Arno Schwarz

Prinzipielle Grenzen der Berechenbarkeit

Lange Zeit haben Philosophen und Mathematiker geglaubt, dass jedes mathematische Problem algorithmisch lösbar sei. So vertrat z.B. David Hilbert (1862- 1943) die Auffassung,

dass ein jedes bestimmte mathematische Problem einer strengen Erledigung notwendig fähig sein müsse, sei es, dass es gelingt, die Beantwortung der gestellten Frage zu geben, sei es, dass die Unmöglichkeit seiner Lösung und damit die Notwendigkeit des Misslingens aller Versuche dargetan wird.

Erst in unserem Jahrhundert erkannte man, dass nicht alle Probleme, die sich computergerecht spezifizieren lassen, auch algorithmisch lösbar sind. Ernüchternd war vor allem die Erkenntnis, dass die Lösung dieser Probleme nicht an den technischen Unzulänglichkeiten der verfügbaren Geräte scheitert, sondern dass dem menschlichen Denken selbst offenbar Grenzen gesetzt sind.

Entscheidungsprobleme in Pascal

Beispiel: Das Goldbach-Problem (C. Goldbach, 1742)
Eines der bekanntesten bis heute ungelösten Probleme der Mathematik ist die sogenannte Goldbachsche Vermutung:
Jede gerade Zahl >= 4 lässt sich als Summe zweier Primzahlen darstellen.

```
 4 = 2 + 2
 6 = 3 + 3
 8 = 5 + 3
10 = 7 + 3
```

Bezeichnung:
Eine gerade Zahl n >= 4 hat die *Goldbach-Eigenschaft*, wenn es eine Zahl k < n gibt mit :

k und n-k sind Primzahlen

Für die Informatik reduzieren wir das Goldbach-Problem auf die Fragestellung:
Gibt es einen Algorithmus, der für eine gegebene gerade Zahl n >= 4 entscheidet, ob sie die Goldbach-Eigenschaft hat oder nicht ?

Es gibt ihn:

Algorithmus: Test auf Goldbach-Eigenschaft

```
Eingabe : n {gerade Zahl >= 4}
FÜR k := 2 BIS n DIV 2 WIEDERHOLE
  WENN Prim(k) UND Prim(n-k)
    DANN Ausgabe : "ja"
    SONST Ausgabe : "Nein"
```

```
FUNKTION Prim (Zahl : integer) : boolean
    WENN Zahl = 2
      DANN Prim := TRUE
      SONST
        {Zahl ist Primzahl genau dann ,wenn Zahl zwischen
         2 und Wurzel aus Zahl keine Teiler besitzt}
        P := TRUE
        i := 2
        WIEDERHOLE
          WENN Zahl MOD i = 0 DANN P := FALSE
          inc( i)
          BIS (NOT P) ODER i > Wurzel aus Zahl
      ENDE_SONST
    Prim := P
```

Die Goldbach-Eigenschaft ist also entscheidbar. Damit ist das Goldbach-Problem für den Informatiker gelöst, aber die Goldbachsche Vermutung lässt sich damit weder beweisen noch widerlegen.

Definition:
Eine Menge M natürlicher Zahlen heißt *entscheidbar*, wenn es einen Algorithmus gibt, der für jede natürliche Zahl feststellt, ob sie Element von M ist oder nicht. Dieser Algorithmus heißt dann ein *zweiseitiges Entscheidungsverfahren*. Ist die Menge M durch eine Eigenschaft gegeben, so nennen wir diese Eigenschaft *entscheidbar*.

Bezeichnung:
Die Funktion

g : N ---> N,
g(n) := 1/2 n , falls n gerade und
g(n) := 3 n + 1 , falls n ungerade

heißt *Collatz-Funktion* (L. Collatz, um 1930).

Bezeichnung:
Eine natürliche Zahl n heißt *wundersam*, wenn es eine natürliche Zahl k gibt mit :

g^k (n) = 1.

