Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Kolmogorov-Komplexität und Datenkompression

Bachelorarbeit

im Studiengang B.Sc. Informatik

von

B.Sc. Sabrina Alexandra Gaube
Matrikelnr: 3069800

Prüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: Prof. Dr. Heribert Vollmer

Hannover, 11.01.2018

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 11.01.2018

B.Sc. Sabrina Alexandra Gaube

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Kolmogorov-Komplexität eines binären Wortes x (geschrieben: $K(x)$). Diese ist definiert als Länge des kürzesten Programms, das x darstellen kann. Außerdem wird als Anwendung die Kompression von binären Wörtern angesprochen.
Die Definition dieser Kolmogorov-Komplexität ist recht simpel, wohingegen die Bestimmung von $K(x)$ für ein bestimmtes Wort x weitaus schwieriger ist, wie wir im Folgenden sehen werden.
Die Arbeit gliedert sich in fünf Kapitel. Im ersten Kapitel werden erste Gedanken zur Datenkompression angeführt. Anhand von beispielhaften Wörtern wird gezeigt, dass man nicht immer direkt sehen kann, was die kleinste Zeichenanzahl ist, auf die man ein Wort durch Kompression reduzieren kann und soll damit eine Motivation geben, die Kolmogorov-Komplexität zu definieren.
In Kapitel 2 wird die Kolmogorov-Komplexität zunächst für bestimmte Turingmaschinen definiert. Es stellt sich heraus, dass die Unterschiede sich nur in Konstanten wiederspiegeln, weshalb wir eine universelle Kolmogorov-Komplexität auf Äquivalenzklassen einführen.
Kapitel 3 beschäftigt sich mit den Eigenschaften der Kolmogorov-Komplexität als mathematische Funktion von den binären Wörtern in die natürlichen Zahlen. Wir werden hier sehen, dass sie nicht berechenbar ist. Des Weiteren werden wir für spezielle Wortmuster obere Schranken der Kolmogorov-Komplexität bestimmen.
Das vierte Kapitel hat eine gedanklich andere Herangehensweise als die vorherigen Kapitel. Die vorherigen Kapitel folgen in den Beweisen hauptsächlich den Gedankengang der Dekompression, wohingegen sich dieses Kapitel mit sogenannten Kompressionsfunktionen beschäftigt. Wir werden sehen, dass es für jedes $m \in \mathbb{N}$ eine Kompressionsfunktion der Länge m gibt, die jedes Wort mit Kolmogorov-Komplexität maximal m optimal komprimiert. Außerdem werden wir sehen, dass es für jede Länge n und Kolmogorov-Komplexität m mit $m < n$ ein Wort x gibt, das nicht durch eine Kompressionsfunktion der Länge kleiner m komprimiert werden kann. Das letzte Kapitel gibt einen kurzen Ausblick, wofür man die Kolmogorov-Komplexität noch gebrauchen kann und erwähnt andere Anwendungen.

Inhaltsverzeichnis

1 Einleitung

In der Praxis möchte man möglichst viel Information in so wenig Platz wie möglich speichern oder übertragen. Möchte man ein Wort speichern oder übertragen, so benötigen gleichlange Wörter prinzipiell gleich viele Bits an Speicherplatz. Um Speicherplatz oder Sendezeit zu sparen ist man auf der Suche nach Algorithmen, die Wörter kürzer darstellen als sie eigentlich sind.
Bei komprimierten Wörtern ist es dabei wichtig, dass wir aus dem verkürzenden Algorithmus verlustfrei wieder das ursprüngliche Wort erhalten können. Denn wenn Wörter verlustbehaftet übertragen oder gespeichert werden, ist die Information im Allgemeinen nicht mehr zu gebrauchen.

Beispiel 1.1. *Man kann das binäre Wort*

$$100000000000000000000000000000000000 = 10^{35}$$

kürzer darstellen durch „Schreibe 1 und dann 35 Nullen“. Diese Zeichenkette enthält 29 Zeichen.
Dadurch wird eine Kompression von 36 auf 29 Zeichen bewirkt.

In der Theorie der Datenkomprimierung verwendet man jedoch keinen deutschen Schriftsatz, um kürzere Beschreibungsformen für binäre Wörter zu finden, sondern benutzt Algorithmen beziehungsweise Programme. Aus der Vorlesung 'Grundlagen der theoretischen Informatik' [Vol13] wissen wir, dass jeder Algorithmus oder auch jedes in Java geschriebene Programm äquivalent (bezüglich des Berechenbarkeitsbegriffs) zu einer deterministischen Turingmaschine ist. Daher wollen wir im Folgenden mit solchen Turingmaschinen argumentieren. Dabei werden wir den Zusatz 'deterministisch' nicht weiter erwähnen, obwohl alle Turingmaschinen in dieser Arbeit deterministisch arbeiten sollen.

Daten liegen in dieser Arbeit in Form von binären Wörtern vor. Deshalb beschäftigt sich diese Arbeit auch nur mit der verlustfreien Datenkompression.
Möchte man Daten komprimieren stellen sich zunächst folgende Fragen:

- Inwiefern ist es möglich Daten zu komprimieren?

- Kann man alle Wörter über dem binären Alphabet komprimieren?
- Wo liegen Grenzen der Datenkompression?

Als Erstes wollen wir uns daher systematisch anhand von Beispielen überlegen, welche Wörter man komprimieren kann.

