Berechenbarkeit

oder: was Programmiersprachen können

mathematisch: ein Programm definiert eine Funktion

p(eingaben,maschinenzustand)→ausgabe

es gibt berechenbare und nicht berechenbare Funktionen

→ Grenzen der Berechenbarkeit bestimmen
Grenzen dessen, was implementierbar ist

Berechenbarkeit

oder: was Programmiersprachen können

Beispiel 1:

der Ausdruck 3+2 hat einen definierten Wert

der Ausdruck 3:0 hat keinen definierten Wert

Allgemein:

ein Ausdruck hat keinen definierten Wert bei

a. Konflikt zwischen Operator und Operand
b. Unendlichkeit der Berechnung

Abhilfe bei a. ?

Berechenbarkeit

oder: was Programmiersprachen können

Beispiel 2:

```
f(x:int) =
    if x = 0 then 0 else x + f(x-2)
```

Problem: sog. partielle Funktion

f(2)= 2

f(3)= ?

f(4)= 6

Berechenbarkeit

oder: was Programmiersprachen können

Partielle Funktionen:

- Funktion, die auf einigen Argumenten definiert ist, auf anderen nicht
- als *Programm* liefert sie bei einigen Eingaben ein Ergebnis, bei anderen terminiert das Programm nicht
- formal:

  $f: A \rightarrow B$ mit geordneten Paaren f aus $A \times B$, die folgende Bedingung erfüllt:

  ist (x,y) Element von f und (x,z) Element von f

  dann ist $y=z$

Berechenbarkeit

oder: was Programmiersprachen können

zum Vergleich: formale Definition einer Funktion:

f: A→B mit geordneten Paaren *f* aus *A x B,* die folgenden Bedingungen erfüllt:

- ist *(x,y)* Element von *f* und *(x,z)* Element von *f,* dann ist *y=z*
- zu jeden *x* aus *A* existiert ein *y* aus *B* mit *(x,y)* Element aus *f*

Umgangsprachlich:

- gleiche Eingaben liefern immer die gleiche Ausgabe
- jedes Argument hat mindestens einen Wert im Wertebereich

Wie lautet korrekte Definition der Beispielfunktion als Funktion?

Berechenbarkeit

oder: was Programmiersprachen können

- Berechenbarkeitstheorie beschreibt exakt, welche Funktionen „im Prinzip“ berechenbar sind
- Klasse von Funktionen oft *partielle rekursive Funktionen* genannt
- „im Prinzip berechenbar“: Funktionen könnten extrem lange brauchen, bis sie halten

- *Intuitiv*: eine Funktion f ist berechenbar, wenn es ein Programm p gibt, das sie berechnet

  $p(x) = y$ mit x aus A, y aus B, $y=f(x)$ und $f:A \rightarrow B$

Berechenbarkeit

oder: was Programmiersprachen können

Probleme:

- erst muss solch ein Programm in einer Programmiersprache geschrieben werden
- evtl. lässt sich solch ein Programm in Sprache S1 schreiben, in Sprache S2 nicht

Church‘sche These:

„ *Die Klasse der intuitiv berechenbaren Funktionen ist genau die Klasse der Turing-berechenbaren Funktionen .“*

Berechenbarkeit

oder: was Programmiersprachen können

Turingmaschine (von Alain Turing):

idealisierte Rechenmaschine mit

- einem unendliche langen Speicherband mit unendlich vielen sequentiell angeordneten Feldern (pro Feld ein Zeichen erlaubt)
- einem Lese-Schreibkopf, der sich feldweise (r,l) bewegen, lesen und schreiben kann (definierter Zeichenvorrat!)
- einer Überführungsfunktion, die angibt, wie die TM schrittweise Bandinhalt, ihren Zustand und Position des Lese-Schreibkopfs ändert
- Anfangszustand, Menge von Endzuständen

Berechenbarkeit

oder: was Programmiersprachen können

Beispiel:

Zeichenvorrat: {* (leeres Zeichen),0,1}

Startzustand: 0, Endzustand: {s}

Überführungsfunktion:

aktuell. Zustand	geles. Symbol		schreibe Symbol	neuer Zustand	Kopf-beweg.
0	*	→	1	s	r
0	0	→	1	s	r
0	1	→	1	s	r

diese TM druckt eine 1 auf ein leeres oder beschriebenes Feld und geht einen Schritt nach rechts

Berechenbarkeit

oder: was Programmiersprachen können

- ebenfalls von Church: Lambda-Kalkül
- Turing bewies Äquivalenz von Lambda-Kalkül zu Turingmaschine
- es ex. weitere äquivalente Algorithmusbegriffe

- alle Programmiersprachen sind „turing-vollständig“: sie können genau alle partiell rekursiven Funktionen umsetzen

  → Berechenbarkeitstheorie hilft nicht zur Klassifikation verschiedener Programmiersprachen bzgl. ihrer Ausdruckskraft

Berechenbarkeit

oder: was Programmiersprachen können

- es gibt auch nicht berechenbare Funktionen
- bekannteste: das Halteproblem
- Halteproblem:

  gegeben ist

  Programm P mit genau einer Eingabe,

  ein Eingabestring x,

  Aufgabe ist

  berechne, ob P bzgl. x hält

- Unentscheidbarkeit des Halteproblems impliziert, dass es Programmfehler gibt, die nicht im Voraus erkannt werden können

Berechenbarkeit

oder: was Programmiersprachen können

- Beispiel: while-Schleife
  - Programmierer möchte, dass die while-Schleife hält
  - Compiler: warnen, falls while-Schleife nicht halten wird
  - Halteproblem liefert: Compiler kann das gar nicht vorhersagen