Elementare Berechenbarkeitstheorie

Christoph Kreitz

Institut für Informatik, Universität Potsdam, 14482 Potsdam

Dieser Artikel ist noch nicht ganz fertig und enthält vor allem noch Tippfehler und kleinere Unsauberkeiten. Er wird im Laufe der nächsten Wochen durch eine leicht überarbeitete Version ersetzt.

Zusammenfassung *Dieser Artikel gibt eine kurze Einführung in die Elementare Berechenbarkeitstheorie. Er dient als Ergänzung der Einheiten 5.4 & 5.5 der Vorlesung "Einführung in die Theoretische Informatik II".*

1 Fundamentale Eigenschaften berechenbarer Funktionen

Die Untersuchung der verschiedenen Berechenbarkeitsmodelle hat gezeigt, daß alle Modelle, mit Ausnahme der Unterklasse der primitiv-rekursiven Funktionen, genauso mächtig sind wie Turingmaschinen. Die Äquivalenzbeweise zeigen, daß die verschiedenen Modelle ineinander übertragbar sind und daher sogar ineinander verzahnt verwendet werden können. Aufgrund der Chuch'schen These geht man daher davon aus, daß Turingmaschinen und äquivalente Modelle genau die intuitiv berechenbaren Funktionen beschreiben. Daher ist es sogar möglich, in Argumenten zur Berechenbarkeit bestimmter Funktionen intuitive Beschreibungen von Algorithmen zu verwenden, anstatt diese Algorithmen im Detail innerhalb eines dieser Modelle auszuprogrammieren.

In diesem Artikel wollen wir uns nun den allgemeingültigen Eigenschaften berechenbarer Funktionen widmen. Es geht also um die Frage, welche Funktionen überhaupt berechenbar sind; welcher Zusammenhang zwischen Berechenbarkeit, Aufzählbarkeit und Entscheidbarkeit besteht, welche Abschlußeigenschaften für diese Konzepte bewiesen werden können, und wo die Grenzen des Berechenbaren liegen.

Wir wollen uns bei der Beantwortung dieser Fragen soweit wie möglich von den konkreten Berechnungsmodellen lösen, da es um Eigenschaften berechenbarer Funktionen als solche und nicht etwa um Eigenschaften von Turingprogrammen, μ-rekursiven Funktionen, λ-Ausdrücken etc. geht. Daher werden wir im folgenden zunächst einige fundamentale Eigenschaften berechenbarer Funktionen formulieren, die unabhängig vom konkreten Berechnungsmodell sind, und dann unter Zuhilfenahme des Modells der Turingmaschine beweisen. Alle weiteren Definitionen, Sätze und Beweise werden sich dann nur noch auf diese Grundeigenschaften stützen und keine Referenzen auf konkrete Berechnungsmodelle verwenden. Diese Vorgehensweise führt zu Formulierungen von Aussagen, die wesentlich abstrakter sind, als dies in den meisten Lehrbüchern heutzutage üblich ist. Sie sind aber auch erheblich präziser und führen zu eleganteren Beweisen, mit denen wir in wenigen Zeilen das ausdrücken können, was in Beweisen, die sich auf konkrete Maschinenmodelle beziehen müssen, mehrere Seiten in Anspruch nimmt.

Wir werden zeigen, daß man *alle berechenbaren Funktionen durchnumerieren* und *mithilfe eines einzigen universellen Programms berechnen* kann, wenn dieses als Eingabe die Programmnummer und die entsprechenden Eingaben erhält. Man kann die *Komplexität von Programmen entscheiden*, also ob ein bestimmtes Programm bei Eingabe gewisser Daten nach einer vorgegebenen Anzahl von Schritten terminiert, *Programme effektiv kombinierenn*, also aus den einzelnen Programmnummern die Nummer der Funktion bestimmen, die von der Kombination dieser Programme berechnet wird.

Wir werden diese vier Eigenschaften beispielhaft mit dem Modell der Turingmaschine beweisen, aber sie gelten in gleicher Weise für jedes andere Berechenbarkeitsmodell. Wir konzentrieren uns in unseren Betrachtungen auf berechenbare Funktionen über den natürlichen Zahlen. Alle anderen Berechenbarkeitskonzepte können hieraus abgeleitet werden.[1] So läßt sich die Entscheidbarkeit und Semi-Entscheidbarkeit von Mengen als Berechenbarkeit der zugehörigen (partiellen) charakteristischen Funktion beschreiben. Berechenbarkeitskonzepte auf Wörtern können durch Codierungen von Wörtern als Zahlen und berechenbare Funktionen auf Zahlen ausgedrückt werden und Funktionen auf Zahlenpaaren und -listen können mithilfe der Standardtupelfunktion einstellig simuliert werden. Für eine allgemeine Theorie der Berechenbarkeit reicht daher die Betrachtung von *einstelligen berechenbaren Funktionen über den natürlichen Zahlen*.

1.1 Die Standardnumerierung der berechenbaren Funktionen

Berechenbare Funktionen können beschrieben werden als Funktionen, deren Ein-/Ausgabeverhalten von Turingmaschinen berechnet wird. Eine naheliegende Numerierung der berechenbaren Funktionen ergibt sich also direkt aus einer Numerierung der entsprechenden Turingmaschinen. Turingmaschinen kann man wiederum mit dem Text identifizieren, der das zugehörige Turingprogramm, also die Zustandsüberführungsfunktionen, beschreibt. Zählt man nun alle Wörter über dem entsprechenden "Programmalaphabet" auf und überspringt dabei die Wörter, die keine korrekten Turingprogramme darstellen, so erhält man die gewünschte Numerierung. Wir wollen nun einige Details dieser Vorgehensweise etwas genauer ausführen.

Für die Codierung von Turingmaschinen als Wörter über einem festen Alphabet reicht es aus, Turingmaschinen mit dem Ein-/Ausgabealphabet $\Sigma = \{0, 1\}$, Bandalphabet $\Gamma = \{0, 1, B\}$, Anfangszustand q_0 und der festen Endzustandsmenge $F = \{q_1\}$ zu betrachten. Dieses eingeschränkte Modell ist bekanntermaßen genauso mächtig wie Turingmaschinen mit mehr Freiheitsgraden, hat aber den Vorteil, daß jede Turingmaschinen eindeutig durch ihre Zustandsüberführungsfunktion definiert ist.

[1] Eine Reihe von Lehrbüchern setzt den Schwerpunkt eher auf berechenbare Funktionen über Worten, weil dies der Idee der Eingabe von Programmen und Daten als Bitketten näher kommt. Zuweilen wirkt dies intuitiver, weil man von Programmen anstatt von Programmnummern reden kann. Dafür wird die Behandlung von Arithmetik und der Rechenzeit komplizierter, da Zahlen durch Worte beschrieben werden müssen. Da Zahlen und Wörter bijektiv ineinander übersetzt werden können, sind beide Ansätze im Endeffekt gleichwertig. Es ist eher eine Frage der persönlichen Präferenz, mit welchem Ansatz man besser zurecht kommt.

Definition 1 (Codierung von Turingmaschinen).
Es sei $M = (\{q_0, .., q_n\}, \Sigma, \Gamma, \delta, q_0, B, \{q_1\})$ *eine beliebige Turingmaschine über dem Bandalphabet* $\Gamma = \{0, 1, B\}$.

- *Die Codierung der Zustandsüberführungsfunktion* δ *an der Stelle* (q,X) *ist definiert durch* $code(\delta(q,X)) \equiv \begin{cases} q\,X\,p\,Y\,D & falls\ \delta(q,X) = (p,Y,D) \\ \epsilon & sonst \end{cases}$
- *Die Codierung der Zustandsüberführungsfunktion von* M *ist definiert durch* $code(\delta) \equiv code(\delta(q_0,0))\ code(\delta(q_0,1))\ code(\delta(q_0,B)) \ldots code(\delta(q_n,B))$
- *Die Codierung der Turingmaschine* M *über dem Alphabet* $\Delta = \{0, 1\}$ *ist definiert durch* $code(M) \equiv \rho_n(code(\delta))$, *wobe* ρ_n *eine Standardrepräsentation der Menge* $\{q_0, .., q_n, 0, 1, B, L, R\}$ *durch Wörter über dem Alphabet* Δ *ist.*

Die Nummer einer Turingmaschine kann damit über die Nummer des entsprechenden Wortes $\rho_n(code(\delta))$ in der Menge der Wörter über $\{0, 1\}$ bestimmt werden. Hierzu müssen wir die Wörter über $\{0, 1\}$ in eine kanonische Reihenfolge bringen und dann durchzählen. Üblicherweise verwendet man hierzu die sogenannte lexikographische Ordnung, die für Wörter über einem Alphabet $\{x_1, .., x_n\}$ wie folgt erklärt ist.

$\epsilon < x_1 < \ldots < x_n < x_1x_1 < x_1x_2 < \ldots < x_nx_n < x_1x_1x_1 < \ldots$

Zählt man Wörter in dieser Reihenfolge entsprechend durch, so ergibt sich

$w_0{:=}\epsilon,\ w_1{:=}x_1, .. w_n{:=}x_n,\ w_{n+1}{:=}x_1x_1, \ldots$

Definition 2 (Ordnung und Numerierung von Wörtern).
Es sei $\Delta = \{x_1, .., x_n\}$ *ein beliebiges Alphabet*

- *Die Standardordnung auf* Δ *ist definiert durch* $x_i <_\Delta x_j \Leftrightarrow i<j$
- *Die lexikographische Ordnung* $<$ *auf* Δ^* *ist definiert durch* $u < v$, *falls* $|u| < |v|$ *oder* $|u| = |v| = m$ *und* $\exists k{\leq}m\,(u_k <_\Delta v_k \wedge \forall i{<}k\ u_i{=}v_i)$
- *Der Index eines Wortes* $w \in \Delta^*$ *ist seine Position in der lexikographischen Ordnung.*

Für die Numerierung von Turingmaschinen können wir uns auf das Alphabet $\{0, 1\}$ konzentrieren.

Definition 3 (Numerierung von Turingmaschinen).
Es sei $TM_{\{0,1\}}$ *die Menge aller Wörter* $\{w \in \{0, 1\}^*$, *die eine Codierung* $code(M)$ *einer Turingmaschine* $M = (\{q_0, .., q_n\}, \Sigma, \Gamma, \delta, q_0, B, \{q_1\})$ *über dem Bandalphabet* $\Gamma = \{0, 1, B\}$ *darstellen.*

- M_i *ist die Turingmaschine* M *deren Codierung* $code(M)$ *in der lexikographischen Ordnung von* $TM_{\{0,1\}}$ *an* i*-ter Stelle erscheint.*
- *Der Index der Turingmaschine* M *ist die eindeutig bestimmte Zahl* i *mit* $M_i = M$. *Die Zahl* i *wird auch Gödelnummer von* M *genannt.*

Da man leicht testen kann, ob ein Wort über dem Alphabet $\{0, 1\}$ die Codierung einer Turingmaschine gemäß Definition 1 darstellt, kann man ein Programm schreiben, das die Wörter aus $TM_{\{0,1\}}$ der Reihe nach aufzählt, und somit das Programm der Turingmaschine M_i effektiv aus der Nummer i berechnen. Im Detail ist diese Prozedur relativ aufwendig, besonders wenn man versucht, sie als ein Turingprogramm zu beschreiben, das wir im Beweis von Satz 2 benötigen werden. Da es aber intuitiv klar ist,

wie solch ein Programm konstruiert werden könnte, verzichten wir in diesem Artikel auf die Details und verweisen stattdessen auf das Buch von Weihrauch [1, §1.9].

Für die Numerierung berechenbarer Funktionen benötigen wir nun nur noch die Semantik von Turingprogrammen über dem Alphabet $\{0,1\}$ und eine Codierung der natürlichen Zahlen. Wir bezeichnen die von der Turingmaschine M berechnete Funktion mit $f_M : \{0,1\}^* \to \{0,1\}^*$ und die Binärcodierung der natürlichen Zahlen im Alphabet $\{0,1\}$ mit $r_b : \mathbb{N} \to \{0,1\}^*$. Für Komplexitätsfragen benötigen wir außerdem die Rechenzeitfunktion der Turingmaschine M, die wir mit $t_M : \{0,1\}^* \to \mathbb{N}$ bezeichnen.

Definition 4 (Numerierung berechenbarer Funktionen).

- *Die Standardnumerierung $\varphi : \mathbb{N} \to \mathcal{R}$ der berechenbaren Funktionen auf den natürlichen Zahlen ist definiert durch* $\varphi(i) = r_b^{-1} \circ f_{M_i} \circ r_b$.
- *Die Schrittzahlfunktion $\Phi : \mathbb{N} \to \mathcal{R}$ ist definiert durch* $\Phi(i)(n) = t_{M_i}(r_b(n))$.

Anstelle von $\varphi(i)$ und $\Phi(i)$ schreiben wir meist $\boldsymbol{\varphi_i}$ bzw. $\boldsymbol{\Phi_i}$. φ_i ist die von M_i berechnete (partielle) Funktion auf $\mathbb{N}$ und Φ_i die zugehörige Schrittzahlfunktion von M_i. Die Nummer i wird auch die *Gödelnummer* oder *Index* der berechenbaren Funktion $f : \mathbb{N} \to \mathbb{N}$ genannt, wenn $f = \varphi_i$ ist.

Per Konstruktion hat jede berechenbare Funktion eine Gödelnummer. Da aber jede berechenbare Funktion unendlich viele Programme hat, die sie berechnen (man muß nur überflüssige Anweisungen ergänzen, die nie genutzt werden), hat jede berechenbare Funktion unendlich viele Indizes. Wir formulieren diese Einsicht als ein Lemma, auf das wir gelegentlich zurückgreifen.

