta}(f(n))$$

- Θ -Kalkül ist **symmetrisch**, d.h.:

$$f(n) = \Theta(g(n)) \text{ genau dann, wenn } g(n) = \Theta(f(n)).$$

Regeln für Kalküle

Satz

Sei $f: N \rightarrow R^+$ mit $f(n) \geq 1$ für alle n .

Weiter sei $k, l \geq 0$ mit $k \geq l$. Dann gilt:

1. $f(n)^l = O(f(n)^k)$
2. $f(n)^k = \Omega(f(n)^l)$

Satz

Seien $\varepsilon, k > 0$ beliebig. Dann gilt:

1. $\log(n)^k = O(n^\varepsilon)$
2. $n^\varepsilon = \Omega(\log(n)^k)$

Laufzeit einer DTM

Definition

DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{n-1}, q_n)$ halte bei jeder Eingabe.

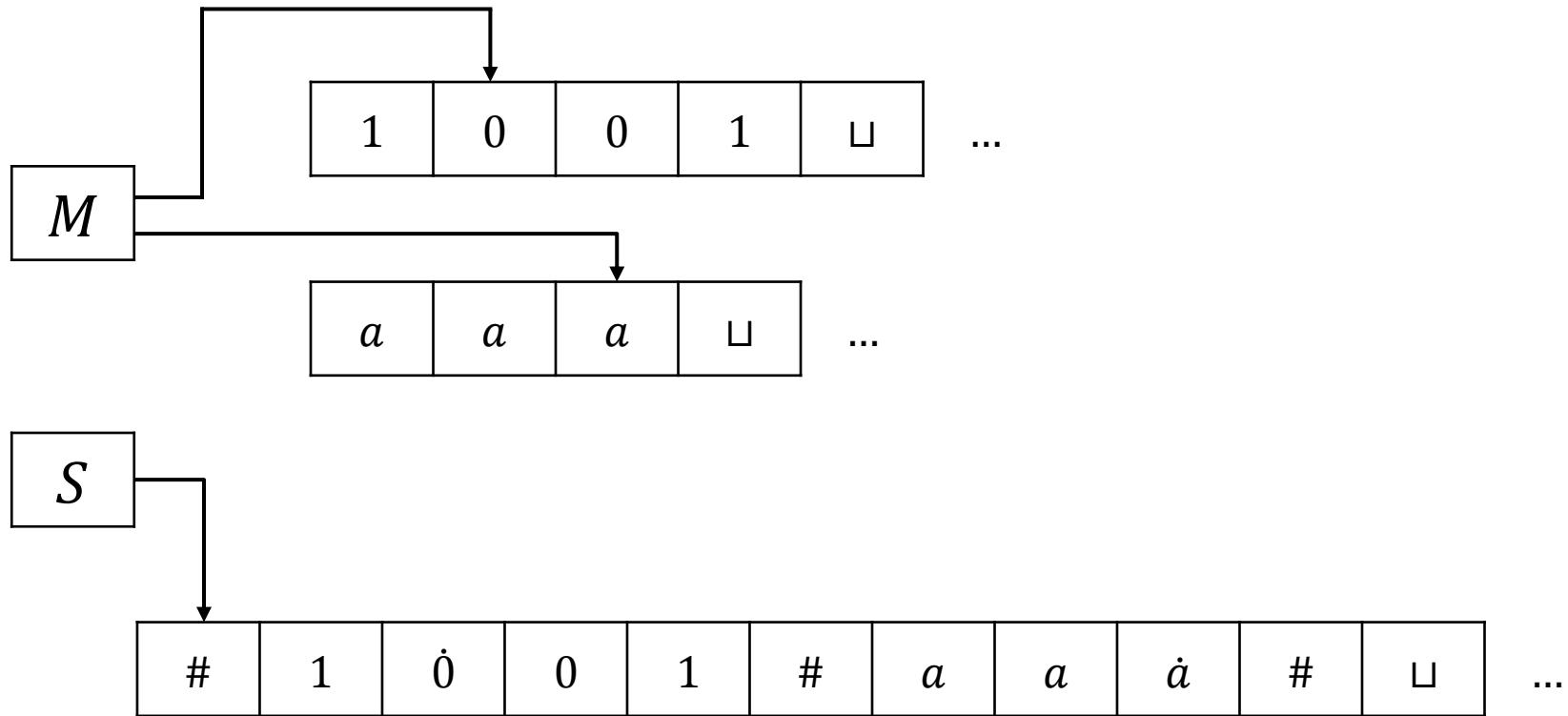
- Für w aus Σ^* ist $T_M(w)$ die Anzahl der Rechenschritte von M bei Eingabe w .
- Für eine natürliche Zahl n ist $T_M(n) := \max\{T_M(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_M heißt Zeitkomplexität oder Laufzeit der DTM M .
- DTM M hat Laufzeit $O(f(n))$, wenn $T_M(n) = O(f(n))$.

Satz

Sei t eine monoton wachsende Funktion mit $t(n) \geq n$.

Jede Mehrband-DTM mit Laufzeit $t(n)$ kann durch eine 1-Band-DTM mit Laufzeit $O(t(n)^2)$ simuliert werden.

3. Komplexität



Die Klasse P

Definition

$$DTIME(t(n)) := \left\{ L \mid L \text{ ist eine Sprache, die von einer DTM mit Laufzeit } t(n) \text{ entschieden wird.} \right\}$$

Definition

Die Klasse P ist definiert als

$$\bigcup_k DTIME(n^k)$$

P ist die Klasse der Sprachen, die durch die DTM mit polynomieller Laufzeit entschieden werden können.

Die Klasse P

Definition

Die Klasse P ist definiert als

$$\bigcup_k DTIME(n^k)$$

P ist die Klasse der Sprachen, die durch die DTM mit polynomieller Laufzeit entschieden werden können.

Beispiel

$\{0^n 1^n \mid n \geq 1\}$ ist in P .

3. Komplexität

Warum P ?

1. P ist eine mathematisch robuste Klasse.
2. Probleme in P lassen sich in der Praxis verhältnismäßig gut lösen, für Probleme außerhalb von P trifft dies in der Regel nicht zu.
3. P liefert eine interessante Theorie.

3. Komplexität

Warum P ?

1. P ist eine mathematisch robuste Klasse.
2. Probleme in P lassen sich in der Praxis verhältnismäßig gut lösen, für Probleme außerhalb von P trifft dies in der Regel nicht zu.
3. **P liefert eine interessante Theorie.**

3. Komplexität

Darstellung von Zahlen und Graphen

1. Repräsentieren Zahlen in der Regel in Binärdarstellung.
 - Darstellung bezüglich anderer Basen ist zulässig, da die Eingabegröße sich dadurch nur um einen konstanten Faktor ändert. Ist die Laufzeit für die Binärdarstellung in P , so trifft dies auch für jede andere Basis zu.

2. Repräsentiere Graphen durch
 - Adjazenzmatrizen oder
 - Adjazenzlisten

Diese Darstellungen unterscheiden sich nur um einen quadratischen Faktor, sodass eine polynomielle Laufzeit bzgl. der einen Eingabeform auch polynomiell bzgl. der anderen ist.

Pfade in Graphen

Definition

Das Problem Pfad ist definiert wie folgt:

$$\text{Pfad} := \left\{ \langle G, s, t \rangle \mid G = (V, E) \text{ ist ein gerichteter Graph mit } s, t \text{ in } V \text{ und einem gerichteten Pfad von } s \text{ nach } t. \right\}$$

Satz

Pfad liegt in P .

Pfade in Graphen

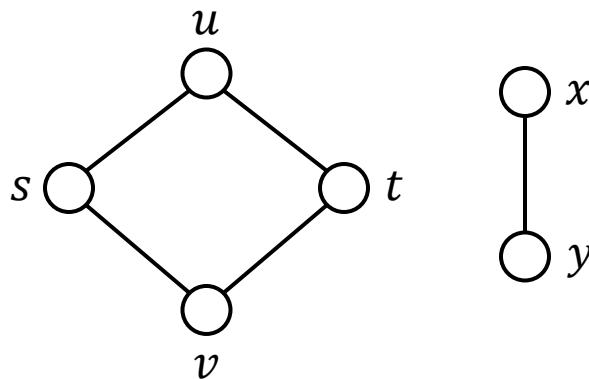
M bei Eingabe $\langle G, s, t \rangle$:

1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.

Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

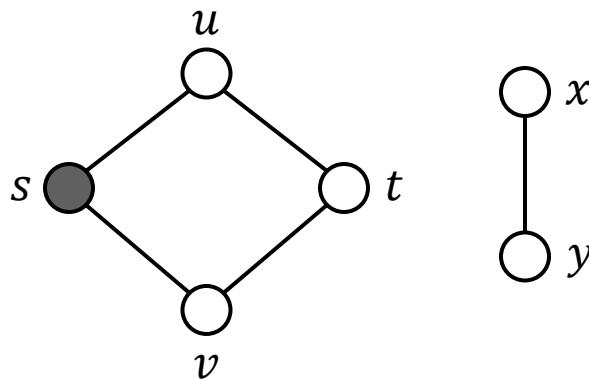
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

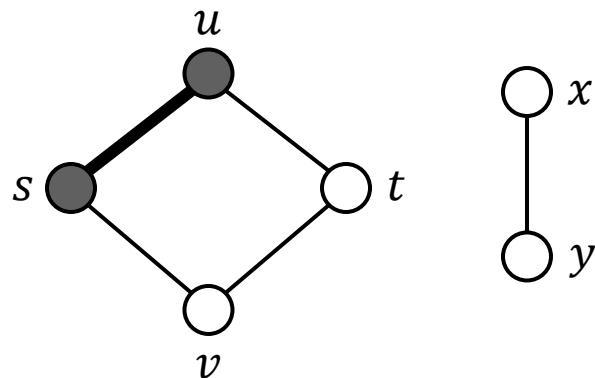
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

