

## 8c

# Intuitive und Turing-Berechenbarkeit

Prof. Dr. Jasmin Blanchette

Lehr- und Forschungseinheit für  
Theoretische Informatik und Theorembeweisen

Stand: 4. April 2024

Basiert auf Folien von PD Dr. David Sabel



# Intuitive Berechenbarkeit

---

- ▶ Was **kann** mit einem Computerprogramm berechnet werden?
- ▶ Was **kann nicht** mit einem Computerprogramm berechnet werden?
- ▶ Aus der Programmiererfahrung hat man ein gewisses Gefühl dafür, was berechnet werden kann (und was nicht).
- ▶ Das ist der intuitive Begriff der Berechenbarkeit.
- ▶ Wie beweist man, dass etwas **nicht berechnet** werden kann?  
Diese Frage ist auch von praktischem Nutzen:  
Die Suche nach passendem Algorithmus ist sinnlos.

- ▶ Berühmte Mathematiker/Informatiker versuchten in den 1930er Jahren den Begriff der Berechenbarkeit zu formalisieren.
- ▶ Dafür entwarfen sie verschiedene Modelle.
- ▶ Insbesondere sind zu nennen:  
Alan Turing (Turingmaschine) und  
Alonzo Church (Lambda-Kalkül)

# Berechenbare Funktion

---

Eine (partielle oder totale) Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  nennen wir **berechenbar**, wenn es einen Algorithmus einer modernen Programmiersprache gibt, der  $f$  berechnet.

- ▶ Bei Eingabe  $(n_1, \dots, n_k)$  stoppt der Algorithmus nach endlich vielen Berechnungsschritten und gibt den Wert von  $f(n_1, \dots, n_k)$  aus.
- ▶ Wenn  $f(n_1, \dots, n_k)$  undefiniert ist ( $f$  ist also eine partielle Funktion), dann stoppt der Algorithmus nicht.

# Beispiele für berechenbare Funktionen

---

Der Algorithmus

**Eingabe:** Zahl  $n \in \mathbb{N}$

**Beginn**

```
| solange true tue  
|   | skip
```

berechnet  $f_1 : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_1(x) = \text{undefiniert}$  für alle  $x$ .

# Beispiele für berechenbare Funktionen

Der Algorithmus

**Eingabe:** Zahl  $n \in \mathbb{N}$

**Beginn**

```
| solange true tue  
|   | skip
```

berechnet  $f_1 : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_1(x) = \text{undefiniert}$  für alle  $x$ .

Der Algorithmus

**Eingabe:** Zahlen  $n_1, n_2 \in \mathbb{N}$

**Beginn**

```
| hilf :=  $n_1 + n_2$ ;  
| return hilf
```

berechnet die Funktion  $f_2 : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_2(x, y) = x + y$ .

# Beispiele für berechenbare Funktionen

Der Algorithmus

**Eingabe:** Zahl  $n \in \mathbb{N}$

**Beginn**

```
| solange true tue  
|   | skip
```

berechnet  $f_1 : \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_1(x) =$  undefiniert für alle  $x$ .

Der Algorithmus

**Eingabe:** Zahlen  $n_1, n_2 \in \mathbb{N}$

**Beginn**

```
| hilf :=  $n_1 + n_2$ ;  
| return hilf
```

berechnet die Funktion  $f_2 : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  mit  $f_2(x, y) = x + y$ .

$f_1$  und  $f_2$  sind daher berechenbar.

# Beispiele für berechenbare Funktionen

---

$$f_3(n) = \begin{cases} 1 & \text{falls } n \text{ ein Präfix der Ziffern der Dezimalzahldarstellung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Z.B. gilt

- ▶  $f_3(31) = 1$  und  $f_3(314) = 1$
- ▶  $f_3(2) = 0$  und  $f_3(315) = 0$

Ist  $f_3$  berechenbar?



