



Halteproblem und Reduzierbarkeit von Problemen

Agenda

- Einführendes Beispiel
- Definitionen
- Beweis einer nichtberechenbaren Funktion
- Reduzierbarkeit von Problemen
- Rekursivität von Sprachen
- Zusammenfassung/ Ausblick

Funktionen ...

➤ Mathematik

- $f: \mathbb{N} \rightarrow \mathbb{N}$ / $g: \mathbb{R} \rightarrow \mathbb{R}$
- $f: \mathbb{N}^2 \rightarrow \mathbb{N}$ / $g: \mathbb{R}^2 \rightarrow \mathbb{R}$
- $f: \mathbb{N}^n \rightarrow \mathbb{N}^m$ / $g: \mathbb{R}^n \rightarrow \mathbb{R}^m$
- BEISPIELE ??

➤ Informatik

- $\text{Sort}(x)$, $\text{Sum}(x)$, etc.



... und ihre Berechenbarkeit

- Alle Funktionen werden auf
 - $f: \{0,1\}^* \rightarrow \{0,1\}^*$ oder
 - $g: \mathbb{N} \rightarrow \mathbb{N}$
- abgebildet.
- **Achtung:** \mathbb{R} kann nicht berechnet werden!

Was sind Probleme und welche sind in P??

■ Zusammenhang Mengen und Prädikate

■ Entscheidungs-**Problem**: $x \in A$

■ Charakteristische Funktion zu A:

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{sonst} \end{cases}$$

■ A entscheidbar genau dann, wenn χ_A berechenbar

■ A in **P** genau dann, wenn χ_A in **polynomialer Zeit** berechenbar

■ Beispiele

- A_1 = Menge der Primzahlen $= \{ x \in \mathbb{N} \mid x \text{ ist prim} \}$
- A_2 = Menge der Quadratzahlen $= \{ x \in \mathbb{N} \mid \text{es gibt } n \in \mathbb{N} : x = n^2 \}$
- A_3 = Graph der Quadratfunktion $= \{ (x, y) \mid y = x^2 \}$
- A_4 = Graph sortierte Listen $= \{ (L, L') \mid L' = L \text{ sortiert} \}$

Halteproblem

- einführendes Beispiel

```
■ void Test() {  
    while (true) {  
        int i = 1;  
    }  
}
```

→ terminiert nicht

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

→ terminiert wahrscheinlich immer
(geprüft bis $\sim 20 \cdot 2^{55}$)

→ Ein ungelöstes mathematisches
Problem

- Kann man formal überprüfen, ob ein Programm terminiert oder nicht?
- Welche Folgen hat dies auf unsere Programme?

Definitionen (I)

Kodierung der Programme bzw. Eingaben:

- Kodierung hier als Dualzahl

Sprachabhängigkeit

- $H_{\text{java}} = \{(i, x) \mid \text{Java-Programm } P_i \text{ hält bei Eingabe } E_x\}$
- $H_C = \{(i, x) \mid \text{C-Programm } P_i \text{ hält bei Eingabe } E_x\}$
- ABER: Anwendbarkeit des Beweises eines Problems auf andere Probleme

Mengendefinition

- Für jedes Problem muss die Menge definiert werden, auf der sie angewandt werden soll

Definitionen (II)

- berechenbare Funktion $f: M \rightarrow N$

Es existiert ein Algorithmus in irgendeiner Programmiersprache (Turingmaschine, Java, C, C++, Haskell, Prolog), der $f(x)$ für jeden Eingabewert $x \in M$ berechnet, andernfalls terminiert der Algorithmus nicht.