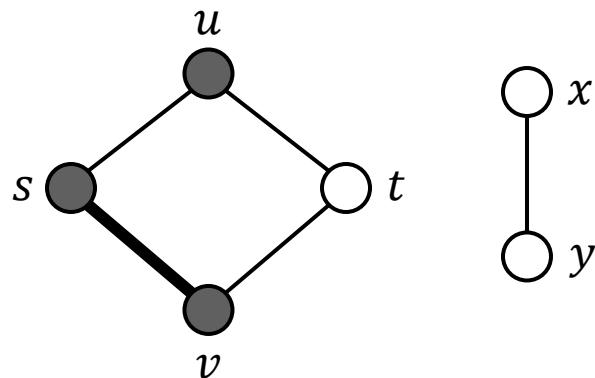
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

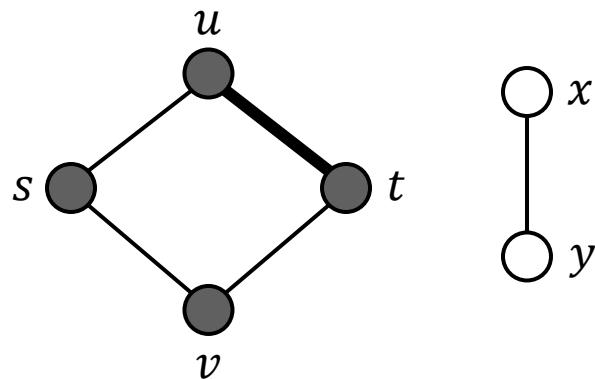
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

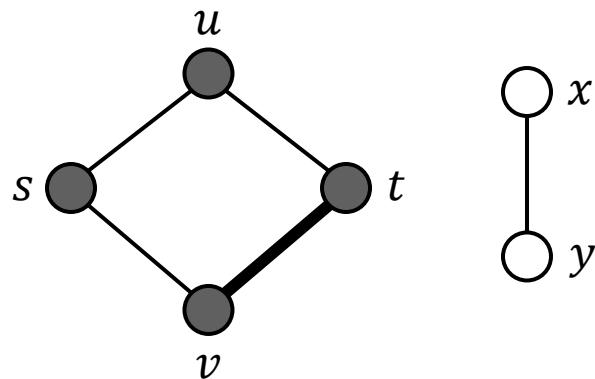
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

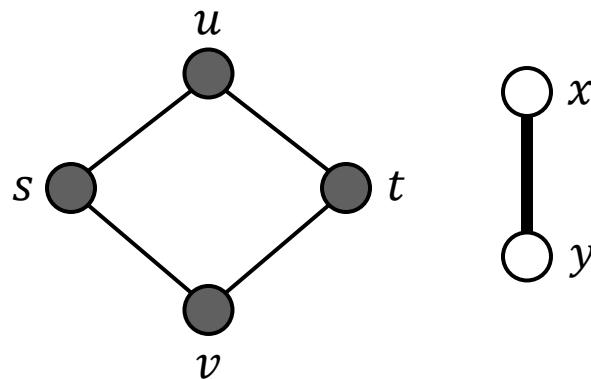
1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Pfade in Graphen

M bei Eingabe $\langle G, s, t \rangle$:

1. Markiere den Knoten s .
2. Wiederhole den folgenden Schritt, bis keine zusätzlichen Knoten mehr markiert werden.
 3. Durchlaufe alle Kanten (a, b) von G .
Ist a markiert und b nicht markiert, so markiere b .
4. Ist t markiert, akzeptiere, sonst lehne ab.



Teilerfremde Zahlen

Definition

Zwei natürliche Zahlen a, b heißen teilerfremd, wenn ihr größter gemeinsamer Teiler (ggT) = 1 ist.

$$\text{RelPrim} := \{\langle x, y \rangle \mid x \text{ und } y \text{ sind teilerfremd}\}$$

Satz

RelPrim liegt in P .

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \text{ mod } y$
 2. vertausche x und y
2. Ausgabe x

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere,
sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

Eingabe: $\langle 16, 9 \rangle$

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \text{ mod } y$
 2. vertausche x und y
2. Ausgabe x

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere,
sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

$$x = 9$$

$$y = 7$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

$$x = 9$$

$$y = 7$$

$$x = 9 \bmod 7 = 2$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

$$x = 9$$

$$y = 7$$

$$x = 9 \bmod 7 = 2$$

$$x = 7$$

$$y = 2$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

$$x = 9$$

$$y = 7$$

$$x = 9 \bmod 7 = 2$$

$$x = 7$$

$$y = 2$$

$$x = 7 \bmod 2 = 1$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Eingabe: $\langle 16, 9 \rangle$

$$x = 16 \bmod 9 = 7$$

$$x = 9$$

$$y = 7$$

$$x = 9 \bmod 7 = 2$$

$$x = 7$$

$$y = 2$$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 9 \bmod 7 = 2$$

$$x = 7$$

$$y = 2$$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \bmod 1 = 1$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Eingabe: $\langle 16, 9 \rangle$

$$x = 9 \bmod 7 = 2$$

$$x = 7$$

$$y = 2$$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \bmod 1 = 1$$

$$x = 1$$

$$y = 1$$

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \bmod 1 = 1$$

$$x = 1$$

$$y = 1$$

$$x = 1 \bmod 1 = 0$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. Ausgabe x

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Eingabe: $\langle 16, 9 \rangle$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \bmod 1 = 1$$

$$x = 1$$

$$y = 1$$

$$x = 1 \bmod 1 = 0$$

$$x = 1$$

$$y = 0$$

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte **bis $y = 0$**
 1. $x = x \text{ mod } y$
 2. vertausche x und y
2. Ausgabe x

Eingabe: $\langle 16, 9 \rangle$

$$x = 7 \text{ mod } 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \text{ mod } 1 = 1$$

$$x = 1$$

$$y = 1$$

$$x = 1 \text{ mod } 1 = 0$$

$$x = 1$$

$$y = 0$$

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere, sonst lehne ab.

Teilerfremde Zahlen

E bei Eingabe $\langle x, y \rangle$:

1. Wiederhole die folgenden Schritte bis $y = 0$
 1. $x = x \bmod y$
 2. vertausche x und y
2. **Ausgabe x**

R bei Eingabe $\langle x, y \rangle$:

1. Simuliere E bei Eingabe $\langle x, y \rangle$
2. Ist die Ausgabe von E 1, so akzeptiere,
sonst lehne ab.

Eingabe: $\langle 16, 9 \rangle$

$$x = 7 \bmod 2 = 1$$

$$x = 2$$

$$y = 1$$

$$x = 2 \bmod 1 = 1$$

$$x = 1$$

$$y = 1$$

$$x = 1 \bmod 1 = 0$$

$$x = 1$$

$$y = 0$$

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

3. Komplexität

Von Berechenbarkeit zu Komplexität

- Berechenbarkeit in Bezug auf Entscheidbarkeit und Aufzählbarkeit betrachtet nur prinzipielle Lösbarkeit von Problemen mit Hilfe von Computern
- Prinzipiell lösbar Probleme können praktisch nicht lösbar sein, weil jeder Algorithmus zur Lösung des Problems zu viel Zeit (und/oder Platz) benötigt.
- Komplexitätstheorie versucht Probleme gemäß des Zeit- und Platzbedarfs des besten Algorithmus zu ihrer Lösung zu klassifizieren.
- Konzentrieren uns auf Zeitbedarf und die Klassen P und NP

Laufzeit einer DTM

Definition

DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{n-1}, q_n)$ halte bei jeder Eingabe.

- Für w aus Σ^* ist $T_M(w)$ die Anzahl der Rechenschritte von M bei Eingabe w .
- Für eine natürliche Zahl n ist $T_M(n) := \max\{T_M(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_M heißt Zeitkomplexität oder Laufzeit der DTM M .
- DTM M hat Laufzeit $O(f(n))$, wenn $T_M(n) = O(f(n))$.

Satz

Sei t eine monoton wachsende Funktion mit $t(n) \geq n$.

Jede Mehrband-DTM mit Laufzeit $t(n)$ kann durch eine 1-Band-DTM mit Laufzeit $O(t(n)^2)$ simuliert werden.

3. Komplexität

Rucksackproblem und Verifizierbarkeit

- **Gegeben:**
 - n Gegenstände mit
 - Gewichten $G = \{g_1, g_2, \dots, g_n\}$ und
 - Werten $W = \{w_1, w_2, \dots, w_n\}$,
 - sowie zulässiges Gesamtgewicht g .
- **Gesucht:**

Teilmenge S aus $\{1, 2, \dots, n\}$ mit $\sum_S w_i$ maximal unter der Bedingung $\sum_S g_i \leq g$.

3. Komplexität

Rucksackproblem und Verifizierbarkeit

- **Gegeben:**

- n Gegenstände mit
- Gewichten $G = \{g_1, g_2, \dots, g_n\}$ und
- Werten $W = \{w_1, w_2, \dots, w_n\}$,
- sowie zulässiges Gesamtgewicht g .

