

# Theorie der Informatik

## 12. Turing-Berechenbarkeit

Malte Helmert   Gabriele Röger

Universität Basel

9. April 2014

# Überblick: Vorlesung

## Vorlesungsteile

- I. Logik ✓
- II. Automatentheorie und formale Sprachen ✓
- III. Berechenbarkeitstheorie
- IV. Komplexitätstheorie

# Leitfrage

Leitfrage in diesem Vorlesungsteil:

**Was können Computer berechnen?**

# Überblick: Berechenbarkeitstheorie

## III. Berechenbarkeitstheorie

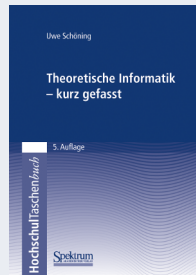
12. **Turing-Berechenbarkeit**
  13. LOOP-, WHILE- und GOTO-Berechenbarkeit
  14. primitive Rekursion und  $\mu$ -Rekursion
  15. Ackermannfunktion
  16. Entscheidbarkeit, Reduktionen, Halteproblem
  17. Postsches Korrespondenzproblem
- ~~Unentscheidbare Grammatik-Probleme~~
- ~~Gödelscher Satz und diophantische Gleichungen~~

# Nachlesen

## Literatur zu diesem Vorlesungskapitel

Theoretische Informatik - kurz gefasst  
von Uwe Schöning (5. Auflage)

- Kapitel 2.1
- Kapitel 2.2



# Berechnungen

# Berechnung

## Was ist eine Berechnung?

- intuitives Berechenbarkeitsmodell (Papier und Bleistift)
- vs. Berechnung auf physikalischen Computern
- vs. formale mathematische Modelle

Wir untersuchen in den folgenden Kapiteln

**Berechnungsmodelle** für **partielle Funktionen**  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ .

- keine echte Einschränkung: beliebige Informationen können als Zahlen kodiert werden

# Formale Berechnungsmodelle

## Formale Berechnungsmodelle

- Turingmaschinen
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen,  $\mu$ -rekursive Funktionen

In den nächsten Vorlesungen werden wir

- diese Berechnungsmodelle **kennen lernen** und
- hinsichtlich ihrer **Mächtigkeit** miteinander **vergleichen**.



# Church-Turing-These

## Church-Turing-These

Alle im **intuitiven Sinne berechenbaren** Funktionen können mit **Turingmaschinen** berechnet werden.

- kann man nicht beweisen (**Warum nicht?**)
- aber wir werden Evidenz dafür sammeln

# Wiederholung: Turingmaschinen

# Formale Berechnungsmodelle

## Formale Berechnungsmodelle: Turingmaschinen

- **Turingmaschinen**
- LOOP-, WHILE-, GOTO-Programme
- primitiv rekursive Funktionen,  $\mu$ -rekursive Funktionen

# Wiederholung: deterministische Turingmaschine (DTM)

## Definition (Deterministische Turingmaschine)

Eine **deterministische Turingmaschine (DTM)** ist gegeben durch ein 7-Tupel  $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \square, E \rangle$  mit:

- $Z$  endliche, nicht-leere Menge von **Zuständen**
- $\Sigma \neq \emptyset$  endliches **Eingabealphabet**
- $\Gamma \supset \Sigma$  endliches **Bandalphabet**
- $\delta : (Z \setminus E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$  **Übergangsfunktion**
- $z_0 \in Z$  **Startzustand**
- $\square \in \Gamma \setminus \Sigma$  **Blank-Zeichen**
- $E \subseteq Z$  **Endzustände**

# Wiederholung: Konfigurationen und Berechnungsschritte

## Wie arbeiten Turingmaschinen?

- **Konfiguration:**  $\alpha z \beta$  mit  $\alpha \in \Gamma^*$ ,  $z \in Z$ ,  $\beta \in \Gamma^+$
- **ein Rechenschritt:**  $c \vdash c'$ , wenn aus Konfiguration  $c$  in einem Berechnungsschritt Konfiguration  $c'$  entsteht
- **mehrere Rechenschritte:**  $c \vdash^* c'$ , wenn aus Konfiguration  $c$  in 0 oder mehr Schritten Konfiguration  $c'$  entsteht ( $c = c_0 \vdash c_1 \vdash c_2 \vdash \dots \vdash c_{n-1} \vdash c_n = c'$ ,  $n \geq 0$ )

(Definition von  $\vdash$ , also wie ein Berechnungsschritt die aktuelle Konfiguration ändert, wird hier nicht wiederholt.  $\rightsquigarrow$  [Kapitel 11.](#))

# Turing-berechenbare Funktionen

# Berechnung von Funktionen?

## Wie kann eine DTM eine Funktion berechnen?

- “Eingabe”  $x$  ist anfänglicher Bandinhalt
- “Ausgabe”  $f(x)$  ist Bandinhalt (ohne Blanks am Rand) beim Erreichen eines Endzustands
- Hält die TM für die gegebene Eingabe nicht an, ist  $f(x)$  an dieser Stelle undefiniert.