Lemma 1. *$\varphi : \mathbb{N} \to \mathcal{R}$ ist surjektiv, aber nicht bijektiv.*

Die Schrittzahlfunktion Φ_i der von M_i berechneten Funktion φ_i gibt für eine Eingabe $j \in \mathbb{N}$ die Anzahl der Rechenschritte bis zur Terminierung an, wenn $\varphi_i(j)$ überhaupt terminiert. Damit ist Φ_i per Konstruktion auf den gleichen Eingaben definiert wie φ_i. Diese scheinbar unbedeutende Erkenntnis wird später eine der fundamentalen Grundeigenschaften der Funktionen φ und Φ darstellen.

Theorem 1. *Für alle i gilt $domain(\Phi_i) = domain(\varphi_i)$*

1.2 Das UTM Theorem

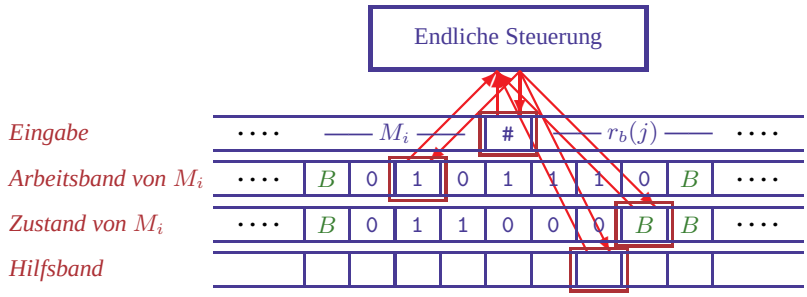
Bisher sind Turingmaschinen nur in der Lage, eine einzige Funktion zu berechnen. Dies bedeutet, daß man für jede Aufgabe eine neue Maschine bauen muß, was aus theoretischer Sicht zwar unproblematisch, aus praktischer Sicht aber völlig indiskutabel ist. Erst in den 1940er Jahren wurden Computer so konzipiert, daß ihr Verhalten programmiert werden kann, und heute ist es eigentlich selbstverständlich, daß man einen Computer als universelles Gerät betrachtet, das Programme und Daten als Eingabe annimmt und dann die von dem Programm zu berechnende Funktion auf den Daten ausführt. Nur in sehr speziellen Anwendungen baut man noch Computer, die nur eine oder einige wenige Funktionen übernehmen.

Die Frage, die hierbei jedoch entsteht ist, ob ein einziger Computer tatsächlich jeden anderen Computer simulieren kann. Können wir also beweisen, daß es eine *universelle*

berechenbare Funktion u gibt, die bei Eingabe einer Gödelnummer i und einer beliebigen Zahl j genau den Wert $\varphi_i(j)$ berechnet? Um dies zu zeigen, müssen wir eine *universelle Turingmaschine* bauen, die auf einem Arbeitsband das Programm von M_i generiert, auf einem anderen die Eingabezahl j verwaltet und dann den Programmcode schrittweise liest und ausführt. Aus diesem Grund wird das Theorem, das wir beweisen wollen, als *UTM Theorem* bezeichnet.

Theorem 2 (UTM Theorem).
Die Funktion $u : \mathbb{N} \to \mathbb{N}$ *mit* $u\langle i, j\rangle = \varphi_i(j)$ *für alle* i, j *ist berechenbar*

Beweis: Wir konstruieren skizzenhaft eine Turingmaschine U, welche die Funktion u berechnet. Hierzu benötigt U mehrere Arbeitsbänder und einige Hilfsbänder.
Auf den ersten Band stehen die Programmnummer i und der Eingabewert j, kombiniert mit der Standard-Tupelfunktion und binär codiert über dem Alphabet $\{0, 1\}$. U berechnet zunächst aus der Eingabe $n = \langle i, j\rangle$ die Zahlen i und j in Binärdarstellung, erzeugt dann das Wort code(δ_i), also die Codierung der Zustandsüberführungsfunktion von M_i, aus der Zahl i und kopiert die Binärdarstellung $r_b(j)$ von j auf das zweite Band. Dieses wird Arbeitsband für die Simulation der Maschine M_i verwendet, enthält also genau den Bandinhalt, den das Arbeitsband der Maschine M_i bei der Verarbeitung von j enthalten würde.



Nun simuliert U die Einzelschritte von M_i. Hierzu sucht die Maschine, abhängig vom gelesenen Symbol auf Band 2 und dem Zustand von M_i, die Codierung des entprechenden Zustandsübergangs auf Band 1 auf, liest dort das zu schreibende Symbol, den neuen Zustand und die Kopfbewegung und führt genau dies aus. Damit dies möglich ist, muß der aktuelle Zustand von M_i auf einem dritten Arbeitsband gelagert sein, da die Anzahl der Zustände aller möglichen Machinen M_i nicht nach oben begrenzt ist, und somit nicht im Zustand von U verwaltet werden kann.
Auf die Art simuliert U Schritt für Schritt das Verhalten von M_i und man kann beweisen, daß nach einer kompletten Simulation eines Programmbefehls das Band 2 identisch mit dem Arbeitsband von M_i in dieser Situation ist und Band 3 genau den entsprechenden Zustand enthält. Somit terminiert U bei Eingabe von $\langle i, j\rangle$ genau dann, wenn M_i bei Eingabe von j terminiert (also den Zustand q_1) erreicht) und Band 2 enthält genau das Ergebnis. Dieses wird nun von Band 2 auf Band 1 kopiert und steht für die Ausgabe bereit.
Es folgt $u\langle i, j\rangle = f_U\langle i, j\rangle = f_{M_i}(j) = phi_i(j)$. □

Eine unmittelbare Konsequenz des obigen Beweises ist, daß es auch eine universelle Methode gibt, zu entscheiden, ob die Berechnung von $\varphi_i(j)$ nach genau t Schritten anhält. Hierzu läßt man die Maschine U die Ausführung von $M_i(j)$ für maximal t Schritte simulieren. Steht zum simulierten Zeitpunkt t der Zustand q_1 auf dem Arbeitsband 3, dann terminiert $M_i(j)$ nach genau t Schritten und ansonsten nicht. Damit haben wir eine weitere wichtige Einsicht beweisen, die wir mit der Schrittzahlfunktion Φ formulieren.

Theorem 3. *Die Menge* $\{\langle i,j,t\rangle \mid \Phi_i(j){=}t\}$ *ist entscheidbar*

1.3 Das smn-Theorem

Eine letzte wichtige Eigenschaften berechenbarer Funktionen ist, daß sie sich nahezu beliebig nach einer Art Baukastenprinzip kombinieren lassen. Man kann neue berechenbare Funktionen generieren, indem man mehrere Funktionen hintereinander ausführt, in einer mehrstelligen Funktion eine Variable festhält, Argumente vertauscht, etc. In jedem dieser Fälle kann die neue Funktion effektiv aus den andere bestimmt werden. In der Denkweise von Programmiersprachen ist dies leicht einzusehen: man nimmt einfach die entsprechenden Teilproramme und schreibt ein Stück Brückencode, um sie miteinander zu verbinden, und erhält das neue Programm. Um zu beweisen, daß diese Idee wirklich zum gewünschten Erfolg führt, müssen wir zeigen, daß sich die Gödelnummern der Programme mithilfe einer berechenbaren Funktion effektiv ineinander umrechnen lassen. Dies führt dann zu einem Theorem, das sehr technisch aussieht und mangels einer intuitiveren Bezeichnung seinen Namen *smn-Theorem* aus den vorkommenden Variablen bezogen hat. Gelegentlich wird es auch als *Übersetzungslemma* bezeichnet.

Theorem 4 (smn Theorem). *Es gibt eine berechenbare totale Funktion* $s{:}\mathbb{N}{\to}\mathbb{N}$ *mit der Eigenschaft* $\varphi_{s\langle m,n\rangle}(i) = \varphi_m\langle n,i\rangle$ *für alle* $m,n,i \in \mathbb{N}$

Beweis: Wir konstruieren skizzenhaft eine Turingmaschine M für die Funktion s. Die Maschine erwartet auf dem Eingabeband die Codierung einer Zahl $\langle m,n\rangle$ und muß daraus die Codierung einer Maschine $M_{s\langle m,n\rangle}$ konstruieren, die bei Eingabe (der Codierung) einer Zahl i dasselbe berechnet wie M_m bei Eingabe von $\langle n,i\rangle$.
Hierzu konstruiert M zunächst auf einem ersten Hilfsband das Wort code(δ_m), also die Codierung der Zuständsüberführungsfunktion von M_m. Auf einem zweiten Hilfsband konstruiert M aus der Zahl n die Codierung einer Maschine $M_{\langle n,.\rangle}$, welche bei Eingabe einer Zahl i den Wert $\langle n,i\rangle$ berechnet.
Auf einem dritten Band konstruiert M dann den Code einer Maschine M', welche bei Eingabe einer Zahl i zunächst die Anwendung des Codes von $M_{\langle n,.\rangle}$ auf i simuliert und anschließend den Code von M_m auf das Ergebnis $\langle n,i\rangle$ anwendet. Damit berechnet M' also genau den Wert $\varphi_m\langle n,i\rangle$ und ihre Gödelnummer ist das gewünschte Ergebnis. Daher berechnet M abschließend die Gödelnummer von M' und gibt dies als Ergebnis zurück. Die von M berechnete Funktion $s{=}f_M$ ist per Konstruktion berechenbar und total, da M für jede Eingabe $\langle m,n\rangle$ ein Ergebnis, die Gödelnummer einer Maschine, produziert. Außerdem gilt
$\varphi_{s\langle m,n\rangle}(i) = f_{M'}(i) = f_{M_m}(f_{M_{\langle n,.\rangle}}(i)) = \varphi_m\langle n,i\rangle$ □

2 Abstrakte Berechenbarkeitstheorie

Im vorhergehenden Abschnitt haben wir vier fundamentale Theoreme über Eigenschaften der Numerierung Turing-berechenbarer Funktionen bewiesen. Dabei ist die konkrete Numerierung der berechenbaren Funktionen eigentlich relativ unbedeutend. Wichtig ist nur, daß es hierzu passend eine berechenbare universelle Funktion, eine Übersetzungsfunktion entsprechend des smn-Theorems und eine Komplexitätsfunktion mit entscheidbarerem Ergebnis gibt.

Man kann die Theorie der Berechenbarkeit daher auch axiomatisch formulieren, also die Existenz einer Numerierung mit den obengenannten Eigenschaften *postulieren* und dann alle weiteren Definitionen und Sätze nur noch auf diese Axiome stützen. Auf diese Art entgeht man der Gefahr, die Formulierung der Theorie von einem bestimmten Maschinenmodell abhängig zu machen und ihr damit ihre Eleganz und Allgemeingültigkeit zu rauben. Die abstrakte Formulierung zeigt, daß alle gewonnenen Erkenntnisse für jede Programmiersprache und jedes Computermodell gelten, das es gibt oder das jemals erfunden wird. Darüber hinaus kann man sich in Beweisen auf die wirklich essentiellen Argumente konzentrieren, anstatt immer wieder aufs neue aufwendige Maschinen zu konstruieren, welche die Kernargumente eigentlich nur verschleiern.

In Anlehnung an unseren Artikel zu den rekursiven Funktionen bezeichnen wir die Menge der möglicherweise partiellen berechenbaren Funktionen auf den natürlichen Zahlen mit $\mathcal{R}$ und Menge der totalen berechenbaren Funktionen mit $\mathcal{TR}$. Dies ermöglicht uns, Formulierungen von Definitionen, Sätzen und Beweisen prägnanter zu gestalten. Statt "$f{:}\mathbb{N}{\to}\mathbb{N}$ *ist berechenbar*" können wir kurz "$f \in \mathcal{R}$" schreiben und "$f \in \mathcal{TR}$" anstelle von "$f{:}\mathbb{N}{\to}\mathbb{N}$ *ist berechenbar und total*". Wenn eine Funktion f an der Stelle x nicht definiert ist, schreiben wir kurz "$f(x) = \perp$ oder "$f(x)\uparrow$" (für "*die Berechnung von* $f(x)$ *divergiert*") und "$f(x)\downarrow$" ("*die Berechnung von* $f(x)$ *terminiert bzw. konvergiert*") oder "$x \in domain(f)$", wenn wir sagen wollen, daß $f(x)$ definiert ist, ohne den konkreten Funktionswert anzugeben.

Definition 5 (Axiome der Berechenbarkeit).
Seien $\varphi{:}\mathbb{N}{\to}\mathcal{R}$ *und* $\Phi{:}\mathbb{N}{\to}\mathcal{R}$. *Die Axiome der Berechenbarkeitstheorie bezüglich* φ *und* Φ *sind definiert durch*

1. φ *ist surjektiv.*
2. *Für alle* i *gilt* $domain(\Phi_i){=}domain(\varphi_i)$.
3. *Es gibt ein* $\chi_\Phi \in \mathcal{TR}$ *mit* $\chi_\Phi\langle i,j,t\rangle = \begin{cases} 1 & falls\ \Phi_i(j) = t \\ 0 & sonst \end{cases}$ *für alle* $i,j,t \in \mathbb{N}$.
4. *Es gibt ein* $u \in \mathcal{R}$ *mit* $u\langle i,j\rangle{=}\varphi_i(j)$ *für alle* i,j.
5. *Es gibt ein* $s \in \mathcal{TR}$ *mit* $\varphi_{s\langle m,n\rangle}(i) = \varphi_m\langle n,i\rangle$ *für alle* $m,n,i \in \mathbb{N}$.

Axiom 1 besagt, daß φ eine *Numerierung aller berechenbaren Funktionen* auf $\mathbb{N}$ ist. Axiom 2 stellt eine enge Beziehung zwischen φ und Φ her, wobei die Intuition ist, daß Φ_i die zu φ_i gehörende Rechenzeitfunktion darstellt. Axiom 3 besagt, daß Rechenzeit entscheidbar ist. Axiom 4 ist das utm-Theorem und postuliert die Existenz einer universellen berechenbaren Funktion. Axiom 5, das smn-Theorem, verlangt daß berechenbare Funktionen effektiv kombiniert werden können. Wir haben im vorigen Abschnitt bewiesen, daß es Funktionen φ und Φ gibt, welche diese fünf Axiome erfüllen.

