Berechenbarkeit und Komplexität

Dozent: Mathias Weller (Skript adaptiert von Rolf Niedermeier)
Betreuer: Leon Kellerhals, Vincent Froese und Philipp Zschoche
Sekretariat: Christlinde Thielcke

Viele Fleißige Tutorinnen und Tutoren

Fachmentorin: Niloofar Nazemi

TU Berlin
Fakultät IV
Fachgebiet Algorithmik und Komplexitätstheorie
https://www.akt.tu-berlin.de

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total
Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind?

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind? Ja! (Diagonalisierung)

reelle Zahlen in $[0,1]$

$r_1 \; 0{,}\,0\,0\,1\,0\,1\,2\,1\ldots$

$r_2 \; 0{,}\,1\,5\,1\,5\,7\,2\,9\ldots$

$r_3 \; 0{,}\,9\,2\,5\,1\,1\,0\,1\ldots$

$r_4 \; 0{,}\,2\,1\,0\,2\,1\,0\,5\ldots$

$\vdots$

$r = 0{,}1663\ldots$

$\Sigma_{LOOP} = \{0, 1, \ldots, 9, ;, :=, x, +, -, \mathbf{LOOP}, \mathbf{DO}, \mathbf{END}\}$

$|\Sigma_{LOOP}| = 18$

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind? Ja! (Diagonalisierung)

Theorem

Sei L eine Liste aller 1-stelligen, LOOP-berechenbaren Funktionen. Dann ist $g : \mathbb{N} \to \mathbb{N}$ mit

$$g(n) := L_n(n) + 1$$

total und nicht LOOP-berechenbar.

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total
Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind? Ja! (Diagonalisierung)

Theorem

Sei L eine Liste aller 1-stelligen, LOOP-berechenbaren Funktionen. Dann ist $g : \mathbb{N} \to \mathbb{N}$ mit

$$g(n) := L_n(n) + 1 \qquad (1)$$

total und nicht LOOP-berechenbar.

Beweis

Wäre g LOOP-berechenbar, so gäbe es ein $k \in \mathbb{N}$ mit $\underline{L_k = g}$. (2)
$\rightsquigarrow g(k) \overset{(1)}{=} \underbrace{L_k(k) + 1}_{= g} \overset{(2)}{=} g(k) + 1$

Grenzen der LOOP-Berechenbarkeit

Wissen: alle LOOP-berechenbaren Funktionen sind total

Frage: gibt es totale Funktionen die nicht LOOP-berechenbar sind? Ja! (Diagonalisierung)

Theorem

Sei L eine Liste aller 1-stelligen, LOOP-berechenbaren Funktionen. Dann ist $g : \mathbb{N} \to \mathbb{N}$ mit

$$g(n) := L_n(n) + 1$$

total und nicht LOOP-berechenbar.

Beweis

Wäre g LOOP-berechenbar, so gäbe es ein $k \in \mathbb{N}$ mit $L_k = g$.

$\leadsto g(k) = L_k(k) + 1 = g(k) + 1$

Aber: Ist g (Turing-)berechenbar?

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

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

„verallgemeinerte Exponentialfunktion"

$$\varphi(0,y) \rightsquigarrow \underbrace{2+2+\ldots+2}_{y} = 2\cdot y$$

$$\varphi(1,y) \rightsquigarrow \underbrace{2\cdot 2\cdot\ldots\cdot 2}_{y} = 2^y$$

$$\varphi(2,y) \rightsquigarrow \left.2^{2^{2^{\cdot^{\cdot^{2}}}}}\right\}y$$

$$\vdots \qquad \vdots \qquad \vdots$$

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$
\begin{aligned}
\mathrm{ack}(0, y) &:= y + 1, \\
\mathrm{ack}(x, 0) &:= \mathrm{ack}(x-1, 1), \\
\mathrm{ack}(x, y) &:= \mathrm{ack}(x-1, \mathrm{ack}(x, y-1)) \\
&= \underbrace{\mathrm{ack}(x-1, \mathrm{ack}(x-1, \mathrm{ack}(x-1, \ldots, \mathrm{ack}(x-1, 1))) \ldots)}_{(y+1)\text{ mal}}
\end{aligned}
$$

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$\begin{aligned}
\mathsf{ack}(0,y) &:= y+1,\\
\mathsf{ack}(x,0) &:= \mathsf{ack}(x-1,1),\\
\mathsf{ack}(x,y) &:= \mathsf{ack}(x-1,\mathsf{ack}(x,y-1))\\
&= \underbrace{\mathsf{ack}(x-1,\mathsf{ack}(x-1,\mathsf{ack}(x-1,\ldots,\mathsf{ack}(x-1,1)))\ldots)}_{(y+1)\text{ mal}}
\end{aligned}$$

Die Ackermannfunktion wächst extrem schnell (z.B. gilt $\mathsf{ack}(4,2) \approx 2 \cdot 10^{19728}$).

