- Endliche Automaten erkennen nicht alle algorithmisch erkennbaren Sprachen.
- Kontextfreie Grammatiken erzeugen nicht alle algorithmisch erzeugbaren Sprachen.

- Welche Berechnungsmodelle erlauben die Berechnung aller algorithmisch beschreibbaren Sprachen?
- Was können allgemeinere Berechnungsmodelle berechnen?
- Wo sind die Grenzen?
- Was heisst überhaupt Berechenbarkeit?

Berechenbarkeitsmodelle

- Präzisieren den Begriff der berechenbaren Funktionen
- Syntax: Sprache zum Beschreiben von Algorithmen
- Semantik: Kalkül zur Ausführung der Algorithmen

Beispiele für Berechenbarkeitsmodelle

- Spezielle Programmier- oder Spezifikationssprachen
- Turingmaschinen
- Typ-0-Grammatiken
- Registermaschinen, λ-Kalkül, μ-rekursive Funktionen

PASCALchen

- Kleine imperative Programmierspache
- Besteht aus $while$-Programmen
- Einziger Datentyp: natürliche Zahlen
- Keine Rekursion

Syntax

Zeichen in PASCALchen

- Variablen: Xn mit $n \in \mathbb{N}$
- Operatoren: $succ$, $pred$, 0
- Zuweisungssymbol: $:=$
- Schlüsselwörter: $begin$, end, $while$, do
- Hilfssymbole: $($, $)$, ;

- Ein *while*-Programm ist eine Reihung.

$$< prog > ::= < comp >$$

- Eine Reihung hat die Form $begin\ S_1; S_2; \ldots; S_m\ end$, wobei $S_1, \ldots, S_m$ für $m \geq 0$ Anweisungen sind.

$$< comp > ::= begin\ < stmtlist >\ end \mid begin\ end$$
$$< stmtlist > ::= < stmt > \mid < stmt >; < stmtlist >$$

- Eine Anweisung ist eine Reihung, eine Zuweisung oder eine Wiederholung

$$<stmt> ::= <comp> \mid <assign> \mid <while>$$

- Eine Zuweisung hat die Form $Xi := 0$, oder $Xi := succ(Xj)$ oder $Xi := pred(Xj)$, wobei Xi, Xj zwei Variablen sind.

$$<assign> ::= <var> := <expr>$$
$$<expr> ::= 0 \mid \mathrm{succ}(<var>) \mid \mathrm{pred}(<var>)$$

- Eine Wiederholung hat die Form $while\ Xi \neq Xj\ do\ S$, wobei S eine Anweisung ist und Xi, Xj zwei Variablen.

$$<while> ::= while\ <var> \neq <var>\ do\ <stmt>$$

- Variablen haben die Form Xn mit $n \in \mathbb{N}$.

$$<var> ::= X<nat>$$
$$<nat> ::= 0 \mid <one-nine><cipherseq>$$
$$<cipherseq> ::= \lambda \mid <cipher><cipherseq>$$
$$<cipher> ::= 0 \mid <one-nine>$$
$$<one-nine> ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Beschreibung der PASCALchen-Syntax mit kontextfreien Regeln

$$
\begin{aligned}
&<prog> ::= <comp> \\
&<comp> ::= \text{begin } <stmtlist> \text{ end} \mid \text{begin end} \\
&<stmtlist> ::= <stmt> \mid <stmt>; <stmtlist> \\
&<stmt> ::= <comp> \mid <assign> \mid <while> \\
&<assign> ::= <var> := <expr> \\
&<while> ::= \text{while } <var> \neq <var> \text{ do } <stmt> \\
&<expr> ::= 0 \mid \text{succ}(<var>) \mid \text{pred}(<var>)
\end{aligned}
$$

$$
\begin{aligned}
&<var> ::= X <nat> \\
&<nat> ::= \mid <one-nine><cipherseq> \\
&<cipherseq> ::= \lambda \mid <cipher><cipherseq> \\
&<cipher> ::= 0 \mid <one-nine> \\
&<one-nine> ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Beispiel

$$
\begin{array}{l}
begin \\
\quad X1 := succ(X2); \\
\quad X1 := pred(X1) \\
end
\end{array}
$$

Weist der Variable $X1$ den Wert von $X2$ zu.