Für den Rest dieses Artikels nehmen wir an, daß φ:$\mathbb{N}\to\mathcal{R}$ und Φ:$\mathbb{N}\to\mathcal{R}$ zwei beliebige Funktionen sind, welche die Axiome der Berechenbarkeitstheorie erfüllen und werden alle weiteren Definitionen, Sätze und Beweise nur noch auf diese Axiome stützen.

Eine spezielle Form des smn-Theorems wird häufig in Beweisen verwendet.

Korollar 5 *Für jede berechenbare Funktion f gibt es eine berechenbare totale Funktion h mit $\varphi_{h(i)}(n) = f\langle i,n\rangle$ für alle $i,n\in\mathbb{N}$.*

Beweis: Es sei $f{=}\varphi_m\in\mathcal{R}$. Dann gilt $f\langle i,n\rangle{=}\varphi_m\langle i,n\rangle{=}\varphi_{s\langle m,\,i\rangle}(n)$ für alle $i,n\in\mathbb{N}$. Damit erfüllt die Funktion h mit $h(i){=}s\langle m,i\rangle$ die Bedingungen. □

Dieses Korollar hat vielfältige Einsatzmöglichkeiten. So kann man z.B. eine Funktion $h\in\mathcal{TR}$ konstruieren mit $\varphi_{h\langle i,\,j\rangle} = \varphi_i\circ\varphi_j$ oder mit $\varphi_{h\langle i,\,j\rangle}(n) = \varphi_i(n)+\varphi_j(n)$. Wir werden auf diese Möglichkeiten gelegentlich in Beweisen zurückkommen.

Auf Grundlage der Numerierungen φ:$\mathbb{N}\to\mathcal{R}$ und Φ:$\mathbb{N}\to\mathcal{R}$ sind wir nun in der Lage, die Begriffe *Berechenbarkeit*, *Entscheidbarkeit* und *Aufzählbarkeit* neu zu spezifizieren.

- *Berechenbarkeit einer Funktion f* bedeutet, daß man eine Menge von Instruktionen angeben kann, mit denen f berechnet wird. Da sich alle Programme berechenbarer Funktionen aufzählen lassen, ist dies äquivalent zu der Aussage, daß f eine der von φ aufgezählten Funktionen sein muß. Berechenbare Funktionen werden gelegentlich auch als *partiell rekursiv* oder einfach nur *rekursiv* bezeichnet. Eine berechenbare Funktion, die total ist, wird auch *total rekursiv* genannt.
- *Entscheidbarkeit einer Menge M* heißt, daß man eine Methode angeben kann, die testet, ob ein bestimmtes Element zu M gehört oder nicht. Dies ist äquivalent zu der Aussage, daß die *charakteristische Funktion χ_M von M* berechenbar ist,
  wobei χ_M:$\mathbb{N}\to\mathbb{N}$ definiert ist durch $\chi_M(x) = \begin{cases} 1 & \text{falls } x\in M,\\ 0 & \text{sonst}\end{cases}$.
  Eine entscheidbare Menge wird auch als *rekursive Menge* bezeichnet.
- *Semi-Entscheidbarkeit einer Menge M* heißt, daß man eine Methode angeben kann, die testet, ob ein bestimmtes Element zu M gehört, und im Mißerfolgsfall eventuell niemals eine Antwort gibt. Dies ist äquivalent zu der Aussage, daß die *partiell-charakteristische Funktion ψ_M von M* berechenbar ist,
  wobei ψ_M:$\mathbb{N}\to\mathbb{N}$ definiert ist durch $\chi_M(x) = \begin{cases} 1 & \text{falls } x\in M,\\ \perp & \text{sonst}\end{cases}$.
- *Aufzählbarkeit einer Menge M* heißt, daß man eine Methode angeben kann, die schrittweise alle Elemente von M generiert, sofern M überhaupt Elemente besitzt. Dies ist äquivalent zu der Aussage, daß M leer ist oder das Bild $range(f)$) einer berechenbaren totalen Funktion f, wobei $range(f) = \{n\in\mathbb{N} \mid \exists i\in\mathbb{N}\ f(i){=}n\}$.

Diese Erkenntnisse führen zu folgendem Satz.

Theorem 6 (Berechenbarkeitskonzepte, abstrakt spezifiziert).

- *Eine Funktion f:$\mathbb{N}\to\mathbb{N}$ ist berechenbar, wenn $f = \varphi_i$ für ein $i\in\mathbb{N}$ ist.*
- *Eine Menge $M{\subseteq}\mathbb{N}$ ist semi-entscheidbar, wenn die partielle charakteristische Funktion ψ_M berechenbar ist.*
- *$M{\subseteq}\mathbb{N}$ ist entscheidbar, wenn die charakteristische Funktion χ_M berechenbar ist.*
- *$M{\subseteq}\mathbb{N}$ ist aufzählbar, wenn $M{=}\emptyset$ ist oder es ein $f\in\mathcal{TR}$ gibt mit $M = range(f)$.*

Die Charakterisierungen aus Theorem 6 können analog auch für andere Grundmengen wie Tupel von Zahlen oder Wörter formuliert werden. Dies wird im folgenden aber nicht weiter verfolgt.

Aufzählbarkeit ist ein Konzept, das vom Namen her leicht mit dem Begriff der *Abzählbarkeit* verwechselt wird. Dieser Begriff stammt aus der Mathematik und hat zum Ziel, die "Größe" einer unendlichen Menge zu beschreiben. Eine *Menge M ist abzählbar, wenn es eine surjektive Funktion f von $\mathbb{N}$ nach M gibt*. Im Gegensatz dazu stehen beim Begriff der Aufzählbarkeit Aspekte der Berechenbarkeit im Vordergrund. Eine *Menge M ist aufzählbar, wenn es eine berechenbare surjektive Funktion f von $\mathbb{N}$ nach M gibt*. Dies bedeutet neben der Tatsache, daß M abzahlbar sein muß, auch daß die Struktur von M einfach genug sein muß, daß man die gesamte Menge M mit berechenbaren Mitteln schrittweise generieren kann. Um diese beiden Begriffe deutlicher voneinander zu unterscheiden, sagt man daher auch "*rekursiv aufzählbar*" anstelle des einfachen "*aufzählbar*". Wir werden später im Abschnitt 3 einige Mengen kennenlernen, die nicht aufzählbar sein können. Als Teilmengen der natürlichen Zahlen bleiben sie aber nichtsdestotrotz abzählbar.

Bei der Definition der Aufzählbarkeit wird auch nicht verlangt, daß die Elemente der aufgezählten Menge genau einmal oder in einer bestimmten Reihenfolge aufgezählt werden. So können werden die ungeraden Zahlen zum Beispiel durch die Funktion $f \in \mathcal{TR}$ mit $f(n) = \begin{cases} n+1 & \text{falls } n \text{ gerade,} \\ n & \text{sonst} \end{cases}$ aufgezählt, aber jede ungerade Zahl wird genau zweimal genannt. Die Funktion $g = \lambda n.2n+1$ dagegen zählt alle ungeraden Zahlen genau einmal auf. Dies zeigt auch, daß jede rekursiv aufzählbare Menge mehrere Aufzählungsfunktionen besitzt. Es gibt sogar unendlich viele Möglichkeiten, die Menge aufzuzählen, da eine Aufzählungsfunktion durch Vertauschen zweier beliebiger Funktionswerte in eine neue Aufzählungsfunktion umgewandelt werden kann.

Wir wollen im folgenden die Zusammenhänge zwischen Entscheidbarkeit, Semi-Entscheidbarkeit und Aufzählbarkeit untersuchen. Dabei stellt sich als erstes heraus, daß Semi-Entscheidbarkeit und Aufzählbarkeit das gleiche ist und daß die Bedingungen für beide Begriffe abgeschwächt werden dürfen.

Theorem 7 (Äquivalente Charakterisierungen aufzählbarer Mengen).
Für eine Menge $M \subseteq \mathbb{N}$ sind folgende Aussagen äquivalent

1. *M ist rekursiv aufzählbar*
2. *$M = range(f)$ für ein $f \in \mathcal{R}$.*
3. *M ist semi-entscheidbar*
4. *$M = domain(g)$ für ein $g \in \mathcal{R}$.*

Die zweite Charakterisierung schwächt die Bedingungen für Aufzählbarkeit in dem Sinne ab, daß die Aufzählungsfunktion nicht notwendigerweise total sein muß. Die vierte Charakterisierung besagt, daß statt der partiellen charakteristischen Funktion ψ_M auch jede andere berechenbare Funktion über ihren Definitionsbereich eine semi-Entscheidung vornimmt. Wir beweisen die Äquivalenz der vier Charakterisierungen durch Ringschluß. Dabei sind die Schritte von der ersten zur zweiten Charakterisierung und von der dritten zur vierten relativ naheliegend, während die anderen beiden jeweils eine nicht ganz triviale Schlüsselidee benötigen.

Wenn M das Bild einer berechenbaren, aber möglicherweise partiellen Funktion f ist, dann gehört eine Zahl n genau dann zu M, wenn $f(j)=n$ für irgendein j im Definitionsbereich von f ist. In diesem Fall muß die partielle charakteristische Funktion von M den Wert 1 liefern und ansonsten braucht sie nicht zu terminieren. Um also $\psi_M(m)$ zu programmieren, müssen wir nach der kleinsten Stelle j suchen, für die $f(j)=n$ ist. Dabei müssen wir allerdings bedenken, daß $f(x)$ für ein $x<j$ eventuell nicht definiert ist, so daß wir nicht einfach die Eingabewerte $0, 1, 2, \ldots$ für f der Reihe nach durchgehen dürfen. Stattdessen müssen wir uns eines Tricks bedienen, der mittlerweile zum Standardrepertoire der theoretischen Informatik gehört: wir zählen simultan die möglichen Eingabewerte und die Grenze für die Anzahl der erlaubten Rechenschritte hoch, bis wir eine Zahl J und eine Grenze t gefunden haben, so daß die Berechnung von $f(j)$ nach t Schritten anhält und den Wert n liefert. Da wir aufgrund des dritten Axioms der Berechenbarkeitstheorie für jede Eingabe j einer berechenbaren Funktion f entscheiden können, ob $f(j)$ nach t terminiert, führt der Suchprozess genau dann zum Erfolg, wenn n zu M gehört und wir können als Ergebnis die Zahl 1 zurückgeben.

Wenn M der Definitionsbereich einer berechenbaren partiellen Funktion g und nicht leer ist (ansonsten wäre M trivialerweise aufzählbar) ist, dann gehört eine Zahl j genau dann zu M, wenn g bei Eingabe von j nach einer gewissen Anzahl t von Rechenschritten terminiert. Wenn wir M mit einer total rekursive Funktion aufzählen wollen, dann können wir genau in diesem Fall die Zahl j zurückgeben. Allerdings müssen wir als Eingabe für f die Zahl j und die Rechenzeitgrenze t gleichzeitig angeben, also die Eingabe als ein Tupel $\langle j,t\rangle$ interpretieren. Außerdem müssen wir im Mißerfolgsfall, also wenn $f(j)$ nicht nach t Schritten anhält, einen Wert zurückgeben, und wählen hierzu ein festes Element k von M, das es nach Voraussetzung geben muß.
Der folgende Beweis ist eine knappe und präzise Formulierung der obigen Argumente.

Beweis:

1↦2: Es sei $M\subseteq\mathbb{N}$ rekursiv aufzählbar, also $M=\emptyset$ oder $M=range(f)$ für ein $f\in\mathcal{TR}$.
Wenn $M=\emptyset$ ist, dann ist $M=range(f_\perp)$ wobei $f_\perp(i)=\perp$ für alle $i\in\mathbb{N}$, also $f_\perp\in\mathcal{R}$.
Andernfalls ist nach Voraussetzung $M=range(f)$ für ein $f\in\mathcal{TR}\subseteq\mathcal{R}$.

2↦3: Es sei $M = range(f)$ für ein $f\in\mathcal{R}$ und $f = \varphi_i$. Für beliebige $n\in\mathbb{N}$ gilt also $n\in M \Leftrightarrow \exists j\in\mathbb{N}\ \varphi_i(j)=n \Leftrightarrow \exists j,t\in\mathbb{N}\ \Phi_i(j)=t\wedge\varphi_i(j)=n$.
Definiere h:$\mathbb{N}\to\mathbb{N}$ durch $h\langle n,\langle j,t\rangle\rangle = \begin{cases} 0 & \Phi_i(j)=t\wedge\varphi_i(j)=n \\ 1 & \text{sonst}\end{cases}$
Wegen Axiom 3 (Definition 5) ist $h\in\mathcal{TR}$ und es folgt $\psi_M = sign\circ\mu\ h\in\mathcal{R}$.
Damit ist M semi-entscheidbar.

3↦4: Es sei M semi-entscheidbar. Dann ist $M = \{i \mid \psi_M(i)=1\} = domain(\psi_M)$ und $\psi_M\in\mathcal{R}$ ist die gewünschte Funktion g.

