

## Das Halteproblem

- ▶ **Eingabe:** Ein *while*-Programm  $P$  und eine Eingabe  $a$  für  $P$
- ▶ **Ausgabe:** 1, falls  $P$  mit der Eingabe  $a$  hält  
0, sonst.

## Unlösbarkeit des speziellen Halteproblems

- Das Problem, ob ein Programm hält, wenn es auf den eigenen Index angewendet wird, ist nicht lösbar, d.h.:

Die Funktion *HALTEPROBLEM* :  $\mathbb{N} \rightarrow \mathbb{N}$ , die definiert ist für alle  $i \in \mathbb{N}$  durch

$$\textit{HALTEPROBLEM}(i) = \begin{cases} 1 & \text{wenn } SEM_i(i) \text{ definiert ist} \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

## Beweisskizze

- ▶ Annahme: *HALTEPROBLEM* ist berechenbar durch das *while*-Programm *HALT1*.
- ▶ Betrachte das Programm *KONFUS*:

```
begin  HALT1;  
  while  $X1 \neq 0$  do  $X1 := X1$ ;  
end
```

- ▶ Sei  $j$  der Index von *KONFUS*. Dann gilt
  - *KONFUS*( $j$ ) hält  $\implies$  *HALT1*( $j$ ) = 0  $\implies$  *KONFUS*( $j$ ) hält nicht.
  - *KONFUS*( $j$ ) hält nicht  $\implies$  *HALT1*( $j$ ) = 1  $\implies$  *KONFUS*( $j$ ) hält.

## Unlösbarkeit des allgemeinen Halteproblems

- Das Problem, ob ein Programm auf eine beliebige Eingabe hält, ist nicht lösbar, d.h.:

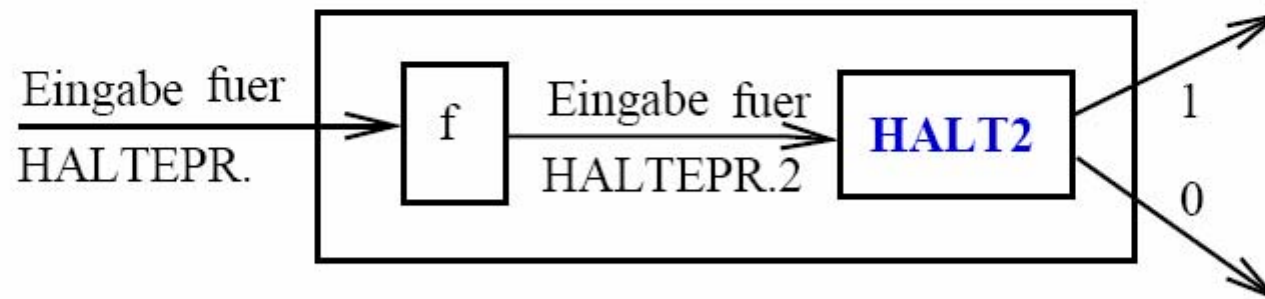
Die Funktion *HALTEPROBLEM2*:  $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  mit

$$\textit{HALTEPROBLEM2}(i, j) = \begin{cases} 1 & \text{wenn } SEM_i(j) \\ & \text{definiert ist} \\ 0 & \text{sonst} \end{cases}$$

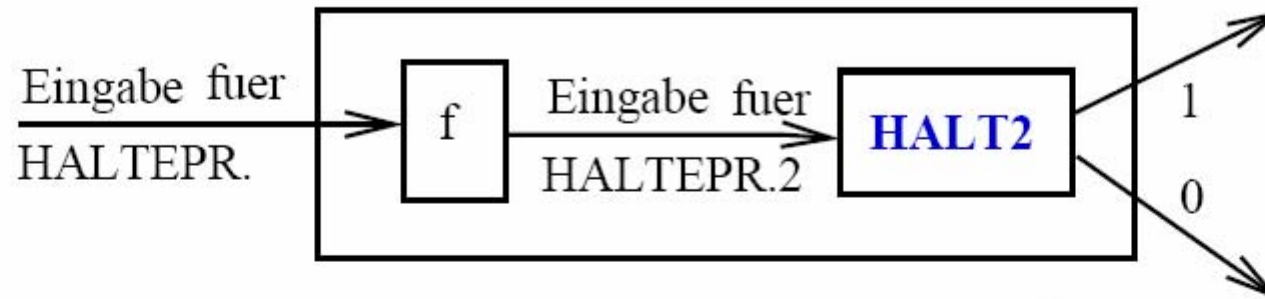
ist nicht berechenbar.