Beispiel 1.2. *(Nach [LV08])*

1. *Betrachten wir das Wort*

$$00000000000000000000000000 = 0^{26}$$

so sehen wir, dass diese Zahl lediglich aus 26 Nullen besteht. Diese Folge ist komprimierbar durch einen zu Beispiel 1.1 analogem Algorithmus (z.B. „Print 26 times a 0“. Hierdurch erhalten wir eine Kompression auf 18 Zeichen).

2. *Das nächste zu betrachtende Wort ist das Wort*

$$01000110110000010100111001.$$

Das Wort sieht auf dem ersten Blick relativ willkürlich aus, jedoch kann man bei näherem Betrachten erkennen, dass es sich um eine Zeichenfolge mit folgendem Konstruktionsmuster handelt.
Wir haben 0,1,00,01,10,11,000,001,010,011,100,... *also Wörter über dem Alphabet* $\{0, 1\}$ *lexikographisch aufgezählt und nach* 26 *Zeichen wurde die Zeichenkette abgebrochen. Das heißt, zuerst kommen die Wörter mit einem Zeichen, dem Wert aufsteigend sortiert, dann kommen die Wörter mit zwei Zeichen, dem Wert aufsteigend sortiert usw.*
Wir erkennen ein Muster und vermuten daher, dass es sich um ein komprimierbares Wort handelt.

3. *Als letztes Beispiel betrachten wir das Wort*

$$10010011011000111011010000.$$

Auch dieses Wort sieht auf dem ersten Blick ebenfalls willkürlich aus. Dieses Wort ist als Ergebnis eines 26-maligen Münzwurfes entstanden. Als Ergebnis eines Münzwurfes interpretiert, besteht die Folge auch einem stochastischen Anpassungstest, wie zum Beispiel den χ^2-Test bestehen. Bei solchen Tests geht es nach [Grü14] um eine Überprüfung der Wahrscheinlichkeitsverteilung einer Stichprobe. Man möchte überprüfen, ob die Verteilung mit einer bestimmten,

vorgegebenen Verteilung übereinstimmt. Eine mögliche Verteilung wäre in diesem Fall eine Binomialverteilung mit 26 Versuchen, der Wahrscheinlichkeit $\frac{1}{2}$ *und der Zahl 12 für die Anzahl der Einsen. Man prüft also ob* $P(X = 12)$ *für* $Binom(26, \frac{1}{2})$ *einen Wert hat, der in einem gewissen Intervall liegt.*

Rein intuitiv merkt man, dass man zumindest periodische oder gespiegelte Wörter komprimieren kann. Wie genau man das tun kann, findet sich in Kapitel 3 wieder. Mit anderen Worten kann man sagen, dass Wörter, die in gewisse Muster fallen, komprimierbar wirken. Die Wörter die hingehen 'zufällig' wirken, scheinen nicht komprimierbar zu sein.
Für diese umgangssprachliche Umschreibung ist jedoch der Begriff des Zufalls, den man unter anderem aus der Stochastik kennt, nicht geeignet, wie wir auf den nächsten Seiten zu Beginn von Kapitel 2 sehen werden.

2 Einführung in die Kolmogorov-Komplexität

In diesem Kapitel wollen wir zeigen, was wir bereits angedeutet haben, dass 'zufällig' im stochastischen Sinn nicht der passende Begriff ist, um mit Worten zu beschreiben, was wir unter unkomprimierbar verstehen. Aus diesem Grund wollen wir die Kolmogorov-Komplexität später in diesem Kapitel definieren.

Nach [Grü14] kann man ein Wort x der Länge n über dem Alphabet $\{0, 1\}$ stochastisch als n-Maligen fairen Münzwurf modellieren. Dazu codiert man beispielsweise das Kopfsymbol als 0 und das Zahlsymbol als 1. Beide Ereignisse haben jeweils die Wahrscheinlichkeit $\frac{1}{2}$. Das resultierende Wort x wird mathematisch als 0-1-Folge betrachtet. Ein Münzwurf ist ein Bernoulliexperiment und daher gedächtnislos. Gedächtnislos heißt in diesem Fall, dass die Münze sich nicht merkt, dass vorher zum Beispiel dreimal ein Kopfsymbol aufgetaucht ist weshalb ggf. mal wieder ein Zahlsymbol dran wäre. Ein solches Verhalten würde die Wahrscheinlichkeitsverteilungen von Wurf zu Wurf verändern, was dem Verhalten eines Bernoulliexperiments widerspricht.
Betrachten wir nun die Wörter aus Beispiel 1.2 als 0-1-Folgen der Länge 26, die wir als solches Ergebnis eines 26-maligen fairen Münzwurfes interpretieren können.

1. Demnach hat die Folge 0^{26} die Wahrscheinlichkeit $(\frac{1}{2})^{26}$

2. die 0-1-Folge 01000110110000010100111001 enthält genau 11 Einsen und 15 Nullen, hat damit also die Wahrscheinlichkeit

$$\left(\frac{1}{2}\right)^{11} \cdot \left(\frac{1}{2}\right)^{15} = \left(\frac{1}{2}\right)^{26}.$$

3. Die Wahrscheinlichkeit für die dritte Folge aus Beispiel 2 berechnet sich wie folgt:

Sie besteht aus 12 Einsen und 14 Nullen.

$$\left(\frac{1}{2}\right)^{12} \cdot \left(\frac{1}{2}\right)^{14} = \left(\frac{1}{2}\right)^{26}.$$

Die Wahrscheinlichkeit eine bestimmte aber beliebige 0-1-Folge mittels fairen Münzwurfs zu erzeugen, lässt sich grundsätzlich berechnen mit

$$\left(\frac{1}{2}\right)^{\#Einsen} \cdot \left(\frac{1}{2}\right)^{\#Nullen}.$$

Da die Summe der Anzahl der Einsen und der Anzahl der Nullen bei einem Wort der Länge 26 über dem Alphabet $\{0,1\}$ genau 26 entspricht, ergibt sich für jedes beliebige solche Wort die obige Wahrscheinlichkeit $\left(\frac{1}{2}\right)^{26}$. Aus stochastischer Sicht ist demnach jedes Wort gleich zufällig, ganz egal ob es komprimierbar ist oder nicht.

