1.6 Die Ackermann–Funktion

Ackermann gab 1925 eine Funktion an, die intuitiv berechenbar (also WHILE–berechenbar), jedoch nicht primitiv rekursiv (also LOOP–berechenbar) ist.

$$\begin{aligned} ack(0,y) &= y+1 \\ ack(x,0) &= ack(x-1,1) \\ ack(x,y) &= ack(x-1,ack(x,y-1)) \end{aligned}$$

z.B.:

$$ack(x,y) = \underbrace{ack(x-1,ack(x-1,\ldots ack(x-1,1)\ldots)}_{y-mal}$$

Betrachte eine ähnlich definierte Funktion a:

$$\begin{aligned} a(0,y) &= a(x,0)=1 \\ a(1,y) &= 3y+1 \\ a(x,y) &= \underbrace{a(x-1,a(x-1,\ldots a(x-1,y)\ldots))}_{y-mal} \end{aligned}$$

- a ist total definiert (vollständige Induktion)

MODULA-Prozedur für die berechnung von a:

```
PROCEDURE a(x,y:CARDINAL):CARDINAL;
VAR i,s : CARDINAL;
BEGIN
  IF (x = 0) OR (y = 0) THEN RETURN 1
  ELSIF x = 1 THEN RETURN 3 * y + 1
  ELSE s := y
    FOR i := 1 TO y DO
      s = a(x - 1, s)
    END;
    RETURN s
  END;
END a;
```

a ist nicht LOOP–berechenbar, denn

Sei P ein LOOP–Programm.
Ordne P die Funktion $f_P : \mathbb{N} \to \mathbb{N}$ zu:
Seien $x_0, \ldots, x_n$ alle in P vorkommenden Variable.

n_i sei der Startwert von x_i.
n_i' sei der Endwert von x_i nach Ablauf von P.
$f_P(n) := max\{\sum_{i=0}^{k} n_i' \mid \sum_{i=0}^{k} n_i \leq n\}$

(größtmögliche Summe aller Variablenendwerte, falls P mit dem Startwert der Summe $\leq n$ gestartet wird.)

Lemma
Für jedes LOOP-Programm P gibt es eine Konstante k mit
$f_P(n) < a(k,n)$ für alle $n \geq k$.

Beweis.
Induktion über den Aufbau von P

- Habe P die Form $x_i := x_j \pm c$:

$$\begin{aligned} \Rightarrow f_P(n) &\leq n + n + c = 2n + c \\ \Rightarrow f_P(n) &< 3n + 1 = a(1,n) \leq a(k,n) \text{ für alle } k \geq 1,\ n \geq c \end{aligned}$$

  Wähle k:=c+1.

- Habe P die Form $P_1; P_2$:

  Nach Vorraussetzung gibt es k_1, k_2 mit

$$\begin{aligned} f_{P_1} &< a(k_1,n) \\ f_{P_2} &< a(k_2,n) \text{ für allen} \geq max\{k_1,k_2\} \end{aligned}$$

  Setze $k_3 := max\{k_1,k_2\}$

  Es gilt:

$$\begin{aligned} f_P(n) &\leq f_{P_2}(f_{P_1}(n)) \\ &\leq f_{P_2}(a(k_1,n)) \\ &< a(k_2, a(k_1,n)) \\ &\leq a(k_3, a(k_3,n)) \\ &\leq \underbrace{a(k_3, a(k_3, \ldots a(k_3,n)\ldots))}_{n-mal} \text{ falls } n \geq 2 \\ &= a(k_3+1,n) \end{aligned}$$

  Wähle k :=max$\{k_3, 2\}$

- Habe P die Form LOOP x_i DO P' END:

  Nach Induktionvorraussetzung gibt es ein k' mit

$f'_P < a(k', n)$ für alle $n \geq k'$

Es gilt:

$$\begin{aligned} f_P(n) &= \underbrace{f'_P(f'_P(\ldots f'_P(n)\ldots))}_{n_i\text{–mal}} \\ &< \underbrace{a(k', a((k', \ldots a(k', n)\ldots))}_{n_i\text{–mal}} \\ &\leq \underbrace{a(k', a(k', \ldots a(k', n)\ldots))}_{n\text{–mal}} \\ &= a(k'+1, n) \end{aligned}$$

wähle $k := k' + 1$

■

Satz.
Die Ackermann–Funktion a ist nicht LOOP–berechenbar.

