

Berechenbarkeit

- ▶ Endliche Automaten und Kellerautomaten erkennen nicht alle algorithmisch beschreibbaren Sprachen.
- ▶ Mächtigeres Berechnungsmodell: z.B. **PASCALchen**
 - **Syntax**: *while*-Programm
 - **Semantik**: partielle Funktion auf den natürlichen Zahlen
- ▶ **Berechenbare Funktion**: Eine Funktion heißt **berechenbar**, falls sie Semantikfunktion eines *while*-Programms ist.

Ein *while*-Programm

```
begin  
   $X4 := 0$ ;  $X1 := X2$ ;  
  while  $X4 \neq X3$  do begin  
     $X4 := succ(X4)$ ;  
     $X1 := succ(X1)$   
  end  
end
```

} $X1 := X2 + X3$

Semantikfunktion für *while*-Programme

Sei P ein k -variables *while*-Programm und $j \in \mathbb{N}$. Dann ist die j -stellige **Semantikfunktion** von P

$$SEM_P: \mathbb{N}^j \rightarrow \mathbb{N}$$

für die Argumente $(x_1, \dots, x_j) \in \mathbb{N}^j$ nach folgenden Regeln definiert:

(1) Aus den Argumenten (x_1, \dots, x_j) wird eine Eingabe $a \in \mathbb{N}^k$ hergestellt:

$$(x_1, \dots, x_j) \rightsquigarrow \begin{cases} (x_1, \dots, x_k) & \text{falls } j \geq k \\ (x_1, \dots, x_j, 0, \dots, 0) & \text{falls } j < k \end{cases}$$

(2) P wird mit Eingabe a berechnet.

(3) Terminiert die Berechnung mit der Ausgabe (y_1, \dots, y_k) , so ist $SEM_P(x_1, \dots, x_j) = y_1$.

(4) Terminiert sie nicht, ist $SEM_P(x_1, \dots, x_j)$ undefiniert.

Berechenbare Funktion

Eine partielle Funktion $f: \mathbb{N}^j \rightarrow \mathbb{N}$ heißt **berechenbar**, wenn ein *while*-Programm existiert mit

$$f = SEM_P^{(j)}.$$

Churchsche These

Die Menge der (durch *while*-Programme) berechenbaren Funktionen ist genau die Menge der partiellen Funktionen $f: \mathbb{N}^j \rightarrow \mathbb{N}$, die im intuitiven Sinne berechenbar sind.

Programme als Eingaben für Programme

- ▶ Verhalten eines Programms hängt vom eingegebenen Programm ab.

Beispiel: Interpreter für PASCALchen



Repräsentation von *while*-Programmen als natürliche Zahlen (Gödelnumerierung)

1. Umwandlung der Zeichen, aus denen *while*-Programme bestehen, in Bitmuster
 - ▶ Der Zeichensatz A von PASCALchen besteht aus 22 Zeichen.
 - ▶ Sei $code: A \rightarrow \{0, 1\}^*$ eine injektive Abbildung mit $code(a) = 1b_1 \cdots b_5$

$code(a)$ liefert eine **eindeutige** 6-Bit-Darstellung von a , die mit 1 beginnt.

2. Repräsentation von *while*-Programmen als Bitmuster

$code^*: A^* \rightarrow \{0, 1\}^*$ mit

(i) $code^*(\lambda) = \lambda$ und

(ii) $code^*(av) = code(a)code^*(v)$ für $a \in A$, $v \in A^*$.

Für jedes *while*-Programm P liefert $code^*(P)$ ein Bitmuster.

3. Umwandlung von *while*-Programmen in natürliche Zahlen

Jedes Bitmuster lässt sich eindeutig als Binärdarstellung einer natürlichen Zahl auffassen.

Injektivität der Gödelnumerierung

Lemma

Die Abbildung $code^*$ ist injektiv.

Index eines *while*-Programms

Der **Index** eines *while*-Programms P ist die natürliche Zahl, deren Binärdarstellung $code^*(P)$ ist.

Bestimmung des Programms eines Indexes

1. Umwandlung von natürlichen Zahlen in Bitmuster

Jede natürliche Zahl n läßt sich eindeutig in ein Bitmuster $B(n)$ umwandeln.

2. Umwandlung von Bitmustern in *while*-Programme

Für alle $B \in \{0, 1\}^*$ sei $decode: \{0, 1\}^* \rightarrow A^*$ definiert durch $decode(B) = \lambda$ für alle B kürzer als 6 und

$$decode(b_1 \cdots b_6 B) = \begin{cases} a \text{ decode}(B) & \text{wenn} \\ & code(a) = b_1 \cdots b_6 \\ \lambda & \text{sonst.} \end{cases}$$

3. Aufzählen von *while*-Programmen

- Falls $decode(B(n))$ ein *while*-Programm ist:

$$AUFZÄHLUNG(n) = decode(B(n))$$

- Falls $decode(B(n))$ kein *while*-Programm ist:

$$\begin{aligned} AUFZÄHLUNG(n) = & \textit{begin} \\ & X1 := 0; X2 := 1; \\ & \textit{while } X1 \neq X2 \textit{ do } X1 := X1 \\ & \textit{end} \end{aligned}$$