# Beispiele für berechenbare Funktionen

---

$$f_3(n) = \begin{cases} 1 & \text{falls } n \text{ ein Präfix der Ziffern der Dezimalzahldarstellung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Z.B. gilt

- ▶  $f_3(31) = 1$  und  $f_3(314) = 1$
- ▶  $f_3(2) = 0$  und  $f_3(315) = 0$

Ist  $f_3$  berechenbar?

**Ja.** Sei  $n$  eine  $k$ -stellige Zahl. Es gibt Algorithmen, die die ersten  $k$  Stellen von  $\pi$  berechnen. Danach kann man vergleichen.

# Beispiele für berechenbare Funktionen

---

$$f_4(n) = \begin{cases} 1 & \text{falls } n \text{ ein Teilwort der Ziffern der Dezimalzahldarstellung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_4$  berechenbar?

# Beispiele für berechenbare Funktionen

---

$$f_4(n) = \begin{cases} 1 & \text{falls } n \text{ ein Teilwort der Ziffern der Dezimalzahldarstellung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_4$  berechenbar?

Unklar. Es gibt die Vermutung, dass jede beliebige Ziffernfolge irgendwann als Folge in  $\pi$  auftaucht, aber die Frage ist bisher ungelöst.

## Beispiele für berechenbare Funktionen

---

$$f_5(n) = \begin{cases} 1 & \text{falls die Dezimalzahldarstellung von } \pi \text{ das Wort } 3^n \text{ als Teilwort enthält} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_5$  berechenbar?

# Beispiele für berechenbare Funktionen

$$f_5(n) = \begin{cases} 1 & \text{falls die Dezimalzahldarstellung von } \pi \text{ das Wort } 3^n \text{ als Teilwort enthält} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_5$  berechenbar?

Ja.

- ▶ Entweder  $f_5(n) = 1$  für alle  $n$
- ▶ oder es gibt  $n_0$ , sodass  $\pi$   $3^{n_0}$  als Teilwort hat, aber alle Teilwörter  $3^n$  mit  $n > n_0$  nicht mehr besitzt.

$$\text{Dann ist } f_5(n) = \begin{cases} 1 & \text{falls } n \leq n_0 \\ 0 & \text{falls } n > n_0 \end{cases}$$

- ▶ Für beide Fälle können wir Algorithmen angeben, die  $f_5$  berechnen.

# Beispiele für berechenbare Funktionen

$$f_5(n) = \begin{cases} 1 & \text{falls die Dezimalzahldarstellung von } \pi \text{ das Wort } 3^n \text{ als Teilwort enthält} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_5$  berechenbar?

Ja.

- ▶ Entweder  $f_5(n) = 1$  für alle  $n$
- ▶ oder es gibt  $n_0$ , sodass  $\pi$   $3^{n_0}$  als Teilwort hat, aber alle Teilwörter  $3^n$  mit  $n > n_0$  nicht mehr besitzt.

$$\text{Dann ist } f_5(n) = \begin{cases} 1 & \text{falls } n \leq n_0 \\ 0 & \text{falls } n > n_0 \end{cases}$$

- ▶ Für beide Fälle können wir Algorithmen angeben, die  $f_5$  berechnen.

Beachte: Unsere Definition von Berechenbarkeit ist **nicht konstruktiv**.

Wir müssen keinen Algorithmus liefern, sondern nur einen Beweis, dass einer existiert.

# Beispiele für berechenbare Funktionen

---

$$f_6(n) = \begin{cases} 1 & \text{falls deterministische LBAs genau die gleichen} \\ & \text{Sprachen erkennen wie nichtdeterministische LBAs} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_6$  berechenbar?

# Beispiele für berechenbare Funktionen

---

$$f_6(n) = \begin{cases} 1 & \text{falls deterministische LBAs genau die gleichen} \\ & \text{Sprachen erkennen wie nichtdeterministische LBAs} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f_6$  berechenbar?

Ja. Entweder ist  $f_6(x) = 1$  oder ist  $f_6(x) = 0$ .

Beide Funktionen sind berechenbar

(unabhängig davon, ob das 1. LBA-Problem eine positive oder negative Lösung hat).



