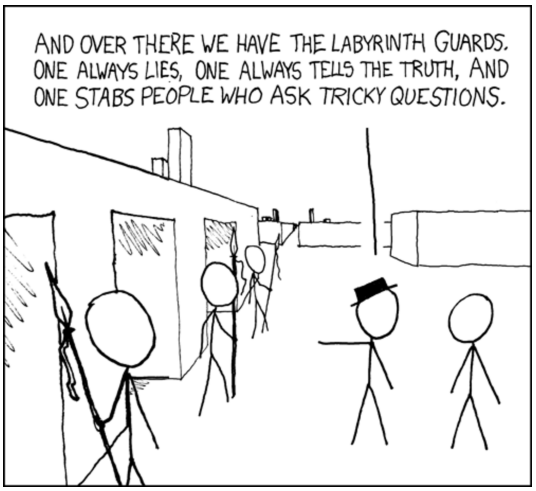
Matheschülerzirkel
Universität Augsburg
Schuljahr 2014/2015
Klassen 10/11/12

Gödels Unvollständigkeitssatz

Zirkelzettel vom 7. und 21. November 2014 und 5. Dezember 2014



http://xkcd.com/246

1 Berrys Paradoxon

Was ist die größte natürliche Zahl, die man im Deutschen in unter 1000 Buchstaben beschreiben kann? Die Beschreibung soll für eine bestimmte Zielgruppe, zum Beispiel Mathematikerinnen und Mathematiker, klar verständlich sein und eine bestimmte Zahl eindeutig festlegen.

1. Ein guter Anfang: „999…999“ (1000 Neuner)
2. Viel besser: „10 hoch 999…999“ (ein paar Neuner weniger)
3. Noch besser: „10 hoch (10 hoch (10 hoch …))“

Aufgabe 1. *Wettbewerb zu großen Zahlen*

Wenn du hinreichend nerdige Freunde hast, kannst du einen Wettbewerb zu großen Zahlen abhalten: Jeder soll auf einem Kärtchen vorgegebener Größe eine Zahl beschreiben. Die Beschreibung muss für eine zuvor ausgemachte Zielgruppe verständlich sein und genau eine natürliche Zahl festlegen. Gewinner ist, wer die größte Zahl beschrieben hat.

a) Was passiert, wenn ein Teilnehmer folgenden Beitrag abgibt? „Die größte Zahl, die meine Konkurrenten beschrieben haben, plus Eins“

b) Was passiert, wenn zwei Teilnehmer diesen Vorschlag machen?

c) Vielleicht verbietest du expliziten Selbstbezug („meine Konkurrenten“). Was aber passiert, wenn dann jemand „Die zweitgrößte Zahl, die eingereicht wurde, plus Eins“ einreicht?

d) Als Ausrichter des Wettbewerbs könntest du eine große Geldsumme aussetzen. Der Gewinner (oder, bei Gleichstand, die Gewinner) erhält eine Million Euro, *geteilt durch* die Zahl, mit der er gewonnen hat. Gibt es eine Strategie, die die Teilnehmer im Vorfeld absprechen könnten, um möglichst viel Geld zu kassieren? Ist es wahrscheinlich, dass sich alle Teilnehmer an eine solche Abmachung halten?

Bemerkung: Der Mengentheoretiker Joel David Hamkins hat sich über dieses Thema auch Gedanken gemacht. `http://jdh.hamkins.org/largest-number-contest/`

Auf den ersten Blick erscheint die eingangs gestellte Frage eine eindeutige Antwort zu haben: Es gibt nur endlich viele Möglichkeiten, die Buchstaben des deutschen Alphabets (und die Interpunktszeichen) zu grammatikalisch korrekten deutschen Phrasen der Länge ≤ 1000 zusammenzusetzen. Viele dieser Phrasen beschreiben keine Zahlen („Ich mag Schokolade.“), beschreiben keine natürlichen Zahlen („Das Verhältnis aus Durchmesser und Radius eines jeden Kreises“) oder beschreiben keine eindeutig bestimmte natürliche Zahl („alle Zahlen größer als 5“). Die restlichen Phrasen aber beschreiben doch jeweils genau eine natürliche Zahl. Sollte es dann nicht unter diesen endlich vielen natürlichen Zahlen eine größte geben?

Berrys Paradoxon zeigt uns, dass diese Frage in Wahrheit nicht korrekt gestellt ist. Die Frage hat keine Antwort, denn „die größte natürliche Zahl, die im Deutschen in unter 1000 Buchstaben beschrieben werden kann, plus Eins“ ist einerseits selbst in unter 1000 Buchstaben beschreibbar (ich habe es in 104 Zeichen geschafft, Leerzeichen mitgezählt), andererseits größer als alle in unter 1000 Buchstaben beschreibbaren Zahlen.

Aufgabe 2. *Uninteressante natürliche Zahlen*

Beweise, dass es keine uninteressanten natürlichen Zahlen gibt.

Tipp: Gehe nach folgendem Muster vor. Angenommen, es gäbe uninteressante natürliche Zahlen (mindestens eine). Dann wäre unter all diesen uninteressanten natürlichen Zahlen eine die kleinste. Aber ..., Widerspruch.