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$\begin{aligned}
\mathrm{ack}(0, y) &:= y + 1, \\
\mathrm{ack}(x, 0) &:= \mathrm{ack}(x - 1, 1), \\
\mathrm{ack}(x, y) &:= \mathrm{ack}(x - 1, \mathrm{ack}(x, y - 1)) \\
&= \underbrace{\mathrm{ack}(x - 1, \mathrm{ack}(x - 1, \mathrm{ack}(x - 1, \ldots, \mathrm{ack}(x - 1, 1)))\ldots)}_{(y+1)\text{ mal}}
\end{aligned}$$

Die Ackermannfunktion wächst extrem schnell (z.B. gilt $\mathrm{ack}(4, 2) \approx 2 \cdot 10^{19728}$).

Eine „modernisierte“ Variante:

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

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$\begin{aligned}
\mathrm{ack}(0,y) &:= y+1,\\
\mathrm{ack}(x,0) &:= \mathrm{ack}(x-1,1),\\
\mathrm{ack}(x,y) &:= \mathrm{ack}(x-1,\mathrm{ack}(x,y-1))\\
&= \underbrace{\mathrm{ack}(x-1,\mathrm{ack}(x-1,\mathrm{ack}(x-1,\dots,\mathrm{ack}(x-1,1)))\dots)}_{(y+1)\text{ mal}}
\end{aligned}$$

Die Ackermannfunktion wächst extrem schnell (z.B. gilt $\mathrm{ack}(4,2) \approx 2 \cdot 10^{19728}$).

Eine „modernisierte" Variante:

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

Beobachtung: a ist total und in beiden Argumenten monoton wachsend

Ackermannfunktion I

Die Ackermannfunktion (Variante Rósza Péter):

$$\begin{aligned}
\mathrm{ack}(0,y) &:= y+1,\\
\mathrm{ack}(x,0) &:= \mathrm{ack}(x-1,1),\\
\mathrm{ack}(x,y) &:= \mathrm{ack}(x-1,\mathrm{ack}(x,y-1))\\
&= \underbrace{\mathrm{ack}(x-1,\mathrm{ack}(x-1,\mathrm{ack}(x-1,\dots,\mathrm{ack}(x-1,1)))\dots)}_{(y+1)\text{ mal}}
\end{aligned}$$

Die Ackermannfunktion wächst extrem schnell (z.B. gilt $\mathrm{ack}(4,2) \approx 2 \cdot 10^{19728}$).

Eine „modernisierte“ Variante:

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

Beobachtung: a ist total und in beiden Argumenten monoton wachsend

Frage: können Sie zeigen, dass $a(x,y) \leq \mathrm{ack}(x,3y)$? **(*)**

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.
$\rightsquigarrow$ definieren zunächst $f_P(n)$ als die maximale Summe aller Variablenendwerte, die das Programm P erzeugen kann, wenn die initiale Belegung höchstens Summe n hat.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.
$\rightsquigarrow$ definieren zunächst $f_P(n)$ als die maximale Summe aller Variablenendwerte, die das Programm P erzeugen kann, wenn die initiale Belegung höchstens Summe n hat.

Definition

Sei P ein LOOP-Programm, welches die Variablen $x_0, x_1, \ldots, x_k$ verwendet.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.
$\rightsquigarrow$ definieren zunächst $f_P(n)$ als die maximale Summe aller Variablenendwerte, die das Programm P erzeugen kann, wenn die initiale Belegung höchstens Summe n hat.

Definition

$n_0 \quad n_1 \quad n_k \xrightarrow{P} x_i$?

Sei P ein LOOP-Programm, welches die Variablen $\underset{}{\overset{n_0 \,=}{x_0}}, \overset{n_1 \,=}{x_1}, \dots, \overset{n_k \,=}{x_k}$ verwendet.
Die i**'te Speicherüberführungsfunktion** $F_i^P : \mathbb{N}^{k+1} \to \mathbb{N}$ an der Stelle $(n_0, n_1, \dots, n_k)$ ist der Wert von x_i am Ende des Programmes P falls P mit $x_j = n_j$ für alle $0 \leq j \leq k$ gestartet wird.