Beweis.
Annahme: a ist LOOP–berechenbar

$\Rightarrow g(n) := a(n, n)$ ist LOOP–berechenbar
Sei P ein LOOP-Programm für g
$\Rightarrow g(n) \leq f_P(n)$
wähle k in P mit: $f_P(n) < a(k, n)$
Für $n = k$ gilt:
$g(k) \leq f_P(k) < a(k, k) = g(k)$ (Widerspruch) ■

- a ist auch formal berechenbar.
  WHILE–Programm für a:

  Zunächst Angabe eines Programmes zur Berechnung von a, daß mit den Stackoperationen PUSH und POP operiert:

```
INPUT(x, y);
INIT(stack);
PUSH(x, stack);
PUSH(y, stack);
WHILE size(stack) ≠ 1 DO
    y :=POP(stack);
    x :=POP(stack);
    IF (x = 0) OR (y = 0) THEN PUSH(1, stack);
```

ELSIF $x = 1$ THEN PUSH$(3 * y + 1, stack)$;
ELSE LOOP y DO PUSH$(k - 1, stack)$ END;
PUSH$(y, stack)$
END;
$result$:=POP$(stack)$;
OUTPUT$(result)$;

Man kann die einzelnen Stackoperationen durch WHILE–Programme simulieren:

Betrachte: $c : \mathbb{N} \rightarrow \mathbb{N}$ mit $c(x, y) := 2^{x+y} + x$
c ist WHILE berechenbar und injektiv;
Beispiel: Werte für c:

	0	1	2	3	4
0	1	3	6	11	20
1	2	5	10	19	36
2	4	9	18	35	68
3	8	17	34	67	132
4	16	33	66	131	260

Man benutzt c zur Kodierung von Paaren.
Man kann aus $c(x, y) = n$ das Paar (x, y) zurückgewinnen:

Sei k max. mit $2^k < n$
$\Rightarrow \quad x := n - 2^k$
$\quad\quad y := k - x$
Definiere:

$$c_1(n) := \begin{cases} x & \text{falls } n \in c(\mathbb{N}^2) \\ 0 & \text{sonst} \end{cases} ; c_2(n) := \begin{cases} y & \text{falls } n \in c(\mathbb{N}^2) \\ 0 & \text{sonst} \end{cases}$$

c_1, c_2 sind WHILE- (sogar LOOP-) berechenbar.

Nun zur Simulation des Stacks durch ein WHILE–Programm:
Sei $(n_1, \ldots, n_k)$ der Inhalt des Stacks,
n_1 das Head–Element.

Kodierung von $(n_1, \ldots, n_k)$ mit Hilfe von c:
$n := c(n1, c(n2, \ldots, c(n_k, 0) \ldots)$

INIT($stack$)	wird simuliert durch	$n := 0$
PUSH($a, stack$)	wird simuliert durch	$n := c(a, n)$
POP($stack$)	wird simuliert durch	$result := c_1(n)$
		$n := c_2(n)$
		RETURN $result$
size($stack \neq 1$)	wird simuliert durch	$c_2(n) \neq 0$

Lemma
Die Ackermannfunktion ist WHILE–berechenbar.

1.7 Entscheidbarkeit und Semi–Entscheidbarkeit

- Der Berechenbarkeitsbegriff betrifft Funktionen.
- Einführung eines entsprechenden Begriffs für Sprachen.

Definition.
$A \subseteq \Sigma^*$ heißt *entscheidbar*, falls die *charakteristische Funktion* $\chi_A : \Sigma^* \to \{0,1\}$

$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{sonst} \end{cases}$$

berechenbar ist.

$A \subseteq \Sigma^*$ heißt *semi–entscheidbar*, falls die charakteristische Funktion $\chi'_A : \Sigma^* \to \{0,1\}$

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{sonst} \end{cases}$$

berechenbar ist.

- Man kann an Stelle von $A \subseteq \Sigma^*$ auch $A \subseteq I\!N$ betrachten.
- Das Entscheidungsproplem für A ist die Frage nach einem stoppenden Algorithmus mit

- Das Semi–Entscheidungsproplem für A ist die Frage nach einem Algorithmus mit

o
Hat der Algorithmus noch nicht gestoppt, dann ist unklar ob $w \in A$ oder nicht.

Beispiel.
Das Entscheidungsproblem für die Prädikatenlogik ("Theorembeweiser").

