VL-10: LOOP und WHILE Programme II

(Berechenbarkeit und Komplexität, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Organisatorisches

- Nächste Vorlesung:
  Donnerstag, Dezember 6, 12:30–14:00 Uhr, Aula
  Freitag, Dezember 7: Tag der Informatik / Keine Vorlesung

- Webseite:
  http://algo.rwth-aachen.de/Lehre/WS1819/BuK.php
  ($\longrightarrow$ Arbeitsheft zur Berechenbarkeit)

Wiederholung

Wdg.: Programmiersprache LOOP

Wir betrachten eine einfache Programmiersprache namens LOOP, deren Programme aus den folgenden syntaktischen Komponenten aufgebaut sind:

Wir betrachten eine einfache Programmiersprache namens LOOP, deren Programme aus den folgenden syntaktischen Komponenten aufgebaut sind:

- Variablen: x_0 x_1 x_2 x_3 ...
- Konstanten: 0 und 1
- Symbole: $:=$ $+$ $;$
- Schlüsselwörter: LOOP DO ENDLOOP

Wdg. LOOP / Semantik

Ein LOOP Programm P mit k Variablen
berechnet eine k-stellige Funktion der Form $[P]\colon \mathbb{N}^k \to \mathbb{N}^k$.

Ist P die Zuweisung $x_i := x_j + c$,
so ist $[P](r_1, \ldots, r_k) = (r_1, \ldots, r_{i-1}, r_j + c, r_{i+1}, \ldots, r_k)$.

Ist $P = P_1; P_2$ eine Hintereinanderausführung,
so ist $[P](r_1, \ldots, r_k) = [P_2]([P_1](r_1, \ldots, r_k))$.

Ist $P =$ LOOP x_i DO Q ENDLOOP ein LOOP-Konstrukt,
so gilt $[P](r_1, \ldots, r_k) = [Q]^{r_i}(r_1, \ldots, r_k)$.

Wdg.: WHILE / Semantik

- Die Eingabe ist in den Variablen $x_1, \ldots, x_m$ enthalten.
  Alle anderen Variablen werden mit 0 initialisiert.
- Das Resultat eines WHILE Programms ist die Zahl, die sich am Ende der Abbarbeitung in der Variablen x_0 ergibt.

Das Programm WHILE $x_i \neq 0$ DO P ENDWHILE hat folgende Bedeutung:
P wird solange ausgeführt, bis x_i den Wert 0 erreicht.

Ein WHILE Programm P mit k Variablen
berechnet eine k-stellige Funktion der Form $[P] \colon \mathbb{N}^k \to \mathbb{N}^k$.

Ist $P =$ WHILE $x_i \neq 0$ DO Q ENDWHILE ein WHILE-Konstrukt,
so ist $[P](r_1, \ldots, r_k) = [Q]^\ell(r_1, \ldots, r_k)$ für die kleinste Zahl ℓ,
für die die i-te Komponente von $[Q]^\ell(r_1, \ldots, r_k)$ gleich 0 ist.
Falls solch ein ℓ nicht existiert, so ist $[P](r_1, \ldots, r_k)$ undefiniert.

Wdg.: WHILE versus LOOP

- Es gibt WHILE Programme, die nicht terminieren.
  LOOP Programme terminieren immer.
- Jede LOOP-berechenbare Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist auch WHILE-berechenbar.

Satz

Die Programmiersprache WHILE ist Turing-mächtig.

Vorlesung VL-10
LOOP und WHILE Programme II

- Die Ackermann Funktion
- Eigenschaften der Ackermann Funktion
- Ackermann Funktion und LOOP Programme

Mächtigkeit von LOOP

Vermutung von David Hilbert (1926)

Die Menge der LOOP-berechenbaren Funktionen $\mathbb{N}^k \to \mathbb{N}$
stimmt mit der Menge der berechenbaren totalen Funktionen überein.

(Als Teil von Hilberts Programm, mit "finiten" Methoden die Widerspruchsfreiheit der Axiomensysteme der Mathematik nachzuweisen.)

Satz (Ackermann, 1926/1928)

Es existieren berechenbare totale Funktionen $\mathbb{N}^k \to \mathbb{N}$, die nicht LOOP-berechenbar sind.