# Beispiele für berechenbare Funktionen

---

$$f^r(n) = \begin{cases} 1 & \text{falls } n \text{ ein Präfix der Ziffern der Dezimalzahldarstellung von } r \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f^r$  für jedes  $r \in \mathbb{R}$  berechenbar?

# Beispiele für berechenbare Funktionen

---

$$f^r(n) = \begin{cases} 1 & \text{falls } n \text{ ein Präfix der Ziffern der Dezimalzahldarstellung von } r \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Ist  $f^r$  für jedes  $r \in \mathbb{R}$  berechenbar?

**Nein.** Wir bräuchten genauso viele verschiedene Algorithmen wie es reelle Zahlen gibt. Es gibt nur abzählbar viele Algorithmen einer Programmiersprache, aber überabzählbar viele reelle Zahlen.

# Churchsche These

---

Im Folgenden untersuchen wir Modelle zur Berechenbarkeit:

- ▶ Turingmaschinen (in dieser Stunde)
- ▶ WHILE-Programme (nur FSK)
- ▶ GOTO-Programme (nur FSK)
- ▶  $\mu$ -rekursive Funktionen (nur FSK)

# Churchsche These

---

Im Folgenden untersuchen wir Modelle zur Berechenbarkeit:

- ▶ Turingmaschinen (in dieser Stunde)
- ▶ WHILE-Programme (nur FSK)
- ▶ GOTO-Programme (nur FSK)
- ▶  $\mu$ -rekursive Funktionen (nur FSK)

Alle führen zum [selben Begriff](#) der Berechenbarkeit.

## Churchsche These

Die Klasse der turingberechenbaren (äquivalent WHILE-berechenbaren, GOTO-berechenbaren,  $\mu$ -rekursiven) Funktionen stimmt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

# Churchsche These

---

Im Folgenden untersuchen wir Modelle zur Berechenbarkeit:

- ▶ Turingmaschinen (in dieser Stunde)
- ▶ WHILE-Programme (nur FSK)
- ▶ GOTO-Programme (nur FSK)
- ▶  $\mu$ -rekursive Funktionen (nur FSK)

Alle führen zum **selben Begriff** der Berechenbarkeit.

## Churchsche These

Die Klasse der turingberechenbaren (äquivalent WHILE-berechenbaren, GOTO-berechenbaren,  $\mu$ -rekursiven) Funktionen stimmt genau mit der Klasse der intuitiv berechenbaren Funktionen überein.

Die Churchsche These kann man nicht beweisen, da der Begriff „intuitiv berechenbar“ nicht formal gefasst werden kann.

# Wiederholung: Turingmaschinen

- ▶ Eine Turingmaschine ist ein 7-Tupel  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  mit Zuständen  $Z$ , Eingabealphabet  $\Sigma$ , Bandalphabet  $\Gamma \supset \Sigma$ , Blank-Symbol  $\square$ , Startzustand  $z_0$ , Endzuständen  $E \subseteq Z$  und Überföhrungsfunktion  $\delta$ .
- ▶ DTM:  $\delta : (Z \setminus E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$
- ▶ NTM:  $\delta : (Z \setminus E) \times \Gamma \rightarrow \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$
- ▶ TM-Konfiguration  $a_1 \cdots a_m z a_{m+1} \cdots a_n$ , wobei  $a_1 \cdots a_n \in \Gamma^*$  der entdeckte Teil des Bands ist, TM in Zustand  $z$  ist und der Schreib-Lesekopf unter  $a_{m+1}$  ist.

## Definition

Eine Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  heißt **turingberechenbar**, falls es eine deterministische Turingmaschine  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  gibt, sodass für alle  $u, v \in \Sigma^*$  gilt:

$$f(u) = v$$

g.d.w.

es gibt  $z \in E$ , sodass  $Start_M(u) \vdash^* \square \dots \square z v \square \dots \square$

## Definition

Eine Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  heißt **turingberechenbar**, falls es eine deterministische Turingmaschine  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  gibt, sodass für alle  $u, v \in \Sigma^*$  gilt:

$$f(u) = v$$

g.d.w.

es gibt  $z \in E$ , sodass  $Start_M(u) \vdash^* \square \dots \square z v \square \dots \square$

Eine Konsequenz der Definition ist:

Falls  $f(n_1, \dots, n_k)$  undefiniert ist, dann kann die Maschine ewig laufen.



