GTI

Hannes Diener

ENC B-0123,
diener@math.uni-siegen.de

6. Juni 2013

Berechenbarkeit

Einleitung

In der zweiten Hälfte der Vorlesung GTI geht es uns darum den Berechenbarkeitsbegriff zu formalisieren und analysieren.

Oft hört man Sätze wie

- „Die Lösung ist berechenbar“
- „Für dieses Problem gibt es einen Algorithmus“
- „Das Problem lässt sich nicht mit Computern lösen“

Berechenbarkeit

Einleitung

Etwas Geschichte.

Das Wort Algorithmus ist eine Abwandlung des Namens des Gelehrten Muhammed al-Chwarizmi (ca. 783–850) dessen arabisches Lehrbuch „Über das Rechnen mit indischen Ziffern“(um 825) in der mittelalterlichen lateinischen Übersetzung mit den Worten „Dixit Algorismi“(Algorismi hat gesagt) beginnt. Später wurde „Algorithmi“synonym mit Anleitungen zum Lösen mathematischer Probleme. Fälschlicherweise wurde die Endung „-i“als lateinischer Plural von „-us“interpretiert, was zum heute gebräuchlichen Wort „Algorithmus“führte.

Berechenbarkeit

Einleitung

Was soll durch einen Algorithmus berechnet werden?

Berechenbarkeit

Einleitung

Was soll durch einen Algorithmus berechnet werden?

Ein Algorithmus soll eine Eingabe in eine Ausgabe überführen.
Mathematisch ist das nichts anderes als eine Funktion.

Berechenbarkeit

Einleitung

Was soll durch einen Algorithmus berechnet werden?

Ein Algorithmus soll eine Eingabe in eine Ausgabe überführen.
Mathematisch ist das nichts anderes als eine Funktion.

Welche Eingabe- und Ausgabewerte wollen wir zulassen?

Berechenbarkeit

Einleitung

Was soll durch einen Algorithmus berechnet werden?

Ein Algorithmus soll eine Eingabe in eine Ausgabe überführen.
Mathematisch ist das nichts anderes als eine Funktion.

Welche Eingabe- und Ausgabewerte wollen wir zulassen?

Wir beschränken uns auf natürliche Zahlen. (Wie wir später erkennen werden ist dies keine wirkliche Einschränkung).

Berechenbarkeit

Einleitung

Was ist nun ein Algorithmus?

Berechenbarkeit

Einleitung

Was ist nun ein Algorithmus?

Für unsere *einleitenden* Überlegungen wollen wir uns auf folgenden etwas vage Beschreibung stützen.

> *Ein Algorithmus ist eine aus endlich vielen Schritten bestehende, eindeutige und ausführbare Handlungsvorschrift zur Lösung eines Problems*

Berechenbarkeit

Einleitung

Eine erste Beobachtung:

Berechenbarkeit

Einleitung

Eine erste Beobachtung:

Wir müssen auch *partielle Funktionen* zulassen.
Eine partielle Funktion $\mathbb{N} \rightharpoonup \mathbb{N}$ ist für jede Eingabe $n \in \mathbb{N}$ entweder definiert und hat den Wert $f(n)$ oder ist undefiniert. In diesem Fall schreiben wir $f(n) = \perp$.

Berechenbarkeit

Einleitung

Eine erste Beobachtung:

Wir müssen auch *partielle Funktionen* zulassen.
Eine partielle Funktion $\mathbb{N} \rightharpoonup \mathbb{N}$ ist für jede Eingabe $n \in \mathbb{N}$ entweder definiert und hat den Wert $f(n)$ oder ist undefiniert. In diesem Fall schreiben wir $f(n) = \bot$.

(Mathematisch ist eine partielle Funktion $\mathbb{N} \rightharpoonup \mathbb{N}$ nichts anderes als eine Funktion $\mathbb{N} \to \mathbb{N} \cup \{\bot\}$. Für uns ist es aber praktisch beide Konzepte zu unterscheiden.)

Berechenbarkeit

Einleitung

Wenn wir deutlich machen wollen, daß eine Funktion überall definiert ist (also eine Funktion im herkömmlichen Sinn) sprechen wir von einer *totalen* Funktion.

Berechenbarkeit

Einleitung

Wenn wir deutlich machen wollen, daß eine Funktion überall definiert ist (also eine Funktion im herkömmlichen Sinn) sprechen wir von einer *totalen* Funktion.

Jede totale Funktion ist, nebenbei bemerkt, eine partielle Funktion. (Wir fordern ja nicht, daß es undefinierte Werte geben muß, sondern nur, daß es solche geben kann).