Wilhelm Ackermann (1896–1962)

Wikipedia: Wilhelm Friedrich Ackermann war ein deutscher Mathematiker (und ein Schüler von Hilbert).

1926 entdeckte er die nach ihm benannte Ackermann Funktion, die in der Berechenbarkeitstheorie eine wichtige Rolle spielt. Ackermann war Gymnasiallehrer an verschiedenen Schulen in Münster, Burgsteinfurt und Lüdenscheid; er hielt auch Vorlesungen an der Universität Münster.

D. Hilbert und W. Ackermann (1928): *"Grundzüge der theoretischen Logik"*

Die Ackermann Funktion

Ackermann Funktion: Definition

Definition

Die Ackermann Funktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist folgendermassen definiert:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

Wilhelm Ackermann:
"Zum Hilbertschen Aufbau der reellen Zahlen"
Mathematische Annalen 99 (1928), pp. 118–133

Rózsa Péter:
"Konstruktion nichtrekursiver Funktionen"
Mathematische Annalen 111 (1935), pp. 42–60

Ackermann Funktion: Beispiele (1)

$$
\begin{array}{lcll}
A(0,n) & = & n+1 & \text{für } n \geq 0 \\
A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\
A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0
\end{array}
$$

Ein paar Beispiele für $m = 1$:

$A(1,0) = A(0,1) = 2$

$A(1,1) = A(0,A(1,0)) = A(1,0) + 1 = 3$

$A(1,2) = A(0,A(1,1)) = A(1,1) + 1 = 4$

$A(1,3) = A(0,A(1,2)) = A(1,2) + 1 = 5$

Beobachtung

$A(1,n) = n+2$

Ackermann Funktion: Beispiele (2)

$$
\begin{array}{lcll}
A(0,n) & = & n+1 & \text{für } n \geq 0 \\
A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\
A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0
\end{array}
$$

Ein paar Beispiele für $m = 2$:

$A(2,0) = A(1,1) = 3$

$A(2,1) = A(1,A(2,0)) = A(2,0) + 2 = 5$

$A(2,2) = A(1,A(2,1)) = A(2,1) + 2 = 7$

Beobachtung

$A(2,n) = 2n + 3$

Ackermann Funktion: Beispiele (3)

$$
\begin{array}{lcll}
A(0,n) & = & n+1 & \text{für } n \geq 0 \\
A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\
A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0
\end{array}
$$

Und ein paar Beispiele für $m = 3$:

$$A(3,0) = A(2,1) = 5$$

$$A(3,1) = A(2,A(3,0)) = 2 \cdot A(3,0) + 3 = 13$$

$$A(3,2) = A(2,A(3,1)) = 2 \cdot A(3,1) + 3 = 29$$

$$A(3,3) = A(2,A(3,2)) = 2 \cdot A(3,2) + 3 = 61$$

Beobachtung

$$A(3,n) = 2^{n+3} - 3$$

Ackermann Funktion: Beispiele (4)

Zusammenfassung der Beispiele

Wenn man den ersten Parameter fixiert ...

- $A(1, n) = n + 2$
- $A(2, n) = 2n + 3$
- $A(3, n) = 2^{n+3} - 3$
- $A(4, n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3 \text{ viele Zweien}} - 3$

Bereits der Wert $A(4, 2) = 2^{65536} - 3$
ist grösser als die Anzahl aller Atome im Weltraum.

Exkurs: UpArrow-Notation (1)

- Addition ist iterierte Nachfolgerbildung:

$$\underbrace{S(S(\ldots(S(a)\ldots)}_{b \text{ mal}} = a + b$$

- Multiplikation ist iterierte Addition:

$$\underbrace{a + \cdots + a}_{b \text{ mal}} = a \cdot b$$

- Potenzierung ist iterierte Multiplikation:

$$\underbrace{a \times \cdots \times a}_{b \text{ mal}} = a^b =: a \uparrow b$$

- Der Potenzturm ist iterierte Potenzierung:

$$\underbrace{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}_{b \text{ mal}} =: a \uparrow\uparrow b$$

