

# Grundbegriffe der Theoretischen Informatik

Sommersemester 2018 - Thomas Schwentick

Teil C: Berechenbarkeit und Entscheidbarkeit

13: Die Church-Turing-These

Version von: 7. Juni 2018 (14:20)

# Plan

- Alle im letzten Kapitel betrachteten Berechnungsmodelle sind gleichmächtig:
  - WHILE-Programme
  - GOTO-Programme
  - Turingmaschinen
- Die Church-Turing-These besagt, dass diese (und andere) Modelle gerade die intuitiv berechenbaren Funktionen erfassen
- Außerdem:
  - Turingmaschinen mit mehreren Strings können durch 1-String-Turingmaschinen simuliert werden

# Inhalt

## ▷ 13.1 WHILE vs. GOTO

13.2 Mehrstring-Turingmaschinen

13.3 Turingmaschinen und WHILE/GOTO-Programme

13.4 Die Church-Turing-These

13.5 Entscheidbarkeit und Berechenbarkeit: Definition

# GOTO $\rightarrow$ WHILE

## Satz 13.1

- Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar

## Beweisidee

$M_1 : A_1;$   
 $M_2 : A_2;$   
 $\vdots$   
 $M_k : A_k$

$\rightarrow$

```

 $x_z := 1;$ 
WHILE  $x_z \neq 0$  DO
    IF  $x_z = 1$  THEN  $A'_1$  END;
    IF  $x_z = 2$  THEN  $A'_2$  END;
     $\vdots$ 
    IF  $x_z = k$  THEN  $A'_k$  END;
    IF  $x_z = k + 1$  THEN  $x_z := 0$ 
END
    
```

$$\bullet A_n = x_i := x_j + c \quad \Rightarrow \quad A'_n = x_i := x_j + c; \quad x_z := x_z + 1$$

$$\bullet A_n = x_i := x_j \div c \quad \Rightarrow \quad A'_n = x_i := x_j \div c; \quad x_z := x_z + 1$$

$$\bullet A_n = \text{IF } x_i = c \text{ THEN GOTO } M_j \Rightarrow A'_n = x_z := x_z + 1; \text{ IF } x_i = c \text{ THEN } x_z := j \text{ END}$$

$$\bullet A_n = \text{HALT} \quad \Rightarrow \quad A'_n = x_z := 0$$

# WHILE $\rightarrow$ GOTO

## Satz 13.2

- Jede WHILE-berechenbare Funktion ist auch GOTO-berechenbar

## Beweisidee

- Wertzuweisungen müssen nicht übersetzt werden
- Reihung von  $P_1$  und  $P_2$  lässt sich durch Konkatination und Ersetzung der HALT-Anweisungen in  $P_1$  übersetzen
- Ein Teilprogramm

WHILE  $x_i \neq 0$  DO  $P$  END

kann durch

$M_1$  : IF  $x_i = 0$  THEN GOTO  $M_2$ ;  
           $P'$ ; GOTO  $M_1$ ;  
 $M_2$  :  $x_i := x_i$

simuliert werden

- Kleines Fazit:
  - Die Klasse der WHILE-berechenbaren Funktionen ist also gleich der Klasse der GOTO-berechenbaren Funktionen
- Es gilt sogar:
  - Jede WHILE-berechenbare Funktion ist durch ein WHILE-Programm mit nur *einer* WHILE-Schleife (aber mehreren IF-Anweisungen) berechenbar

# Inhalt

13.1 WHILE vs. GOTO

▷ **13.2 Mehrstring-Turingmaschinen**

13.3 Turingmaschinen und WHILE/GOTO-Programme

13.4 Die Church-Turing-These

13.5 Entscheidbarkeit und Berechenbarkeit: Definition

# Mehrstring-Turingmaschinen: Beispiel

- Um die Umwandlung von WHILE-Programmen in Turingmaschinen zu erleichtern, gönnen wir uns etwas mehr Komfort:
  - Turingmaschinen mit mehreren Strings
- Wichtig: Bei jeder *einzelnen* Turingmaschine ist die Anzahl der Strings fest

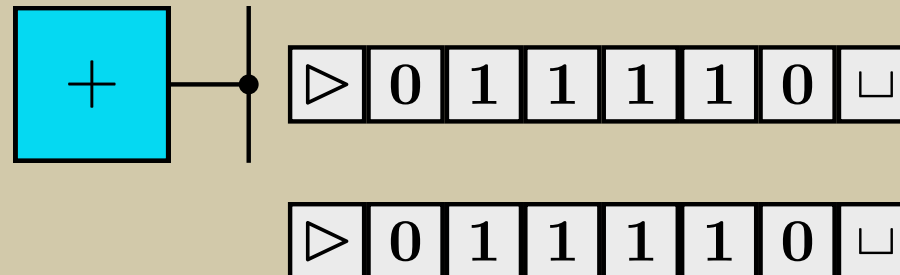
## Beispiel

2-String-Turingmaschine zum Test, ob die Eingabe von der Form  $ww^R$  ist:

**a:** Kopiere Eingabewort vom ersten auf den zweiten String (bis  $\sqcup$ )

**b/c:** Bewege Kopf 2 zurück an Anfang, teste dabei, ob die Anzahl der Zeichen gerade oder ungerade ist

**d:** Bewege Kopf 1 nach links, Kopf 2 nach rechts, vergleiche jeweils die gelesenen Zeichen



# Mehrstring-Turingmaschinen: Transitionsfunktion

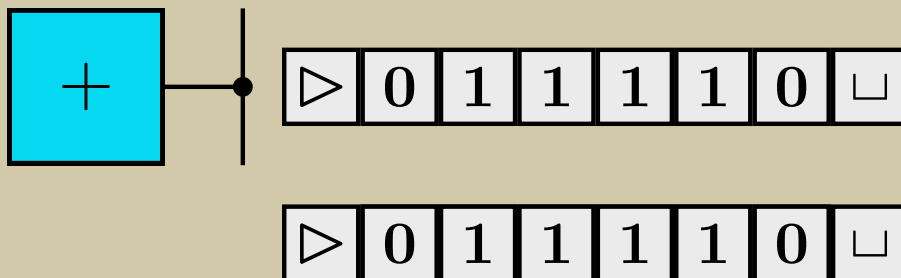
## Beispiel

2-String-Turingmaschine zum Test, ob die Eingabe von der Form  $ww^R$  ist:

**a:** Kopiere Eingabewort vom ersten auf den zweiten String (bis  $\sqcup$ )

**b/c:** Bewege Kopf 2 zurück an Anfang, teste dabei, ob die Anzahl der Zeichen gerade oder ungerade ist

**d:** Bewege Kopf 1 nach links, Kopf 2 nach rechts, vergleiche jeweils die gelesenen Zeichen







## Beispiel

Vorher			Nachher				
$q$	$\gamma_1$	$\gamma_2$	$q$	$\gamma_1$	$\gamma_2$	$d_1$	$d_2$
$a$	$\triangleright$	$\triangleright$	$a$	$\triangleright$	$\triangleright$	$\rightarrow$	$\rightarrow$
$a$	0	$\sqcup$	$a$	0	0	$\rightarrow$	$\rightarrow$
$a$	1	$\sqcup$	$a$	1	1	$\rightarrow$	$\rightarrow$
$a$	$\sqcup$	$\sqcup$	$b$	$\sqcup$	$\sqcup$	$\downarrow$	$\leftarrow$
$b$	$\sqcup$	0	$c$	$\sqcup$	0	$\downarrow$	$\leftarrow$
$c$	$\sqcup$	1	$b$	$\sqcup$	1	$\downarrow$	$\leftarrow$
$b$	$\sqcup$	1	$c$	$\sqcup$	1	$\downarrow$	$\leftarrow$
$c$	$\sqcup$	0	$b$	$\sqcup$	0	$\downarrow$	$\leftarrow$
$b$	$\sqcup$	$\triangleright$	$d$	$\sqcup$	$\triangleright$	$\leftarrow$	$\rightarrow$
$d$	0	0	$d$	0	0	$\leftarrow$	$\rightarrow$
$d$	1	1	$d$	1	1	$\leftarrow$	$\rightarrow$
$d$	$\triangleright$	$\sqcup$	+	$\triangleright$	$\sqcup$	$\downarrow$	$\downarrow$



# Mehrstring-Turingmaschinen: Definition (1/2)

## Definition (Mehrstring-TM (Syntax))

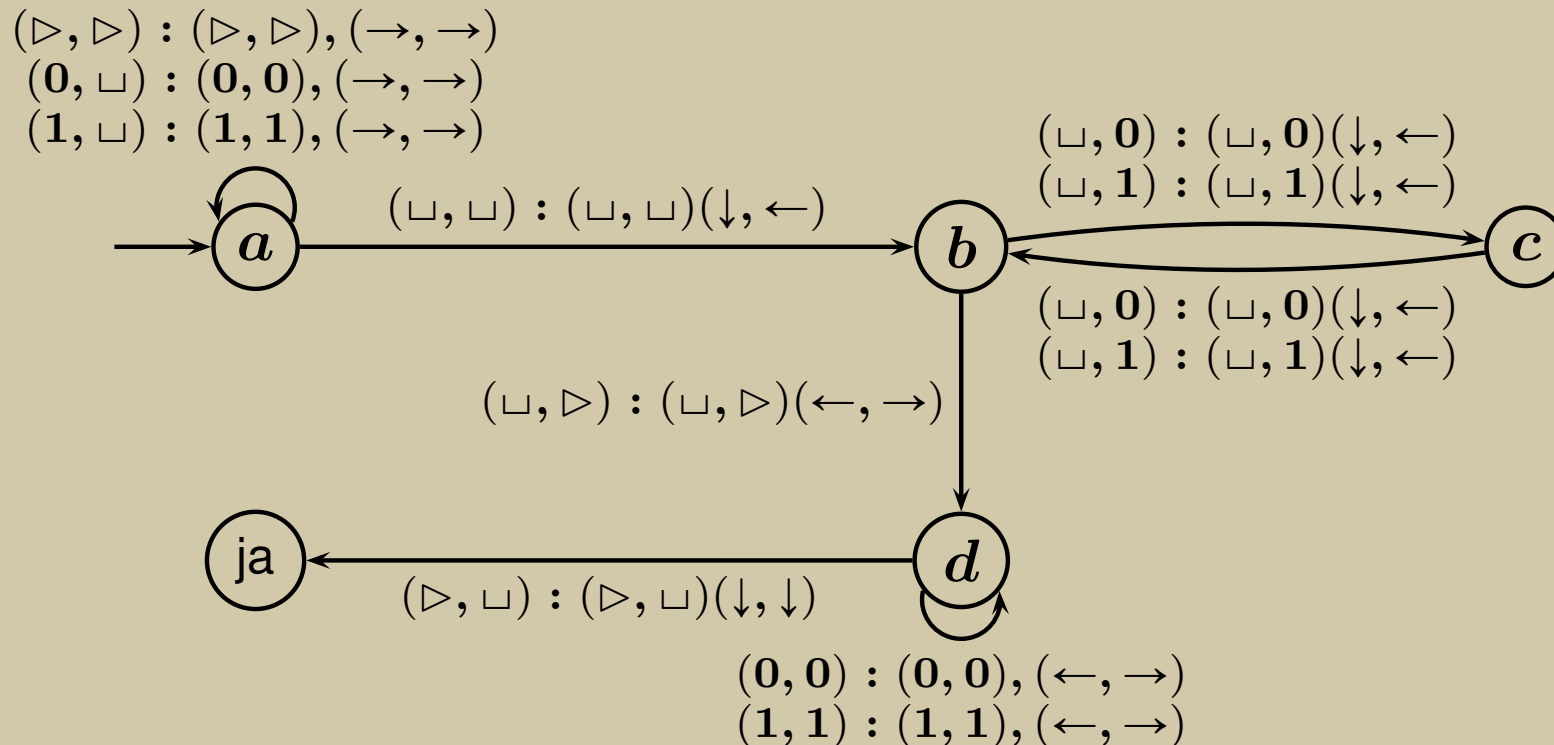
- Sei  $k \geq 1$
- Eine  $k$ -String-Turingmaschine  $M = (Q, \Gamma, \delta, s)$  besteht aus
  - einer endlichen Menge  $Q$ ,  Zustandsmenge
  - einem Alphabet  $\Gamma$ , mit  $\sqcup, \triangleright \in \Gamma$ ,  Arbeitsalphabet
  - einer Funktion
$$\delta : Q \times \Gamma^k \rightarrow (Q \cup \{h, \text{ja}, \text{nein}\}) \times \Gamma^k \times \{\leftarrow, \downarrow, \rightarrow\}^k$$
 Transitionsfunktion
- einem ausgezeichneten Zustand  $s \in Q$   Startzustand
- Dabei seien  $Q, \Gamma, \{h, \text{ja}, \text{nein}\}$  und  $\{\leftarrow, \downarrow, \rightarrow\}$  paarweise disjunkt

## Bemerkungen

- Die Anzahl  $k$  der Strings ist implizit durch  $\delta$  gegeben
- Wenn es auf das genaue  $k$  nicht ankommt, sagen wir auch Mehrstring-Turingmaschine statt  $k$ -String-Turingmaschine