4↦1: Es sei $M = domain(g)$ für ein $g\in\mathcal{R}$ und $g = \varphi_i$. Für beliebige $j\in\mathbb{N}$ gilt also $j\in M \Leftrightarrow j\in domain(\varphi_i) \Leftrightarrow \exists t\in\mathbb{N}\ \Phi_i(j)=t$.
Falls $M=\emptyset$ ist, dann ist M per Definition rekursiv aufzählbar.
Andernfalls gibt es ein $k\in\mathbb{N}$ mit $g(k)\neq\perp$. Sei $f\langle j,t\rangle = \begin{cases} j & \text{falls } \Phi_i(j)=t, \\ k & \text{sonst}\end{cases}$
Wegen Axiom 3 (Definition 5) ist $f\in\mathcal{TR}$ und es gilt
$j\in M \Leftrightarrow j=k\vee\exists t\in\mathbb{N}\ \Phi_i(j)=t \Leftrightarrow j=k\vee\exists t\in\mathbb{N}\ f\langle j,t\rangle=j \Leftrightarrow M=range(f)$
Damit ist $M\subseteq\mathbb{N}$ rekursiv aufzählbar. □

2.1 Abschlußeigenschaften aufzählbarer und entscheidbarer Mengen

Wir wollen nun untersuchen, welche Abschlußeigenschaften für rekursiv aufzählbare und entscheidbare Mengen gelten. Neben den üblichen Mengenoperationen wie *Vereinigung* $M \cup M'$, *Durchschnitt* $M \cap M'$, *Komplement* $\overline{M}$ und *Differenz* M-M' interessieren uns auch das *Bild* $f(M)$ und das *Urbild* $f^{-1}(M)$ unter einer berechenbaren Funktion sowie das durch die Standard-Tupelfunktion gebildete *Produkt* $M \times M'$, definiert durch $M \times M' = \{\langle i,j\rangle \mid i \in M \wedge j \in M'\}$ und die entsprechenden *Projektionen* $\pi_i(M)$, definiert durch $\pi_1(M) = \{i \mid \exists j\ \langle i,j\rangle \in M\}$ und $\pi_2(M) = \{j \mid \exists i\ \langle i,j\rangle \in M\}$.

Da entscheidbare Mengen über die Berechenbarkeit ihrer charakteristischen Funktionen definiert sind, sind sie abgeschlossen unter allen Operationen, die sich effektiv mit charakteristischen Funktionen kombinieren lassen. So muß die charakteristische Funktion von $M \cup M'$ genau dann den Wert 1 liefern, wenn χ_M oder $\chi_{M'}$ eine 1 liefern. Dieser Effekt kann durch eine einfache Addition und einer anschließenden Normierung auf 1 erzielt werden. Ähnlich ist die charakteristische Funktion von $M \cap M'$ durch eine Multiplikation der Funktionen χ_M oder $\chi_{M'}$ programmierbar, die charakteristische Funktion von $\overline{M}$ durch Umkehrung von χ_M, die charakteristische Funktion von M-M' durch Durchschnitt mit dem Komplement von M und die charakteristische Funktion von $f^{-1}(M)$ durch Komposition von χ_M und f. Für die charakteristische Funktion von $M \times M'$ benötigt man die Umkehrfunktionen π_1 und π_2 der Standard-Tupelfunktion, um auf die potentiellen Elemente von M bzw. M' zuzugreifen und verfährt dann wie bei der Durchschnittsbildung.

Die Projektion einer entscheidbaren Menge kann dagegen zu einer unendlichen Suche nach der fehlenden Komponente führen, die $\langle i,j\rangle \in M$ liefert, und ist daher im Allgemeinen nicht mehr entscheidbar. So ist z.B. $M = \{\langle i,j,t\rangle \mid \Phi_i(j) = t\}$ entscheidbar, aber ihre erste Projektion ist $H = \{\langle i,j\rangle \mid \exists t\ \Phi_i(j) = t\} = \{\langle i,j\rangle \mid j \in domain(\varphi_i)\}$, das sogenannte *Halteproblem*. Wir werden in Abschnitt 3 zeigen, daß das Halteproblem unentscheidbar ist.

Auch das Bild $f(M)$ einer entscheidbaren Menge M unter einer berechenbaren Funktion f ist nicht notwendigerweise wieder entscheidbar, selbst wenn f total ist. Wie zuvor müsste man, um $n \in f(M)$ zu entscheiden, das Element i mit $f(i) = n$ finden, was eventuell zu einem unendlichen Suchprozeß führt. Ein einfaches Gegenbeispiel ist $f = \pi_1$, was genau zu der oben beschriebenen Situation führt.

Theorem 8 (Abschlußeigenschaften entscheidbarer Mengen).
Die Klasse der entscheidbaren Mengen sind abgeschlossen unter Vereinigung, Durchschnitt, Komplement, Differenz, Produkt und Urbild berechenbarer totaler Funktionen. Sie ist nicht abgeschlossen unter Projektionen oder Bild berechenbarer Funktion.

Beweis:

$M \cup M'$: Es ist $n \in M \cup M' \Leftrightarrow \chi_M(n)=1 \vee \chi_{M'}(n)=1 \Leftrightarrow \chi_M(n) + \chi_{M'}(n) \geq 1$.
Damit ist $\chi_{M \cup M'}(n) = \text{sign}(\chi_M(n) + \chi_{M'}(n))$, also $\chi_{M \cup M'} \in \mathcal{TR}$.

$M \cap M'$: Es ist $n \in M \cap M' \Leftrightarrow \chi_M(n)=1 \wedge \chi_{M'}(n)=1 \Leftrightarrow \chi_M(n) * \chi_{M'}(n) = 1$.
Damit ist $\chi_{M \cap M'}(n) = \chi_M(n) * \chi_{M'}(n)$, also $\chi_{M \cap M'} \in \mathcal{TR}$.

$\overline{M}$: Es ist $n \in \overline{M} \Leftrightarrow \chi_M(n)=0$. Damit ist $\chi_{\overline{M}}(n) = 1 \dot{-} \chi_M(n)$, also $\chi_{\overline{M}} \in \mathcal{TR}$.

M-M': Es ist $n \in M - M' \Leftrightarrow \chi_M(n)=1 \wedge \chi_{M'}(n)=0 \Leftrightarrow \chi_M(n) * (1 \dot{-} \chi_{M'}(n)) = 1$.
Damit ist $\chi_{M-M'}(n) = \chi_M(n) * 1 \dot{-} \chi_{M'}(n))$, also $\chi_{M-M'} \in \mathcal{TR}$.

$M \times M'$: Es ist $n \in M \times M' \Leftrightarrow n = \langle i, j \rangle \wedge i \in M \wedge j \in M'$
$\Leftrightarrow \chi_M(\pi_1(n))=1 \wedge \chi_{M'}(\pi_2(n))=1$.
Damit ist $\chi_{M \times M'}(n) = \chi_M(\pi_1(n)) * \chi_{M'}(\pi_2(n))$, also $\chi_{M \times M'} \in \mathcal{TR}$.

$f^{-1}(M)$: Es ist $n \in f^{-1}(M) \Leftrightarrow f(n) \in M \Leftrightarrow \chi_M(f(n))=1$.
Damit ist $\chi_{f^{-1}(M)} = \chi_M \circ f \in \mathcal{TR}$. □

Für all diese Abschlußeigenschaften kann man mithilfe des smn-Theorems sogar den φ-Index der charakteristischen Funktion der neuen Menge aus denen der Komponenten berechnen. Man sagt daher, daß die Abschlußeigenschaften *effektiv* sind. Im Rahmen dieser Einführung wollen wir dies aber nicht weiter vertiefen.

Für rekursiv aufzählbaren Mengen ist die Argumentation in vielen Fällen ähnlich wie bei den entscheidbaren Mengen, da wir wissen, daß rekursiv aufzählbare Mengen identisch mit den semi-entscheidbaren Mengen sind und somit berechenbare partielle charakteristische Funktionen besitzen. Für den Abschluß unter Durchschnitt, Produkt und Urbild berechenbarer Funktionen, die in diesem Fall auch partiell sein dürfen, können wir daher dieselben Konstruktionen verwenden. Dies funktioniert jedoch nicht mehr bei Vereinigung, Komplement und Differenz. In diesen Fällen führen Zahlen, die nicht zu den betrachteten Mengen gehören, zu undefinierten Funktionswerten bei der partiellen charakteristischen Funktion, die dann nicht für weitere Konstruktionen verwendet werden können. Während diese Problematik dazu führt, daß rekursiv aufzählbare Mengen tatsächlich nicht unter Komplement oder Differenz abgeschlossen sind, läßt sich im Falle der Vereinigung $M \cup M'$ ein Nachweis der Abgeschlossenheit mithilfe der Aufzählungsfunktion konstruieren, der dann rein hypothetisch mit Theorem 7 auch auf partielle charakteristische Funktionen übertragen werden kann. Man zählt einfach wechselweise die Elemente der von M und M' auf und erhält so eine Aufzählung von $M \cup M'$. Auf die gleiche Art kann man auch die Elemente des Bildes $f(M)$ einer rekursiv aufzählbaren Menge M unter einer (möglicherweise partiellen) berechenbaren Funktion f aufzählen. Für den Abschluß unter Projektionen $\pi_i(M)$ können wir die partielle charakteristische Funktion mit einem Suchmechansmus kombinieren um die fehlende andere Komponente zu finden.

Theorem 9 (Abschlußeigenschaften aufzählbarer Mengen).
Die Klasse der rekursiv aufzählbaren Mengen sind abgeschlossen unter Vereinigung, Durchschnitt, Produkt und Projektionen, sowie unter Bild und Urbild berechenbarer, möglicherweise partieller Funktionen.
Sie ist nicht abgeschlossen unter Komplement oder Differenz.

Beweis:

$M \cup M'$: Es seien $M = range(h)$ und $M' = range(h')$ für $h, h' \in \mathcal{R}$. Dann gilt $n \in M \cup M' \Leftrightarrow \exists i\ h(i)=n \vee h'(i)=n$. Wir definieren eine Funktion g durch

$$g(n) = \begin{cases} h(\lfloor n/2 \rfloor) & \text{falls } n \text{ gerade,} \\ h'(\lfloor n/2 \rfloor) & \text{sonst} \end{cases}$$

Dann ist $g \in \mathcal{R}$ und $M \cup M' = range(g)$ rekursiv aufzählbar

$M \cap M'$: Es ist $n \in M \cap M' \Leftrightarrow \psi_M(n)=1 \wedge \psi_{M'}(n)=1 \Leftrightarrow \psi_M(n) * \psi_{M'}(n) = 1$.
Damit ist $\psi_{M \cap M'}(n) = \psi_M(n) * \psi_{M'}(n)$, also $\psi_{M \cap M'} \in \mathcal{TR}$.

$M \times M'$: Es ist $n \in M \times M' \Leftrightarrow n = \langle i, j \rangle \wedge i \in M \wedge j \in M'$
$\Leftrightarrow \psi_M(\pi_1(n))=1 \wedge \psi_{M'}(\pi_2(n))=1$.
Damit ist $\psi_{M \times M'}(n) = \psi_M(\pi_1(n)) * \psi_{M'}(\pi_2(n))$, also $\psi_{M \times M'} \in \mathcal{TR}$.

$\pi_i(M)$: Dann gilt $i \in \pi_1(M) \Leftrightarrow \exists j \ \langle i, j \rangle \in M \Leftrightarrow \exists j \ \psi_M i, j=1$.
Wir definieren eine Funktion g durch $g(i) = \mu_j[\psi_M i, j=1]$. Dann ist $g \in \mathcal{R}$ und $\pi_1(M) = \{i \mid \mu_j[\psi_M i, j=1] \neq \perp\} = domain(g)$ rekursiv aufzählbar.

Der Beweis für die Aufzählbarkeit von $\pi_2(M)$ verläuft analog.

$f(M)$: Es seien $M = range(h)$ und $f, h \in \mathcal{R}$.
Dann gilt $n \in f(M) \Leftrightarrow \exists j \in M \ f(j)=n \Leftrightarrow \exists i \ f(h(i))=n$.
Dann ist $g = f \circ h \in \mathcal{R}$ und $f(M) = range(g)$ rekursiv aufzählbar.

$f^{-1}(M)$: Es ist $n \in f^{-1}(M) \Leftrightarrow f(n) \in M \Leftrightarrow \psi_M(f(n))=1$.
Damit ist $\psi_{f^{-1}(M)} = \psi_M \circ f \in \mathcal{TR}$. □

Weder die Aufzählungsfunktion noch die partielle charakteristische Funktion können zum Nachweis eines Abschlusses rekursiv aufzählbarer Mengen unter Komplement verwendet werden. Im ersten Fall müsste man einen Weg finden, die Elemente zu identifizieren, die niemals aufgezählt werden und im zweiten die Elemente bestimmen, bei denen die partielle charakteristische Funktion niemals anhalten wird. Es gibt keine Möglichkeit, diese Entscheidung nach endlich vielen Schritten zu fällen, denn unmittelbar nachdem ein Algorithmus diese Entscheidung gefällt hätte, könnte die Aufzählungsfunktion das Element doch noch aufzählen oder die partielle charakteristische Funktion das Element akzeptieren.

Dieses Argument ist zwar intuitiv einsichig, aber nicht hinreichend als Beweis, da die Argumentation zu unpräzise ist und nicht beweist, daß man nicht auf eine völlig andere Weise zu einer Lösung kommen könnte. Die Tatsache, daß ein bestimmter Weg nicht funktioniert ist kein Beweis dafür, daß es überhaupt keinen Weg gibt. Wir werden jedoch im Theorem 10 beweisen, daß jede rekursiv aufzählbar Menge, deren Komplement ebenfalls rekursiv aufzählbar ist, auch entscheidbar ist. In Abschnitt 3 werden wir dann zeigen, daß es rekursiv aufzählbare Mengen gibt, die nicht entscheidbar sind. Damit können rekursiv aufzählbare Mengen nicht unter Komplementbildung abgeschlossen sein. Hieraus folgt dann auch unmittelnar, daß sie auch nicht unter Differenzbildung abgeschlossen sein können, denn das Komplement $\overline{M}$ ist identisch mit der Differenz $\mathbb{N} - M$. Ein Abschluß unter Differenzbildung würde also automatisch einen Abschluß unter Komplement zur Folge haben.