- Wiederholte Potenzturmbildung gibt $\underbrace{a \uparrow\uparrow a \uparrow\uparrow \ldots \uparrow\uparrow a}_{b \text{ mal}} =: a \uparrow\uparrow\uparrow b$

Exkurs: UpArrow-Notation (2)

Definition (UpArrow-Notation von Donald Knuth)

$$a \uparrow^m b := \begin{cases} 1 & \text{wenn } b = 0 \\ a \cdot b & \text{wenn } m = 0 \\ a^b & \text{wenn } m = 1 \\ a \uparrow^{m-1} (a \uparrow^m (b-1)) & \text{sonst} \end{cases}$$

Es gilt:

- $A(1, n) = 2 + (n + 3) - 3$
- $A(2, n) = 2 \cdot (n + 3) - 3$
- $A(3, n) = 2 \uparrow (n + 3) - 3$
- $A(4, n) = 2 \uparrow\uparrow (n + 3) - 3$
- $A(5, n) = 2 \uparrow\uparrow\uparrow (n + 3) - 3$

$\vdots$

- $A(m, n) = 2 \uparrow^{m-2} (n + 3) - 3 \qquad$ für $m \geq 2$

Donald Ervin Knuth (1938)

Wikipedia: Don Knuth is an American computer scientist, mathematician, and professor emeritus at Stanford University.

Knuth is the author of the multi-volume work *"The Art of Computer Programming"*, he popularized the asymptotic notation (big-O) in algorithmics, he created the TeX computer typesetting system and the METAFONT font definition language.

- Knuth-Morris-Pratt algorithm
- Knuth-Bendix completion algorithm

Berechenbarkeit der Ackermann Funktion

Die Ackermann Funktion ist berechenbar

$$
\begin{array}{lcll}
A(0,n) & = & n+1 & \text{für } n \geq 0 \\
A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\
A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0
\end{array}
$$