Ackermannfunktion II

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Strategie: zeigen, dass a schneller wächst als jede LOOP-berechenbare Funktion.
$\rightsquigarrow$ definieren zunächst $f_P(n)$ als die maximale Summe aller Variablenendwerte, die das Programm P erzeugen kann, wenn die initiale Belegung höchstens Summe n hat.

Definition

$\boxed{n_0 + n_1 + \ldots + n_k} \leq n \xrightarrow{P} x_1 + x_2 + x_3 \ldots x_k + x_0$

Sei P ein LOOP-Programm, welches die Variablen $x_0, x_1, \ldots, x_k$ verwendet.
Die i**'te Speicherüberführungsfunktion** $F_i^P : \mathbb{N}^{k+1} \to \mathbb{N}$ an der Stelle $(n_0, n_1, \ldots, n_k)$ ist der Wert von x_i am Ende des Programmes P falls P mit $x_j = n_j$ für alle $0 \leq j \leq k$ gestartet wird.
Außerdem sei die Funktion $f_P : \mathbb{N} \to \mathbb{N}$ definiert als

$$f_P(n) := \max \left\{ \sum_{i=0}^{k} F_i^P(n_0, \ldots, n_k) \;\middle|\; \sum_{i=0}^{k} n_i \leq n \right\}.$$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: P = „$x_i := x_j \pm c$“

$\rightsquigarrow f_P(n) \leq 2n + c$

Anfang : $x_i + x_j \leq n$

~~Ende~~ Ende : $x_i = x_j \pm c$

$$
\begin{array}{r}
x_i = x_j \pm c \\
+ \; x_j \\
\hline
2x_j + c \leq 2n + c
\end{array}
$$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1$

$\ell \geq c \Rightarrow \boxed{n \geq c}$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$.

Df.

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. $\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

$\leq a(\ell, n)$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $\underline{f_P(n) < a(\ell, n)}$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“

$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. $\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P =$ „$\underline{P_1}; \underline{P_2}$“

Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\underline{\ell_1, \ell_2} \in \mathbb{N}$, sodass für alle $\underline{n} \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:

$\underline{f_{P_1}(n) < a(\ell_1, n)} \underline{\leq} a(\ell_3, n)$ und $\underline{f_{P_2}(n) < a(\ell_2, n)} \underline{\leq} a(\ell_3, n)$.

Ackermannfunktion III

Lemma

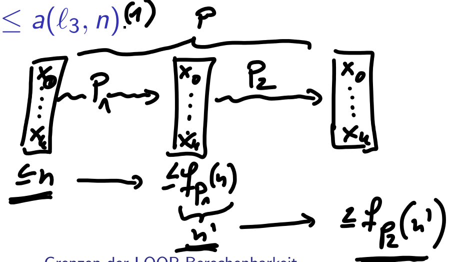
Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. $\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P =$ „$P_1; P_2$“
Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell_1, \ell_2 \in \mathbb{N}$, sodass für alle $n \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:
$f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ und $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$. (1)

Da $\underline{f_P(n) \leq f_{P_2}(f_{P_1}(n))}$ folgt (falls $\ell_3 \geq 2$):
$f_P(n) \underset{(1)}{<} a(\ell_3, \underline{f_{P_1}(n)})$



Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$.$\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P =$ „$P_1; P_2$“
Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell_1, \ell_2 \in \mathbb{N}$, sodass für alle $n \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:
(2) $f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ und $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$.

Da $f_P(n) \leq f_{P_2}(f_{P_1}(n))$ folgt (falls $\ell_3 \geq 2$):
$f_P(n) < a(\ell_3, \underline{f_{P_1}(n)}) \underset{(2)}{\leq} a(\ell_3, \underline{a(\ell_3, n)})$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$“
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$.$\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P =$ „$P_1; P_2$“
Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell_1, \ell_2 \in \mathbb{N}$, sodass für alle $n \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:
$f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ und $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$.

Da $f_P(n) \leq f_{P_2}(f_{P_1}(n))$ folgt (falls $\ell_3 \geq 2$):
$$f_P(n) < a(\ell_3, f_{P_1}(n)) \leq a(\ell_3, a(\ell_3, \underline{n})) \leq \underbrace{a(\ell_3, a(\ell_3, \overbrace{\ldots, a(\ell_3, n)}^{\geq n} \ldots)}_{n-\text{mal}}$$

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 1: $P =$ „$x_i := x_j \pm c$"
$\rightsquigarrow f_P(n) \leq 2n + c \leq 3n < 3n + 1 = a(1, n)$. $\rightsquigarrow$ Wähle $\ell := \max\{c, 1\}$.

