

# Theoretische Informatik

Prof. Dr. Barbara Staehle

HTWG Konstanz  
Fakultaet für Informatik

WS 2021/2022

## Teil V

# Kontextsensitive, rekursiv aufzählbare Sprachen und Turing-Maschinen

# Teil V Typ 0&1 Sprachen und TM

## 1. Kontextsensitive und rekursiv aufzählbare Sprachen

- 1.1 Kontextsensitive Sprachen
- 1.2 Rekursiv aufzählbare Sprachen

## 2. Turing-Maschinen

- 2.1 Einfache Turing-Maschinen
- 2.2 Erweiterte und eingeschränkte Turing-Maschinen
- 2.3 Turing-Maschinen und Typ 0&1 Sprachen

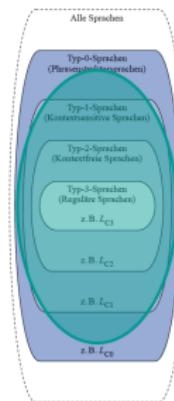
## Abschnitt 1

# Kontextsensitive und rekursiv aufzählbare Sprachen

# Kontextsensitive Sprachen - Einführung

Kontextsensitive, Typ 1 Sprachen, Sprachen der Klasse  $\mathcal{L}_1$

- Erweiterung (Obermenge) von  $\mathcal{L}_2$  in der Chomsky-Hierarchie
- sehr ausdrucksstark (Ersetzung eines Nonterminals in Abhängigkeit seiner Umgebung möglich), daher schwer zu verarbeiten
- Modell einiger Programmiersprachen und natürlicher Sprachen
- Erzeugung: kontextsensitive Grammatiken
- Akzeptanz: nichtdeterministische linear beschränkten Turingmaschinen (LBA, siehe Abschnitt 2)



Quelle:  
[Hoffmann, 2011]

## Definition

Eine formale Sprache heißt **kontextsensitiv**, falls Sie von einer kontextsensitiven Grammatik erzeugt wird.

# Beispiel: Erzeugung der Sprache $L_{C1}$ I

Wir erinnern uns:  $L_{C1} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

Wieso kann  $L_{C1}$  nicht von einer Typ 2 oder 3 Grammatik erzeugt werden?

Weil  $L_{C1}$  nicht kontextfrei, damit auch nicht regulär ist. Siehe Beweis via Pumping-Lemma oder gescheiterter Versuch der Konstruktion einer entsprechenden Grammatik, eines PDAs oder eines D/NEAs.

Konstruktion einer kontextsensitiven Regelmenge für  $L_{C1}$

- Nonterminalmenge: Startsymbol  $S$ , sowie für jedes Terminal ein Nonterminal ( $A, B, C$ ).
- Regelmenge besteht aus Gruppen die für Verschiedenes sorgen:
  - ▶ Gruppe 1:  $A, B, C$  müssen immer gleich oft vorkommen.
  - ▶ Gruppe 2:  $A, B, C$  müssen in die richtige Reihenfolge sortiert werden.
  - ▶ Gruppe 3: Nonterminale müssen durch Terminale ersetzt werden, aber nur, wenn die Reihenfolge stimmt.

# Beispiel: Erzeugung der Sprache $L_{C1} \sqcup$

Kontextsensitive Grammatik für  $L_{C1}$ :  $G_{C1} = (\{S, A, B, C\}, \{a, b, c\}, P, S)$   
mit der Regelmenge  $P$ :

## Gruppe 1 (richtige Anzahl)

$$\begin{array}{lcl} S & \rightarrow & SABC \\ S & \rightarrow & abc \end{array}$$

## Gruppe 3 (Terminale erzeugen)

$$\begin{array}{lcl} aA & \rightarrow & aa \\ bB & \rightarrow & bb \\ cC & \rightarrow & cc \end{array}$$

## Gruppe 2 (richtige Reihenfolge)

$$\begin{array}{lcl} BA & \rightarrow & AB \\ CB & \rightarrow & BC \\ CA & \rightarrow & AC \\ bA & \rightarrow & Ab \\ cB & \rightarrow & Bc \\ cA & \rightarrow & Ac \end{array}$$

$$\begin{aligned} \mathcal{L}(G_{C1}) = \\ \{abc, aabbcc, aaabbbccc, \dots\} \end{aligned}$$

# Beispiel: Erzeugung der Sprache $L_{C1}$ III

Ableitung des Wortes „aaabbbccc“:

Erzeugen von 3\*3 Symbolen

$$\begin{aligned} S &\Rightarrow SABC \\ &\Rightarrow SABCABC \end{aligned}$$

$$\Rightarrow abAABBcCC$$

$$\Rightarrow aAbABBcCC$$

$$\Rightarrow aAAbBBcCC$$

Sortieren

$$\begin{aligned} &\Rightarrow abcABCABC \\ &\Rightarrow abcABACBC \\ &\Rightarrow abcAABCBC \\ &\Rightarrow abcAABBCC \\ &\Rightarrow abAcABBCC \\ &\Rightarrow abAAcBBCC \\ &\Rightarrow abAABcBCC \end{aligned}$$

Erzeugen der Terminate

$$\begin{aligned} &\Rightarrow aaAbBBcCC \\ &\Rightarrow aaabbbccc \end{aligned}$$

# Exkurs: Entscheidungsprobleme kontextsensitiver Sprachen

Problem	Eingabe	Fragestellung	Entscheidbar?
Wortproblem	$L$ , Wort $\omega \in \Sigma^*$	Ist $\omega \in L$ ?	✓ Ja
Leerheitsproblem	$L$	Ist $L = \emptyset$	✗ Nein
Endlichkeitsproblem	$L$	Ist $ L  < \infty$ ?	✗ Nein
Äquivalenzproblem	$L_1$ und $L_2$	Ist $L_1 = L_2$ ?	✗ Nein

Tabelle: Entscheidungsprobleme für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_1$ 

## Bemerkung:

- Das Problem „ $\omega \in L$ “ lässt sich entscheiden, da ausgenutzt werden kann, dass Regeln die Form  $I \Rightarrow r$  mit  $|I| \leq |r|$  haben, damit kann das Wort in einer Ableitung nie kürzer werden. Insbesondere können also alle Wörter der Länge  $|\omega|$  erzeugt und untersucht werden, ob  $\omega$  enthalten ist.