Makros

while-Programme sind Anweisungen, die in anderen *while*-Programmen verwendet werden können.

$$\left.\begin{array}{l} begin \\ \quad X1 := succ(X2); \\ \quad X1 := pred(X1) \\ end \end{array}\right\} X1 := X2$$

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

$$
\left.
\begin{array}{l}
begin \\
\quad Z := 0; V := 0; \\
\quad while\ V \neq Y\ do\ begin \\
\quad\quad Z := Z + X; \\
\quad\quad V := succ(V) \\
\quad end \\
end
\end{array}
\right\} Z := X * Y
$$

Beachte: Wir benennen Variablen durch beliebige Großbuchstaben, wenn die spezielle Form Xn nicht gebraucht wird.

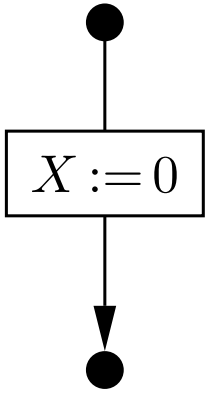
Weitere mögliche Makros

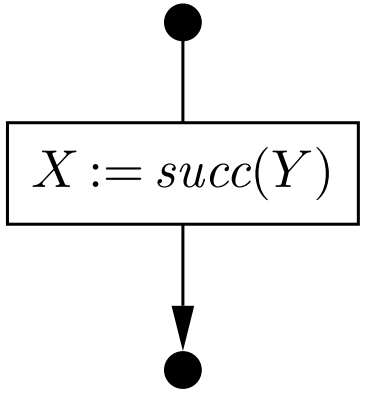
- $X1 := n$,
- $X1 := X2 \dot{-} X3$,
- $X1 := 2^{X2}$,
- $\mathit{while}\ B\ \mathit{do}\ S$,
- $\mathit{if}\ B\ \mathit{then}\ S_1\ \mathit{else}\ S_2$,
- $\mathit{repeat}\ S\ \mathit{until}\ B$ . . .

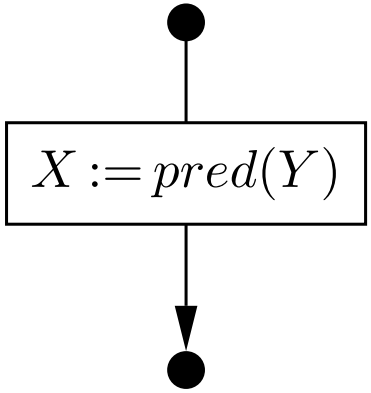
mit B: Boolescher Ausdruck, der aus Variablen, Konstanten, $=$, $\neq$, $<$, and, or, not aufgebaut ist.