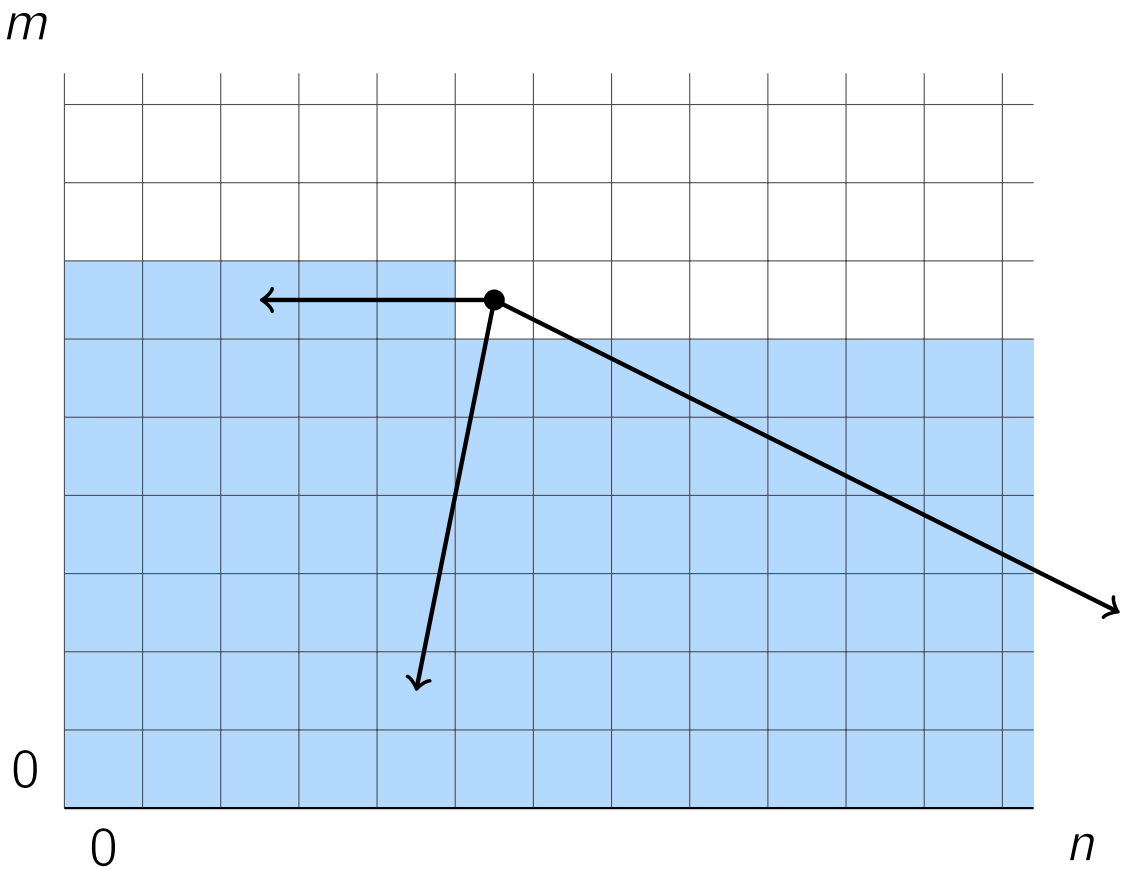
Satz

Die Ackermann Funktion ist Turing-berechenbar.

Beweis:

- Wir zeigen mit Induktion über $m \geq 0$, dass jede Funktion $f_m : \mathbb{N} \to \mathbb{N}$ mit $f_m(x) = A(m,x)$ berechenbar ist.
- Für $m = 0$ ist f_0 die Nachfolgerfunktion $f_0(x) = x + 1$.
- Für $m \geq 1$ berechnen wir $f_m(x)$, indem wir der Reihe nach induktiv alle Werte $f_m(0)$, $f_m(1)$, $f_m(2)$, $\ldots$, $f_m(x-1)$ bestimmen.
- Zum Schluss berechnen wir:
  $f_m(x) \;=\; A(m,x) \;=\; A(m-1,A(m,x-1)) \;=\; f_{m-1}(f_m(x-1))$

Induktion über lexikographisch sortierte Paare (m, n)



Monotonie-Verhalten der Ackermann Funktion

Monotonie-Verhalten

Monotonie

(M.1) $A(m, n+1) > A(m, n)$

(M.2) $A(m+1, n) > A(m, n)$

(M.3) $A(m+1, n-1) \geq A(m, n)$

Einfache Folgerung aus (M.1) und (M.2)

Für $m \geq m'$ und $n \geq n'$ gilt immer:

$$A(m, n) \geq A(m', n')$$

Monotonie-Beweis (M.1)

$$
\begin{array}{lcll}
A(0,n) & = & n+1 & \text{für } n \geq 0 \\
A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\
A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0
\end{array}
$$

Wir wollen nun Ungleichung (M.1) zeigen: $A(m,n+1) > A(m,n)$

Die Ackermann'sche Rekursionsgleichung liefert:

$$
\begin{aligned}
A(m,n+1) &= A(m-1,A(m,n)) \\
A(m,n) &= A(m-1,A(m,n-1))
\end{aligned}
$$

Induktion liefert zuerst $x := A(m,n) > A(m,n-1) =: y$
und danach $A(m-1,x) > A(m-1,y)$.

Monotonie-Beweis (M.2)

Ein analoges induktives Argument beweist die Ungleichung (M.2).

Übung

Beweisen Sie die Ungleichung (M.2) mit vollständiger Induktion über die lexikographisch sortierten Paare (m, n).

$$A(m+1, n) \;>\; A(m, n) \qquad \text{für alle } m, n \geq 0.$$

Monotonie-Beweis (M.3)

(M.3) $A(m+1, n-1) \geq A(m, n)$

Induktionsanfang: Es gilt $A(1, n-1) = n+1 = A(0, n)$ für alle $n \geq 1$.
Es gilt $A(m+1, 0) = A(m, 1)$ für alle $m \geq 0$.