## Definition

Eine Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  heißt **turingberechenbar**, falls es eine deterministische Turingmaschine  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  gibt, sodass für alle  $n_1, \dots, n_k, m \in \mathbb{N}$  gilt:

$$f(n_1, \dots, n_k) = m$$

g.d.w.

es gibt  $z \in E$ , sodass  $z_0 \text{bin}(n_1) \# \dots \# \text{bin}(n_k) \vdash^* \square \dots \square z \text{bin}(m) \square \dots \square$

wobei  $\text{bin}(n)$  die Binärzahldarstellung von  $n \in \mathbb{N}$  ist.

Eine Konsequenz der Definition ist:

Falls  $f(u)$  undefiniert ist, dann kann die Maschine ewig laufen.

# Beispiele für turingberechenbare Funktionen

---

## Nachfolgerfunktion

Die Funktion  $f(x) = x + 1$  für alle  $x \in \mathbb{N}$  ist turingberechenbar.  
Wir haben ein Beispiel bereits gesehen.

# Beispiele für turingberechenbare Funktionen

## Nachfolgerfunktion

Die Funktion  $f(x) = x + 1$  für alle  $x \in \mathbb{N}$  ist turingberechenbar.  
Wir haben ein Beispiel bereits gesehen.

## Identitätsfunktion

Die Funktion  $f(x) = x$  für alle  $x \in \mathbb{N}$  ist turingberechenbar:  
Für die Turingmaschine  $M = (\{z_0\}, \{0, 1, \#\}, \{0, 1, \#, \square\}, \delta, z_0, \square, \{z_0\})$  mit  $\delta(z_0, a) = (z_0, a, N)$  für alle  $a \in \{0, 1, \#, \square\}$  gilt:  $z_0 \text{bin}(n) \vdash^* z_0 \text{bin}(n)$  für alle  $n \in \mathbb{N}$ .

# Beispiele für turingberechenbare Funktionen

## Nachfolgerfunktion

Die Funktion  $f(x) = x + 1$  für alle  $x \in \mathbb{N}$  ist turingberechenbar.  
Wir haben ein Beispiel bereits gesehen.

## Identitätsfunktion

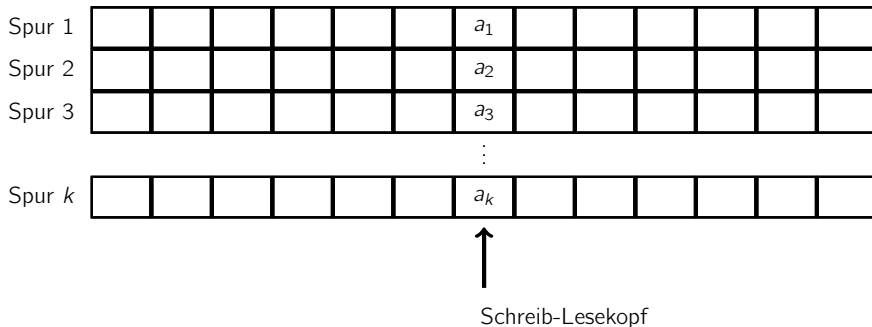
Die Funktion  $f(x) = x$  für alle  $x \in \mathbb{N}$  ist turingberechenbar:  
Für die Turingmaschine  $M = (\{z_0\}, \{0, 1, \#\}, \{0, 1, \#, \square\}, \delta, z_0, \square, \{z_0\})$  mit  $\delta(z_0, a) = (z_0, a, N)$  für alle  $a \in \{0, 1, \#, \square\}$  gilt:  $z_0 \text{bin}(n) \vdash^* z_0 \text{bin}(n)$  für alle  $n \in \mathbb{N}$ .

