Mächtigkeit von LOOP-Programmen

Prof. Dr. Berthold Vöcking
Lehrstuhl Informatik 1
Algorithmen und Komplexität
RWTH Aachen

25. November 2011

Die Programmiersprache LOOP – Syntax

Elemente eines LOOP-Programms

- Variablen x_0 x_1 x_2 ...
- Konstanten -1 0 1
- Symbole ; $:=$ $+$ $\neq$
- Schlüsselwörter LOOP DO END

Induktive Definition – Induktionsanfang

Zuweisung

Für jedes $c \in \{-1, 0, 1\}$ ist die Zuweisung

$$x_i := x_j + c$$

ein LOOP-Programm.

Induktive Definition – Induktionsschritte:

Hintereinanderausführung

Falls P_1 und P_2 LOOP-Programme sind, dann ist auch

$$P_1; P_2$$

ein LOOP-Programm.

LOOP-Konstrukt

Falls P ein LOOP-Programm ist, dann ist auch

$$\text{LOOP } x_i \text{ DO } P \text{ END}$$

ein LOOP-Programm, wobei x_i nicht in P vorkommen darf.

Die Programmiersprache LOOP – Semantik

Ein LOOP-Programm P berechnet eine k-stellige Funktionen der Form $f : \mathbb{N}^k \to \mathbb{N}$.

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

- Programme der Form $x_i := x_j + c$ sind Zuweisungen des Wertes $x_j + c$ an die Variable x_i.
- In einem LOOP-Programm P_1; P_2 wird zunächst P_1 und dann P_2 ausgeführt.
- Das Programm LOOP x_i DO P END hat folgende Bedeutung: P wird x_i mal mal hintereinander ausgeführt.

Die Programmiersprache LOOP – Mächigkeit

Definition

Die durch LOOP-Programme berechenbaren Funktionen werden als *primitiv-rekursiv* bezeichnet.

Vermutung von Hilbert (1926): Die Klasse der primitiv rekursiven Funktionen stimmt mit der Klasse der rekursiven (berechenbaren) Funktionen überein.

Ackermann (1929): Diese Vermutung stimmt nicht!

Die Ackermann-Funktion – Definition

Definition

Die Ackermannfunktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist folgendermaßen definert:

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

Die Ackermann-Funktion – Eigenschaften

Monotonie

- $A(m+1,n) > A(m,n)$
- $A(m,n+1) > A(m,n)$
- $A(m+1,n-1) \geq A(m,n)$ (Übungsaufgabe)

Wenn man den ersten Parameter fixiert ...

- $A(1,n) = n+2$,
- $A(2,n) = 2n+3$,
- $A(3,n) = 8 \cdot 2^n - 3$,
- $A(4,n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+2 \text{ viele Potenzen}} - 3$,

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

Wachstum der Variableninhalte in einem LOOP-Programm

Definition der Funktion F_P

- Sei P ein LOOP-Programm
- Seien $x_0, x_1, \ldots, x_k$ die Variablen in P.
- Wenn die Variablen initial die Werte $a = (a_0, \ldots, a_k) \in \mathbb{N}^k$ haben, dann sei $f_P(a)$ das $(k+1)$-Tupel der Variablenwerte nach Ausführung von P.
- Sei $|f_P(a)|$ die Summe der Einträge im $(k+1)$-Tupel $f_P(a)$.
- Wir definieren nun die Funktion $F_P : \mathbb{N} \to \mathbb{N}$ durch

$$F_P(n) = \max \left\{ |f_P(a)| \,\middle|\, a \in \mathbb{N}^{k+1} \text{ mit } \sum_{i=0}^{k} a_i \leq n \right\} .$$

Intuitiv beschreibt die Funktion F_P das maximale Wachstum der Variablenwerte im LOOP-Programm P.

Ackermannfunktion versus F_P

Wir zeigen nun, dass $F_P(n)$ für alle $n \in \mathbb{N}$ echt kleiner ist als $A(m, n)$, wenn der Parameter m genügend groß in Abhängigkeit von P gewählt wird.

Lemma

Für jedes LOOP-Programm P gibt es eine natürliche Zahl m, so dass für alle n gilt: $F_P(n) < A(m, n)$.

Beachte, für ein festes Programm P ist der Parameter m eine Konstante.

Beweis durch Strukturelle Induktion (Überblick)

Induktionsanfang

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

Induktionsschritt (1. Art)

- 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 (2. Art)

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

Beweis des Lemmas

Der Induktionsanfang

- Sei P von der Form $x_i := x_j + c$ für $c \in \{-1, 0, 1\}$.
- Dann gilt $F_P(n) \leq 2n + 1$.
- Somit folgt $F_P(n) < A(2, n)$.

Erläuterung: Vor Ausführung von P könnte gelten $x_j = n$ und alle anderen Variablen haben den Wert 0. Ferner könnte c den Wert 1 haben. Nach Ausführung von P gilt somit $x_i = n + 1$ und somit ist die Summe der Variableninhalte $x_i + x_j = 2n + 1$. Ein größeres Wachstum der Variableninhalte ist nicht möglich.

