

# Theorem von Rice

Definition, Beweis, Beispiele

# Kurze Wiederholung

- Codierung
  - Programm P, Eingabe x, Ausgabe etc.
  - Wir bleiben in der Java-Welt
- Halteproblem
  - spezifisch für Java
- Berechenbarkeit
  - Es kann für eine Funktion ein Algorithmus formuliert werden
  - Eingabe -> Algorithmus -> Ausgabe
  - Auch hier: Achtung, passende Codierung!
- Entscheidbarkeit
  - Charakteristische Funktion ist berechenbar
  - $\sim$ Mengenzugehörigkeit

# Agenda

- Informelle Aussage
- Einführende Beispiele
- Formale Aussage & ihr Beweis
- Nachweise für ausgewählte Beispiele
- Zusammenhang zur Zeitkomplexität
- Zusammenfassung

# Informelle Aussage

# (Informelle) Aussage

- Alle nicht-trivialen semantischen Eigenschaften von Programmen sind nicht entscheidbar
- nicht-trivial?
  - weder wahr für alle Programme, noch für keines
- semantische Eigenschaften von Programmen?
  - Eigenschaft der durch das Programm P berechneten Funktion f
    - Mathematik: Eigenschaften von Funktionen, Funktionsklassen, etc.
    - Informatik: f berechnet Sortierfunktion, Suchfunktion ...
  - Nicht dazu zählt bspw.:
    - Zeitkomplexität  $\mathcal{O}(n)$

# Einführende Beispiele

# Einfache Beispiele

## Mathematik:

- $\{f \mid f \text{ ist [in,sur,bi]jektiv}\}$
- $\{f \mid f \text{ ist konstante Funktion}\}$
- $\{f \mid f(x) \text{ ist durch 3 teilbar}\}$

## Informatik:

- $\{f \mid f \text{ ist definiert bei } x\}$
- $\{f \mid f \text{ ist überall definiert}\}$
- $\{f \mid f \text{ sortiert Liste } L\}$

# Veranschaulichung

**class** Example

```
{  
    public int square(int x)  
    {  
        return x^2;  
    }  
}
```

**class** AbsurdExample

```
{  
    public int square(int x)  
    {  
        return x*x;  
    }  
}
```



# Veranschaulichung

**class** Example

```
{  
    public int square(int x)  
    {  
        return x^2;  
    }  
}
```

**class** AnotherAbsurdExample

```
{  
    public int square(int x)  
    {  
        int i = 0;  
        //or 1,2,..n  
        return x^2;  
    }  
}
```

# Collatz-Programm

- Programm, das (vermutlich) für beliebige  $n > 1$  mit 1 endet
- Ungelöstes math. Problem
- Für positive Zahlen bis  $20 * 2^{58}$  durch Ausprobieren bestätigt
- Für  $n = 3$  bspw.:
  - 10, 5, 16, 8, 4, 2, 1

```
while (n > 1) {  
    if (n%2 == 0) n = n/2;  
    else n = 3*n + 1;  
}
```

- Frage: Berechnet das Collatz-Programm die Konstante 1?
- Ggf. durch Überlauf zusätzlich erschwert in manchen Programmiersprachen

# Formale Aussage & ihr Beweis

# Satz von Rice (1953)

Sei  $R = \{f \mid \text{Funktion } f \text{ ist berechenbar}\}$  und  
sei  $\emptyset \neq S \subsetneq R$  ( $S$  ist nicht-triviale Teilmenge von  $R$ ).

Dann ist die Sprache

$$L_J(S) = \{ p \mid \text{Programm } p \text{ berechnet eine Funktion aus } S \}$$

nicht rekursiv (nicht entscheidbar).

„Lax formuliert“

# Satz von Rice (1953)

Sei  $R = \{f \mid \text{Funktion } f \text{ ist berechenbar}\}$  und  
sei  $\emptyset \neq S \subsetneq R$  ( $S$  ist nicht-triviale Teilmenge von  $R$ ).

Dann ist die Sprache

$$L_J(S) = \{ p \mid \text{Java-Programm } p \text{ berechnet eine} \\ \text{Funktion aus } S \}$$

nicht rekursiv (nicht entscheidbar).