## Beweisskizze

- ▶ Annahme: *HALTEPROBLEM2* wird durch **HALT2** berechnet.
- ▶ Reduktion von *HALTEPROBLEM* auf *HALTEPROBLEM2*:



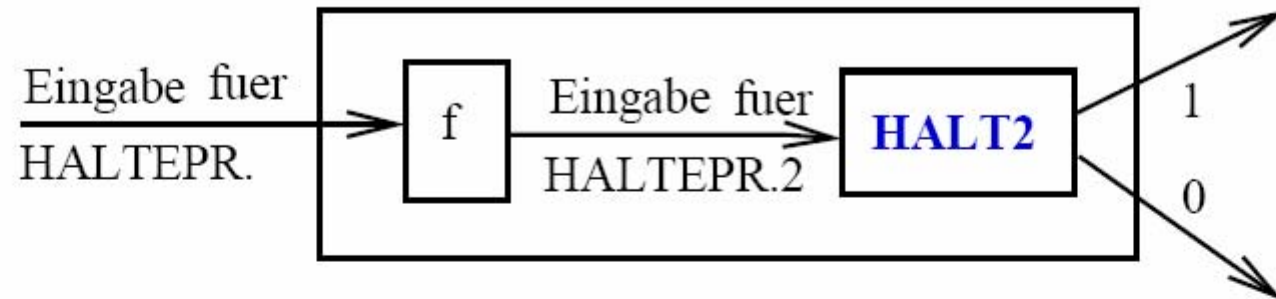
Gesucht ist  $f: \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$  **total** und **berechenbar**, so dass  $\text{HALTEPROBLEM}(i) = \text{HALTEPROBLEM2}(f(i))$ .



- Definiere  $f(i) = (i, i)$ . Dann gilt:  $f$  ist **total** und **berechenbar\*** und

$$\begin{aligned}
 & \text{HALTEPROBLEM}(i) \\
 &= \left\{ \begin{array}{ll} 1 & \text{wenn } SEM_i(i) \text{ definiert ist} \\ 0 & \text{sonst} \end{array} \right\} \\
 &= \text{HALTEPROBLEM2}(f(i))
 \end{aligned}$$

\* Ein bisschen gemogelt, da bei uns berechenbare Funktionen keine Paare ausgeben können; ist aber leicht zu beheben.



► Wir erhalten:

*HALTPROBLEM2* berechenbar  $\implies$   
*HALTPROBLEM* berechenbar. (Widerspruch)

## Das Korrektheitsproblem

- ▶ **Eingabe:** Ein *while*-Programm  $P$  und eine Funktion  $f: \mathbb{N}^j \rightarrow \mathbb{N}$
- ▶ **Ausgabe:** 1, falls  $P$  die Funktion  $f$  berechnet.  
0 sonst.



## Das Äquivalenzproblem

- ▶ **Eingabe:** Zwei *while*-Programme  $P$  und  $P'$
- ▶ **Ausgabe:** 1, falls  $SEM_P = SEM_{P'}$ .  
0, sonst.

## Der Satz von Rice

- ▶ **Eingabe:** Ein *while*-Programm  $P$  und eine (nicht triviale) Eigenschaft  $E$  berechenbarer Funktionen.
- ▶ **Ausgabe:** 1, falls  $P$  Eigenschaft  $E$  erfüllt.  
0, sonst.

Sei  $M$  die Menge aller berechenbaren Funktionen.

Beispiele für nicht triviale Eigenschaften:

- $\{f \in M \mid f(x) = 3 \text{ für alle } x \in \mathbb{N}\}$
- $\{f \in M \mid f \text{ ist total}\}$

Eine Eigenschaft  $E \subseteq M$  ist **nicht trivial**, falls  $E \neq \emptyset$  und  $E \neq M$ .

