Turing, Church, Gödel und Rechenmaschinen

Ege Girit

12. Juli 2020, Dortmund

Inhaltsverzeichnis

1 Einleitung

Mit den Entwicklungen im Bereich Formalisierung der Mathematik, hat man sich immer mehr mit den Eigenschaften und Grenzen der Systeme beschäftigt als mit Theoremen, die in einem formalen System abgeleitet werden können. Ein formales System wird verwendet, um Sätze aus Axiomen mithilfe der definierten Regeln abzuleiten. Die drei wichtigsten Fragestellungen für ein formales System nach [1] sind:

Vollständigkeit: Ein formales System heißt vollständig, wenn jede wahre Aussage, innerhalb des Systems mithilfe einer endlichen Kette von Regelanwendungen beweisbar(ableitbar) ist.

Widerspruchsfreiheit: Ein formales System heißt widerspruchsfrei, wenn für eine Aussage ϕ niemals gleichzeitig ϕ und seine Negation ($\neg\phi$) abgeleitet werden kann. In einem widersprüchlichen System könnte man jede beliebige Aussage beweisen.

Entscheidbarkeit: Ein formales System heißt entscheidbar, wenn ein systematisches Verfahren existiert, mit dem für jede Aussage entschieden werden kann, ob sie innerhalb des Systems beweisbar ist.

Mit der Eigenschaft der Entscheidbarkeit hatte man den Wunsch, eine mechanisierte Mathematik zu schaffen. Wäre z. B. die Zahlentheorie[1] entscheidbar, so könnte man jede wahre zahlentheoretische Aussage systematisch beweisen. Heute wissen wir, dass es nicht möglich ist, die Mathematik in einem formalen System zusammenzufassen, in dem alle wahren mathematischen Aussagen als solche bewiesen werden können.

2 Was haben Alan Turing, Alonzo Church und Kurt Gödel gemeinsam?

Ein eng verwandter Begriff zur Entscheidbarkeit ist die **Berechenbarkeit**. Man bezeichnet eine Funktion genau dann berechenbar (auch in der Literatur verwendete Begriffe: rekursiv, effektiv berechenbar), wenn es einen Algorithmus für die Realisierung dieser Funktion existiert.

Es wurden in den 1930er Jahren mehrere unabhängige Versuche unternommen, den Begriff der Berechenbarkeit zu formalisieren. Einige der bedeutendsten Namen, die in diesem Gebiet gearbeitet haben, sind: Alan Turing, Alonzo Church und Kurt Gödel.
Alonzo Church gilt als einer der Begründer der theoretischen Informatik, der sich mit Mathematik, Logik und Philosophie beschäftigt hat. Er ist bekannt geworden für seine Entwicklung des Lambda-Kalküls. Der Lambda-Kalkül ist eine formale Sprache, mit der man Funktionen untersuchen kann. Es werden mithilfe des Lambda-Kalküls berechenbare

[1] Die Zahlentheorie(oder Arithmetik oder höhere Arithmetik) ist ein Teilgebiet der Mathematik, das sich hauptsächlich mit den Eigenschaften der ganzen Zahlen und ganzzahligen Funktionen beschäftigt.

Funktionen zu einer Klasse zusammengefasst. Alonzo Church hat mit seinem Lambda-Kalkül die Entwicklung funktionaler Programmiersprachen und die Typsystemeforschung von Programmiersprachen wesentlich beeinflusst. Er hat außerdem demonstriert, dass es unentscheidbare(nicht mathematisch berechenbare) Probleme gibt. Church's Entwicklung hat einen seiner Studenten, nämlich Alan Turing, zum Überlegen gebracht, ob das Halteproblem auch unentscheidbar sei.

Alan Mathison Turing hat die theoretische Grundlage der heutigen Computer mit seinem Turingmaschinenmodell geschaffen, das gleichmächtig zu heutigen Computern ist, obwohl es mit sehr beschränkten Funktionen arbeitet. Der Begriff der Turing-Maschine wurde durch Alonzo Church geprägt [1]. 1936 publizierte er mit „On computable numbers, with an application to the Entscheidungsproblem" eine der wichtigsten Arbeiten auf dem Gebiet der Logik.

In den Zeiten, in denen Turing lebte, gab es noch keine programmierbaren Rechner. Seine abstrakte Turingmaschine kann man sich als einen Menschen vorstellen, der nach einem festgelegten Verfahren etwas berechnet. Einer der Gründe, warum er die Turingmaschine konstruiert hat, war sein Wunsch, zu beweisen, dass es kein systematisches Verfahren gibt, das zu jeder mathematischen Aussage entscheidet, ob sie wahr oder falsch ist.