# Definitionen

- Wir betrachten das allgemeine Halteproblem für Java-Programme  $\mathcal{H}_J$
- Java-Programm  $p$  und Eingabe  $x$  “passend” zu  $p$
- Simulator  $\text{Sim}(p, x)$ , der Java-Programm  $p$  auf der Eingabe  $x$  simuliert
  - offensichtlich berechenbar
- Überall undefinierte Funktion  $u$ 
  - offensichtlich berechenbar
- Ein Java-Programm  $J_f$ , welches eine Funktion  $f$  berechnet

# Konstruktion

```
public void Sim(Object p, Object x) {...}  
public void Jf(Object y) {...}
```

```
public void Rice(Object y)  
{  
    Sim(p, x);  
    Jf(y);  
}
```

- Nebenstehendes Programm  $Rice_{(p, x)}$  kann für jedes Paar  $(p, x)$  berechnet werden
- Im Folgenden werden wir durch Reduktion auf das Halteproblem das Theorem von Rice beweisen

# 1. Fall - $u \in R - S$

- wähle  $f \in S$
- Falls  $(p, x) \in \mathcal{H}_J \Rightarrow \text{Sim}(p, x)$  hält  $\Rightarrow \text{Rice}_{(p, x)} \equiv f \in S$
- Falls  $(p, x) \notin \mathcal{H}_J \Rightarrow \text{Sim}(p, x)$  hält nicht  $\Rightarrow \text{Rice}_{(p, x)} \equiv u \in R - S$
- Also:  $\text{Rice}_{(p, x)}$  hält in beiden Fällen und akzeptiert  $\mathcal{H}_J$ 
  - $\mathcal{H}_J \leq_m L_J(S)$
  - $\chi_{\mathcal{H}_J}(p, x) = \chi_{L_J(S)}(\text{Rice}_{(p, x)})$ 
    - $\text{Rice}_{(p, x)}$  berechenbar; wäre  $\chi_{L_J(S)}(\text{Rice}_{(p, x)})$  berechenbar, so auch  $\chi_{\mathcal{H}_J}(p, x)$  und damit  $\mathcal{H}_J$  entscheidbar



## 2. Fall - $u \in S$

- wähle  $f \in R - S$
- Falls  $(p, x) \in \mathcal{H}_J \Rightarrow \text{Sim}(p, x)$  hält  $\Rightarrow \text{Rice}_{(p, x)} \equiv f \in R - S$
- Falls  $(p, x) \notin \mathcal{H}_J \Rightarrow \text{Sim}(p, x)$  hält nicht  $\Rightarrow \text{Rice}_{(p, x)} \equiv u \in S$
- Also:  $\text{Rice}_{(p, x)}$  hält in beiden Fällen und akzeptiert  $\neg \mathcal{H}_J$ 
  - $\neg \mathcal{H}_J \leq_m L_J(S)$
  - $\chi_{\neg \mathcal{H}_J}(p, x) = \chi_{L_J(S)}(\text{Rice}_{(p, x)})$ 
    - $\text{Rice}_{(p, x)}$  berechenbar; wäre  $\chi_{L_J(S)}(\text{Rice}_{(p, x)})$  berechenbar, so auch  $\chi_{\neg \mathcal{H}_J}(p, x)$  und damit  $\neg \mathcal{H}_J$  entscheidbar

# Nachweise für ausgewählte Beispiele

# Totalitätsproblem

- $\forall x$ : Java-Programm  $p$  hält für Eingabe  $x$
- $S_{\text{tot}} = \{ f \mid f \text{ ist rekursiv und total} \}$
- $L(S_{\text{tot}}) = \{ p \mid \text{Java-Programm } p \text{ berechnet Funktion aus } S_{\text{tot}} \}$
- Direktes Korollar aus dem Satz von Rice:
  - $S$  ist nicht-triviale Teilmenge von  $R \Rightarrow L(S_{\text{tot}})$  nicht entscheidbar

# Spezielles Äquivalenzproblem

- Für fest vorgegebenes Programm  $p'$ :
  - $\text{Equiv}(p) = 1 \iff \text{Java-Programm } p \text{ berechnet die gleiche Funktion } f \text{ wie Programm } p'$
- $S = \{f\}$
- $f$  = Funktion, die durch eine Spezifikation gegeben ist  
~ z.B. Programmverhalten durch Regeln und Fakten (Prolog)
- Java-Programm  $p$  ist Implementierung dieser Funktion in Java

# Zusammenhang zur Zeitkomplexität

# Entscheidbarkeit von $\mathcal{O}(n^k)$

- Beispiele

- $P_J^1 = \{p \mid \text{Java-Programm } p \text{ hat Zeitkomplexität } \subseteq \mathcal{O}(n)\}$
- $P_J^2 = \{p \mid \text{Java-Programm } p \text{ hat Zeitkomplexität } \subseteq \mathcal{O}(n^2)\}$

...