# Exkurs: Abschlusseigenschaften kontextsensitiver Sprachen

Operation	Eingabe	Fragestellung	Antwort
Vereinigung	$L_1, L_2$	Ist $L_1 \cup L_2 \in \mathcal{L}_2$ ?	✓ Ja
Schnitt	$L_1, L_2$	Ist $L_1 \cap L_2 \in \mathcal{L}_2$ ?	✓ Ja
Komplement	$L$	Ist $\Sigma^* \setminus L \in \mathcal{L}_2$ ?	✓ Ja
Produkt	$L_1, L_2$	Ist $L_1 L_2 \in \mathcal{L}_2$ ?	✓ Ja
Stern	$L$	Ist $L^* \in \mathcal{L}_1$ ?	✓ Ja

Tabelle: Abschlusseigenschaften für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_1$ 

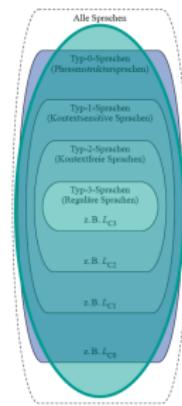
Bemerkung:

- Achtung: Abgeschlossenheit für Schnitt und Komplement gilt nur für kontextsensitive Sprachen, die nicht kontextfrei sind.
- Beweis der Abgeschlossenheit für Vereinigung, Produkt und die Kleene'sche Hülle über Konstruktion geeigneter Grammatiken.

# Rekursiv aufzählbare Sprachen - Einführung

## Rekursiv aufzählbare, Typ 0 Sprachen, Sprachen der Klasse $\mathcal{L}_0$

- Erweiterung (Obermenge) von  $\mathcal{L}_1$  in der Chomsky-Hierarchie
- Stellen keinerlei Restriktionen an die verwendeten Regeln
- Modell aller Sprachen, die algorithmisch berechenbar sind  
**Achtung:** es gibt auch Sprachen, die **nicht** vom Typ 0 sind
- Erzeugung: Typ 0 Grammatiken
- Akzeptanz: deterministische und nichtdeterministische Turingmaschinen (TM, NTM siehe Abschnitt 2)



Quelle:  
[Hoffmann, 2011]

## Definition

Eine formale Sprache heißt **rekursiv aufzählbar** oder **Phrasenstruktursprache** falls Sie von einer Phrasenstrukturgrammatik (Typ 0) erzeugt wird.

# Exkurs: Entscheidungsprobleme rekursiv aufzählbarer Sprachen

Problem	Eingabe	Fragestellung	Entscheidbar?
Wortproblem	$L$ , Wort $\omega \in \Sigma^*$	Ist $\omega \in L$ ?	✗ Nein
Leerheitsproblem	$L$	Ist $L = \emptyset$	✗ Nein
Endlichkeitsproblem	$L$	Ist $ L  < \infty$ ?	✗ Nein
Äquivalenzproblem	$L_1$ und $L_2$	Ist $L_1 = L_2$ ?	✗ Nein

Tabelle: Entscheidungsprobleme für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_0$ 

## Bemerkung:

- Typ 0 Sprachen sind sehr ausdrucksstark. Daher können nur wenige Aussagen über die erzeugten Sprachen gemacht werden, für die Beantwortung der obigen Fragen existiert also jeweils kein Algorithmus.

# Exkurs: Abschlusseigenschaften rekursiv aufzählbarer Sprachen

Operation	Eingabe	Fragestellung	Antwort
Vereinigung	$L_1, L_2$	Ist $L_1 \cup L_2 \in \mathcal{L}_2$ ?	✓ Ja
Schnitt	$L_1, L_2$	Ist $L_1 \cap L_2 \in \mathcal{L}_2$ ?	✓ Ja
Komplement	$L$	Ist $\Sigma^* \setminus L \in \mathcal{L}_2$ ?	✗ Nein
Produkt	$L_1, L_2$	Ist $L_1 L_2 \in \mathcal{L}_2$ ?	✓ Ja
Stern	$L$	Ist $L^* \in \mathcal{L}_1$ ?	✓ Ja

Tabelle: Abschlusseigenschaften für Sprachen über  $\Sigma^*$ ,  $L, L_1, L_2 \in \mathcal{L}_1$ 

## Bemerkung:

- Achtung: Abgeschlossenheit für Schnitt gilt nur für Typ 0 Sprachen, die nicht kontextfrei sind.
- Beweis der Abgeschlossenheit für Vereinigung, Schnitt, Produkt und die Kleene'sche Hülle über Konstruktion geeigneter Grammatiken.

## Abschnitt 2

# Turing-Maschinen

# Historische Annäherung an die Turing-Maschine

Turing-Maschinen sind das Schweizer Taschenmesser der (theoretischen) Informatik: Sie akzeptieren Sprachen **und** berechnen Funktionen.

Benannt nach ihrem Erfinder, Alan Turing. Ursprünglicher Zweck:  
Beweis, dass es Funktionen gibt, die nicht berechenbar sind.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions  $q_1, q_2, \dots, q_k$  which will be called “*m*-configurations”. The machine is supplied with a “tape” (the analogue of paper) running through it, and divided into sections (called “squares”) each capable of bearing a “symbol”. At any moment there is just one square, say the  $r$ -th, bearing the symbol  $\mathfrak{S}(r)$  which is “in the machine”. We may call this square the “scanned square”. The symbol on the scanned square may be called the “scanned symbol”. The “scanned symbol” is the only one of which the machine is, so to speak, “directly aware”. However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has “seen” (scanned) previously. The possible behaviour of the machine at any moment is determined by the *m*-configuration  $q_n$  and the scanned symbol  $\mathfrak{S}(r)$ . This pair  $q_n, \mathfrak{S}(r)$  will be called the “configuration”:

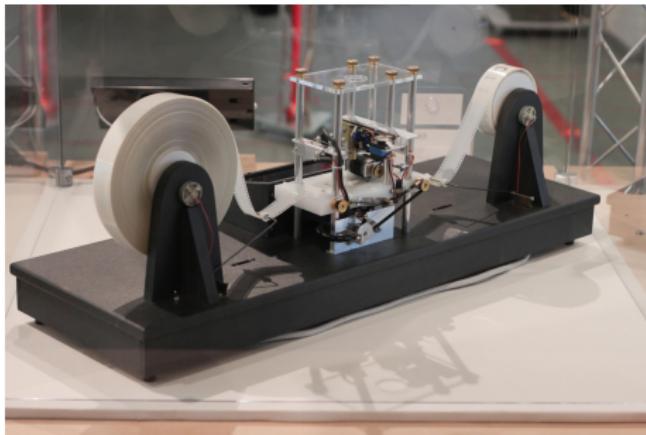
Bild: Originaldefinition der Turing-Maschine [Turing, 1936]



Bild: Alan Turing (1912 – 1954) auf der 2021 neu gedruckten 50£-Note  
**Quelle:** [www.iflscience.com](http://www.iflscience.com)

Video: Alan Turing - betrayed by the country he saved

# Bildliche Darstellungen der Turing-Maschine



Quelle: [en.wikipedia.org](https://en.wikipedia.org)

Turing-Maschinen in Aktion:  
aus Holz, aus Lego und im Film.

