Grenzen der Berechenbarkeit

Seminar 1935: Didaktik der Informatik

ausgearbeitet von

Inga Saatz, Matrikel-Nr. 2319705

Fachbereich Informatik der
FernUniversität -GHS- Hagen
SS 1996

Inhaltsverzeichnis

Kapitel 1

Einleitung

Aufwendige Berechnungen wurden bereits in der Antike durchgeführt. Beispiele finden sich unter anderem in folgenden Bereichen der Astronomie (das Epizykel-Modell von Ptolomäus (100-160 n.Chr.), der Kalender der Mayas), der Statik und des Vermessungswesens (Tempelbauten, Bewässerungskanäle der Babylonier), aber auch im Staatswesen (Volkszählungen, Steuererhebungen) und im Handel (Zahlungsverkehr). Als Hilfsmittel für diese Berechnungen dienten zunächst die Finger, der Abakus und die bis dahin entwickelten Zahlen- und Rechensymbole. Um den Aufwand der Berechnungen zu verringern, wurden erste mechanische Rechenmaschinen für die Grundrechenarten entwickelt, deren Weiterentwicklungen in unserem Jahrhundert zu den Computern führten. Die Zeit für die Durchführung einer arithmetischen Operation hat im letzten Jahrzehnt drastisch abgenommen, so daß immer größere und komplexere Berechnungen durchgeführt werden können.
Doch heißt dies auch, daß alles immer genauer berechnet werden, bzw. in naher Zukunft noch berechnet werden kann ?
Wir vertrauen inzwischen darauf, daß die alle Berechnungen durch Computer schnell und exakt durchgeführt werden können. Dies führt dazu, daß der Kassenzettel beim Einkauf nicht mehr nachgerechnet wird oder der Bug im Pentium-Chip erst relativ spät aufgedeckt wurde, als auch, daß den Rechenfehlern bei der Interpretation von aufbereitetem Datenmaterial zuwenig Beachtung geschenkt wird. Insbesondere durch die rasante Entwicklung der elektronischen Datenverarbeitung in den letzten Jahren ist man geneigt anzunehmen, daß auch die letzten noch ungelösten Probleme in den nächsten Jahren gelöst werden können, wenn genügend großer Speicherplatz und hohe Rechengeschwindigkeit vorhanden sein wird.
Daß auf der anderen Seite auch der Berechenbarkeit Grenzen gesetzt sind, wollen wir eigentlich nicht wahrhaben, verlangen aber, daß alle Programme die gewünschten Ergebnisse möglichst schnell und fehlerfrei durchführen. Am Besten wäre es doch, wenn ein Programm geschrieben werden könnte, daß auf jedes Problem auf Tastendruck eine Lösung liefert.

Die Suche nach einem solchen Programm gleicht einem perpetuum mobile und würde hohe wirtschaftliche verursachen. Es soll in diesem Artikel auf einfache Art erläutert werden, wo die Grenzen der Berechenbarkeit liegen als auch den naiven Berechenbarkeitsbegriff zu konkretisierten.

Kapitel 2

Mathematische und Informatische Grundlagen

2.1 Ein naiver Algorithmusbegriff

Solange es darum geht, für bestimmte Probleme einen Lösungsalgorithmus zu entwickeln, reicht der folgende anschauliche Algorithmusbegriff aus:

Definition 2.1 *Ein Algorithmus ist ein schrittweise ablaufendes Verfahren für Objekte eines bestimmten Bereichs, das folgenden Bedingungen genügt[Bre 91]:*

1. *endliche Beschreibbarkeit*
2. *Determiniertheit*
3. *Universalität*
4. *effektive Ausführbarkeit*

Auf dieser Basis können dann die Begriffe der berechenbaren Funktionen auf der Menge der natürlichen Zahlen $\mathbb{N}$ und der entscheidbaren Mengen eingeführt werden:

Definition 2.2

1. *Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt berechenbar, wenn es einen Algorithmus gibt, der zu beliebig vorgegebenem Argument x aus dem Definitionsbereich nach endlich vielen Schritten den Funktionswert $f(x)$ liefert[Bre 91].*
2. *Eine Menge $M \subseteq \mathbb{N}$ heißt entscheidbar, wenn es einen Algorithmus gibt, der für jedes $x \in \mathbb{N}$ nach endlich vielen Schritten die Antwort „ja“oder „nein“auf die Frage liefert, ob x ein Element von M ist[Bre 91].*

Mit diesen Definitionen könnte wir uns zufrieden geben, wenn es nicht Probleme gäbe, zu denen keine algorithmische Lösung existiert. Beispiele für solche Probleme sind:

- die Vermutung von Fermat (1601-1665):
  Es gibt keine natürliche Zahl größer gleich drei, für die positive ganze Zahlen x, y, z mit $x^n + y^n = z^n$ existieren.
- die Vermutung von Goldbach (1690-1764):
  Jede gerade Zahl größer vier ist als Summe von zwei Primzahlen darstellbar.

Dies ist ein Indiz dafür, daß unsere bisherigen Definitionen noch nicht präzise genug sind. Insbesondere bleibt in der Definition des Algorithmus unklar, was unter einer effektiven Ausführbarkeit zu verstehen ist.