- $P_J^k = \{p \mid \text{Java-Programm } p \text{ hat Zeitkomplexität } \subseteq \mathcal{O}(n^k)\}$

# Entscheidbarkeit von $\mathcal{O}(n^k)$ - Konstruktion

```
public boolean Sim(Object p, Object
x, Object n) {...}

public int one() { return 1; }

public void compl(String y) {
    int n = y.length();
    if (Sim(p, x, n)) {
        while (true) {}
    }
    return 1
}
```

- Nebenstehendes Programm  $\text{compl}_{(p, x)}$  kann für jedes Paar  $(p, x)$  berechnet werden
- Dieses Beispiel betrachtet zunächst  $k = 1$

# Entscheidbarkeit von $\mathcal{O}(n^k)$ - Beweis

- Falls  $(p, x) \notin \mathcal{H}_J$ 
  - $\Rightarrow \text{Sim}(p, x, n)$  immer false  $\Rightarrow \text{compl}_{(p, x)}$  für kein  $n$  in Endlosschleife
  - $\Rightarrow$  Zeitaufwand  $\mathcal{O}(n)$  für  $\text{sim}(p, x, n) \Rightarrow$  für beliebiges  $k$  aus  $\mathcal{O}(n^k)$
- Falls  $(p, x) \in \mathcal{H}_J$ 
  - $\Rightarrow \text{Sim}(p, x, n)$  true für ein  $n_1 \Rightarrow \text{compl}_{(p, x)}$  für ein  $n_1$  in Endlosschleife
  - $\Rightarrow$  Programm terminiert nicht für Eingabelängen  $> n_1 \Rightarrow$  für kein  $k$  aus  $\mathcal{O}(n^k)$
- Also:  $(p, x) \notin \mathcal{H}_J \Leftrightarrow \text{compl}_{(p, x)}$  aus  $\mathcal{O}(n^k)$ 
  - $\neg \mathcal{H}_J <_m L(S)$
  - $\chi_{\neg \mathcal{H}_J}(p, x) = \chi_{L(S)}(\text{compl}_{(p, x)})$ 
    - $\text{compl}_{(p, x)}$  berechenbar;
    - wäre  $\chi_{L(S)}(\text{compl}_{(p, x)})$  berechenbar;
  - dann damit  $\neg \mathcal{H}_J$  entscheidbar  $\nleftrightarrow$



# Kann Programm P (effizienter) in $\mathcal{O}(n^k)$ implementiert werden?

- Beispiele
  - $P_J^1 = \{f \mid \exists p: \text{Java-Programm } p \text{ berechnet } f \text{ in Zeitkomplexität } \subseteq \mathcal{O}(n)\}$
  - $P_J^2 = \{f \mid \exists p: \text{Java-Programm } p \text{ berechnet } f \text{ in Zeitkomplexität } \subseteq \mathcal{O}(n^2)\}$
  - ...
  - $P_J^k = \{f \mid \exists p: \text{Java-Programm } p \text{ berechnet } f \text{ in Zeitkomplexität } \subseteq \mathcal{O}(n^k)\}$
- Alle diese Mengen sind nichttrivial

# Kann Programm P (effizienter) in $\mathcal{O}(n^k)$ implementiert werden?

## Beweis:

- $P_J^k \neq \emptyset$ 
  - $f(x) = 1$
  - `public int one() {return 1;}`
- $P_J^k \neq R$ 
  - Traveling Salesman Problem
  - !? nur mittels z.B. „generate & test“ sicher in  $\mathcal{O}(n!)$

# Einschub: P-NP-Problem

- Ist TSP in polynomialer Zeit lösbar, oder nicht?
- Noch nicht gezeigt, daher anderes Vorgehen
- Programm  $p \in \mathcal{O}(n^k) \Leftrightarrow \exists c: \text{ Laufzeit } T_p(n) \leq c * n^k$

# Kann Programm P (effizienter) in $\mathcal{O}(n^k)$ implementiert werden?

- Idee: Alle Programme mit Zeitkomplexität  $\subseteq \mathcal{O}(n^k)$  können aufgezählt werden
  - Tupel  $(c, i)$  mit Konstante  $c$  für Schranke und
  - $i$  als (Java-) Programm-Nummer
- Diagonalisierungsfunktion muss sich von allen diesen Programmen unterscheiden
  - $\Rightarrow$  kann nicht in  $\mathcal{O}(n^k)$  liegen
  - $\Rightarrow$  Interessanter Nebeneffekt:
    - Diagonalisierungsfkt. für  $\mathcal{O}(n^k)$  in  $\mathcal{O}(n^{k+1})$  berechenbar;
    - Hierarchiebildung
- Also ist  $P_J^k$  eine nicht-triviale Teilmenge von  $R$ 
  - $\Rightarrow$  Es ist für kein  $k$  entscheidbar, ob ein Java-Programm  $i$  in  $P_J^k$  liegt

# Zusammenfassung

# Was haben wir gelernt

- Ob ein Programm eine Funktion, eine Eigenschaft etc. berechnet, können wir nicht entscheiden
- Darunter fallen ALLE berechenbaren Funktionen
- Zusammenhang zur Zeitkomplexität