Dieses Problem ist auch unter dem Namen Allgemeingültigkeitsproblem bekannt. Die Formulierung des Allgemeingültigkeitsproblems lautet in [5] wie folgt:
Gegeben prädikatenlogische Formel F, dann ist die Frage, ob für alle passenden Modelle M die Eigenschaft gilt, dass M die Formel F erfüllt.

Seine Gedanken für die Berechenbarkeitstheorie ermöglichen, den Begriff des Algorithmus mathematisch präzise zu erfassen. Mit seinen Gedanken wurde es klar, dass es Probleme existieren, die sich nicht mithilfe systematischer Verfahren lösen lassen. Dadurch erhalten wir aber auch indirekte Fortschritte in der Beweistheorie, die ermöglichen, kürzere Beweise zu führen, was bisher nicht möglich gewesen war.

Church und Turing haben gemeinsam herausgefunden, dass die Turingmaschine und der Lambda-Kalkül die gleiche Ausdruckskraft haben. Sie haben bemerkt, dass es noch weitere äquivalente Modelle zum Berechnen von Funktionen gibt. Die sind zum Beispiel µ-rekursiven Funktionen, Zweikellerautomaten, WHILE-Programme, Registermaschinen und GOTO-Programme. Formal bedeutet das: Eine Funktion ist genau dann Lambda-berechenbar, wenn sie Turing-berechenbar ist, und genau dann, wenn sie allgemein rekursiv ist. Sie haben außerdem die nicht beweisbare (aber im Prinzip widerlegbare) **Church-Turing-These** aufgestellt, die besagt:

„Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein."

Weil man den Begriff „intuitiv berechenbare Funktion" nicht formal definieren kann, ist

diese These unbeweisbar. In der Informatik wird es in der Regel angenommen, dass die These wahr ist. Unter intuitiv berechenbaren Funktionen versteht man alle Funktionen, die auf irgendeine Weise berechnet werden können.
Es gibt auch eine erweiterte Version der Church-Turing-These, die besagt, dass alle „vernünftigen“ Berechnungsmodelle sich bezüglich ihres Zeitaufwandes nur höchstens polynomiell unterscheiden [5].

Im Laufe der Zeit wurden unterschiedliche Berechnungsmodelle entwickelt, man hat aber festgestellt, dass sie entweder weniger stark als die Turingmaschinen sind oder gleich stark sind. Daher besagt die Church-Turing-These, dass die Turingmaschinen den Begriff der Berechenbarkeit schildern. In der Berechenbarkeitstheorie werden genau die Funktionen Turing-berechenbar genannt, die berechenbar sind.

Kurt Friedrich Gödel war ein Mathematiker, Philosoph und Logiker. Durch seine grundlegende Entdeckungen in der Logik, mussten wir unsere Interpretation der bisher verwendeten mathematischen Methode durchaus überdenken. Er hat 1929 die Vollständigkeit des Funktionenkalküls bewiesen. Mit diesem Beweis war es möglich, ein formales System zu konstruieren, in dem jede allgemeingültige prädikatenlogische Formel erster Stufe in endlich vielen Schritten aus den Axiomen ableitbar sind.

Mit Jacques Herbrand erstellte Kurt Gödel die formale Definition der **Klasse der allgemein rekursiven Funktionen**, was wiederum gleichmächtig zu Turingmachinen ist.

Mit den rekursiven Funktionen beschreibt man nicht die einzelnen Schritte des Rechenverfahrens, sondern die Lösungen von Problemen. Dieses Berechnungsmodell ist daher ähnlich wie die funktionalen Programmiersprachen. Er hat die rekursiven Funktionen zum Beweis seines Unvollständigkeitssatzes verwendet.

Die Gödel'schen Unvollständigkeitssätze sind sehr wichtig für die Beweistheorie. Der erste Gödel'sche Unvollständigkeitssatz lautet in [1] wie folgt: "Die Begriffe der Wahrheit und der Beweisbarkeit lassen sich in hinreichend ausdrucksstarken formalen Systemen nicht in Einklang bringen". Das zeigt also die Existenz unbeweisbarer Aussagen in hinreichend starken widerspruchsfreien Systemen. Der Zweite Unvollständigkeitssatz besagt, dass die Widerspruchsfreiheit von hinreichend starken widerspruchsfreien Systeme innerhalb des Systems nicht beweisbar sind. Das System kann also seine eigene Widerspruchsfreiheit nicht beweisen.