2.1 Der Begriff der Kolmogorov-Komplexität bezüglich Turingmaschinen

Der Begriff des Zufalls aus der Stochastik ist also nicht der Begriff den wir benötigen, um das beschreiben zu können, was wir intuitiv als unkomprimierbar ansehen würden und zufällig nennen würden, wie wir es in Kapitel 1 beobachtet hatten.

Aus diesem Grund definieren wir die Kolmogorov-Komplexität zunächst bezüglich einer Turingmaschine M. Dafür bezeichne $M(x)$ die Ausgabe der Turingmaschine M bei Eingabe x. Später in diesem Kapitel wollen wir die Kolmogorov-Komplexität unabhängig von einer speziellen Turingmaschine definieren.
Dieses Unterkapitel benutzt die Definitionen und Sätze aus [LV08] und Beweise überwiegend aus [LG99].
Da wir die Kolmogorov-Komplexität anhand von Turingmaschinen einführen wollen, richten wir uns bei der einführenden Definition nach [Wat11].

Definition 2.1 (Kolmogorov-Komplexität)**.**
Die Kolmogorov-Komplexität $K_M(x)$ eines Wortes $x \in \{0,1\}^+$ bezüglich einer Turingmaschine M ist definiert als die Länge der kürzesten Eingabe $p \in \{0,1\}^+$ mit $M(p) = x$.

$$K_M(x) := \min\{|p| : p \in \{0,1\}^+, M(p) = x\}$$

Gibt es ein solches p nicht, so setzen wir

$$K_M(x) = \infty.$$

Um die generelle Komprimierbarkeit eines Wortes zu untersuchen, reicht es nicht aus, die Kolmogorov-Komplexität bezüglich einer beliebigen Turingmaschine zu untersuchen. Würde man dies tun, so kann es theoretisch sein, dass diese Turingmaschine das Wort optimal komprimiert, also $K_M(x)$ bereits das Minimum der Kolmogorov-Komplexitäten über alle Turingmaschinen ist. Allerdings ist dies im Allgemeinen nicht so wie folgendes Beispiel zeigen soll.

Beispiel 2.2. *Sei x ein unkomprimierbares Wort. Unkomprimierbar soll nun heißen, dass es keine Turingmaschine M' gibt, für die $K_{M'}(x) < |x|$ gilt. Dann ist die Turingmaschine M, die direkt hält und die Eingabe zurückgibt für x eine optimal komprimierende Turingmaschine. M hält immer genau nach $|x|$ Schritten, wenn x die Eingabe ist.*
Jedoch ist M für das Wort $x = 0^{26}$ nicht optimal komprimierend, da wir bereits in Beispiel 1.2 gesehen haben, dass wir das Wort zumindest auf $18 < 26 = |x|$ Zeichen komprimieren können.

Unser Ziel ist daher die Kolmogorov-Komplexität bezüglich einer universellen Turingmaschine zu definieren. Hierfür ist zunächst (nach [LV08]) die Vergleichbarkeit von Turingmaschinen bezüglich der Kolmogorov-Komplexitäten, die sie induzieren, notwendig.

Definition 2.3. *Eine Turingmaschine M_2 heißt additiv optimal in Bezug zu einer weiteren Turingmaschine M_1, wenn es eine Konstante c so gibt, dass*

$$K_{M_1}(x) \leq K_{M_2}(x) + c,$$

für alle $x \in \{0,1\}^*$ gilt.

Im Folgenden bezeichne $\langle M \rangle$ eine Beschreibung der Turingmaschine $M = (Z, \Gamma, \delta, z_0, E)$ durch ein Binärwort. Ein Beispiel für eine solche Beschreibung ist die aus [Vol13] bekannte Gödelisierung.

Als nächstes definieren wir die universelle Turingmaschine, um später die Kolmogorov-Komplexität darauf zurückführen zu können.

Definition 2.4. *Eine Turingmaschine $U = (\{0,1\}, \Gamma, \delta, z_0, E)$ heißt universell, falls für jede Turingmaschine M und jedes $w \in \{0,1\}^*$ gilt:*

- *U hält bei Eingabe $\langle M \rangle w$ genau dann, wenn M bei Eingabe w hält.*
- *Falls M bei Eingabe w hält, hat sie die gleiche Ausgabe wie U bei Eingabe $\langle M \rangle w$. Insbesondere akzeptiert M bei Eingabe w genau dann, wenn U bei Eingabe $\langle M \rangle w$ akzeptiert.*