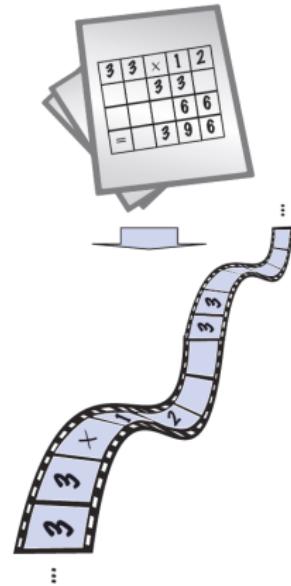


Bild: Kernidee der Turing-Maschine: alles was auf einem karierten Blatt Papier berechnet werden kann, kann auch auf einem Streifen berechnet werden  
[Hoffmann, 2011]

# Turing-Maschine (TM)

## Definition

Eine (deterministische) **Turing-Maschine**, kurz **TM** ist ein 6-Tupel  $T = (Q, \Sigma, \Pi, \delta, q_0, F)$  mit

- der endlichen **Zustandsmenge**  $Q$ ,
- dem endlichen **Eingabealphabet**  $\Sigma$  mit  $\Pi \supset \Sigma$ ,
- dem endlichen **Bandalphabet**  $\Pi$  mit dem **Blank-Symbol**  $\square \in \Pi \setminus \Sigma$ ,
- der **Zustandsübergangsfunktion**  $\delta : Q \times \Pi \rightarrow Q \times \Pi \times \{\leftarrow, \rightarrow\}$
- dem **Startzustand**  $q_0$ ,
- der Menge der **Finalzustände**  $F \subseteq Q$ .

**Bemerkung:** Eine TM ist einem DET sehr ähnlich, jedoch kann eine TM den **Schreib-/Lesekopf** beliebig nach **links und rechts bewegen**. Außerdem hat eine TM kein dediziertes Ausgabeband, sondern kann auf dem **unendlich langen Arbeitsband** **lesen und schreiben**.

# Arbeitsweise einer TM I

- Zu Beginn
  - ▶ befindet sich die TM im Zustand  $q_0$ ,
  - ▶ auf dem unendlich langen Arbeitsband steht das Eingabewort  $\omega$ , links und rechts davon Blanks,
  - ▶ Schreib-/Lesekopf steht auf erstem Zeichen von  $\omega$ .
- In jedem Verarbeitungsschritt führt die Maschine die folgenden Aktionen aus:
  1. Einlesen des aktuellen Bandzeichens  $\sigma$
  2. Berechnung (für aktuellen Zustand  $q$ ) von  $\delta(q, \sigma) = (q', \sigma', b)$ .
    - 2.1 Ersetzung des aktuellen Bandzeichens  $\sigma$  durch  $\sigma'$ .
    - 2.2 Bewegung des Lesekopfes nach links ( $b = \leftarrow$ ) oder rechts ( $b = \rightarrow$ )
    - 2.3 Zustandsübergang in den Folgezustand  $q'$
- Nach der Verarbeitung des Eingabewortes hat die TM
  - ▶ das Wort ENTWEDER akzeptiert ODER nicht akzeptiert
  - ▶ UND ein Ergebnis berechnet (siehe nächstes Kapitel)

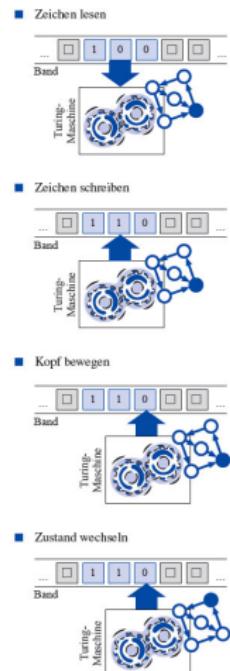


Bild: Arbeitsweise einer TM  
[Hoffmann, 2011]

## Arbeitsweise einer TM II

- Die Verarbeitung des Eingabewortes ist zu Ende, falls für das aktuell gelesene Symbol und den aktuellen Zustand kein nächster Schritt definiert ist.
- **Falls Endzustand akzeptierender Zustand ist:**
  - ▶ Berechnung erfolgreich, Eingabewort akzeptiert.
  - ▶ **Der Inhalt des Arbeitsbandes ab dem Schreib-/Lesekopf ist das Ergebnis der Berechnung.**
- **Andernfalls:** Berechnung nicht erfolgreich, Eingabewort nicht akzeptiert.

### Weitere Definitionsvarianten:

- „L“, „R“ statt  $\leftarrow$ ,  $\rightarrow$  für die Bewegungen; sowie „nicht bewegen“ erlaubt:  $\leftarrow$ ,  $\rightarrow$ ,  $\circlearrowleft$  bzw. „L,R,0“
- Kein Endzustand, statt dessen Anweisung „H“ (für Halten) zum Beenden der Berechnung.
- Kein Zurücksetzen des Kopfes, Ergebnis = gesamter Bandinhalt.

**Beispiel:** TM zur Berechnung von  $n + 1$  im Turing-Maschinen-Simulator der PUC Chile. (Programme siehe Moodle)

# Beispiel zum Mitdenken

**Gesucht:** TM  $T_1$  welche für Eingabe  $n$  die Zahl  $n + 1$  ausgibt.

**Idee:** Codiere Zahlen **unär** (eine 1 pro Bandzelle) und lasse TM einfach so lange nach rechts laufen, bis die letzte 1 gefunden ist, und schreibe dann eine weitere 1, dann fahre den Kopf zurück zum Wortanfang.

**Lösung:**  $T_1 = (Q, \Sigma, \Pi, \delta, q_0, F)$  mit

- $Q = \{q_0, q_1, q_2\}$ ,  $\Sigma = \{1\}$ ,  $\Pi = \{1, \square\}$ ,  $F = \{q_2\}$
- $\delta$ :
  1.  $\delta(q_0, 1) = (q_0, 1, \rightarrow)$  // Solange 1 gefunden wird, lasse diese unverändert und laufe nach rechts
  2.  $\delta(q_0, \square) = (q_1, 1, \leftarrow)$  // Wortende gefunden: Eine 1 hinten anfügen und zurück laufen
  3.  $\delta(q_1, 1) = (q_1, 1, \leftarrow)$  // Solange 1 gefunden wird, lasse diese unverändert und laufe nach links
  4.  $\delta(q_1, \square) = (q_2, \square, \rightarrow)$  // Wortbeginn gefunden: Kopf auf die erste 1 bewegen und in den Endzustand übergehen.

# Grafische Darstellung einer TM

Idee: Verwende ein **erweitertes Zustandsübergangsdiagramm**

- Erweitere Zustandsübergangsdiagramm durch Berücksichtigung des geschriebenen Zeichens und der Bandbewegung
- Beschriffe jeden Pfeil mit  
**(gelesenes Eingabezeichen; zu schreibendes Zeichen, Bandbewegung)**
- Berücksichtige bei der Simulation der Verarbeitung eines Wortes auch den Zustand des Eingabebandes.