Fall 2: $P =$ „$P_1; P_2$"
Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell_1, \ell_2 \in \mathbb{N}$, sodass für alle $n \geq \max\{\ell_1, \ell_2\} =: \ell_3$ gilt:
$f_{P_1}(n) < a(\ell_1, n) \leq a(\ell_3, n)$ und $f_{P_2}(n) < a(\ell_2, n) \leq a(\ell_3, n)$.

Da $f_P(n) \leq f_{P_2}(f_{P_1}(n))$ folgt (falls $\ell_3 \geq 2$):
$f_P(n) < a(\ell_3, f_{P_1}(n)) \leq a(\ell_3, a(\ell_3, n)) \leq \underbrace{a(\ell_3, a(\ell_3, \ldots, a(\ell_3, n) \ldots)}_{n-\text{mal}} = a(\ell_3 + 1, n)$.

Df. $\quad a(2, n)$

$\rightsquigarrow$ Wähle $\ell := \max\{\ell_3 + 1, 2\}$.

Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 3: $P =$ „**LOOP** x_i **DO** P' **END**“
Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell' \in \mathbb{N}$, sodass für alle $n \geq \ell'$ gilt: $f_{P'}(n) < a(\ell', n)$.

Ackermannfunktion III

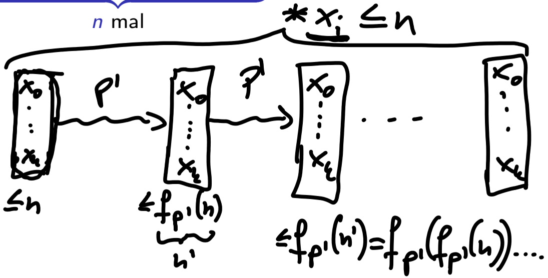
Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 3: $P =$ „**LOOP** x_i **DO** P' **END**“
Induktionsvoraussetzung $\leadsto$ es gibt $\ell' \in \mathbb{N}$, sodass für alle $n \geq \ell'$ gilt: $\boxed{f_{P'}(n) < a(\ell', n).}$

Dann gilt $f_P(n) \leq \underbrace{(f_{P'} \circ \ldots \circ f_{P'})}_{n \text{ mal}}(n) < \underbrace{a(\ell', a(\ell', \ldots, a(\ell', n) \ldots)}_{n \text{ mal}} \overset{\text{def.}}{=} a(\ell' + 1, n)$.



Ackermannfunktion III

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Beweis (Induktion über Termstruktur von P)

Fall 3: $P =$ „**LOOP** x_i **DO** P' **END**“

Induktionsvoraussetzung $\rightsquigarrow$ es gibt $\ell' \in \mathbb{N}$, sodass für alle $n \geq \ell'$ gilt: $f_{P'}(n) < a(\ell', n)$.

Dann gilt $f_P(n) \leq \underbrace{(f_{P'} \circ \ldots \circ f_{P'})}_{n \text{ mal}}(n) < \underbrace{a(\ell', a(\ell', \ldots, a(\ell', n) \ldots)}_{n \text{ mal}} = a(\ell' + 1, n)$.

$\rightsquigarrow$ Wähle $\ell := \cancel{\ell' + 1}.\ \max\{\ell'+1, 2\}$

Ackermannfunktion IV

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

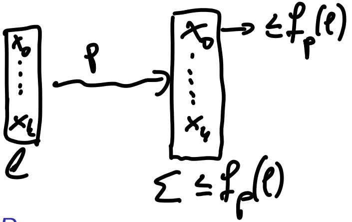
Ackermannfunktion IV

Lemma

Zu jedem LOOP-Programm P existiert ein $\ell \in \mathbb{N}$ sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$.

Theorem

Die Ackermannfunktion a ist nicht LOOP-berechenbar.



Beweis

Annahme: a LOOP-berechenbar.

$\rightsquigarrow$ $g(n) \overset{(2)}{:=} a(n, n)$ LOOP-berechenbar vermöge LOOP-Programm P.

$\rightsquigarrow$ es gibt ein $\ell \in \mathbb{N}$, sodass für alle $n \geq \ell$ gilt: $f_P(n) < a(\ell, n)$. (1)

$\rightsquigarrow$ $g(\ell) \underbrace{\leq}_{\text{df } f_P} f_P(\ell) \overset{(1)}{<} a(\ell, \ell) \overset{(2)}{=} g(\ell)$.

Bemerkung: Funktion g im Beweis wächst schneller als jede LOOP-berechenbare Funktion

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

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

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

$$a(0, y) := 1,$$
$$a(1, y) := 3y + 1,$$
$$a(x, y) := \underbrace{a(x-1, a(x-1, \ldots, a(x-1, y) \ldots)}_{y \text{ mal}}$$

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Schwierigkeit: Unbeschränkte Rekursionstiefe mit beschränkter Anzahl Variablen!