Zum Beispiel ist die Zahl 15 wundersam, denn :

15 ---> 46 ---> 23 ---> 70 ---> 35 ---> 106 ---> 53 ---> 160 ---> 80 --->
40 ---> 20 ---> 10 ---> 5 ---> 16 ---> 8 ---> 4 ---> 2 ---> **1**

Lothar Collatz stellte die Vermutung auf, dass jede natürliche Zahl wundersam ist. Auch diese Vermutung ist bis heute weder bewiesen noch widerlegt. Analog zum Goldbach-Problem schwächen wir die Fragestellung für die Informatik ab:

Gibt es einen Algorithmus, der für eine gegebene natürliche Zahl n entscheidet, ob sie wundersam ist oder nicht ?

Naheliegend ist folgender

Algorithmus: Test auf Wundersamkeit

```
Eingabe : n
SOLANGE n > 1 WIEDERHOLE
  WENN n gerade DANN n := n DIV 2 SONST n := 3 * n + 1
ENDE_SOLANGE
Ausgabe : "Wundersam"
```

Dieser Algorithmus stellt aber kein zweiseitiges Entscheidungsverfahren dar: Ist n wundersam, so liefert dieser Algorithmus nach endlich vielen Schritten die Ausgabe "Wundersam". Gibt man aber eine "unwundersame" Zahl ein, kommt der Algorithmus zu keinem Ende und damit zu keiner Entscheidung. Allerdings hat bis heute noch niemand eine unwundersame Zahl gefunden.

Definition:
Eine Menge M natürlicher Zahlen heißt *partiell entscheidbar*, wenn es einen Algorithmus gibt, der für jede natürliche Zahl feststellt, ob sie Element von M ist. Dieser Algorithmus heißt dann ein *einseitiges Entscheidungsverfahren*. Ist die Menge M durch eine Eigenschaft gegeben, so nennen wir diese Eigenschaft *partiell entscheidbar*. Ein zweiseitiges Entscheidungsverfahren für die Eigenschaft "Wundersam" ist bisher nicht bekannt.

Anmerkung:
Jede entscheidbare Eigenschaft ist auch partiell entscheidbar.

Beispiel: Game of Life (J. Conway)
Gegeben ist ein zweidimensionales Feld aus n Zeilen und m Spalten, dessen Komponenten Zellen genannt werden. Jede Zelle besitzt zwei Zustände (lebend, tot). Die zu Beginn lebenden Zellen bilden die Anfangsgeneration. Eine Folgegeneration entsteht nach folgenden Regeln :

> Eine tote Zelle wird lebendig, wenn drei ihrer acht Nachbarn leben.
> Eine Zelle stirbt, wenn mindestens vier Nachbarn leben (Übervölkerung).
> Eine Zelle stirbt, wenn höchstens ein Nachbar lebt (Vereinsamung).

Gesucht ist ein Algorithmus, der für eine beliebige gegebene Anfangsgeneration des Lebensspiels entscheidet, ob diese unsterblich ist oder nicht.
Der Algorithmus könnte so aussehen, dass er den Lebenslauf der Anfangsgeneration Schritt für Schritt nachvollzieht. Stirbt sie, so ist die Sterblichkeit festgestellt. Stirbt sie aber nicht, gelangt der Algorithmus zu keinem Ende und damit auch zu keiner Entscheidung. Der Algorithmus stellt also nur ein einseitiges Entscheidungsverfahren dar. Im Gegensatz zum Problem der Wundersamkeit ist jedoch für das Game of Life inzwischen bewiesen, dass kein zweiseitiges Entscheidungsverfahren existiert. Die Sterblichkeit einer Anfangsgeneration im Game of Life ist also zwar partiell entscheidbar, aber nachgewiesenermaßen nicht entscheidbar.

Die Algorithmen zu den beiden voranstehenden Beispielen können für bestimmte Eingabewerte in Endlos-Schleifen eintreten. Das ist der Grund dafür, dass es sich bei diesen Algorithmen nur um einseitige Entscheidungsverfahren handelt. Einen Hilfsalgorithmus, der zum Beispiel im Game of Life für beliebige Anfangsgenerationen im voraus eine Endlosschleife entdeckt, gibt es nachweisbar nicht.
Wie man solche Nachweise führt, soll an folgendem Satz gezeigt werden:

SATZ VON TURING (eingeschränkt für Pascal-Programme):
Es gibt kein Pascal-Programm, das für ein beliebiges Pascal-Programm P und eine beliebige Eingabe x die Frage beantwortet, ob P auf x angewendet nach endlich vielen Schritten terminiert oder nicht.
D.h. Die Halte-Eigenschaft für Pascal-Programme ist unentscheidbar und nur partiell entscheidbar.

