Unberechenbar!

Andreas Fackler

7. September 2010

Funktionen

Funktion $f : A \to B$**:** Ordnet jedem $a \in A$ ein $b \in B$ zu.

Beispiel 1: Quadratfunktion

$s : \mathbb{N} \to \mathbb{N}, s(n) = n^2$ ordnet jeder Zahl ihr Quadrat zu:

n:	0	1	2	3	4	5	6	7	...
$s(n)$:	0	1	4	9	16	25	36	49	...

Beispiel 2: Fibonacci-Zahlenfolge

$F : \mathbb{N} \to \mathbb{N}$ mit $F(0)=0$, $F(1)=1$ und $F(n+2) = F(n)+F(n+1)$:

n:	0	1	2	3	4	5	6	7	...
$F(n)$:	0	1	1	2	3	5	8	13	...

Programme

Programm: Liste von Anweisungen für einen Computer.

Beispiel 1: Gibt das Quadrat der Eingabe aus.

```
def square(n):
    return n*n
```

Beispiel 2: Gibt die n-te Fibonacci-Zahl aus.

```
def fibonacci(n):
    if n<2: return n
    else: return fibonacci(n-2)+fibonacci(n-1)
```

Berechenbarkeit

Schreibe $\mathtt{p}(a)$ für die Ausgabe des Programms $\mathtt{p}$ bei Eingabe a.
Zum Beispiel ist $\mathtt{square}(5)$ gleich 25.

Gibt $\mathtt{p}(a)$ für jedes $a \in A$ ein $b \in B$ zurück, so definiert $\mathtt{p}$ eine Funktion $f : A \to B$, nämlich:

$$f(a) = \mathtt{p}(a)$$

Sprich: $\mathtt{p}$ *berechnet* f.
f heißt *berechenbar*, wenn es ein Programm gibt, das f berechnet.

Berechenbarkeit

Beispiele

Das Programm `square` berechnet die Funktion s.
Das Programm `fibonacci` berechnet die Funktion F.

Gegenbeispiel

Dieses Programm hier berechnet keine Funktion $f : \mathbb{N} \to \mathbb{N}$:

```
def p(n):
    b=0
    while 11>4: b=b+1
    return b
```

Unberechenbare Funktionen

Gibt es Funktionen $f : \mathbb{N} \to \mathbb{N}$, die nicht berechenbar sind?

- Funktionen, die man nicht einmal definieren kann?
- "Zufällige" Funktionen?

Aber ganz "konkrete" nicht berechenbare Funktionen ...?

Fleißige Biber

Zahl $n \in \mathbb{N}$
$\Rightarrow$ nur endlich viele Programme `p` mit höchstens n Zeichen
$\Rightarrow$ nur endlich viele Werte `p(`n`)` mit absturzsicherem `p`
$\Rightarrow$ ein größter Wert `p(`n`)`

Ist `p(`n`)` maximal, so heißt `p` "fleißiger Biber" ("busy beaver").

Beispiel

Ein fleißiger Biber zur Zahl $n = 16$ ist:

```
def p(n):
    return 99
```

Die Fleißiger-Biber-Funktion

Definiere $\Sigma(n) = \mathrm{p}(n) + 1$, wobei p ein fleißiger Biber zu n ist, und $\Sigma(n) = 0$, falls es keinen gibt.
$\Sigma : \mathbb{N} \to \mathbb{N}$ ist eine Funktion.

$$
\begin{aligned}
\Sigma(14) &= 0 \\
\Sigma(15) &= 16 \\
\Sigma(16) &= 100 \\
\Sigma(17) &= 1000 \\
\Sigma(18) &= 3934640807529653757 5425 \\
\Sigma(19) &= 3950085484118525497024052097529573238897 04 \\
&\quad 9443942607252543835470215867415609371393 30 \\
&\quad 7890303310063934009000876624970111349493580
\end{aligned}
$$

Die Fleißiger-Biber-Funktion

Ist p absturzsicher mit höchstens n Zeichen, so ist

$$\Sigma(n) > \mathrm{p}(n).$$

Σ wächst *schneller als jede berechenbare Funktion*!

Es gibt aber auch "kleine" unberechenbare Funktionen: Sei P die Menge aller Programme. $H : P \to \mathbb{N}$ mit

$$H(\mathrm{p}) = \begin{cases} 1 & \text{, falls } \mathrm{p} \text{ nicht abstürzt} \\ 0 & \text{, falls } \mathrm{p} \text{ abstürzt} \end{cases}$$

ist nicht berechenbar!

Das Halteproblem

Satz (Turing 1936)

Es gibt kein Programm, das für beliebige Eingaben `p` *und* `x` *entscheidet, ob der Programmaufruf* `p(x)` *jemals anhält.*

Beweis: Angenommen, `haelt` wäre so ein Programm:

```
>>> haelt(einsdurch, 5)
True
>>> haelt(einsdurch, 0)
False
```

Alan Turing,
1912 – 1954

Denkmal (Manchester)

Das Halteproblem

Wir verwenden `haelt` in einem neuen Programm:

```
def absurd(p):
    if(haelt(p,p)):
        b=0
        while 11>4: b=b+1
    else: return 0
```

`absurd(p)` stürzt also genau dann ab, wenn `p(p)` nicht abstürzt.

```
>>> absurd(absurd)
```

Das Halteproblem