- Entscheidbare Menge

$$\chi_M(x) = \begin{array}{ll} 1 & \text{falls } x \in M \\ 0 & \text{sonst} \end{array}$$

charakteristische Funktion χ_M ist berechenbar

→ rekursive Menge

- Rekursiv aufzählbare Menge

$$\chi_M(x) = \begin{array}{ll} 1 & \text{falls } x \in M \\ \text{undefiniert} & \text{sonst} \end{array}$$

charakteristische Funktion χ_M ist im positiven Fall berechenbar

→ die Menge heißt auch semi-entscheidbar

Definitionen (III)

- Halteproblem:
Algorithmus, der für ein beliebiges Programm mit beliebiger Eingabe erkennt, ob dieses terminiert oder nicht
→ Programmiersprache sowie Kodierung des Programmes und der Eingabe müssen eindeutig definiert sein
- Totalitätsproblem:
Ein Programm heißt total, falls das Programm für alle Eingaben terminiert.
Kann dies durch ein Programm bestimmbar sein?
- Äquivalenzproblem:
Ist es durch ein Programm berechenbar, ob zwei Programme P1 und P2 verhaltensgleich sind?

2.16 Grenzen endlicher Akzeptoren

Es gibt eine formale Sprache, die von *keinem* endlichen Akzeptor erkannt werden kann. Selbst dann, wenn man nur $X = \{0\}$ erlaubt. Verschiedene Beweise; z. B.:

1. Es gibt abzählbar unendlich viele („ \mathbb{N} viele“) endliche Akzeptoren. **TURING MASCHINEN**
2. Es gibt überabzählbar unendlich viele („ \mathbb{R} viele“) formale Sprachen über $\{0\}$. **TEILMENGEN von \mathbb{N} \Rightarrow Entscheidungsprobleme**
3. Es gibt keine surjektive Abbildung von \mathbb{N} auf \mathbb{R} (Cantor).

Man lernt aber auch etwas an konkreten Beispielen ...

Nichtberechenbare Funktion

Einführung

- Unentscheidbares Problem = Problem mit einer nicht-berechenbaren charakteristischen Funktion
- Bekanntestes unentscheidbares Problem: Halteproblem
- Beweis erfolgt in einem Widerspruchsbeweis

Nichtberechenbare Funktion

Diagonalisierung

- Aufstellen einer unendlichen Matrix
 - X-Achse: alle möglichen Eingaben E
 - Y-Achse: alle möglichen Programme P
 - Die Elemente auf den Achsen sind sortiert nach ihrer Kodierung

- Elemente der Matrix:

$f(i, x) = P_i(x)$ wenn $\text{Sim}(i, x)$ terminiert
 undefiniert sonst

$P \backslash E$	1	2	3	4
1	3	0	5	2
2	1	0	u	1
3	u	0	2	2
4	u	3	0	0

- Idee: Nehme eine Funktion, die sich von jeder berechenbaren Funktion unterscheidet
- Definiere $\text{Diag}(i) = \begin{matrix} P_i(i) + 1 & \text{wenn } \text{Sim}(i,i) \text{ terminiert} \\ 0 & \text{sonst} \end{matrix}$

Nichtberechenbare Funktion Beweis (I)

- Betrachten einer beliebigen Programmiersprache, hier Java
- Halteproblem H_{Java} mit charakteristischer Funktion
$$\chi_H(i, x) = \begin{cases} 1 & \text{wenn Java-Programm } P_i \text{ bei Eingabe } E_x \text{ terminiert} \\ 0 & \text{sonst} \end{cases}$$
- Annahme Halteproblem ist entscheidbar
 - es gibt ein Java-Programm JavaStop mit der folgenden charakteristischen Funktion:
$$\text{JavaStop}(i, x) = \begin{cases} 1 & \text{wenn JavaSim}(i, x) \text{ terminiert} \\ 0 & \text{sonst} \end{cases}$$

Nichtberechenbare Funktion Beweis (II)

- Dann wäre $\text{Diag}(i)$ durch folgendes Java Programm berechenbar


```

int JavaStop(int P, int x) { ... }
int JavaSim(int P, int x) { ... }
int Diag(int i) {
  if (JavaStop(i, i) == 1)
    {return JavaSim(i, i) + 1;}
  else { return 0; }
}
      
```
- Sei \hat{i} Programm-Nr von Diag
 - $\text{Diag}(\hat{i}) = \text{JavaSim}(\hat{i}, \hat{i}) + 1$ da Diag total
 - $\forall i : \text{Diag}(i) = \text{JavaSim}(\hat{i}, i)$
 - $\text{Diag}(\hat{i}) = \text{Diag}(\hat{i}) + 1$