Die Definition der universellen Turingmaschine ist sinnvoll, sofern sie existiert und additiv optimal ist. Um zuerst zeigen zu können, dass es universelle Turingmaschinen gibt müssen wir die Gödelisierung, wie wir sie aus [Vol13] kennen, umdefinieren. Dort werden lediglich die Zustandsübergänge kodiert. Für den Beweis müssen wir allerdings auch das Ende der Gödelisierung der Turingmaschine, wenn wir an dessen Ende direkt ein weiteres Wort dranhängen und die einzelnen Zustände in der Zustandsübergangsfunktion wiederfinden.
Hierzu definieren wir uns daher die Gödelisierung nach [Goo97] neu:

Definition 2.5. *Sei M eine Turingmaschine, wir kodieren die Zustandsübergänge*

$$\delta(z_i, a_j) = (z_{i'}, a_{j'}, X) \text{ durch}$$

$$0^{i+1}10^j10^{i'+1}10^{j'}10^m, \text{ wobei } m = \begin{cases} 1, & X = L \\ 2, & X = N \\ 3, & X = R \end{cases}.$$

Diese Übergänge zählen wir durch und bezeichnen nun mit $Code_1$ den ersten Übergang, $Code_2$ den zweiten und so weiter. Anschließend kodieren wir die Turingmaschine M wie folgt:

$$\langle M \rangle = 111Code_1 11Code_2 11Code_3 ... 11Code_g 111$$

Hierbei sei g die Anzahl an Zustandsübergängen.

Man erkennt also das Ende eines jeden Zustandsübergangs in dieser Gödelisierung durch das Auftauchen von zwei aufeinanderfolgenden Einsen und das Ende der Gödelisierung durch das zweite Auftauchen von drei aufeinanderfolgenden Einsen. Diese Eigenschaften werden wir uns im Beweis zu folgendem Satz zunutze machen.

Satz 2.6. *(nach [Goo97])*
Es gibt universelle Turingmaschinen.

Beweis. Gegeben sei eine feste aber beliebige Turingmaschine M. Wir konstruieren nun eine 2-Band Turingmaschine U, die den Anforderungen einer universellen Turingmaschine genügt.
U soll mit Eingabe $\langle M \rangle w$ simuliert werden. Dabei soll zunächst der Anfang des Wortes w, bzw. das Ende von $\langle M \rangle$ bestimmt werden. Anhand der oben definierten Gödelisierung kann man das Ende von $\langle M \rangle$ durch das zweite Auftauchen von '111' in der Zeichenkette finden. U kopiert dann w auf ihr zweites Band und löscht es vom

Ersten. Sie bewegt den Kopf des zweiten Bandes auf das erste Zeichen in w. Als nächstes schreibt U eine 0 hinter $\langle M \rangle$, da eine 0 der Kodierung des Startzustandes z_0 entspricht. Dann bewegt U den Kopf des ersten Bandes zum linken Ende. Eine Konfiguration $\alpha z_i a_j \beta$ M wird von U wie folgt kodiert:

- Auf Band 1 steht $\langle M \rangle$ und der aktuelle Zustand z_i kodiert durch 0^{1+i}.
- Auf Band 2 steht die Bandinschrift $\alpha a_j \beta$ von M.
- Der Kopf von Band 2 steht auf a_j.

Nun werden die Schritte folgendermaßen simuliert:

1. Suche auf Band 1 in $\langle M \rangle$ die Zeichenfolge $110^{i+1}10^j1$. Diese bildet den Anfang der Kodierung des Übergangs $\delta(z_i, a_j)$. Der Anfang der Folge lässt sich eindeutig wiederfinden, da in der oben definierten Gödelisierung zwei Einsen hintereinander nur als Trennung zweier Übergänge zu finden sind. Dazu kann $j \in \{1, 2, 3, 4\}$ im Zustand gespeichert sein. Ein Zustand z_i ist durch 0^{i+1} kodiert. Der aktuelle Zustand lässt sich durch die Abtrennung einzelner Einsen erkennen.

2. Lese die dahinterstehende Zeichenreihe der Form $0^{k+1}10^l10^m$. Ersetze hinter $\langle M \rangle$ die Kodierung des Zustands z_i, also 0^{i+1} durch 0^{k+1} und speichere l und m im Zustand. Damit ist im Zustand die Information über den nächsten auszuführenden Schritt gespeichert:
   Überschreibe die Zelle der Kopfposition mit a_l und bewege den Kopf.

3. Verändere Band 2 entsprechend der im Zustand gespeicherten Daten.

Als letztes muss noch gezeigt werden, dass die neu konstruierte Turingmaschine auch hält:
Dazu soll die Turingmaschine U halten, wenn der Zustand $z_{n-1} = z_e$, für einen Endzustand z_e von M auf Band 1 hinter $\langle M \rangle$ steht.
Also hält U genau dann, wenn M hält und ist, da M beliebig war, universell. □

Satz 2.7. *Sei U eine universelle Turingmaschine und $x \in \{0,1\}^*$. Dann gilt*

$$K_U(x) \leq K_M(x) + c,$$

für jede Turingmaschine M. Hierbei hängt c nur von M und nicht von x ab.

Der Satz sagt also aus, dass jede universelle Turingmaschine U additiv optimal ist.

Beweis. Sei dazu U eine universelle Turingmaschine und M eine Turingmaschine. Sei weiterhin $\langle M \rangle$ die Gödelisierung der Turingmaschine M, die U als Eingabe erhält, um M zu simulieren.
U berechnet bei Eingabe $\langle M \rangle p$ der Länge $|\langle M \rangle| + |p|$ die Ausgabe von M bei Eingabe p.
Sei nun p das kürzeste Wort, für das $M(p) = x$ gilt. Dann ist

$$K_A(x) = |p|.$$

Für U folgt damit

$$K_U(x) = \min_{|y|} \{U(y) = x\} \leq |\langle M \rangle| + |p| = |p| + c = K_A(x) + c.$$

Damit hängt c nur von der Länge der Gödelisierung von M ab und ist damit konstant. □

Wir können an dieser Stelle die Kolmogorov-Komplexität für universelle Turingmaschinen definieren, um schließlich ein Maß für die Komprimierbarkeit eines Wortes zu bekommen, das unabhängig von der betrachteten Turingmaschine ist.