Hinweis: Offensichtlich ist „interessant“ kein rigoroses mathematisches Adjektiv und diese Aufgabe daher nicht besonders tiefsinnig.

2 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 2.1. Die *Goldbachsche Vermutung* besagt, dass jede gerade Zahl größer als 3 die Summe zweier Primzahlen ist. Für jede konkrete gerade Zahl n größer als 3 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 2.2. 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 2.3. *Es gibt kein Programm, das bei Eingabe eines beliebigen 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 (Beispiele 2.1 und 2.2). Vielmehr behauptet Turing, dass ein Halteorakel rein prinzipiell nicht existieren kann. Auch Außerirdische mit überlegener Technologie oder transzendente Wesen können kein Halteorakel programmieren.

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.

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.

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.

[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.

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`
	⋮

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.

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. □

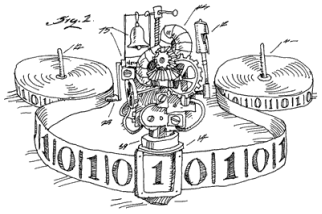
Aufgabe 3. *Verstanden?*

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?

[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.

2.1 Turingmaschinen



Eine künstlerische Darstellung einer Turing-Maschine.[4]

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 abstrahieren, 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. 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.

2.2 Weitere Übungsaufgaben

Aufgabe 4. *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 5. *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.

[4] `http://www.worldofcomputing.net/wp-content/uploads/2013/01/turingMachine.gif`

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 6. *Die Church–Turing-These*

Informiere dich auf Wikipedia, was es mit der Church–Turing-These auf sich hat; manche Leute halten sie für eine der größten ungeklärten Fragen der Physik und Informatik. Hast du eine Meinung dazu?

3 Die unberechenbare Fleißiger-Biber-Funktion

Die *Fleißiger-Biber-Funktion* ist eine Funktion $\mathrm{BB} : \mathbb{N} \to \mathbb{N}$, die eng mit dem Halteproblem verknüpft ist. Per Definition ist $\mathrm{BB}(n)$ die größte Anzahl Rechenschritte, die irgendein haltendes Programm mit Quelltextlänge $\leq n$ tätigt.

Anders formuliert: Unter den Programmen mit Länge $\leq n$ gibt es welche, die halten, und welche, die nicht halten. Unter denen, die halten, gibt es ein Programm, dass unter all diesen Programmen am meisten Rechenschritte ausführt, bevor es schlussendlich hält. Die Anzahl dieser Rechenschritte ist $\mathrm{BB}(n)$.

Die exakten Werte der Fleißiger-Biber-Funktion hängen von der Wahl der Programmiersprache ab, denn in verschiedenen Sprachen drückt man sich beim Programmieren unterschiedlich aus. In der Literatur verwendet man daher die abstrakten Turingmaschinen als Referenzpunkt.

Die Fleißiger-Biber-Funktion wächst rasant an, viel schneller als exponentiell; nur sehr wenige Funktionswerte sind bekannt (Tabelle 2). Tatsächlich wächst sie (asymptotisch) *schneller als jede durch Programme berechenbare Funktion*. Wie auch bei Turings Unmöglichkeitsresultat liegt der Grund dafür nicht in menschlichen technologischen Beschränkungen, sondern ist prinzipieller Natur.

Theorem 3.1. *Die Fleißiger-Biber-Funktion ist nicht berechenbar, d. h. es gibt kein Programm P, das bei Eingabe einer natürlichen Zahl n die Zahl* $\mathrm{BB}(n)$ *zurückgibt.*

Aufgabe 7. *Unberechenbarkeit der Fleißiger-Biber-Funktion*

Beweise das Theorem, indem du folgende Argumentation ausformulierst. Schau dir erst danach den weiter unten ausgeführten Beweis an.

Wenn es ein Programm gäbe, dass die Fleißiger-Biber-Funktion berechnen könnte, dann könnte man daraus ein Halteorakel konstruieren (wie geht das?). Dass es ein solches nicht gibt, wissen wir schon.

Beweis des Theorems. Angenommen, es gibt ein Programm, dass die Fleißiger-Biber-Funktion berechnet. Dann können wir mit dessen Hilfe wie folgt ein Halteorakel programmieren:

1. Lese ein Programm P als Eingabe ein.
2. Bestimme mit dem Hilfsprogramm die Zahl $\mathrm{BB}(n)$, wobei n die Länge von P ist.
3. Lasse P für $\mathrm{BB}(n)$ Rechenschritte lang ablaufen.
4. Wenn P bis dahin gehalten hat: Gib „P hält“ aus und beende. Sonst gib „P hält nicht“ aus und beende.

