

Theorie der Informatik

16. Ackermannfunktion

Malte Helmert Gabriele Röger

Universität Basel

29. April 2015

Theorie der Informatik

29. April 2015 — 16. Ackermannfunktion

16.1 Einleitung

16.2 Ackermannfunktion

16.3 Zusammenfassung

Überblick: Vorlesung

Vorlesungsteile

- I. Logik ✓
- II. Automatentheorie und formale Sprachen ✓
- III. Berechenbarkeitstheorie
- IV. Komplexitätstheorie

Überblick: Berechenbarkeitstheorie

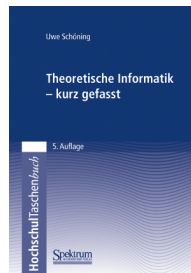
III. Berechenbarkeitstheorie

- 13. Turing-Berechenbarkeit ✓
- 14. LOOP-, WHILE- und GOTO-Berechenbarkeit ✓
- 15. primitive Rekursion und μ -Rekursion ✓
- 16. Ackermannfunktion
- 17. Entscheidbarkeit, Reduktionen, Halteproblem
- 18. Postsches Korrespondenzproblem
 - ~~Unentscheidbare Grammatik-Probleme~~
 - ~~Gödelscher Satz und diophantische Gleichungen~~

Literatur zu diesem Vorlesungskapitel

Theoretische Informatik - kurz gefasst
von Uwe Schöning (5. Auflage)

- ▶ Kapitel 2.5



Aus Zeitgründen kürzen wir das Thema stark ab.

16.1 Einleitung

Formale Berechnungsmodelle

Formale Berechnungsmodelle

- ▶ Turingmaschinen
- ▶ LOOP-, WHILE-, GOTO-Programme
- ▶ primitiv rekursive Funktionen, μ -rekursive Funktionen

Wir wissen jetzt, dass es **zwei Klassen**
von verschiedenen mächtigen Berechnungsmodellen gibt:

- ▶ LOOP-Programme und primitiv rekursive Funktionen
- ▶ Turingmaschinen, WHILE-/GOTO-Programme,
 μ -rekursive Funktionen

Gibt es einen praktischen Unterschied?

- ▶ Wir haben gezeigt, dass LOOP-Programme
(und primitiv rekursive Funktionen)
echt weniger mächtig sind als die anderen Formalismen.
- ▶ Die **Beispiele** hierfür waren aber wenig praktisch relevant,
da unser Argument nur darauf abzielt,
dass LOOP-berechenbare Funktionen immer **total** sind.
- ▶ Bei jeder Eingabe zu terminieren, ist in der Praxis
nicht unbedingt ein Problem. (Im Gegenteil.)
- ▶ Gibt es auch **totale** Funktionen,
die nicht LOOP-berechenbar sind?

16.2 Ackermannfunktion

Ackermannfunktion: Geschichte

- ▶ **David Hilbert** vermutet, dass **alle berechenbaren** totalen Funktionen primitiv rekursiv sind (1926).
- ▶ **Wilhelm Ackermann** widerlegt die Vermutung durch Angabe eines Gegenbeispiels (1928).
- ▶ Gegenbeispiel vereinfacht von **Rózsa Péter** (1935)
- ▶ **hier**: vereinfachte Version

Ackermannfunktion

Definition (Ackermannfunktion)

Die **Ackermannfunktion** $a : \mathbb{N}_0^2 \rightarrow \mathbb{N}_0$ ist wie folgt definiert:

$$\begin{aligned} a(0, y) &= y + 1 && \text{für alle } y \geq 0 \\ a(x, 0) &= a(x - 1, 1) && \text{für alle } x > 0 \\ a(x, y) &= a(x - 1, a(x, y - 1)) && \text{für alle } x, y > 0 \end{aligned}$$

Anmerkung: die Rekursion in der Definition ist endlich, so dass eine totale Funktion definiert wird. (**Warum?**)

Wertetabelle

	$y = 0$	$y = 1$	$y = 2$	$y = 3$	$y = k$
$a(0, y)$	1	2	3	4	$k + 1$
$a(1, y)$	2	3	4	5	$k + 2$
$a(2, y)$	3	5	7	9	$2k + 3$
$a(3, y)$	5	13	29	61	$2^{k+3} - 3$
$a(4, y)$	13	65533	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{k+3} - 3$

Berechenbarkeit der Ackermannfunktion

Satz

Die Ackermannfunktion ist WHILE-berechenbar, aber nicht LOOP-berechenbar.

(Ohne Beweis.)

Berechenbarkeit der Ackermannfunktion: Beweisidee

Beweisidee:

► WHILE-Berechenbarkeit:

- zeige, wie WHILE-Programme einen Stack simulieren können (im Wesentlichen: push/pop mit *encode/decode*)
- beseitige Rekursion durch Verwendung eines Stacks
~> WHILE-Programm einfach anzugeben

► keine LOOP-Berechenbarkeit:

- zeige, dass es für jedes LOOP-Programm eine Zahl k gibt, so dass der berechnete Funktionswert kleiner als $a(k, n)$ ist, wenn n der höchste Eingabewert ist
~> Ackermann-Funktion wächst schneller als jede LOOP-berechenbare Funktion

16.3 Zusammenfassung

Zusammenfassung

- **Ackermannfunktion**: sehr schnell wachsende numerische Funktion (total)
- **WHILE-berechenbar**, aber **nicht LOOP-berechenbar**
- wächst schneller als **jede** LOOP-berechenbare Funktion