Matheschülerzirkel
Universität Augsburg
Schuljahr 2014/2015
Klassen 10/11/12

Gödels Unvollständigkeitssatz

Zirkelzettel vom 7. November 2014

Inhaltsverzeichnis

1 Einleitung

2 Berrys Paradoxon

3 Das Halteproblem

Manche Computerprogramme stoppen nach endlich vielen Rechenschritten („halten“), andere nicht. Etwa hält das Programm „Lese vom Benutzer eine Zahl ein, verdopple diese Zahl und gebe das Ergebis aus“, während das Programm „Gebe alle natürlichen Zahlen aus“ nicht hält. Im Allgemeinen ist es sehr schwer, einem Programm anzusehen, ob es hält oder nicht.

Beispiel 3.1. Die *Goldbachsche Vermutung* besagt, dass jede gerade Zahl größer als 2 die Summe zweier Primzahlen ist. Für jede konkrete gerade Zahl n größer als 2 kann man das leicht überprüfen, indem man einfach alle Paare von Primzahlen, die kleiner als n sind, durchgeht; etwa sieht man durch Ausprobieren, dass $14 = 3 + 11$ als Summe zweier Primzahlen geschrieben werden kann. Noch ist die Vermutung im allgemeinen Fall aber ein offenes Forschungsproblem. Daher ist nicht klar, ob folgendes Programm hält oder nicht:

1. Beginne mit $n := 4$.

2. Prüfe, ob n die Summe zweier Primzahlen ist.
3. Falls ja: Erhöhe n um Zwei und gehe zurück zu Schritt 2.
4. Falls nein: Halte.

Dieses Programm hält genau dann, wenn es ein Gegenbeispiel zur Goldbachschen Vermutung gibt.

Beispiel 3.2. Eine *Fermatsche Primzahl* ist eine Primzahl der Form $2^{(2^n)} + 1$. Die Primzahlen 3, 5, 17, 257 und 65537 sind von diesem Typ (für $n = 0, 1, 2, 3, 4$), aber es ist ein offenes Forschungproblem, ob es weitere Fermatsche Primzahlen gibt. Daher ist von folgendem Programm nicht klar, ob es hält:

1. Beginne mit $n := 5$.
2. Prüfe, ob $2^{(2^n)}$ eine Primzahl ist.
3. Falls ja: Halte.
4. Falls nein: Erhöhe n um Eins und gehe zurück zu Schritt 2.

Beim *Halteproblem* geht es darum, von einem gegebenen Programm festzustellen, ob es hält oder nicht. Der britische Logiker, Mathematiker, Kryptoanalytiker und Informatiker Alan Turing[1] (* 1912, † 1954) bewies 1937, dass das Halteproblem *nicht entscheidbar* ist. Genau formuliert bedeutet das:

Theorem 3.3. *Es gibt kein Programm, das bei Eingabe eines Programms P mit einer korrekten Ausgabe von „P hält“ oder „P hält nicht“ hält.*

Ein hypothetisches solches Programm wird auch *Halteorakel* genannt. Bemerkenswert an dem Theorem ist, dass es eine absolute Aussage trifft – Turing behauptet nicht nur, dass *wir* kein solches Halteorakel kennen. Diese schwächere Behauptung ist auch gar nicht erstaunlich, erfordert doch im Allgemeinen die Lösung des Halteproblems beliebige offene mathematische Probleme zu lösen (Beispiel 3.1). Vielmehr behauptet Turing, dass ein Halteorakel rein prinzipiell nicht existieren kann. Auch Außerirdische mit überlegener Technologie oder transzendente Wesen können kein Halteorakel programmieren.

Bemerkung 3.4. Programme können durchaus andere Programme simulieren. Das hilft aber bei der (zum Scheitern verurteilten) Konstruktion eines Halteorakels nicht: Wir können zwar ein gegebenes Programm P simulieren und, zum Beispiel, 10.000 Rechenschritte abwarten. Wenn P bis dahin aber nicht gehalten hat, wissen wir immer noch nicht, ob P später halten wird oder nicht.