Dieses Halteorakel arbeitet wirklich korrekt: Denn wenn P nach $\mathrm{BB}(n)$ Rechenschritten nicht gehalten hat, wird es niemals halten – nach Definition ist ja $\mathrm{BB}(n)$ die Maximalzahl Rechenschritte, die ein haltendes Programm der Länge $\leq n$ ausführen kann, bevor es hält. □

4 Chaitins Haltewahrscheinlichkeit

Bevor wir uns der präzisen Formulierung und des Beweises des Gödelschen Unvollständigkeitssatzes widmen, möchten wir noch einen Abstecher zu *Chaitins Haltewahrscheinlichkeit* Ω machen, einer bestimmte reellen mit faszinierenden Eigenschaften. In

Tafel 2 Die bekannten Werte der Fleißiger-Biber-Funktion, entnommen aus ihrem Wikipedia-Eintrag. Mit *Programm* ist hier *Turingmaschine* und mit *Länge* die Anzahl der Zustände gemeint.

n	Anzahl Programme der Länge $\leq n$	$\mathrm{BB}(n)$
1	64	1 (1962)
2	20736	4 (1962)
3	16777216	6 (1965)
4	$25{,}6 \cdot 10^9$	13 (1972)
5	$\approx 63{,}4 \cdot 10^{12}$	≥ 4098 (1989)
6	$\approx 232 \cdot 10^{15}$	$> 3{,}514 \cdot 10^{18267}$ (2010)

einem Artikel schrieben der amerikanische Physiker und Informatiker Charles Bennet (* 1943) und der berühmte Wissenschaftsjournalist Martin Gardner (* 1914, † 2010) folgendes über diese Zahl:[5]

> Die Konstante Ω verkörpert eine enorme Menge an Wissen auf sehr kleinem Raum. Die ersten paar Tausend Ziffern, die problemlos auf einem kleinen Stück Papier Platz finden könnten, enthalten die Antworten auf mehr mathematische Fragen, als man im ganzen Universum aufschreiben könnte.
>
> Im Laufe der Menschheitsgeschichte strebten Mystiker und Philosophen stets nach einem kompakten Schlüssel zu universeller Weisheit, einer endlichen Formel oder einem Text, der, wenn bekannt und verstanden, Antworten auf alle Fragen liefern würde; man denke nur an die Versuche, der Bibel, dem Koran oder dem I Ging Weissagungen zu entlocken [...].
>
> Solche Quellen universeller Weisheit sind herkömmlicherweise vor beiläufigem Zugriff geschützt: indem sie schwer zu finden, wenn gefunden schwierig zu verstehen und gefährlich zu benutzen sind, dazu neigend, mehr und tiefere Fragen zu beantworten als sich der Suchende wünschte. Das esoterische Buch ist, wie Gott, einfach und dennoch unbeschreibbar. Es ist allwissend, und verändert alle, die es kennen.
>
> Ω ist in vielerlei Hinsicht eine kabbalistische Zahl. Dem menschlichen Verstand ist sie bekannt, aber unkennbar. Um sie im Detail zu erfahren, müsste man ihre unberechenbare Ziffernfolge als Glaubensgrundsatz einfach hinnehmen, genau wie die Worte eines heiligen Texts.

Angesichts dieser Auszeichnung überrascht es vielleicht, dass die Konstante Ω durch eine kurze Formel definiert werden kann:

$$\Omega := \sum_p 2^{-|p|}.$$

Das ist eine Kurzschreibweise, die ausgeschrieben für die unendliche Summe

$$\Omega := 2^{-|p_1|} + 2^{-|p_2|} + 2^{-|p_3|} + \cdots$$

[5] Ω embodies an enormous amount of wisdom in a very small space ... inasmuch as its first few thousand digits, which could be written on a small piece of paper, contain the answers to more mathematical questions than could be written down in the entire universe.

Throughout history mystics and philosophers have sought a compact key to universal wisdom, a finite formula or text which, when known and understood, would provide the answer to every question. The use of the Bible, the Koran and the I Ching for divination and the tradition of the secret books of Hermes Trismegistus, and the medieval Jewish Cabala exemplify this belief or hope.

Such sources of universal wisdom are traditionally protected from casual use by being hard to find, hard to understand when found, and dangerous to use, tending to answer more questions and deeper ones than the searcher wishes to ask. The esoteric book is, like God, simple yet undescribable. It is omniscient, and transforms all who know it.

Ω is in many senses a cabalistic number. It can be known of, but not known, through human reason. To know it in detail, one would have to accept its uncomputable digit sequence on faith, like words of a sacred text. (C. Bennett und M. Gardner, „The random number Ω bids fair to hold the mysteries of the universe“, Scientific American (1979), Ausgabe 241, Seiten 20—34.)

steht. Dabei ist p_1, p_2, p_3, ... die unendliche Liste aller *anhaltenden Programme*, und für ein Programm p steht „$|p|$“ für seine Länge. Die Konstante Ω errechnet sich also dadurch, indem man gedanklich alle anhaltenden Programme durchgeht und jeweils $2^{-\text{Programmlänge}}$ addiert. Der Zahlenwert von Ω hängt von der verwendeten Programmiersprache ab; wenn wir eine Definition wünschen, die nicht von einer solchen willkürlichen Wahl abhängt, können wir Turingmaschinen verwenden.

Man kann zeigen, dass die unendliche Addition konvergiert, dass durch die Formel also wirklich genau eine reelle Zahl festgelegt wird, und dass diese zwischen 0 und 1 liegt.[6] Die Bezeichnung als *Haltewahrscheinlichkeit* erklärt sich dadurch, als dass Ω gerade die Wahrscheinlichkeit ist, dass ein willkürzlich gezogenes Programm hält.