Flussdiagramme

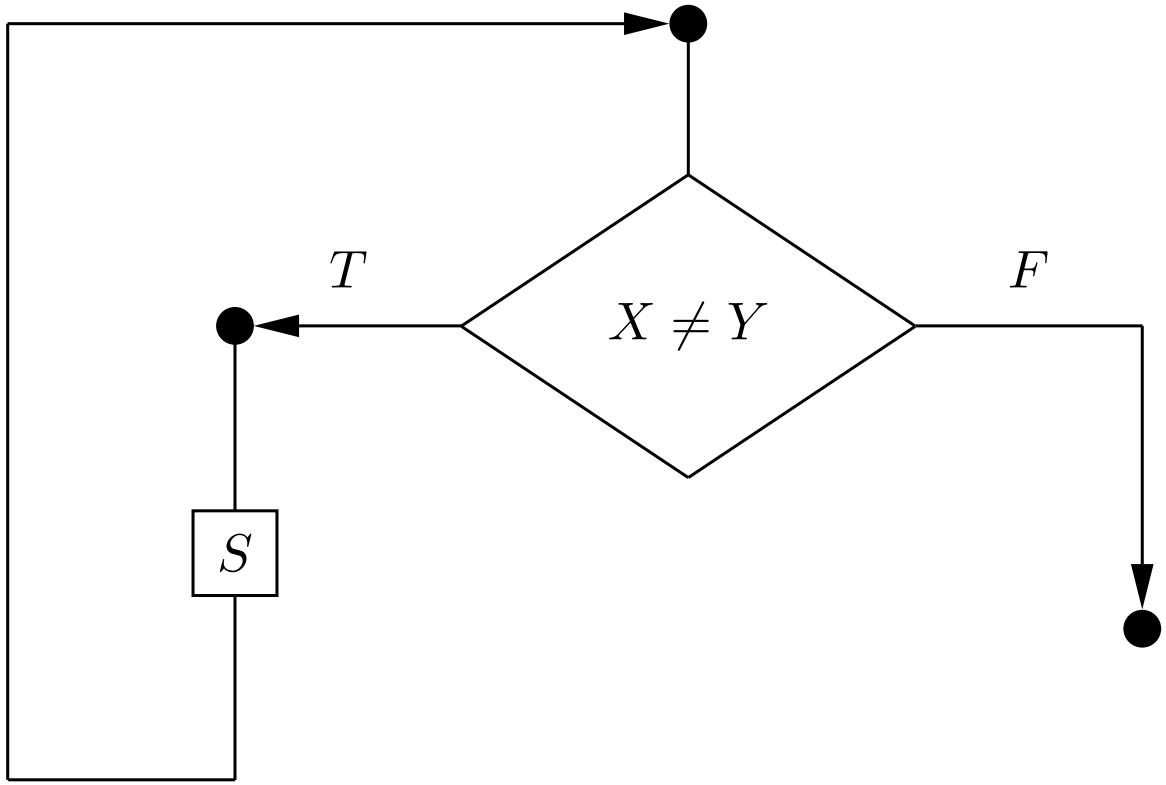
- Zuweisungen

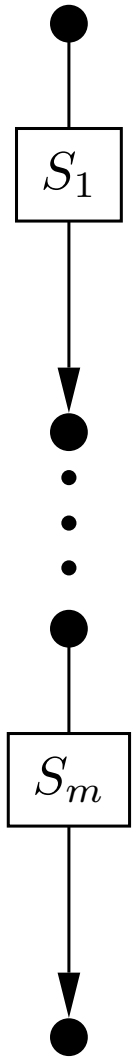






- Wiederholung und Reihung





Semantik (operationell)

Ein *while*-Programm wird k-variabel genannt, wenn höchstens die Variablen $X1, \ldots, Xk$ darin vorkommen.

Ein Berechnungszustand eines k-variablen *while*-Programms ist ein k-dimensionaler Vektor über den natürlichen Zahlen

$$a = (x_1, \ldots, x_k) \in \mathbb{N}^k,$$

der auch Zustand oder Zustandsvektor genannt wird.

Berechnung

$$a_0 A_1 a_1 A_2 a_2 \cdots a_{i-1} A_i a_i A_{i+1} a_{i+1} \cdots$$

- a_i: Berechnungszustände
- a_0 wird Eingabe genannt
- A_i: Instruktionen (d.h. Tests oder Zuweisungen)
- die im Folgenden genannten Bedingungen müssen erfüllt sein.

Bedingungen für $a_0 A_1 a_1 A_2 a_2 \cdots$ (1)

- Es gibt einen Weg durch das zugehörige Flussdiagramm, der beim Eingang beginnt und genau die Instruktionen $A_1, A_2, \ldots, A_i, \ldots$ in dieser Reihenfolge durchläuft.
- a_0 ist frei wählbar.
- $A_i = Xu \neq Xv \Longrightarrow a_i = a_{i-1}$.

Bedingungen für $a_0 A_1 a_1 A_2 a_2 \cdots$ (2)

$$A_i = \left\{ \begin{array}{l} Xu := 0 \\ Xu := succ(Xv) \\ Xu := pred(Xv) \end{array} \right\} \text{ und } a_{i-1} = (x_1, \ldots, x_k)$$

$$\Longrightarrow$$

$$a_i = (x_1, \ldots, x_{u-1}, \left\{ \begin{array}{c} 0 \\ succ(Xv) \\ pred(Xv) \end{array} \right\}, x_{u+1}, \ldots, x_k).$$

Bedingungen für $a_0A_1a_1A_2a_2\cdots$ (3)

- A_i: letzte Instruktion der Berechnung $\Longrightarrow$
  hinter A_i ist der Ausgang.
- $A_i = Xu \neq Xv$ und $a_{i-1} = (x_1, \ldots, x_k)$ $\Longrightarrow$
  - Falls $x_u \neq x_v$, ist A_{i+1} die erste Instruktion nach der herausführenden T-Kante.
  - Falls $x_u = x_v$, ist A_i die letzte Instruktion der Berechnung, oder A_{i+1} ist die erste Instruktion nach der herausführenden F-Kante.