2.2 Aufzählbarkeit und Entscheidbarkeit

Da entscheidbare Mengen insbesondere auch semi-entscheidbar sind, wissen wir aufgrund von Theorem 7, daß jede entscheidbare Menge M auch rekursiv aufzählbar ist. Außerdem wissen wir, daß entscheidbare Mengen abgeschlossen unter Komplementbildung sind und daher ist auch das Komplement $\overline{M}$ einer entscheidbaren Menge rekursiv aufzählbar. Umgekehrt wissen wir aber auch, daß die Aufzählbarkeit von M und $\overline{M}$ dazu genutzt werden kann, um M zu entscheiden. Man muß hierzu nur beide Mengen

wechselweise aufzählen bis die zu testende Zahl erscheint und macht die Entscheidung davon abhängig, welche der beiden Aufzählungen die Zahl geliefert hat. Diese Erkenntnisse führen uns zu dem folgenden Satz.

Theorem 10 (Aufzählbarkeit vs. Entscheidbarkeit).

1. *Jede entscheidbare Menge M ist rekursiv aufzählbar.*
2. *Eine Menge M ist genau dann entscheidbar, wenn M und $\overline{M}$ aufzählbar sind.*

Beweis:

1. Es sei M entscheidbar. Dann ist $\chi_M \in \mathcal{TR}$ und $\psi_M(n) = \begin{cases} 1 & \text{falls } \chi_M(n)=1, \\ \perp & \text{sonst} \end{cases}$, also $\psi_M \in \mathcal{R}$. Damit ist M semi-entscheidbar und wegen Theorem 7 auch aufzählbar.

2. $\Rightarrow$: Es sei M entscheidbar. Dann ist nach Theorem 8 auch $\overline{M}$ entscheidbar und nach Teil (1) sind M und $\overline{M}$ aufzählbar.

2. $\Leftarrow$: Seien M und $\overline{M}$ aufzählbar. Falls M=$\emptyset$ oder $\overline{M}$=$\emptyset$, so ist M entscheidbar.
Andernfalls gibt es Funktionen $f, g \in \mathcal{TR}$ mit $M = range(f)$ und $\overline{M} = range(g)$.
Definiere h:$\mathbb{N}\to\mathbb{N}$ durch $h(n) = min\{i \mid f(i)=n \vee g(i)=n\}$.
Dann ist h berechenbar und total, da für jedes n entweder $n \in M$ oder $n \in \overline{M}$ gilt, und es folgt $\chi_M(n) = \begin{cases} 1 & \text{falls } f(h(n)) = n, \\ 0 & \text{sonst} \end{cases}$ also $\chi_M \in \mathcal{TR}$. □

Der folgende Satz beschreibt einige wichtige entscheidbare und aufzählbare Mengen.

Theorem 11 (Wichtige entscheidbare und aufzählbare Mengen).
Es sei $f \in \mathcal{R}$ eine berechenbare Funktion.

1. *Jede endliche Menge ist entscheidbar und aufzählbar.*
2. *$domain(f)$ und $range(f)$ sind aufzählbar.*
3. *$range(f)$ ist entscheidbar, wenn f total und monoton wachsend ist.*
4. *$graph(f) = \{\langle i,j\rangle \mid f(i) = j\}$ ist aufzählbar.*
5. *$graph(f)$ ist entscheidbar, wenn f total ist.*
6. *Die Menge $\{\langle i,n,t\rangle \mid \Phi_i(n)=t\}$ ist entscheidbar.* *(Rechenzeit ist entscheidbar)*
7. *Die Menge $\{\langle i,n,y\rangle \mid \varphi_i(n)=y\}$ ist aufzählbar.*
8. *Die Menge $S = \{i \mid i \in domain(\varphi_i)\}$ ist aufzählbar* *(Selbstanwendbarkeitsproblem)*
9. *Die Menge $H = \{\langle i,n\rangle \mid n \in domain(\varphi_i)\}$ ist aufzählbar.* *(Halteproblem)*

In den meisten Fällen sind die Beweisideen recht naheliegend.

Beweis:

1. Für $M = \{x_1, .., x_n\}$ muß die charakteristische Funktion χ_M nur endlich viele Vergleiche durchführen. Es ist $\chi_M(n) = \begin{cases} 1 & n=x_1 \vee ... \vee n=x_n, \\ 0 & \text{sonst} \end{cases}$, also $\chi_M \in \mathcal{TR}$.

2. $domain(f)$ und $range(f)$ sind nach Theorem 7 aufzählbar.

3. Sei $f \in \mathcal{TR}$ monoton und $M = range(f)$.
Wenn M endlich ist, dann ist M nach Teil (1) entscheidbar.
Andernfalls wächst f unbegrenzt, d.h. für jedes n gibt es ein i mit $f(i) \geq n$. Daher ist χ_M beschreibbar als $\chi_M(n) = \begin{cases} 1 & f(\mu_z[f(z) \geq n] = n \\ 0 & \text{sonst} \end{cases}$ und damit total rekursiv.

4. Die Funktion $\psi_{graph(f)}$ mit $\psi_{graph(f)}\langle i,j\rangle = \begin{cases} 1 & falls\, f(i){=}j, \\ \perp & sonst \end{cases}$
ist berechenbar für jedes $f \in \mathcal{R}$ und damit ist $graph(f)$ aufzählbar.

5. Für $f \in \mathcal{TR}$ ist $\chi_{graph(f)}\langle i,j\rangle$ = $\begin{cases} 1 \text{ falls } f(i){=}j, \\ 0 \text{ sonst} \end{cases}$ also $\chi_{graph(f)} \in \mathcal{TR}$.

6. Die Menge $\{\langle i,n,t\rangle \mid \Phi_i(n){=}t\}$ ist wegen Axiom 3 (Definition 5) entscheidbar.

7. Die Menge $\{\langle i,n,y\rangle \mid \varphi_i(n){=}y\}$ ist Graph der universellen Funktion (Axiom 4) und damit aufzählbar.

8. $S = \{i \mid i \in domain(\varphi_i)\}$ ist der Definitionsbereich von $\lambda i.u\langle i,i\rangle$, also aufzählbar.

9. $H = \{\langle i,n\rangle \mid n \in domain(\varphi_i)\}$ ist der Definitionsbereich der universellen Funktion, also aufzählbar.

□

Rekursiv aufzählbare Mengen lassen sich auch als Projektionen entscheidbarer Mengen charakterisieren.

Theorem 12 (Projektionssatz).
Eine Menge M ist genau dann rekursiv aufzählbar, wenn es eine entscheidbare Menge M' gibt mit $M = \{y \mid \exists n\ \langle n,y\rangle \in M'\}$

Beweis:

$\Rightarrow$: Es sei M rekursiv aufzählbar. Falls M leer ist, so ist $M = \{y \mid \exists n\ \langle n,y\rangle \in \emptyset\}$.
Andernfalls ist $M = range(f) = \{y \mid \exists n\ \langle n,y\rangle \in graph(f)\}$ für ein $f \in \mathcal{TR}$.
Nach Theorem 11 ist der Graph einer berechenbaren totalen Funktion entscheidbar.

$\Leftarrow$: Es sei $M = \{y \mid \exists n\ \langle n,y\rangle \in M'\} = \pi_2(M')$ für eine entscheidbare Menge M'.
Da M' insbesondere auch aufzählbar ist, gilt dies nach Satz 9 auch für M. □

Zum Abschluß dieses Abschnitts präsentieren wir einen neuen Beweis des Kleene Normalform Theorems, das wir schon im Artikel über die rekursiven Funktionen vorgestellt hatten. Es besagt, daß jede berechenbare Funktion durch eine einzige Minimierung aus einer totalen berechenbaren Funktion generiert werden kann. Der bisherige Beweis basiert im wesentlichen auf dem aufwendigen Beweis für die Turing-Mächtigkeit der rekursiven Funktionen. Der neue Beweis benötigt kein Maschinenmodell, sondern nur noch die Axiome der Berechenbarkeitstheorie. Darüber hinaus läßt sich das Theorem so formulieren, daß wir nur eine einzige totale berechenbare Funktion benötigen, um jede berechenbare Funktion und ihre Rechenzeitfunktion zu erzeugen.

Theorem 13 (Kleene Normalform Theorem).
Es gibt berechenbare totale Funktionen f, g und h mit der Eigenschaft
$\varphi_i(n) = g(\mu f\langle i,n\rangle)$ *und* $\Phi_i(n) = h(\mu f\langle i,n\rangle)$

Beweis:

Wir definieren f durch $f\langle i,n,\langle y,t\rangle\rangle = \begin{cases} 0 \text{ falls } \Phi_i(n){=}t \text{ und } \varphi_i(n){=}y \\ 1 \text{ sonst} \end{cases}$
Da $\{(i,n,t) \mid \Phi_i(n){=}t\}$ entscheidbar ist, ist f total und berechenbar und es gilt $\varphi_i(n) = \pi_1^2(\mu f\langle i,n\rangle)$ und $\Phi_i(n) = \pi_2^2(\mu f\langle i,n\rangle)$. Damit erfüllen f und $g{=}\pi_1^2$ und $h{=}\pi_2^2$ die Bedingungen des Theorems. □

3 Unlösbare Probleme

Nachdem wir im vorhergehenden Abschnitt die Grundlagen einer abstrakten Theorie der Berechenbarkeit gelegt und die wichtigsten Abschlußeigenschaften aufzählbarer und entscheidbarer Mengen untersucht haben, wollen wir uns nun der Frage widmen, ob es Probleme gibt, die nicht mithilfe eines Computers gelöst werden können. Dabei geht es nicht darum, die Fähigkeiten eines konkreten Computers zu analysieren, sondern herauszufinden, welche Probleme prinzipiell unlösbar sind, also mit keinem Computermodell jemals berechnet werden können, unabhängig davon welche Fortschritte die Informatik je erzielen wird.

Aufgrund der Church'schen These dürfen wir davon ausgehen, daß alles, was intuitiv berechnet werden kann, die fünf Axiome der Berechenbarkeit in Definition 5 erfüllt, also in unserem abstrakten Modell beschrieben werden kann. Damit konkretisiert sich die Frage nach Unlösbarkeit auf die Frage nach Funktionen, die nicht berechenbar sind, und nach Mengen (bzw. Sprachen oder Probleme), die nicht entscheidbar oder nicht einmal rekursiv aufzählbar sind. Die Antwort auf diese Fragen zeigt die prinzipiellen Grenzen für die Fähigkeiten von Computern auf, die keineswegs so allmächtig sind, wie man manchmal zu glauben scheint.

Es ist leicht einzusehen, daß es derartige unlösbare Probleme geben muß. Hierzu reicht ein einfaches Kardinalitätsargument, welches zeigt, daß es mehr Funktionen als Programme gibt und folglich nicht jede Funktion berechenbar sein kann. Die Tatsache, daß die Menge der Programme numerierbar ist, macht die Klasse der berechenbaren Funktionen *abzählbar unendlich*, was in der Mathematik die kleinste Kardinalität unendlicher Mengen ist. Die Menge der (totalen) Funktionen auf den natürlichen Zahlen ist dagegen *überabzählbar*, also nicht abzählbar, und gleiches gilt für die Menge der Sprachen über einem Alphabet. Der Beweis hierfür geht zurück auf eine Methode, die Cantor für seinen Nachweis der Überabzählbarkeit der reellen Zahlen entwickelt hatte.

Beispiel 6 (Cantors Diagonalbeweis für Überabzählbarkeit von $\mathbb{N}\to\mathbb{N}$)

Um zu zeigen, daß die Menge $\mathbb{N}\to\mathbb{N}$ nicht abzählbar ist, nehmen wir an, es wäre möglich, alle totalen Funktionen über $\mathbb{N}$ durchzuzählen, also jede Funktion aus $\mathbb{N}\to\mathbb{N}$ als ein f_i zu bezeichnen. Mithilfe dieser Numerierung konstruieren wir nun eine neue Funktion $g : \mathbb{N}\to\mathbb{N}$, die sich von jeder Funktion f_i an mindestens einer Stelle unterscheidet. Die Funktion g muß dann offensichtlich in der Numerierung aller Funktionen fehlen, was bedeutet, daß die Numerierung nicht vollständig sein kann.

Die Konstruktion von g läßt sich am einfachsten anhand einer Tabelle illustrieren. Wir tragen hierzu senkrecht alle numerierten Funktionen und waagerecht alle möglichen Eingaben auf und tragen in der Tabelle die entsprechenden Funktionswerte ein.

	0	1	2	3	4	...
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$f_0(4)$	...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$f_1(4)$	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$f_2(4)$	...
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	$f_3(4)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Nun konstruieren wir die Funktion g, indem wir diagonal durch diese Tabelle hindurchgehen und dabei jede Funktion an einer Stelle verändern, was sich am einfachsten durch Addition von 1 erreichen läßt. Wir definieren $g : \mathbb{N} \to \mathbb{N}$ durch $g(x) = f_x(x)+1$.

	0	1	2	3	4	...
f_0	$f_0(0)+1$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$f_0(4)$	...
f_1	$f_1(0)$	$f_1(1)+1$	$f_1(2)$	$f_1(3)$	$f_1(4)$	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)+1$	$f_2(3)$	$f_2(4)$	...
f_3	$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)+1$	$f_3(4)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Per Konstruktion ist g offensichtlich total, kann aber in der Tabelle selbst nicht als eine der Funktionen f_i vorkommen, da es ansonsten an der Schnittstelle zwischen der Diagonalen und der Waagerechten für g einen Konflikt gäbe. Denn wenn $g = f_i$ für ein i wäre, dann müsste an dieser Stelle die Gleichung $g(i) = f_i(i)$, nach Definition von g aber auch $g(i) = f_i(i)+1$ gelten. Insgesamt hätten wir also $f_i(i) = f_i(i)+1$, was für totale Funktionen ein Widerspruch ist.

Da die Numerierung von $\mathbb{N} \to \mathbb{N}$ beliebig gewählt war, kann es somit keine Möglichkeit geben, alle totalen Funktionen auf den natürlichen Zahlen abzuzählen.