# Mehrstring-Turingmaschinen: Diagrammdarstellung

## Beispiel-TM in Diagramm-Darstellung



- In diesem Beispiel gilt die Konvention:
  - Ist für  $(q, \sigma_1, \dots, \sigma_k) \in Q \times \Gamma^k$  kein Übergang eingezeichnet, so sei  $\delta(q, \sigma_1, \dots, \sigma_k) \stackrel{\text{def}}{=} (\text{nein}, \sigma_1, \dots, \sigma_k, \downarrow, \dots, \downarrow)$

# Mehrstring-Turingmaschinen: Definition (2/2)

## Definition (Mehrstring-TM (Semantik))

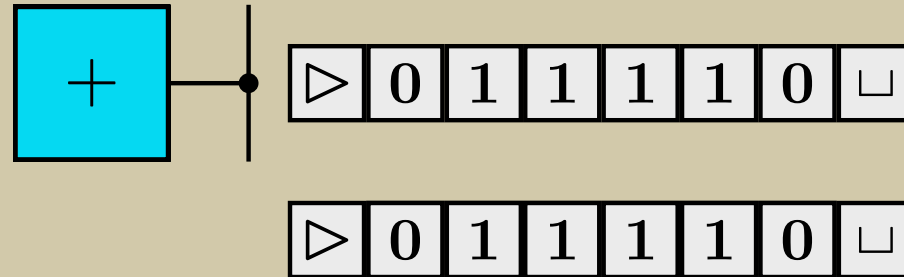
- Sei  $k \geq 1$  und  $M = (Q, \Gamma, \delta, s)$  eine  $k$ -string-TM
- **Ein-/Ausgabealphabet:**  
 $\Sigma \subseteq \Gamma - \{\sqcup, \triangleright\},$
- **Konfiguration** von  $M$ :  $k + 1$ -Tupel  $(q, s_1, \dots, s_k)$ , wobei
  - $q \in Q$
  - $s_i$  String-Zeiger-Beschreibung für  $i$ -ten String
- **Startkonfiguration**  $K_0(u)$  von  $M$  bei Eingabe  $u \in \Sigma^*$ :  
 $(s, (\triangleright u, 1), (\triangleright, 1), \dots, (\triangleright, 1))$
- $(q, s_1, \dots, s_k)$  ist **Haltekonfiguration**, falls  $q \in \{h, \text{ja}, \text{nein}\}$

## Definition (Forts.)

- Sei  $K = (q, (u_1, z_1), \dots, (u_k, z_k))$  eine Konfiguration von  $M$  und sei, für jedes  $i$ ,  $\sigma_i \stackrel{\text{def}}{=} w[i]$
- Ist  $\delta(q, \sigma_1, \dots, \sigma_k) = (q', \tau_1, \dots, \tau_k, d_1, \dots, d_k)$ , so ist  $K' = (q', (u'_1, z'_1), \dots, (u'_k, z'_k))$  die **Nachfolgekonfiguration von  $K$** , wenn für alle  $i$  gilt:
  - $z'_i = z_i + 1$ , falls  $d_i = \rightarrow$ ,
  - $z'_i = z_i$ , falls  $d_i = \downarrow$ ,
  - $z'_i = z_i - 1$ , falls  $d_i = \leftarrow$ ,
  - $u'_i = u_i[z_i/\tau_i]\sqcup$ , falls  $z_i = |u'_i|$  und  $d_i = \rightarrow$ ,
  - $u'_i = u'_i[z_i/\tau_i]$ , andernfalls
- Schreibweise:  $K \vdash_M K'$ 
  - Sprechweise:  $M$  erreicht  $K'$  von  $K$  aus in einem Schritt
- Die übrigen Begriffe wie Berechnungen, Akzeptieren, Ablehnen,  $\vdash_M^*$ ,  $L(M)$  sind definiert wie bei (1-String)-Turingmaschinen

# Semantik von Mehrstring-TM: Beispiel

## Beispiel



$$\begin{aligned}
 &(a, (\epsilon, \triangleright, 011110), (\epsilon, \triangleright, \epsilon)) \vdash_M (a, (\triangleright, 0, 11110), (\triangleright, \sqcup, \epsilon)) \vdash_M \\
 &(a, (\triangleright 0, 1, 1110), (\triangleright 0, \sqcup, \epsilon)) \vdash_M (a, (\triangleright 01, 1, 110), (\triangleright 01, \sqcup, \epsilon)) \vdash_M \\
 &(a, (\triangleright 011, 1, 10), (\triangleright 011, \sqcup, \epsilon)) \vdash_M (a, (\triangleright 0111, 1, 0), (\triangleright 0111, \sqcup, \epsilon)) \vdash_M \\
 &(a, (\triangleright 01111, 0, \epsilon), (\triangleright 01111, \sqcup, \epsilon)) \vdash_M (a, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 011110, \sqcup, \epsilon)) \vdash_M \\
 &(b, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 01111, 0, \sqcup)) \vdash_M (c, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 0111, 1, 0\sqcup)) \vdash_M \\
 &(b, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 011, 1, 10\sqcup)) \vdash_M (c, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 01, 1, 110\sqcup)) \vdash_M \\
 &(b, (\triangleright 011110, \sqcup, \epsilon), (\triangleright 0, 1, 1110\sqcup)) \vdash_M (c, (\triangleright 011110, \sqcup, \epsilon), (\triangleright, 0, 11110\sqcup)) \vdash_M \\
 &(b, (\triangleright 011110, \sqcup, \epsilon), (\epsilon, \triangleright, 011110\sqcup)) \vdash_M (d, (\triangleright 01111, 0, \sqcup), (\triangleright, 0, 11110\sqcup)) \vdash_M \\
 &(d, (\triangleright 0111, 1, 0\sqcup), (\triangleright 0, 1, 1110\sqcup)) \vdash_M (d, (\triangleright 011, 1, 10\sqcup), (\triangleright 01, 1, 110\sqcup)) \vdash_M \\
 &(d, (\triangleright 01, 1, 110\sqcup), (\triangleright 011, 1, 10\sqcup)) \vdash_M (d, (\triangleright 0, 1, 1110\sqcup), (\triangleright 0111, 1, 0\sqcup)) \vdash_M \\
 &(d, (\triangleright, 0, 11110\sqcup), (\triangleright 01111, 0, \sqcup)) \vdash_M (d, (\epsilon, \triangleright, 011110\sqcup), (\triangleright 011110, \sqcup, \epsilon)) \vdash_M \\
 &(\text{„}ja\text{“}, (\epsilon, \triangleright, 011110\sqcup), (\triangleright 011110, \sqcup, \epsilon))
 \end{aligned}$$

# Turingmaschinen: Robustheit

## Satz 13.3

- Zu jeder Mehrstring-TM  $M = (Q, \Gamma, \delta, s)$  gibt es eine 1-String-TM  $M'$  mit  $L(M') = L(M)$