## Überall undefinierte Funktion

Die Funktion  $f(x) = \text{undefiniert}$  für alle  $x$  ist turingberechenbar, da die Turingmaschine  $M = (\{z_0\}, \{0, 1, \#\}, \{0, 1, \#, \square\}, \delta, z_0, \square, \emptyset)$  mit  $\delta(z_0, a) = (z_0, a, N)$  für keine Eingabe akzeptiert (sondern stets in eine Endlosschleife geht).

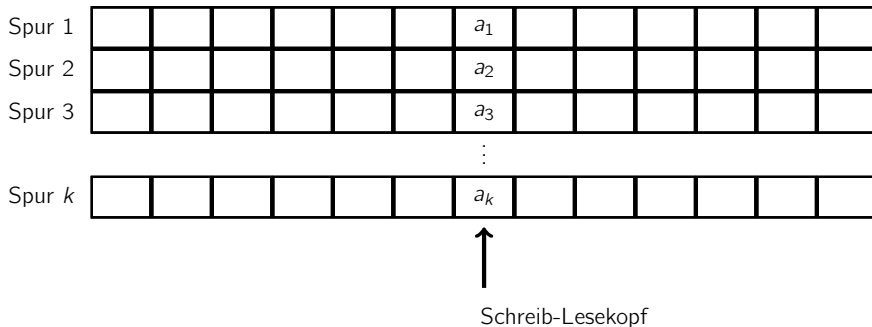
# Mehrspuren-Turingmaschinen

Erweiterung: Das Band hat  $k$  Spuren:



# Mehrspuren-Turingmaschinen

Erweiterung: Das Band hat  $k$  Spuren:



Mehrspuren-Turingmaschinen machen manche Konstruktionen einfacher.

## Definition

Eine  $k$ -Spuren-Turingmaschine (für  $k \in \mathbb{N}_{>0}$ ) ist ein 7-Tupel  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ , wobei:

- ▶  $Z$  ist eine endliche Menge von Zuständen
- ▶  $\Sigma$  ist das (endliche) Eingabealphabet
- ▶  $\Gamma \supset \Sigma$  ist das (endliche) Bandalphabet
- ▶  $\delta$  ist die Überföhrungsfunktion
  - ▶ für DTM:  $\delta : (Z \setminus E) \times \Gamma^k \rightarrow Z \times \Gamma^k \times \{L, R, N\}$
  - ▶ für NTM:  $\delta : (Z \setminus E) \times \Gamma^k \rightarrow \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\})$
- ▶  $z_0 \in Z$  ist der Startzustand
- ▶  $\square \in \Gamma \setminus \Sigma$  ist das Blank-Symbol
- ▶  $E \subseteq Z$  ist die Menge der Endzustände.

# Mehrspuren-Turingmaschinen

---

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## **Satz**

Jede Mehrspuren-Turingmaschine kann von einer 1-Spur-Turingmaschine simuliert werden.



# Mehrspuren-Turingmaschinen

---

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## Satz

Jede Mehrspuren-Turingmaschine kann von einer 1-Spur-Turingmaschine simuliert werden.

**Beweis** Konstruktion einer 1-Spur-TM mit  $\Gamma \cup \Gamma^k$  als Bandalphabet,  $\Sigma$  als Eingabealphabet und  $\square$  als Blank-Symbol:

# Mehrspuren-Turingmaschinen

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## Satz

Jede Mehrspuren-Turingmaschine kann von einer 1-Spur-Turingmaschine simuliert werden.

**Beweis** Konstruktion einer 1-Spur-TM mit  $\Gamma \cup \Gamma^k$  als Bandalphabet,  $\Sigma$  als Eingabealphabet und  $\square$  als Blank-Symbol:

1. Aus Eingabe  $w \in \Sigma^*$  erzeuge die Mehrspurendarstellung:  
Ersetze  $a \in \Sigma$  durch das  $k$ -Tupel  $(a, \square, \dots, \square)$ .

# Mehrspuren-Turingmaschinen

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## Satz

Jede Mehrspuren-Turingmaschine kann von einer 1-Spur-Turingmaschine simuliert werden.