Existenz eines universellen  
*while*-Programms — ein  
lösbares Problem

## Universelle Funktion

- ▶ **Eingabe:**
  - Index  $i$  eines *while*-Programms
  - $(x_1, \dots, x_j) \in \mathbb{N}^j$
- ▶ **Ausgabe:**  $SEM_i(x_1, \dots, x_j)$ .



## Berechenbarkeit der universellen Funktion

### Theorem

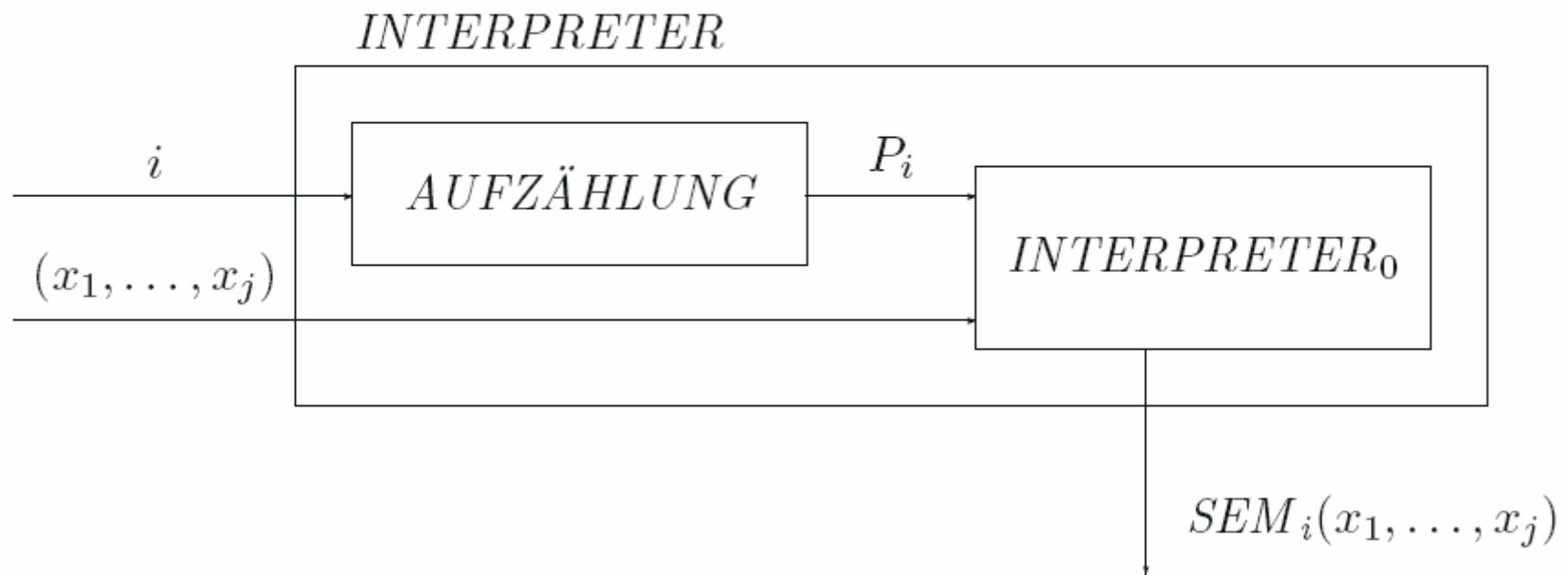
Die partielle Funktion  $interpret: \mathbb{N}^{j+1} \rightarrow \mathbb{N}$ , die für alle  $j \in \mathbb{N}$  definiert ist durch

$$interpret(i, x_1, \dots, x_j) = SEM_i(x_1, \dots, x_j),$$

ist berechenbar.

**Beachte:** Gemäß der Churchschen These ist  $interpret$  berechenbar, falls es einen Algorithmus für  $interpret$  gibt.

# PASCALchen-Interpreter als universelle Funktion



## Algorithmus für $INTERPRETER_0$



1. Wandle das  $k$ -variable Programm  $P$  in sein Flussdiagramm um.
2. Wandle  $(x_1, \dots, x_j)$  in eine Eingabe  $a$  um.
3. Berechne  $P$  für Eingabe  $a$ .
4. Gib bei Termination den Wert von  $X1$  aus.