→ WIDERSPRUCH

→ WIDERSPRUCH

Nichtberechenbare Funktion Folgerungen

- Praktische Konsequenz:
 - Es wird kein Programm geben, welches prüft, ob ein gegebenes Programm auf einer Eingabe terminiert.
 - Es wird niemals ein Programm geben, welches die semantische Korrektheit von Programmen prüft
- partielle Korrektheit von Programmen = ein Programm berechnet das Gewünschte für alle Eingaben, für die sie stoppen

Reduzierbarkeit

Definition

- A ist auf B reduzierbar, wenn es eine totale und berechenbare Funktion $f: A \rightarrow B$ gibt, so dass für alle $x \in A$ gilt:
 $x \in A \Leftrightarrow f(x) \in B$
- Schreibweise: $A \leq_m B$
- wenn B entscheidbar ist, so ist dies auch A
 $\chi_A(x) = \chi_B(f(x))$
- Umkehrung:
 Wenn A unentscheidbar ist, dann ist B auch unentscheidbar
- $H_{\text{Java}} \leq_m H_C$
 $H_C \leq_m H_{\text{java}}$
 $\rightarrow f(x)$ ist hierbei ein Compiler zwischen den Sprachen Java und C
 \rightarrow alle Halteprobleme sind gleich schwer

Reduzierbarkeit

Totalitätsproblem

Reduktion Halteproblem auf Totalitätsproblem

- Konstruktion der Reduktionsfunktion ReduktionTotal mit $(i, x) \in H_{\text{java}} \Leftrightarrow \text{ReduktionTotal}(i, x) \in \text{Total}_{\text{java}}$
- ReduktionTotal(i, x) ist Programm Nr von folgendem Java Programm

```
int JavaSim(int P, int x) { ... }
{
    JavaSim(i, x);
    return 0
}
```

- Offensichtlich gilt: Reduktionsfunktion ist berechenbar
- $H_{\text{java}} \leq_m \text{Total}_{\text{java}} \Leftrightarrow \chi_H(i, x) = \chi_{\text{Total}}(\text{ReduktionTotal}(i, x))$
- Totalitätsproblem nicht entscheidbar, da Halteproblem nicht entscheidbar

Reduzierbarkeit

Äquivalenzproblem

Reduktion Halteproblem auf Äquivalenzproblem

- Konstruktion der Reduktionsfunktion ReduktionÄquival mit $(i, x) \in H_{\text{java}} \Leftrightarrow \text{ReduktionÄquival}(i, x) \in \text{Äquival}_{\text{java}}$
- $\text{ReduktionÄquival}(i, x) = (P_1, P_2)$ mit P_1 und P_2 Programm Nr von folgenden Java Programmen

<pre>int P₁() { return 1; }</pre>	<pre>int JavaSim(int P, int x) { ... } int P₂() { JavaSim(i, x); return 1; }</pre>
--	---
- Offensichtlich gilt: Reduktionsfunktion ist berechenbar
- $H_{\text{java}} \leq_m \text{Äquival}_{\text{java}} \Leftrightarrow \chi_H(i, x) = \chi_{\text{Äquival}}(\text{ReduktionÄquival}(i, x))$
- Äquivalenzproblem nicht entscheidbar, da Halteproblem nicht entscheidbar

Rekursivität von Sprachen

Rekursive Sprache

- Charakteristische Funktion ist berechenbar
→ Entscheidbarkeit
$$\chi_M(x) = \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{sonst} \end{cases}$$
- M_2 ist rekursiv und M_1 auf M_2 mit Reduktionsfunktion f reduzierbar ($M_1 \leq_m M_2$), dann ist M_1 rekursiv, da
$$\chi_{M_1}(x) = \chi_{M_2}(f(x)) = 1$$
- Abgeschlossen gegenüber Vereinigung, Durchschnitt und Komplement