4.1 Unberechenbarkeit und Unkennbarkeit

Inwiefern sind in Ω Antworten auf unzählige mathematische Fragen enthalten? Darüber gibt die folgende Proposition Auskunft. Um ihre volle Tragweite würdigen zu können, erinnern wir uns daran, dass wir viele bedeutende Forschungsprobleme als Fragen der Form „Hält folgendes Programm?“ ausdrücken können (Beispiele 2.1 und 2.2).

Proposition 4.1. *Kennen wir die ersten N Nachkommaziffern von Ω im Binärsystem, so können wir das Halteproblem für alle Programme der Länge $\leq N$ lösen, also für jedes solche Programm in endlicher Zeit entscheiden, ob es hält oder nicht.*

Beweis. Wir machen eine Liste aller Programme der Länge $\leq N$. Das wird eine sehr lange, aber endliche Liste. Dann lassen wir all diese Programme in verzahnter Art und Weise laufen: Wir erlauben jedem Programm, einige Rechenschritte zu tätigen; danach ist das nächste Programm an der Reihe. Nach dem letzten Programm auf der Liste kommt wieder das erste an der Reihe.

Im Laufe der Zeit werden immer mehr Programme anhalten. Immer, wenn das passiert, notieren wir uns die Zahl $2^{-\text{Länge dieses Programms}}$. Die Summe dieser Zahlen wird immer weiter anwachsen und sich von unten der Konstante Ω annähern. Sobald diese Summe größer oder gleich als die als bekannt vorausgesetzte untere Schranke für Ω (gegeben durch die ersten N Binärziffern) ist, brechen wir alle noch laufenden Programme ab und ziehen folgende Bilanz:

Alle Programme, die bis zu diesem Zeitpunkt noch nicht angehalten haben, werden niemals anhalten. Denn diese würden in der Summe die ersten N Nachkommaziffern beeinflussen. So wissen wir also nun von allen Programmen der Länge $\leq N$, ob sie halten oder nicht. □

Die Konstante Ω ist also ein Beispiel für eine *unberechenbare Zahl* – es kann aus prinzipiellen Gründen kein Programm geben, das ihre Nachkommaziffern berechnet. Das

[6] Damit das stimmt, muss man zwei technische Einschränkungen treffen. Zum einen muss man Programme im Binärsystem notieren, Programme also durch 0/1-Folgen beschreiben. (Wenn man partout ein Alphabet aus mehr zwei Zeichen verwenden möchte, muss man in der Definition von Ω etwa $26^{-|p|}$ statt $2^{-|p|}$ schreiben.) Zum anderen muss die verwendete Programmiersprache *präfixfrei* sein. Das bedeutet, dass man an ein syntaktisch korrektes Programm keine weiteren Zeichen anhängen kann, ohne Syntaxfehler zu verursachen. Beide Einschränkungen spielen im Folgenden aber kaum eine Rolle.

unterscheidet sie von anderen wichtigen Konstanten wie π und e, für die viele Programme zu ihrer Berechnung bekannt sind.

Es kommt aber noch schlimmer: Die Ziffern von Ω sind nicht nur durch Programme unberechenbar, sie entziehen sich in einem gewissen Sinn auch allen anderen Arten der Erkenntnis; das ist, was Bennet und Gardner als *unkennbar* bezeichnet haben. Die genaue Aussage enthält das folgende Theorem. Leider übersteigt sein Beweis etwas unsere technischen Mittel.

Theorem 4.2. *Zu jedem formalen System gibt es eine Schranke N, sodass für alle $n \geq N$ die Aussagen „Ω hat als n-te Nachkommaziffer eine Null“ und „Ω hat als n-te Nachkommaziffer eine Eins“ in dem System weder bewiesen noch widerlegt werden können.*

Wenn wir, auf welche Art und Weise auch immer, von den Nachkommaziffern von Ω erfahren würden, könnten wir diese Erkenntnis also nicht beweisen.

4.2 Informationsgehalt und Zufall

Die Konstante Ω hat eine weitere wundersame Eigenschaft: Ihre Ziffern sind *Zufallsdaten*. Das ist natürlich nicht wörtlich zu verstehen – Ω ist eine präzise eindeutig definierte Zahl – aber in einem gewissen Sinn stimmt es doch. Um das einzusehen, müssen wir uns klarmachen, was eine *zufällige Ziffernfolge* überhaupt ist. Das ist nicht ganz einfach! Eine erste Idee, Zufall zu charakterisieren, ist folgende:

Vorläufige Definition 4.1. Eine Ziffernfolge heißt genau dann *zufällig*, wenn in ihr alle Ziffern im Mittel gleich oft vorkommen.

Nach dieser Definition würde aber auch die völlig regelmäßige und anschaulich überhaupt nicht zufällige Ziffernfolge 01234567890123456789 ... als zufällig gelten. Etwas besser ist folgende Definition.

Vorläufige Definition 4.2. Eine Ziffernfolge heißt genau dann *zufällig*, wenn in ihre alle Ziffern, alle Ziffernpaare, alle Zifferntripel und so weiter im Mittel gleich oft vorkommen.