Endliche Berechnung

$$a_0 A_1 a_1 \cdots a_{n-1} A_n a_n \qquad (n \geq 0)$$

- n: Länge der Berechnung
- a_n: Ausgabe
- $n = 0 \Longrightarrow$ Flussdiagramm enthält keine Instruktion

Beobachtung

Zu jedem k-variablen $while$-Programm und jedem Berechnungszustand a_0 gibt es genau eine Berechnung mit a_0 als Anfang.

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

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

Eine Berechnung von P

$(0, 3, 2, 7)$ $X4 := 0$
$(0, 3, 2, 0)$ $X1 := 0$
$(0, 3, 2, 0)$ $X4 \neq X3$
$(0, 3, 2, 0)$ $X1 := X1 + X2$
$(3, 3, 2, 0)$ $X4 := succ(X4)$
$(3, 3, 2, 1)$ $X4 \neq X3$
$(3, 3, 2, 1)$ $X1 := X1 + X2$
$(6, 3, 2, 1)$ $X4 := succ(X4)$
$(6, 3, 2, 2)$ $X4 \neq X3$
$(6, 3, 2, 2)$

Semantikfunktion für $while$-Programme

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

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

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

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

Churchsche These

Jede partielle Funktion $f : \mathbb{N}^j \rightarrow \mathbb{N}$, 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

$code^*\colon A^* \to \{0,1\}^*$ 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:

$$AUFZÄHLUNG(n) = decode(B(n))$$

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

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

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 $num\colon \mathbb{N} \to M$ gibt.
- M heißt effektiv aufzählbar (rekursiv aufzählbar), wenn diese Nummerierung durch einen Algorithmus vorgenommen wird.

Berechenbare Funktionen bzw. *while*-Programme sind aufzählbar

0	1	2	$\ldots$	n	$\ldots$
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	
P_0	P_1	P_2	$\ldots$	P_n	$\ldots$
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	
SEM_0	SEM_1	SEM_2	$\ldots$	SEM_n	$\ldots$

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

Das Halteproblem

- Eingabe: Ein *while*-Programm P und eine Eingabe a für P
- Ausgabe: 1, falls P mit der Eingabe a hält
  0, falls P mit der Eingabe a nicht hält

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 $HALTEPROBLEM : \mathbb{N} \longrightarrow \mathbb{N}$, die definiert ist für alle $i \in \mathbb{N}$ durch

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

ist nicht berechenbar.

Unlösbarkeit des allgemeinen Halteproblems

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

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

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

ist nicht berechenbar.

Beweisskizze

- Annahme: $HALTEPROBLEM2$ wird durch HALT2 berechnet.
- Dann berechnet das folgende Programm das spezielle Halteproblem:

$$begin\ X2 := X1;\ HALT2\ end$$

(Widerspruch, da das spezielle Halteproblem nicht berechenbar ist.)

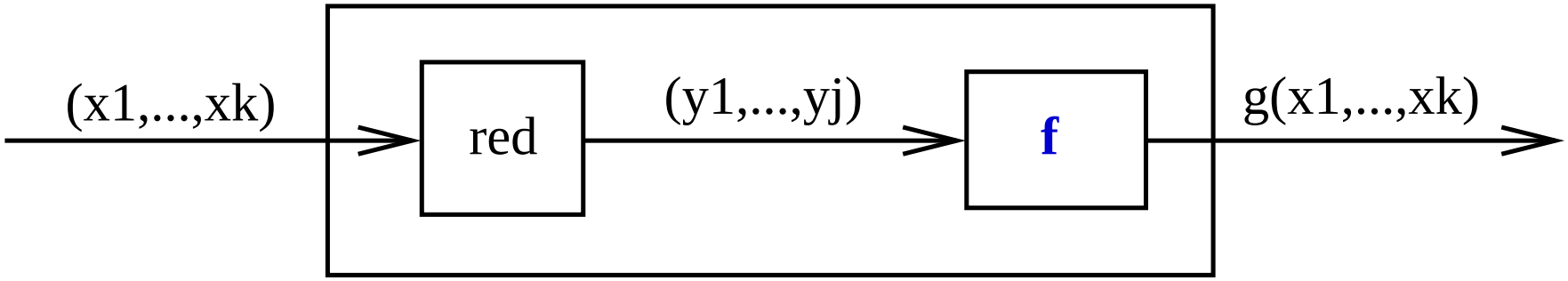
Beweise mittels Reduktion (Skizze)