Beweis:
Annahme: Es gibt es eine boolesche Funktion "Haelt", die folgendes leistet:

> Ist P ein Pascal-Programm und x eine Zeichenfolge, so liefert Haelt nach Eingabe von P und x den Wert TRUE, wenn P auf x angewendet nach endlich vielen Schritten hält, andernfalls den Wert FALSE.

```
FUNCTION Haelt( P, x : Text) : boolean;
   BEGIN
     IF {P terminiert bei Eingabe von x}
        THEN Haelt := TRUE
        ELSE Haelt := FALSE;
   END;
```

Diese Funktion wird nun in ein Programm "Seltsam" eingebettet:

```
PROGRAM Seltsam;
   FUNCTION Haelt(P, x : Text) : boolean;
   {wie oben }
   BEGIN
     {Lies Programm P}
     WHILE Haelt(P, P) DO; {Endlosschleife}
     writeln('Fertig !');
   END.
```

Das heißt:
Hält das Programm P, wenn ihm sein eigener Quelltext eingegeben wird, so begibt sich das Programm "Seltsam" in eine Endlosschleife, andernfalls hält es an. Da P beliebig ist, kann insbesondere auch P = "Seltsam" sein. In diesem Fall gilt :

- Ist Haelt(Seltsam, Seltsam) = TRUE, so begibt sich "Seltsam" in eine Endlosschleife.
- Ist Haelt(Seltsam, Seltsam) = FALSE, so terminiert "Seltsam".

D.h. Das Programm "Seltsam" terminiert bei der Eingabe seines eigenen Quelltextes genau dann, wenn es nicht terminiert.
Die obige Annahme der Existenz der Funktion "Haelt" führt demnach zu einem WIDERSPRUCH !

Zusammenfassung:
Ein Entscheidungsproblem wird durch die Angabe einer Menge M und einer Eigenschaft E, die für die Elemente von M definiert ist, spezifiziert. Ein Algorithmus, der für ein beliebiges Element x der Menge M entscheidet, ob x die Eigenschaft E hat oder nicht, heißt *zweiseitiges Entscheidungsverfahren*. Wenn ein solches Entscheidungsverfahren existiert, heißt die Eigenschaft E *entscheidbar*, sonst *unentscheidbar*. Im letzen Fall heißt das Entscheidungsproblem *algorithmisch unlösbar*.

Ist ein Entscheidungsproblem lösbar, so muss das Problem im Sinne der Mathematik noch lange nicht gelöst sein (vergl. Goldbach-Problem).

Unentscheidbarkeit

Wir betrachten das Halteproblem nun losgelöst von der speziellen Programmiersprache Pascal.

Beispiel: Das Halteproblem
Gibt es einen universellen Algorithmus, der einen beliebigen Algorithmus untersucht und die Frage beantwortet, ob er bei der Eingabe beliebiger Daten in eine Endlosschleife gerät oder nicht?

Zur Beantwortung dieser Frage ohne Bezug auf eine höhere Programmiersprache und sogar ohne Bezug auf ein bestimmtes Computermodell muss man das Arbeitsprinzip eines Computers in einem theoretischen Modell erfassen. Dieses Modell ist die sog. *Turing-Maschine*.
Alan Turing konzipierte 1936, zehn Jahre vor Fertigstellung des ersten elektronischen Computers (ENIAC), auf dem Papier eine "rechnende Maschine", um Aussagen darüber zu gewinnen, was eine solche Maschine prinzipiell leisten kann. Ausgangspunkt seiner Überlegungen war die Forderung, dass die Maschine Algorithmen abarbeiten kann.

