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

Gödels Unvollständigkeitssatz

Zirkelzettel vom 7. und 21. 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 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 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 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 3.1 und 3.2). 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.

[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.

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`
	⋮

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.

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.

[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 $F(n)$.

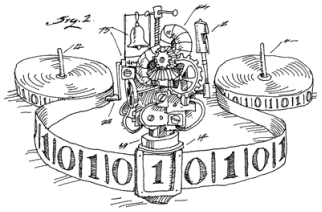
Wie jedes Programm ist auch R in der Liste aller Programme verzeichnet, etwa an m-ter Stelle: Es gilt also $F(m) = R$. Wir können uns nun fragen, ob R bei Eingabe von m hält oder nicht. Verfolgen wir den Programmfluss, sehen wir aber, dass beide Fälle zu einem Widerspruch führen. □

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?

3.1 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.

3.2 Weitere Übungsaufgaben

Aufgabe 2. *Das Halteproblem für halbreale Computerprogramme*

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

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; manche Leute halten sie für eine der größten ungeklärten Fragen der Physik und Informatik. Hast du eine Meinung dazu?

4 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 4.1. *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 5. *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.

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)

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. □

5 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 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

[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.)

> 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

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

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.

5.1 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 3.1 und 3.2).

[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.

Proposition 5.1. *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, dass ihre Nachkommaziffern berechnet. Das 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 5.2. *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.

5.2 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:

Vorläufge Definition 5.1. 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.

Vorläufge Definition 5.2. 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.

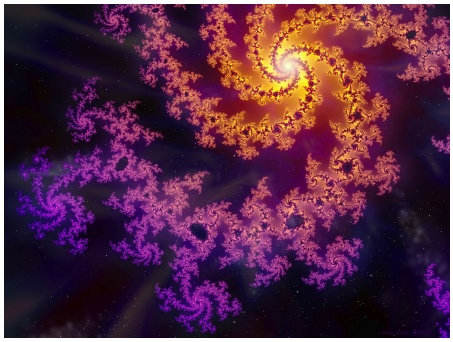
Um eine wirklich nützliche Definition von Zufälligkeit zu erhalten, sollten wir daher Abstand von statistischen Anforderungen nehmen und eher einen *informationstheoretischen Standpunkt* einnehmen: Lässt sich die regelmäßige Struktur einer Ziffernfolge in wenigen Worten beschreiben, so schätzen wir sie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

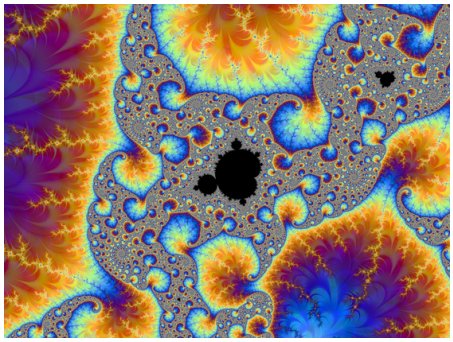
Definition 5.3. Der *Informationsgehalt* einer Ziffernfolge ist die Länge des kürzesten Programms, das diese Ziffernfolge berechnet und ausgibt.

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.

Abbildung 1 Wer interaktiv in 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





Definition 5.4. 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

897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748815209209628

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 5.5. *Die Ziffern von Ω sind zufällig.*

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. □

6 Formale Systeme und Modelle

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

Beispiel 6.1. 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 6.2. 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 6. *Grundlegende Fakten über Zahlen*

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 6.3. 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 informal erachtet – muss die Diskussion in diesem Abschnitt ihrerseits in einem bestimmten formalen System interpretieren. Irgendwo aber muss man beginnen! Man kommt nicht umhin, *Metalogik* vorauszusetzen.

7 Gödels Vollständigkeitssatz und sein Unvollständigkeitssatz