Induktionsschritt: Wir nehmen an, dass für (m', n') mit $(m' < m)$ oder $(m' = m) \wedge (n' < n)$ immer $A(m'+1, n'-1) \geq A(m', n')$ gilt.

$$
\begin{array}{lcll}
A(m+1, n-1) & = & A(m, A(m+1, n-2)) & \text{(Def)} \\
& \geq & A(m, A(m, n-1)) & \text{(Ind+Mono)} \\
& \geq & A(m-1, A(m, n-1)+1) & \text{(Ind)} \\
& \geq & A(m-1, A(m, n-1)) & \text{(Mono)} \\
& = & A(m, n) & \text{(Def)}
\end{array}
$$

Das Wachstumslemma

Wachstumslemma: Vorbereitungen (1)

Wir werden in diesem Kapitel ausschliesslich LOOP Programme betrachten, die die folgende Annahme erfüllen:

Annahme: In LOOP-Konstrukten LOOP x_i DO P ENDLOOP kommt die Zählvariable x_i nicht im inneren Programmteil P vor.

Andernfalls führen wir für die Schleife eine frische Zählvariable x_i' ein:

$x_i' := x_i$;
LOOP x_i' DO P ENDLOOP

Wachstumslemma: Vorbereitungen (2)

- Wir betrachten ein fixes LOOP Programm P mit den k Variablen $x_1, x_2, \ldots, x_k$.
- Wir erinnern uns (aus VL-09): Das LOOP Programm P berechnet die k-stellige Funktion $[P]\colon \mathbb{N}^k \to \mathbb{N}^k$. Das LOOP Programm P übersetzt also einen Eingabevektor $\vec{a} = (a_1, \ldots, a_k) \in \mathbb{N}^k$ in einen Ausgabevektor $(b_1, \ldots, b_k)$.

Definition

Für die Anfangswerte $\vec{a} = (a_1, \ldots, a_k) \in \mathbb{N}^k$ der Variablen definieren wir

$$f_P(\vec{a}) := b_1 + b_2 + \cdots + b_k$$

als die Summe aller Ergebniswerte in $[P](\vec{a}) = (b_1, \ldots, b_k)$.

Wachstumslemma: Vorbereitungen (3)

Definition

Die Wachstumsfunktion $F_P : \mathbb{N} \to \mathbb{N}$ ist definiert durch

$$F_P(n) = \max \left\{ f_P(\vec{a}) \;\middle|\; \vec{a} \in \mathbb{N}^k \text{ mit } \sum_{i=1}^{k} a_i \leq n \right\}$$

Intuitiv: Die Funktion F_P beschreibt das maximale Wachstum (die maximale Zunahme) der Variablensumme durch das LOOP Programm P.

Wachstumslemma: Zentrale Aussage

Wir werden nun zeigen,
dass für alle LOOP Programme P und für alle $n \in \mathbb{N}$
der Wert $F_P(n)$ echt kleiner ist als der Wert $A(m, n)$,
wenn nur der Parameter m gross genug gewählt wird.

Lemma (Wachstumslemma)

Für jedes LOOP Programm P existiert eine natürliche Zahl m_P,
sodass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m_P, n)$.

Man beachte:
Für jedes feste Programm P ist der Parameter m_P eine Konstante.

Beweis: Induktion über Aufbau des Programms

Induktionsanfang (Zuweisungen)

- Es sei P von der Form $x_i := x_j + c$ mit $c \in \{0, 1\}$.
- Wir werden zeigen: $F_P(n) < A(2, n)$.

Induktionsschritt (Hintereinanderausführung)

- Es sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Induktionsschritt (LOOP-Konstrukt)

- Es sei P von der Form "LOOP x_i DO Q ENDLOOP"
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Wir werden zeigen: $F_P(n) < A(q + 1, n)$.

Beweis: Induktionsanfang

Induktionsanfang (Zuweisungen)

- Es sei P von der Form "$x_i := x_j + c$" mit $c \in \{0, 1\}$
- Dann gilt: $F_P(n) \leq 2n + 1$, und somit $F_P(n) < A(2, n)$

Argument:

- P verändert nur den Wert der Variablen x_i.
- Am Anfang war $x_i \geq 0$, und am Ende ist $x_i = x_j + c \leq n + 1$.
- Die Summe aller Variablenwerte erhöht sich um höchstens $n + 1$, und springt damit von höchstens n (am Anfang) auf höchstens $2n + 1$ (am Ende).