Damit wissen wir also, daß es Funktionen geben muß, die nicht berechenbar sind, denn gemessen an der Anzahl der Funktionen, die es überhaupt gibt, sind fast alle Funktionen nicht berechenbar. Andererseits sind aber so gut wie alle Funktionen, die in der Realität vorkommen, leicht zu berechnen. Dies macht es relativ schwierig, Unlösbarkeit für ein konkretes Beispiel zu beweisen, obwohl – wie sich im Nachhinein herausstellt – einige unlösbare Probleme durchaus in der Realität der Informatik auftauchen.

Einer der Gründe, warum Unlösbarkeit so schwer zu zeigen ist, ist daß syntaktische Methoden wie das Pumping Lemma für allgemeine Berechenbarkeitsmodelle nicht mehr greifen. Wenn eine syntaktische Struktur mit endlichen Mitteln beschreibbar ist, dann ist sie auch entscheidbar. Unentscheidbare Probleme müssen daher eine sehr kompliziertere Struktur besitzen.

Man beginnt daher mit Beispielen unlösbarer Probleme, die auf den ersten Blick etwas konstruiert wirken, aber im Endeffekt doch eine durchaus praktische Relevanz besitzen. Hierzu zählt das sogenannte *Selbstanwendbarkeitsproblem*, das wir in Satz 11 als aufzählbare Menge vorgestellt hatten. Das Selbstanwendbarkeitsproblem stellt die Frage, ob ein Programm bei Eingabe seiner eigenen Gödelnummer anhält oder nicht. Es ist verhältnismäßig leicht, die Unentscheidbarkeit dieser Problemstellung nachzuweisen, da *Cantors Diagonalmethode* bereits in die Problemstellung hineincodiert ist. Das Selbstanwendbarkeitsproblem ist aber weit mehr als ein rein technisches Problem, denn es stellt die Frage, wann Programme auf sich selbst referenzieren dürfen. Je nach Programmiersprache hat diese Frage sehr verschiedene Lesarten – wann darf ein λ-Term auf sich selbst angewandt werden, wann terminiert die Anwendung eines Fixpunktkombinators, wann terminiert eine Schleife in einer imperativen Sprache, wann darf ein Objekt auf sich selbst verweisen, usw.

Andere Probleme, die realer sind als das Selbstanwendbarkeitsproblem, können entweder ebenfalls mit der Cantorschen Diagonalmethode bewiesen oder auf ein bereits als unlösbar bekanntes Problem zurückgeführt werden. So kann man zum Beispiel zeigen,

daß das *Halteproblem*, also die Frage, ob ein Programm bei einer beliebigen Eingabe terminiert, mit einem einfachen Argument als unentscheidbar nachweisen. Da man nicht einmal testen kann, ob ein Programm auf seiner eigenen Gödelnummer terminiert, kann dies auch nicht für eine beliebige Eingage möglich sein. Mit einem Ähnlichen Argument kann man auch zeigen, daß das *Korrektheitsproblem* (erfüllt ein Programm eine gegebene Spezifikation) und das *Äquivalenzproblem* (berechnen zwei Programme die gleiche Funktion) unentscheidbar sind: wenn man nicht einmal testen kann, ob ein Programm terminiert, dann kann man mit sicherheit nicht seine Korrektheit oder seine Äquivalenz zu einem anderen Programm testen. Diese Methode, welche die Unlösbarkeit eines Problems auf die eines anderen zurückführt nennt man *Problemreduktion* oder kurz *Reduktion*. Wir werden im folgenden beide Beweisverfahren präzisieren und an einer Reihe wichtiger Beispiele illustrieren.

3.1 Unlösbarkeitsbeweise mit Cantors Diagonalmethode

Diagonalisierung ist eine mächtige, auf den ersten Blick etwas kompliziert wirkende Technik, um Gegenbeispiele für allgemeingültige Aussagen über unendliche Objekte wie Mengen oder Funktionen zu konstruieren. Sie geht zurück auf Cantors Beweis der Überabzählbarkeit der reellen Zahlen, den wir in Beispiel 6 in seiner Variante für Funktionen auf natürlichen Zahlen vorgestellt hatten. In der Informatik wird sie im wesentlichen für den Nachweis der Unlösbarkeit gewisser Probleme verwendet, also für den Beweis der Unentscheidbarkeit oder Nicht-Aufzählbarkeit einer Menge oder der Unberechenbarkeit gewisser Funktionen. Der Name *Diagonalbeweis* deutet dabei an, daß die Konstruktion diagonal durch eine fiktiven Tabelle aller Möglichkeiten hindurchgeht und dabei aus der Annahme, das Problem sei lösbar, einen Widerspruch erzeugt. Wir wollen diese Technik im folgenden ausführlich an einem Beispiel illustrieren.

Beispiel 7 (Diagonalbeweis für die Unentscheidbarkeit des Halteproblems)
Wir wollen die Methode der Diagonalisierung nutzen um zu beweisen, daß das Halteproblem $H = \{\langle i, n\rangle \mid n \in domain(\varphi_i)\}$ *unentscheidbar ist. Hierzu nehmen wir an,* H *sei entscheidbar und führen diese Annahme zum Widerspruch.*

1. *Da die Menge* H *nach Annahme entscheidbar ist, ist die charakteristische Funktion* $\chi_H : \mathbb{N} \to \mathbb{N}$ *mit* $\chi_H\langle i, n\rangle = \begin{cases} 1 & n \in domain(\varphi_i) \\ 0 & sonst \end{cases}$ *berechenbar.*
2. *Mit* χ_H *konstruieren wir nun eine neue berechenbare Funktion* f *durch Diagonalisierung. Zur Illustration dieser Konstruktion erzeugen wir eine Tabelle der relevanten Funktionsberechnungen. In unserem Beispiel kennzeichnet die Stelle* i, n*, ob die Berechnung der Funktion* φ_i *bei Eingabe von* n *terminiert. Sie enthält also ein* $\times$*, wenn* $n \in domain(\varphi_i)$*, und sonst ein* $\perp$*.*

$\varphi_i(n)$	0	1	2	3	4	...
φ_0	$\times$	$\times$	$\times$	$\perp$	$\times$	...
φ_1	$\perp$	$\perp$	$\times$	$\times$	$\times$	...
φ_2	$\times$	$\times$	$\perp$	$\times$	$\times$	...
φ_3	$\perp$	$\times$	$\perp$	$\times$	$\perp$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Die Funktion f konstruieren wir nun durch Abwandlung der Funktionswerte auf der Diagonalen dieser Tabelle. Hierbei müssen wir auf zwei Aspekte achten, um den gewünschten Widerspruch zu erzeugen.
Einerseits müssen wir darauf achten, daß die neu konstruierte Funktion von der Art her zu den in der Tabelle aufgelisteten Funktionen gehören muß. In unserem Beispiel heißt dies, daß f eine der Funktionen φ_i, also eine berechenbare Funktion, sein muß. Andererseits wollen wir, daß die Funktion sich per Konstruktion von allen in der Tabelle genannten Funktionen unterscheidet. In unserem Beispiel kann dies dadurch erreichen, daß f an der Stelle n genau dann terminiert, wenn φ_n an der Stelle n nicht terminiert. In der Tabelle würden f also daurch beschrieben, daß man auf der Diagonalen die Werte $\times$ und $\perp$ vertauscht.

$\varphi_i(n)$	0	1	2	3	4	...
φ_0	$\perp$	$\times$	$\times$	$\perp$	$\times$	...
φ_1	$\perp$	$\times$	$\times$	$\times$	$\times$	...
φ_2	$\times$	$\times$	$\times$	$\times$	$\times$	...
φ_3	$\perp$	$\times$	$\perp$	$\perp$	$\perp$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Diese Überlegung führt zur Definition $f(n) = \begin{cases} 0 & \text{falls } n \notin domain(\varphi_n) \\ \perp & \text{sonst} \end{cases}$
Damit ist erreicht, daß f sich von allen Funktionen φ_i an einer Stelle unterscheidet. Anderseits ist f aber auch berechenbar, da der Test $n \notin domain(\varphi_n)$ durch die nach Annahme berechenbare Funktion χ_H ausgedrückt werden kann. Es ist nämlich

$$f(n) = \begin{cases} 0 & \text{falls } \chi_H\langle n,n\rangle{=}0 \\ \perp & \text{sonst} \end{cases}.$$

d.h. f entsteht durch Fallunterscheidung aus berechenbaren Funktionen.

3. *Um den Beweis abzuschließen müssen wir noch zeigen, daß durch die obige Konstruktion tatsächlich ein Widerspruch entstanden ist.*
   Da die Funktion f berechenbar ist, muß es nach Theorem 6 einen Index j geben, für den $f = \varphi_j$ ist. In der Tabelle wird f also gleichzeitig durch die Zeile j und durch die Diagonale beschrieben. Wir betrachten nun daß Verhalten von f an der Schnittstelle der Diagonalen mit der Zeile j, also das Verhalten von f an der Stelle j. Da es beim Halteproblem nur um Terminierung geht, untersuchen wir, ob j zum Definitionsbereich von f gehört oder nicht.
   Nach Definition von f ist $f(j)$ genau dann definiert, wenn $j \notin domain(\varphi_j)$ ist, also $j \in domain(f) \Leftrightarrow j \notin domain(\varphi_j)$. Andererseits ist $f = \varphi_j$, also wissen wir $j \in domain(f) \Leftrightarrow j \in domain(\varphi_j)$. Damit haben wir einen Widerspruch, denn es folgt $j \in domain(\varphi_j) \Leftrightarrow j \notin domain(\varphi_j)$.

Aus dem Widerspruch folgt, daß χ_H nicht berechenbar sein kann und damit ist das Halteproblem nicht entscheidbar.

Die Unentscheidbarkeit des Halteproblems zeigt, daß es kein allgemeines Verfahren geben kann, mit dem sich die *Terminierung von Programmen* automatisch testen läßt. Damit ist es auch nicht möglich, die *Korrektheit von Software* automatisch zu testen. Softwaretestverfahren können bestenfalls Fehler aufdecken, aber niemals die Korrektheit von Software garantieren. In sicherheitskritischen Anwendungen bleibt einem also nichts übrig, als die Korrektheit eigener Programme von Hand zu beweisen.

Beispiel 7 gibt einen ausführlichen Beweis für die Unentscheidbarkeit des Halteproblems und erklärt sehr detailliert, warum dies der Fall ist. Zur Illustration der Methodik haben wir eine Tabelle verwendet, die für den Beweis aber nicht wirklich benötigt wird. Der eigentliche Beweis kann daher erheblich prägnanter aufgeschrieben werden.

Theorem 14. *Das Halteproblem* $H = \{\langle i, n\rangle \mid n \in domain(\varphi_i)\}$ *ist unentscheidbar.*

Beweis: Wir nehmen an, H sei entscheidbar. Dann ist die charakteristische Funktion χ_H von H berechenbar. Wir definieren eine Funktion f:$\mathbb{N}\to\mathbb{N}$ durch

$$f(n) = \begin{cases} 0 & \text{falls } n \notin domain(\varphi_n) \\ \perp & sonst \end{cases}.$$

f ist berechenbar, da $f(n) = \mu_z[\chi_H\langle n, n\rangle{=}0]$. Also gibt es ein j mit $f{=}\varphi_j$ und es folgt $j \in domain(\varphi_j) \Leftrightarrow j \in domain(f) \Leftrightarrow j \notin domain(\varphi_j)$.
Dies ist ein Widerspruch Also kann H nicht entscheidbar sein. □

Die Unentscheidbarkeit des *Selbstanwendbarkeitsproblems* $S = \{i \mid i \in domain(\varphi_i)\}$ läßt sich sogar noch schneller beweisen, da da Diagonalisierung bereits in der Definition des Problems enthalten ist. Wir zeigen hierzu zunächst, daß das Komplement $\overline{S}$ des Selbstanwendbarkeitsproblems nicht rekursiv aufzählbar ist, woraus mit Theorem 10 dann die Unentscheidbarkeit von S folgt.

Theorem 15. *Die Menge* $\overline{S}{=}\{i \mid i \notin domain(\varphi_i)\}$ *ist nicht rekursiv aufzählbar.*

Beweis: Wir nehmen an $\overline{S}$ sei rekursiv aufzählbar. Nach Theorem 7 gibt es dann eine berechenbare Funktion f mit $\overline{S} = domain(f)$, also $\overline{S} = domain(\varphi_j)$ für ein j. Es folgt $j \in \overline{S} \Leftrightarrow j \in domain(\varphi_j) \Leftrightarrow j \notin \overline{S}$. Dies ist ein Widerspruch. Also kann $\overline{S}$ nicht rekursiv aufzählbar sein. □

Korollar 16
Das Selbstanwendbarkeitsproblem $S = \{i \mid i \in domain(\varphi_i)\}$ *ist unentscheidbar.*

Diese Beispiele zeigen, daß Diagonalbeweise einem gewissen Schema folgen. Um zu zeigen, daß eine Menge M eine Eigenschaft P (also z.B. Entscheidbarkeit, Aufzählbarkeit oder Abzählbarkeit) nicht besitzt, nehmen wir an, M habe die Eigenschaft P und konstruieren ein Objekt x, das sich widersprüchlich verhält. Einerseits wird x per Konstruktion *verschieden von allen Elementen* der Menge M sein. Andererseits wird x aufgrund der Annahme zur Menge M gehören. Aus diesem Widerspruch folgt dann, daß die Annahme nicht gelten kann.