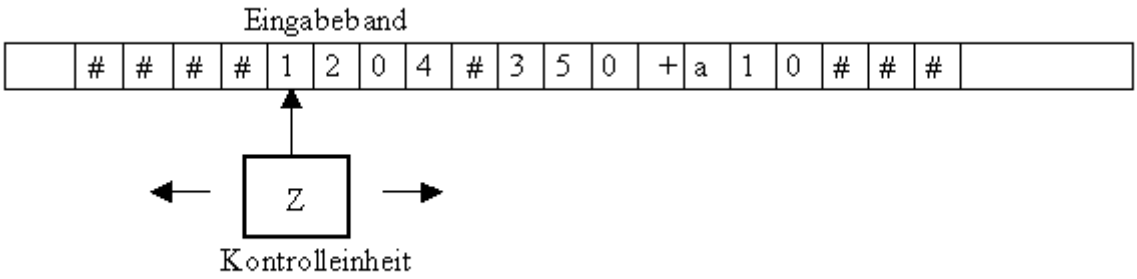
Intuitiver Algorithmusbegriff
Ein Algorithmus ist eine Folge von Handlungsanweisungen zur Lösung eines Problems, die folgende Anforderungen erfüllt:

- Allgemeingültigkeit
- Ausführbarkeit
- Eindeutigkeit
- Endlichkeit

Turing analysierte, was beim Abarbeiten eines Algorithmus geschieht und reduzierte dies auf das Wesentliche:

Die Maschine liest Zeichen von einem Eingabemedium und schreibt Zeichen auf ein Ausgabemedium, und zwar nur endlich viele, da der Algorithmus endlich ist. Die Zeichen stammen also aus einer endlichen Menge X. Im Prinzip kann die Maschine auf das gleiche Medium schreiben, von dem es liest. Das Medium ist in einzelne Felder aufgeteilt, die jeweils nur ein Zeichen aufnehmen können. Eine eventuell zweidimensionale Anordnung der Felder (z.B. auf einem Bildschirm) kann auf eine eindimensionale (ein Band) reduziert werden. Um die Ausgabe nicht zu beschränken, können bei Bedarf an beiden Enden weitere Felder angefügt werden. Das Medium ist also ein potentiell unendlich langes Arbeitsband, das von einem Lese-/Schreibkopf abgetastet wird.
Zu jedem Zeitpunkt sind fast alle Felder leer, d.h. alle bis auf endlich viele, sind mit einem Leerzeichen (#) beschriftet. Der Algorithmus lässt sich in eine Folge elementarer Operationen auflösen.
Eine solche besteht im Lesen und Beschreiben eines einzelnen Feldes. Es genügt dabei, dass die Maschine zu jedem Zeitpunkt nur ein einziges Feld, das Arbeitsfeld, überblickt.

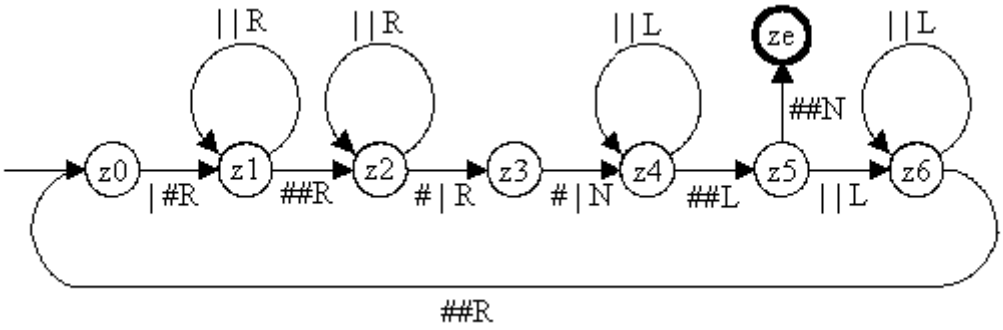
Informationen aus mehreren Feldern lassen sich nämlich zu einem einzigen neuen Zeichen zusammenfassen.
Eine weitere elementare Operation besteht darin, anschließend das links oder rechts benachbarte Feld zum neuen Arbeitsfeld zu machen, oder das alte Arbeitsfeld beizubehalten.
Welche Operation die Maschine ausführt, kann von Zwischenergebnissen abhängen, d.h. sie verfügt über ein Gedächtnis. Zu jedem Zeitpunkt wird ihr daher ein innerer Zustand zugeordnet. Da der Algorithmus endlich ist, benötigt die Maschine nur eine endliche Zustandsmenge Z. Darin muss genau ein Anfangszustand und mindestens ein Endzustand ausgezeichnet sein.
Da der Algorithmus eindeutig ist, muss die Maschine zu jedem Zeitpunkt wissen, welche Operation als nächste auszuführen ist. Das bedeutet:
In Abhängigkeit vom momentanen Zustand z und dem gelesenen Zeichen x führt die Maschine einen Arbeitsschritt aus, indem sie das gelesene Zeichen durch x' überschreibt, in den Nachfolgezustand z' übergeht und den Lese-/Schreibkopf um ein Feld nach links (L) oder nach rechts (R) oder gar nicht (N) bewegt.
Ein solcher Arbeitsschritt lässt sich formal durch eine Übergangsfunktion beschreiben : f(z, x) = (x', k, z') , wobei k = L, R, N.