## Effektivität von *AUFZÄHLUNG*



1.  $B(n)$ : algorithmisch bestimmbar
2.  $decode(B(n))$ : algorithmisch bestimmbar
3. Ist  $decode(B(n))$  ein *while*-Programm?: algorithmisch entscheidbar

## Universelle Funktion

- ▶ **Eingabe:**
  - Index  $i$  eines *while*-Programms
  - $(x_1, \dots, x_j) \in \mathbb{N}^j$
- ▶ **Ausgabe:**  $SEM_i(x_1, \dots, x_j)$ .



# **Grenzlinien zwischen Berechenbarem und Unberechenbarem**

Die Funktion

$$halt_1(x, y) = \begin{cases} y & \text{wenn } SEM_x(x) \text{ definiert ist} \\ \perp & \text{sonst} \end{cases}$$

ist **berechenbar**.

- **Algorithmus:** (1) Berechne  $P_x(x)$ . (2) Gib  $y$  aus.

Die Funktion

$$\text{halt}_2(x, y) = \begin{cases} y & \text{wenn } SEM_x(x) \text{ definiert ist} \\ 0 & \text{sonst} \end{cases}$$

ist **nicht berechenbar**.

- Reduktion von *HALTEPROBLEM* auf  $\text{halt}_2$ :

Definiere  $f(i) = (i, 1)$  (total und berechenbar). Dann gilt  
*HALTEPROBLEM*( $i$ ) =  $\text{halt}_2(f(i))$ .

Die Funktion

$$\text{halt}_3(x, y) = \begin{cases} f(y) & \text{wenn } SEM_x(x) \text{ definiert ist} \\ g(y) & \text{sonst} \end{cases}$$

ist **berechenbar**, falls

$$g(y) \text{ definiert} \implies f(y) \text{ definiert und } f(y) = g(y).$$

Beispiel

$$f(y) = \begin{cases} y^2 & \text{für } y < 200 \\ \perp & \text{sonst} \end{cases}$$

$$g(y) = \begin{cases} y^2 & \text{für } y < 100 \\ \perp & \text{sonst.} \end{cases}$$

Die Funktion

$$\text{halt}_3(x, y) = \begin{cases} f(y) & \text{wenn } SEM_x(x) \text{ definiert ist} \\ g(y) & \text{sonst} \end{cases}$$

ist **berechenbar**, falls

$$g(y) \text{ definiert} \implies f(y) \text{ definiert und } f(y) = g(y).$$

► **Algorithmus** (Skizze):

- (a) Berechne gleichzeitig  $g(y)$  und  $SEM_x(x)$ .
- (b) Terminiert eines von beiden, gib  $f(y)$  aus.

$$\text{halt}_4(x, y, z) = \begin{cases} 1 & \text{wenn } P_x \text{ mit Eingabe } y \text{ nach} \\ & \text{spätestens } z \text{ Schritten hält} \\ 0 & \text{sonst} \end{cases}$$

ist **berechenbar**.

► **Algorithmus** (Skizze):

- (a)  $\text{count} := 0$
- (b) Solange  $\text{count} < z$  und die Berechnung von  $P_x(x)$  noch nicht zu Ende ist:
  - i. berechne nächsten Schritt von  $P_x(x)$
  - ii.  $\text{count} := \text{succ}(\text{count})$
- (c) Falls die Berechnung von  $P_x(x)$  zu Ende ist, gib 1 aus, sonst 0.



## Ein merkwürdiges Beispiel

$$foggy(x) = \begin{cases} 1 & \text{wenn } \pi \text{ eine 5er-Sequenz} \\ & \text{der Länge } x \text{ enthält} \\ 0 & \text{sonst.} \end{cases}$$

1. Fall:  $\pi$  enthält 5er-Sequenzen beliebiger Länge:  
 $foggy(x) = 1$  für alle  $x$ .
2. Fall: Maximale 5er-Sequenz in  $\pi$  hat die Länge  $k$ :

$$foggy(x) = \begin{cases} 1 & \text{wenn } x \leq k \\ 0 & \text{sonst.} \end{cases}$$

$\implies$  *foggy* berechenbar.

