SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

13. Juni 2005

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y)$ $\forall x,y$
3. $a(x,y) < a(x,y+1)$ $\forall x,y$
4. $a(x,y-1) \leq a(x+1,y)$ $\forall x,y$
5. $a(x,y) \leq a(x+1,y)$ $\forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Definition 134

Die Ackermann-Funktion $a : \mathbb{N}_0^2 \to \mathbb{N}_0$:

$$a(x,y) := \begin{cases} y+1 & \text{falls } x = 0 \\ a(x-1,1) & \text{falls } x \geq 1, y = 0 \\ a(x-1,a(x,y-1)) & \text{falls } x,y \geq 1 \end{cases}$$

Einige Eigenschaften der Ackermann-Funktion, die man per Induktion zeigen kann:

1. $a(1,y) = y+2$, $a(2,y) = 2y+3$ für alle y
2. $y < a(x,y) \qquad \forall x,y$
3. $a(x,y) < a(x,y+1) \qquad \forall x,y$
4. $a(x,y+1) \leq a(x+1,y) \qquad \forall x,y$
5. $a(x,y) < a(x+1,y) \qquad \forall x,y$

Genauer:

		$\mathbf{y} \rightarrow$						
		0	1	2	3	4	5
	0	1	2	3	4	5	6
	1	2	3	4	5	6	7
$\mathbf{x}$	2	3	5	7	9	11	13
$\downarrow$	3	5	13	29	61	125	253
	4	13	65533	$2^{65536}-3$	$2^{2^{65536}}-3$	$2^{2^{2^{65536}}}-3$	$\ldots$
	5	65533	$\ldots$				

Lemma 135

Sei P ein LOOP-Programm mit den Variablen $x_1, \ldots, x_k$, und sei

$$f_P(n) = \max\{\sum_i n_i'; \sum_i n_i \leq n\},$$

wobei, für $i = 1, \ldots, k$, n_i' der Wert von x_i nach Beendigung von P und n_i der Startwert von x_i vor Beginn von P ist. Dann gibt es ein $t \in \mathbb{N}$, so dass

$$\forall n \in \mathbb{N}_0 : f_P(n) < a(t, n).$$

Beweis:

durch Induktion über den Aufbau von P.

Induktionsanfang:

$P =$ $x_i := x_j \pm c$, o.E. $c \in \{0, 1\}$. Es ist klar, dass

$$f_P(n) \leq 2n + 1 < 2n + 3 = a(2, n) \text{ für alle } n$$

$\Rightarrow t = 2$ tut es!

Beweis:

durch Induktion über den Aufbau von P.

Induktionsschritt:

1. Fall: $P = P_1; P_2$. Nach Induktionsannahme gibt es $k_1, k_2 \in \mathbb{N}$, so dass für $i = 1, 2$ und für alle $n \in \mathbb{N}_0$ $f_{P_i} < a(k_i, n)$.

Damit

$$
\begin{aligned}
f_P(n) &\leq f_{P_2}(f_{P_1}(n)) \\
&\leq f_{P_2}(a(k_1, n)) \\
&< a(k_2, a(k_1, n)) && \text{Setze } k_3 := \max\{k_2, k_1 - 1\}. \\
&\leq a(k_3, a(k_3 + 1, n)) && \text{Monotonie!} \\
&= a(k_3 + 1, n + 1) \\
&\leq a(k_3 + 2, n) && \text{Eigenschaft 4 der Ackermannfkt.}
\end{aligned}
$$

und $t = k_3 + 2$ tut es hier!

Beweis:

durch Induktion über den Aufbau von P.

Induktionsschritt:

2. Fall: $P =$ LOOP x_i DO Q END.

Die Abschätzung erfolgt analog zum 1. Fall und wird als Übungsaufgabe überlassen. □

Satz 136

Die Ackermann-Funktion ist nicht LOOP-berechenbar.

Beweis:

Angenommen doch. Sei P ein zugehöriges LOOP-Programm, das

$$g(n) := a(n, n)$$

berechnet.

Nach Definition von f_P gilt $g(n) \leq f_P(n)$ für alle $n \in \mathbb{N}_0$.

Wähle gemäß dem obigen Lemma t mit $f_P(\cdot) < a(t, \cdot)$ und setze $n = t$:

$$f_P(t) < a(t, t) = g(t) \leq f_P(t)$$

$\Rightarrow$ Widerspruch! $\square$

Satz 136

Die Ackermann-Funktion ist nicht LOOP-berechenbar.

Beweis:

Angenommen doch. Sei P ein zugehöriges LOOP-Programm, das

$$g(n) := a(n, n)$$

berechnet.
Nach Definition von f_P gilt $g(n) \leq f_P(n)$ für alle $n \in \mathbb{N}_0$.