Damit gilt 01234567890123456789 ... nicht mehr als zufällig, denn zum Beispiel kommt das Ziffernpaar 00 überhaupt nicht vor. Allerdings ordnet auch diese Definition Zahlenfolgen falsch ein, denn die *Champernowne-Folge* 0123456789101112131415161718192021 ... ist offensichtlich nicht zufällig, sie erfüllt aber das statistische Kriterium der Definition. Andererseits gibt es auch Folgen, die anschaulich zufällig sind, das Kriterium aber nicht erfüllen. Notieren wir etwa die Augenzahlen eines fairen Würfels in einer Folge, so kommen in dieser Folge gewiss nicht alle Ziffern gleich oft vor – die Ziffern 0, 7, 8 und 9 kommen überhaupt nicht vor.

Um eine wirklich nützliche Definition von Zufälligkeit zu erhalten, sollten wir daher Abstand von statistischen Anforderungen nehmen und eher einen *informationstheoretischen Standpunkt* einnehmen: Lässt sich die regelmäßige Struktur einer Ziffernfolge in wenigen Worten beschreiben, so schätzen wir sie als eher unzufällig ein; ist dagegen die einzige

Möglichkeit, die Ziffernfolge zu beschreiben, die, schlichtweg die Ziffern nacheinander aufzuzählen, so erachten wir sie als zufällig. Im ersten Fall sprechen wir von einem niedrigen *Informationsgehalt*, im zweiten von einem hohen.

Definition 4.3. Der *Informationsgehalt* einer Ziffernfolge ist die Länge des kürzesten Programms, das diese Ziffernfolge berechnet und ausgibt.

Wir können hier auch an *Beschreibungen durch deutsche Texte* statt an Programme denken, solange wir nur klare und verständliche Beschreibungen akzeptieren – sonst wird, wie bei Berrys Paradoxon, die Definition widersprüchlich.

Definition 4.4. Eine Ziffernfolge heißt genau dann *zufällig*, wenn ihr Informationsgehalt nicht deutlich kleiner als ihre Länge ist.

Diese Definition lässt sich nicht austricksen. Zum Beispiel erscheint die aus 325 Ziffern bestehende Folge

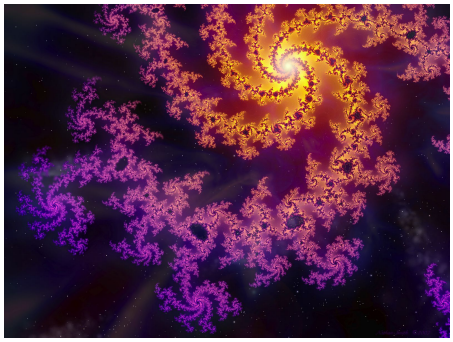
$$
\begin{aligned}
&89793238462643383279502884197169399375105820974944592307816406286\\
&20899862803482534211706798214808651328230664709384460955058223172\\
&53594081284811174502841027019385211055596446229489549303819644288\\
&10975665933446128475648233786783165271201909145648566923460348610\\
&45432664821339360726024914127372458700660631558817488152092096282
\end{aligned}
$$

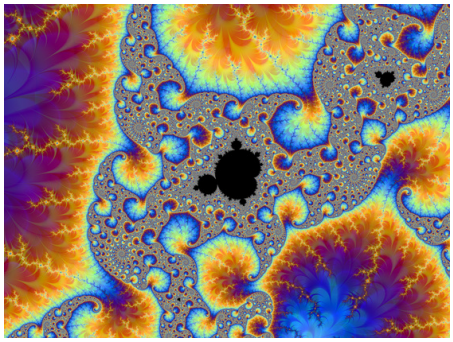
auf den ersten Blick völlig unregelmäßig. Tatsächlich aber handelt es sich einfach um die Nachkommaziffern von π ab der elften Stelle, ihr Informationsgehalt ist also viel geringer als 325. Ähnlich verhält es sich mit den Grafiken in Abbildung 1: Wegen ihrer filigranen Struktur könnte man denken, dass eine präzise Beschreibung dieser Grafiken viel Platz in Anspruch nimmt. Tatsächlich sind es aber mathematische Fraktale, die schon durch eine kurze Formel und die Angabe von Koordinaten eindeutig bestimmt sind.

Theorem 4.5. *Die Ziffern von Ω sind zufällig.*

Abbildung 1 Wer interaktiv Fraktale dieser Art untersuchen möchte, kann sich das freie Programm XaoS herunterladen.

http://www.dvice.com/sites/dvice/files/Fractal3.jpg http://upload.wikimedia.org/wikipedia/commons/8/8b/Fractal_KRkr_City1_5600.jpg





Beweisskizze. Wir haben schon gesehen, dass Ω unberechenbar ist. Ein Programm, dass die ersten N Nachkommaziffern von Ω ausgibt, kann also die Ziffern nicht auf eine clevere Art und Weise berechnen, sondern muss sie schon im Quelltext gespeichert haben. Also muss der Quelltext in etwa selbst schon die Länge N haben. □

5 Formale Systeme und Modelle

Ein *formales System* gibt Axiome und logische Schlussregeln vor.

Beispiel 5.1. Das formale System *Peano-Arithmetik* (PA) hat als Axiome:

- Es gibt eine Zahl $\overline{0}$.
- Jede Zahl n besitzt einen und nur einen Nachfolger $S(n)$.
- Sind n und m Zahlen mit gleichem Nachfolger, so sind n und m selbst schon gleich.
- Die Zahl $\overline{0}$ ist kein Nachfolger einer Zahl.
- $n + \overline{0} := n$.
- $n + S(m) := S(n + m)$.
- $\overline{1} := S(\overline{0})$, $\overline{2} := S(S(\overline{0}))$, $\overline{3} := S(S(S(\overline{0})))$, $\overline{4} := S(S(S(S(\overline{0}))))$, ...