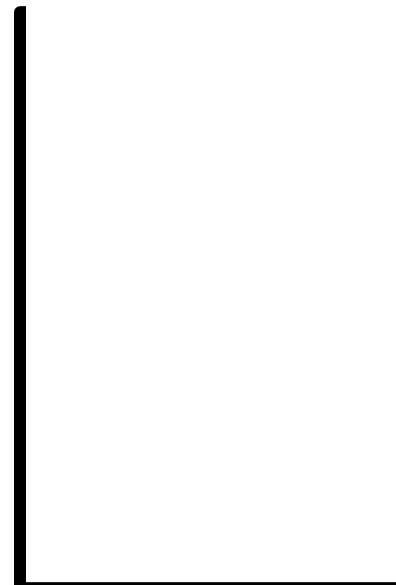
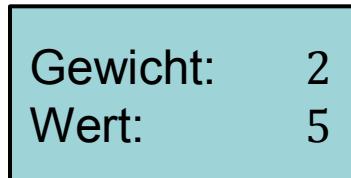
- **Gesucht:**

Teilmenge S aus $\{1, 2, \dots, n\}$ mit $\sum_S w_i$ maximal unter der Bedingung $\sum_S g_i \leq g$.

$$RS_{ent} := \left\{ \langle G, W, g, w \rangle \mid \begin{array}{l} \text{es existiert eine Teilmenge } S \text{ aus } \{1, 2, \dots, n\} \\ \text{mit } \sum_S g_i \leq g \text{ und } \sum_S w_i \geq w \end{array} \right\}$$

Rucksackproblem

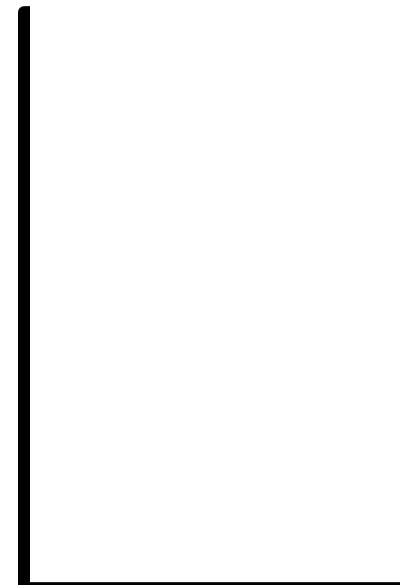
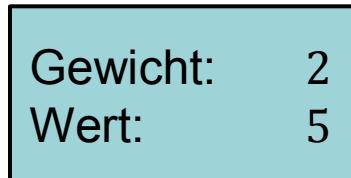
Beispiel:



Gesamtwert: 13

Rucksackproblem

Beispiel:



Gesamtwert: 15

3. Komplexität

Rucksackproblem und Verifizierbarkeit

- Liegt RS_{ent} in P ?
- Algorithmus mit Laufzeit $O(n W)$ aus Algodat ist kein polynomieller Algorithmus!
- Kennen weder Polynomialzeit DTM für RS_{ent} , noch können wir beweisen, dass eine solche existiert.
- RS_{ent} teilt diese Eigenschaften mit vielen anderen Problemen.

Verifizierer

Definition

Sei L eine Sprache. DTM V heißt Verifizierer für L , falls

$$L = \{w \mid \text{es gibt ein } c, \text{ so dass } V \langle w, c \rangle \text{ akzeptiert}\}$$

c : Zertifikat oder Zeuge

V heißt polynomieller Verifizierer, falls eine natürliche Zahl k existiert mit

$$L = \{w \mid \text{es gibt ein } c \text{ mit } |c| \leq |w|^k, \text{ sodass } V \langle w, c \rangle \text{ akzeptiert}\}$$

und die Laufzeit von V bei Eingabe $\langle w, c \rangle$ polynomiell in $|w|$ ist.

L heißt dann polynomiell verifizierbar.

Klasse NP

Definition

NP ist die Klasse der Sprachen, die polynomiell verifizierbar sind.

RS_{ent}, TSP_{ent} sind in NP .

Satz

P ist eine Teilmenge von NP .

Millenium-Problem

Ist $P = NP$? (Clay Mathematics Institute)

3. Komplexität

Nichtdeterministische Turingmaschinen

- Liefern alternative Beschreibung von NP .
- Erlaubt uns, die schwierigsten Probleme in NP zu identifizieren.
 - NP -Vollständigkeit
- Geben der Turingmaschine die Möglichkeit, einen von mehreren möglichen Rechenschritten auszuwählen.
 - Nichtdeterminismus
- Realisierung: $\delta(q, a)$ ist Menge von Tripeln der Form (q', b, D)
- NTMs nicht realistisch, aber sehr nützlich für das Verständnis der Komplexität von Problemen

Nichtdeterministische Turingmaschinen

Definition

Eine nichtdeterministische 1-Band-Turingmaschine (NTM) ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, wobei Q, Σ, Γ endliche Mengen sind.

Weiter gilt:

1. Q ist die Zustandsmenge mit $q_0, q_{accept}, q_{reject}$ Elementen dieser Menge und $q_{accept} \neq q_{reject}$
2. Σ ist das Eingabealphabet
3. Γ ist das Bandalphabet
4. $\delta: Q \setminus \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ ist die Übergangsfunktion. $\mathcal{P}(M)$ bezeichnet hier die Potenzmenge von M .
5. Rechenschritt: einmalige Anwendung der Übergangsfunktion.

Nichtdeterministische Turingmaschinen

$\text{NTM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. Berechnung bei Eingabe w .

1. Startet im Zustand q_0 , mit Bandinhalt w und Lesekopf auf dem ersten Zeichen von w .
2. wendet in jedem Rechenschritt Übergangsfunktion δ an,
3. bis Zustand q_{accept} oder q_{reject} erreicht wird,
4. falls einer dieser Zustände erreicht wird, sonst Endlosschleife.

Nichtdeterministische Turingmaschinen

NTM = $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ ist in Konfiguration $K = \alpha q \beta$, wenn gilt:

1. auf dem Band von N steht $\alpha \beta$, gefolgt von Blanks,
2. N befindet sich im Zustand q ,
3. der Lesekopf von N steht auf dem ersten Symbol von β .

NTM – Rechenschritt

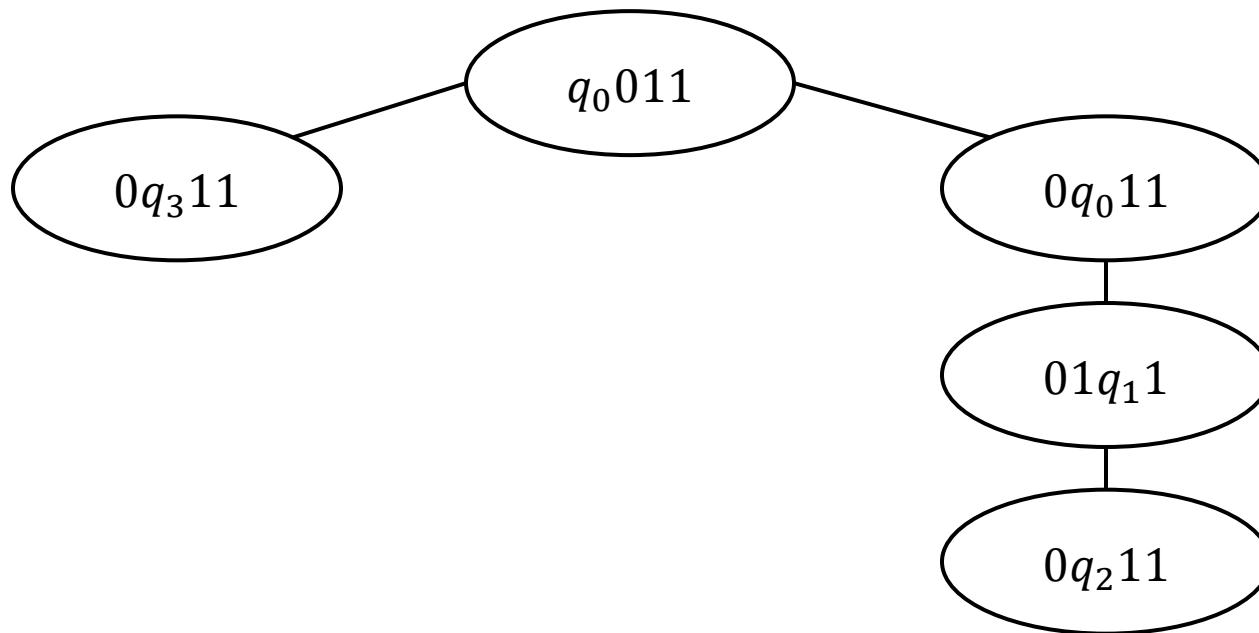
1. NTM N in Konfiguration $K = \alpha q a \beta$.
2. $\delta(q, a) = \{(q_1, b_1, D_1), \dots, (q_l, b_l, D_l)\}$.
3. N kann jeden durch ein Tripel (q_i, b_i, D_i) aus $\delta(q, a)$ beschriebenen Rechenschritt ausführen.

Berechnungen und Konfiguration

1. Ist die Eingabe für $N w$, so heißt $q_0 w$ Startkonfiguration.
2. $K = \alpha q \beta$ heißt akzeptierende Konfiguration, falls $q = q_{accept}$.
3. $K = \alpha q \beta$ heißt ablehnende Konfiguration, falls $q = q_{reject}$.
4. Berechnung von N bei Eingabe w führt zu Folge K_1, K_2, \dots von Konfigurationen.
5. Es gibt mehrere Berechnungen von N bei Eingabe w , abhängig von den ausgewählten Rechenschritten.
6. Darstellung möglicher Berechnungen im sogenannten Berechnungsbaum.

Nichtdeterministische Turingmaschinen

δ	0	1	\sqcup
q_0	$\{(q_0, 0, R), (q_3, 0, R)\}$	$\{(q_1, 1, R)\}$	$\{(q_3, \sqcup, R)\}$
q_1	$\{(q_0, 0, R), (q_3, 0, R)\}$	$\{(q_2, 1, L)\}$	$\{(q_3, \sqcup, R)\}$



Akzeptieren und Entscheiden

Definition

Sei N eine NTM. N akzeptiert w , wenn es mindestens eine akzeptierende Berechnung von N bei Eingabe w gibt.

NTM N hält bei Eingabe w , wenn alle Berechnungspfade von N bei Eingabe w endlich sind.

Definition

Die von einer NTM N akzeptierte Sprache $L(N)$ ist definiert als

$$L(N) := \{w \mid N \text{ akzeptiert } w\}$$

NTM N akzeptiert die Sprache L , falls $L = L(N)$. N entscheidet die von ihr akzeptierte Sprache $L(N)$, wenn N immer hält.

Laufzeit einer NTM

Definition

Sei N eine NTM, die immer hält.