2.2 Primitiv-rekursive Funktionen

Um von den anschaulichen Algorithmusbegriff präzisieren zu können, wollen wir zunächst einige Grundfunktionen betrachten, die im anschaulichen Sinne berechenbar sind und zudem eine einfache Gestalt besitzen. Ausgehend von diesen Grundfunktionen werden kompliziertere Funktionen durch die Anwendung solcher Operationen konstruiert, daß auch die Berechenbarkeit der konstruierten Funktionen gewährleistet ist.

Definition 2.3 *Die primitiv-rekursiven Grundfunktionen sind gegeben durch:*

$$\begin{array}{lcll} N, O & : & \mathbb{N} \to \mathbb{N} & \\ O(x) & := & 0 & \text{Nullfunktion} \\ N(x) & := & x+1 & \text{Nachfolgerfunktion} \\ P & : & \mathbb{N}^n \to \mathbb{N} & \\ P_i^n(x_1, \cdots, x_n) & := & x_i & \text{i.Projektion}, 1 \leqslant \text{i} \leqslant \text{n} \end{array} .$$

Aus diese Grundfunktionen können folgende Operationen angewandt werden:

Definition 2.4 *1. Der Substitutionsoperator*
Es seien $k, m \in \mathbb{N}, m \geqslant 1, f : \mathbb{N}^m \to \mathbb{N}$ und $g_1, \cdots g_m : \mathbb{N}^k \to \mathbb{N}$. Dann sei eine Funktion für alle $\mathbf{x} \in \mathbb{N}^k$

$$Sub(f, g_1, \cdots, g_m) := h : \mathbb{N}^k \to \mathbb{N}$$

definiert durch

$$h(\mathbf{x}) := f(g_1(\mathbf{x}, \cdots, g_m(\mathbf{x})) \,. \tag{2.1}$$

2. *Der Operator der primitiven Rekursion*
Es sei $k \in \mathbb{N}, g : \mathbb{N}^k \to \mathbb{N}$ *und* $h : \mathbb{N}^{k+2} \to \mathbb{N}$. *Dann sei*

$$Prk(g,h) := f : \mathbb{N}^{k+1} \to \mathbb{N}$$

die durch folgende Gleichungen für alle $\mathbf{x} \in \mathbb{N}^k, y \in \mathbb{N}$ *eindeutig bestimmte Funktion:*

$$\begin{aligned} f(\mathbf{x},0) &= g(\mathbf{x}) \\ f(\mathbf{x},y+1) &= h(\mathbf{x},y,f(\mathbf{x},y)) \,. \end{aligned} \tag{2.2}$$

3. *Der Operator der* $\mu-$*Rekursion*
Es sei $k \in \mathbb{N}, h : \mathbb{N}^{k+1} \to \mathbb{N}$. *Dann sei*

$$\overline{\mu}(h) : \mathbb{N}^k \to \mathbb{N}$$

wie folgt definiert:

$$\overline{\mu}(h)(\mathbf{x}) := \begin{cases} \min(M) & \text{falls } M := \{n \mid h(n) = 0 \,\text{und}\, (\forall i < n)\, i \in Def(h)\} \neq \emptyset \\ \text{div} & \text{sonst} \end{cases} \tag{2.3}$$

Wie reichhaltig die Funktionenklasse ist, die durch diese Operationen aus den Grundfunktionen erzeugt werden können, erkennen wir bereits an den folgenden Beispielen:

Beispiel 2.1 *Es sei* $f(x_1,x_2) := x_1 + x_2), pr_3^{(3)} = x_i$. *Dann gilt:*

$$\begin{aligned} h &:= \text{Sub}(f, pr_1^{(3)}, pr_3^{(3)}) : \mathbb{N}^3 \to /BbbN \\ h(x,y,z) &= f(pr_1^{(3)}(x,y,z), pr_3^{(3)}(x,y,z)) = f(x,z) = x+z \end{aligned}$$

Beispiel 2.2

Es seien $g : \mathbb{N} \to \mathbb{N}, h : \mathbb{N}^3 \to \mathbb{N}$ definiert durch $g(x) = 0, h(x,y,z) = x + z$. Es sei $f : \text{Prk}(g,h)$. Dann gilt für alle $x \in \mathbb{N}$:

$$\begin{aligned} f(x,0) &= g(x) = 0 \\ f(x,1) &= h(x,0,f(x,0)) = x + f(x,0) = x \\ f(x,2) &= h(x,1,f(x,1)) = x + f(x,1) = 2x \\ &\vdots \\ f(x,y) &= x \cdot y \,. \end{aligned}$$

Beispiel 2.3 *Es sei* $h(x,y) := (y^2 \dot{-} x) + (x \dot{-} y^2)$. *Dann ist*

$$\begin{aligned} \{n \mid h(x,n) &= 0 \,\text{und}\, (\forall i < n)(x,i) \in \mathrm{Def}(h)\} \\ &= \{n \mid h(x,n) = 0\} = \{n \mid n^2 = x\}\,. \end{aligned}$$

Daher gilt:

$$\begin{aligned} \tilde{\mu}(h)(x) &= \begin{cases} \min M & \text{falls}\, M := \{n \mid n^2 = x\} \neq \emptyset \\ div & \text{sonst} \end{cases} \\ &= \begin{cases} \sqrt{x} & \text{falls}\, x \,\text{Quadratzahl ist} \\ div & \text{sonst} \end{cases} \end{aligned}$$