Behauptung

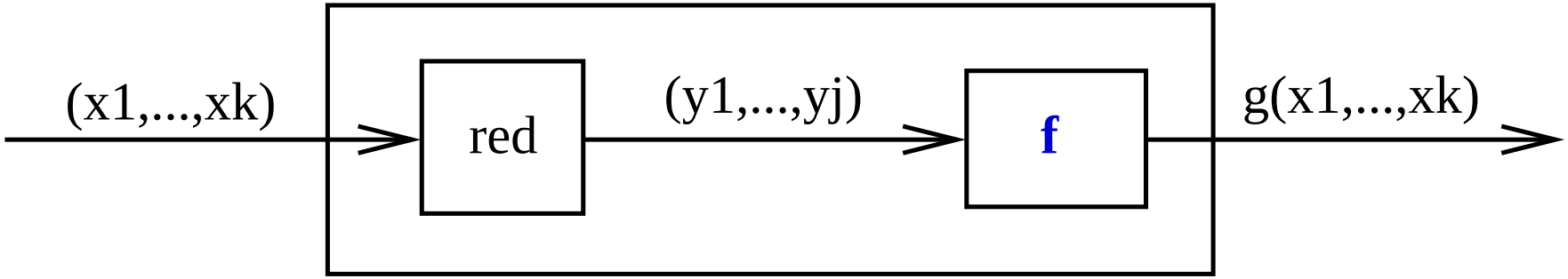
Die Funktion $f : \mathbb{N}^j \longrightarrow \mathbb{N}$ ist nicht berechenbar.

Beweisskizze

Annahme: f ist berechenbar.
Wähle eine nicht berechenbare Funktion g: $\mathbb{N}^k \longrightarrow \mathbb{N}$ und reduziere g auf f:



Gesucht ist somit eine totale berechenbare Funktion red: $\mathbb{N}^k \longrightarrow \mathbb{N}^j$, so dass $f(red(i)) = g(i)$ für alle $i \in \mathbb{N}^k$. (Ein bisschen gemogelt, da bei uns berechenbare Funktionen keine Tupel ausgeben können; ist aber leicht zu beheben.)



Wenn man g auf f reduziert, wird g berechenbar.
(Widerspruch, da g nicht berechenbar ist.)

Beispiel

Behauptung Die Funktion $HALTEPROBLEM2$ ist nicht berechenbar.

Beweis (Reduktion von $HALTEPROBLEM$ auf $HALTEPROBLEM2$)

Sei $red\colon \mathbb{N} \to \mathbb{N}^2$ definiert durch $red(i) = (i, i)$.

Dann ist red berechenbar durch

$$begin\ X2 := X1\ end$$

und es gilt

$$HALTEPROBLEM2(red(i)) =$$
$$HALTEPROBLEM2(i, i) =$$
$$HALTEPROBLEM(i)$$

Weitere unlösbare Probleme

Das Korrektheitsproblem

Sei f eine berechenbare Funktion.

- Eingabe: Index i eines $while$-Programms.
- Ausgabe: 1, falls $SEM_i = f$.
  0 sonst.

Das Korrektheitsproblem stellt die Frage, ob ein Programm eine bestimmte Funktion berechnet; es ist nicht lösbar.

Das Äquivalenzproblem

- Eingabe: $i, j \in \mathbb{N}$
- Ausgabe: 1, falls $SEM_i = SEM_j$.
  0, sonst.

Das Äquivalenzproblem stellt die Frage, ob 2 Programme dieselbe Semantik haben; es ist nicht lösbar.