- Für w ist $T_N(w)$ die maximale Anzahl von Rechenschritten in einer Berechnung von N bei Eingabe w .
- Für eine natürliche Zahl n ist $T_N(n) := \max\{T_N(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_N heißt Zeitkomplexität oder Laufzeit der NTM N .
- N hat Laufzeit $O(f(n))$, wenn $T_N(n) = O(f(n))$.

Akzeptieren und Entscheiden

Definition

Sei t eine monoton wachsende Funktion. Die Klasse $NTIME(t(n))$ ist dann definiert als

$$NTIME(t(n)) := \left\{ L \mid L \text{ ist eine Sprache, die von einer NTM mit Laufzeit } O(t(n)) \text{ entschieden wird.} \right\}$$

Satz

NP ist die Klasse der Sprachen, die von einer nichtdeterministischen Turingmaschine mit polynomieller Laufzeit entschieden werden, d.h.,

$$NP = \bigcup_k NTIME(n^k)$$

Simulation einer NTM durch eine DTM

Satz

Sei t eine monoton wachsende Funktion mit $t(n) \geq n$ für alle natürlichen Zahlen n . Für jede NTM mit Laufzeit $t(n)$ gibt es eine DTM mit Laufzeit $2^{o(t(n))}$, die dieselbe Sprache entscheidet.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Laufzeit einer DTM

Definition

DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{n-1}, q_n)$ halte bei jeder Eingabe.

- Für w aus Σ^* ist $T_M(w)$ die Anzahl der Rechenschritte von M bei Eingabe w .
- Für eine natürliche Zahl n ist $T_M(n) := \max\{T_M(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_M heißt Zeitkomplexität oder Laufzeit der DTM M .
- DTM M hat Laufzeit $O(f(n))$, wenn $T_M(n) = O(f(n))$.

Satz

Sei t eine monoton wachsende Funktion mit $t(n) \geq n$.

Jede Mehrband-DTM mit Laufzeit $t(n)$ kann durch eine 1-Band-DTM mit Laufzeit $O(t(n)^2)$ simuliert werden.

Verifizierer

Definition

Sei L eine Sprache. DTM V heißt Verifizierer für L , falls

$$L = \{w \mid \text{es gibt ein } c, \text{ so dass } V \langle w, c \rangle \text{ akzeptiert}\}$$

c : Zertifikat oder Zeuge

V heißt polynomieller Verifizierer, falls eine natürliche Zahl k existiert mit

$$L = \{w \mid \text{es gibt ein } c \text{ mit } |c| \leq |w|^k, \text{ sodass } V \langle w, c \rangle \text{ akzeptiert}\}$$

und die Laufzeit von V bei Eingabe $\langle w, c \rangle$ polynomiell in $|w|$ ist.

L heißt dann polynomiell verifizierbar.

Klasse NP

Definition

NP ist die Klasse der Sprachen, die polynomiell verifizierbar sind.

RS_{ent}, TSP_{ent} sind in NP .

Satz

P ist eine Teilmenge von NP .

Millenium-Problem

Ist $P = NP$? (Clay Mathematics Institute)

Nichtdeterministische Turingmaschinen

NTM $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ ist in Konfiguration $K = \alpha q \beta$, wenn gilt:

1. auf dem Band von N steht $\alpha \beta$, gefolgt von Blanks,
2. N befindet sich im Zustand q ,
3. der Lesekopf von N steht auf dem ersten Symbol von β .

NTM – Rechenschritt

1. NTM N in Konfiguration $K = \alpha q a \beta$.
2. $\delta(q, a) = \{(q_1, b_1, D_1), \dots, (q_l, b_l, D_l)\}$.
3. N kann jeden durch ein Tripel (q_i, b_i, D_i) aus $\delta(q, a)$ beschriebenen Rechenschritt ausführen.

Akzeptieren und Entscheiden

Definition

Sei N eine NTM. N akzeptiert w , wenn es mindestens eine akzeptierende Berechnung von N bei Eingabe w gibt.

NTM N hält bei Eingabe w , wenn alle Berechnungspfade von N bei Eingabe w endliche sind.

Definition

Die von einer NTM N akzeptierte Sprache $L(N)$ ist definiert als

$$L(N) := \{w \mid N \text{ akzeptiert } w\}$$

NTM N akzeptiert die Sprache L , falls $L = L(N)$. N entscheidet die von ihr akzeptierte Sprache $L(N)$, wenn N immer hält.

Laufzeit einer NTM

Definition

Sei N eine NTM, die immer hält.

- Für w ist $T_N(w)$ die maximale Anzahl von Rechenschritten in einer Berechnung von N bei Eingabe w .
- Für eine natürliche Zahl n ist $T_N(n) := \max\{T_N(w) \mid w \text{ aus } \Sigma^{\leq n}\}$.
- Die Funktion T_N heißt Zeitkomplexität oder Laufzeit der NTM N .
- N hat Laufzeit $O(f(n))$, wenn $T_N(n) = O(f(n))$.

Nichtdeterministische Zeitkomplexität

Definition

Sei t eine monoton wachsende Funktion. Die Klasse $NTIME(t(n))$ ist dann definiert als

$$NTIME(t(n)) := \left\{ L \mid L \text{ ist eine Sprache, die von einer NTM mit Laufzeit } O(t(n)) \text{ entschieden wird} \right\}$$

Satz

NP ist die Klasse der Sprachen, die von einer nichtdeterministischen Turingmaschine mit polynomieller Laufzeit entschieden werden, d.h.,

$$NP = \bigcup_k NTIME(n^k)$$

Simulation einer NTM durch eine DTM

Satz

Sei t eine monoton wachsende Funktion mit $t(n) \geq n$ für alle natürlichen Zahlen n . Für jede NTM mit Laufzeit $t(n)$ gibt es eine DTM mit Laufzeit $2^{o(t(n))}$, die dieselbe Sprache entscheidet.

Polynomielle Reduktion

Definition

Sei Σ ein Alphabet. Eine Funktion $f: \Sigma^* \rightarrow \Sigma^*$ heißt polynomiell berechenbar, wenn es eine DTM M mit polynomieller Laufzeit gibt, die f berechnet.

Definition

Seien A, B zwei Sprachen. A heißt auf B polynomiell reduzierbar, wenn es eine polynomiell berechenbare Funktion f gibt mit:

$$w \text{ in } A \Leftrightarrow f(w) \text{ in } B$$

Die Funktion f wird polynomielle Reduktion genannt und man schreibt

$$A \leq_P B$$

Polynomielle Reduktion – Eigenschaften

Satz

Seien A, B zwei Sprachen. Gilt $A \leq_P B$ und B ist in P , so ist auch A in P .

Lemma

Die Relation \leq_P ist transitiv.

Boolesche Variablen, Operatoren, Formeln

- **Boolesche Variablen** x können die beiden Werte wahr (1) oder falsch (0) annehmen
- **Boolesche Operatoren:** und (\wedge); oder (\vee); nicht (\neg).
- **Boolesche Formeln:** Ausdruck bestehend aus Booleschen Variablen und Operatoren, korrekt formatiert.

Beispiel

$$\varphi = (\neg x \wedge y) \vee (x \wedge \neg z)$$

Boolesche Variablen, Operatoren, Formeln

- **Boolesche Formel** φ heißt erfüllbar, wenn es eine Belegung der Variablen in φ mit 1 und 0 gibt, sodass die Formel dann wahr ist.

Beispiel

$\varphi = (\neg x \wedge y) \vee (x \wedge \neg z)$ ist erfüllbar. Belegung $x = 0, y = 1, z = 0$.

Beispiel

$\varphi = (x \wedge \neg x) \vee (y \wedge \neg y)$ ist nicht erfüllbar.

Die Sprache SAT

Definition

$SAT := \{\varphi \mid \varphi \text{ ist eine erfüllbare Boolesche Formel}\}$

Satz

SAT liegt in NP .

Boolesche Variablen, Operatoren, Formeln

- Literale sind Boolesche Variablen oder Negationen Boolescher Variablen.
- Eine Klausel ist die Disjunktion von Literalen.
- Eine Formel ist in konjunktiver Normalform (KNF), wenn sie die Konjunktion von Klauseln ist.
- In 3-KNF enthält jede Klausel 3 Literale.

Definition

$3SAT := \{\varphi \mid \varphi \text{ ist eine erfüllbare 3-KNF Formel}\}$

Graphen und Cliques

Definition

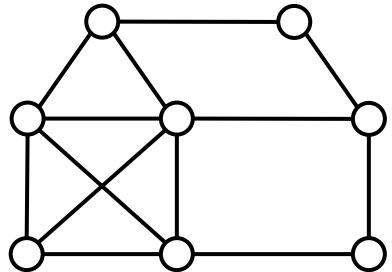
Sei $G = (V, E)$ ein ungerichteter Graph. Eine Teilmenge C von V heißt Clique, wenn alle Knoten aus C miteinander verbunden sind. C heißt k -Clique, wenn C genau k Knoten hat.

Definition

$Clique := \{(G, k) \mid G \text{ ist ein ungerichteter Graph mit einer } k\text{-Clique}\}$

Polynomielle Reduktion – Eigenschaften

$$G = (V, E)$$



$(G, 4) \in \text{Clique}$

$(G, 5) \notin \text{Clique}$

Satz

3SAT ist auf Clique polynomiell reduzierbar.

NP-Vollständigkeit

Definition

Eine Sprache L heißt NP -vollständig, wenn sie die folgenden Bedingungen erfüllt:

- L ist in NP
- Für jede Sprache L' aus NP gilt: $L' \leq_P L$

Satz

Ist L NP -vollständig und in P , so gilt $P = NP$.

Satz

Ist L in NP und gilt $L' \leq_P L$ für eine Sprache L' , die NP -vollständig ist, so ist auch L NP -vollständig.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Klasse NP

Definition

NP ist die Klasse der Sprachen, die polynomiell verifizierbar sind.

Satz

P ist eine Teilmenge von NP .

Millenium-Problem

Ist $P = NP$? (Clay Mathematics Institute)

Polynomielle Reduktion

Definition

Sei Σ ein Alphabet. Eine Funktion $f: \Sigma^* \rightarrow \Sigma^*$ heißt polynomiell berechenbar, wenn es eine DTM M mit polynomieller Laufzeit gibt, die f berechnet.