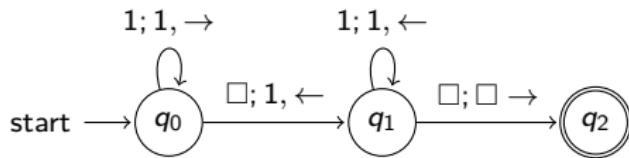


Bild: Erweitertes Zustandsübergangsdiagramm für  $T_1$

# Beschreibung einer TM durch Konfigurationen

**Beobachtung:** Der Zustand, die Position des Kopfes sowie der Inhalt des Bandes beschreiben die aktuelle Situation einer TM vollständig.

## Definition

Eine **Konfiguration**  $k = (\omega_l Y, q, X \omega_r)$  einer TM  $T = (Q, \Sigma, \Pi, \delta, q_0, F)$  enthält

- die Zeichenkette  $\omega_l Y$ , die links vom Schreib-/Lesekopf steht,
- den aktuellen Zustand  $q \in Q$ ,
- das Zeichen  $X$ , welches direkt unter dem Schreib-/Lesekopf steht,
- die Zeichenkette  $\omega_r$ , die rechts vom Schreib-/Lesekopf steht.

$K_T$  ist die Menge aller Konfigurationen für  $T$ ,  $X, Y \in \Pi$  sind einzelne Zeichen,  $\omega_l, \omega_r \in \Pi^*$  sind Wörter mit 0 bis beliebig viele Zeichenn.

**Bemerkung:** Die unendlich vielen Blanks links und rechts des Eingabewortes werden nur durch  $X, Y$  (ein einzelnes Blank, falls der Eingabekopf am Wortrand steht) erfasst.

# Beschreibung einer TM durch die Übergangsrelation

Die Arbeitsweise einer TM wird durch die Verkettung von Konfigurationen mittels der **Übergangsrelation** beschrieben:

## Definition

Sei  $T = (Q, \Sigma, \Pi, \delta, q_0, F)$  eine beliebige Turing-Maschine.

Für beliebige  $q_1, q_2 \in Q, \omega_l, \omega_r \in \Pi^*, X, Y, Z \in \Pi$  ist die **Übergangsrelation**  $\vdash \subseteq K_T \times K_T$  definiert wie folgt:

1. **Rechtsbewegung:**  $\delta(q_1, X) = (q_2, Z, \rightarrow)$

$$(\omega_l Y, q_1, X\omega_r) \vdash \begin{cases} (\omega_l Y Z, q_2, \omega_r) & \text{falls } \omega_r \neq \varepsilon, \\ (\omega_l Y Z, q_2, \square) & \text{falls } \omega_r = \varepsilon. \end{cases}$$

2. **Linksbewegung:**  $\delta(q_1, X) = (q_2, Z, \leftarrow)$

$$(\omega_l Y, q_1, X\omega_r) \vdash \begin{cases} (\omega_l, q_2, Y Z \omega_r) & \text{falls } \omega_l \neq \varepsilon, \\ (\square, q_2, Y Z \omega_r) & \text{falls } \omega_l = \varepsilon. \end{cases}$$

**Bemerkung:** Bewegt sich die TM über das ursprüngliche Eingabewort hinaus, wird diese leere besuchte Zelle durch ein Blank dargestellt.

# Beispiel: Arbeitsweise von $T_1$ beschrieben durch $\vdash$

Eingabe: 6 unär codiert (111111).

( $\square$ ,	$q_0$ ,	111111)	// Startkonfiguration
$\vdash$ ( $\square$ 1,	$q_0$ ,	11111)	// 1 gefunden, laufe weiter nach rechts
$\vdash$ ( $\square$ 11,	$q_0$ ,	1111)	// 1 gefunden, laufe weiter nach rechts
$\vdash$ ( $\square$ 111,	$q_0$ ,	111)	// 1 gefunden, laufe weiter nach rechts
$\vdash$ ( $\square$ 1111,	$q_0$ ,	11)	// 1 gefunden, laufe weiter nach rechts
$\vdash$ ( $\square$ 11111,	$q_0$ ,	1)	// 1 gefunden, laufe weiter nach rechts
$\vdash$ ( $\square$ 111111,	$q_0$ ,	$\square$ )	// $\square$ gefunden, schreibe 1, laufe nach links
$\vdash$ ( $\square$ 11111,	$q_1$ ,	11)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ 1111,	$q_1$ ,	111)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ 111,	$q_1$ ,	1111)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ 11,	$q_1$ ,	11111)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ 1,	$q_1$ ,	111111)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ ,	$q_1$ ,	1111111)	// 1 gefunden, laufe weiter nach links
$\vdash$ ( $\square$ ,	$q_1$ ,	$\square$ 1111111)	// $\square$ gefunden, laufe nach rechts und beende
$\vdash$ ( $\square$ $\square$ ,	$q_2$ ,	1111111)	// Endkonfiguration

Ausgabe: 7 unär codiert (1111111), Eingabewort akzeptiert.

# Beispiel: Arbeitsweise von $T_1$ beschrieben durch Tabelle

Zustand	Bandinhalt	Kommentar
$q_0$	$\square 111111$	// Startkonfiguration, Eingabe: 6 unär codiert
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111$	// 1 gefunden, laufe weiter nach rechts
	$\uparrow$	
$q_0$	$\square 111111\square$	// $\square$ gefunden, schreibe 1, laufe nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// 1 gefunden, laufe weiter nach links
	$\uparrow$	
$q_1$	$\square 1111111$	// $\square$ gefunden, laufe nach rechts und beende
	$\uparrow$	
$q_2$	$\square 1111111$	// Endkonfiguration, Ausgabe: 7 unär codiert, Eingabe akzeptiert.
	$\uparrow$	

# Die erweiterte Übergangsrelation

Ähnlich wie bei PDAs kann man die **erweiterte Übergangsrelation**  $\vdash^*$ , welche die Konfigurations-Änderung einer TM für keine, eine, oder mehrere Bewegungen beschreibt, induktiv definieren:

$$\vdash^* \subseteq K_T \times K_T \quad \text{mit}$$

$$k \vdash^* k, \text{ für alle } k \in K_T$$

$$k_1 \vdash^* k_n, \text{ wenn es } k_1, k_2, k_n \in K_T \text{ gibt mit} \\ k_1 \vdash k_2, k_2 \vdash^* k_n$$

**Alternativ:**  $k_1 \vdash^* k_n$ , falls Kette  $k_1 \vdash k_2, k_2 \vdash k_3, \dots, k_{n-1} \vdash k_n$  existiert.