Bemerkung 3.5. Einzelne Instanzen des Halteproblems können durchaus algorithmisch lösbar sein,[2] Turings Resultat sagt nur aus, dass es kein *einzelnes* Programm gibt, welche *alle* Instanzen des Problems lösen kann – welches also bei Eingabe eines *beliebigen* Programms in endlicher Zeit mit der Meldung „hält“ oder „hält nicht“ terminiert.

Tafel 1 Ein Beispiel, wie die Liste aller Programme aussehen könnte, wenn die verwendete Programmiersprache nur die Zeichen `a` bis `z` verwendet.

1.	`a`
2.	`b`
⋮	
26.	`z`
27.	`aa`
28.	`ab`
⋮	
41320.	`bice`
⋮	

Beweis des Theorems. Als Vorbemerkung halten wir fest, dass wir die Gesamtheit aller Programme in einer unendlichen Liste organisieren können (Tafel 1) und darüber hinaus ein Programm F schreiben können, das bei Eingabe einer natürlichen Zahl n das n-te Programm dieser Liste ausgibt.

Angenommen, es gibt ein Halteorakel H.[3] Dann können wir ein Programm R programmieren, das wie folgt abläuft:

1. Lese eine Zahl n als Eingabe ein.
2. Lasse das Halteorakel ablaufen, um herauszufinden, ob das Programm $F(n)$ (also das n-te Programm auf der Liste) bei Eingabe von n hält oder nicht.
3. Falls es hält: Gehe in eine Endlosschleife.
4. Falls es nicht hält: Halte.

[1] Turing leistete während des Zweiten Weltkriegs entscheidende Beiträge zur Kryptoanalyse der deutschen Verschlüsselungsmaschine Enigma und ermöglichte so die Entschlüsselung deutscher Funksprüche. Wegen seiner Homosexualität wurde er im März 1952 zur chemischen Kastration verurteilt. Er erkrankte an Depression und beging Suizid.

[2] Tatsächlich gibt es sogar für *jedes* Programm P ein auf P maßgeschneidertes Orakelprogramm, welches korrekt die Meldung „P hält“ oder „P hält nicht“ ausgibt. Denn es ist ja klar, dass P entweder hält oder nicht hält. Im ersten Fall ist das triviale Programm, das sich direkt nach seinem Aufruf mit der Meldung „P hält“ sofort wieder beendet, das gesuchte Halteorakel. Im zweiten Fall ist es das ebenso triviale Programm, das die Meldung „P hält nicht“ ausgibt. Die Situation wirkt vielleicht etwas wundersam: Eines dieser beiden (trivialen!) Programme ist das gesuchte Halteorakel, wir können nur im Allgemeinen nicht sagen, welches es ist.

[3] Eigentlich müssen wir den Rest des Beweises in den Konjunktiv setzen, da wir ab dieser Stelle eine Annahme treffen (mit dem Ziel, einen Widerspruch zu erkennen, um die Falschheit der Annahme nachzuweisen). Das ist aber umständlich.

Das Programm R zeigt bei Eingabe von n also genau das entgegengesetzte Halteverhalten von $F(n)$.

Wie jedes Programm ist auch R in der Liste aller Programme verzeichnet, etwa an m-ter Stelle: Es gilt also $F(m) = R$. Wir können uns nun fragen, ob R bei Eingabe von m hält oder nicht. Verfolgen wir den Programmfluss, sehen wir aber, dass beide Fälle zu einem Widerspruch führen. □

Aufgabe 1. *Verstanden?*

a) Vollziehe den Beweis des Theorems nach. Könntest du jemand anderem erklären, welche Widersprüche die Existenz eines Halteorakels nach sich ziehen würde?

b) Wieso war es für den Beweis wichtig, dass es das Programm F gibt? Wieso also war es wichtig, dass wir bei Eingabe einer Zahl n das n-te Programm auf der Liste aller Programme berechnen können?

Was ist eigentlich ein *Programm*? Auf diese Frage sollten wir eine präzise Antwort haben, wenn wir rigoros Mathematik betreiben möchten. Zur Vorstellung ist es – insbesondere, wenn man Programmiererfahrung hat – hilfreich, sich Programme einer realen Programmiersprache (Haskell, Perl, Python, ...) vorzustellen, welche aber auf *idealisierten Computern* ablaufen: Computer, die niemals kaputt gehen und über beliebig viel Arbeitsspeicher verfügen.