Definition

Seien A, B zwei Sprachen. A heißt auf B polynomiell reduzierbar, wenn es eine polynomiell berechenbare Funktion f gibt mit:

$$w \text{ in } A \Leftrightarrow f(w) \text{ in } B$$

Die Funktion f wird polynomielle Reduktion genannt und man schreibt

$$A \leq_P B$$

Polynomielle Reduktion – Eigenschaften

Satz

Seien A, B zwei Sprachen. Gilt $A \leq_P B$ und B ist in P , so ist auch A in P .

Lemma

Die Relation \leq_P ist transitiv.

NP-Vollständigkeit

Definition

Eine Sprache L heißt NP -vollständig, wenn sie die folgenden Bedingungen erfüllt:

- L ist in NP
- Für jede Sprache L' aus NP gilt: $L' \leq_P L$

Satz

Ist L NP -vollständig und in P , so gilt $P = NP$.

Satz

Ist L in NP und gilt $L' \leq_P L$ für eine Sprache L' , die NP -vollständig ist, so ist auch L NP -vollständig.

3. Komplexität

Definition

$Clique := \{(G, k) \mid G \text{ ist ein ungerichteter Graph mit einer } k\text{-Clique}\}$

Definition

$3SAT := \{\varphi \mid \varphi \text{ ist eine erfüllbare 3-KNF Formel}\}$

NP-Vollständigkeit

Satz von Cook-Levin

SAT ist *NP*-vollständig.

Satz von Cook-Levin

3SAT ist *NP*-vollständig.

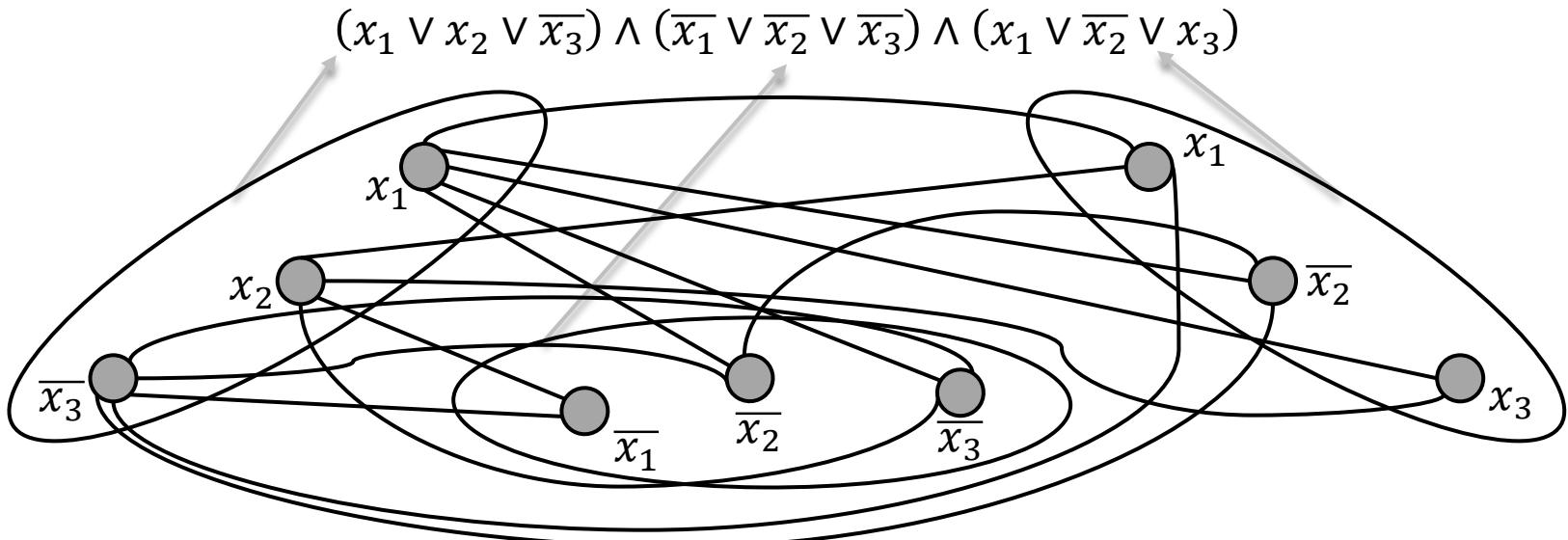
NP-Vollständigkeit

Satz

Clique ist NP-vollständig.

Satz

3SAT ist auf *Clique* polynomiell reduzierbar.



NP-Vollständigkeit

Satz

SubsetSum ist NP-vollständig.

$\text{SubsetSum} := \left\{ (S, t) \mid S = \{s_1, \dots, s_n\}, \text{ wobei } s_1, \dots, s_n, t \text{ natürliche Zahlen } \right. \\ \left. \text{ und es gibt eine Teilmenge } T \text{ aus } \{1, \dots, n\} \text{ mit } \sum_i s_i = t \right\}$

3. Komplexität

$$\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

	x_1	x_2	x_3	c_1	c_2
y_1	1	0	0	1	0
z_1	1	0	0	0	1
y_2	0	1	0	1	1
z_2	0	1	0	0	0
y_3	0	0	1	0	1
z_3	0	0	1	1	0
g_1	0	0	0	1	0
h_1	0	0	0	1	0
g_2	0	0	0	0	1
h_2	0	0	0	0	1
t	1	1	1	3	3

3. Komplexität

$$\varphi = (x_1 \vee \textcircled{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \textcircled{x}_3)$$

	x_1	x_2	x_3	c_1	c_2
y_1	1	0	0	1	0
z_1	1	0	0	0	1
y_2	0	1	0	1	1
z_2	0	1	0	0	0
y_3	0	0	1	0	1
z_3	0	0	1	1	0
g_1	0	0	0	1	0
h_1	0	0	0	1	0
g_2	0	0	0	0	1
h_2	0	0	0	0	1
t	1	1	1	3	3

$$\begin{aligned}x_1 &= 0 \\x_2 &= 1 \\x_3 &= 1\end{aligned}$$

Graphen und Knotenüberdeckung

Satz

Knotenüberdeckung ist NP -vollständig.

Definition

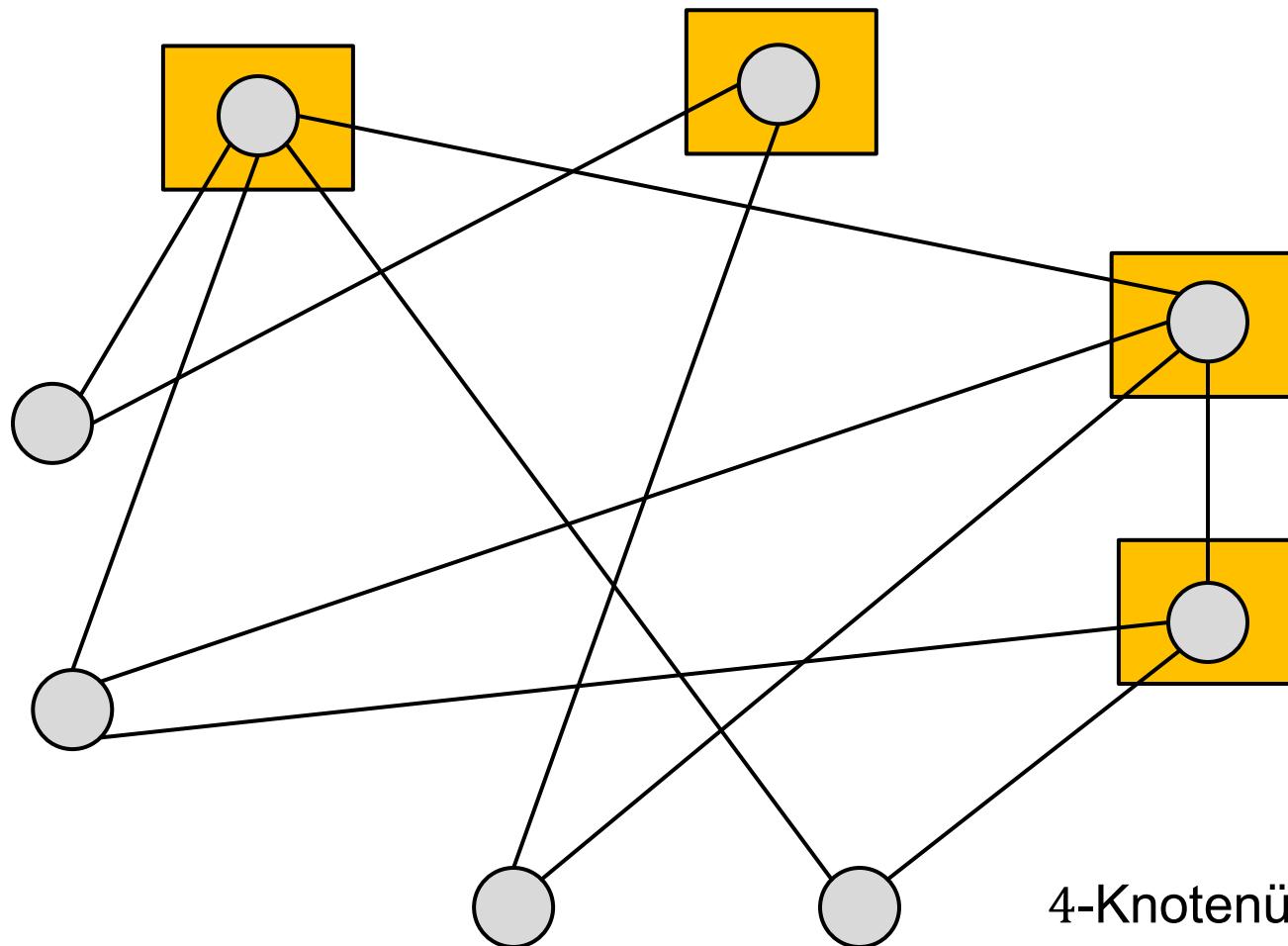
$G = (V, E)$ ist ein ungerichteter Graph.