**Beispiele für TM  $T_1$  (welche unär 1 addiert):**

- $(\square, q_0, 111) \vdash^* (\square, q_0, 111)$
- $(\square, q_0, 111) \vdash^* (\square 1, q_0, 11)$
- $(\square 111, q_0, \square) \vdash^* (\square 11, q_1, 11)$
- $(\square, q_0, 111) \vdash^* (\square \square, q_2, 1111)$

# Sprache einer Turing-Maschinen

## Definition

Die von einer Turing-Maschine  $T = (Q, \Sigma, \Pi, \delta, q_0, F)$  **akzeptierte Sprache**  $\mathcal{L}(T)$  ist definiert als

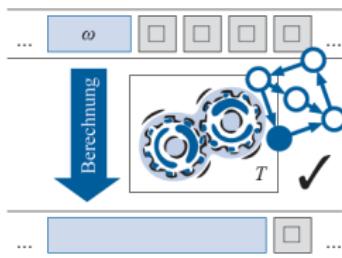
$$\begin{aligned}\mathcal{L}(T) &= \{\omega \in \Sigma^* \mid T \text{ akzeptiert } \omega\} \\ &= \{\omega \in \Sigma^* \mid \text{mit der Eingabe } \omega \text{ beendet } T \text{ die Bearbeitung nach endlich vielen Schritten in einem Finalzustand } q_f\} \\ &= \{\omega \in \Sigma^* \mid (\square, q_0, \omega) \vdash^* (\alpha, q_f, \beta), q_f \in F, \alpha, \beta \in \Pi^*\}.\end{aligned}$$

## Bemerkungen:

- $T$  **akzeptiert**  $\omega$  genau dann wenn  $T$  nach endlich vielen Schritten in einem Finalzustand  $q_f \in F$  terminiert.
- Falls  $T$   $\omega$  **nicht akzeptiert**, wird die Berechnung entweder in einem nicht-finalen Zustand  $q_x \in Q \setminus F$  beendet, oder  $T$  gerät in eine Endlosschleife.
- Der Inhalt des Bandes entscheidet nicht über Akzeptanz, links und rechts des Kopfes können beliebige Wörter  $\alpha, \beta \in \Pi^*$  stehen.

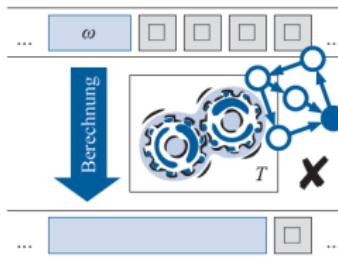
# Arbeitsweise akzeptierender Turing-Maschinen

## ■ Fall 1



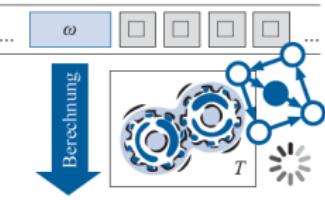
Die Turing-Maschine terminiert  
in einem Endzustand  $s \in E$   
 $\Rightarrow \omega \in \mathcal{L}(T)$

## ■ Fall 2



Die Turing-Maschine terminiert  
in einem Zustand  $s \notin E$   
 $\Rightarrow \omega \notin \mathcal{L}(T)$

## ■ Fall 3



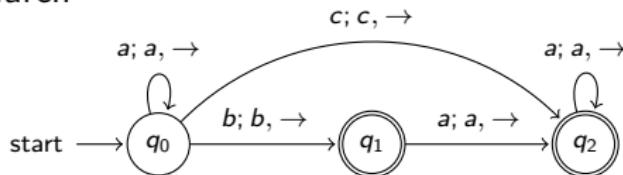
Die Turing-Maschine gerät in  
eine Endlosschleife  
 $\Rightarrow \omega \notin \mathcal{L}(T)$

Quelle: [Hoffmann, 2011]

# Beispiele zum Mitdenken: Die Turing-Maschine $T_1$

**Frage:** Welche Sprache wird von der TM  $T_1$  akzeptiert?

$T_1 = (Q, \Sigma, \Pi, \delta, q_0, F) = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, c, \square\}, q_0, \{q_1, q_2\})$   
und  $\delta$  gegeben durch



**Ansatz:** Betrachte Testwörter

- $\omega_1 = aaa$ :  $(\square, q_0, aaa) \vdash (\square a, q_0, aa) \vdash (\square aa, q_0, a) \vdash (\square aaa, q_0, \square)$   
 $\Rightarrow \omega_1$  wird nicht akzeptiert. ↗
- $\omega_2 = b$ :  $(\square, q_0, b) \vdash (\square b, q_1, \square)$   
 $\Rightarrow \omega_2$  wird akzeptiert. ✓
- $\omega_3 = acaa$ :  $(\square, q_0, acaa) \vdash (\square a, q_0, caa) \vdash (\square ac, q_2, aa) \vdash (\square aca, q_2, a) \vdash (\square acaa, q_2, \square)$   
 $\Rightarrow \omega_3$  wird akzeptiert. ✓

**Antwort:**  $\mathcal{L}(T_1) = \{a^n x a^m \mid n, m \in \mathbb{N}_0, x \in \{b, c\}\}$  (regexp:  $a^*[bc]a^*$ )

# Beispiele zum Mitdenken: Die Turing-Maschine $T_2$ I

**Frage:** Welche Sprache wird von der TM  $T_2$  akzeptiert?

$T_2 = (Q, \Sigma, \Pi, \delta, q_0, F)$  mit

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Pi = \{0, 1, x, y, \square\}$
- $F = \{q_4\}$
- $\delta$  gegeben durch

$\delta$	0	1	x	y	$\square$
$q_0$	$(q_1, x, \rightarrow)$	-	-	$(q_3, y, \rightarrow)$	-
$q_1$	$(q_1, 0, \rightarrow)$	$(q_2, y, \leftarrow)$	-	$(q_1, y, \rightarrow)$	-
$q_2$	$(q_2, 0, \leftarrow)$	-	$(q_0, x, \rightarrow)$	$(q_2, y, \leftarrow)$	-
$q_3$	-	-	-	$(q_3, y, \rightarrow)$	$(q_4, \square, \rightarrow)$
$q_4$	-	-	-	-	-

**Antwort:** Schwierig. Ansatz: Graphische Darstellung und Verarbeitung von Testworten.

## Beispiele zum Mitdenken: Die Turing-Maschine $T_2$ II

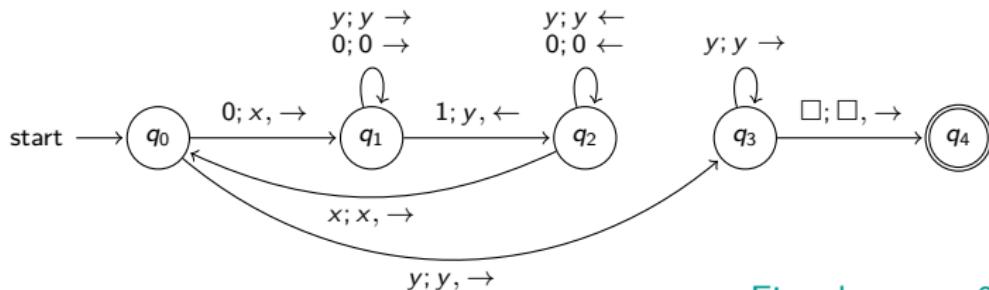


Bild: Erweitertes Zustandsübergangsdiagramm für  $T_2$

Eingabe  $\omega_1 = 01$ :

$(\square, \quad q_0, \quad 01)$
$\vdash (\square x, \quad q_1, \quad 1)$
$\vdash (\square, \quad q_2, \quad xy)$
$\vdash (\square x, \quad q_0, \quad y)$
$\vdash (\square xy, \quad q_3, \quad \square)$
$\vdash (\square xy\square, \quad q_4, \quad \square)$

$\Rightarrow \omega_1$  wird akzeptiert. ✓

$$\mathcal{L}(T_2) = \{0^n 1^n \mid n \in \mathbb{N}\}$$

Eingabe  $\omega_2 = 001$ :

$(\square, \quad q_0, \quad 001)$
$\vdash (\square x, \quad q_1, \quad 01)$
$\vdash (\square x0, \quad q_1, \quad 1)$
$\vdash (\square x, \quad q_2, \quad 0y)$
$\vdash (\square, \quad q_2, \quad x0y)$
$\vdash (\square x, \quad q_0, \quad 0y)$
$\vdash (\square xx, \quad q_1, \quad y)$
$\vdash (\square xxy, \quad q_1, \quad \square)$

$\Rightarrow \omega_2$  wird nicht akzeptiert.