Um diese Vorstellung aber zu präzisieren, hat Turing das nach ihm benannte Konzept der *Turingmaschine* entworfen. Turingmaschinen sind abstrakte Rechenmaschinen, die sich, weil sie von den vielen Details realer Computer abtrahieren, für mathematische Untersuchungen besonders gut eignen. Mit *Programm* meinen wir also eigentlich *Turingmaschine*.

Eine Turingmaschine arbeitet auf einem in beide Richtungen unendlich langem Band, dessen Zellen die Werte 0 und 1 fassen können (Abbildung 1). In jedem Rechenschritt kann die Turingmaschine den gespeicherten Wert auf dem Band an der aktuellen Position lesen, einen neuen Wert schreiben und das Band um eine Zelle nach links oder rechts verschieben. Dabei befolgt sie ein für sie eindeutiges, endliches Regelwerk.

Aufgabe 2. *Das Halteproblem für halbreale Computerprogramme*

In dieser Aufgabe soll es um Programme gehen, die auf Computern laufen, welche zwar nie kaputt gehen und auch nicht aufgrund äußerer Einflüsse in ihrem Verhalten gestört werden (etwa durch kosmische Strahlung), aber trotzdem nur über endlichen Speicher verfügen. Außerdem sollen sie von der Außenwelt in dem Sinn isoliert seien, dass sie etwa keine Verbindung zum Internet besitzen. Kurz und präzise, aber vielleicht weniger anschaulich: Es soll um Turingmaschinen mit endlichem Band gehen.

Erkläre, wieso das Halteproblem für solche Programme (zumindest in der Theorie) trivial lösbar ist.

Tipp: In wie vielen verschiedenen Zuständen kann sich ein derartiger Computer befinden?

Aufgabe 3. *Der Satz von Rice*

Der *Satz von Rice* ist eine Verschärfung von Turings Unmöglichkeitstheorem. Er besagt: Für keine nichttriviale extensionale Programmeigenschaft E gibt es ein Orakel, das bei Eingabe eines Programms P mit einer korrekten Ausgabe von „P hat Eigenschaft E“ oder „P hat Eigenschaft E nicht“ hält.

Dabei meint *nichttrivial*, dass manche Programme die Eigenschaft E haben und andere nicht. Etwa ist die Eigenschaft „P hält oder hält nicht“ ein Beispiel für eine Eigenschaft, welche *nicht* nichttrivial ist.

Extensionalität verlangt, dass sich die Eigenschaft E nur auf das von außen sichtbare Verhalten des Programms, nicht aber seine innere Struktur (seinen Quellcode) bezieht. Etwa sind die Eigenschaften „P hält“, „P gibt als Ergebnis die Zahl 37 aus“ und „P gibt eine gerade Zahl als Ergebnis aus“ extensionale Eigenschaften. Keine extensionalen Eigenschaften sind „P besteht aus weniger als 20 Zeilen Code“ und „P tätigt mindestens 100 Rechenschritte“.

a) Überlege, wieso es ganz einfach ist, ein Orakel zu programmieren, dass die nichtextensionale Eigenschaft „P besteht aus weniger als 20 Zeilen Code“ prüft.

b) Beweise den Satz von Rice. Du kannst dich dabei am Beweis der Unentscheidbarkeit des Halteproblems orientieren.

Aufgabe 4. *Die Church–Turing-These*

Informiere dich auf Wikipedia, was es mit der Church–Turing-These auf sich hat. Hast du eine Meinung dazu?

4 Die unberechenbare Fleißiger-Biber-Funktion

5 Chaitins Haltewahrscheinlichkeit

6 Formale Systeme und Modelle

7 Gödels Vollständigkeitssatz und sein Unvollständigkeitssatz

Abbildung 1 Eine künstlerische Darstellung einer Turing-Maschine. Quelle: `http://www.worldofcomputing.net/wp-content/uploads/2013/01/turingMachine.gif`