Berechenbarkeit

Einleitung

Eine zweite Beobachtung:

Nehmen wir nun an wir haben uns auf eine gewisse Formalisierung des Berechenbarkeitsbegriffes geeinigt (also z.B. eine Programmiersprache und deren Semantik erklärt). Ein Algorithmus in diesem Formalismus ist nichts anderes als eine Zeichenkette (also ein Wort) über irgendeinem endlichen Alphabet Σ. Im Falle von übliche Programmiersprachen z.B. eine Folge von Ascii-zeichen. Natürlich ist nicht jede solche Zeichenkette ein Algorithmus, aber

Berechenbarkeit

Einleitung

Eine zweite Beobachtung:

Nehmen wir nun an wir haben uns auf eine gewisse Formalisierung des Berechenbarkeitsbegriffes geeinigt (also z.B. eine Programmiersprache und deren Semantik erklärt). Ein Algorithmus in diesem Formalismus ist nichts anderes als eine Zeichenkette (also ein Wort) über irgendeinem endlichen Alphabet Σ. Im Falle von übliche Programmiersprachen z.B. eine Folge von Ascii-zeichen. Natürlich ist nicht jede solche Zeichenkette ein Algorithmus, aber

Satz

Es gibt höchstens so viele Algorithmen wie Wörter über Σ. Also gibt es höchstens abzählbar viele Algorithmen.

Berechenbarkeit

Einleitung

Gleichzeitig gilt aber

Satz

Es gibt überabzählbar viele (partielle) Funktionen $\mathbb{N} \to \mathbb{N}$.

Beweis.

Nehmen wir an wir haben eine Auflistung $f_1, f_2, \ldots$ aller Funktionen von $\mathbb{N} \to \mathbb{N}$. Dann ist auch $g : \mathbb{N} \to \mathbb{N}$ definiert durch $g(n) = f_n(n) + 1$ eine Funktion. g ist aber nicht in der Auflistung: nehmen wir an es gibt $k \in \mathbb{N}$ mit $g \equiv f_k$. Allerdings ist dann

$$f_k(k) = g(k) = f_k(k) + 1 \; ;$$

ein Widerspruch. D.h. die Auflistung war unvollständig, aber, da das Argument für jede Auflistung funktioniert kann es keine vollständige geben. □

Berechenbarkeit

Einleitung

Die in diesem Beweis angewandte Technik nennt man *Diagonalisierung*. Der Name stammt von folgender Sichtweise:

$$\begin{array}{ccccc}
\boxed{f_0(0)} & f_0(1) & \cdots & f_0(i) & \cdots \\
f_1(0) & \boxed{f_1(1)} & \cdots & f_1(i) & \cdots \\
\vdots & \vdots & \ddots & \vdots & \\
f_i(0) & f_i(1) & \cdots & \boxed{f_i(i)} & \cdots \\
\vdots & \vdots & & \vdots & \ddots
\end{array}$$

Stellen wir uns die Funktionen $f_1, f_2, \ldots$ jeweils als Zeile vor (wobei in der n-ten Spalte der Wert $f_i(n)$ steht), so erhalten wir g indem wir sicherstellen, daß g zumindest jeweils auf der Diagonalen mit keiner der Funktionen übereinstimmen kann.

Berechenbarkeit

Einleitung

Zusammen erhalten wir folgende Einsicht.

Es gibt also nur abzählbar viele Algorithmen.

Es gibt überabzählbar viele Funktionen $\mathbb{N} \to \mathbb{N}$.

Es muss also Funktionen geben, die sich nicht durch einen Algorithmus berechnen lassen.

$\longrightarrow$

Berechenbarkeit

Einleitung

Im folgenden wollen wir den Begriff der Berechenbarkeit einer Funktion $f : \mathbb{N} \rightharpoonup \mathbb{N}$ auf drei verschiedene Arten formalisieren:

1. Turingmaschinen (Maschinenmodell)
2. While-Programme (abstrakte Programmiersprache)
3. Rekursive Funktionen (rein mathematisches Modell)

Berechenbarkeit

Einleitung

Es gibt noch eine Vielzahl mehr Modelle mit noch einer größeren Vielzahl von Varianten

1. Turingmaschinen (Maschinenmodell)
2. While-Programme (abstrakte Programmiersprache)
3. Rekursive Funktionen (rein mathematisches Modell)
4. λ-Kalkül
5. Registermaschinen
6. Markov-Algorithmen (Wortersetzungssystem)
7. Kombinatorische Logik

Berechenbarkeit

Einleitung

Die überraschende Pointe der Berechenbarkeitstheorie im 20. Jhd. ist, daß all diese Modelle äquivalent sind. Ausserdem gibt es kein Berechenbarkeitsmodell, daß zu einem anderen Begriff geführt hat. Es ist also überzeugend (aber nicht/nie beweisbar), daß eines bzw. alle dieser Modelle die korrekte Wahl ist.

> **(Church-Turing-These)** *Die in einem der oberen Sinne berechenbaren Funktionen sind genau die im intuitiven Sinne berechenbaren*