Dazu kommen noch einige weitere, insbesondere das wichtige *Induktionsaxiom.*

In dem, was kommt, dürfen wir keinesfalls die *Objektebene* mit der *Metaebene* verwechseln. Seit Kindesbeinen an kennen wir die gewohnten natürlichen Zahlen; diese schreiben wir ganz normal, ohne Überstriche: $0, 1, 2, \ldots$ Diese natürlichen Zahlen gehören zur Metaebene.

Die Axiome der Peano-Arithmetik sind ein Versuch, unsere Intuition über die natürlichen Zahlen einzufangen und ihren wesentlichen Kern herauszuarbeiten. Dabei haben die in den Axiomen verwendeten Symbole „$\overline{0}$“, „$\overline{1}$“ usw. der Objektebene zunächst *keine inhaltliche Bedeutung.* Insbesondere beziehen sie sich nicht auf die gewohnten natürlichen Zahlen. Um diesen Unterschied nicht zu vergessen, notieren wir diese Symbole daher mit einem Überstrich.

In einem gegebenen formalen System kann man *formale Beweise* führen. Das sind Beweise, die nur die durch das System vorgegebenen Axiome und Schlussregeln verwenden.

Beispiel 5.2. In PA geht ein Beweis der Behauptung $\overline{2} + \overline{2} = \overline{4}$ wie folgt:

$$\overline{2} + \overline{2} = \overline{2} + S(S(\overline{0})) = S(\overline{2} + S(\overline{0})) = S(S(\overline{2} + \overline{0})) = S(S(\overline{2})) = S(S(S(S(\overline{0})))) = \overline{4}.$$

Aufgabe 8. *Formale Beweise in Peano-Arithmetik*

a) Vollziehe den formalen Beweis, dass $\overline{2} + \overline{2} = \overline{4}$ ist, im Detail nach. Welches Axiom wird in welchem Schritt verwendet?

b) Die Axiome für die Multiplikation lauten

$$n \cdot \overline{0} := \overline{0},$$
$$n \cdot S(m) := n \cdot m + n.$$

Führe mit diesen Axiomen einen formalen Beweis der Behauptung $\overline{2} \cdot \overline{2} = \overline{4}$.

Warnung 5.3. Für uns sind die gewohnten natürlichen Zahlen *nicht* durch die Peano-Axiome definiert. Wir setzen stattdessen voraus, dass wir aus der Schule und der Erfahrung einen intuitiven Zugang zu natürlichen Zahlen haben. Wenn man damit nicht einverstanden ist – etwa, da man das unrigoros erachtet – muss man die Diskussion in diesem Abschnitt ihrerseits in einem bestimmten formalen System interpretieren. Irgendwo aber muss man beginnen! Man kommt nicht umhin, *Metalogik* vorauszusetzen.

Ein *Modell* eines formalen Systems ist eine Struktur, in der die Axiome und Schlussregeln des Systems tatsächlich erfüllt sind. Zum Beispiel bilden die gewöhnlichen natürlichen Zahlen ein Modell von Peano-Arithmetik, wenn man $\overline{0}$ als 0, $\overline{1}$ als 1, und so weiter, interpretiert.

Kein Modell von Peano-Arithmetik bilden die ganzen Zahlen $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$. Denn dort ist die Zahl 0 durchaus ein Nachfolger einer anderen Zahl (nämlich der Zahl -1).

Aufgabe 9. *Ein Nichtstandardmodell von Peano-Arithmetik*

Man kann beweisen, dass Peano-Arithmetik neben den gewöhnlichen natürlichen Zahlen noch viele weitere Modelle besitzt: Strukturen, die wie die natürlichen Zahlen auch die Peano-Axiome erfüllen, aber nicht äquivalent zu den natürlichen Zahlen sind. Eines dieser Modelle umfasst die gewöhnlichen natürlichen Zahlen, enthält zusätzlich aber auch eine spezielle Zahl λ, über die folgendes bekannt ist: λ ist größer als 10, größer als 100, größer als 1000, und allgemein größer als jede gewöhnliche natürliche Zahl.

Versuche, dir ein möglichst gutes Bild dieses wundersamen Modells zu machen! Die Zahl λ steht *rechts* vom gewöhnlichen Zahlenstrahl; aber zusätzlich zu λ gibt es auch $S(\lambda)$, $S(S(\lambda))$, und so weiter. Außerdem muss es auch einen Vorgänger von λ geben. Kannst du auf diese Weise weiter denken?

6 Gödels Vollständigkeitssatz und sein Unvollständigkeitssatz

6.1 Der Unvollständigkeitssatz

Grob formuliert besagt der Unvollständigkeitssatz: Es gibt Aussagen – so genannte *Gödel-Sätze* – die wahr, aber unbeweisbar sind. Das hört sich fantastisch, aber auch etwas seltsam an. Denn wie sollten wir denn von der Wahrheit dieser Gödel-Sätze überzeugt sein, wenn nicht durch einen Beweis?