Definition (Alan Turing, 1936):
Eine *Turing-Maschine* ist ein 7-Tupel T = (X, B, Z, f, #, z0 , Ze) mit folgenden Eigenschaften:

- X ist eine nichtleere, endliche Menge, das *Eingabealphabet*,
- B ist eine nichtleere, endliche Menge, das *Bandalphabet*. Dabei ist X eine Teilmenge von B.
- Z ist eine nichtleere, endliche Menge, die *Zustandsmenge*,
- f : Z\Ze x B ---> B x {L, R , N} x Z ist eine Funktion, die *Übergangsfunktion*, welche jedem Paar (Zustand, gelesenes Zeichen) ein Tripel (zu schreibendes Zeichen, Kopfbewegung, Folgezustand) zuordnet,
- # aus B\X ist das *Leerzeichen*,
- z0 ist der *Anfangszustand*,
- die Teilmenge Ze von Z ist die Menge der *Endzustände*.

Beispiel: Turingverdoppler
Gegeben sei eine Folge von n Strichen auf dem Arbeitsband. Die nachfolgend beschriebene Turing-Maschine verdoppelt die Anzahl der Striche. Dazu wird die gegebene Strichfolge abgetastet. Jeder gefundene Strich wird gelöscht, dafür werden rechts neben der Folge zwei Striche angefügt.
Der Turing-Verdoppler läßt sich durch einen *Zustandsgraphen* veranschaulichen:



Nach der von Alan Turing vorgenommenen Normierung der Abarbeitung eines Algorithmus lässt sich feststellen :
Jedes Problem, das überhaupt maschinell lösbar ist, kann von einer Turing-Maschine gelöst werden. Diese Erkenntnis führt auch zu einer Präzisierung des Algorithmusbegriffs:

Definition:
Unter einem *Algorithmus* versteht man ein Verfahren, das von einer Turing-Maschine ausführbar ist.
Oder kürzer:
Ein *Algorithmus* ist eine Turing-Maschine.

TURING-THESE
Der intuitive Begriff des Algorithmus und der präzise Begriff des Algorithmus stimmen überein.

Nun wenden wir uns wieder dem Halteproblem zu:

SATZ VON TURING:
Es gibt keine Turing-Maschine (Algorithmus), die für eine beliebige Turing-Maschine T und eine beliebige Bandinschrift x die Frage beantwortet, ob T auf x angewendet nach endlich vielen Schritten terminiert oder nicht.
D.h. Die Halte-Eigenschaft ist prinzipiell unentscheidbar.

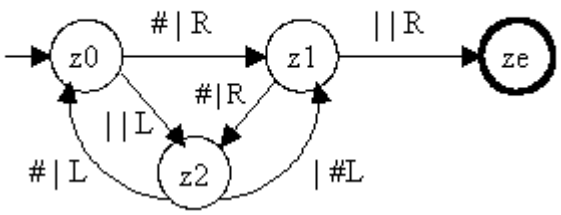
Der Beweis wird analog zum Satz von Turing für Pascal-Programme geführt und ist ausführlich z.B. in Gasper/Leiß/Spengler/Stimm, Technische und theoretische Informatik, bsv-Verlag, 1992 dargestellt.