Definition 2.8. *Es sei U eine fest gewählte universelle Turingmaschine. Dann definieren wir für $x \in \{0,1\}^+$ die Kolmogorov-Komplexität $K(x)$ durch*

$$K(x) := K_U(x).$$

Hierbei ist es wichtig zu betonen, dass U zwar eine feste Turingmaschine ist, aber beliebig unter den universellen Turingmaschinen gewählt werden kann.
Außerdem können wir die Kolmogorov-Komplexität für weitere Fälle definieren.

Definition 2.9.

- *Seien $x, y \in \{0,1\}^*$. Dann heißt*

$$K(x \mid y)$$

die bedingte Kolmogorov-Komplexität von x unter y und entspricht der Länge der kleinsten Turingmaschine, die unter Eingabe von y das Wort x erzeugt.

- *Die Kolmogorov-Komplexität einer natürlichen Zahl n ist definiert als*

Kolmogorov-Komplexität der Binärdarstellung dieser natürlichen Zahl.

$$K(n) = K(bin(n)).$$

Außerdem gibt es in der Literatur (z.B. [LV08]) den Begriff der längenbedingten Kolmogorov-Komplexität, den wir hier der Vollständigkeit halber angeben wollen.

Definition 2.10. *Die längenbedingte Kolmogorov-Komplexität eines Wortes x ist definiert als*

$$K(x \mid |x|).$$

2.2 Wohldefiniertheit der Kolmogorov-Komplexität

Die Definition der Kolmogorov-Komplexität ist zunächst von einer universellen Turingmaschine abhängig. Wir haben gezeigt, dass es eine universelle Turingmaschine gibt, aber wir haben bisher keine Aussage darüber getroffen, wie sich die $K_U(x)$ verändert, wenn wir die Turingmaschine U durch eine andere Turingmaschine U' austauschen und von K_U zu $K_{U'}$ übergehen.
Wir wollen nun im Folgenden zeigen, dass $K(x)$ wohldefiniert ist. Dazu führen wir zwei zusätzliche Bedingungen an die universelle Turingmaschine an. Durch diese zusätzlichen Bedingungen wollen wir Äquivalenzklassen über eine Äquivalenzrelation einführen. Die Konstruktion der Äquivalenzklassen folgt der Argumentation von [LV08].
Wir nehmen dazu an es würde ein Wort DATA (ohne Beschränkung der Allgemeinheit soll $|DATA| = 4$ gelten. Dies ist zulässig, da wir nur bis auf Konstanten unsere Aussagen treffen wollen) geben mit den folgenden Eigenschaften:

- Jede 1-Band Turingmaschine kann mit einem Programm simuliert werden, das dieses Wort DATA nicht als Teilwort enthält.

- Wird die Turingmaschine mit einem Wort der Form xDATAy auf dem Band gestartet, wobei x nicht das Teilwort DATA enthält, dann hält die Turingmaschine genau dann, wenn es auch mit y auf dem Ausgabeband und x auf dem Eingabeband gestartet hält und die gleiche Ausgabe erzeugt.

Bemerkung 2.11. *Jede universelle Turingmaschine kann so modifiziert werden, dass diese beiden Annahmen gelten. Dazu modifizieren wir die Übergangsfunktion so, dass die obigen Fälle erfüllt werden.*

Im Folgenden wollen wir daher annehmen, dass unsere universellen Turingmaschinen diese Eigenschaften haben.

Satz 2.12. *Es existiert eine Konstante $c \in \mathbb{N}$, sodass*

$$K(x) \leq |x| + c$$

gilt.

Beweis. Sei U eine universelle Turingmaschine. Daher kann U insbesondere die Turingmaschine, die nichts tut und direkt hält durch ein Programm p simulieren, welches DATA nicht enthält.
Aber dann wird U für jedes Wort x das Programm $p\text{DATA}x$ das Wort x ausgeben und halten. Daher erhält man für die Konstante $c := |p| + 4$ die gewünschte Ungleichung

$$K(x) \leq |x| + |p| + 4 = |x| + p$$

gilt. □

Eine ähnliche Aussage erhalten wir auch für die bedingte Kolmogorov-Komplexität. Hierfür werden wir die Standard-Kopplungsfunktion benutzen.

Definition 2.13. *Seien $x, y \in \{0,1\}^+$. Die Abbildung $\langle \cdot, \cdot \rangle$ mit*

$$\langle x, y \rangle = 1^{|x|}0xy$$

heißt Standard-Kopplungsfunktion.

Satz 2.14. *Es existiert eine Konstante $c \in \mathbb{N}$, sodass für alle x und y*

$$K(x \mid y) \leq K(x) + c$$

gilt.