Beweis des Lemmas

Der Induktionsschritt (1. Art)

- Sei P von der Form $P_1; P_2$.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_{P_1}(\ell) < A(q, \ell)$ und $F_{P_2}(\ell) < A(q, \ell)$.
- Somit gilt

$$F_P(n) \;\leq\; F_{P_2}(F_{P_1}(n)) \;<\; A(q, A(q, n)) \;\; .$$

- Wir verwenden die Abschätzung $A(q, n) \leq A(q+1, n-1)$.
- Es folgt

$$F_P(n) \;<\; A(q, A(q+1, n-1)) \;=\; A(q+1, n) \;\; .$$

Beweis des Lemmas

Der Induktionsschritt (2. Art)

- Sei P von der Form LOOP x_i DO Q END.
- Induktionsannahme: $\exists q \in \mathbb{N} : F_Q(\ell) < A(q, \ell)$.
- Sei $\alpha = \alpha(n)$ derjenige Wert $\{1, \ldots, n\}$ für x_i der $F_P(n)$ maximiert.
- 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.

Der Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt

$$F_P(n) \;\leq\; F_Q(F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha \;\;,$$

  wobei die Funktion $F_Q(\cdot)$ hier α-fach ineinander eingesetzt ist.
- Aus der Induktionsannahme folgt $F_Q(\ell) \leq A(q,\ell) - 1$.
- Dies wenden wir auf die äußerste Funktion F_Q an und erhalten

$$F_P(n) \;\leq\; A(q, F_Q(\ldots F_Q(F_Q(n-\alpha))\ldots)) + \alpha - 1 \;\;.$$

- Wiederholte Anwendung liefert

$$\begin{aligned} F_P(n) &\leq A(q, A(q, \ldots A(q, A(q, n-\alpha))\ldots)) \\ &\leq A(q, A(q, \ldots A(q, A(q+1, n-\alpha))\ldots)) \;\;. \end{aligned}$$

Beweis des Lemmas

Der Induktionsschritt (2. Art) – Fortsetzung

- Bisher haben wir gezeigt

$$F_P(n) \leq A(q, A(q, \ldots A(q, A(q+1, n-\alpha))\ldots)) \ .$$

- Der Definition der Ackermannfunktion entnehmen wir $A(q+1, y+1) = A(q, A(q+1, y))$.
- Auf die innere Verschachtelung angewendet ergibt sich

$$F_P(n) \leq A(q, A(q, \ldots A(q+1, n-\alpha+1)\ldots)) \ ,$$

  wobei die Schachtelungstiefe nur noch $\alpha - 1$ ist.
- Nach weiteren $\alpha - 2$ vielen Anwendungen, folgt

$$F_P(n) \leq A(q+1, n-1) < A(q+1, n) \ .$$

□

Ackermannfunktion nicht LOOP-berechenbar

Satz

Die Ackermannfunktion ist nicht primitiv-rekursiv.

Beweis:

- Angenommen es gibt ein LOOP-Programm, das die Ackermannfunktion berechnet.
- Dann gibt es auch ein LOOP-Programm, das die Funktion $B(n) = A(n, n)$ berechnet. Sei P dieses LOOP-Programm.
- Aus dem Lemma folgt, es gibt $m \in \mathbb{N}$, so dass für jedes $n \in \mathbb{N}$ gilt: $F_P(n) < A(m, n)$.
- Wenn P aufgerufen wird mit Eingabe m, so berechnet P den Funktionswert $B(m)$. Somit gilt $B(m) \leq F_P(m)$.
- Es folgt

$$B(m) \;\leq\; F_P(m) \;<\; A(m,m) \overset{\text{Def. von } B}{=} B(m) \;.$$

- Widerspruch! Also folgt der Satz. □

Schlussfolgerung

Da die Ackermannfunktion (durch eine TM) berechenbar ist, folgt

Korollar

Die Klasse der primitiv-rekursiven Funktionen ist eine echte Teilmenge der rekursiven Funktionen.

Zur Klärung: Technisch beschränken wir uns in diesem Kapitel auf Funktionen $f : \mathbb{N}^k \to \mathbb{N}$, $k \in \mathbb{N}$. Dieselbe Aussage gilt auch für Funktionen der Form $f : \Sigma^* \to \Sigma^*$ über einem beliebigem endlichen Alphabet Σ.

Zusammenfassung – Berechenbarkeit

Wir haben die folgenden Turing-mächtigen Rechenmodelle und Programmiersprachen kennen gelernt.

- Turingmaschine (TM)
- k-Band TM
- Registermaschine (RAM)
- eingeschränkte RAM
- WHILE-Programme (und somit C, Java, Pascal, Postscript, etc.)

LOOP-Programme sind hingegen nicht Turing-mächtig.