**Beweis** Konstruktion einer 1-Spur-TM mit  $\Gamma \cup \Gamma^k$  als Bandalphabet,  $\Sigma$  als Eingabealphabet und  $\square$  als Blank-Symbol:

1. Aus Eingabe  $w \in \Sigma^*$  erzeuge die Mehrspurendarstellung:  
Ersetze  $a \in \Sigma$  durch das  $k$ -Tupel  $(a, \square, \dots, \square)$ .
2. Simuliere anschließend die Mehrspurenmaschine.

# Mehrspuren-Turingmaschinen

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## Satz

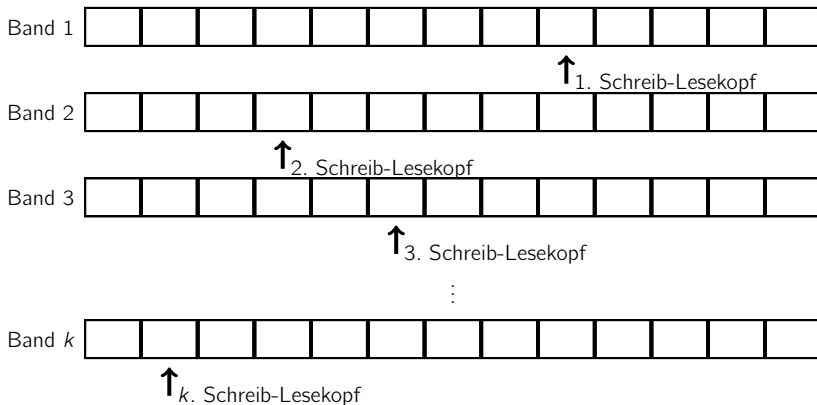
Jede Mehrspuren-Turingmaschine kann von einer 1-Spur-Turingmaschine simuliert werden.

**Beweis** Konstruktion einer 1-Spur-TM mit  $\Gamma \cup \Gamma^k$  als Bandalphabet,  $\Sigma$  als Eingabealphabet und  $\square$  als Blank-Symbol:

1. Aus Eingabe  $w \in \Sigma^*$  erzeuge die Mehrspurendarstellung:  
Ersetze  $a \in \Sigma$  durch das  $k$ -Tupel  $(a, \square, \dots, \square)$ .
2. Simuliere anschließend die Mehrspurenmaschine.
3. Nach Akzeptanz der Mehrspurenmaschine:  
Erzeuge 1-Spur-Darstellung, d.h. ersetze alle  $k$ -Tupel  $(a_1, \dots, a_k)$  durch  $a_1$ .  $\square$

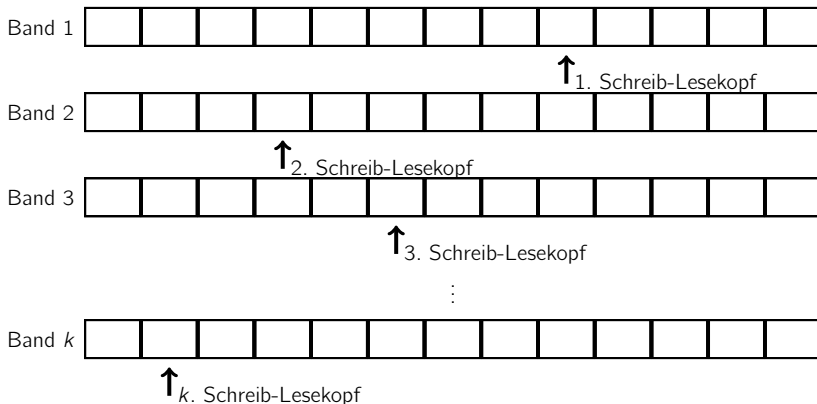
# Mehrband-Turingmaschinen

Erweiterung: Die Schreib-Leseköpfe bewegen sich unabhängig.



# Mehrband-Turingmaschinen

Erweiterung: Die Schreib-Leseköpfe bewegen sich unabhängig.



Mehrband-Turingmaschinen machen manche Konstruktionen noch einfacher.