Eine Teilmenge U von V heißt Knotenüberdeckung, wenn für alle Kanten $\{u, v\}$ aus E gilt $|\{u, v\} \cap U| \neq 0$.

U heißt k -Knotenüberdeckung, wenn U Knotenüberdeckung und $|U| = k$.

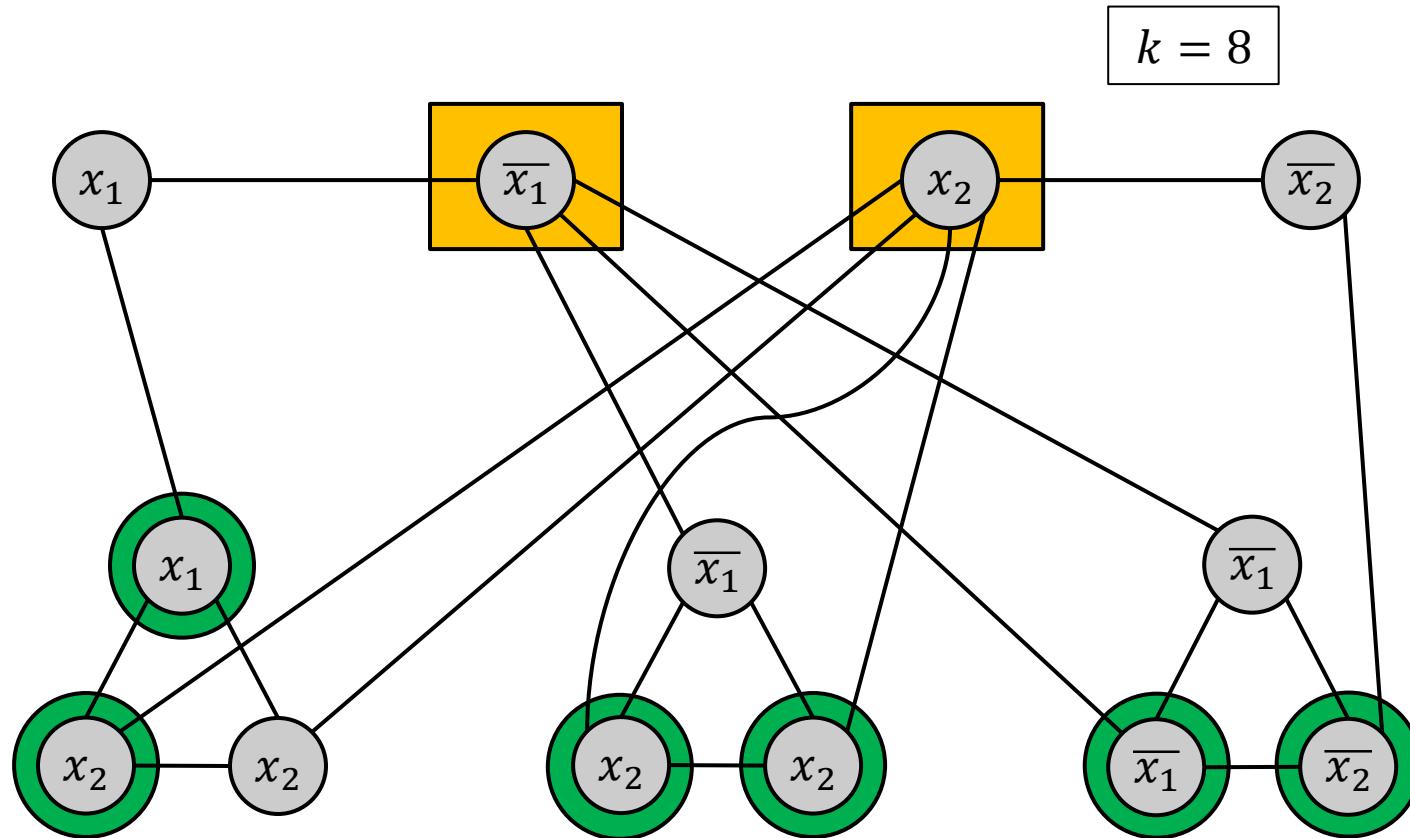
Knotenüberdeckung := $\{(G, k) \mid G \text{ besitzt eine } k\text{-Knotenüberdeckung}\}$

Knotenüberdeckung



Reduktion 3SAT auf Knotenüberdeckung

$$\varphi = (x_1 \vee x_2 \vee x_2) \wedge (\overline{x_1} \vee x_2 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_2})$$



NP-Vollständigkeit

Satz

SubsetSum ist polynomiell reduzierbar auf RS_{ent} .

Satz

RS_{ent} ist NP-vollständig.

Definition

$RS_{ent} := \left\{ (G, W, g, w) \mid G = \{g_1, \dots, g_n\}, W = \{w_1, \dots, w_n\} \text{ und es gibt eine } \right. \\ \left. \text{Teilmenge } S \text{ aus } \{1, \dots, n\} \text{ mit } \sum_i g_i \leq g \text{ und } \sum_i w_i \geq w \right\}$

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

4. Formale Sprachen

Grammatiken

- dienen zur formalen Beschreibung von Sprachen
- werden benutzt zur Beschreibung von Programmiersprachen, Anfragesprachen bei Datenbanken, usw.
- sollen einfach sein, um eine effiziente Analyse zu erlauben
- sollen mächtig genug sein, um aufwändige Konstrukte wie z.B. geschachtelte Schleifen beschreiben zu können.
- Wortprobleme bei eingeschränkten Grammatiken führen zu speziellen Rechenmodellen, die genau diese Wortprobleme lösen.

4. Formale Sprachen

Grammatiken

- Rechenmodelle sind endliche Automaten für reguläre Grammatiken und Kellerautomaten für kontextfreie Grammatiken.
- Varianten von kontextfreien Grammatiken werden zur Beschreibung von Programmiersprachen eingesetzt.
- Reguläre Grammatiken und endliche Automaten werden bei Kontrollsystmen und in der lexikographischen Analyse eingesetzt.
- Allgemeine Grammatiken liefern alternative Beschreibungen von rekursiv aufzählbaren Sprachen.

Grammatiken

Definition

Eine Grammatik (vom Typ Chomsky-0) ist ein 4-Tupel (V, Σ, P, S) , für den gilt:

- V ist ein endliches Alphabet von Variablen
- Σ ist ein endliches Alphabet von Terminalen
- S ist das Startsymbol
- P ist eine endliche Menge von Produktionen oder Ersatzregeln, d.h. P ist eine Teilmenge von $((V \cup \Sigma)^+ \setminus \Sigma^*) \times (V \cup \Sigma)^*$

4. Formale Sprachen

Grammatiken

- w' ist aus w direkt ableitbar, wenn es eine Ersetzungsregel $u \rightarrow v$ und α, β in $(V \cup \Sigma)^*$ gibt, so dass $w = \alpha u \beta$ und $w' = \alpha v \beta$, geschrieben $w \rightarrow w'$.
- w' ist aus w ableitbar, falls w' durch endlich viele Ableitungsschritte aus w erhalten werden kann, geschrieben $w \xrightarrow{*} w'$

Äquivalent: es gibt $w_0 = w, w_1, \dots, w_{n-1}, w_n = w'$ mit $w_{i-1} \rightarrow w_i$

Grammatiken

Definition

Sei $G = (V, \Sigma, P, S)$ eine Grammatik. Dann ist

$$L(G) := \{w \text{ aus } \Sigma^* \mid S \xrightarrow{*} w\}$$

die von G erzeugte Sprache.

4. Formale Sprachen

Grammatiken

In einer Linksableitung wird in jedem Schritt die am weitesten links stehende Variable im nächsten Schritt ersetzt.

Ableitungen sind in der Regel nicht eindeutig.

Grammatiken

Satz

Eine Sprache L ist genau dann rekursiv aufzählbar, wenn es eine Grammatik G vom Typ Chomsky-0 gibt mit $L(G) = L$.

Eingeschränkte Grammatiken

Definition

- Eine Grammatik heißt kontextsensitiv oder vom Typ Chomsky-1, falls für jede Regel $u \rightarrow v$ gilt: $|u| \leq |v|$.
- Eine Grammatik heißt kontextfrei oder vom Typ Chomsky-2, falls für jede Regel $u \rightarrow v$ gilt: $u \in V$.
- Eine Regel heißt regulär oder vom Typ Chomsky-3, falls alle Regeln der Art $u \rightarrow v$ mit $u \in V$ und:
 - $v = \epsilon$
 - $v = a$, $a \in \Sigma$ oder
 - $v = aw$ mit $a \in \Sigma$ und $w \in V$sind.

Eingeschränkte Grammatiken

Definition

- Eine Grammatik heißt kontextsensitiv oder vom Typ Chomsky-1, falls für jede Regel $u \rightarrow v$ gilt: $|u| \leq |v|$.

Ausnahme: Bei kontextsensitiven Grammatiken wird die Regel $S \rightarrow \varepsilon$ zugelassen.
Es muss dann allerdings für alle Regeln $u \rightarrow v$ gelten, dass S in v nicht vorkommt.

4. Formale Sprachen

Definition

Eine Sprache L heißt kontextsensitiv, kontextfrei oder regulär, wenn es eine kontextsensitive, kontextfreie oder reguläre Grammatik G gibt mit $L(G) = L$.

Satz

Jede kontextsensitive Sprache ist entscheidbar.

Formale Sprachen und Komplexitätstheorie

WS 2018/19
Robert Elsässer

Eingeschränkte Grammatiken

Definition

- Eine Grammatik heißt kontextsensitiv oder vom Typ Chomsky-1, falls für jede Regel $u \rightarrow v$ gilt: $|u| \leq |v|$.
- Eine Grammatik heißt kontextfrei oder vom Typ Chomsky-2, falls für jede Regel $u \rightarrow v$ gilt: $u \in V$.
- Eine Regel heißt regulär oder vom Typ Chomsky-3, falls alle Regeln der Art $u \rightarrow v$ mit $u \in V$ und:
 - $v = \epsilon$
 - $v = a$, $a \in \Sigma$ oder
 - $v = aw$ mit $a \in \Sigma$ und $w \in V$sind.