Eingabe  $\omega_3 = 0011$ :

$(\square, \quad q_0, \quad 001)$
$\vdash (\square x, \quad q_1, \quad 011)$
$\vdash (\square x0, \quad q_1, \quad 11)$
$\vdash (\square x, \quad q_2, \quad 0y1)$
$\vdash (\square, \quad q_2, \quad x0y1)$
$\vdash (\square x, \quad q_0, \quad 0y1)$
$\vdash (\square xx, \quad q_1, \quad y1)$
$\vdash (\square xxy, \quad q_1, \quad 1)$
$\vdash (\square xx, \quad q_2, \quad yy)$
$\vdash (\square x, \quad q_2, \quad xyy)$
$\vdash (\square xx, \quad q_0, \quad yy)$
$\vdash (\square xxy, \quad q_3, \quad y)$
$\vdash (\square xxyy, \quad q_3, \quad \square)$
$\vdash (\square xxyy\square, \quad q_4, \quad \square)$

$\Rightarrow \omega_3$  wird akzeptiert. ✓

## Zwischenstop

**Frage:** Kann  $\mathcal{L}(T_1) = \{a^n x a^m \mid n, m \in \mathbb{N}_0, x \in \{b, c\}\}$  auch von einem PDA oder DEA akzeptiert werden?

**Antwort:** Ja. Von einem DEA. Bei genauerem Hinsehen arbeitet die TM  $T_1$  wie ein DEA, da sie nur Zustandsübergänge macht und den Bandinhalt unverändert lässt.

**Frage:** Kann  $\mathcal{L}(T_2) = \{0^n 1^n \mid n \in \mathbb{N}\}$  auch von einem PDA oder DEA akzeptiert werden?

**Antwort:** Ja. Zwar nicht von einem DEA aber von einem PDA. Dieser muss in einem Vorwärtslesezustand für jede gelesene 0 ein Zeichen im Stack speichern und in einem Rückwärtslelezustand für jede gelesene 1 ein Zeichen aus dem Stack löschen.

**Frage:** Kann jede von einer TM akzeptierte Sprache auch von einem PDA oder DEA akzeptiert werden?

**Antwort:** Nein. Siehe Beispiel nächste Folie.

# Beispiele zum Mitdenken: Eine TM für $L_{C1}$ I

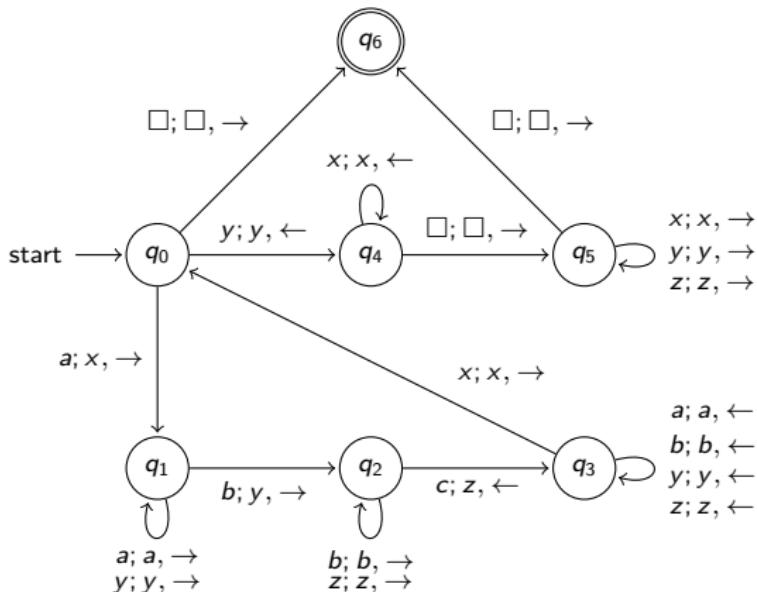
**Erinnerung:**  $L_{C1} = \{a^n b^n c^n \mid n \in \mathbb{N}_0\}$  ist vom Chomsky-Typ 1, aber nicht vom Typ 2. Daher wird  $L_{C1}$  **nicht** von einem PDA akzeptiert.

**Arbeitsweise** der TM  $T_{C1}$  mit  $\mathcal{L}(T_{C1}) = L_{C1}$

1. Ersetze erstes Zeichen des Eingabewortes ( $a$ ) durch  $x$
2. Laufe nach rechts bis zum ersten  $b$  und ersetze dies durch  $y$
3. Laufe nach rechts bis zum ersten  $c$  und ersetze dies durch  $z$
4. Laufe nach links bis zum ersten  $x$ , laufe nach rechts und wiederhole ab 2.
5. Falls kein  $a$  mehr gefunden wurde, laufe noch einmal nach links und rechts über das Band und überprüfe ob nur noch  $x, y, z$  auf dem Band stehen, falls ja, akzeptiere.

## Beispiele zum Mitdenken: Eine TM für $L_{C1} \text{ II}$

$T_{C1} = (Q, \Sigma, \Pi, \delta, q_0, F) = (\{q_0, q_1, \dots, q_6\}, \{a, b, c\}, \{a, b, c, x, y, z, \square\}, q_0, \{q_6\})$  und  $\delta$  gegeben durch



Demo im [Turing-Maschinen-Simulator](#) (Programm siehe Moodle).