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

$$a(0,y) := 1,$$
$$a(1,y) := 3y+1,$$
$$a(x,y) := \underbrace{a(x-1, a(x-1, \ldots, a(x-1,y)\ldots)}_{y \text{ mal}}$$

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Schwierigkeit: Unbeschränkte Rekursionstiefe mit beschränkter Anzahl Variablen!

$\rightsquigarrow$ speichern mehrere Zahlen in einer Variable.

$\rightsquigarrow$ injektive, LOOP-berechenbare „Pairing-Funktion“, z.B. $c(x,y) := 2^{x+y} + x$

(Umkehrfunktionen first$(c(x,y)) := x$ und second$(c(x,y)) := y$ LOOP-berechenbar)

$$1\overbrace{00\underbrace{0\ldots0}_{\mathrm{BIN}(x)}}^{x+y}$$

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

$$a(0,y) := 1,$$
$$a(1,y) := 3y+1,$$
$$a(x,y) := \underbrace{a(x-1, a(x-1, \dots, a(x-1,y)\dots)}_{y \text{ mal}}$$

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Schwierigkeit: Unbeschränkte Rekursionstiefe mit beschränkter Anzahl Variablen!

$\leadsto$ speichern mehrere Zahlen in einer Variable.

$\leadsto$ injektive, LOOP-berechenbare „Pairing-Funktion“, z.B. $c(x,y) := 2^{x+y} + x$

(Umkehrfunktionen $\text{first}(c(x,y)) := x$ und $\text{second}(c(x,y)) := y$ LOOP-berechenbar)

$\leadsto$ Kellerinhalt $n_1, n_2, \dots, n_k$ in Zahl $n := c(n_1, c(n_2, \dots, c(n_k, 0)\dots))$ gespeichert

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

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

Beweis

Idee: Rekursion in WHILE-Schleife pressen

Schwierigkeit: Unbeschränkte Rekursionstiefe mit beschränkter Anzahl Variablen!

$\leadsto$ speichern mehrere Zahlen in einer Variable.

$\leadsto$ injektive, LOOP-berechenbare „Pairing-Funktion“, z.B. $c(x,y) := 2^{x+y} + x$

(Umkehrfunktionen $\text{first}(c(x,y))) := x$ und $\text{second}(c(x,y)) := y$ LOOP-berechenbar)

$\leadsto$ Kellerinhalt $n_1, n_2, \ldots, n_k$ in Zahl $n := c(n_1, c(n_2, \ldots, c(n_k, 0)\ldots))$ gespeichert

$$
\begin{aligned}
\text{INIT} &\leadsto n := 0\\
\text{PUSH}(x) &\leadsto n := c(x,n)\\
\text{POP} &\leadsto x := \text{first}(n);\, n := \text{second}(n);\, \textbf{return } x
\end{aligned}
$$

Ein WHILE-Programm für die Ackermannfunktion

Theorem

a ist WHILE-berechenbar.

$$a(0,y) := 1,$$
$$a(1,y) := 3y+1,$$
$$a(x,y) := \underbrace{a(x-1, a(x-1, \ldots, a(x-1, y)\ldots)}_{y \text{ mal}}$$

Beweis

```
INIT; PUSH(x); PUSH(y);
while STACK SIZE > 1 do // second(n) ≠ 0
    y ← POP;
    x ← POP;
    if x = 0 then
        PUSH(1)
    else if x = 1 then
        PUSH(3 · y + 1)
    else
        LOOP y DO PUSH(x − 1) END;
        PUSH(y)
x0 ← POP;
```

$a(4,2) = a(3, a(3,2))$

stack: 3 3 2 *

$a(3,2)$

3 2

$a(3,z)$

z'

Frage 1: wenn wir versuchen mit dem gleichen Beweis zu zeigen dass a nicht WHILE-berechenbar ist, muss dies fehlschlagen. wo genau schlägt es fehl?

Frage 2: können Sie mithilfe von Diagonalisierung zeigen, dass es nicht berechenbare Funktionen gibt? (*)

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE

Gliederung

1. Einführung
2. Berechenbarkeitsbegriff
3. LOOP-, WHILE-, und GOTO-Berechenbarkeit
4. Primitive und partielle Rekursion
5. Grenzen der LOOP-Berechenbarkeit
6. (Un-)Entscheidbarkeit, Halteproblem und Reduzierbarkeit
7. Das Postsche Korrespondenzproblem
8. Komplexität – Einführung
9. NP-Vollständigkeit
10. PSPACE