Kontextfrei vs. regulär

- wir betrachten $L = \{0^n 1^n \mid n \geq 1\}$
- L ist nicht regulär (*Pumping Lemma* für reguläre Sprachen – siehe VO „Formale Systeme“)
- L ist kontextfrei, Grammatik gegeben z.B. durch
 1. $S \rightarrow 0S1$
 2. $S \rightarrow 01$

Nichtdeterministische Automaten

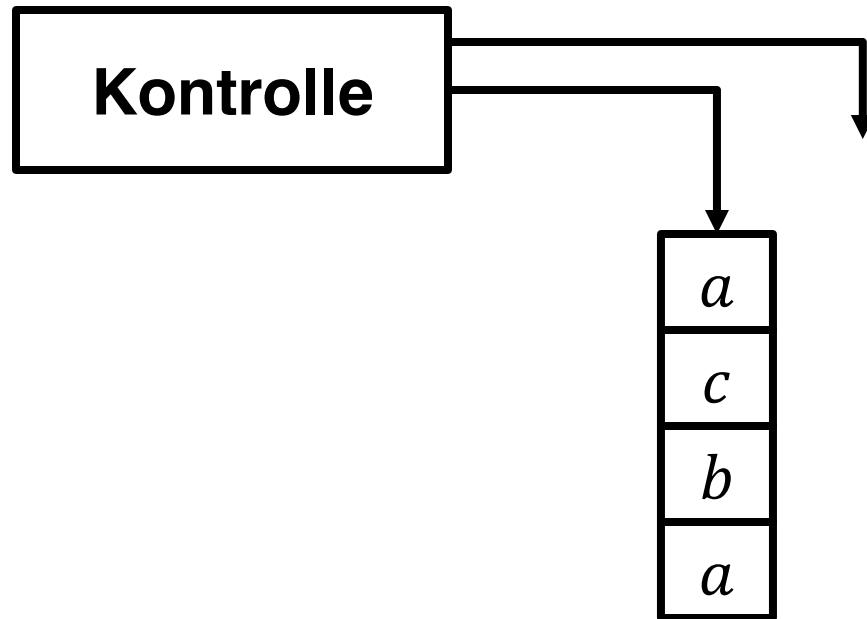
Definition

Ein nichtdeterministischer endlicher Automat $A = (Q, \Sigma, \delta, q_0, F)$ besteht aus:

- einer endlichen Menge von Eingabesymbolen Σ
- einer endlichen Menge von Zuständen Q
- einer Übergangsfunktion $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$
- einem Anfangszustand $q_0 \in Q$
- einer Menge $F \subseteq Q$ von Endzuständen

Kellerautomaten und Stapel

- Kellerautomaten sind nichtdeterministische Automaten mit zusätzlichem **Stapel** als Speicher
- Kellerautomaten durchlaufen einmal die Eingabe (fast)
- Stapel erlaubt Ablegen, Entfernen und Lesen von Elementen
- es kann nur das zuletzt abgelegte Element gelesen oder entfernt werden (last-in-first-out)



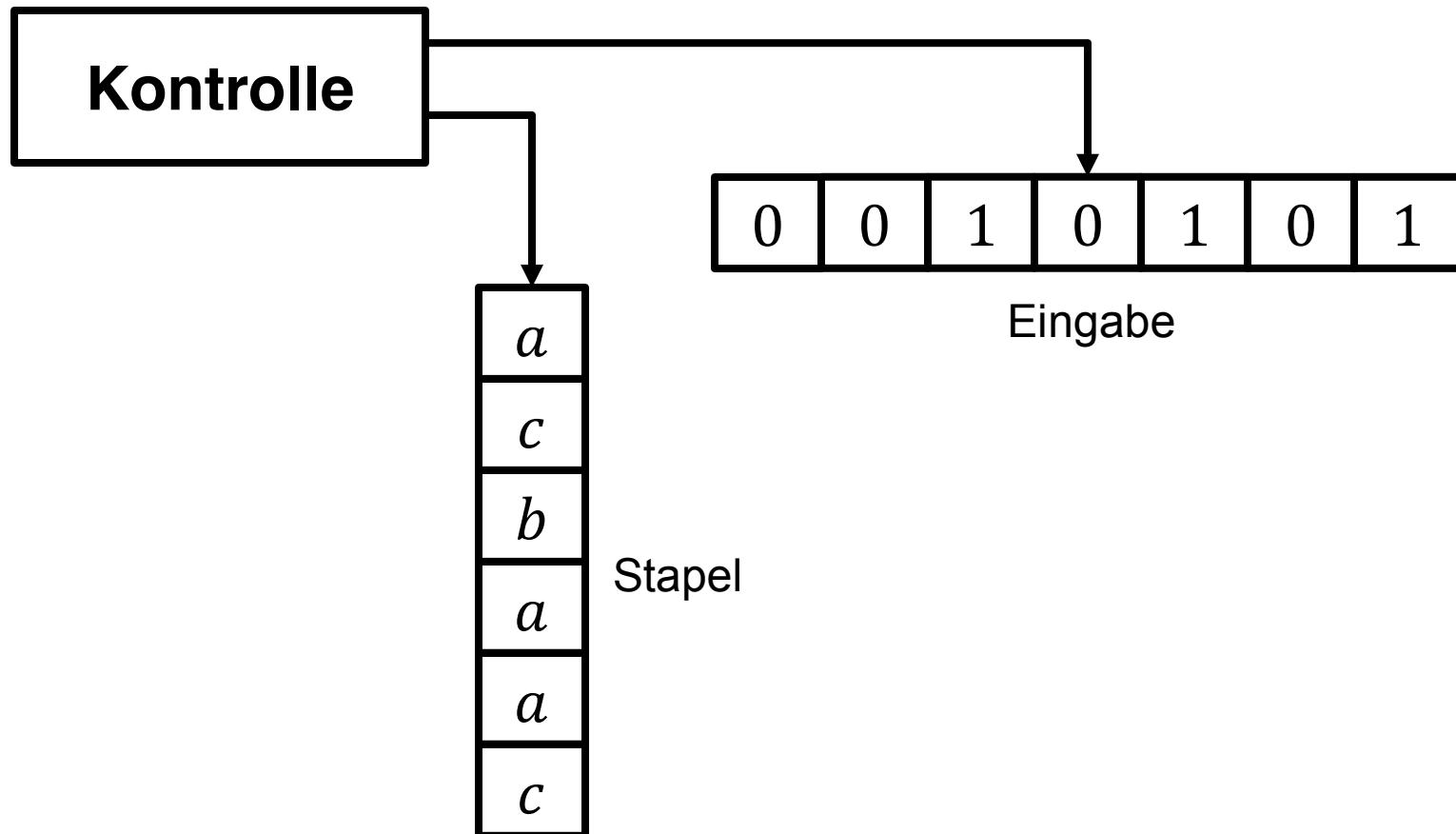
Kellerautomaten

Definition

Ein Kellerautomat (PDA) ist definiert durch ein 6-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$, wobei

1. Q eine endliche Menge von Zuständen ist,
2. Σ das endliche Eingabealphabet ist,
3. Γ das endliche Stapelalphabet ist,
4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ die Übergangsfunktion ist,
5. $q_0 \in Q$ der Startzustand und $F \subseteq Q$ die Menge der akzeptierenden Zustände ist.

Kellerautomaten – schematische Darstellung

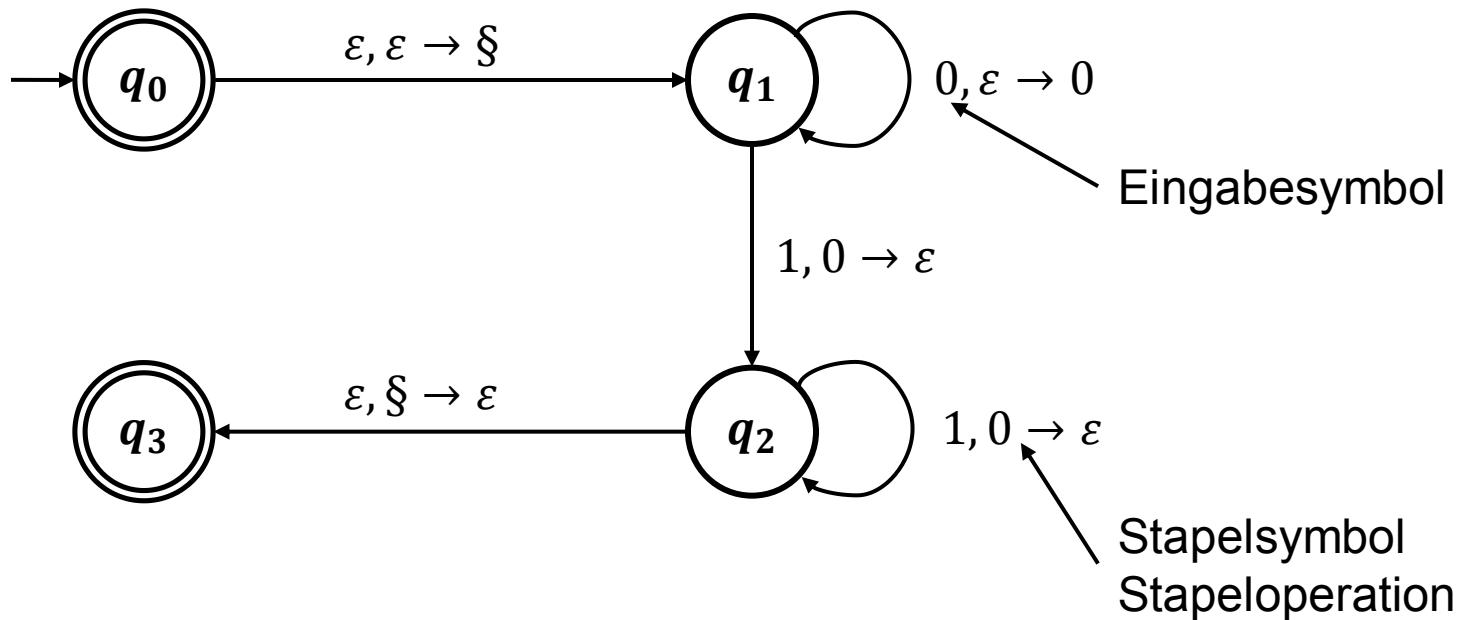


Kellerautomaten

Definition

- Kellerautomaten sind nichtdeterministisch
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$
- Übergangsfunktion $(q_2, c) \in \delta(q_1, a, b)$, $a, b, c = \varepsilon$ zugelassen
- $a = \varepsilon$: Lesekopf wird nicht bewegt, also Kellerautomaten durchlaufen einmal die Eingabe, dürfen aber dabei anhalten
- $b = \varepsilon$: c wird zusätzlich auf Stapel gelegt (push-Operation)
- $c = \varepsilon$: b wird vom Stapel entfernt (pop-Operation)

