Beispiel einer nicht-berechenbaren Funktion

Aufzählbarkeit

- im Folgenden: um mit Hilfe der Diagonalisierung konkrete nicht berechenbare Funktionen zu finden, reicht es nicht die berechenbaren Funktionen abzuzählen
- wir verlangen noch, dass die Abzählung berechenbar ist – rekursiv aufzählbar

Definition 1.21 (Rekursive Aufzählbarkeit)

Sei Σ ein Alphabet. Eine Menge $M \subseteq \Sigma^*$ heißt **rekursiv aufzählbar** (oder schlicht **aufzählbar**), falls

- $M = \emptyset$ oder
- es eine totale, surjektive und berechenbare Funktion $f : \mathbb{N} \to M$ gibt.

Beispiele und Eigenschaften aufzählbarer Mengen

Theorem 1.22

Sei Σ ein Alphabet, dann ist Σ^* aufzählbar.

Beweis. Folgt aus der Berechenbarkeit der Umkehrung der Abzählungsfunktion aus Theorem 1.16 (Übung).

Aufzählbarkeit vs. Abzählbarkeit:

- im Gegensatz zur Abzählbarkeit gilt jetzt nicht mehr, dass jede Teilmenge einer aufzählbaren Menge auch aufzählbar ist
- zum Beispiel gibt es $L \subseteq \Sigma^*$ die nicht aufzählbar sind (Warum?)
- das nächste Theorem muss daher gesondert bewiesen werden

Theorem 1.23

Die Menge aller berechenbaren Funktionen ist aufzählbar.

Beweis. Analog zur Abzählbarkeit, zählen wir diese Menge in Form von miniPy-Programmen auf. Gesucht ist also eine totale, surjektive und berechenbare Funktion $\mathrm{enum}_{\mathbb{F}_{\mathsf{ber}}} : \mathbb{N} \to L_{\mathsf{miniPy}}$.

- Erinnerung: Abzählbarkeit der Menge der berechenbaren Funktionen haben wir mit Hilfe der (bereits mehrfach diskutierten) Funktion $(i_1 \dots i_s)_b := \sum_{j=1}^{s} i_j \cdot b^{s-j}$ bewiesen
- für die Aufzählung benötigen wir die Umkehrfunktion und müssen zeigen, dass diese berechenbar ist
- die Umkehrfunktion $\mathrm{b\_ad} : \mathbb{N} \to \{0, \dots, b-1\}^*$ ist definiert als die Funktion, die eine natürliche Zahl in ihre b-adische Darstellung umwandelt
  - Übung: Zeige, dass $\mathrm{b\_ad}$ berechenbar ist.

Beweis Theorem 1.23 (cont.):

- sei $L_{\text{miniPy}} \subseteq \Sigma^*$, wobei zur Vereinfachung $\Sigma := \{\texttt{while}, \texttt{\#endwhile}, \texttt{x}, \texttt{+=1}, \texttt{-=1}, \texttt{!=0:}, \texttt{;}\}$
  - die Wahl von Σ stellt keine Einschränkung an die Variablenmenge dar, denn z.B. $\texttt{xxxx} \hat{=} x_4$
- wähle Nummerierung $\texttt{while} \hat{=} 1$, $\texttt{\#endwhile} \hat{=} 2$, $\texttt{x} \hat{=} 3$, $\texttt{+=1} \hat{=} 4$, $\texttt{-=1} \hat{=} 5$, $\texttt{!=0:} \hat{=} 6$, $\texttt{;} \hat{=} 7$ und damit $b = 7 + 1 = 8$
- definiere:

$$\text{enum}_{\mathbb{F}_{\text{ber}}}(n) := \begin{cases} w, & \text{falls } w = 8\_\text{ad}(n) \text{ und } w \text{ ist ein syntaktisch korrektes miniPy-Programm} \\ \texttt{while x!=0: x+=1 \#endwhile}, & \text{falls } 8\_\text{ad}(n) \text{ enthält 0en oder ist kein syntaktisch korrektes miniPy-Programm} \end{cases}$$

n.z.z.: $\text{enum}_{\mathbb{F}_{\text{ber}}}$ ist surjektiv (Übung) und berechenbar (siehe `enum_miniPy.py`)
Übung: Gib eine Aufzählung der TM an.

Beobachtungen:

- durch $\mathrm{enum}_{\mathbb{F}_{\text{ber}}}$ wird jeder Zahl ein Programm zugewiesen
  - Beachte: die Injektivität der Funktion wird nur für das Programm `while x!=0: x+=1 #endwhile` verletzt, da diesem mehrere Zahlen zugeordnet werden (Warum?)
- mit der Funktion $()_8$ kann für ein gegebenes Programm dessen, durch $\mathrm{enum}_{\mathbb{F}_{\text{ber}}}$ festgelegte, eindeutige (bis auf `while x!=0: x+=1 #endwhile`) Zahl berechnet werden
  - Welche Zahl wird `while x!=0: x+=1 #endwhile` zugewiesen?