Mit diesen Definitionen gewappnet können wir zur Definition zweier durch die primitiv-rekursiven Grundfunktionen und den obigen Operationen erzeugten Funktionenklassen schreiten:

Definition 2.5
1. *Eine Funktion* $f : \mathbb{N}^k \to \mathbb{N}$ *heißt genau dann primitiv-rekursiv, wenn sie eine primitive Grundfunktion ist oder wenn sie aus diesen Grundfunktionen durch wiederholte Anwendung der Substitution und der primitiven Rekursion in endlich vielen Schritten konstruiert werden kann.*
2. *Eine Funktion* $f : \mathbb{N}^k \to \mathbb{N}$ *heißt genau dann partiell-rekursiv, wenn sie eine primitive Grundfunktion ist oder aus diesen Grundfunktionen durch wiederholte Anwendung der Substitution, der primitiven Rekursion und des* $\mu-$*Operators in endlich vielen Schritten erhalten werden kann.*

Bei den obigen Beispielen handelt es sich bei den ersten Beiden um primitiv rekursive Funktionen und bei dem Drittten um partiell-rekursive Funktionen. Es stellt sich nun die Frage, ob die Definition der μ-rekursiven Funktionen eine echte Erweiterung der Klasse der primitiv-rekursiven Funktionen darstellt. Ein Beispiel einer Funktion, die intuitiv berechenbar aber nicht primitiv rekursiv ist, ist durch die sogenannte Ackermann-Funktion gegeben.

Beispiel 2.4 *Es seien Funktionen* $A_0, A_1, A_2, \cdots$ *wie folgt definiert:*

$$A_0(x) := \begin{cases} 1 & \text{falls}\, x = 0 \\ 2 & \text{falls}\, x = 1 \\ x+2 & \text{sonst} \end{cases}$$

$$A_{n+1}(x) := A_n^x(1) = \underbrace{A_n \circ \cdots \circ A_n}_{\text{x}-\text{mal}}(1)\,.$$

Diese Funktionenfolge ist primitiv-rekursiv und durch vollständige Induktion können die folgenden Eigenschaften gezeigt werden:

1. *Für alle $x, x', n, n' \in \mathbb{N}$ gilt: $(x \leqslant x'$ und $n \leqslant n') \Rightarrow A_n(x) \leqslant A_{n'}(x')$.*
2. *Für alle einstelligen primitiv rekursiven Funktionen f existieren $i, x_0 \in \mathbb{N}$, so daß für alle $x \geqslant x_0$ gilt: $f(x) \leqslant A_i(x)$.*

Aufbauend auf dieser Funktionenfolge wird die Ackermann-Funktion definiert durch $A : \mathbb{N} \to \mathbb{N}, A(x) := A_x(x)$. Die Ackermann-Funktion ist nicht primitiv-rekursiv, denn wäre sie es, so wäre auch die Funktion $B := A + 1$ primitiv-rekursiv. Daher existieren Konstanten i, x_0, so daß $B(x) \leqslant A_i(x)$ für $x \geqslant x_0$ gilt. Andererseits gilt $A_i(x) \leqslant A_x(x)$ für $x \geqslant i$. Mit $A(x) + 1 \leqslant A_i(x) \leqslant A_x(x) = A(x)$ für alle $x \geqslant \max(x_0, i)$ ergibt sich ein Widerspruch. Daher kann die Annahme nicht gültig sein und die Ackermann-Funktion ist nicht primitiv-rekursiv.

Wir haben bisher gesehen, daß eine große Vielfalt von intuitiv berechenbaren Funktionen zur Klasse der primitiv-rekursiven Funktionen gehört, diese Klasse jedoch nicht alle intuitiv-berechenbare Funktion umfaßt. Durch Erweiterung der Operationen um die μ-Rekursion gelangten wir zu der Klasse der partiell-rekursiven Funktionen.
Nun stellt sich hier die Frage, ob zur Beschreibung aller intuitiv berechenbaren Funktionen weitere Operationen notwendig sind. Hierzu hat A.Church [Chu 36] die folgende Hypothese aufgestellt:

Satz 2.1 *Die partiell-rekursiven Funktionen sind genau die intuitiv und im anschaulichen Sinne berechenbaren Funktionen.*

Diese Hypothese läßt sich nicht beweisen, da nicht genau definiert ist, was unter intuitiv-berechenbaren Funktionen zu verstehen ist. Ein Indiz dafür, daß diese Hypothese wahr ist, ergibt sich daraus, daß auch andere Maschinenmodelle [Wei 90] (Turingmaschinen, Register-Berechenbare Maschinen) auf hierzu äquivalente Aussagen führen.
Bis zu diesem Punkt haben wir uns mit einer berechenbaren Funktionenklasse beschäftigt und noch kein Wort über nicht-berechenbare bzw. nicht-algorithmisierbare Problemstellungen verloren. Die in Abschnitt 2.1 genannten Beispielen können als Anwendungen des Halteproblems betrachtet werden, welches folgendermaßen gestellt wird:

Existiert ein Algorithmus, der für jedes beliebige Programm P und jeder beliebige Eingabe X nach endlicher Zeit entscheiden kann, ob das Programm P mit der Eingabe X terminiert oder nicht.

