# Berechenbarkeit

- Endliche Automaten und Kellerautomaten erkennen nicht alle algorithmisch beschreibbaren Sprachen.
- Mächtigeres Berechnungsmodell: z.B. PASCALchen

Syntax: $while$-Programm

Semantik: Berechnungen

# Ein *while*-Programm $P$

$$\left.\begin{array}{l} begin \\ \quad X4 := 0; X1 := 0; \\ \quad while\ X4 \neq X3\ do\ begin \\ \quad\quad X1 := X1 + X2; \\ \quad\quad X4 := succ(X4) \\ \quad end \\ end \end{array}\right\} X1 := X2 * X3$$

# Eine Berechnung von $P$

$$
\begin{array}{ll}
(0,3,2,7) & X4 := 0 \\
(0,3,2,0) & X1 := 0 \\
(0,3,2,0) & X4 \neq X3 \\
(0,3,2,0) & X1 := X1 + X2 \\
(3,3,2,0) & X4 := succ(X4) \\
(3,3,2,1) & X4 \neq X3 \\
(3,3,2,1) & X1 := X1 + X2 \\
(6,3,2,1) & X4 := succ(X4) \\
(6,3,2,2) & X4 \neq X3 \\
(6,3,2,2) &
\end{array}
$$

# Semantikfunktion für *while*-Programme

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

$$SEM_P \colon \mathbb{N}^j \to \mathbb{N}$$

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

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

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

(2) $P$ wird mit Eingabe $a$ berechnet.

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

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

# Bemerkungen

1. Semantikfunktion total, falls jede Berechnung terminiert.
2. Wahlfreiheit von $j \Longrightarrow$ Jedes Programm berechnet unendlich viele Funktionen, die sich aber nur wenig voneinander unterscheiden.
3. Statt $SEM_P$ kann auch $SEM_P^{(j)}$ geschrieben werden.

# Berechenbare Funktion

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

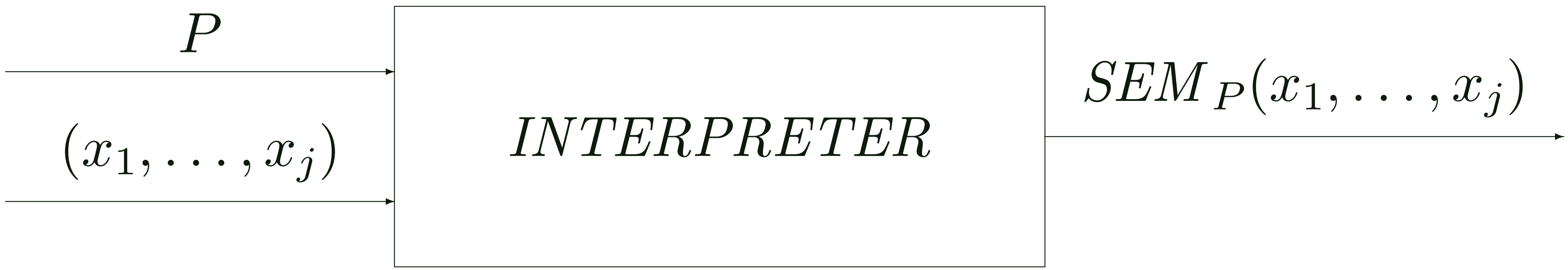
$$f = SEM_P^{(j)}.$$

# Churchsche These

Jede partielle Funktion $f\colon \mathbb{N}^j \to \mathbb{N}$, die durch irgendeinen Mechanismus oder auf Grund irgendeiner Überlegung algorithmisch berechnet werden kann, ist bereits berechenbar (durch ein $while$-Programm).

# Programme als Eingaben für Programme

- Verhalten eines Programms hängt vom eingegebenen Programm ab.



# Repräsentation von *while*-Programmen als natürliche Zahlen (Gödelnummerierung)

1. Umwandlung der Zeichen, aus denen *while*-Progamme bestehen, in Bitmuster
   - Der Zeichensatz $A$ von PASCALchen besteht aus aus 22 Zeichen.
   - Fixiere eine injektive Abbildung $code\colon A \to \{0,1\}^*$ mit $length(code(a)) = 6$ und $head(code(a)) = 1$ für alle $a \in A$. (Das ist möglich, da es 32 Bitmuster der Länge 5 gibt.)

2. Repräsentation von *while*-Progammen als Bitmuster

   $code^*: A^* \longrightarrow \{0,1\}^*$ 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*-Progammen in natürliche Zahlen

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

# Injektivität der Gödelnummerierung

**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 $decode\colon \{0,1\}^* \to A^*$ definiert durch $decode(B) = \lambda$ für alle $B$ kürzer als 6 und

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

# 3. Aufzählen von *while*-Programmen

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

$$AUFZ\ddot{A}HLUNG(n) = decode(B(n))$$

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

$$\begin{aligned} AUFZ\ddot{A}HLUNG(n) = \ & begin \\ & \quad X1 := 0;\ X2 := 1; \\ & \quad while\ X1 \neq X2\ do\ X1 := X1 \\ & end \end{aligned}$$

**Lemma**

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

**Schreibweisen:**

- $P_n = AUFZÄHLUNG(n)$: *while*-Programm mit Index $n$
- $SEM_i = SEM^{(1)}_{P_i}$

# Aufzählbarkeit

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

# Berechenbare Funktionen bzw. *while*-Programme sind aufzählbar

$$
\begin{array}{cccccc}
0 & 1 & 2 & \dots & n & \dots \\
\downarrow & \downarrow & \downarrow & & \downarrow & \\
P_0 & P_1 & P_2 & \dots & P_n & \dots \\
\downarrow & \downarrow & \downarrow & & \downarrow & \\
SEM_0 & SEM_1 & SEM_2 & \dots & SEM_n & \dots
\end{array}
$$

# Effektivität von $AUFZ\ddot{A}HLUNG$


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 $f : \mathbb{N} \to \mathbb{N}$, die nicht berechenbar ist.

# Das Halteproblem

- Eingabe: Ein *while*-Programm $P$ und eine Eingabe $a$ für $P$
- Ausgabe: $\begin{array}{ll} 1, & \text{falls } P \text{ mit der Eingabe } a \text{ hält} \\ 0, & \text{falls } P \text{ mit der Eingabe } a \text{ nicht hält} \end{array}$

# Unlösbarkeit des speziellen Halteproblems

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

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

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

ist nicht berechenbar.