4 Berechenbarkeit und deren Grenzen

4.1 Berechnungsmodelle und Church'sche These

Die Berechenbarkeitstheorie beschäftigt sich mit folgenden Fragestellungen:

- Was ist eine berechenbare Funktion bzw. ein algorithmisch-lösbares Problem?
- Wie formalisiert man das Konzept „Algorithmus"?
- Gibt es nicht-berechenbare Funktionen?
- Gibt es Problemstellungen, die wohldefiniert, aber nicht algorithmisch lösbar sind?

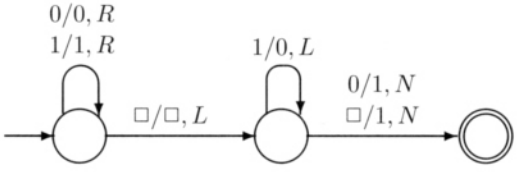
Dahinter verbergen sich mehrere Ansinnen. Zum einen geht es darum, überhaupt erst einmal eine adäquate Definition der Begriffe „Algorithmus" und „berechenbar" zu finden. Zum anderen steckt hier ein bisschen Philosophie oder Erkenntnistheorie dahinter: Was können wir überhaupt mit Computern berechnen; gibt es Grenzen der Algorithmisierbarkeit? Um die Antworten gleich vorweg zu nehmen: Ja, Berechenbarkeit und Algorithmus lassen sich in zufriedenstellender Weise formalisieren, und es gibt hier wirklich nur *eine* (und zwar die richtige) Definition. Es bleiben keine Mehrdeutigkeiten, alternative Definitionen und andere Zweifel übrig. Und ja, es gibt algorithmisch unlösbare Probleme. Es gibt eine klare Grenze, die berechenbare Funktionen von nicht-berechenbaren unterscheidet. Für manche praktischen Anwendungen ist das eine Situation, mit der man leben muss. Einige Konsequenzen aus der Unlösbarkeit bestimmter Aufgabenstellungen sind zum Beispiel, dass es eine vollständig automatische Programmverifikation, die garantiertermaßen fehlerfreie Programme ermöglichen würde, nicht gibt. Ebenso wenig gibt es intelligente (deduktive) Datenbanksysteme, die aus den gespeicherten Fakten jede nur denkbare logisch-korrekte Folgerung ziehen können.

Beginnen wir mit den formalen Konzepten „berechenbar" und „Algorithmus". Turing war einer der ersten, der sich mit diesem Thema beschäftigte. Sein (Definitions-) Vorschlag von 1936 ist, Berechenbarkeit mithilfe der Turingmaschine zu definieren: Eine Funktion (etwa auf den natürlichen Zahlen) ist berechenbar, wenn es eine Turingmaschine gibt, die diese Funktion berechnen kann. Das heißt, wenn man eine entsprechende Turingmaschine auf eine beliebige Eingabe x ansetzt (wobei x als Binärdarstellung einer natürlichen Zahl interpretiert wird),

aus
Uwe Schöning,
Ideen der Informatik. München: Oldenbourg 2002

so stoppt die Maschine nach endlich vielen Schritten und liefert auf ihrem Arbeitsband das Rechenergebnis, nämlich $f(x)$, ab.

Beispiel: Die folgende Turingmaschine berechnet die Funktion $f(x) = x+1$. Wenn man diese Maschine auf eine Zahl x (in Binärdarstellung) ansetzt, so stoppt diese nach endlich vielen Schritten, indem sie einen Endzustand betritt, und auf dem Arbeitsband steht dann die Zahl $x + 1$ (in Binärdarstellung). Also ist $f(x) = x + 1$ eine (im Sinne von Turing) berechenbare Funktion.



Diese Turingmaschine arbeitet wie folgt. Zunächst bewegt sich der Schreib-Lese-Kopf (ohne Veränderung des Bandinhalts) ganz nach rechts bis zur niederwertigsten Bitstelle. Sofern dieses Bit Null ist, wird es auf Eins gesetzt und wir sind fertig. Wenn es eine Eins ist, so wird es auf Null gesetzt und eine Position weiter links mit derselben Aktion wie zuvor fortgefahren. Im Extremfall können diese Überträge sich bis in die erste Position fortpflanzen, so dass vorne eine Eins angefügt werden muss.

Man fragt sich natürlich sofort, ob Turingmaschinen nicht zu einfache Konzepte sind, um wirklich alle Funktionen, die man intuitiv als berechenbar ansehen muss, tatsächlich auch berechnen zu können. Deshalb sollte man Turings Vorschlag mit einem möglichst mächtigen Berechnungsmodell vergleichen. Wir stellen uns einen Computer vor, der (beliebig viele) Speicherzellen besitzt. Jede solche Speicherzelle kann als Inhalt eine (beliebig große) natürliche Zahl aufnehmen. Unser Computer ist ferner mit einem üblichen Befehlssatz ausgestattet, der es gestattet, Speicherinhalte in andere Speicherzellen zu kopieren und Rechenoperationen wie Addition, Subtraktion, Multiplikation, (ganzzahlige) Division auszuführen. Diese Programme verwenden Variablen-Bezeichnungen $x_1, x_2, \ldots$, welche sich auf die durchnummerierten Speicherzellen beziehen. Programme für diesen Computer dürfen darüber hinaus die üblichen Ablaufsteuerungskonstrukte wie Schleifen, Sprünge, Unterprogramme und Abfragen von Bedingungen enthalten. Wenn eine Funktion (auf den natürlichen Zahlen) durch einen Computer wie diesen (welcher manchmal auch *Registermaschine* genannt wird) berechenbar ist, so wollen wir die Funktion *Registermaschinen-berechenbar* nennen.