Ein Spezialfall dieses Halteproblems ist das Selbstanwendungsproblem.
Dieses lautet:

Existiert ein Algorithmus, das für jedes beliebige Programm P, welches auf sich selbst angewandt wird, entscheidet, ob es nach endlich vielen Schritten terminiert oder nicht.

Es kann gezeigt werden, daß Programme, die bei Anwendung auf sich selbst stoppen nicht existieren können (s. 3.3.1). Hieraus folgt auch die Unlösbarkeit des Halteproblems und dessen Anwendungen.

2.3 Die Grenze der Berechenbarkeit und reale Rechner

Das Maschinenmodell, welches einem realen in der Assembler programmierten Computer am nächsten kommt, ist das der Register-Maschine. Diese besteht aus endlich vielen Registern R_i in denen natürliche Zahlen gespeichert werden können und mit dem Befehlssatz des Hoch- und Herunterzählens der Registerinhalte und der bedingten und unbedingten Sprünge zu Sprungmarken programmiert werden.
Die Rechenzeit einer Turingmaschine bzw. einer Registermaschine ist gleich der Zahl der durchgeführten Rechenschritte für eine Registerbelegung $\mathbf{x}$, gegeben durch die Rechenzeitfunktion $Zeit_M(\mathbf{x})$. Für Register-berechenbare Funktionen ergibt sich nun die folgende Äquivalenz:

Satz 2.2 *Es sei $f : \mathbb{N}^k \to \mathbb{N}$ und $Zeit_M$ die Rechenzeitfunktion einer Register-berechenbar Funktionen f_M. Dann sind äquivalent:*

1. *f ist primitiv-rekursiv*
2. *Es gibt eine register-berechenbare Funktion $f_M : \mathbb{N}^k \to \mathbb{N}$ und eine primitiv-rekursive Funktion $b : \mathbb{N}^k \to \mathbb{N}$, so daß $f_M = f$ und $Zeit_M(\mathbf{x}) \leqslant b(\mathbf{x})$ für alle $\mathbf{x} \in \mathbb{N}^k$ gilt.*

Um zu zeigen, daß eine Funktion f primitiv-rekursiv ist, kann daher eine Register-Maschine anzugeben, die f berechnet und zu beweisen, daß die Rechenzeitfunktion durch eine primitiv-rekursive Funktion beschränkt ist. Mit der Ackermannschen Funktionenfolge (A_n) ist uns bereits eine sehr schnell wachsende Folge von primitiv-rekursiven Funktionen bekannt, daher sollte jede praktisch-berechenbare totale Funktion primitiv-rekursiv sein.
Wenn die Speicherplatzschranke s eines Computers bekannt (berechenbar) ist, so kann die Zahl der verschiedenen Registerbelegungen für k Register und der möglichen Eingaben abgeschätzt werden. Abhängig von der Zahl der Zustände im Zustandsdiagramm des jeweiligen Programms kann daher die Rechenzeit abgeschätzt werden, die ein Computer benötigt, alle möglchen Speicherplatzbelegungen durchzuführen[1]. Gerät ein Algorithmus andererseits in eine Endlosschleife, so durchläuft dieser immer dieselben Konfigurationen. Wird die vorgenannte Zeitschranke überschritten, so kann der Algorithmus für die vorliegende Eingabe nicht terminieren. In diesem Sinne ist für reale Computer das Halte- und das Selbstanwendungsproblem lösbar. Die sich ergebenden Zeitschranken gehen jedoch mindestens exponentiell mit dem Speicherplatz, so daß sich dieses Verfahren der Praxis nicht durchführen läßt.
In der Abbildung 1 werden nochmals die behandelten Begriffe gegenübergestellt.

[1] Für Turingmaschinen wird in [Weg 96](Seite 33) eine Abschätzung angegeben

Kapitel 3

Didaktische Aufbereitung

3.1 Vorzunehmende Vereinfachnungen und Lehrziele

Natürlich ist der ganze formale Apparat des letzten Kapitels nicht für den Schulunterricht geeignet. Um den hier betrachteten Sachverhalt sowohl bereits in der Sekundarstufe 1 als auch in der Sekundarstufe 2 in den Unterricht einbringen zu können sind die folgenden Vereinfachungen vorzunehmen:

1. Mit der Existenz von Computern ist gezeigt, daß Register-berechenbare Funktionen existieren. Im Folgenden wird daher nicht mehr zwischen dem Modell der Registermaschinen und realen Computern unterschieden. Folglich wird auch auf eine Definition einer Maschine und auf die Einführung von Register- bzw. Kellermaschinen verzichtet.

2. Die Grenze der Berechenbarkeit soll anhand von Beispielen exemplarisch herausgearbeitet werden. Dieses bezieht sich zum einen auf die Grenze im Modell als auch auf die Grenze, die durch die Zahlendarstellung im realen Computer gegeben ist.

Es kommt in erster Linie darauf an, den Schülern die Ideen und Konzepte der Berechenbarkeitstheorie anhand von Beispielen näherzubringen. Daher ergeben sich folgende Lehrziele:
Die Schüler sollen

1. die Schächen des anschaulichen Algorithmusbegriffs erkennen können,

2. Beispiele für algorithmisch „harte“Probleme angeben und bei einfachen Beispielen die Rechenzeit abschätzen können,

3. den Einfluß der Rechengenauigkeit kennen,

4. das Selbstanwendungs- und das Halteproblem für Pascal-Programme formulieren und erläutern können,

5. und weitere Beispiele von nicht-algorithmisierbaren Problemen angeben können.