## Zwischenstop

- Turing-Maschinen sind sehr mächtige Werkzeuge - viele Probleme lassen sich damit lösen, viele Funktionen berechnen
  - Beispielsweise können TMs auch genutzt werden, um minimalistische Programmiersprachen zu erstellen. Beispiele: Brainfuck, Ook!, Blub, ... (siehe [esolangs.org](http://esolangs.org))

Command	Description
>	Move the pointer to the right
<	Move the pointer to the left
+	Increment the memory cell under the pointer
-	Decrement the memory cell under the pointer
.	Output the character signified by the cell at the pointer
,	Input a character and store it in the cell at the pointer
[	Jump past the matching ] if the cell under the pointer is 0
]	Jump back to the matching [ if the cell under the pointer is nonzero

Bild: Brainfuck-Syntax Quelle: esolangs.org

**Frage:** Kann man die einfache TM erweitern oder einschränken und so ihre Fähigkeiten ändern?

```
++++++[>++++[>+>++++>+<<<<-]>+>+>->+[<]<-]>>.>---.  
+++++++.+++.>>.<-.<.+++.-----.-----.>>+.>++.
```

Bild: Hello World! in Brainfuck und Ook [Quelle](#): esolangs.org

# Einseitige beschränkte Turing-Maschinen

## Einseitig beschränkte Turing-Maschinen

- haben ein Arbeitsband, das sich (z.B von links) **nur in eine Richtung unendlich weit ausbreitet**.
- Arbeitsweise: Der Schreib-/Lesekopf kann von der linkesten Zelle nicht weiter nach links gehen, solche Anweisungen werden ignoriert.
- lassen sich von einer einfachen (unbeschränkten) TM durch Hinzufügen eines speziellen Zeichens simulieren: immer wenn die TM das Randzeichen (von rechts kommend) findet, sorgen Regeln dafür, dass nur nach rechts, nicht nach links weitergearbeitet wird.
- können eine unbeschränkte TM simulieren: immer wenn die beschränkte TM am Bandrand angelangt ist, wird der Algorithmus der TM  $T_V$  verwendet, um alle Zeichen eine Zelle weiter nach rechts zu verschieben und die Berechnung kann nach links weitergehen.

**Folgerung:** Einseitig beschränkte und unbeschränkte TM sind äquivalent und können die gleichen Probleme lösen.

# Linear beschränkte Turing-Maschinen

## Linear beschränkte Turing-Maschinen (LBAs)

- haben in Arbeitsband, mit **nur genau den  $n$  Zellen, die durch das Eingabewort benötigt werden.**
- Arbeitsweise: Der Schreib-/Lesekopf kann also weder am linken, noch am rechten Rand des Eingabewortes über das Wort hinaus laufen.
- Alternative äquivalente Definition: Zur Berechnung kann ein konstantes Vielfaches von Zellen ( $kn$ ) genutzt werden.
- lassen sich von einer unbeschränkten TM durch Hinzufügen zweier spezieller Zeichens die den Rand markieren simulieren.
- können nicht alle unbeschränkten TMs simulieren: Funktionen, für welche das Ausgabewort länger ist als das Eingabewort, können z.B. von LBAs nicht berechnet werden.

Beispiel:  $f(n) = n + 1$  mit  $n, n + 1$  unär codiert.

**Folgerung:** Linear beschränkte und unbeschränkte TM sind **nicht** äquivalent. LBA können weniger Probleme lösen als unbeschränkte TM.

# Mehrspur-Turing-Maschinen

Mehrspur-Turing-Maschinen bzw.  $k$ -Spur-Turing-Maschinen

- haben **ein unendlich langes Arbeitsband, das in  $k$  separate Spuren unterteilt ist.**
- Arbeitsweise: Der Schreib-/Lesekopf kann immer  $k$  Zellen auf einmal auslesen / beschreiben, allerdings können die Spuren nur alle gemeinsam bewegt werden.
- lassen sich von einer einfachen TM durch Nutzung der Tupel-Alphabete  $\Sigma^k, \Pi^k$  simulieren.
- können eine einfache TM durch Nutzung von nur  $k = 1$  Spur simulieren.

**Folgerung:** Mehrspur-Turing-Maschinen und einfache TM sind äquivalent und können die gleichen Probleme lösen.

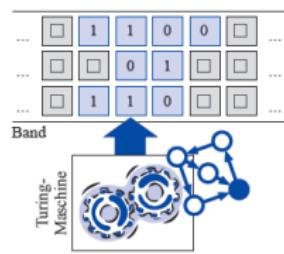


Bild: 3-Spur-TM  
[Hoffmann, 2011]

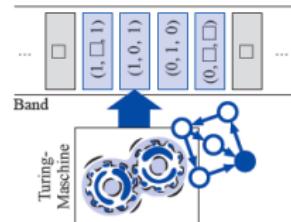


Bild: Simulation einer 3-Spur-TM  
durch eine einfache TM  
[Hoffmann, 2011]

# Mehrband-Turing-Maschinen

## Mehrband-Turing-Maschinen bzw. $k$ -Band-Turing-Maschinen

- haben  $k$  unendlich lange Arbeitsbänder, die unabhängig voneinander bewegt werden können.
- Arbeitsweise: Der Schreib-/Lesekopf kann immer  $k$  Zellen auf einmal auslesen / beschreiben, in jedem Schritt muss entschieden werden, welches Band wie bewegt wird.
- lassen sich von einer  $2k$ -Band TM durch Nutzung eines eigenen Positionsbandes pro Datenband, auf welchem die Position des  $k$ -Lesekopfes angegeben wird, simulieren.
- können eine einfache TM durch Nutzung von nur  $k = 1$  Band simulieren.

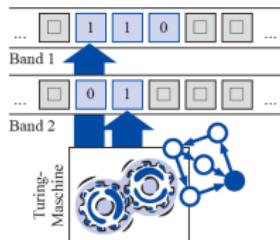


Bild: 2-Band-TM  
[Hoffmann, 2011]

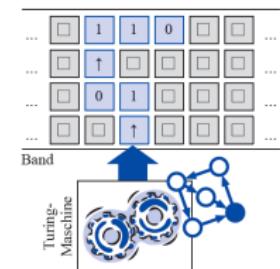


Bild: Simulation einer 2-Band-TM  
durch eine 4-Spur-TM  
[Hoffmann, 2011]

**Folgerung:** Mehrband-TM und einfache TM sind äquivalent und können die gleichen Probleme lösen.

# Nichtdeterministische Turing-Maschinen

**Frage:** Erweitert der Nichtdeterminismus die Fähigkeit von Turing-Maschinen? Können also **nichtdeterministisch arbeitende** Turing-Maschinen (NTM) eine größere Menge von Sprachen akzeptieren als deterministisch arbeitende Turing-Maschinen (DTM)?