## Was für Funktionen werden so berechnet?

- zunächst einmal Funktionen von **Wörtern**:  $f : \Sigma^* \rightarrow \Sigma^*$
- Interpretation als Funktionen von **Zahlen**  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ :  
kodierte Zahlen als Wörter

# Turingmaschinen: berechnete Funktion

## Definition (von einer Turingmaschine berechnete Funktion)

Eine DTM mit Eingabealphabet  $\Sigma$  **berechnet** die (partielle) Funktion  $f : \Sigma^* \rightarrow \Sigma^*$ , für die gilt:

für alle  $x, y \in \Sigma^*$ :  $f(x) = y$  gdw.  $z_0x \vdash^* \square \dots \square z_e y \square \dots \square$

mit  $z_e \in E$ . (Spezialfall:  $z_0 \square$  statt  $z_0x$  wenn  $x = \varepsilon$ )

- Was passiert, wenn Zeichen aus  $\Gamma \setminus \Sigma$  (z. B.  $\square$ ) in  $y$  stehen?
- Was passiert, wenn der Lesekopf am Ende nicht auf dem ersten Zeichen von  $y$  steht?
- Ist  $f$  durch die Definition eindeutig definiert? Warum?



# Turing-berechenbare Funktionen auf Wörtern

Definition (Turing-berechenbar,  $f : \Sigma^* \rightarrow \Sigma^*$ )

Eine (partielle) Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  heisst **Turing-berechenbar**, wenn eine DTM existiert, die  $f$  berechnet.

# Kodierung von Zahlen als Wörter

## Definition (kodierte Funktion)

Sei  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  eine (partielle) Funktion.

Die **kodierte Funktion**  $f^{\text{code}}$  zu  $f$  ist die partielle Funktion  $f^{\text{code}} : \Sigma^* \rightarrow \Sigma^*$  mit  $\Sigma = \{0, 1, \#\}$  und  $f^{\text{code}}(w) = w'$  gdw.

- es gibt  $n_1, \dots, n_k, n' \in \mathbb{N}_0$ , so dass
- $f(n_1, \dots, n_k) = n'$ ,
- $w = \text{bin}(n_1)\#\dots\#\text{bin}(n_k)$  und
- $w' = \text{bin}(n')$ .

Hierbei bezeichnet  $\text{bin} : \mathbb{N}_0 \rightarrow \{0, 1\}^*$  die Binärkodierung (z.B.  $\text{bin}(5) = 101$ ).

**Beispiel:**  $f(5, 2, 3) = 4$  entspricht  $f^{\text{code}}(101\#10\#11) = 100$ .

# Turing-berechenbare numerische Funktionen

Definition (Turing-berechenbar,  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ )

Eine (partielle) Funktionen  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$  heisst **Turing-berechenbar**, wenn eine DTM existiert, die  $f^{\text{code}}$  berechnet.

# Beispiele

# Beispiel: Turing-berechenbare Funktionen (1)

## Beispiel

Sei  $\Sigma = \{a, b, \#\}$ .

Die Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  mit  $f(w) = w\#w$  für alle  $w \in \Sigma^*$  ist Turing-berechenbar.

↪ Tafel

# Beispiel: Turing-berechenbare Funktionen (2)

## Beispiel

Die folgenden numerischen Funktionen sind Turing-berechenbar:

- $\text{succ} : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  mit  $\text{succ}(n) := n + 1$

- $\text{pred}_1 : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  mit  $\text{pred}_1(n) := \begin{cases} n - 1 & \text{falls } n \geq 1 \\ 0 & \text{falls } n = 0 \end{cases}$

- $\text{pred}_2 : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  mit  $\text{pred}_2(n) := \begin{cases} n - 1 & \text{falls } n \geq 1 \\ \text{undefined} & \text{falls } n = 0 \end{cases}$

↪ Tafel/Hausaufgaben

# Beispiel: Turing-berechenbare Funktionen (3)

## Beispiel

Die folgenden numerischen Funktionen sind Turing-berechenbar:

- $add : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  mit  $add(n_1, n_2) := n_1 + n_2$
- $sub : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  mit  $sub(n_1, n_2) := \max\{n_1 - n_2, 0\}$
- $mul : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  mit  $mul(n_1, n_2) := n_1 \cdot n_2$
- $div : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$  mit  $div(n_1, n_2) := \begin{cases} \left\lceil \frac{n_1}{n_2} \right\rceil & \text{falls } n_2 \neq 0 \\ \text{undefined} & \text{falls } n_2 = 0 \end{cases}$

↪ Skizze?

# Zusammenfassung



# Zusammenfassung

- Fragestellung: Was können Computer berechnen?
- Ansatz: untersuche formale Berechnungsmodelle
- zunächst: deterministische Turingmaschinen
- Turing-berechenbare Funktion  $f : \Sigma^* \rightarrow \Sigma^*$ :  
es gibt eine DTM, die auf jede „Eingabe“  $w \in \Sigma^*$   
die „Ausgabe“  $f(w)$  produziert (undefiniert, wenn DTM  
nicht oder in ungültiger Konfiguration anhält)
- Turing-berechenbare Funktion  $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$ :  
genauso; Zahlen binär kodiert und durch # getrennt