3.2 Die Rechengenauigkeit

Fehler in Daten können verschiedene Ursachen besitzen. So können diese auf Meßfehler beruhen, oder bereits sich aus Näherungen oder Annahmen der zugrundeliegenden beschreibenden Theorien und Modelle (z.B. ein Epizykel-Modell von Kepler zur Beschreibung der Planetenbewegung) ergeben. Auf diese Probleme der Fehlerfortpflanzung soll hier jedoch nicht weiter eingegangen werden. Andererseits besteht die weitverbreitete Annahme, daß die Berechnungen, die mit einem Computer durchgeführt werden, exakt sind. Hierauf soll in diesem Abschnitt näher eingegangen werden.

Wie auch mit dem Taschenrechner können Berechnungen mit dem Computer im allgemeinen nicht exakt durchgeführt werden, da zum einen die reellen Zahlen prinzipiell nicht durch endliche Dezimalzahlen dargestellt werden können und zum anderen nur mit endlicher Genauigkeit Rechnungen durchgeführt werden. Daher sollen zunächst die Zahlenräume und die Zahldarstellungen wiederholt werden.

Zahlenraum	Zahlentyp	Speicherdarst.	Wertebereich		
ganze Zahlen (Z)	Integer	2 Byte	-32.768	bis	32.767
			MinInt	bis	Maxint
rationale Zahlen (Q)	Real	4 Byte	$-3,4..\cdot 10^{38}$	bis	$-4,9..\cdot 10^{-45}$
			$1,4..\cdot 10^{-45}$	bis	$3,4..\cdot 10^{38}$
			MinReal	bis	MaxReal

Neben diesen aufgeführten Zahlentypen finden noch weitere Verwendung, die eine acht Byte lange Zahlendarstellung besitzen. Wichtig für uns ist es hier, daß durch die interne Zahlendarstellung im Computer nur bestimmte Zahlen aus einem bestimmten Zahlenbereich erfaßt werden.

Als Anwendung können hierzu die Zahlenrepräsentationen der rationalen Zahlen ($\frac{1}{6}, \frac{1}{7}, \cdots$ im Taschenrechner und Computer im Unterricht behandelt werden. Als Programmieraufgaben bieten sich folgende Aufgaben an:

1. Quadratwurzelberechnung mit dem Bisektionsverfahren
Programmeingaben: relle Zahl, obere und untere Intervallgrenzen
Bei der Berechnung von $\sqrt{2}$ ergibt sich nach einigen Iterationsschritten die Situation, daß die obere und untere Intervallgrenze in der computerinternen Darstellung zwei benachbarte Zahlen sind. Die Folge ist dann, daß das Programm bei unpassend gewählten Abbruchbedingungen nicht terminiert. Das Programm endet ebenfalls dann nicht, wenn negative Zahlen oder nicht passende Intervallgrenzen eingegeben werden. Dieses Verhalten soll herausgestellt werden und entsprechende Bedingungsabfragen in das Programm eingebaut werden. Desweiteren kann nach der Anzahl der Iterationsschritte gefragt werden sowie nach Möglichkeiten, die Startintervallgrenzen vom Programm wählen zu lassen. Beispielsweis als untere Grenze Null und als obere Grenze Eins, für Quadratzahlen kleiner Eins und sonst die Quadratzahl selbst.

2. Nullstellenberechnung quadratischer Polynome mit dem Bisektionsverfahren
Das vorangehende Programm soll so abgewandelt werden, daß die Nullstellen quadratischer Polynome bestimmt werden können.
Hierbei soll erkannt werden, daß die Wahl der Funktionsdarstellung $f(x) = ax^2 + bx + c$ ungeeignet ist nullstellenfreie Polynome zu erkennen. Dieses ist durch Umschreibung der Funktionsdarstellung auf die Form $f(x) = a(x+b)^2 + c$ möglich, wobei der Fall $a = 0$ separat zu behandeln ist. Auch hier kann nach der Rechengenauigkeit und der Anzahl der Iterationsschritte in Abhängigkeit von den Startwerten und der Terminierung des Programms gefragt werden.

3. Berechnung der Kreiszahl π
Bestimmung der Näherungen an den Flächeninhalt eines Kreises ($A = \pi r^2$) durch Berechnung des Flächeninhaltes von einbeschriebenen (A_n) bzw. umbeschriebenen (B_n) gleichseitigen Vielecken. Eine Aufstellung von Berechnungsergebnissen zeigt die Abbildung 2 des Anhanges. Ziel ist es hierbei, zu erkennen, daß zwei verschiedene Algorithmen zur Berechnung desselben Sachverhaltes unterschiedliche Genauigkeiten besitzen können.