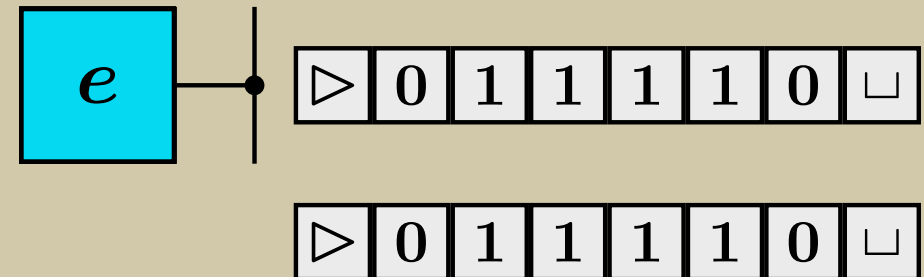
## Beweisidee

- Idee: „Spurentechnik“,  $M'$  verwendet Symbole aus  $(\Gamma \times \{-, +\})^k$ :
  - Die  $i$ -te Komponente  $(\gamma_i, p_i)$  bedeutet:
    - \* Zeichen  $\gamma_i$  im  $i$ -ten String
    - \*  $i$ -ter Kopf an dieser Position, falls  $p_i = +$
- Für einen Schritt von  $M$ 
  - (1) läuft  $M'$  zum rechten Rand und liest alle Kopfsymbole von  $M$ ,
  - (2) läuft  $M'$  zum linken Rand zurück und macht dabei alle nötigen Änderungen,
  - (3) geht  $M'$  in den neuen Zustand über

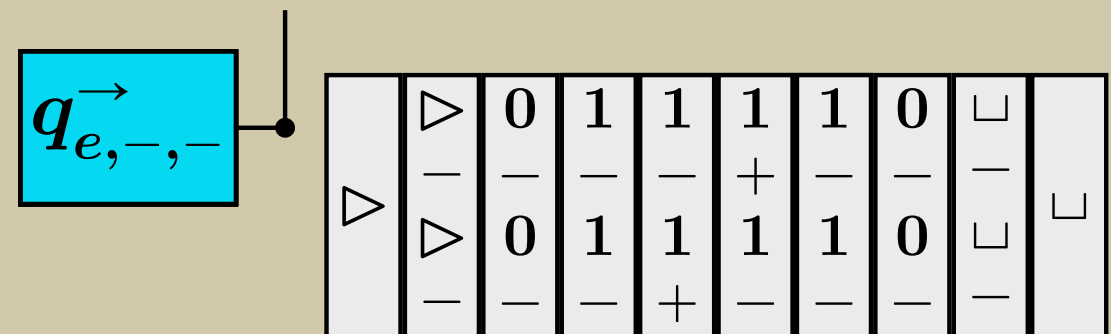
## Beispiel

- Simulation eines Schrittes einer 2-String-TM durch eine 1-String-TM
  - Hier:  $\delta(d, 1, 1) = (e, 1, 1, \leftarrow, \rightarrow)$

- 2-String-TM:



- 1-String-TM:



- Analog kann auch  $f_{M'} = f_M$  erreicht werden

# Inhalt

13.1 WHILE vs. GOTO

13.2 Mehrstring-Turingmaschinen

▷ **13.3 Turingmaschinen und WHILE/GOTO-Programme**

13.4 Die Church-Turing-These

13.5 Entscheidbarkeit und Berechenbarkeit: Definition

# Strings vs. Zahlen

- Turingmaschinen  $M$  berechnen partielle Funktionen  $f_M : \Sigma^* \rightarrow \Sigma^*$   
☞ OBdA:  $\Sigma = \{0, 1\}$
- WHILE-Programme  $P$  berechnen partielle Funktionen  $f_P : \mathbb{N}_0 \rightarrow \mathbb{N}_0$
- Um die beiden Modelle miteinander zu vergleichen, müssen wir Strings und Zahlen ineinander umwandeln können

- Wir verwenden dazu die beiden wie folgt definierten Umwandlungsfunktionen
- Str2N bildet jeden Binärstring auf die durch ihn kodierte Zahl ab, also z.B.:
  - $\text{Str2N}(110) = 6$
  - $\text{Str2N}(00110) = 6$
  - $\text{Str2N}(00000) = 0$
  - $\text{Str2N}(\epsilon) = 0$
- N2Str bildet jede natürliche Zahl auf ihren Binärstring ohne führende Nullen ab, also z.B.:
  - $\text{N2Str}(6) = 110$
  - $\text{N2Str}(0) = \epsilon$

# WHILE-Programme → Turingmaschinen (1/2)

## Satz 13.4

- Jede WHILE-berechenbare Funktion ist Turing-berechenbar
- Genauer: Für jede WHILE-berechenbare Funktion  $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  gibt es eine Mehrstring-Turingmaschine  $M$ , so dass für alle  $n \in \mathbb{N}_0$  gilt:  
$$f(n) = \text{Str2N}(f_M(\text{N2Str}(n)))$$

## Beweisskizze

- Sei  $P$  ein WHILE-Programm für  $f$ 
  - ➔ es gibt ein  $k > 0$ , so dass  $P$  keine anderen Variablen als  $x_1, \dots, x_k$  benutzt
- Idee:  $P$  wird durch eine  $k$ -String Turingmaschine  $M$  simuliert
  - Jeder String von  $M$  repräsentiert dabei den Wert einer Variablen  $x_i$
  - Zu Beginn steht in String 1 der String  $\text{N2Str}(n)$  und auf den anderen Strings der Leerstring (entspricht 0)
  - Am Ende der Simulation steht auf String 1 die Binärkodierung des Ergebnisses
    - ✎ ... und muss dann noch in den  $k$ -ten String kopiert werden
  - Jedes Teilprogramm  $P'$  von  $P$  wird durch eine TM  $M_{P'}$  simuliert
    - \*  $M_{P'}$  ist dabei induktiv definiert




# WHILE-Programme → Turingmaschinen (2/2)

Beweisskizze für Satz 13.4 (Forts.)