Beweis: Schritt für Hintereinanderausführung

Induktionsschritt (Hintereinanderausführung)

- Es sei P von der Form "$P_1; P_2$"
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Dann gilt: $F_P(n) < A(q+1, n)$

Argument:

- Wir verwenden die Abschätzung (M.3): $A(q, n) \leq A(q+1, n-1)$
- Dann folgt mit den Monotonie Eigenschaften, dass

$$\begin{aligned} F_P(n) &\leq F_{P_2}(F_{P_1}(n)) && \text{(Def)} \\ &< A(q, A(q, n)) && \text{(Ind)} \\ &< A(q, A(q+1, n-1)) && \text{(Mono+M.3)} \\ &= A(q+1, n) && \text{(Def)} \end{aligned}$$

Beweis: Schritt fürs LOOP-Konstrukt (1)

Induktionsschritt (LOOP-Konstrukt)

- Es sei P von der Form "LOOP x_i DO Q ENDLOOP"
- Induktionsannahme liefert: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$
- Dann gilt: $F_P(n) \leq A(q+1, n)$.

Argument:

- Wir betrachten jenen Wert $\alpha = \alpha(n)$ für die Variable x_i,
  mit dem der grösstmögliche Funktionswert $F_P(n)$ angenommen wird
- Dann gilt $F_P(n) \leq F_Q(F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha$,
  wobei die Funktion $F_Q(\cdot)$ hier α-fach ineinander eingesetzt ist.
- Die Induktionsannahme gibt uns $F_Q(\ell) < A(q, \ell)$
- Dies wenden wir auf die äusserste Funktion F_Q an und erhalten

$$F_P(n) < A(q, F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha$$

Beweis: Schritt fürs LOOP-Konstrukt (2)

Und weiter geht's:

- Wiederholte Anwendung liefert die α-zeilige Ungleichungskette

$$\begin{aligned} F_P(n) < & \quad A(q, F_Q(, F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha \\ < & \quad A(q, A(q, F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha \\ & \quad \vdots \qquad\qquad \vdots \\ < & \quad A(q, A(q, A(q, \ldots, A(q, n-\alpha))\ldots)) + \alpha \end{aligned}$$

- Daraus ergibt sich $F_P(n) \leq A(q, A(q, A(q, \ldots, A(q, n-\alpha))\ldots))$

- Im innersten Teil verwenden wir $A(q, n-\alpha) \leq A(q+1, n-\alpha-1)$
  Ergo: $F_P(n) \leq A(q, A(q, A(q, \ldots A(q, A(q+1, n-\alpha-1)))\ldots))$

- Jetzt arbeiten wir uns $(\alpha-1)$-mal von innen nach aussen vor, und wenden in jedem Schritt die Ackermann Definition an.
- Das ergibt $F_P(n) \leq A(q+1, n-1)$ und $F_P(n) < A(q+1, n)$.

Die Mächtigkeit von LOOP

Die Mächtigkeit von LOOP (1)

Satz

Die Ackermann Funktion ist nicht LOOP-berechenbar.

Beweis:

- Zwecks Widerspruchs nehmen wir an, dass die Ackermann Funktion LOOP-berechenbar ist. Dann gibt es auch ein LOOP Programm P, das die Hilfsfunktion $B(n) = A(n, n)$ berechnet.
- Das Wachstumslemma liefert uns eine Zahl m_P, sodass für alle $n \in \mathbb{N}$ gilt: $F_P(n) < A(m_P, n)$.
- Wird nun das LOOP Programm P mit der Eingabe m_P aufgerufen, so berechnet es den Funktionswert $B(m_P)$.
- Da F_P das Wachstum des Programms P beschränkt, gilt per Definition: $B(m_P) \leq F_P(m_P)$.
- Zusammengefasst: $B(m_P) \leq F_P(m_P) < A(m_P, m_P) = B(m_P)$
- Widerspruch! QED!

Die Mächtigkeit von LOOP (2)

Wir erinnern uns: Die Ackermann Funktion ist Turing-berechenbar.

Satz

Die Ackermann Funktion ist WHILE-berechenbar.

Wir fassen zusammen:

Folgerung

Die Menge der LOOP-berechenbaren Funktionen
bildet eine echte Teilmenge der berechenbaren Funktionen.

Ergo: Die Programmiersprache LOOP ist nicht Turing-mächtig

Landschaft um LOOP, WHILE, TM, RAM