Wenn die Menge M aus unendlichen Objekten wie Funktionen oder reellen Zahlen besteht und selbst unendlich ist, dann ist Diagonalisierung ein hilfreiches Mittel, um ein derart widersprüchliches Objekt zu konstruieren. Man geht "diagonal" durch die Elemente von M hindurch, so daß jedes Element von M an einer Stelle berührt werden kann. Zur Illustration kann man hierzu alle Elemente von M als Zeilen einer fiktiven Tabelle auftragen und findet dann auf der Diagonalen dieser Tabelle jeweils die i-te Stelle des i-tene Elements von M. Dieser Eintrag wird nun minimal modifiziert, z.B. indem man 1 addiert oder Werte vertauscht, so daß das Objekt auf der Diagonalen immer noch ein Element von M sein müsste, aber per Konstruktion anders ist als alle Zeilen der Tabelle. Wir wollen diese Methode an einem weiteren Beispiel illustrieren.

Beispiel 8 (Total rekursive Funktionen sind nicht aufzählbar)
Wir wollen zeigen, daß die Menge $TR = \{i \mid \varphi_i \ total\}$ der Indizes total-rekursiver Funktionen nicht rekursiv aufzählbar ist. Hierzu nehmen wir an, TR sei rekursiv aufzählbar und führen diese Annahme zum Widerspruch.

1. *Da TR nach Annahme rekursiv aufzählbar ist, gibt es eine total rekursive Funktion f, deren Bildbereich genau die Menge TR ist, kurz range(f) = TR.*
2. *Mit f konstruieren wir nun eine neue berechenbare Funktion h durch Diagonalisierung. Zur Illustration dieser Konstruktion erzeugen wir eine Tabelle der aufgezählten Funktionen, in der an der Stelle i, n, der Wert der i-ten aufgezählten Funktion an der Stelle n,also $\varphi_{f(i)}(n)$ steht.*

	0	1	2	3	4	...
$\varphi_{f(0)}$	$\varphi_{f(0)}(0)$	$\varphi_{f(0)}(1)$	$\varphi_{f(0)}(2)$	$\varphi_{f(0)}(3)$	$\varphi_{f(0)}(4)$	...
$\varphi_{f(1)}$	$\varphi_{f(1)}(0)$	$\varphi_{f(1)}(1)$	$\varphi_{f(1)}(2)$	$\varphi_{f(1)}(3)$	$\varphi_{f(1)}(4)$	...
$\varphi_{f(2)}$	$\varphi_{f(2)}(0)$	$\varphi_{f(2)}(1)$	$\varphi_{f(2)}(2)$	$\varphi_{f(2)}(3)$	$\varphi_{f(2)}(4)$	...
$\varphi_{f(3)}$	$\varphi_{f(3)}(0)$	$\varphi_{f(3)}(1)$	$\varphi_{f(3)}(2)$	$\varphi_{f(3)}(3)$	$\varphi_{f(3)}(4)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Die Funktion h konstruieren wir nun durch Abwandlung der Funktionswerte auf der Diagonalen dieser Tabelle. Hierzu addieren wir jeweils eine Eins zum jeweiligen Funktionswert.

	0	1	2	3	4	...
$\varphi_{f(0)}$	$\varphi_{f(0)}(0)+1$	$\varphi_{f(0)}(1)$	$\varphi_{f(0)}(2)$	$\varphi_{f(0)}(3)$	$\varphi_{f(0)}(4)$	...
$\varphi_{f(1)}$	$\varphi_{f(1)}(0)$	$\varphi_{f(1)}(1)+1$	$\varphi_{f(1)}(2)$	$\varphi_{f(1)}(3)$	$\varphi_{f(1)}(4)$	...
$\varphi_{f(2)}$	$\varphi_{f(2)}(0)$	$\varphi_{f(2)}(1)$	$\varphi_{f(2)}(2)+1$	$\varphi_{f(2)}(3)$	$\varphi_{f(2)}(4)$	...
$\varphi_{f(3)}$	$\varphi_{f(3)}(0)$	$\varphi_{f(3)}(1)$	$\varphi_{f(3)}(2)$	$\varphi_{f(3)}(3)+1$	$\varphi_{f(3)}(4)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

Wir definieren also $h(n) = \varphi_{f(n)}(n)+1$ und haben damit erreicht, daß h sich von allen aufgezählten Funktionen an einer Stelle unterscheidet. Außerdem ist h total, da $\varphi_{f(n)}$ nach Annahme überall definiert ist. Schließlich ist h berechenbar, da $h(n) = u\langle f(n), n\rangle+1$ ist.
3. *Um den Beweis abzuschließen müssen wir noch zeigen, daß durch die obige Konstruktion tatsächlich ein Widerspruch entstanden ist.*
   Da h total-rekursiv ist, gehört der Index von h zur Menge TR. Nach Annahme gibt es also eine Zahl j, so daß $h = \varphi_{f(j)}$ ist. Betrachtet man nun das Verhalten von h an dieser Stelle, so ergibt sich einerseits $h(j) = \varphi_{f(j)}(j)$ wegen $h = \varphi_{f(j)}$ und andererseits $h(j) = \varphi_{f(j)}(j)+1$ nach Konstruktion von h.

Dies ist ein Widerspruch und somit kann TR nicht rekursiv aufzählbar sein.

Eine wichtige Konsequenz der Nicht-Aufzählbarkeit der total-rekursiven Funktionen ist, daß man keine Programmiersprachen entwerfen kann, in denen alle totalen berechenbaren Funktionen programmiert werden können, aber keine partiellen. Solche Programmiersprachen hätten den Vorteil, daß es ohne Einschränkungen der Ausdruckskraft keine Endlosschleifen mehr gäbe. Sie wären aber eine Aufzählung der total-rekursiven Funktionen und können daher nicht existieren.

Beim Führen von Widerspruchsbeweisen ist darauf zu achten, daß der Widerspruch wirklich logisch aus der Annahme folgt und nicht etwa auf versteckte weitere Annahmen oder Fehler im Beweis zurückzuführen ist. Ein häufiger Fehler istauf einen undefinierten Wert eine Eins zu addieren und daraus einen Widerspruch abzuleiten. Mit einer derartigen Argumentation könnte man z.B. versuchen, den Beweis in Beispiel 8 so abzuwandeln, daß die Nicht-Aufzählbarkeit der partiell-rekursiven Funktionen gezeigt würde. Aufgrund des ersten Axioms der Berechenbarkeitstheorie (Definition 5) ist dies natürlich nicht möglich. Derartige Trugschlüsse fallen schneller auf, wenn man die Beweise kurz und prägnant formuliert. Wir geben daher noch einen knappen Beweis für die Nicht-Aufzählbarkeit der total-rekursiven Funktionen.

Theorem 17. *Die Menge* $TR = \{i \mid \varphi_i \text{ total}\}$ *ist nicht rekursiv aufzählbar.*

Beweis: Wir nehmen an, TR sei rekursiv aufzählbar. Dann gibt es eine total-rekursive Funktion f mit $TR = range(f)$. Wir definieren eine Funktion $h{:}\mathbb{N}\to\mathbb{N}$ durch $h(n) = \varphi_{f(n)}(n)+1 = u\langle f(n), n\rangle+1$. Dann ist h berechenbar und total und somit gibt es ein j mit $h = \varphi_{f(j)}$. Für dieses j gilt: $\varphi_{f(j)}(j) = h(j) = \varphi_{f(j)}(j)+1$. Dies ist ein Widerspruch und somit kann TR nicht rekursiv aufzählbar sein. □

3.2 Reduktionsbeweise

Der Einsatz von Diagonalisierung für den Nachweis der Unlösbarkeit eines Problems ist naheliegend bei maschinennah formulierten Problemen wie dem Selbstanwendbarkeitsproblem oder dem Halteproblem. Bei vielen anderen Problemstellungen ist ein Diagonalbeweis allerdings recht kompliziert und gelegentlich auch fast undurchführbar. In diesen Fällen bietet es sich an, die Unlösbarkeit des Problems auf die eines anderen zurückzuführen.

Beispiel 9 (Das Halteproblem braucht keinen Diagonalbeweis)
Die Unentscheidbarkeit von $H = \{\langle i, n\rangle \mid n \in domain(\varphi_i)\}$ *läßt sich auf die Unentscheidbarkeit von* $S = \{i \mid i \in domain(\varphi_i)\}$ *zurückführen.*

Wäre H entscheidbar, dann wäre seine charakteristische Funktion χ_H berechenbar. Da aber für alle $i \in \mathbb{N}$ gilt $i \in S \Leftrightarrow \langle i, i\rangle \in H$ und damit $\chi_S(i) = \chi_H\langle i, i\rangle$ ist, wäre auch die charakteristische Funktion von S berechenbar. Da S aber unentscheidbar ist, kann dies nicht der Fall sein und damit ist H unentscheidbar.

Das obige Argument bettet das Selbstanwendbarkeitsproblem in das Halteproblem ein, indem mögliche "Eingaben" für S in solche für H transformiert werden. Dabei ist die Transformation eine berechenbare Funktion f mit der Eigenschaft $x \in S \Leftrightarrow f(x) \in H$ und damit kann jede Entscheidung für H direkt auf S übertragen werden. In diesem Sinne *reduziert* f das Selbstanwendbarkeitsproblem auf das Halteproblem.

Definition 10 (Funktionale Reduzierbarkeit).
Ein Problem[2] P *ist reduzierbar auf* P'*, kurz* $P \leq P'$*, wenn es eine berechenbare totale Funktion f gibt mit der Eigenschaft* $P = f^{-1}(P')$.

[2] Im Kontext von Unlösbarkeit spricht man üblicherweise von Problemen, verwendet aber zur formalen Beschreibung die Menge der positiven Antworten (oder Lösungen) für das Problem, so daß diese Begriffe als synonym anzusehen sind. Technisch betrachtet müsste man eigentlich von der Reduzierbarkeit von Mengen sprechen.

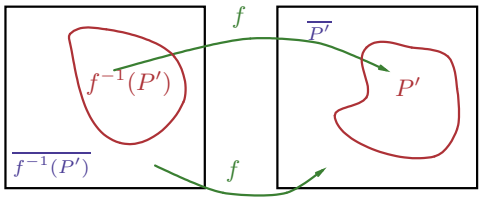


Abbildung 1. Funktionale Reduktion durch Urbildkonstruktion

Die Notation $P \leq P'$ kennzeichnet, daß das Problem (bzw. die Menge) P *nicht schwerer zu lösen* ist als P', denn die Reduktionsfunktion hilft, eine Lösungsmethode für das Problem P' in eine für das Problem P umzuwandeln. Zur Lösung von P transformiert man einfach eine Eingabe x in den Wert $f(x)$, wendet hierauf die Lösungsmethode für P' an und übernimmt die Antwort unverändert. Im Kontext von Entscheidbarkeit und Aufzählbarkeit ist dies möglich, weil entscheidbare bzw. aufzählbare Mengen abgeschlossen unter Urbild berechenbarer Funktionen sind (Sätze 8 und 9).

Theorem 18. *Es gelte $P \leq P'$.*
Wenn P' entscheidbar ist, dann ist auch P entscheidbar.
Wenn P' rekursiv aufzählbar ist, dann ist auch P rekursiv aufzählbar.

Abbildung 1 illustriert die Vorgehensweise der funktionalen Reduktion. Die Funktion f bildet das Problem P vollständig auf Elemente von P' ab, wobei keinesfalls jedes Element von P' getroffen werden muß. Wichtig ist nur, daß alle Objekte, die nicht zu P gehören, in Elemente des Komplementes von P' abgebildet werden. Insgesamt muß also $x \in P \Leftrightarrow f(x) \in P'$ für alle x gelten (was oft anschaulicher ist als $P = f^{-1}(P')$). Nur dann kann f die Lösungsmethode für P' in eine von P transformieren.

Man beachte, daß Reduzierbarkeit $P \leq P'$ nichts damit zu tun hat, ob $P \subseteq P'$ gilt oder nicht. So ist zum Beispiel jede entscheidbare Menge M reduzierbar auf die endliche Menge $\{1\}$, denn für alle n gilt $n \in M \Leftrightarrow \chi_M(n) \in \{1\}$. Umgekehrt ist das Selbstanwendbarkeitsproblem S eine Teilmenge der Menge $\mathbb{N}$, die trivialerweise entscheidbar ist. Da S aber nicht entscheidbar ist, muß $S \not\leq \mathbb{N}$ gelten. Reduzierbarkeit hängt also nicht von der Größe der Mengen ab, sondern von der Frage, wie kompliziert ihre Struktur ist.

Interessanter als die Tatsache, daß sich die Lösbarkeit vom schwereren auf das leichtere Problem überträgt ist deren Umkehrung, nämlich daß die Unlösbarkeit von des leichteren Problems auch die Unlösbarkeit des schwereren zur Folge hat.

Korollar 19 *Es gelte $P \leq P'$.*
Wenn P unentscheidbar ist, dann ist auch P' unentscheidbar.
Wenn P nicht rekursiv aufzählbar ist, dann ist auch P' nicht rekursiv aufzählbar.

Damit werden Unlösbarkeitsbeweise erheblich einfacher. Man muß nur ein bekanntes unlösbares Problem P finden, das sich auf das gesuchte Problem P' reduzieren läßt, hierfür eine Reduktionsfunktion f angeben und nachweisen, daß tatsächlich $P = f^{-1}(P') = \{x \mid f(x) \in P'\}$ gilt. Mit Korollar 19 folgt dann die Unlösbarkeit von P'.[3] Wir wollen dies an einigen Beispielen illustrieren.

[3] In informalen Formulierungen spricht man aus diesem Grunde gelegentlich davon, daß die *Unlösbarkeit von P' auf die von P reduziert* wird. Hierbei wird der Begriff der Reduzierbarkeit

Theorem 20 (Wichtige funktionale Reduzierbarkeiten).
Es sei z *die konstante Nullfunktion und* $PROG_z = \{i \mid \varphi_i{=}z\}$.