An diesen Beispielen können folgende Situationen erkannt werden:
Es gibt Wertebereiche, für die Programme nicht terminieren sowie Problemstellungen, für die es keine Programme gibt. Desweiteren ergeben sich durch die Zahlendarstellungen im Computer Rechenfehler.
Diese Rechenfehler sind jedoch unabhängig von den Fehlern, mit denen die Ausgangsdaten bereits behaftet sein können oder der zugrundeliegenden Theorie.
Ein weiterführendes Programmierbeispiel führt auf das Themengebiet des deterministischen Chaos. Dieses ergibt sich, wenn ein nichtlineares System rückgekoppelt wird. Als Beispiel betrachten wir die sogenannte Verhulst-Funktion, die erstmals in einer Arbeit

über Populationsdynamik eingeführt wurde:

$$x_{n+1} = F(x_n, a) = a \cdot x_n \cdot (1 - x_n) \quad (n \geqslant 0)\,.x_0 \in (0,1)$$

Im Bereich $1 < a \leqslant 3$ konvergiert die Folge (x_n) gegen $1 - \frac{1}{a}$, während die einzelnen Folgenglieder für a aus dem Intervall $[3.57, 4]$ jeden Wert auf dem Intervall $(0, 1)$ annehmen und nirgends konvergieren . Es ist sogar so, daß für kleine Abweichung der Startwerte x_0 und $\tilde{x_0}$ voneinander die n-te Iterierten stark voneinander abweichen können[Sch 93, Sch 96].
Man erkennt hieran, daß kleine Variationen der Startwerte zu großen Unterschieden in den Ergebnissen führen können, und somit kleine Rechenfehler eine große Wirkung bekommen. Phänomene dieser Art machen es unmöglich, langfristige Wettervorhersagen zu erstellen. In solchen Fällen sind die Probleme in einem gewissen Umfang zwar berechenbar, aber die Ergebnisse sind mit so großen Fehlern behaftet, daß Schlußfolgerungen nicht mehr gezogen werden können.

3.3 Die Rechenzeit

Wir haben bereits Algorithmen kennengelernt, die für bestimmte Eingaben nicht terminieren. Hierzu zählt beispielsweise die Bestimmung der Quadratwurzel mit dem Bisektionsverfahren bei der Eingabe eines negativen Wertes oder die Nullstellenbestimmung nullstellenfreier Funktionen. Bei diesen Beispielen können wir Kontrollen in den Algorithmus einfügen, so daß dieser terminiert. Anders gelagert ist das Problem der Berechnung der Kreiszahl π, die nur bis zu einer endlichen Genauigkeit gelöst werden kann.
Neben der Rechengenauigkeit, die über die interne Zahlendarstellung mit dem Speicherplatz eines realen Rechners verknüpft ist, handelt es sich bei der Rechenzeit um eine Grenze der Berechenbarkeit auf realen Rechnern.
Um den Schülern ein Gefühl für diese Problematik zu vermitteln, soll als Programmier- und Übungsaufgabe das Problem des Handelsreisenden gestellt werden:
Betrachte die folgende Problemstellung:

Ein Handelsreisender soll, ausgehend von einer Stadt, n-1 andere Städte genau einmal durchreisen und am Ende wieder in die Ausgangsstadt zurückkehren. Gesucht ist die Reihenfolge, in der der Handelsreisende die Städte besuchen muß, damit die Summe der zurückgelegten Entfernungen minimal ist.
Die Entfernungstabelle für eine Fünf-Städte-Tour mit Anfangs- und Endpunkt A ist gegeben durch [Bre 90]:

	A	*B*	*C*	*D*	*E*
A	-	*7*	*4*	*5*	*3*
B	*4*	-	*7*	*9*	*4*
C	*6*	*3*	-	*4*	*7*
D	*2*	*6*	*4*	-	*12*
E	*8*	*5*	*5*	*3*	-

1. *Stelle die Entfernungstabelle graphisch dar, indem Du die Städte durch Kreise markierst und an den Verbindungslinien die Weglänge kennzeichnest.*

2. *Bestimme die Reihenfolge der Fünf-Städte-Tour mit der kleinsten und die mit der längsten Weglänge.*

3. *Wieviele mögliche Reiserouten gibt es ?*

Im anschließenden Unterrichtsgespräch kann geklärt werden, wie die einzelnen Schüler zu ihren Reiserouten gekommen sind. Hierauf aufbauend kann eine Algorithmus entwickelt werden, der dieses Problem löst. Um im nächsten Schritt den Rechenzeit einfach abschätzen zu können, wird ein auf Permutationen basierender Lösungsalgorithmus[1] entwickelt und nicht der Algorithmus von Dijkstra [Güt 92]. Ausgehend von der Zahl der möglichen Reiserouten von (n-1)! bei einer n-Städte-Tour kann nun folgende Aufgabe gestellt werden:
Wenn für die Berechnung der Weglänge jeder Reiseroute der Algorithmus Millisekunde benötigt, wie lange dauert dann die Berechnung der minimalen Tour für n= 5, 7, 10, 12, 14, 15, 20, 22 ?
Gebe Deine Ergebnisse in passenden Zeiteinheiten an und finde hierzu passende Vergleichszeiträume. Messe die Laufzeit des Programmes für n=7 und vergleiche sie mit Deinen berechneten Werten.

n	Zeit	Vergleichszeitraum
5	0,12 s	Lidschlag
7	5 s	
10	1 Stunde	
12	5,5 Tage	
13	2,5 Monate	
14	2 Jahre 9 Monate	
15	100 Jahre	ein Menschenleben
20	77 Mio Jahre	Zeitalter der Dinosaurier
21	1,5 Milliarden Jahre	Entstehung erster Landmasse
22	35 Milliarden Jahre	Alter des Universums $\approx$ 12-15 Milliarden Jahre

[1] Ein Pascal-Programmlisting zu diesen Problem ist in dem Artikel von Breier [Bre 90] zu finden.