$P'$	$M_{P'}$
$x_j := x_i + c$	<ul style="list-style-type: none"> <li>Falls <math>j \neq i</math> <ul style="list-style-type: none"> <li>String <math>j</math> mit <math>\sqcup</math> überschreiben</li> <li>String <math>i</math> nach String <math>j</math> kopieren</li> </ul> </li> <li><math>c</math>-mal 1 zu String <math>j</math> addieren</li> </ul>
$x_j := x_i \div c$  $x_j := c$  $x_j := x_i$	analog
$P_1; P_2$	Führe zuerst $M_{P_1}$ aus, dann $M_{P_2}$
WHILE $x_i \neq 0$ DO $P_1$ END	<p>(a) Wenn <math>i</math>-ter String Leerstring ist: fertig</p> <p>(b) Andernfalls <math>M_{P_1}</math> ausführen, dann weiter mit (a)</p>

## Turingmaschinen → GOTO-Programme (1/6)

- Die Simulation von Turingmaschinen durch GOTO-Programme wirft ein Problem auf
- Turingmaschinen können, abhängig von der Eingabe, beliebig viele Positionen benutzen
- Jedes GOTO- (oder WHILE-) Programm hat aber nur eine feste Zahl von Variablen
  - Wir können also leider *nicht* für jede Position des Turingmaschinen-Strings eine Variable verwenden
    -  mit indirekter Adressierung ginge das...
- Wir werden deshalb String-Zeigerbeschreibungen, durch je drei Zahlen kodieren, damit sie in drei Variablen gespeichert werden können
- Strings über dem Arbeitsalphabet  $\Gamma = \{\sigma_1, \dots, \sigma_\ell\}$  interpretieren wir dazu als Zahlen in  $(\ell+1)$ -adischer Darstellung gemäß  $\sigma_i \mapsto i$ , für jedes  $i$
- 0 wird als Ziffer *nicht* verwendet, um Komplikationen durch führende Nullen zu vermeiden

## Turingmaschinen → GOTO-Programme (2/6)

### Beispiel

- Für  $\Gamma = \{\triangleright, \sqcup, 0, 1\}$  ergibt sich
  - $\triangleright \mapsto 1$
  - $\sqcup \mapsto 2$
  - $0 \mapsto 3$
  - $1 \mapsto 4$
- Z.B.:  $\text{Str2N}_\Gamma(100) =$ 
$$4 \times 5^2 + 3 \times 5 + 3 = 118$$
- $\text{Str2N}_\Gamma(w)$  ist induktiv definiert durch:
  - $\text{Str2N}_\Gamma(\epsilon) \stackrel{\text{def}}{=} 0$  und
  - $\text{Str2N}_\Gamma(u\sigma_i) \stackrel{\text{def}}{=} (\ell + 1) \times \text{Str2N}_\Gamma(u) + i$
- Es gelten:
  - $\text{Str2N}_\Gamma(u\sigma) \div (\ell + 1) = \text{Str2N}_\Gamma(u)$
  - $\text{Str2N}_\Gamma(u\sigma) \bmod (\ell + 1) = \text{Str2N}_\Gamma(\sigma)$

## Turingmaschinen → GOTO-Programme (3/6)


- Für die Simulation von Turingmaschinen durch GOTO-Programme verwenden wir die Notation  $(u, \sigma, v)$  für String-Zeigerbeschreibungen
- Konfigurationen der TM  $M$  werden durch Speicherinhalte der Variablen  $x_1, x_2, x_3, x_4$  repräsentiert:
  - $S_M(q_i, (u, \sigma, v)) \stackrel{\text{def}}{=} i, \text{Str2N}_\Gamma(u), \text{Str2N}_\Gamma(\sigma), \text{Str2N}_\Gamma(v^R), \dots$

 Warum  $v^R$ ?

\* Damit das erste Zeichen von  $v$  durch  $x_4 \bmod (\ell + 1)$  berechnet werden kann

### Beispiel

- Die Startkonfiguration  $(q_1, (\epsilon, \triangleright, 001))$  entspricht also dem Speicherinhalt  $1, 0, 1, 118, \dots$

 Zu beachten:  $\text{Str2N}_\Gamma(001^R) = \text{Str2N}_\Gamma(100) = 118$

- Die Umkehrabbildung  $\text{N2Str}_\Gamma : \mathbb{N} \rightarrow \Gamma^*$  von  $\text{Str2N}_\Gamma$  sei wie folgt definiert:
  - $\text{N2Str}_\Gamma(n) \stackrel{\text{def}}{=} \begin{cases} w & \text{falls } \text{Str2N}_\Gamma(w) = n, \text{ für ein } w \in \Gamma^* \\ \perp & \text{andernfalls} \end{cases}$

# Turingmaschinen → GOTO-Programme (4/6)

## Satz 13.5

- Jede Turing-berechenbare Funktion ist auch GOTO-berechenbar
- Genauer: für jede Turing-berechenbare Funktion  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  gibt es ein GOTO-Programm  $P$ , so dass für alle  $w \in \Sigma^*$  gilt:  

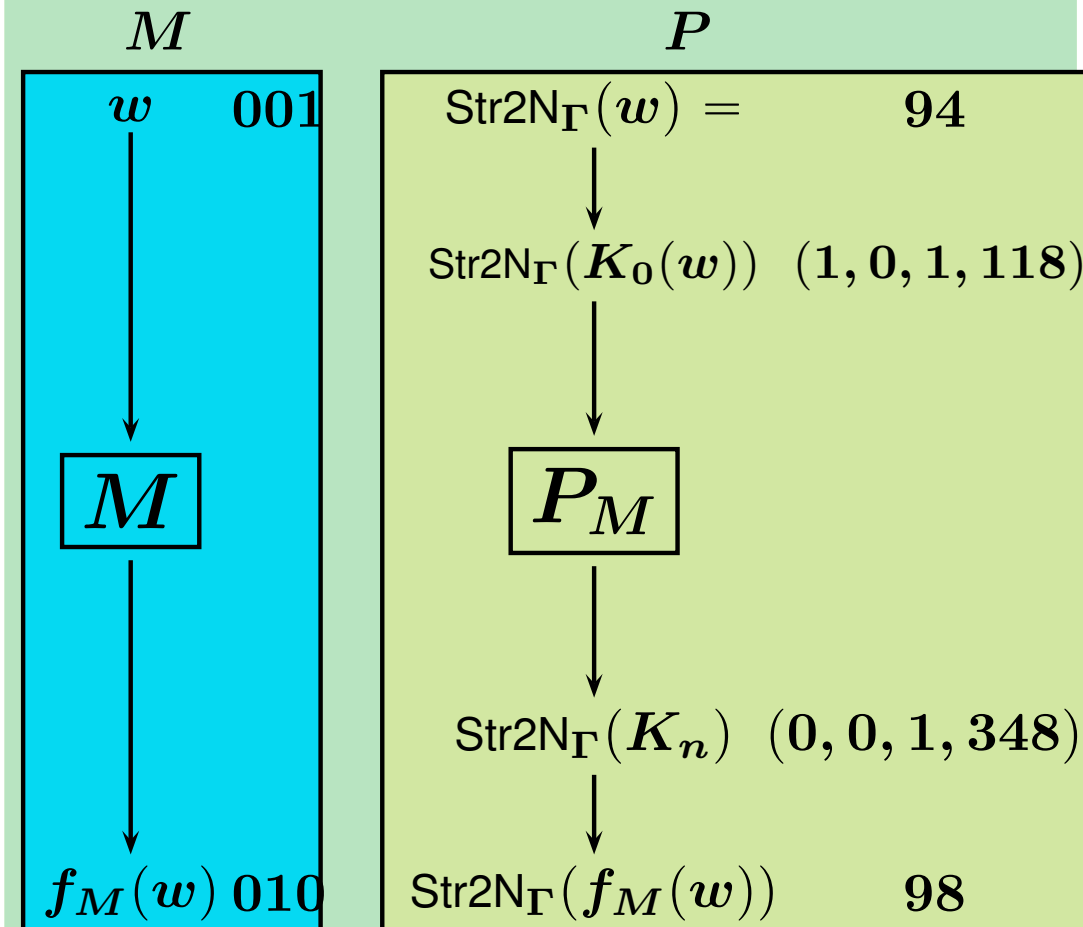
$$f(w) = \text{N2Str}_\Gamma(f_P(\text{Str2N}_\Gamma(w)))$$