Aus seinen Unvollständigkeitsergebnissen hatte dann Gödel **Interesse an einer genauen Definition der Berechenbarkeit**. Die genaue Definition hatte er dafür gebraucht, um die Allgemeinheit seiner Ergebnisse festzustellen. Alonzo Church hatte Gödel vorgeschlagen, die "berechenbaren" Funktionen als die Lambda-Funktionen zu definieren. Gödel war nicht überzeugt von Church's Idee und bezeichnete den Vorschlag als "völlig unbefriedigend". Gödel wollte den Begriff der "effektiven Berechenbarkeit" axiomatisieren. Er hat aber nach wenigen Jahren diesen Vorschlag doch akzeptiert, weil die Arbeit von Turing seine Meinung geändert habe. Laut Gödel hat Turing eine genaue Definition eines formalen Systems geliefert. In [6] werden die Ansichten von Turing und Gödel normalerweise als Kontrast zueinander gesehen. Die Diskussion zwischen Turing und Gödel

war es, ob Maschinen niemals Mathematiker ersetzen können, oder allgemeiner, ob der Mensch jede Maschine übertrifft. Allgemein wird es nach [6] angenommen, dass Gödel der Meinung ist, dass der Geist(Mensch) mächtiger als die Maschine sei, während Turing die gegenteilige Position eingenommen haben soll.

3 Turingmaschinen

Das Konzept eines Computers hat Alan Turing in seiner Arbeit wie folgt beschrieben: Eine Turing-Maschine hat eine endliche Anzahl von **Zuständen**. Die Gesamtheit der Zustände wird als Turingprogramm bezeichnet. Es gibt ein **Speicherband**, das in mehrere Felder unterteilt ist. Es gibt ein **Alphabet**, das die zulässigen Symbole für die Maschine beschreibt. In jedem Feld des Speicherbands kann ein Symbol aus diesem Alphabet stehen. In einem Schritt kann die Maschine nur ein Feld lesen. Es existiert ein **Zeiger**, der auf das aktuell zu lesende Feld zeigt.

Das **Verhalten** der Turingmaschine wird durch den entsprechenden Zustand und das gelesene Symbol entschieden. Das mögliche Verhalten(Funktionen) der Turing-Maschine wird in elementare Operationen zusammengefasst, die nicht weiter zerlegt werden können und aus denen sich komplexe Berechnungen zusammensetzen. Die Operationen einer Turingmaschine sind:

- Auf ein Feld ein Symbol schreiben
- Den Inhalt eines Feldes löschen
- Den Zeiger ein Feld nach rechts oder links bewegen(oder gar nicht bewegen)
- Stoppen

Die Kombination aus Zustand, gelesenem Symbol und dem Zeigerposition nennt man eine **Konfiguration**. Eine Turingmaschine wird also definiert durch die Summe ihrer Konfigurationen.

Mathematisch kann man eine Turing-Maschine (wie es in [5] zu finden ist) wie folgt formulieren:
Eine Turingmaschine ist ein 4-Tupel M = (Q, Σ, δ, s) bestehend aus:
– einer endliche Menge Q, (Zustände)
– einem Alphabet Σ, mit $\epsilon, \$ \in \Sigma$, (Arbeitsalphabet mit speziellem Blank- und Bandsymbol)
– einer Funktion $\delta : Q \times \Sigma \to (Q \cup \{ja, nein, h\}) \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$ (Transitionsfunktion),
– einem ausgezeichneten Zustand $s \in Q$ (Startzustand)
Wobei:
– $Q, \Sigma, \{ja, nein, h\}$ und $\{\leftarrow, \downarrow, \rightarrow\}$ paarweise disjunkte Mengen sind.

Es existieren unterschiedliche Varianten der Turing-Maschine, die die originale Turing-Maschine leicht modifizieren. Hier sind einige Beispiele aufgelistet:

- Mehrband-Turingmaschinen: mehrere Zeigerköpfe/Bänder (mit je einem Kopf).
- Nichtdeterministische Turingmaschinen: mehrere mögliche Transitionen.
- Mehrdimensionale Turingmaschinen: die Felder sind n-dimensionale Arrays (Adressierung mittels $\mathbb{Z}^n$).
- Halbband-Turingmaschinen: das Band (oder die Bänder) nur nach rechts unbeschränkt.

Eine partielle Funktion $f : \Sigma^* \to \Sigma^*$ heißt **Turing-berechenbar**, wenn es eine Turingmaschine existiert, die die Funktion f berechnet. Man verwendet hier den Begriff der partiellen Funktion, weil die Berechnung möglicherweise nicht immer terminiert oder ein ablehnender Zustand erreicht wird.

4 Turingmaschinen und GOTO-Programme

Wir haben früher erwähnt, dass es weitere äquivalente Mechanismen zu Turingmaschinen gibt, die die gleiche Ausdruckskraft haben. Ein Beispiel davon sind die GOTO-Programme.