**Problem:** Man weiß nicht, welcher Fall zutrifft.

# **Zusammenfassung und Ausblick**

# Zusammenfassung

Dieser Kurs beinhaltete folgende Themen:

1. Automatentheorie
2. Formale Sprachen
3. Berechenbarkeit

## Endliche Automaten

- ▶ erkennen genau die regulären Sprachen
- ▶ Äquivalenz von nichtdeterministischen und deterministischen endlichen Automaten (Potenzautomat)
- ▶ korrekte Übersetzung in rechtslineare Grammatiken
- ▶ schnelle Worterkennung,
- ▶ werden in vielen Bereichen praktisch eingesetzt (Model-Checking, UML, XML-Parser, Kontrollbedingungen usw.),
- ▶ können keine gängigen Programmiersprachen erkennen

## Kellerautomaten

- ▶ erkennen genau die kontextfreien Sprachen
- ▶ deterministische Kellerautomaten können weniger Sprachen erkennen als nichtdeterministische
- ▶ können Programmiersprachen erkennen
- ▶ deterministische Kellerautomaten werden im Compilerbau eingesetzt
- ▶ können nicht alle algorithmisch beschreibbaren Sprachen erkennen

## Reguläre Ausdrücke

- ▶ Beschreibungsmittel für die regulären Sprachen
- ▶ finden Anwendung im Compilerbau, Softwaretechnik, Webbrowsern usw.

# Grammatiken

## ► kontextfreie Grammatiken

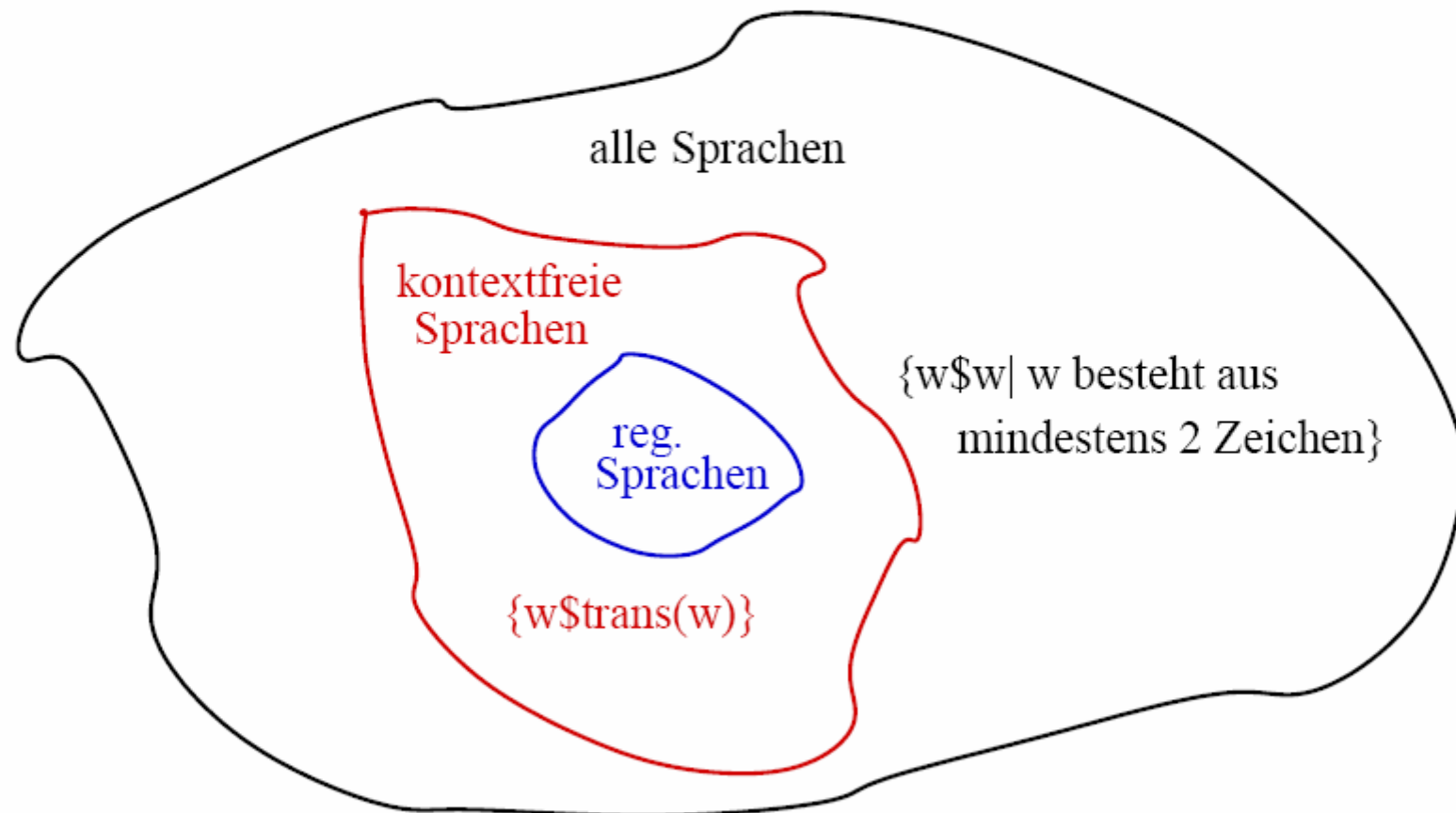
- erzeugen genau die kontextfreien Sprachen
- lange Ableitungen lassen sich in viele kleine zerlegen (Kontextfreiheitslemma)
- korrekte Übersetzung in Kellerautomaten