Beweis. Sei M eine Turingmaschine, die für alle y, z die Ausgabe x berechnet, bei Eingabe $\langle z, y \rangle$. Dies ist genau dann der Fall, wenn eine universelle Turingmaschine U bei Eingabe $\langle z, \varepsilon \rangle$ die Ausgabe x berechnet. Es gilt also

$$K_M(x \mid y) = K_U(x) = K(x).$$

Nach [LV08] gibt es dann ein konstantes c, so dass

$$K(x \mid y) \leq K_M(x \mid y) + c = K(x) + c$$

gilt. □

Theorem 2.15 (Invarianz-Theorem)**.**
Sei $x \in \{0,1\}^*$ und seien S, T universelle Turingmaschinen, die die obigen Eigenschaften erfüllen. Dann existiert eine Konstante c, so dass

$$|K_T(x) - K_S(x)| \leq c$$

gilt.

Beweis. Im Beweis zu Satz 2.6 haben wir gesehen, dass es universelle 2-Band Turingmaschinen gibt, daher wollen wir ohne Beschränkung der Allgemeinheit annehmen, dass S und T solche 2-Band Turingmaschinen sind. Die 2-Band Turingmaschine S kann durch eine 1-Band Turingmaschine S_0 simuliert werden, indem man q auf das Band von S_0 schreibt, falls $S(q) = x$, also S bei Eingabe von q das Wort x erzeugt. Außerdem hält S_0 in endlich vielen Schritten mit x auf dem Band.
Wir können S_0 durch ein Programm pS_0 auf T simulieren, das nicht das Wort DATA als Teilwort enthält, nach Bemerkung 2.11.
Sei $x \in \{0,1\}^*$ ein Wort und sei q_x das kürzeste Programm das x auf S schreibt. Es gilt also

$$K_S(x) = |q_x|.$$

Betrachte nun das Programm $pS_0\text{DATA}q_x$. Diese gibt x aus und hat die Länge

$$|q_x| + |pS_0| + 4 \geq K_T(x).$$

Damit erhalten wir

$$K_T(x) - K_S(x) \leq |pS_0| + 4 =: c.$$

Die betragsmäßig andere Richtung der Ungleichung geht analog durch Vertauschen der Betrachtungen von S durch T und andersherum, sowie durch Vertauschen von S_0 durch T_0.

□

Das Invarianz-Theorem sagt also aus, dass man von einer universellen Kolmogorov-Komplexität sprechen kann, die nicht auf einem Interpretationsmechanismus für Zeichenketten oder Wörtern beruht, da die Unterschiede lediglich in einer Konstanten auftauchen. Wir können daher im Nachfolgenden Äquivalenzklassen definieren. Zuerst wollen wir jedoch ein Beispiel anführen, das das Invarianz-Theorem veranschaulicht.

Das folgende Beispiel betrachtet die Kolmogorov-Komplexität eines Programmes in LISP und die des entsprechenden Programms in FORTRAN.

Beispiel 2.16 (nach [LV08])**.**
Seien $\lambda_1, \lambda_2, ...$ lexikographisch aufgezählte syntaktisch korrekte LISP-Programme und seien weiterhin $\pi_1, \pi_2, ...$ lexikographisch aufgezählte syntaktisch korrekte FORTRAN-Programme.
Wir können für ein x, die Kolmogorov-Komplexitäten $K_{LISP}(x)$ und $K_{FORTRAN}(x)$ definieren.
Nach [LV08] beinhaltet jede Aufzählung ein universelles Programm. Die LISP-Aufzählung beinhaltet einen LISP-Interpreterprogramm, um ein beliebiges LISP-Programm interpretieren zu können. Allerdings gibt es auch ein LISP-Programm λ_p, dass gleichzeitig ein FORTRAN-Interpreter ist, in dem Sinne, dass es ein beliebiges FORTRAN-Programm ausführen kann.
Aus diesem Grund gilt

$$K_{LISP}(x) \leq K_{FORTRAN}(x) + |\lambda_p|$$

nach Satz 2.15.
Analog gibt es ein FORTRAN-Programm π_L, dass gleichzeitig ein LISP-Interpreter ist und es gilt folglich:

$$K_{FORTRAN}(x) \leq K_{LISP}(x) + |\pi_L|$$

Setzt man beide Gleichungen zusammen, so erhält man:

$$|K_{LISP}(x) - K_{FORTRAN}(x)| \leq |\lambda_p| + |\pi_L| := c,$$

für alle x. Die Kolmogorov-Komplexitäten von x der Sprachen LISP und FORTRAN unterscheiden sich also um eine Konstante, die nur von $|\lambda_p|$ und $|\pi_L|$, also der Länge der jeweiligen Interpreterprogramme, abhängt.

Mithilfe des Invarianz-Theorems lässt sich eine Äquivalenzrelation definieren woraus sich Äquivalenzklassen ableiten lassen.

Definition 2.17. *Zwei Kolmogorov-Komplexitäten $K_A(x)$ und $K_B(x)$ heißen äquivalent, in Zeichen*

$$K_A(x) \equiv K_B(x),$$

wenn es eine Konstante $c \in \mathbb{N}$ gibt, so dass

$$|K_A(x) - K_B(x)| \leq c$$

gilt.

Bemerkung 2.18. *Die Relation $\equiv$ aus Definition 2.17 ist eine Äquivalenzrelation.*

Beweis. Wir müssen zeigen, dass $\equiv$ reflexiv, symmetrisch und transitiv ist. Seien A, B und C universelle Turingmaschinen und seien c, c_1 und $c_2 \in \mathbb{N}$ Konstanten. Hierfür benutzen wir, dass alle Kolmogorov-Komplexitäten, als Funktion aufgefasst, in die natürlichen Zahlen abbilden.