Ein GOTO-Programm beinhaltet eine endliche Folge von Marken (Labels) und Anweisungen. Dabei sind folgende Anweisungen möglich: Addition, modifizierte Subtraktion, Programmende, Sprung und bedingter Sprung. Zwischen Turingmaschinen und GOTO-Programmen gilt der folgende Zusammenhang:

> *Jede Turing-berechenbare Funktion ist auch GOTO-berechenbar[2] und jede partielle oder totale GOTO-berechenbare Funktion ist auch Turing-berechenbar[4].*

Man kann also jede (deterministische) Turingmaschine durch ein GOTO-Programm simulieren. Der Beweis dafür kann über WHILE-Programme erfolgen. GOTO-Programme sind mindestens so mächtig wie WHILE-Programme und WHILE-Programme mindestens so mächtig sind wie deterministische Turingmaschinen. Dann schließt man einen Kreis, indem man zeigt, dass Turingmaschinen mindestens so mächtig wie WHILE-Programme sind.

5 Das Halteproblem

Das erste Hauptresultat von Turing stellt im Grunde die Unlösbarkeit des Halteproblems dar. Man sagt, dass das Bewusstsein dieser Grenzen den Menschen von jeder intelligenten Maschine unterscheidet. Das Halteproblem beschäftigt sich mit den Terminierungseigenschaften von Turing-Maschinen. Es geht um die Frage, ob auf algorithmischem Wege entschieden werden kann, ob eine Turing-Maschine unter bestimmten Eingaben terminiert oder endlos weiter rechnet. Das Halteproblem ist unlösbar, es kann also keinen Algorithmus geben, der dieses Problem entscheidet.

6 Beweis von Entscheidbarkeitseigenschaften des Halteproblems

Für den Beweis der Unlösbarkeit von dem Halteproblem, argumentiert man mit Kontraposition. Wir machen uns Gedanken darüber, ob es einen Algorithmus dafür überhaupt geben kann. Wir nehmen an, wir hätten eine korrekte Implementierung. Unser Ziel ist einen Widerspruch zu unserer Annahme zu bekommen. Da wir eine Implementierung für das Problem haben, wäre dann das Problem nach Definition entscheidbar.

Wir erstellen eine Matrix, bei der auf der vertikalen Achse alle Turing-Maschinen und auf der horizontalen Achse alle Eingabewörter stehen[2]. Die Felder der Matrix geben uns Informationen über das Terminierungsverhalten der Turingmachinen. In diesen Feldern kann „Terminiert“ oder „Nicht terminiert“ stehen. Falls z.B. ein Eintrag in der i-ten Zeile und der j-ten Spalte als „Terminiert“ bezeichnet ist, dann würde das bedeuten, dass die Turing-Maschine i unter Eingabe von j terminiert. Die Reihenfolge der Zeilen und Spalten ist nicht wichtig, wir möchten nur erreichen, dass jede Turing-Maschine und jedes Eingabewort in irgendeiner Zeile oder Spalte erscheinen.

Falls das Halteproblem entscheidbar ist, müsste es dann eine Turing-Maschine M geben, die für eine Turing-Maschine T und Eingabewort w immer korrekt entscheidet, ob T bei Eingabe von w terminiert. Man kann dann aus M wieder eine zweite Turing-Maschine M' konstruieren, die genau dann für das Eingabewort w_i terminiert, wenn die Turing-Maschine M_i bei Eingabe von w_i unendlich lange rechnet. Weil M' eine Turing-Maschine ist, muss M' in einer Stelle von unserer Matrix auftauchen. Wenn man aber M' in der Matrix sucht, kommt man zu einem Widerspruch für die Konstruktion von M, weil alle Turing-Maschinen in der Matrix eine Eingabe genau dann akzeptieren, wenn sie von M' abgelehnt wird. Daraus schließen wir, dass unsere Konstruktion der Maschine M nicht existieren kann, und das Halteproblem ist damit unlösbar.

[2]Den detallierten Beweis, der diese Idee verfolgt, kann man in [1] finden.

7 Quellen

1- Grenzen der Mathematik - Eine Reise durch die Kerngebiete der mathematischen Logik - Dirk W. Hoffmann

2- Formale Grundlagen der Programmierung - Markus Nebel

3- Didaktik der Informatik - Peter Hubwieser

4- Theoretische Informatik - Lutz Priese und Katrin Erk

5- Grundlagen der Theoretischen Informatik Skript SS18 TU Dortmund - Thomas Schwentik

6- Computability: Turing, Gödel, Church, and Beyond - B. Jack Copeland , Carl J. Posy , Oron Shagrir