Der Satz von Rice

Sei M die Menge der berechenbaren Funktionen.
Sei $E \subset M$ mit $\emptyset \neq E$.

Das Problem, ob ein Programm eine Funktion aus E berechnet, ist nicht lösbar, d.h.:

Die Funktion $check_E : \mathbb{N} \longrightarrow \mathbb{N}$ mit

$$check_E(i) = \begin{cases} 1 & \text{falls } SEM_i \in E \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

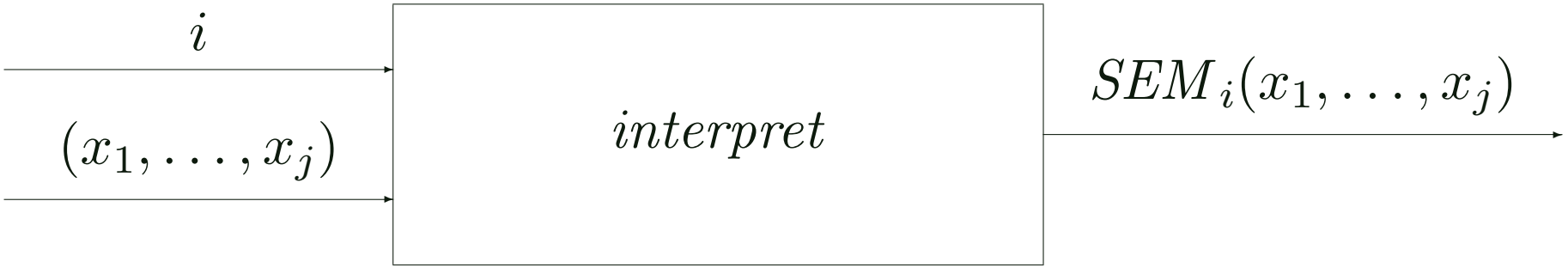
Beispiele

- $E = \{f \in M \mid f(n) = 3 \text{ für alle } n \in \mathbb{N}\}$
  ($check_E(i) = 1$ gdw. P_i immer den Wert 3 ausgibt.)
- $E = \{f \in M \mid f(n) = n \text{ für alle } n \in \mathbb{N}\}$
  ($check_E(i) = 1$ gdw. P_i immer die Eingabe ausgibt.)
- $E = \{f \in M \mid f \text{ ist total}\}$
  ($check_E(i) = 1$ gdw. P_i für alle Eingaben anhält.)
- $E = \{f \in M \mid f(n) \in \mathbb{N}\}$ mit $n \in \mathbb{N}$
  ($check_E(i) = 1$ gdw. P_i für die Eingabe n anhält.)
- $E = \{f \in M \mid f(n) = g(n) \text{ für alle } n \in \mathbb{N}\}$ mit $g \in M$
  ($check_E$ ist das Korrektheitsproblem.)

Existenz eines universellen *while*-Programms — ein lösbares Problem

Universelle Funktion

- Eingabe:
  - Index i eines $while$-Programms
  - $(x_1, \ldots, x_j) \in \mathbb{N}^j$
- Ausgabe: $SEM_i(x_1, \ldots, x_j)$.