Wähle gemäß dem obigen Lemma t mit $f_P(\cdot) < a(t, \cdot)$ und setze $n = t$:

$$f_P(t) < a(t, t) = g(t) \leq f_P(t)$$

$\Rightarrow$ Widerspruch! □

Satz 136

Die Ackermann-Funktion ist nicht LOOP-berechenbar.

Beweis:

Angenommen doch. Sei P ein zugehöriges LOOP-Programm, das

$$g(n) := a(n, n)$$

berechnet.
Nach Definition von f_P gilt $g(n) \leq f_P(n)$ für alle $n \in \mathbb{N}_0$.

Wähle gemäß dem obigen Lemma t mit $f_P(\cdot) < a(t, \cdot)$ und setze $n = t$:

$$f_P(t) < a(t, t) = g(t) \leq f_P(t)$$

$\Rightarrow$ Widerspruch! $\square$

1.6 μ-rekursive Funktionen

Für eine Funktion f liefert der so genannte μ-Operator das kleinste Argument, für das f gleich 0 wird (falls ein solches existiert). Der μ-Operator gestattet so z.B., vorab die Anzahl der Durchläufe einer WHILE-Schleife zu bestimmen, bis die Laufvariable gleich 0 wird.

Definition 137

Sei f eine (nicht notwendigerweise totale) $k+1$-stellige Funktion. Die durch Anwendung des μ-Operators entstehende Funktion f_μ ist definiert durch:

$$f_\mu : \mathbb{N}_0^k \to \mathbb{N}_0$$

$$(x_1, \ldots, x_k) \mapsto \begin{cases} \min\{n \in \mathbb{N}_0;\ f(n, x_1, \ldots, x_k) = 0\} & \text{falls} \\ \qquad f(m, x_1, \ldots, x_k) \text{ definiert für alle } m \leq n & \\ \perp \text{ (undefiniert)} & \text{sonst} \end{cases}$$

1.6 μ-rekursive Funktionen

Für eine Funktion f liefert der so genannte μ-Operator das kleinste Argument, für das f gleich 0 wird (falls ein solches existiert). Der μ-Operator gestattet so z.B., vorab die Anzahl der Durchläufe einer WHILE-Schleife zu bestimmen, bis die Laufvariable gleich 0 wird.

Definition 137

Sei f eine (nicht notwendigerweise totale) $k+1$-stellige Funktion. Die durch Anwendung des μ-Operators entstehende Funktion f_μ ist definiert durch:

$$f_\mu : \mathbb{N}_0^k \to \mathbb{N}_0$$

$$(x_1, \ldots, x_k) \mapsto \begin{cases} \min\{n \in \mathbb{N}_0;\ f(n, x_1, \ldots, x_k) = 0\} & \text{falls} \\ \qquad f(m, x_1, \ldots, x_k) \text{ definiert für alle } m \leq n & \\ \perp \text{ (undefiniert)} & \text{sonst} \end{cases}$$

Definition 138

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen (konstante Funktionen, Nachfolgerfunktion, Projektionen) enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion und/oder des μ-Operators gewinnen kann.

Satz 139

f μ-rekursiv $\Longleftrightarrow$ f WHILE-berechenbar.

Beweis:

Der Beweis ist elementar. □

Definition 138

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen (konstante Funktionen, Nachfolgerfunktion, Projektionen) enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion und/oder des μ-Operators gewinnen kann.

Satz 139

f μ-rekursiv $\Longleftrightarrow$ f WHILE-berechenbar.

Beweis:

Der Beweis ist elementar. □

Definition 138

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen (konstante Funktionen, Nachfolgerfunktion, Projektionen) enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion und/oder des μ-Operators gewinnen kann.

Satz 139

f μ-rekursiv $\Longleftrightarrow$ f WHILE-berechenbar.

Beweis:

Der Beweis ist elementar. □

2. Entscheidbarkeit, Halteproblem

Wir wollen nun zeigen, dass es keinen Algorithmus geben kann, der als Eingabe ein (beliebiges) Programm P und Daten x für P erhält und (für jedes solches Paar (P, x)!) entscheidet, ob P hält, wenn es mit Eingabe x gestartet wird.

Wir erinnern uns:

1. Eine Sprache A ist rekursiv gdw die charakteristische Funktion χ_A berechenbar ist.
2. Eine Sprache A ist rekursiv aufzählbar (r.a.) gdw die semi-charakteristische Funktion χ'_A berechenbar ist.