Kellerautomaten – graphische Darstellung



$\$$ markiert Boden des Stapels

PDA – Berechnung

$PDA = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Berechnung bei Eingabe $w \in \Sigma^*$:

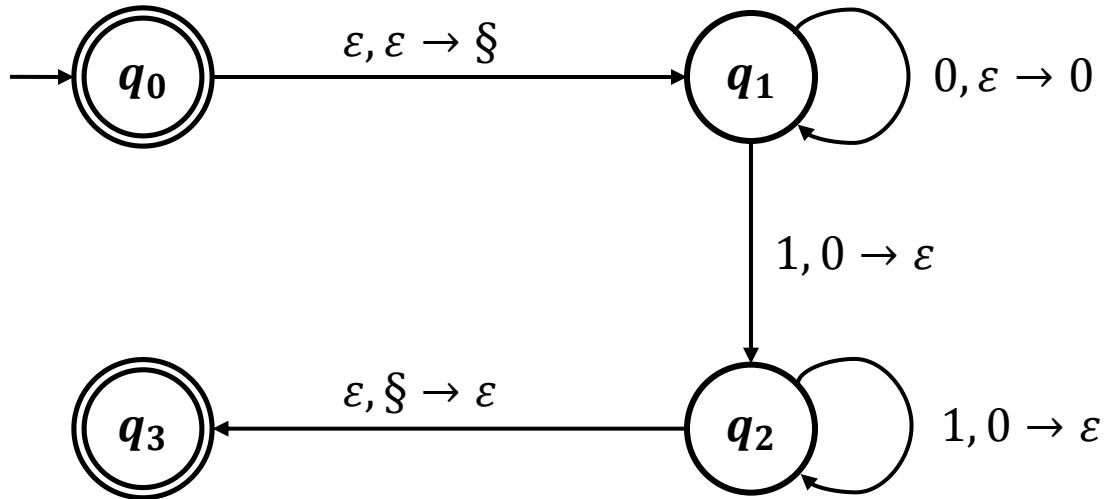
- wir schreiben $w = w_1 w_2 \dots w_m, w_i \in \Sigma_\varepsilon$, um Anhalten zu modellieren
- wendet in jedem Rechenschritt Übergangsfunktion δ an, dabei wird stets das nächste w_i gelesen
- Berechnung endet, falls Ende der Eingabe erreicht wurde oder für gelesenes Tripel (Zustand, Eingabesymbol, Stapelsymbol) Wert der Übergangsfunktion = \emptyset ist.
- **Konfiguration** ist gegeben durch aktuelle Position i in der Eingabe, den Zustand und den Stapelinhalt

PDA – Berechnung

PDA $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Berechnung bei Eingabe $w \in \Sigma^*$:

- Berechnung startet in den Zustand q_0 , mit leerem Stapel und Lesekopf auf w_1
- durchläuft Folge von Konfigurationen
- Berechnung heißt **akzeptierend**, wenn Ende der Eingabe erreicht wird und letzter Zustand in F liegt
- abbrechende Berechnungen ($\delta(q, a, v) = \emptyset$) sind **ablehnend**
- w wird von K akzeptiert, falls es eine akzeptierende Berechnung von K bei Eingabe w gibt

PDA – Berechnung



Stapel:

ε
§
0§
00§
0§
§
ε

- $w = 0011$, schreiben $w = \epsilon 0011 \epsilon$
- Folge von Zuständen ist $q_0, q_1, q_1, q_1, q_2, q_2, q_3$

PDA – akzeptierte Sprache

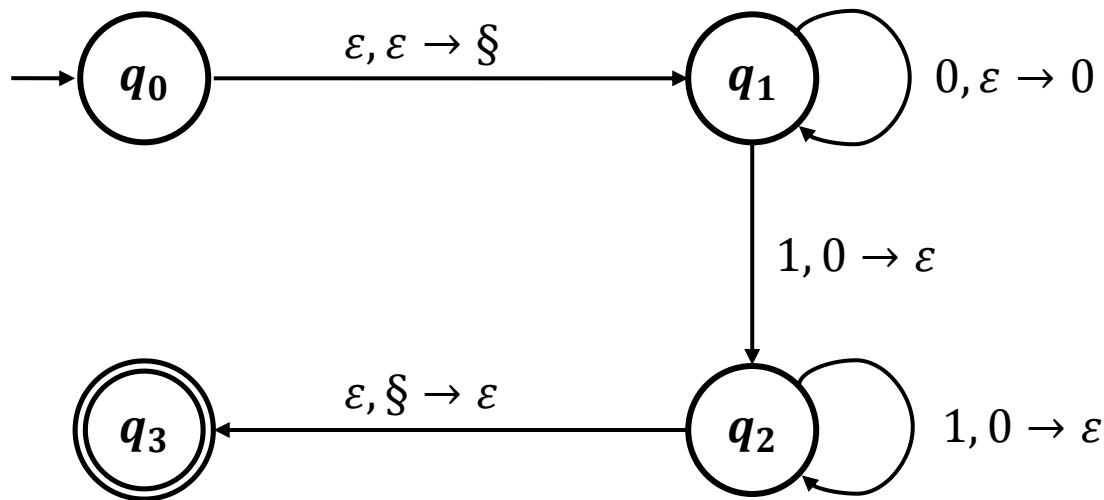
PDA $K = (Q, \Sigma, \Gamma, \delta, q_0, F)$

$L(K) = \{w \in \Sigma^* \mid \text{es gibt akzeptierende Berechnung von } K \text{ bei Eingabe } w\}$

PDA für $L_1 = \{0^n 1^n \mid n \geq 1\}$

1. Solange das Eingabesymbol eine 0 ist, lege eine 0 auf den Stapel.
2. Wird eine 1 gelesen, entferne eine 0 vom Stapel.
3. Entferne bei jeder weiteren gelesenen 1 eine 0 vom Stapel, bis entweder das Ende der Eingabe erreicht worden ist, oder eine 0 gelesen wird, oder der Stapel leer ist.
4. Akzeptiere, wenn am Ende der Rechnung der Stapel leer ist.

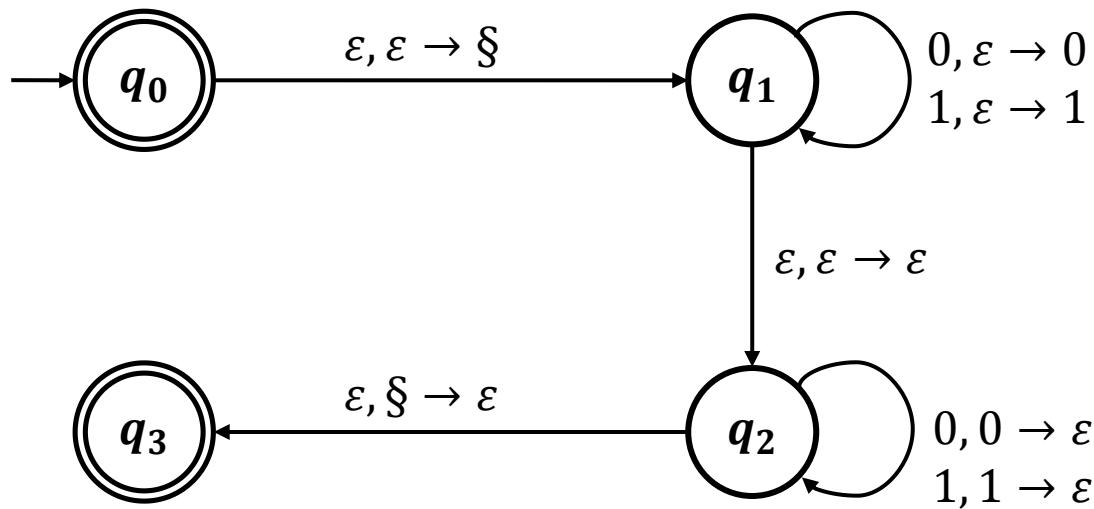
PDA für $L_1 = \{0^n 1^n \mid n \geq 1\}$



PDA für $L_2 = \{ww^R \mid w \in \{0, 1\}^*\}$

1. Lege nichtdeterministisch einen Präfix der Eingabe auf den Stapel.
2. Überprüfe, ob Rest der Eingabe mit dem Präfix (in der Reihenfolge der Symbole auf dem Stapel) übereinstimmt.
3. Akzeptiere, falls dies der Fall ist, sonst lehne ab.

PDA für $L_2 = \{ww^R \mid w \in \{0, 1\}^*\}$



PDAs und kontextfreie Sprachen

Satz —

Sei L eine kontextfreie Sprache.

Dann gibt es einen PDA K , der L akzeptiert.

Satz —

Sei K ein PDA und $L = L(K)$.

Dann ist L eine kontextfreie Sprache.