Satz.
A ist entscheidbar $\Leftrightarrow$ sowohl A als auch $\overline{A}$ sind semi–entscheidbar.

Beweis.
($\rightarrow$): klar.
($\leftarrow$): Sei M_1 ein Semi–Entscheidungsverfahren für A.
Sei M_2 ein Semi–Entscheidungsverfahren für $\overline{A}$.
Erhalte ein Entscheidungsverfahren für A:
INPUT(x);
FOR s:=1,2,3,...DO
IF M_1 bei Eingabe x in s Schritten stoppt
THEN OUTPUT(1) END;
IF M_2 bei Eingabe x in s Schritten stoppt
THEN OUTPUT(0) END;
END

Definition.
$A \subseteq \Sigma^*$ heißt *rekursiv aufzählbar*, falls $A = \emptyset$ oder es eine totale und berechenbare Funktion $f : I\!N \rightarrow \Sigma^*$ gibt mit

$$A = \{f(0), f(1), f(2), \ldots\}$$

"f zählt A auf." (evtl. mit $f(i) = f(j)$ für $i \neq j$!)

Satz.
Eine Sprache ist rekursiv aufzählbar, genau dann wenn sie semi–entscheidbar ist.

Beweis.
($\rightarrow$): Sei A rekursiv aufzählbar mittels der Funktion f.
Erhalte ein Semi–Entscheidungsverfahren für A:
INPUT(x);
FOR $n := 0, 1, 2, 3, \ldots$DO
IF $f(n) = x$ THEN OUTPUT(1) END;
END

($\leftarrow$): Sei $A \neq \emptyset$ semi–entscheidbar mittels Algorithmus M.
Sei $a \in A$ fixiert
Definiere eine totale und berechenbare Funktion f
mit $f(I\!N)$=A mittels folgendem Algorithmus:
INPUT(n);
$*$ Interpretiere n als Kodierung $n = c(x, y)$
mit $x = c_1(n)$, $y = c_2(n)*$
$x := c_1(n)$;
$y := c_2(n)$;
IF M angesetzt auf x in y Schritten stoppt
THEN OUTPUT(x) ELSE OUTPUT(a) END;

Der Algorithmus stoppt stets und gibt nur Elemente aus A aus.

$\Rightarrow$ f ist total und berechenbar,
$f(\mathbb{N}) \subseteq A$
Noch zu zeigen: $f(\mathbb{N}) = A$, denn

Sei $b \in A$ beliebig.
$\Rightarrow$ M stoppt bei Eingabe b in s Schritten.

Betrachte: $n = c(b, s)$
$\Rightarrow$ $f(n) = b$ nach Konstruktion des Algorithmus ■

Insgesamt erhält man:

Satz.
Eine Sprache A ist entscheidbar, genau dann wenn A und $\overline{A}$ rekursiv aufzählbar sind.

Zusammenfassung:

Bisher ist die Äquivalenz der folgenden Aussagen gezeigt worden:

A ist rekursiv aufzählbar.
$\Leftrightarrow$ A ist semi–entscheidbar.
$\Leftrightarrow$ A ist vom Typ 0 (als formale Sprache).
$\Leftrightarrow$ $A = L(M)$ für eine TM M.
$\Leftrightarrow$ χ'_A ist berechenbar.
$\Leftrightarrow$ A ist Definitionsbereich einer berechenbaren Funktion.
$\Leftrightarrow$ A ist Wertebereich einer berechenbaren Funktion.

Abschließende Bemerkung zum Zusammenhang

Abzählbarkeit — rekursive Aufzählbarkeit

Definition.
A heißt *abzählbar*, falls $A = \emptyset$ oder es gibt eine totale Funktion f gibt mit

$$A = \{f(0), f(1), f(2), \ldots\}$$

- A ist rekursiv aufzählbar, falls A durch eine totale rekursive Funktion abzählbar ist.

Unterschied:
Sei A abzählbar, $A' \subseteq A \Rightarrow A'$ ist abzählbar

Beweis.
Sei A abzählbar mittels f, $a \in A'$ fixiert.
Betrachte:

$$g(n) = \begin{cases} f(n) & \text{falls } f(n) \in A' \\ a & \text{sonst} \end{cases}$$

$g(n)$ zählt A' ab, da $A' = \{g(0), g(1), \ldots\}$ ■

Sei A rekursiv abzählbar.
Es gibt Teilmengen $A'' \subseteq A$, die nicht rekursiv abzählbar sind.

Beweis.
später.