3.3.1 Das Halte- und das Selbstanwendungsproblem

Das bereits im vorletzten Abschnitt verwendete Bisektionsverfahren kann auf verschiedene Weise implementiert werden. Als Konstruktionselemente kommen hierbei Repeat- und While-Schleifen, aber auch Rekursionen in Betracht.
Letzteren wollen wir einführend unsere Aufmerksamkeit widmen. Erstellt man ein Flußdiagramm zu dem Algorithmus zur Berechnung von Quadratwurzeln, so erkennt man, daß ein Teil des Diagramms mehrfach durchlaufen wird. Um das Selbstanwendungsproblem als Spezialfall des Halteproblems zu motivieren, sollen von den Schülern Beispiele gefunden werden, bei denen eine Rückkehr zum Ausgangspunkt unter gleichen oder unterschiedlichen Bedingungen erfolgt.
Beispiele hierfür können sein:

- das Spiegelbild zweier sich genau gegenüber angebrachten Spiegel
- Pendelschwingungen
- Regelkreise (Thermostat in der Heizung/Klimaanlage)
- Blutkreislauf
- Wasser- und Luftkreislauf
- Geldkreislauf
- der sogenannte „Schweinezyklus“in den Wirtschaftswissenschaften, der die Abhängigkeit von Angebot und Preisen am Beispiel der Schweinemast darstellt
- Populationsentwicklungen (Räuber- Beute-Modelle)

Im ersten Beispiel wird das Bild eines Objektes, welches sich zwischen den beiden Spiegeln befindet, jeweils im gegeüberliegenden Spiegel verkleinert abgebildet. Da das jeweilige Bild <u>kleiner</u> als das Original ist, ergibt sich eine Konvergenz der Bilderfolge in der geometrischen Mitte der Spiegel.
Als Anwendung bieten sich Aufgaben aus der Populationsentwicklung an, wie zum Beispiel die Aufgabe von Fibonacci, welche die Kanninchenvermehrung untersucht:
Neugeborene Kaninchenpaare bringen nach dem ersten und dem zweiten Monat jeweils ein neues Kaninchenpaar zur Welt, und stellen danach ihre Fortpflanzung ein, während die neugeborenen Paare sich wie ihre Eltern verhalten
Man erhält also folgende Rekursionsgleichungen:

$$a_1 = a_2 := 1 \qquad a_n := a_{n-1} + a_{n-2} \quad (n = 3, 4, \cdots),$$

$$a_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right],$$

wobei die letzte Aussage durch vollständige Induktion gezeigt werden kann.

Neben diesen Beispielen gibt es aber auch Fälle, bei denen die Rückkehr zum Ausgangspunkt zu einer Katastrophe führt. Ein Beispiel hierfür ist das Auftreten einer Resonanzkatastrophe beim getriebenen Pendel, wenn die treibende Kraft genau die Eigenfrequenz besitzt. Desweiteren treten Probleme dann auf, wenn der Ausgangs- und der Folgezustand nicht mehr unterschieden werden können und so eine Vorschrift auf sich selbst angewendet wird. Von den Schülern gefundende Beispiele können hierbei sein:

1. Was war früher da: die Henne oder das Ei?
2. Ein Kreter sagt: „Alle Kreter lügen".
3. Bild von Escher: zeichnende Hände (siehe Abbildung 3)
4. Die Menge aller Mengen
   Lösung: Man spricht von einer Klasse von Mengen
5. Das Zeitreiseparadoxon
6. Schreibe ein Programm, das sich selbst löscht

Welche Auswirkung haben diese Beispiele auf die Terminierung von Programmen ?
Für unsere Programme bedeutet dies, daß ein Ausgangszustand nicht mit einem Folgezustand zusammenfallen darf. Speziell bei dem Programm zur Berechnung der Quadratwurzel bedeutete dies, daß in dem Fall, als die Intervallgrenzen sich nicht mehr änderten, der Ausgangs- und Endzustand zusammenfielen und sich so eine Endlosschleife ergab.
Dieses Phänomen tritt nicht bei Rekursionen auf, da bei jedem Rekursionsschritt wird der zugehörende Parametersatz auf dem Stack abgelegt wird, so daß durch die Reihenfolge der Parametersätze die einzelnen Rekursionsaufrufe unterscheidbar sind, solange sich der Inhalt der Parametersätze voneinander unterscheidet.
Betrachten wir andererseits PASCAL-Programme als Zeichenketten, so können diese Zeichenketten durchaus als Eingabe eines PASCAL-Programmes dienen. Wir werden durch diese Fragestellung auf das Selbstanwendungsproblem (für PASCAL-Programme) geführt, wenn wir nach der Terminierung eines solchen Systems fragen.
Bevor die Unlösbarkeit des Selbstanwendungsproblems und des Halteproblems für PASCAL-Programme im Unterrichtsgespräch entwickelt wird, sollte zunächst die Argumentationsweise von indirekten Beweisen wiederholt bzw. eingeführt werden. Im Anschluß hieran kann der Beweis argumentativ geführt werden:
Beweis: Annahme: Es gibt ein PASCAL-Programm P_S, das entscheidet, ob ein PASCAL-Programm mit seinem eigenen Programmtext als Eingabe nach endlich vielen Schritten

stoppt oder nicht.
Es kann angenommen werden, daß dieses Programm P_S am Ende folgende Ausgaberoutine mit (*stop*) als Boolesche Variable, die die zugehörende Entscheidung enthält:

```
BEGIN
        IF      stop
        THEN
                WRITELN ('Das Programm ist selbststoppend')
        ELSE
                WRITELN ('Das Programm ist nicht selbststoppend')
        END;
END;
```