Programme (und damit die berechenbaren Funktion) lassen sich eindeutig durch einen Algorithmus nummerieren. Dabei bezeichnen wir das

Programm mit der Nummer j als P_j (und dessen berechnete Funktion als f_{P_j}).

Eine solche Nummerierung ist als Gödelisierung bekannt (nach *Kurt Gödel*).

Eine nicht-berechenbare Funktion

Das Pseudocode-Verfahren `selbstanw`:

```
def selbstanw(⟨P⟩):
    if P auf Eingabe ⟨P⟩ termininert:
        while True:
            pass
    else:
        return '1'
```

Was passiert, wenn `selbstanw` auf sich selbst angewendet wird?

Den Algorithmus(!) `selbstanw` gibt es nicht!

`selbstanw` beschreibt damit eine **nicht-berechenbare Funktion** $f_{\texttt{selbstanw}}$.

Idee zur Konstruktion der Funktion $f_{\texttt{selbstanw}}$:

- **Zweite Anwendung des Diagonalenarguments**
- Welche Erkenntnisse wurden im Programm `selbstanw` ausgenutzt?
  - durch die Programme $\mathrm{enum}_{\mathbb{F}_{\mathsf{ber}}}$ und 8_ad lässt sich jeder Zahl ein eindeutiges Programm zuordnen (Theorem 1.23)
    - damit sind berechenbare Funktionen und deren Eingaben (als natürliche Zahlen codiert) auflistbar
    - weiterhin folgt daraus (und bereits aus den Betrachtungen zur Codierungen von Zeichenketten als Zahlen), dass es o.B.d.A genügt einstellige Funktionen zu betrachten
  - mit Theorem 1.12 gibt es ein universelles Programm, welches die Funktion eines als Eingabe gegebenen Programms berechnet

- wir sind im Folgenden nur noch daran interessiert, ob eine berechenbare Funktion auf einer Eingabe definiert ist oder nicht
- dies lässt sich wie folgt darstellen:

	0	1	2	$\cdots$
f_{P_0}	def/nicht def.	def./nicht def.	def/nicht def.	
f_{P_1}	def/nicht def.	def./nicht def.	def/nicht def.	
f_{P_2}	def/nicht def.	def./nicht def.	def/nicht def.	
$\vdots$				$\ddots$

Definiere (partielle) Funktion $f_{\texttt{selbstanw}} \colon \mathbb{N} \to \mathbb{N}$:

$$f_{\texttt{selbstanw}}(n) := \begin{cases} 1, & \text{falls } f_{P_n}(n) \text{ undefiniert;} \\ \text{n. d.}, & \text{sonst.} \end{cases}$$

Lemma 1.24

$f_{\texttt{selbstanw}}$ ist nicht berechenbar.

Beweis. (durch Widerspruch) **Annahme:** $f_{\texttt{selbstanw}}$ ist berechenbar

- dann gibt es ein j so, dass das Programm P_j die Funktion berechnet, also gilt $f_{P_j}(n) = f_{\texttt{selbstanw}}(n)$ für alle $n \in \mathbb{N}$
- Was passiert nun mit der Programm P_j bei Eingabe j?

$$
\begin{aligned}
& P_j \text{ hält bei Eingabe } j \text{ nicht an} \\
\overset{\text{Def. Ber.}}{\Leftrightarrow} \quad & f_{P_j}(j) \text{ ist undefiniert} \\
\overset{\text{Def. } f_{\texttt{selbstanw}}}{\Leftrightarrow} \quad & f_{\texttt{selbstanw}}(j) = 1 \\
\overset{f_j = f_{\texttt{selbstanw}}}{\Leftrightarrow} \quad & f_{P_j}(j) = 1 \text{ (Hier ist schon der Widerspruch!)} \\
\overset{\text{Def. Ber.}}{\Leftrightarrow} \quad & P_j \text{ hält bei Eingabe } j \text{ an}
\end{aligned}
$$

Widerspruch! Es folgt: $f_{\texttt{selbstanw}}$ ist nicht berechenbar.

- die Funktion $f_{\texttt{selbstanw}}$ wird später noch von großer Wichtigkeit sein
- sie drückt im Endeffekt die Frage aus, ob ein Programm einem anderen Programm „ansehen“ kann ob es hält oder nicht – siehe Halteproblem (später)
- Übung: Warum wurde für die Konstruktion der Funktion $f_{\texttt{selbstanw}}$ eigentlich eine Aufzählung aller Funktionen benutzt und nicht nur eine Abzählung?

Lernziele

Man sollte ...

- Abzählungen und Aufzählungen angeben können
- „einfache“ Aufzählungen implementieren können
- zeigen können, dass eine Menge nicht-abzählbar/nicht-aufzählbar ist
  - durch Diagonalisierung
  - oder der Anwendung von Eigenschaften des Begriffs Abzählbarkeit/Aufzählbarkeit