$S{\leq}H$**:** *Das Selbstanwendbarkeitsproblem ist reduzierbar auf das Halteproblem.*

$\overline{H} \leq TR$**:** *Das Komplement von* H *ist reduzierbar auf die total-rekursiven Programme.*

$\overline{H} \leq PROG_z$**:** $\overline{H}$ *ist reduzierbar auf die Programme der Nullfunktion.*

Beweis:

$S{\leq}H$**:** Es gilt $i \in S \Leftrightarrow \langle i,i\rangle \in H$. Wir definieren die Funktion f durch $f(i) := \langle i,i\rangle$. Dann ist f total-berechenbar und $S = f^{-1}(H)$.

$\overline{H} \leq TR$**:** Es gilt $\langle i,n\rangle \in \overline{H} \Leftrightarrow \forall t \in \mathbb{N}.\, \Phi_i(n){\neq}t$.

Da Rechenzeit universell entscheidbar ist, gibt es eine berechenbare Funktion h mit der Eigenschaft $h\langle i,n,t\rangle = \begin{cases} \bot \text{ falls } \Phi_i(n){=}t \\ 0 \text{ falls } \Phi_i(n){\neq}t \end{cases}$

Nach dem snm-Theorem existiert eine total-berechenbare Funktion f mit der Eigenschaft $\varphi_{f\langle i,n\rangle}(t) = h\langle i,n,t\rangle$ für alle i,n,t. Damit ist $\varphi_{f\langle i,n\rangle}$ die konstante Nullfunktion, wenn $\langle i,n\rangle \in \overline{H}$ und ansonsten an genau einer Stelle nicht definiert:

$\langle i,n\rangle \in \overline{H} \;\Rightarrow\; \forall t \in \mathbb{N}.\, \Phi_i(n){\neq}t \;\Rightarrow\; \forall t \in \mathbb{N}.\, \varphi_{f\langle i,n\rangle}(t){=}0 \;\Rightarrow\; f\langle i,n\rangle \in TR$

$\langle i,n\rangle \notin \overline{H} \;\Rightarrow\; \exists t \in \mathbb{N}.\, \Phi_i(n){=}t \;\Rightarrow\; \exists t \in \mathbb{N}.\, \varphi_{f\langle i,n\rangle}(t){=}\bot \;\Rightarrow\; f\langle i,n\rangle \notin TR$

Insgesamt folgt also $\overline{H}{=}f^{-1}(TR)$.

$\overline{H} \leq PROG_z$**:** Der Beweis ist fast identisch und wird in verkürzter Form angegeben.

Nach dem snm-Theorem gibt es eine total-berechenbare Funktion f mit der Eigenschaft $\varphi_{f\langle i,n\rangle}(t) = \begin{cases} 1 \text{ falls } \Phi_i(n){=}t \\ 0 \text{ falls } \Phi_i(n){\neq}t \end{cases}$. Damit ist

$\langle i,n\rangle \in \overline{H} \;\Leftrightarrow\; \forall t \in \mathbb{N}.\, \Phi_i(n){\neq}t \;\Leftrightarrow\; \forall t \in \mathbb{N}.\, \varphi_{f\langle i,n\rangle}(t){=}0 \;\Leftrightarrow\; f\langle i,n\rangle \in PROG_z$

Insgesamt folgt also $\overline{H}{=}f^{-1}(PROG_z)$. □

Damit hätten wir bis auf die Unentscheidbarkeit des Selbstanwendbarkeitsproblems (Theorem 15) eigentlich kein Problem mit Diagonalisierung behandeln müssen. Die Unentscheidbarkeit des Halteproblems folgt mit Korollar 19 aus $S{\leq}H$. Die Nicht-Aufzählbarkeit von $\overline{H}$ ergibt sich hieraus und aus der Aufzählbarkeit von H mit Theorem 10. Hieraus folgt wiederum Nicht-Aufzählbarkeit der total-rekursiven Funktionen Korollar 19 aus $\overline{H} \leq TR$.

Unlösbarkeitsbeweise mit Hilfe von Reduzierbarkeit sind eigentlich nur eine der Möglichkeiten, die Abschlußeigenschaften entscheidbarer und aufzählbarer Mengen für den Nachweis negativer Resultate zu verwenden. Dreht man die Ergebnisse der Sätze 10, 8 und 9 um, so erhält man sofort die folgenden Erkenntnisse.

1. Ist P aufzählbar, aber unentscheidbar, dann ist $\overline{P}$ nicht aufzählbar.
2. Ist P unentscheidbar dann ist auch $\overline{P}$ unentscheidbar.
3. Ist P entscheidbar aber $P{\cup}P'$, $P{\cap}P'$ oder $P{\setminus}P'$ unentscheidbar, dann ist P' nicht entscheidbar.
4. Ist P aufzählbar aber $P{\cup}P'$ oder $P{\cap}P'$ nicht, dann ist P' nicht aufzählbar.

---

aber mehr in einem umgangssprachlichen Sinne verwendet und die Richtung der Reduktion verdreht. Deswegen sollte man sich in der Literatur immer die genaue Definition ansehen.

In den meisten Fällen beschränkt sich die Anwendung von Abschlußeigenschaften in Unlösbarkeitsbeweisen jedoch auf das Komplement und das Urbild unter totalen berechenbaren Funktionen, also Reduzierbarkeiten.

3.3 Der Satz von Rice

In den vergangenen Abschnitten haben wir die Unlösbarkeit einer Reihe von Problemen nachgewiesen und die praktischen Konsequenzen dieser Aussagen diskutiert. Die *Unentscheidbarkeit des Halteproblems* bedeutet, daß es kein allgemeines Verfahren geben kann, mit dem sich die Terminierung von Programmen prüfen läßt. Die *Nicht-Aufzählbarkeit der total-rekursiven Funktionen* hat zur Folge, daß man keine Programmiersprachen entwerfen kann, deren Programme immer terminieren, ohne dafür Einschränkungen in der Ausdruckskraft hinnehmen zu müssen. Wie aber sieht es mit anderen Fragestellungen aus?

In Anbetracht der vielen Probleme mit fehlerhafter Software wäre es äußerst wichtig, die *Korrektheit von Programmen* testen zu können. Aber kann man von einem beliebigen Programm entscheiden, ob es eine bestimmte Funktion berechnet oder nicht (*Korrektheitsproblem*)? Leider ist eine positive Antwort nicht zu erwarten, da man diesen Test nicht einmal für die konstante Nullfunktion durchführen kann.

Aus dem gleichen Grund ist auch für das *Spezifikationsproblem* keine positive Antwort zu erwarten. Die Frage, ob man von einem beliebigen Programm entscheiden kann, ob es bestimmte spezifizierte Eigenschaften besitzt oder nicht, ist eine Verallgemeinerung des Korrektheitsproblems, und damit nicht leichter zu bentworten.

In der Softwareentwicklung werden vielfältige Optimierungstechniken eingesetzt, welche die Effizienz eines vorgegebenen Programms steigern sollen, und häufig führen auch die Programmierer selbst noch Optimierungen von Hand durch. Dabei entsteht die Frage, ob das optimierte Programm wirklich die gleiche Funktionalität besitzt wie das ursprüngliche oder ob diese durch die Optimierungen verändert wurde. Auch hier ist nicht zu erwarten, daß man dies mit einem automatischen Verfahren testen kann, denn die Frage, ob man für zwei beliebigen Programm entscheiden kann, ob sie die gleiche Funktionalität besitzen (*Äquivalenzproblem*), ist ebenfalls eine Verallgemeinerung des Korrektheitsproblems.

Die obigen informalen Argumente deuten an, daß man derartige Fragen mithilfe von Reduktionsbeweisen lösen könnte. Allerdings scheint dieser Weg recht mühsam. Außerdem entsteht in Anbetracht der vielen negativen Aussagen der Eindruck, daß es noch viel mehr unlösbare Fragestellungen in der Informatik geben wird.

Gibt es einen einfacheren Weg, diese Fragen anzugehen, anstatt jedes Mal einen neuen Reduktionsbeweis zu führen? Der folgende Satz gibt darauf eine allgemeine Antwort. Er besagt, daß so gut wie keine Eigenschaft berechenbarer Funktionen entscheidbar sein kann.

Theorem 21 (Satz von Rice).
Es sei P eine nichttriviale Teilmenge der berechenbaren Funktionen, also $\emptyset \neq P \subset \mathcal{R}$. Dann ist die Menge $L_P = \{i \mid \varphi_i \in P\}$ der Indizes von P unentscheidbar.

Beweis:

Wir zeigen, daß das Selbstanwendbarkeitsproblem auf die Menge L_P oder ihr Komplement reduziert werden kann. Dazu sei g die nirgends definierte Funktion, also $g(x)=\perp$ für alle x.

Falls $g \notin P$ ist, dann wählen wir eine beliebige Funktion $f \in P$ und definieren

$$f'\langle i,x\rangle = f(x) * \psi_S(i) = \begin{cases} f(x) & \text{falls } i \in S \\ \perp & \text{sonst} \end{cases}$$

Da f' berechenbar ist, gibt es nach dem smn-Theorem eine total-berechenbare Funktion h mit der Eigenschaft $\varphi_{h(i)}(x) = f'\langle i,x\rangle$ für alle i,x und es folgt

$i \in S \Rightarrow \forall x.\, \varphi_{h(i)}(x)=f(x) \Rightarrow \varphi_{h(i)}=f \in P \Rightarrow h(i) \in L_P$

$i \notin S \Rightarrow \forall x.\, \varphi_{h(i)}(x)=\perp \Rightarrow \varphi_{h(i)}=g \notin P \Rightarrow h(i) \notin L_P$

Insgesamt gilt also $i \in S \Leftrightarrow h(i) \in L_P$, d.h. $S \leq L_P$ und damit ist L_P unentscheidbar.

Falls $g \in P$ ist, dann wählen wir eine beliebige Funktion $f \notin P$ und zeigen analog, daß $S \leq \overline{L_P}$ gilt, woraus ebenfalls die Unentscheidbarkeit von L_P folgt. □

Wenn man sich den obigen Beweis genau ansieht, dann stellt man sogar fest, daß L_P wegen der Reduktion $S \leq \overline{L_P}$ (und damit $\overline{S} \leq L_P$) nicht einmal rekursiv aufzählbar ist, wenn die Eigenschaft P die nirgends definierte Funktion umfaßt. Es ist zu vermuten, daß dies auch für viele andere Eigenschaften der Fall ist. Dies geht aber weit über die Aussage des eigentlichen Satzes von Rice hinaus.

Weniger formal ausgedrückt besagt der Satz von Rice, daß *keine nichttriviale extensionale Eigenschaft von Programmen entscheidbar* ist. Natürlich ist es möglich, syntaktische Eigenschaften von Programmen zu testen, also z.B. ob alle Variablen deklariert oder sauber typisierbar sind, wieviel verschachtelte Ausdrücke vorkommen, etc. Aber derartige Eigenschaften sind *intensional*, beschreiben also die innere Struktur der Programme. Hat aber die Eigenschaft nur mit dem äußeren Verhalten, also der Funktionalität des Programms, zu tun, dann ist es unmöglich, sie zu testen. Wir wollen die Bedeutung dieser Aussage anhand einiger Beispiele illustrieren.

Beispiel 11 (Konsequenzen des Satzes von Rice)
Die folgenden Mengen sind unentscheidbar.

- $MON = \{i \mid \forall k \in \mathbb{N}\ \ \varphi_i(k) < \varphi_i(k+1)\}$ *(Monotone Funktionen)*
- $EF = \{i \mid \forall j \in \mathbb{N}\ \ \varphi_i(j) \in \{0,1\}\}$ *(Entscheidungsfunktionen)*
- $PROG_f = \{i \mid \varphi_i = f\}$ *für beliebige* $f \in \mathcal{R}$ *(Korrektheitsproblem)*
- $PROG_{spec} = \{i \mid \varphi_i$ *erfüllt Spezifikation* $spec\}$ *(Spezifikationsproblem)*
- $EQ = \{\langle i,j\rangle \mid \varphi_i = \varphi_j\}$ *(Äquivalenzproblem)*
- $RG = \{\langle i,j\rangle \mid j \in$ *range*$(\varphi_i)\}$ *(Bildbereiche)*

Der Beweis verläuft in jedem Fall gleich. Wir müssen die zu untersuchende Menge als Menge L_P einer Menge P von Funktionen ausdrücken und dann nachweisen, daß P weder leer ist noch alle berechenbarer Funktionen umfaßt. Dies ist meist sehr einfach. So ist z.B. für die monotonen Funktionen $P = \{f \in \mathcal{R} \mid \forall k \in \mathbb{N}\ \ f(k) < f(k+1)\}$ und man erkennt sofort, daß $MON = L_P = \{i \mid \varphi_i \in P\}$ ist. Außerdem ist P nicht leer, da $f = \lambda i.i$ zu P gehört, und eine echte Teilmenge von $\mathcal{R}$, da $g = \lambda i.0$ nicht zu P gehört.

Da es also unmöglich ist, Programmeigenschaften automatisch zu testen, müssen Korrektheitsbeweise notgedrungenermaßen von Hand geführt werden. Nun sind Korrektheitsbeweise schon für kleinere Programme verhältnismäßig aufwendig. Daher verläßt

man sich in der Praxis meist auf Softwaretests. Man muß sich aber bewußt sein, daß erfolgreich bestandene Softwaretests keinerlei Garantie für die Korrektheit darstellen. Dies ist in der Vergangenheit leider oft mißachtet worden und hat zu gravierenden Fehlern mit erheblichen Folgeschäden geführt. Zumindest in kritischen Anwendungen sind Korrektheitsbeweise unbedingt erforderlich.

Um dieser Problematik zu begegnen, richtet sich ein Teil der Grundlagenforschung in der Informatik auf die Entwicklung von logischen Werkzeugen zur rechnergestützten und teilautomatisierten Beweisführung. Dies ist jedoch ein Thema für weiterführende Lehrveranstaltungen.

Literatur

1. Klaus Weihrauch. *Computability*. Springer Verlag, 1987. 4
2. Hartley jr. Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, 1967.