## Definition

Eine  $k$ -Band-Turingmaschine (für  $k \in \mathbb{N}_{>0}$ ) ist ein 7-Tupel  $(Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  mit

- ▶  $Z$  ist eine endliche Menge von Zuständen
- ▶  $\Sigma$  ist das (endliche) Eingabealphabet
- ▶  $\Gamma \supset \Sigma$  ist das (endliche) Bandalphabet
- ▶  $\delta$  ist die Überföhrungsfunktion
  - ▶ für DTM:  $\delta : (Z \setminus E) \times \Gamma^k \rightarrow Z \times \Gamma^k \times \{L, R, N\}^k$
  - ▶ für NTM:  $\delta : (Z \setminus E) \times \Gamma^k \rightarrow \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$
- ▶  $z_0 \in Z$  ist der Startzustand
- ▶  $\square \in \Gamma \setminus \Sigma$  ist das Blank-Symbol
- ▶  $E \subseteq Z$  ist die Menge der Endzustände.

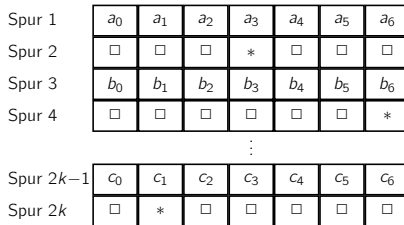
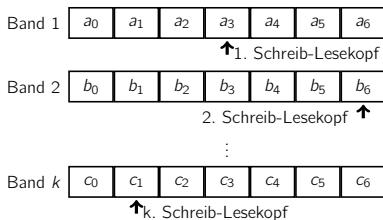
# Mehrband-Turingmaschinen

Für Berechenbarkeit erfolgt die Ein- und Ausgabe auf der ersten Spur.  
Anfangs stehen alle anderen leer.

## Theorem

Jede Mehrband-Turingmaschine kann von einer 1-Band-Turingmaschine simuliert werden.

**Beweis** Sei  $M$  eine  $k$ -Band-TM. Wir simulieren  $M$  mit einer  $2k$ -Spuren-TM  $M'$ , die sich wiederum mit einer 1-Spuren-TM simulieren lässt.





# Mehrband-Turingmaschinen

---

Die Eingabe  $w \in \Sigma^*$  liegt auf dem Eingabeband von  $M'$ .

# Mehrband-Turingmaschinen

---

Die Eingabe  $w \in \Sigma^*$  liegt auf dem Eingabeband von  $M'$ .

1. Erzeuge Darstellung in  $2k$  Spuren.

# Mehrband-Turingmaschinen

---

Die Eingabe  $w \in \Sigma^*$  liegt auf dem Eingabeband von  $M'$ .

1. Erzeuge Darstellung in  $2k$  Spuren.
2. Simuliere anschließend Berechnungsschritte von  $M$ . Für jeden Schritt:
  - 2.1 Lies den verwendeten Bandbereich von links nach rechts, um die Kopfpositionen zu finden.
  - 2.2 Speichere dabei alle  $k$  Bandinhalte an den Kopfpositionen durch Zustände.
  - 2.3 Führe den Schritt von  $M$  aus, durch Anpassen der Bandinhalte und der Kopfpositionen.

# Mehrband-Turingmaschinen

---

Die Eingabe  $w \in \Sigma^*$  liegt auf dem Eingabeband von  $M'$ .

1. Erzeuge Darstellung in  $2k$  Spuren.
2. Simuliere anschließend Berechnungsschritte von  $M$ . Für jeden Schritt:
  - 2.1 Lies den verwendeten Bandbereich von links nach rechts, um die Kopfpositionen zu finden.
  - 2.2 Speichere dabei alle  $k$  Bandinhalte an den Kopfpositionen durch Zustände.
  - 2.3 Führe den Schritt von  $M$  aus, durch Anpassen der Bandinhalte und der Kopfpositionen.
3. Bei Akzeptanz durch  $M$  transformiere die Spurendarstellung in die Darstellung der Ausgabe. □