## Beweisskizze

- Sei  $M = (Q, \Gamma, \delta, q_1)$  eine TM mit Zuständen  $q_1, \dots, q_k$ , die  $f$  berechnet  
☞ Proviso:  $q_0 = h$
- Wir repräsentieren Konfigurationen wie beschrieben durch die Variablen  $x_1, \dots, x_4$
- $P$  simuliert  $M$  in drei Phasen:
  1. Variablen initialisieren
  2.  $M$  schrittweise simulieren  
(Teilprogramm:  $P_M$ )
  3. Funktionswert aus  $x_2, x_3, x_4$  umkodieren

## Beweisskizze (Forts.)

- Simulation bei Eingabe **001** und Ausgabe **010**



- 13.1

## Turingmaschinen → GOTO-Programme (5/6)

### Beweisskizze (Forts.)

- $P$  simuliert die Berechnung von  $M$  Schritt für Schritt auf die folgende Art

```
 $M_1$ : IF ( $x_1 = 1$ ) AND ( $x_3 = 1$ )  
      THEN GOTO  $M_{11}$   
      IF ( $x_1 = 1$ ) AND ( $x_3 = 2$ )  
      THEN GOTO  $M_{12}$   
      ⋮  
      IF ( $x_1 = k$ ) AND ( $x_3 = \ell$ )  
      THEN GOTO  $M_{k\ell}$   
      IF ( $x_1 = 0$ ) THEN HALT  
 $M_{11}$ :  $P_{1,1}$   
      GOTO  $M_1$   
 $M_{12}$ :  $P_{1,2}$   
      GOTO  $M_1$   
      ⋮  
 $M_{k\ell}$ :  $P_{k,\ell}$   
      GOTO  $M_1$ 
```

- Zur Erinnerung:  $x_1$  speichert die Nummer des Zustandes,  $x_3$  die Kodierung des aktuellen Zeichens

# Turingmaschinen → GOTO-Programme (6/6)

## Beispiel

- Wir betrachten ein Beispiel für die Konstruktion der Teilprogramme  $P_{i,j}$
- Ist  $\delta(q_5, \sigma_6) = (q_8, \sigma_9, \rightarrow)$ , dann ist  $P_{5,6}$  das folgende Teilprogramm

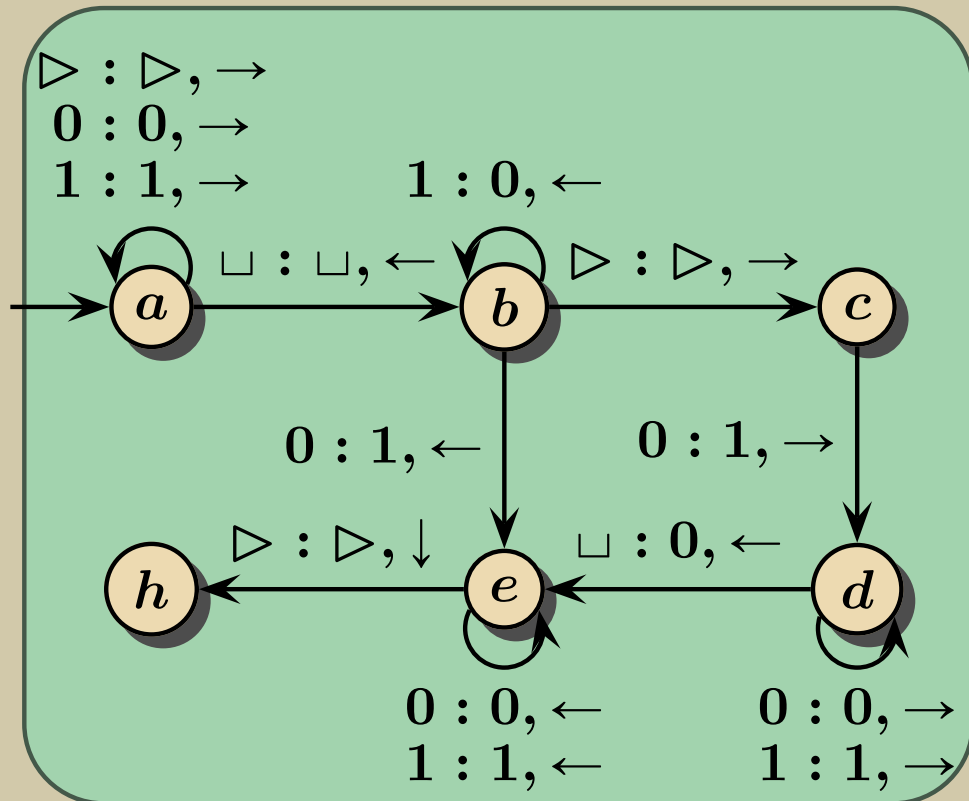
$M_{5,6}: x_1 := 8;$   
IF  $x_3 = 0$  THEN GOTO  $M'_{5,6};$   
 $x_2 := (\ell + 1) \times x_2 + 9;$   
 $M'_{5,6}: x_3 := 2;$   
IF  $x_4 = 0$  THEN GOTO  $M''_{5,6};$   
 $x_3 := x_4 \bmod (\ell + 1);$   
 $M''_{5,6}: x_4 := x_4 \div (\ell + 1);$

## Beispiel: Erläuterungen

- $x_1 := 8$ : Neuer Zustand  $q_8$
- IF  $x_3 = 0$  THEN GOTO  $M'_{5,6}$   
Falls Kopf auf dem linken Rand ist, wird  $x_2$  nicht verändert ( $x_2 = 0$ )
- $x_2 := (\ell + 1) \times x_2 + 9$   
Kodierung des neuen Strings links vom Kopf
- Die drei Zeilen ab  $M'_{5,6}$  bewirken, dass
  - im Falle einer Rechtsbewegung zu einer Position, die kein Eingabesymbol enthält und noch nicht besucht wurde, das aktuelle Zeichen zu einem Leerzeichen wird ( $= 2$ ),
  - andernfalls das neue aktuelle Zeichen das erste Zeichen des bisherigen, durch  $x_4$  kodierten Strings rechts vom Kopf, wird
- $x_4 := x_4 \div (\ell + 1)$   
Der neue String rechts vom Kopf (auch im Falle, dass dieser leer ist, weil gerade erst ein Blank erzeugt wurde)