2. Entscheidbarkeit, Halteproblem

Wir wollen nun zeigen, dass es keinen Algorithmus geben kann, der als Eingabe ein (beliebiges) Programm P und Daten x für P erhält und (für jedes solches Paar (P, x)!) entscheidet, ob P hält, wenn es mit Eingabe x gestartet wird.

Wir erinnern uns:

1. Eine Sprache A ist rekursiv gdw die charakteristische Funktion χ_A berechenbar ist.
2. Eine Sprache A ist rekursiv aufzählbar (r.a.) gdw die semi-charakteristische Funktion χ'_A berechenbar ist.

2. Entscheidbarkeit, Halteproblem

Wir wollen nun zeigen, dass es keinen Algorithmus geben kann, der als Eingabe ein (beliebiges) Programm P und Daten x für P erhält und (für jedes solches Paar (P, x)!) entscheidet, ob P hält, wenn es mit Eingabe x gestartet wird.

Wir erinnern uns:

1. Eine Sprache A ist rekursiv gdw die charakteristische Funktion χ_A berechenbar ist.
2. Eine Sprache A ist rekursiv aufzählbar (r.a.) gdw die semi-charakteristische Funktion χ'_A berechenbar ist.

2. Entscheidbarkeit, Halteproblem

Wir wollen nun zeigen, dass es keinen Algorithmus geben kann, der als Eingabe ein (beliebiges) Programm P und Daten x für P erhält und (für jedes solches Paar (P, x)!) entscheidet, ob P hält, wenn es mit Eingabe x gestartet wird.

Wir erinnern uns:

1. Eine Sprache A ist rekursiv gdw die charakteristische Funktion χ_A berechenbar ist.
2. Eine Sprache A ist rekursiv aufzählbar (r.a.) gdw die semi-charakteristische Funktion χ'_A berechenbar ist.

2.1 Rekursive Aufzählbarkeit

Definition 140

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv auflistbar, falls es eine berechenbare Funktion $f : \mathbb{N}_0 \to \Sigma^*$ gibt, so dass

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

Bemerkung: Es ist nicht verlangt, dass die Auflistung in einer gewissen Reihenfolge (z.B. lexikalisch) erfolgt!

2.1 Rekursive Aufzählbarkeit

Definition 140

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv auflistbar, falls es eine berechenbare Funktion $f : \mathbb{N}_0 \to \Sigma^*$ gibt, so dass

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

Bemerkung: Es ist nicht verlangt, dass die Auflistung in einer gewissen Reihenfolge (z.B. lexikalisch) erfolgt!

Beispiel 141

Σ^* (mit $\Sigma = \{0, 1\}$) ist rekursiv auflistbar. Wir betrachten dazu etwa folgende Funktion:

alle nullstelligen Wörter $f(0) = \epsilon$

alle einstelligen Wörter
$$\begin{cases} f(1) = 0 \\ f(2) = 1 \end{cases}$$

alle zweistelligen Wörter
$$\begin{cases} f(3) = 00 \\ f(4) = 01 \\ f(5) = 10 \\ f(6) = 11 \end{cases}$$

alle dreistelligen Wörter
$$\begin{cases} f(7) = 000 \\ \quad \vdots \end{cases}$$

Beispiel 141

Eine weitere Möglichkeit, eine Funktion f anzugeben, die alle Wörter $\in \{0,1\}^*$ auflistet, ist:

$$f(n) = \text{Binärkodierung von } n+1 \text{ ohne die führende } 1$$

Also:

$$\begin{aligned} f(0) &= 1\epsilon \\ f(1) &= 10 \\ f(2) &= 11 \\ f(3) &= 100 \\ \vdots &\qquad \vdots \end{aligned}$$

Beispiel 141

$L_{TM} = \{w \in \{0,1\}^*;\ w$ ist Codierung einer TM$\}$ ist rekursiv auflistbar:

Wir listen $\{0,1\}^*$ rekursiv auf, prüfen jedes erzeugte Wort, ob es eine syntaktisch korrekte Codierung einer Turing-Maschine ist, und verwerfen es, falls nicht.

Beispiel 141

$L_{TM} = \{w \in \{0,1\}^*;\ w$ ist Codierung einer TM$\}$ ist rekursiv auflistbar:

Wir listen $\{0,1\}^*$ rekursiv auf, prüfen jedes erzeugte Wort, ob es eine syntaktisch korrekte Codierung einer Turing-Maschine ist, und verwerfen es, falls nicht.

Wir wählen stattdessen die kanonische Auflistung von $\{0,1\}^*$ und ersetzen jedes dabei erzeugte Wort, das keine korrekte Codierung darstellt, durch den Code einer Standard-TM, die $\emptyset$ akzeptiert.