Dieses Programm kann nun folgendermaßen abgeändert werden:

```
BEGIN
        IF      stop
        THEN    BEGIN
                WRITELN ('Das Programm ist selbststoppend')
                WHILE TRUE DO;
                END;
        ELSE
                WRITELN ('Das Programm ist nicht selbststoppend')
        END;
END;
```

Da wir voraussetzen, daß das Programm P_S selbststoppend ist, können wir uns fragen, ob auch dieses Programm P_T selbststoppend ist. Wenn wir annehmen, daß es selbststoppend ist, so wird die Variable *stop* den Wert True annehmen, so daß die positive Antwort ausgegeben wird. Anschließend gerät das Programm P_T jedoch in eine Endlosschleife, so daß dieses Programm im Widerspruch zur Annahme nicht selbststoppend ist. Nehmen wir jedoch an, daß das Programm nicht-selbststoppend ist, so erhält die Variable *stop* den Wert False und die negative Antwort wird ausgegeben und das Programm P_T terminiert im Widerspruch zu unserer Annahme. Folglich ist das Programm weder selbststoppend noch nicht-selbststoppend. Dieser logische Widerspruch kann nur dadurch gelöst werden, daß unsere Annahme, daß ein selbststoppendes Programm P_S existiert falsch ist, womit die Unlösbarkeit des Selbstanwendungsproblems bewiesen wird. [Bre 90]□

Aufbauend auf die Erkenntnisse aus dem Selbstanwendungsproblem kann auf die Unlösbarkeit des Halteproblems für PASCAL-Programme im Unterrichtsgespräch eingegangen werden:

Gibt es ein PASCAL-Programm, mit dessen Hilfe man für jedes beliebige PASCAL-Programm entscheiden kann, ob es mit jeder beliebigen Eingabe nach endlich vielen Schritten abbricht oder nicht ?

3.3.2 Konsequenzen des Halteproblems

Dadurch, daß das Halteproblem nicht-algorithmisiert werden kann, sind auch die folgenden Entscheidungsproblem nicht-algorithmisierbar, für die es bis heute keine Lösung gibt:

- die Vermutung von Fermat (s. Abschnitt 2)
- die Vermutung von Goldbach (s. Abschnitt 2)
- Es gibt kein Programm das entscheidet, ob zwei beliebige Programme und gleicher Eingabe dieselbe Ausgabe liefern.
- der $3A + 1$-Algorithmus[Bre 90]
  Gegeben sei folgender Algorithmus zur Erzeugung einer Zahlenfolge:

  1. *Starte mit einer beliebigen natürlichen Zahl A.*
  2. *Wenn A=1, dann stoppe.*
  3. *Wenn A gerade ist, ersetze A durch* $A/2$ *und gehe zu 2.*
  4. *Wenn A ungerade ist, ersetze A durch* $3A + 1$ *und gehe zu 2.*

  Stoppt der Algorithmus für jeden Startwert nach endlich vielen Schritten?

Insbesondere der $3A + 1$-Algorithmus ist leicht zu implementieren und durch verschiedenen Startwerten durchführbar. Anhand dieser Beispiele können die im Unterrichtsgespräch gewonnenen Erkenntnisse aus dem Halteproblem vertieft werden indem hierzu Struktogramme bzw. Programmablaufpläne erstellt werden.

Literaturverzeichnis

[Bre 90] N. Breier,
Grenzen des Computereinsatzes, Ein Unterrichtsbeispiel,
LOG IN **10** (1990) Heft 3, 37-42

[Bre 91] N. Breier,
Berechenbarkeit und Entscheidbarkeit, Grundlagen für den Unterricht,
LOG IN **11** (1991) Heft 3, 29-35

[Chu 36] A. Church,
An unsolvable problem of elementary number theory,
In: Amer. J. Math.,**58** (1936), 345-363

[Güt 92] R.H. Güting,
Datenstrukturen und Algorithmen,
B.G. Teubner Stuttgart (1992)

[Sch 96] N. Schirra,
Quantifizierbares Chaos,
In: Praxis der Naturwissenschaften, Physik, **45**, Heft 1, 12 - 17 (1996)

[Sch 93] R. Schössler,
Nichtlineares Verhalten im Reellen und Komplexen,
Diplomarbeit, FernUniversität Hagen 1993

[Weg 96] I. Wegener,
Kompendium Theoretische Informatik - eine Ideensammlung
B.G. Teubner Stuttgart (1996)

[Wei 90] K. Weihrauch et al.,
Einführung in die Theoretische Informatik A,
Kurs 1653 der FernUniversität Hagen, (1990,1995)