Zusammenfassung – Berechenbarkeit

Church-Turing-These

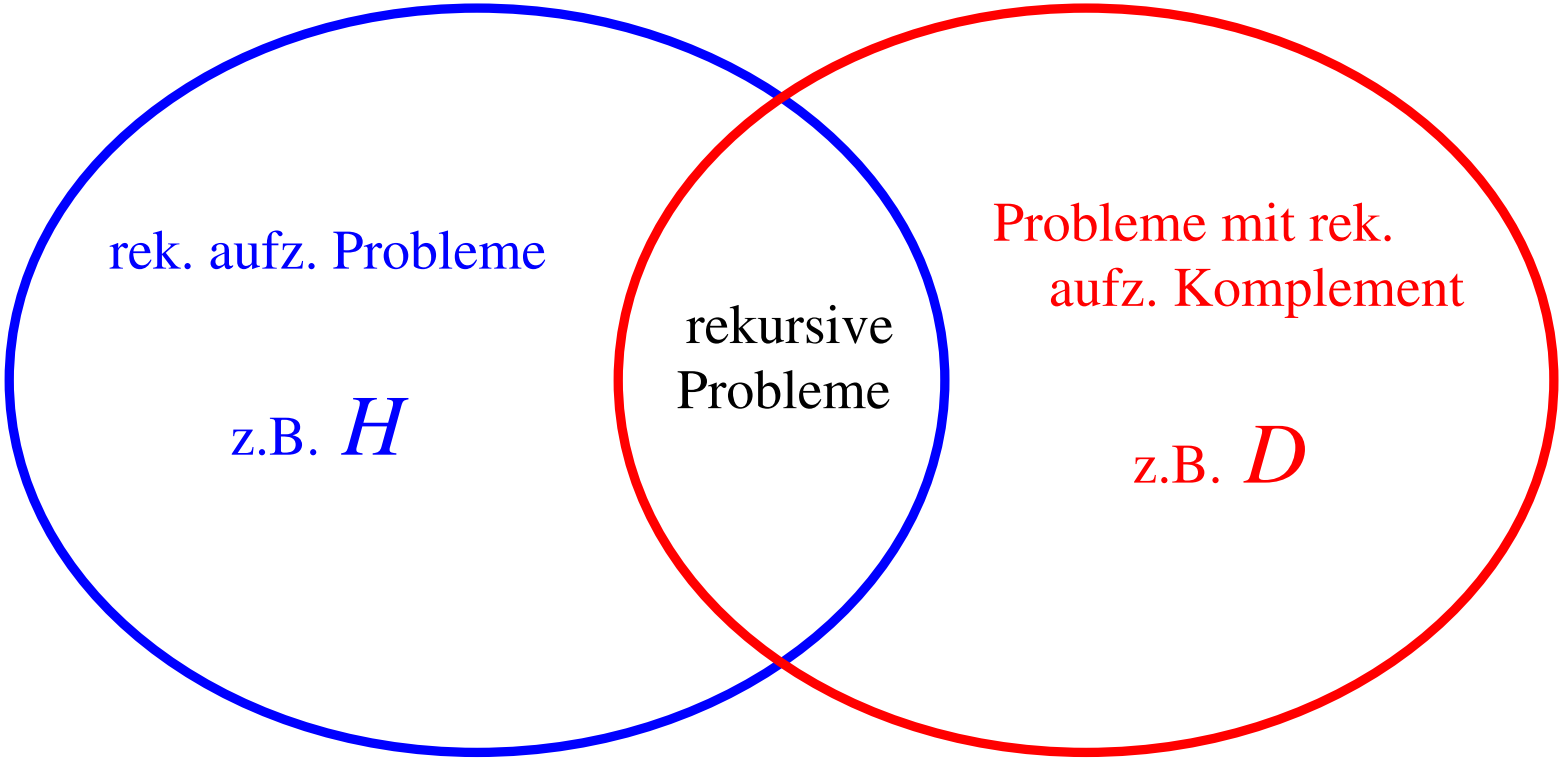
Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der „intuitiv berechenbaren“ Funktionen überein.

In anderen Worten:

Ein Problem kann genau dann „algorithmisch gelöst werden“, wenn es eine TM für dieses Problem gibt.

An Stelle des Begriffs „TM“ können wir auch jedes andere Turing-mächtige Rechenmodell einsetzen.

Berechenbarkeitslandschaft:



nicht rek. aufz. Probleme, deren Komplement ebenfalls nicht rek. aufz. ist

z.B. H_{all}

Zusammenfassung – Berechenbarkeit

Bedeutende nicht berechenbare Probleme:

- Halteproblem, in verschiedenen Varianten
- *Satz von Rice:* Aussagen über Eigenschaften von Funktionen, die durch eine gegebene TM berechnet werden, sind nicht entscheidbar
- *Schlussfolgerung:* Die automatische Verifikation von Programmen in einer TM-mächtigen Programmiersprachen ist nicht möglich
- Hilberts 10. Problem
- Post'sches Korrespondenzproblem

Zusammenfassung – Berechenbarkeit

Methoden zum Nachweis von Nicht-Berechenbarkeit:

- Diagonalisierung
- Unterprogrammtechnik
- Satz von Rice
- Reduktion (spezielle Variante der Unterprogrammtechnik)