Satz 142

Eine Sprache A ist genau dann rekursiv auflistbar, wenn sie rekursiv aufzählbar (semi-entscheidbar) ist.

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen zunächst „$\Rightarrow$“.

Sei $f : \mathbb{N}_0 \to \Sigma^*$ eine berechenbare Funktion, die A auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $w \in \Sigma^*$
$x := 0$
while true do
  if $w = f(x)$ **then return** („ja“); **halt fi**
  $x := x + 1$
od

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Beweis:

Wir zeigen nun „$\Leftarrow$“.

Sei P eine WHILE-Programm, das die semi-charakteristische Funktion χ'_A berechnet, und sei f eine berechenbare Funktion, die Σ^* auflistet.

Betrachte folgenden Algorithmus:

lies die Eingabe $n \in \mathbb{N}_0$
$count := -1$; $k := -1$
repeat
  $k := k + 1$
  $w := f(c_1(k))$; $m := c_2(k)$
  if P hält bei Eingabe w in genau m Schritten **then**
    $count := count + 1$
until $count = n$
return w

Hier sind c_1 und c_2 die Umkehrfunktionen einer Paarfunktion.

2.2 Halteproblem

Definition 143

Unter dem speziellen Halteproblem H_s versteht man die folgende Sprache:

$$H_s = \{w \in \{0,1\}^*;\ M_w \text{ angesetzt auf } w \text{ hält}\}$$

Hierbei ist $(M_\epsilon, M_0, M_1, \ldots)$ eine berechenbare Auflistung der Turing-Maschinen.

Wir definieren weiter

Definition 144

$$L_d = \{w \in \Sigma^*;\ M_w \text{ akzeptiert } w \text{ nicht}\}$$

2.2 Halteproblem

Definition 143

Unter dem speziellen Halteproblem H_s versteht man die folgende Sprache:

$$H_s = \{w \in \{0,1\}^*;\ M_w \text{ angesetzt auf } w \text{ hält}\}$$

Hierbei ist $(M_\epsilon, M_0, M_1, \ldots)$ eine berechenbare Auflistung der Turing-Maschinen.

Wir definieren weiter

Definition 144

$$L_d = \{w \in \Sigma^*;\ M_w \text{ akzeptiert } w \text{ nicht}\}$$

Satz 145

L_d ist nicht rekursiv aufzählbar.

Beweis:

Wäre L_d r.a., dann gäbe es ein w, so dass $L_d = L(M_w)$.
Dann gilt:

$$\begin{aligned} M_w \text{ akzeptiert } w \text{ nicht} &\Leftrightarrow w \in L_d \\ &\Leftrightarrow w \in L(M_w) \\ &\Leftrightarrow M_w \text{ akzeptiert } w \end{aligned}$$

„$\Longrightarrow$“ Widerspruch! □

Korollar 146

L_d ist nicht entscheidbar.

Satz 145

L_d ist nicht rekursiv aufzählbar.

Beweis:

Wäre L_d r.a., dann gäbe es ein w, so dass $L_d = L(M_w)$.
Dann gilt:

$$\begin{aligned} M_w \text{ akzeptiert } w \text{ nicht} &\Leftrightarrow w \in L_d \\ &\Leftrightarrow w \in L(M_w) \\ &\Leftrightarrow M_w \text{ akzeptiert } w \end{aligned}$$

„$\Longrightarrow$“ Widerspruch! □

Korollar 146

L_d ist nicht entscheidbar.

Satz 145

L_d ist nicht rekursiv aufzählbar.

Beweis:

Wäre L_d r.a., dann gäbe es ein w, so dass $L_d = L(M_w)$.
Dann gilt:

$$\begin{aligned} M_w \text{ akzeptiert } w \text{ nicht} &\Leftrightarrow w \in L_d \\ &\Leftrightarrow w \in L(M_w) \\ &\Leftrightarrow M_w \text{ akzeptiert } w \end{aligned}$$

„$\Longrightarrow$“ Widerspruch! □

Korollar 146

L_d ist nicht entscheidbar.

Satz 145

L_d ist nicht rekursiv aufzählbar.

Beweis:

Wäre L_d r.a., dann gäbe es ein w, so dass $L_d = L(M_w)$.
Dann gilt:

$$\begin{aligned} M_w \text{ akzeptiert } w \text{ nicht} &\Leftrightarrow w \in L_d \\ &\Leftrightarrow w \in L(M_w) \\ &\Leftrightarrow M_w \text{ akzeptiert } w \end{aligned}$$

„$\Longrightarrow$“ Widerspruch! □

Korollar 146

L_d ist nicht entscheidbar.