# Turingmaschinen → GOTO-Programme: Beispiel

Diagramm zur 2. TM



- Wir betrachten die Simulation dieser TM bei Eingabe **001**
- $\sigma_1 = \triangleright, \sigma_2 = \square, \sigma_3 = 0, \sigma_4 = 1$
- $q_0 = h, q_1 = a, q_2 = b, \dots$

Beispiel

- Statt der Eingabe  $w = 001$  erhält  $P$  die Zahl  $\text{Str2N}_\Gamma(w) = \text{Str2N}_\Gamma(001) = 94$
- Daraus berechnet  $P$  die Kodierung der Startkonfiguration  $(a, (\epsilon, \triangleright, 001))$  von  $M$ 
  - Es ergibt sich die Speicherbelegung **1, 0, 1, 118, ...**
- $M$  bewegt nun den Kopf nach rechts und bleibt im Zustand  $a$ 
  - Die neuen Werte für  $x_2, x_3, x_4$  ergeben sich durch:
    - \*  $x_2 := 5 \times x_2 + 1 = 1$ , da  $\triangleright$  „angehängt“ wird
    - \*  $x_3 := x_4 \bmod 5 = 118 \bmod 5 = 3$ 
      - entsprechend dem Zeichen 0
    - \*  $x_4 := x_4 \div 5 = 23$ 
      - entsprechend dem restlichen String  $((01)^R = 10)$
  - Die Konfiguration nach dem ersten Schritt entspricht also der Speicherbelegung **1, 1, 3, 23, ...**



# Inhalt

13.1 WHILE vs. GOTO

13.2 Mehrstring-Turingmaschinen

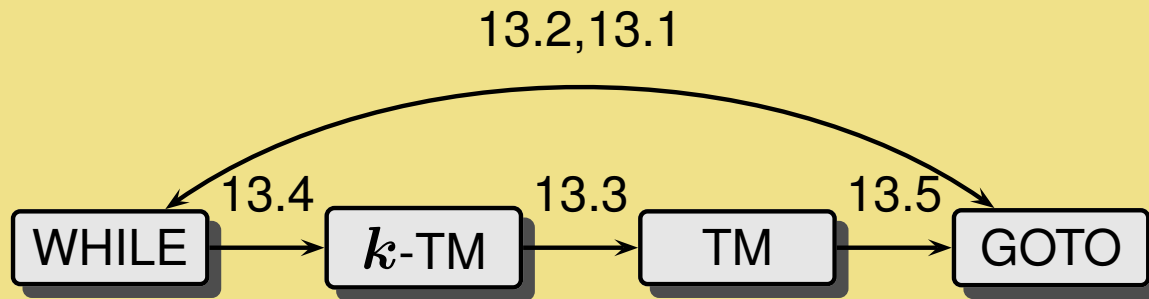
13.3 Turingmaschinen und WHILE/GOTO-Programme

▷ **13.4 Die Church-Turing-These**

13.5 Entscheidbarkeit und Berechenbarkeit: Definition

# Die Church-Turing-These

- Wir haben gesehen, dass alle bisher betrachteten Berechnungsmodelle äquivalent sind:



- Es gibt viel weitere Ansätze zur Formalisierung des Begriffes Algorithmus, die hinsichtlich ihrer Berechnungsstärke äquivalent sind, zum Beispiel:
  - 2-Kellerautomaten
  - Markov-Algorithmen
  - Typ-0-Grammatiken
  - $\lambda$ -Kalkül [Kleene 35, Church 36]
  - Register-Maschinen
- Aus diesem Grunde wird die Klasse der durch Turingmaschinen und die anderen genannten Modelle berechenbaren Funktionen als die „richtige“ Formalisierung des Algorithmus-Begriffs angesehen

- Es gibt wohl keine stärkeren „realistischen“ Berechnungsmodelle

- **Church-Turing-These:**

- Die Klasse der durch Turingmaschinen (WHILE-Programme,...) berechenbaren Funktionen umfasst alle intuitiv berechenbaren Funktionen

- Die Church-Turing-These wurde explizit erstmals von Kleene 1943 formuliert, aber dort schon auf Church und Turing zurückgeführt
- Sie ist nicht beweisbar
- Sie wäre aber im Prinzip widerlegbar: durch den Bau von Computern, die Funktionen berechnen, die nicht Turing-berechenbar sind

# Inhalt

13.1 WHILE vs. GOTO

13.2 Mehrstring-Turingmaschinen

13.3 Turingmaschinen und WHILE/GOTO-Programme

13.4 Die Church-Turing-These

▷ **13.5 Entscheidbarkeit und Berechenbarkeit: Definition**

# Entscheidbar und berechenbar

## Definition (entscheidbar)

- Eine Menge  $L \subseteq \Sigma^*$  heißt entscheidbar, falls es eine TM  $M$  gibt, die  $L$  entscheidet
- Zu beachten:
  - Bei einer entscheidbaren Menge muss die TM für *alle* Eingaben anhalten
- ✎ Statt „nicht entscheidbar“ sagen wir oft auch „unentscheidbar“
- Klar: Wenn  $L$  entscheidbar ist, dann ist auch das Komplement  $\overline{L}$  von  $L$  entscheidbar

## Definition (berechenbar, $\mathcal{R}$ )

- Eine partielle Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  heißt berechenbar, falls es eine TM  $M$  gibt, die
  - für alle  $w \in D(f)$  mit Ausgabe  $f(w)$  anhält und
  - für alle  $w \notin D(f)$  nicht anhält
- $\mathcal{R} \stackrel{\text{def}}{=} \text{Menge der berechenbaren partiellen Funktionen}$
- ✎ Zur Erinnerung: auch totale Funktionen sind partielle Funktionen
  - $f$  kann also auch überall definiert sein...

# Algorithmische Probleme vs. Sprachen und Funktionen (1/4)

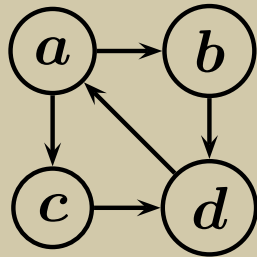
- Unsere bisherigen Berechnungsmodelle beziehen sich nur auf
  - Sprachen und Stringfunktionen bzw.
  - Mengen natürlicher Zahlen und Zahlenfunktionen
- Die soeben definierten Begriffe „entscheidbar“ und „berechenbar“ sind auch für Sprachen und Stringfunktionen definiert
- Wie hängt dies denn mit „richtigen“ algorithmischen Problemen zusammen?