**Antwort:** Nein.

- Wir betrachten die NTM  $T_N$ , die für jeden ihrer  $n$  Schritt maximal  $k$  Nachfolgekonfigurationen hat, insgesamt also einen Berechnungsbaum definiert.
- Um die Berechnung von  $T_N$  nachzuvollziehen, kann eine DTM  $T_D$  einfach alle möglichen Berechnungspfade (mit höchstens  $nk$  verschiedenen Konfigurationen) nacheinander simulieren.
- $T_D$  wird also die selben Worte akzeptieren wie  $T_N$  allerdings
  - ▶ mehr Zeit benötigen, da  $T_D$  alle Schritte nacheinander, nicht gleichzeitig ausführen muss
  - ▶ mehr Bandzellen benötigen, da  $T_D$  das Eingabewort und die möglichen Pfade speichern und rechnen muss

# Vergleich Turing-Maschinen Modelle

Basismodell: DETMU - Deterministische Einband TM mit unendlich langem Band

- Beschränkung: ein Bandende fest  
⇒ Keine funktionale Einschränkung: äquivalent zur DETMU
- Beschränkung: beide Bandenden fest ((N)LBA))  
⇒ Funktionale Einschränkung: nicht äquivalent zur DETMU
- Erweiterung: mehrere Spuren  
⇒ Keine funktionale Erweiterung: äquivalent zur DETMU
- Erweiterung: mehrere Bänder  
⇒ Keine funktionale Erweiterung: äquivalent zur DETMU
- Erweiterung: Nichtdeterminismus (NTM)  
⇒ Keine funktionale Erweiterung: äquivalent zur DETMU

# Turing-Maschinen und Typ-0-Sprachen

## Satz

*Die Klasse der Typ-0-Sprachen ist mit der Klasse der von Turing-Maschinen akzeptierten Sprachen identisch:  $\mathcal{L}(TM) = \mathcal{L}_0$*

### Beweisidee:

- Für jede Typ-0 Sprache  $L$  existiert eine TM  $T$  welche  $L$  akzeptiert.
  - ▶  $L$  ist Typ-0 Sprache, also existiert Typ-0-Grammatik  $G = (N, \Sigma, P, S)$  mit  $\mathcal{L}(G) = L$ .
  - ▶ Für jedes Wort  $\omega \in L$  existiert also eine Ableitungssequenz  $S \rightarrow \omega_1 \rightarrow \omega_2 \rightarrow \dots \rightarrow \omega$  mit  $\omega_i \in N \cup \Sigma$ .
  - ▶ Konstruiere  $T$  so, dass diese ausgehend von  $\omega$  nach passenden Produktionsregeln für Zeichenketten sucht und diese nichtdeterministisch ersetzt.
  - ▶ Steht das Startsymbol  $S$  auf dem Band, geht  $T$  in einen Finalzustand über.
  - ▶ Da aus jeder NTM eine DTM konstruiert werden kann, wird  $L$  also auch von einer deterministischen TM akzeptiert.

# Turing-Maschinen und Typ-0-Sprachen II

Fortsetzung der Beweisidee:

- Für jede rekursiv aufzählbare Sprache  $L$  existiert eine Typ-0 Grammatik  $G$  welche  $L$  erzeugt.
  - ▶  $L$  ist rekursiv aufzählbar, also existiert TM  $T = (Q, \Sigma, \Pi, \delta, q_0, F)$  mit  $\mathcal{L}(T) = L$ .
  - ▶ Konstruiere Typ-0 Grammatik  $G$  so, dass für jeden möglichen Konfigurationsübergang  $(\omega_l, q, \omega_r) \vdash (\omega'_l, q', \omega'_r)$  eine Regel  $(\omega_l q \omega_r) \rightarrow (\omega'_l q' \omega'_r)$  existiert.
  - ▶ Füge weitere geeignete Regeln hinzu, so dass aus dem Startsymbol  $S$  die akzeptierende Finalkonfiguration für das Wort  $\omega \in L$  hergeleitet werden kann und dieses weiter zu  $\omega$  abgeleitet werden kann.
- Wer mehr wissen will: siehe [[Hoffmann, 2011](#)].

# Turing-Maschinen und Typ-1-Sprachen

## Satz

Die Klasse der Typ-1-Sprachen ist mit der Klasse der von nichtdeterministischen, linear beschränkten Turing-Maschinen akzeptierten Sprachen identisch:  $\mathcal{L}(\text{LBA}) = \mathcal{L}_1$ .

### Bemerkungen:

- Beweis funktioniert analog zum Beweis von  $\mathcal{L}(\text{TM}) = \mathcal{L}_0$ , allerdings
  - ▶ entspricht die Bandbeschränkung der LBA der Anforderung  $|l| \leq |r|$  an Regeln  $l \rightarrow r$  vom Typ 1,
  - ▶ funktioniert der Beweis nur für nichtdeterministische LBA, offenes Problem, ob dies auch für deterministische LBA gilt.
- Generell ist die Frage offen, ob *nichtdeterministische* LBA die gleiche Sprachklasse akzeptieren wie *deterministische* LBA.

**Beispiel:** Die TM  $T_{C1}$ , die  $\mathcal{L}_{C1} = \{a^n b^n c^n \mid n \in \mathbb{N}_0\}$  akzeptiert, ist ein LBA, da sie nur auf den Bandzellen des Eingabewortes arbeitet.

# Rückblick und Zusammenfassung: die Chomsky-Hierarchie

	Sprachklasse	Erzeugung	Akzeptanz	Beispiele
Typ 3	reguläre Sprachen	reguläre Grammatik, regulärer Ausdruck	DEA NEA	$(ab)^n$ $n \in \mathbb{N}$
Typ 2	kontextfreie Sprachen	kontextfreie Grammatik	PDA	$a^n b^n$ $n \in \mathbb{N}$
Typ 1	kontextsensitive Sprachen	kontextsensitive Grammatik	LBA	$a^n b^n c^n$ $n \in \mathbb{N}$
Typ 0	rekursiv aufzählbare Sprachen	Typ-0 Grammatik	TM NTM	$L_\pi$

# Rückblick und Zusammenfassung: Automatenmodelle

Automatenmodell	Nichtdeterministisch	Deterministisch	Äquivalenz
endlicher Automat	NEA	DEA	ja ✓
Keller-automat	PDA	DPDA	nein ⚡
linear beschränkte Turing-Maschine	LBA	DLBA	offen ?
Turing-Maschine	NTM	TM	ja ✓

# Verwendete oder empfohlene Literatur I

[Hedtstück, 2012] Hedtstück, U. (2012).

*Einführung in die theoretische Informatik: formale Sprachen und Automatentheorie.*

Oldenbourg Verlag.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2011] Hoffmann, D. W. (2011).

*Theoretische Informatik.*

Carl Hanser Verlag GmbH & Co. KG, 2. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Hoffmann, 2015] Hoffmann, D. W. (2015).

*Theoretische Informatik.*

Carl Hanser Verlag GmbH & Co. KG, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

## Verwendete oder empfohlene Literatur II

[Hopcroft et al., 2011] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2011).

*Einführung in die Automatentheorie, formale Sprachen und Berechenbarkeit (bzw. Komplexitätstheorie); engl.: Introduction to automata theory, languages and computation.*

Pearson, 3. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.

[Turing, 1936] Turing, A. M. (1936).

On computable numbers, with an application to the entscheidungsproblem.

*Journal of Math*, 58(345-363):5.

## Verwendete oder empfohlene Literatur III

[Wagenknecht and Hielscher, 2014] Wagenknecht, C. and Hielscher, M. (2014).

*Formale Sprachen, abstrakte Automaten und Compiler.*

Springer Vieweg, 2. Auflage.

Als eBook in der HTWG-Bibliothek verfügbar.