Unberechenbarkeit

Nach der Betrachtung der Unentscheidbarkeit des Halteproblems soll abschließend der Begriff der Unberechenbarkeit am Beispiel einer speziellen Funktion vorgestellt werden.

Definition: (T. Rado, 1962)
Es sei T eine Turing-Maschine mit dem Eingabealphabet X = {|} und dem Bandalphabet B = {#, |}. Ihre Zustandsmenge Z enthalte genau einen Endzustand ze und n weitere Zustände. Dann heißt die Maschine T ein *Biber mit n Zuständen*, wenn T angesetzt auf das leere Arbeitsband nach endlich vielen Schritten hält.
Ein Biber mit n Zuständen, der die maximale Anzahl von Strichen auf das leere Arbeitsband schreibt, heißt ein *fleißiger Biber mit n Zuständen*.
Die Zahl seiner erzeugten Striche wird mit bb(n) (busy-beaver) bezeichnet.
Die Funktion bb : N ---> N, n ---> bb(n) heißt *Rado-Funktion*.

Beispiel:
Dieser Zustandsgraph beschreibt einen fleißigen Biber mit 3 Zuständen, der 6 Striche auf dem leeren Eingabeband erzeugt und dann anhält.



Gibt es einen Algorithmus zur Berechnung von bb(n)?

Naheliegend ist der folgende Algorithmus:

Algorithmus : Rado-Funktion
Eingabe : n
Notiere alle Turing-Maschinen mit X = {|} und B = {#, |} die genau einen Endzustand und n weitere Zustände besitzen.
Sondere alle nicht haltenden Maschinen aus.
Starte die haltenden Maschinen (Biber mit n Zuständen) auf dem leeren Arbeitsband und notiere die Strichzahlen.
Bilde bb(n) := Maximum der Strichzahlen.

Satz:
Für jedes n aus N ist die Anzahl aller Turing-Maschinen mit X = {|} und B = {#, |}, die genau einen Endzustand und n weitere Zustände haben, gleich (6*(n+1)) ^(2n) .

Beweis:
Die Übergangsfunktion einer solchen Maschine f : Z\Ze x B ---> B x {L, R , N} x Z ist wegen | Z\ Ze | = n und | B | = 2 durch genau 2n Zuordnungen der Form f(z, x) = (x', k, z') gegeben.
Wegen | Z | = n+ 1 sind für jede dieser Zuordnungen 2*3*(n+ 1) = 6*(n+ 1) verschiedene Funktionswerte möglich.
Daher gibt es für jedes n aus N insgesamt (6*n+1)) ^(2n) Übergangsfunktionen.

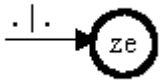
Die folgende Tabelle zeigt in Abhängigkeit von n die Anzahl der zu untersuchenden Turing-Maschinen sowie die bisher ermittelten Funktionswerte der Rado-Funktion.

n	Anzahl der Maschinen	bb(n)
1	144	1
2	104976	4
3	1,9*10^8	6
4	6,6*10^11	13
5	3,7*10^15	?

1984 fand man einen Biber mit 5 Zuständen, der 1915 Striche schreibt. Dieser Rekord wurde 1989 mit 4098 Strichen gebrochen. Ob dies der fleißige Biber mit 5 Zuständen ist, konnte bisher nicht bewiesen werden. Erst recht sind die Funktionswerte der Rado-Funktion für n > 5 nicht bekannt.

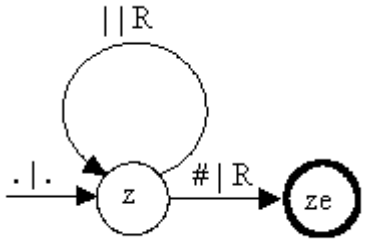
Satz:
Die Rado-Funktion ist streng monoton steigend.

Beweis:
Es sei n aus N beliebig vorgegeben und F der fleißige Biber mit n Zuständen, der bb(n) Striche erzeugt. Wie der Zustandsgraph dieses Bibers endet, zeigt die nebenstehende Abbildung:



Dabei steht der erste Punkt der Kantenbeschriftung für ein | oder ein # und der zweite Punkt für eine Kopfbewegung.