## Algorithmische Probleme vs. Sprachen und Funktionen (2/4)

- Informatikerinnen und Informatikern wissen: alle Arten von Strukturen lassen sich durch 0-1-Strings kodieren
- Graphen können z.B. wie folgt durch Strings kodiert werden

### Beispiel

- Der Graph  $G =$



kann durch die Adjazenzmatrix

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

und diese dann durch den String  
 $\text{enc}(G) = 0110000100011000$  kodiert werden

- Solche Kodierungen ermöglichen uns, die Lücke zu schließen, die besteht zwischen
  - algorithmischen Problemen mit „komplizierteren“ Eingaben wie Graphen, Automaten etc., deren Lösbarkeit wir eigentlich untersuchen wollen, und
  - Sprachen und Funktionen auf Strings, die wir mit Turingmaschinen entscheiden bzw. berechnen können



Die Frage der Eindeutigkeit der Kodierung werden wir hier ignorieren

- Wichtig ist, dass jedem syntaktisch korrekten String eine (bis auf Isomorphie eindeutige) Eingabe zugeordnet werden kann

# Algorithmische Probleme vs. Sprachen und Funktionen (3/4)

- Algorithmische Entscheidungsprobleme entsprechen also Sprachen

## Definition (REACH)

**Gegeben:** Graph  $G$ , Knoten  $s$  und  $t$

**Frage:** Gibt es in  $G$  einen Weg von  $s$  nach  $t$ ?


- Das algorithmische Entscheidungsproblem REACH entspricht der Sprache  $L_{\text{REACH}}$  aller 0-1-Strings, die einen gerichteten Graphen  $G$  und zwei Knoten  $s$  und  $t$  kodieren, in dem es einen Weg von  $s$  nach  $t$  gibt
- Besteht die Eingabe zu einem algorithmischen Problem aus mehreren Komponenten, so trennen wir diese in der Kodierung als Strings durch  $\#$
- $L_{\text{REACH}} = \{ \text{enc}(G) \# \text{enc}(s) \# \text{enc}(t) \mid G \text{ hat Weg von } s \text{ nach } t \}$   
für geeignete Kodierungsfunktionen  $\text{enc}$  für Graphen und Knoten

- Algorithmische Berechnungsprobleme entsprechen also Funktionen auf Strings

## Definition (MINGRAPHCOL)

**Gegeben:** Ungerichteter Graph  $G$

**Gesucht:** Kleinstmögliche Anzahl von Farben, mit denen der Graph zulässig gefärbt werden kann

- Die zugehörige Funktion  $f_{\text{MINGRAPHCOL}}$  ordnet jedem String, der einen ungerichteten Graphen  $G$  kodiert, die (Kodierung der) kleinsten Zahl  $k$ , für die  $G$  eine  $k$ -Färbung hat, zu  
 Graphfärbungen werden wir in Teil D der Vorlesung noch genauer definieren

# Algorithmische Probleme vs. Sprachen und Funktionen (4/4)


- Wir werden die Begriffe „entscheidbar“ und „berechenbar“ auch für die entsprechenden algorithmischen Probleme verwenden
- Also: ist  $A$  ein algorithmisches Entscheidungsproblem und  $L_A$  entscheidbar, so nennen wir auch  $A$  entscheidbar
- Außerdem werden wir uns häufig die Church-Turing-These zunutze machen:
  - Statt eine TM für  $L_A$  zu konstruieren genügt es, einen Algorithmus für  $A$  anzugeben, um zu zeigen, dass  $A$  entscheidbar ist
- Ein Entscheidungsalgorithmus für  $A$  ist also künftig ein Algorithmus, der für jede Eingabe anhält und korrekt angibt, ob sie eine „Ja-Eingabe“ ist



# Entscheidbar und berechenbar: Beispiele

## Beispiel

- REACH ist entscheidbar
  - Der Tiefensuche-Algorithmus terminiert immer und gibt immer die richtige Antwort
- Das Wortproblem für kontextfreie Sprachen ist entscheidbar
  - Gegeben eine Grammatik  $G$  und ein Wort  $w$  kann mit dem CYK-Algorithmus überprüft werden, ob  $w \in L(G)$  ist
  - Der CYK-Algorithmus terminiert bei jeder Eingabe und gibt immer die richtige Antwort

-  Bei den Grammatik-Beispielen nehmen wir der Einfachheit halber an, dass die Grammatiken in CNF sind
- Wenn nicht, können sie in eine CNF-Grammatik umgewandelt werden

## Beispiel

- Die Funktion, die jedem endlichen Automaten  $\mathcal{A}$  die Anzahl der Zustände seines Minimalautomaten zuordnet, ist berechenbar und total
- Die Funktion, die jedem Paar  $(G_1, G_2)$  kontextfreier Grammatiken den lexikographisch kleinsten String  $w \in L(G_1) \cap L(G_2)$  zuordnet, ist berechenbar, aber nicht total
  - Sie ist undefiniert für Paare  $(G_1, G_2)$  mit  $L(G_1) \cap L(G_2) = \emptyset$
- Die Funktion, die jeder TM  $M$  und jeder Eingabe  $x$  den Wert  $f_M(x)$  zuordnet, ist berechenbar, aber nicht total

# Zusammenfassung

- Die verschiedenen Varianten des Turingmaschinen-Modells sind hinsichtlich ihrer Berechnungsstärke äquivalent
- Sie sind hinsichtlich ihrer Berechnungsstärke ebenfalls äquivalent zu WHILE-Programmen und GOTO-Programmen
- **Church-Turing-These:** Turingmaschinen und die dazu äquivalenten Modelle sind die richtige Formalisierung des informellen Begriffes von **Algorithmus**
- Algorithmische Probleme können durch Sprachen und Funktionen auf Strings repräsentiert werden

# Erläuterungen

## Bemerkung 13.1

- Die Zahl 348 kodiert den String der TM am Ende der Berechnung:
  - $N2Str_{\Gamma}(348) = 010\sqcup$ , da
$$Str2N_{\Gamma}((010\sqcup)^R) = Str2N_{\Gamma}(\sqcup 010) = 2 \times 125 + 3 \times 25 + 4 \times 5 + 3 = 348$$
- $P$  berechnet daraus die Zahl 98, die den Ergebnisstring **010** kodiert

# Literatur

- Turingmaschinen:
  - A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936
- $\lambda$ -Kalkül:
  - S. C. Kleene. A theory of positive integers in formal logic. *American Journal of Mathematics*, 57, 1935
  - Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936