Die Verwirrung löst sich bei präzisen Formulierungen des Unvollständigkeitssatzes aus. Eine solche lautet wie folgt.

Theorem 6.1 (Gödels Unvollständigkeitssatz)**.** *In jedem formalen System, das*

a) effektiv axiomatisierbar ist,

b) Peano-Arithmetik umfasst und

c) korrekt ist,

gibt es Aussagen, die im intendierten Modell wahr sind, aber keinen Beweis innerhalb des Systems zulassen.

Dabei bedeutet *effektiv axiomatisierbar*, dass die Axiome und Schlussregeln des Systems von so einfacher Form sind, dass ein Computerprogramm (eine Turingmaschine) mit ihnen umgehen kann. Das bedeutet insbesondere, dass man ein Programm schreiben kann, das nacheinander alle möglichen Versuche formaler Beweise durchgeht, jeweils prüft, ob sie korrekte Beweise innerhalb des Systems darstellen und wenn ja die durch sie bewiesene Aussage ausgeben. So erhält man also ein Programm, das nacheinander alle beweisbaren (und keine unbeweisbaren) Aussagen ausgibt.

Die Forderung, dass ein formalen System effektiv axiomatisierbar sein soll, ist vernünftig. Denn ein formales System soll ja aus wenigen und einfachen Grundbausteinen bestehen. Ein System, indem man nur mit Mühe entscheiden könnte, ob ein vorliegender Beweisversuch überhaupt korrekt ist, hat keinen großen Wert.

Die zweite Forderung, *Peano-Arithmetik zu umfassen*, bedeutet, dass das System insbesondere die Axiome und Schlussregeln von Peano-Arithmetik enthält und daher über Eigenschaften von Zahlen sprechen kann. Das erwartet man von jedem formalen System, das eine Grundlage für unser gesamtes mathematisches Wissen sein möchte. Es gibt aber auch interessante Systeme, die mit Peano-Arithmetik und Zahlen nichts am Hut haben; auf solche bezieht sich Gödels Unvollständigkeitssatz nicht.

Schließlich ist mit *Korrektheit* eines formalen Systems gemeint, dass jede Aussage, die im System formal beweisbar ist, auch tatsächlich im angedachten (intendierten) Modell wahr ist. Das ist etwa bei Peano-Arithmetik der Fall. Indem man unsinnige Axiome hinzufügt, kann man aber auch leicht formale Systeme konstruieren, die nicht korrekt sind. Solche Axiome könnten zum Beispiel sein: „Die Null ist kein Nachfolger einer Zahl, und gleichzeitig ist die Null schon ein Nachfolger einer Zahl.“ Oder auch: „Das Doppelte einer jeden Zahl ist Null.“ Beide Aussagen sind in den gewöhnlichen natürlichen Zahlen nicht erfüllt.

Mit unseren Vorarbeiten zur Berechenbarkeitstheorie sind wir nun gerüstet, um einen Beweis von Gödels Unvollständigkeitssatz zu führen. Dabei sollten wir besonderes Augenmerk darauf richten, wo die drei Voraussetzungen eine Rolle spielen.

Beweis von Gödels Unvollständigkeitssatz. Sei ein effektiv axiomatisierbares, Peano-Arithmetik umfassendes und korrekte System gegeben. Wir wollen die Annahme, es sei außerdem *vollständig*, das heißt, dass jede im angedachten Modell wahre Aussage auch einen formalen Beweis im System besitze, zu einem Widersprüch führen.

Dazu fixieren wir irgendeine Funktion $f : \mathbb{N} \to \{0, 1\}$, welche nicht berechenbar ist. Im Abschnitt über das Halteproblem haben wir gelernt, dass es solche Funktionen gibt; zum Beispiel könnten wir die Funktion

$$f(n) = \begin{cases} 1, & \text{falls das } n\text{-te Programm hält,} \\ 0, & \text{sonst,} \end{cases}$$

betrachten. Wir behaupten nun, dass unter der getätigten Annahme diese Funktion paradoxerweise doch berechenbar ist, und zwar durch den folgenden Algorithmus. Das zeigt dann den gewünschten Widerspruch.

1. Lese eine Zahl n als Eingabe ein.
2. Generiere nacheinander alle beweisbaren Aussagen des formalen Systems.
3. Wenn dabei die Aussage „$f(n) = 0$“ auftritt: Beende mit der Meldung „0“.
4. Wenn dabei die Aussage „$f(n) = 1$“ auftritt: Beende mit der Meldung „1“.

Dieser Algorithmus berechnet wirklich die Funktion f, denn das untersuchte formale System ist korrekt. Das hat zur Folge, dass wenn es im System einen Beweis der Aussage „$f(n) = 0$“ (bzw. „$f(n) = 1$“) gibt, dann auch tatsächlich $f(n) = 0$ (bzw. $f(n) = 1$) gilt.