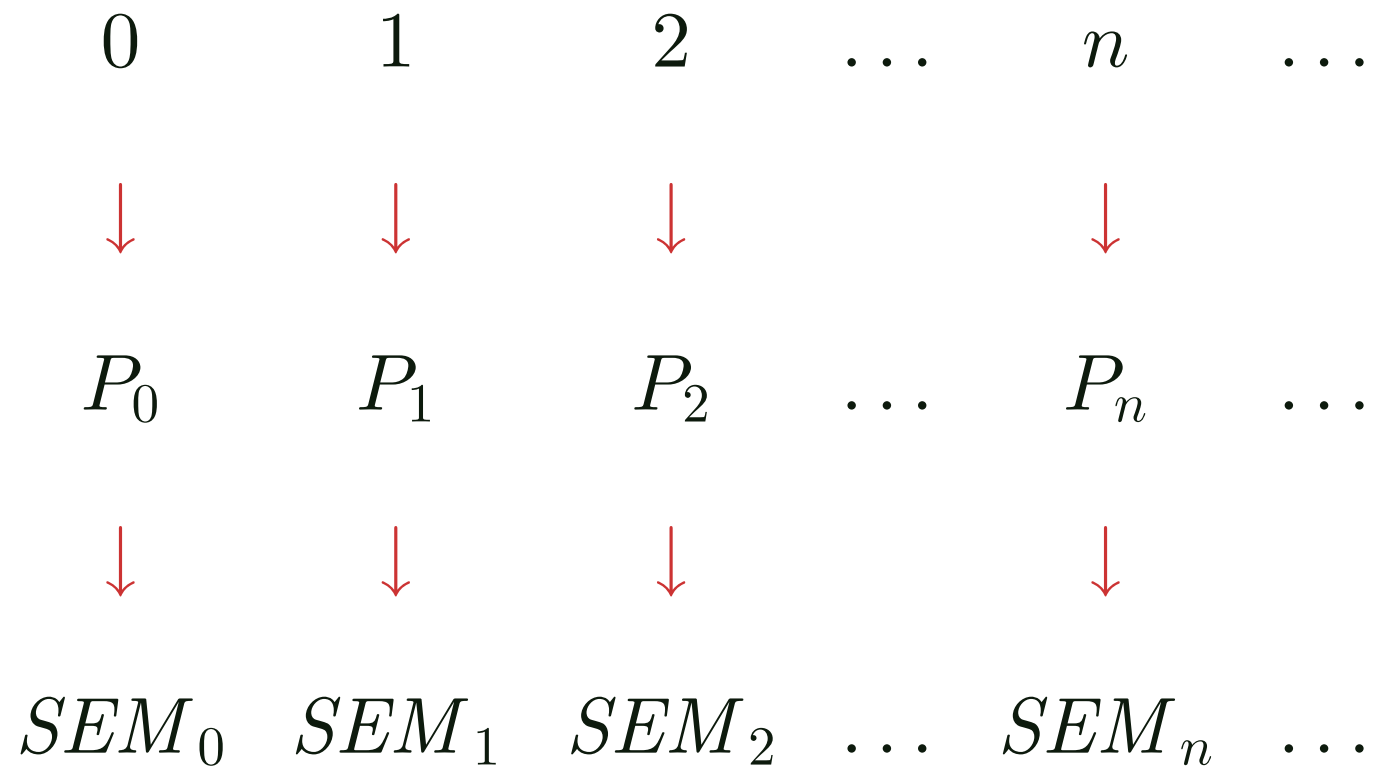
Lemma

Durch *AUFZÄHLUNG* wird der Index eines Programms P auf P abgebildet.

Schreibweisen:

- ▶ $P_n = \text{AUFZÄHLUNG}(n)$: *while*-Programm mit Index n
- ▶ $SEM_i = SEM_{P_i}^{(1)}$

Folgerung: Berechenbare Funktionen und *while*-Programme sind aufzählbar



Effektive Aufzählbarkeit

- ▶ Eine Menge M heißt **aufzählbar** (**abzählbar**), wenn es eine surjektive Abbildung $num: \mathbb{N} \rightarrow M$ gibt.
- ▶ M heißt **effektiv aufzählbar** (**rekursiv aufzählbar**), wenn diese Numerierung durch einen Algorithmus vorgenommen wird.

Effektivität von *AUFZÄHLUNG*



1. $B(n)$: algorithmisch bestimmbar
2. $decode(B(n))$; algorithmisch bestimmbar
3. Ist $decode(B(n))$ ein *while*-Programm?: algorithmisch entscheidbar

Existenz nicht-berechenbarer Funktionen

Satz

Es gibt eine Funktion $f: \mathbb{N} \rightarrow \mathbb{N}$, die nicht berechenbar ist.

Das Halteproblem

- ▶ **Eingabe:** Ein *while*-Programm P und eine Eingabe a für P
- ▶ **Ausgabe:** *True*, falls P mit der Eingabe a hält
False, falls P mit der Eingabe a nicht hält

Unlösbarkeit des Halteproblems

- Das Problem, ob ein Programm hält, wenn es auf den eigenen Index angewendet wird, ist nicht lösbar, d.h.:

Die Funktion *HALTEPROBLEM* : $\mathbb{N} \rightarrow \mathbb{N}$, die definiert ist für alle $i \in \mathbb{N}$ durch

$$\textit{HALTEPROBLEM}(i) = \begin{cases} 1 & \text{wenn } SEM_i(i) \text{ definiert ist} \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

Unlösbarkeit des allgemeinen Halteproblems

- Das Problem, ob ein Programm auf eine beliebige Eingabe hält, ist nicht lösbar, d.h.:

Die Funktion *HALTEPROBLEM2*: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ mit

$$\textcolor{red}{HALTEPROBLEM2}(i, j) = \begin{cases} 1 & \text{wenn } SEM_i(j) \\ & \text{definiert ist} \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.