Berechenbarkeit der universellen Funktion

Theorem

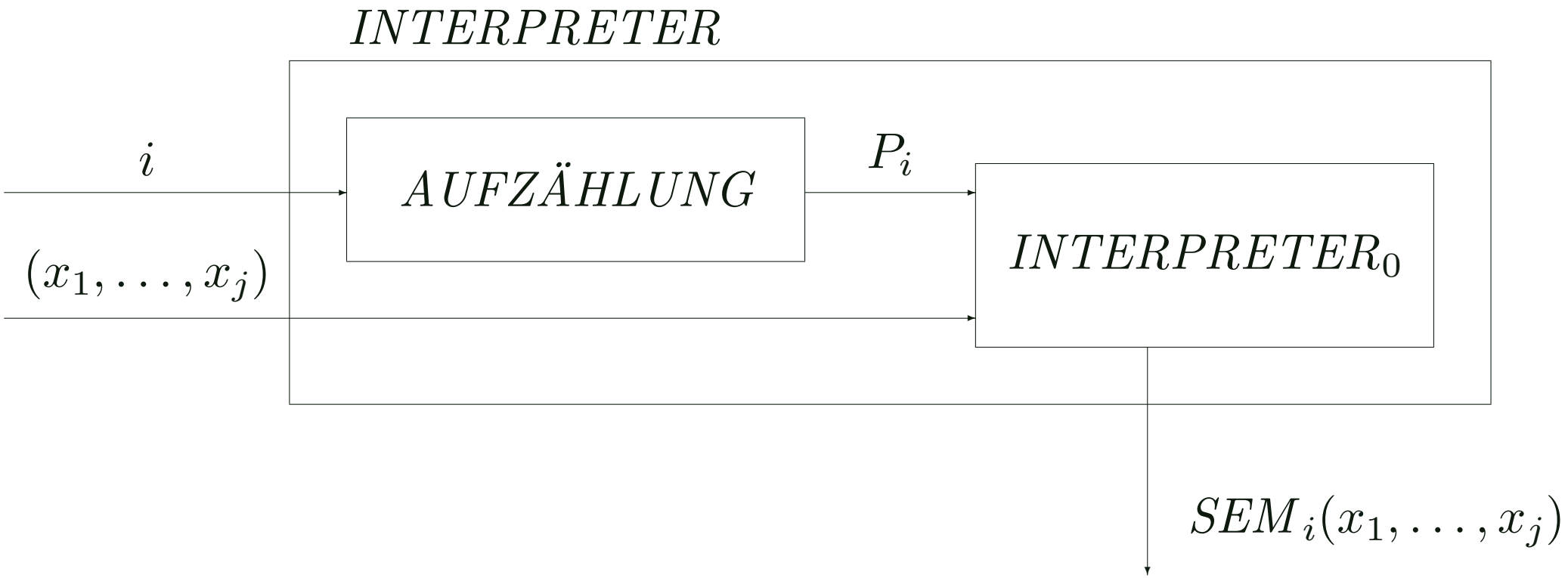
Die partielle Funktion $interpret\colon \mathbb{N}^{j+1} \longrightarrow \mathbb{N}$, die für alle $j \in \mathbb{N}$ definiert ist durch

$$interpret(i, x_1, \ldots, x_j) = SEM_i(x_1, \ldots, x_j),$$

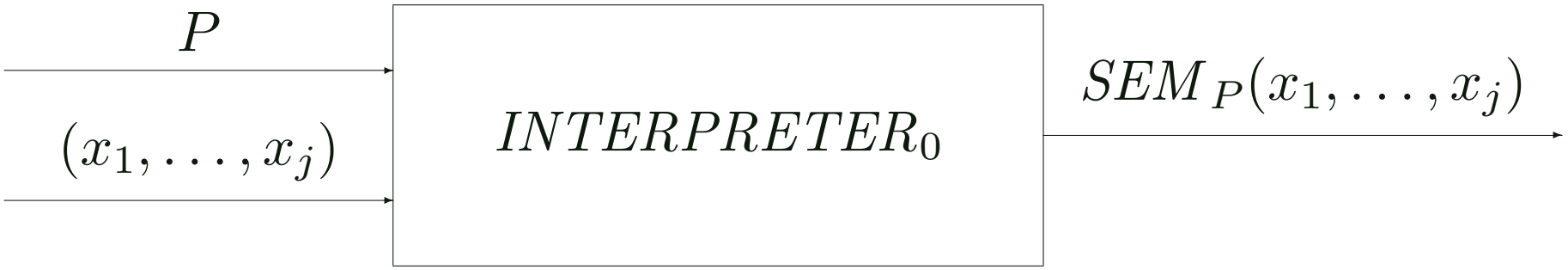
ist berechenbar.

Beachte: Gemäß der Churchschen These ist $interpret$ berechenbar, falls es einen Algorithmus für $interpret$ gibt.

PASCALchen-Interpreter als universelle Funktion



Algorithmus für $INTERPRETER_0$



1. Wandle das k-variable Programm P in sein Flussdiagramm um.
2. Wandle $(x_1, \ldots, x_j)$ in eine Eingabe a für P um.
3. Berechne P für Eingabe a.
4. Gib bei Termination den Wert von $X1$ aus.