Abgeschlossen gegen Durchschnitt, Komplement, Vereinigung


- Seien die Mengen M_1 und M_2 rekursiv
- Es existieren zwei funktionale Programme $f_i(x) = \chi_{M_i}(x)$
- $\chi_{M_1 \cap M_2}(x) = f_1(x) * f_2(x)$
- $\chi_{M_1^c}(x) = 1 - f_1(x)$
- $\chi_{M_1 \cup M_2}(x) = \min(f_1(x) + f_2(x), 1)$

Rekursiv aufzählbare Sprache

- Charakteristische Funktion ist im positiven Fall berechenbar
 → Semi-Entscheidbarkeit

$$\chi_M(x) = \begin{cases} 1 & \text{falls } x \in M \\ \text{undefiniert} & \text{sonst} \end{cases}$$
- Wenn M_2 rekursiv aufzählbar und
 M_1 auf M_2 mit Reduktionsfunktion f reduzierbar ($M_1 \leq_m M_2$),
 dann M_1 rekursiv aufzählbar,
 da $\chi_{M_1}(x) = \chi_{M_2}(f(x)) = 1$
 - Umkehrung:
 M_1 ist nicht rekursiv aufzählbar und
 M_1 auf M_2 mit Reduktionsfunktion r reduzierbar ($M_1 \leq_m M_2$),
 dann ist M_2 nicht rekursiv aufzählbar.
- Abgeschlossen gegenüber Vereinigung und Durchschnitt
- Nicht abgeschlossen gegenüber Komplement
 Lemma: M und M^c rekursiv aufzählbar $\Rightarrow M$ rekursiv
 → Halteproblem rekursiv aufzählbar, sein Komplement aber nicht

Abgeschlossen gegen Durchschnitt, Komplement, Vereinigung

- Seien die Mengen M_1 und M_2 rekursiv aufzählbar
- Es existieren zwei funktionale Programme $f_i(x) = \chi_{M_i}(x)$
- $\chi_{M_1 \cap M_2}(x) = f_1(x) * f_2(x)$
- $\chi_{M_1^c}(x) = 1 - f_1(x)$  terminiert nicht !!
- $\chi_{M_1 \cup M_2}(x) =$ Simuliere parallel $f_1(x)$ und $f_2(x)$,
Gib 1 aus, sofern $f_1(x)$ oder $f_2(x)$, anhält

Rekursivität von Sprachen

Zusammenhang Prädikatenlogik (I)