Außerdem hält dieser Algorithmus bei jeder beliebigen Eingabe nach endlicher Zeit an. Denn wir nehmen ja an, dass jede wahre Aussage im formalen System auch beweisbar ist. Da entweder $f(n) = 0$ oder $f(n) = 1$ wahr ist, muss daher eine dieser beiden Aussagen auch einen Beweis besitzen. □

Aufgabe 10. *Die Voraussetzung nach Umfassung von Peano-Arithmetik*

Auch die Voraussetzung, dass das untersuchte formale System Peano-Arithmetik umfasst, geht – etwas versteckt – in den Beweis von Gödels Unvollständigkeitssatz ein. Siehst du, wo?

Tipp: Für den Beweis war es wichtig, dass „$f(n) = 0$“ und „$f(n) = 1$“ Aussagen des untersuchten formalen Systems sind.

6.2 Ein konkretes Beispiel für einen Gödel-Satz

Mit dem vorgestellten Beweis von Gödels Unvollständigkeitssatz kann man noch etwas unzufrieden sein: Er ist nämlich *unkonstruktiv*, das heißt, er liefert kein explizites Beispiel für eine Aussage, die wahr, aber nicht formal beweisbar ist. Daher führen wir nun noch einen zweiten Beweis, der allerdings einen anderen Makel hat: Er ist unvollständig.

Alternativer Beweis von Gödels Unvollständigkeitssatz. Wir betrachten folgende Aussage A: „Diese Aussage ist im formalen System nicht beweisbar."

Die Aussage A ist tatsächlich nicht im formalen System beweisbar. Denn wenn sie es wäre, würde sie – wegen der vorausgesetzten Korrektheit des formalen Systems – auch stimmen. Dann aber wäre sie nicht beweisbar. Das wäre ein Widerspruch.

Da A ihre eigene formale Unbeweisbarkeit behauptet, und A – wie wir gerade gesehen haben – tatsächlich im gegebenen System nicht beweisbar ist, ist A wahr. Somit ist A ein Beispiel für eine wahre und im System unbeweisbare Aussage. □

Das Problem an diesem Beweis ist, dass gar nicht klar ist, dass die Aussage A wirklich eine wohlgeformte Aussage des untersuchten Systems ist. Als deutscher Satz ist sie zwar grammatikalisch korrekt gebildet, doch schon Berrys Paradoxon (Abschnitt 1) hat uns gezeigt, dass nicht alle grammatikalisch korrekten Aussagesätze wirklich Aussagen im Sinne der Logik sein können.

Man kann nun beweisen, dass die Aussage A tatsächlich wohlgeformt ist, und so die Lücke in diesem alternativen Beweis schließen. Dabei gehen dann auch die bisher im Alternativbeweis nicht explizit verwendeten Voraussetzungen a) und b) ein. Zum Nachweis müssten wir uns aber tiefer mit technischen Details beschäftigen.

Ist die Lücke wirklich so groß? Wie schlimm kann es sein, mit nicht wohlgeformten Aussagen zu operieren? Ja und *sehr schlimm*. Das illustriert *Currys Paradoxon*:

Proposition 6.2. *Der Mond ist aus Käse.*

Beweis. Wir betrachten folgende Aussage B: „Wenn diese Aussage stimmt, dann ist der Mond aus Käse."

Wir möchten zunächst rein hypothetisch untersuchen, was wäre, wenn Aussage B stimmen würde. Nach Definition von B würde dann also aus der Korrektheit von B folgen, dass der Mond aus Käse ist. Somit wäre (in dieser hypothetischen Situation) der Mond wirklich aus Käse.

In dem vorherigen Absatz haben wir gezeigt: *Sollte B stimmen, so ist der Mond aus Käse.* Das ist aber gerade die Aussage von B. Somit haben wir gezeigt, dass B stimmt.

Da nun B stimmt, ist der Mond aus Käse. □

Da die Behauptung Unsinn ist, muss dieser „Beweis" fehlerhaft sein. Die eigentliche Argumentation war aber völlig korrekt; der Fehler liegt schon in der ersten Zeile: Die so genannte Aussage B ist in Wahrheit keine, zumindest nicht im strengen Sinn eines formalen Systems.

6.3 Der Vollständigkeitssatz

Theorem 6.3 (Gödels Vollständigkeitssatz)**.** *Gilt eine Aussage in allen Modellen eines formalen Systems (und nicht nur im intendierten Modell), so ist sie auch im System beweisbar.*

7 Wie geht's weiter?

In diesen 18 Seiten haben wir nur an der Oberfläche gekratzt. Folgende weitere Themen sind spannend:

- Philosophische Implikationen.
- Gödels Vollständigkeitssatz in mehr Details.
- Gödels zweiter Unvollständigkeitssatz, die Beweisbarkeit der Konsistenz von formalen Systemen betreffend.
- Beispiele für Aussagen, die in gewöhnlichen Axiomensystemen (wie Peano-Arithmetik oder Zermelo–Fraenkel-Mengenlehre) weder beweisbar noch widerlegbar sind und keinen selbstbezüglichen Charakter haben: die Kontinuumshypothese, das Auswahlaxiom, das Axiom der Konstruktibilität; aber auch konkrete Dinge wie die Frage nach dem Verhalten der *Goodstein-Folgen*.

Einiges davon kann man in folgendem netten Buch nachlesen, das ich sehr empfehle:

> Dirk W. Hoffmann. *Grenzen der Mathematik: Eine Reise durch die Kerngebiete der mathematischen Logik*. Springer-Verlag, 2013.