Dann konstruiert man daraus folgendermaßen einen Biber mit n+1 Zuständen, der bb(n)+1 Striche schreibt:



Ein fleißiger Biber mit n+1 Zuständen schreibt also mindestens bb(n)+1 Striche.

Anmerkung:
Das Wachstum der Rado-Funktion ist sogar stärker als das jeder Exponentialfunktion. So wurde z.B. für bb(8) bereits eine untere Schranke von 10^43 ermittelt.

Das Problem bei der Suche nach fleißigen Bibern liegt im Aussondern aller nicht haltenden Maschinen. Nach dem Satz von Turing gibt es keinen Algorithmus, der für eine beliebige Turing-Maschine T und eine beliebige Bandinschrift x die Frage beantwortet, ob T auf x angewendet nach endlich vielen Schritten terminiert oder nicht. Da aber die Menge der Turing-Maschinen bei der Suche nach fleißigen Bibern stark eingeschränkt ist, liefert der Satz keine Begründung dafür, dass ein Algorithmus zur Erkennung nichthaltender Maschinen unter den hier betrachteten nicht existiert. Außerdem könnte es sein, dass es einen ganz anderen Algorithmus gibt, der die Rado-Funktion berechnet.
Dass dies nicht der Fall ist, soll im folgenden bewiesen werden.

Definition:
Eine Funktion f: N ---> N heißt *berechenbar*, wenn es einen Algorithmus gibt, der für jede natürliche Zahl n (nach endlich vielen Schritten) den Funktionswert f(n) errechnet.

Anmerkung:
Da die Menge aller Turingmaschinen abzählbar unendlich ist, die Menge aller Funktionen f: N --->N nicht abzählbar unendlich ist, muss es offenbar nicht berechenbare Funktionen der Form f: N --> N geben.
Eine dieser Funktionen ist die Rado-Funktion.

Satz von Rado:
Die bb-Funktion ist nicht berechenbar.

Beweis:
Angenommen, es gibt einen Algorithmus, der für jede natürliche Zahl n den Wert der Rado-Funktion bb(n) berechnet. Das heißt, es gibt eine Turing-Maschine B mit endlich vielen Zuständen, die angesetzt auf das Eingabeband, auf dem die Zahl n steht, nach endlich vielen Schritten die Zahl bb(n) auf dem Eingabeband hinterläßt. Die Anzahl der Zustände der Turingmaschine B sei gleich m.

Speziell sei nun n eine Zahl der Form 2^k-1, k aus N. Da alle Zahlsysteme gleichwertig sind, werde n von einer Turingmaschine A in Binärdarstellung auf das Band geschrieben. Dabei werde das Zeichen | für das Zeichen 1 verwendet. Da n von der angegebenen besonderen Form ist, muss A dazu genau k Striche auf dem leeren Eingabeband erzeugen. Der Zustandsgraph von A lautet:

Die Maschine A ist also ein Biber mit k = log n Zuständen, der k Striche erzeugt.
Die Maschine B sei so beschaffen, dass sie die Zahl n in Binärdarstellung auf dem Eingabeband liest und nach dem Halten die Zahl bb(n) in Unärdarstellung auf dem Eingabeband zurücklässt.

Schaltet man die beiden Maschinen A und B hintereinander und setzt sie auf das leere Eingabeband an, so schreibt die Maschine AB genau bb(n) Striche auf das Band und hält dann an. Das heißt, die Maschine AB ist ein Biber mit log n + m Zuständen, der bb(n) Striche erzeugt, also genauso viele Striche, wie der fleißige Biber mit n Zuständen.
Ist aber n groß genug, dann gilt log n + m < n. Das bedeutet, dass der Biber AB mit weniger Zuständen als der fleißige Biber genauso viele Striche wie dieser erzeugt. Dies steht im Widerspruch zur strengen Monotonie der Rado-Funktion.

Damit ist nachgewiesen, dass die Rado-Funktion nicht berechenbar ist - und die mühsame Suche nach fleißigen Bibern niemals enden wird.