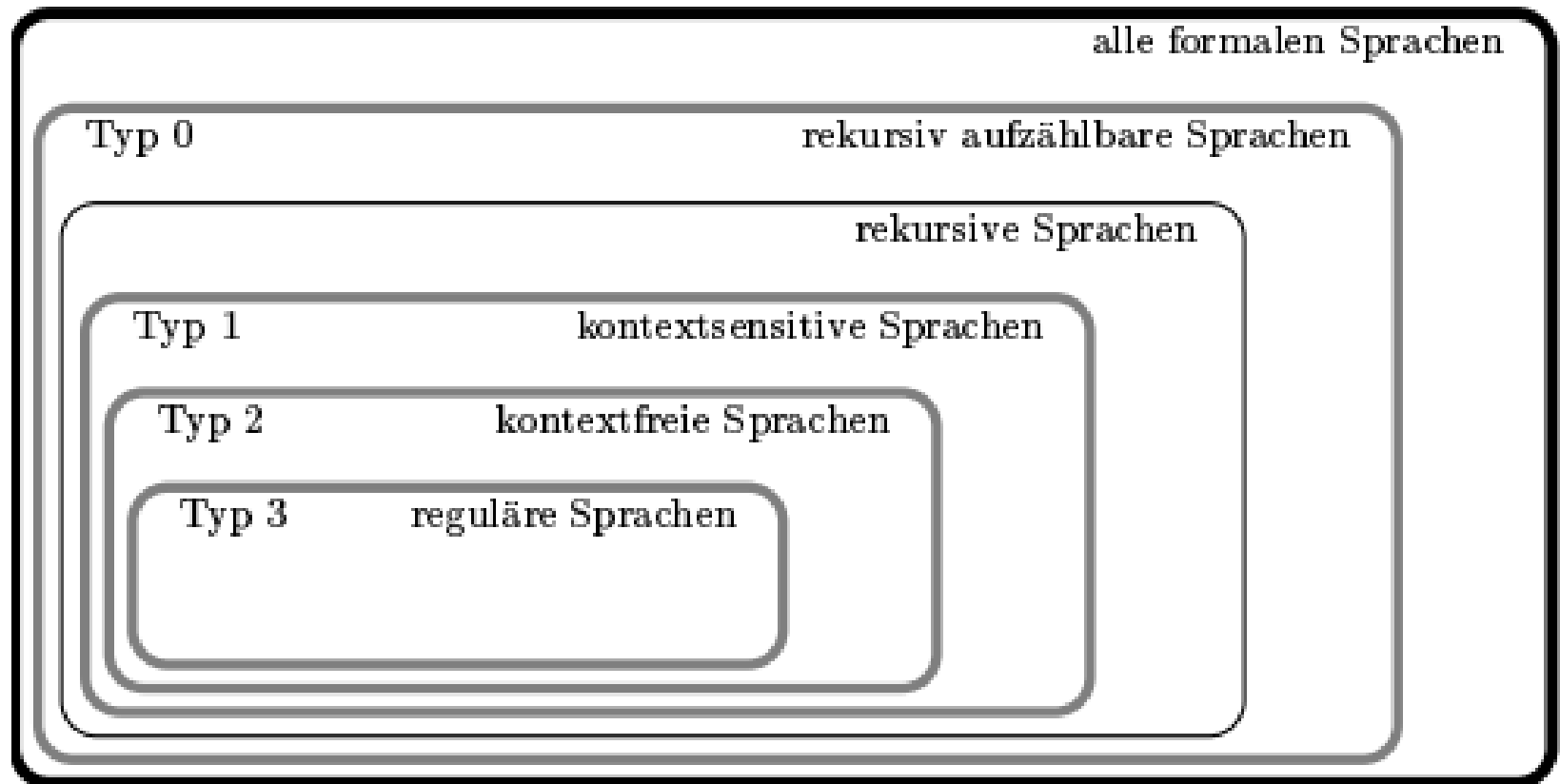
- Voraussetzung: berechenbare Prädikate
$$\chi_M(x) = \begin{array}{ll} 1 & \text{falls } x \in M \\ 0 & \text{sonst} \end{array}$$
- Quantoren für Vereinigung, Durchschnitt, Komplement von rekursiven Sprachen sind anwendbar
- Wie sieht es mit Existenzquantor und Allquantor aus?

Rekursivität von Sprachen

Zusammenhang Prädikatenlogik (II)

- Existenzquantor und Allquantor gelten nur bedingt
- Voraussetzung:
 - $T(P, x, t) = 1 \Leftrightarrow$ Programm P hält bei Eingabe x nach spätestens t Schritten
 - $H(P, x) = 1 \Leftrightarrow$ Programm P hält bei Eingabe x
- Dann ist
 - $T(P, x, t)$
berechenbar
 - $H(P, x) = \exists t T(P, x, t)$
nur semi-berechenbar, aber nicht berechenbar
 - $1-H(P, x) = \neg \exists t T(P, x, t) = \forall t \neg T(P, x, t)$
nicht semi-berechenbar
 - $Total(p) = \forall x \exists t T(P, x, t)$
weder entscheidbar noch rekursiv aufzählbar noch Komplement rekursiv aufzählbar

Chomsky Hierarchie



Zusammenfassung/ Ausblick

- Es kann keinen Algorithmus geben, mit dem überprüft werden kann, ob ein beliebiges Programm terminiert,
- für eigene Programme sollte das gehen! Wie??
- Verschiedene Probleme sind ineinander überführbar
→ Reduktion von Problemen
- In der Berechenbarkeitstheorie gibt es eine Hierarchie von Sprachen und ihren Eigenschaften
- Weiterhin kann es keinen Algorithmus geben, der für eine beliebiges Programm prüft, ob dieses eine Nicht-Triviale Eigenschaft besitzt.
→ Satz vom Rice