- symmetrisch: Es gilt $|K_A(x) - K_A(x)| = 0 \leq c$, für alle natürlichen Zahlen c.

- reflexiv: Angenommen es gilt $|K_A(x) - K_B(x)| \leq c$. Dann gilt

  $$|K_A(x) - K_B(x)| = |K_B(x) - K_A(x)|,$$

  da die Differenz von zwei natürlichen Zahlen innerhalb eines Betrags kommutativ ist. Demnach ist auch

  $$|K_B(x) - K_A(x)| \leq c.$$

- transitiv: Angenommen es gelten sowohl $|K_A(x) - K_B(x)| \leq c_1$ als auch $|K_B(x) - K_C(x)| \leq c_2$. Dann gilt

  $$\begin{aligned} |K_A(x) - K_C(x)| &= |K_A(x) - K_B(x) + K_B(x) - K_C(x)| \\ &= |K_A(x) - K_B(x) - (K_C(x) - K_B(x))| \\ &\leq |K_A(x) - K_B(x)| + |K_C(x) - K_B(x)|, \end{aligned}$$

  wobei die letzte Zeile mittels Dreiecksungleichung folgt.
  Wir hatten bereits bei der Reflexivität gesehen, dass wir die Differenz im Betrag ganz rechts umdrehen dürfen. Daher folgt

  $$\begin{aligned} &|K_A(x) - K_B(x)| + |K_C(x) - K_B(x)| \\ = &|K_A(x) - K_B(x)| + |K_B(x) - K_C(x)| \\ &\leq c_1 + c_2. \end{aligned}$$

Wenn wir unsere neue Konstante auf

$$c := c_1 + c_2$$

setzen, dann erhalten wir das gewünschte:

$$|K_A(x) - K_C(x)| \leq c$$

□

Definition 2.19.

- *Durch die Relation $\equiv$ aus Definition 2.17 werden die Äquivalenzklassen*
$$[K_A] := \{K_B \mid K_B \equiv K_A\}$$
induziert.

- *Die Ordnung $\leq$ auf den Äquivalenzklassen definieren wir wie folgt.*
Für Äquivalenzklassen $[K_A]$ und $[K_B]$ gilt $[K_A] \leq [K_B]$, wenn für den kanonischen Vertreter
$$K_A(x) \leq K_B(x) + c$$
gilt.

Bemerkung 2.20. *Die Ordnung aus Definition 2.19 ist eine partielle Ordnung auf den Äquivalenzklassen. Insbesondere gibt es ein minimales Element K_{A_0}, so dass für alle K_B die Ungleichung*

$$[K_{A_0}] \leq [K_B]$$

gilt.

Beweis. Eine Ordnung heißt partiell, wenn sie reflexiv, antisymmetrisch und transitiv ist. Reflexivität und Transitivität lassen sich ähnlich zu Bemerkung 2.18 zeigen.

- reflexiv: $K_A(x) \leq K_A(x) + c$, gilt für alle Konstanten natürlichen Zahlen c.

- antisymmetrisch: Angenommen es gelten sowohl $K_A(x) \leq K_B(x) + c_1$ als auch $K_B(x) \leq K_A(x) + c_2$. Subtrahiert man in der ersten Ungleichung beide Seiten mit $K_B(x)$ und in der zweiten Ungleichung beide Seiten mit $K_A(x)$, so erhält man
$$K_A(x) - K_B(x) \leq c_1 \text{ und } K_B(x) - K_A(x) \leq c_2.$$

Insgesamt erhält man damit, da sowohl $K_B(x)$ als auch $K_A(x)$ positiv sind

$$|K_A(x) - K_B(x)| \leq \max\{c_1, c_2\} =: c.$$

Damit liegen $K_A(x)$ und $K_B(x)$ in der selben Äquivalenzklasse und sind dementsprechend äquivalent.

- transitiv: Angenommen es gilt $K_A(x) \leq K_B(x) + c_1$ und außerdem noch $K_B(x) \leq K_C(x) + c_2$. Dann lässt sich $K_B(x)$ in der ersten Ungleichung durch die rechte Seite der zweiten Ungleichung ersetzen, da $\geq$ dann immer noch gilt und man erhält
$$K_A(x) \leq (K_C(x) + c_2) + c_1.$$
Setzt man die neue Konstante auf $c := c_1 + c_2$, so erhält man das gewünschte
$$K_A(x) \leq K_C(x) + c.$$

Wir haben also eine partielle Ordnung. Da eine partielle Ordnung auf einer Menge, die nach unten beschränkt ist, ein minimales Element besitzt, können wir dieses K_{A_0} nennen und haben unsere Aussage gezeigt. Die Beschränkung nach unten kommt durch die natürliche Grenze der 0 zustande, da kein binäres Wort auf weniger als 0 Zeichen komprimiert werden kann. □

Dadurch ist also die Kolmogorov-Komplexität auch wirklich so definiert, wie es ursprünglich geplant war, nämlich als Maß für die Komprimierungsfähigkeit eines Wortes unabhängig von der Realisierung der Turingmaschine. Wir können dadurch auch Turingmaschinen in der Definition der Kolmogorov-Komplexität ersetzen durch z.B. Programme in Java.

Da die Kolmogorov-Komplexität eines Wortes nur bis auf eine Konstante definiert ist, lässt sich statt unserer Definition auch die Folgende benutzen.