Man kann zeigen, dass Registermaschinen-Berechenbarkeit und Turings Vorschlag von Berechenbarkeit ein und dasselbe sind. Das heißt insbesondere, dass jedes Registermaschinen-Programm zur Berechnung einer Funktion f in ein Programm für eine Turingmaschine umgeschrieben werden kann. Die Turingmaschine wird natürlich wesentlich mehr Schritte benötigen. Aber es geht hier nur ums Prinzip, nicht um Effizienz.

Es wurden auch noch ganz anders geartete Vorschläge gemacht, um Berechenbarkeit zu definieren. Ein ganz anderer, mathematischer Ansatz wurde von S.C. Kleene (1936) vorgeschlagen.

Steven Cole Kleene

(1909–1994) Amerikanischer Mathematiker und Logiker. Studium der Mathematik in Amherst bis 1930. Promotion 1934 in Princeton. Professor an der University of Wisconsin in Madison 1935–1979. Wichtige Arbeiten auf dem Gebiet der Berechenbarkeitstheorie und Automatentheorie.

Es werden zunächst einige wenige Funktionen (wie zum Beispiel die Funktion $f(x) = x + 1$), sozusagen „per Axiom" als „berechenbar" erklärt. Der weitere Verlauf der Definition erfolgt nun induktiv. Es werden einige Konstruktionsprinzipien angegeben, die besagen, wie man aus Funktionen, die bereits „berechenbar" sind, weitere erhalten kann, die nun auch per Definition als „berechenbar" erklärt werden. Eine solche Konstruktionsvorschrift ist das Einsetzungsprinzip: Sofern f und g bereits „berechenbar" sind, so soll es auch die Funktion $f(g(x))$ sein. Ein weiteres Prinzip ist die Iteration (oder primitive Rekursion): Wenn f bereits „berechenbar" ist, so soll auch die zweistellige Funktion $g(n, x) = \underbrace{f(f(\ldots f}_{n\text{-mal}}(x)\ldots))$ „berechenbar" sein. Eine weitere Konstruktionsvorschrift ist die so genannte μ-Rekursion (oder μ-Operator): Wenn $f(n, x)$ bereits „berechenbar" ist, dann soll es auch die folgende Funktion sein: $g(x) = \min\{n \mid f(n, x) = 0\}$. Hierbei wird vorausgesetzt, dass das Minimum für jede Eingabe x tatsächlich existiert.

Wichtig ist noch die Beobachtung, dass diese Konstruktionsprinzipien tatsächlich „programmierbar" sind. Das heißt, die solcherart definierten Funktionen lassen sich tatsächlich berechnen. In einer Programmiersprachen-Terminologie bedeutet die Einsetzung nichts anderes als die Hintereinanderausführung zweier Wertzuweisungen. Die Iteration lässt sich mithilfe einer FOR-Schleife und die μ-Rekursion mithilfe einer WHILE-Schleife programmieren.

Einsetzung:	Iteration:	μ-Rekursion:
INPUT x;	INPUT n, x;	INPUT x;
$y := f(x)$;	$y := x$;	$n := 0$;
$z := g(y)$;	FOR $i := 1$ TO n	WHILE $f(n, x) \neq 0$
OUTPUT z;	DO $y := f(y)$;	DO $n := n + 1$;
	OUTPUT y;	OUTPUT n;

Diese „berechenbaren" Funktionen nach Kleene werden auch *μ-rekursive Funktionen* genannt. Lässt man das letzte Konstruktionsprinzip, die μ-Rekursion, weg, so erhält man eine echt kleinere Klasse von Funktionen, die so genannten *primitiv rekursiven* Funktionen. Wir kommen auf diese Klasse von Funktionen später nochmals zurück.

Feststellung

Es hat sich herausgestellt, dass alle vorgeschlagenen Definitionen für „Berechenbarkeit" (wie zum Beispiel Turingmaschinen-berechenbar, Registermaschinen-berechenbar, μ-rekursiv, aber auch andere, hier nicht besprochene, wie λ-definierbar, Markov-Algorithmus-berechenbar, Post-Kalkül-berechenbar, allgemein-rekursiv) untereinander äquivalent sind.

Insbesondere wurde bisher kein Berechnungsmodell gefunden, das in der Lage wäre, Funktionen zu berechnen, die eine Turingmaschine nicht (im Prinzip) auch berechnen könnte. Dies

gilt auch für die erst vor kurzem entdeckten Berechnungsmöglichkeiten, wie zum Beispiel stochastische Algorithmen, Quantencomputer oder DNA-Computer.
Aufgrund dieser Äquivalenztatsache hat schon Alonzo Church (1936) die folgende These aufgestellt, die bis heute akzeptiert wird. (Man beachte, dass man diese These nicht beweisen kann, denn sie stellt keine mathematische Aussage dar – allenfalls könnte man sie widerlegen.)

Church'sche These
Der Berechenbarkeitsbegriff wird (zum Beispiel) durch das Modell der Turingmaschine adäquat definiert.

Alonzo Church
(1903–1995) Amerikanischer Mathematiker und Logiker. Mathematik-Studium in Princeton. Promotion 1927. Studienaufenthalte in Harvard, Göttingen und Amsterdam. Seit 1929 Professor in Princeton. 1967 wechselte er an die Universität von Kalifornien in Los Angeles (UCLA) als Professor für Mathematik und Philosophie. Wichtige Beiträge im Bereich der Berechenbareitstheorie und mathematischen Logik. Church zeigte 1936 die Nicht-Entscheidbarkeit der Prädikatenlogik. Er erfand in den 30er Jahren den λ-Kalkül, der die Grundlage für funktionale Programmiersprachen wie LISP darstellt. Doktorvater von Alan Turing, Steven Kleene und Michael Rabin.