## ► Rechtslineare Grammatiken

- sind eine Teilmenge der kontextfreien Grammatiken
- erzeugen genau die regulären Sprachen
- erzeugen Wörter von links nach rechts



# Sprachklassen



## Reguläre Sprachen

- ▶ werden von **endlichen Automaten** erkannt
- ▶ werden von **regulären Ausdrücken** beschrieben
- ▶ werden von **rechtslinearen Grammatiken** erzeugt
- ▶ genügend lange Wörter sind pumpbar (Pumping-Lemma)
- ▶ Wortproblem entscheidbar
- ▶ Leerheitsproblem entscheidbar

## Kontextfreie Sprachen

- ▶ werden von Kellerautomaten erkannt
- ▶ werden von kontextfreien Grammatiken erzeugt
- ▶ genügend lange Wörter sind pumpbar (Pumping-Lemma)

## Abschlusseigenschaften regulärer und kontextfreier Sprachen

	Regulär	Kontextfrei
Vereinigung	+	+
Konkatenation	+	+
Kleene Hülle	+	+
Durchschnitt	+	-
Komplement	+	-
Differenz	+	-

+: In diesem Kurs behandelt.

## Berechenbarkeit

- ▶ *while*-Programme als Berechenbarkeitsmodell
- ▶ Churchsche These
- ▶ Effektive Aufzählbarkeit der *while*-Programme
- ▶ Unlösbarkeit des Halteproblems
- ▶ Existenz eines universellen *while*-Programms

## Ausblick

Theoretische Informatik 2 umfasst folgende Themen:

- ▶ **Formale Sprachen** (Fortführung)
- ▶ **Berechenbarkeit** (Fortführung)
- ▶ **Komplexitätstheorie**

## Formale Sprachen

- ▶ Welche **allgemeineren Grammatiktypen** gibt es, und wie hängen diese mit den bisher betrachteten zusammen?
- ▶ Für welche Sprachklassen ist das **Wortproblem** (noch) entscheidbar?
- ▶ Algorithmen für die Lösung des Wortproblems

## Berechenbarkeit

- ▶ Weitere **Berechenbarkeitsmodelle** (Turingmaschinen, rekursive Funktionen)
- ▶ Zusammenhang zu *while*-Programmen



## Komplexitätstheorie

- ▶ Wie lange dauert es mindestens und höchstens, bis ein Computer ein Problem berechnet hat?
- ▶ Welche **Berechnungszeiten** sind (noch) akzeptabel?
- ▶ Wie komplex sind **konkrete Algorithmen**, wie z.B. Sortieren oder Matrizenmultiplikation?