Definition 2.21. *Die Kolmogorov-Komplexität $K(x)$ eines Wortes x ist die Länge der kürzesten Turingmaschinenbeschreibung die x erzeugt.*

Beide Definitionen sind äquivalent, da wir die Länge der kürzesten Turingmaschinenbeschreibung in der vorherigen Definition in der Konstanten c wiederfinden. Mithilfe der neuen Definition lässt sich die Kolmogorov-Komplexität auf die bedingte Kolmogorov-Komplexität zurückführen.

Bemerkung 2.22. *Die Kolmogorov-Komplexität $K(x)$ eines Wortes x ist ein Spezialfall der bedingten Kolmogorov-Komplexität. Es gilt*

$$K(x) = K(x \mid \varepsilon).$$

2.3 Nichtkomprimierbarkeit

Wir haben die Kolmogorov-Komplexität ursprünglich eingeführt, um einen neuen 'Zufalls'begriff einzuführen, nachdem ein Wort 'zufällig' ist, wenn es unkomprimierbar ist. Daher definieren wir mithilfe der Kolmogorov-Komplexität die c-Komprimierbarkeit und die daraus resultierende Unkomprimierbarkeit bzw. 'Zufälligkeit'.

Definition 2.23. *(nach[LV08])*
Sei $c \in \mathbb{N}$. Ein Wort $x \in \{0,1\}^+$ heißt c-komprimierbar, falls

$$K(x) < |x| - c$$

gilt.
Gilt $K(x) \geq |x| - c$, so heißt x c-unkomprimierbar oder zufällig.

Mithilfe dieses Begriffes können wir nun eine unserer Eingangsfragen klären, nämlich die, ob alle Wörter komprimierbar sind:

Satz 2.24. *(nach[LV08])*
Es gibt mindestens

$$2^n - 2^{n-c+1} + 1$$

c-unkomprimierbare Wörter der Länge n.

Beweis. Damit ein Wort x der Länge n c-unkomprimierbar ist, muss

$$K(x) \leq |x| - c = n - c$$

gelten. Demnach müssen Wörter auf die Länge $n-c$ verlustfrei komprimiert werden können. Als mögliche Resultate einer solchen Komprimierung kommen ebensolche Wörter der Länge $n - c$ oder kleiner infrage. Hiervon gibt es

$$\sum_{i=0}^{n-c} 2^i = 2^{n-c+1} - 1.$$

Da es 2^n Wörter der Länge n gibt, können also mindestens

$$2^n - (2^{n-c+1} - 1) = 2^n - 2^{n-c+1} + 1$$

Wörter nicht c-komprimierbar sein. □

Um uns einmal vor Augen zu führen, was die Aussage des Satzes genau bedeutet, beziehungsweise wie groß die Folgen für die Datenkompression sind, betrachten wir folgendes Beispiel, das angelehnt an ein Korollar von [LG99] ist.

Beispiel 2.25. *Wir betrachten die Wörter, die 7-unkomprimierbar sind. Die 7-unkomprimierbaren Wörter sind nach Definition genau die Wörter, deren Kolmogorov-Komplexität geringer als $n - 7$ ist, also genau die Wörter, bei denen man durch eine andere Darstellung mindestens 7 Zeichen einspart.*
Setzen wir diesen Wert für c in die Formel aus Satz 2.24 ein, erhalten wir

$$2^n - 2^{n-c+1} + 1 = 2^n - 2^{n-7+1} + 1 = 2^n - 2^{n-6} + 1$$

Wörter die 7-unkomprimierbar sind. Möchte man daraus nun den Anteil dieser Wörter unter den Wörtern der Länge n bestimmen, so berechnet man das grundsätzlich mit

$$\frac{2^n - 2^{n-6} + 1}{2^n} = 1 - \frac{1}{2^6} + \frac{1}{2^n} = \frac{63}{64} + \frac{1}{2^n}.$$

Da der Nenner mit wachsendem n immer größer wird, wird der hintere Bruch immer kleiner, also geht der Anteil der 7-unkomprimierbaren Wörter bei wachsendem n von oben gegen $\frac{63}{64} \approx 98,4375\%$. Für konkrete Werte für n erhalten wir

$n = 7$ *:* $\frac{63}{64} + \frac{1}{2^7} = \frac{127}{128} \approx 99,219\%$

$n = 10$*:* $\frac{63}{64} + \frac{1}{2^{10}} = \frac{1009}{1024} \approx 98,535\%$

$n = 15$*:* $\frac{63}{64} + \frac{1}{2^{15}} \approx 98,44\%$

$n = 16$*:* $\frac{63}{64} + \frac{1}{2^{16}} \approx 98,439\%$

Wir sehen also, dass bereits bei Wörtern der Länge 16 sich der Anteil der 7-unkomprimierbaren Wörtern vom Grenzwert erst in der dritten Nachkommastelle unterscheidet.

3 Eigenschaften der Kolmogorov-Komplexität

Wir wollen nun die Eigenschaften der Kolmogorov-Komplexität untersuchen. Zunächst wollen wir die Unberechenbarkeit und weitere mathematische Eigenschaften zeigen. Da die Kolmogorov-Komplextät unmittelbar mit der Datenkompression zusammenhängt, wollen wir am Ende des Kapitels Oberschranken für gewisse x für $K(x)$ bestimmen.

3.1 Kolmogorov-Komplexität als Funktion in die natürlichen Zahlen

Hierzu betrachten wir in diesem Unterkapitel $K(x) : \{0,1\}^+ \to \mathbb{N}$, als Funktion in die natürlichen Zahlen, wie es in [LV08] der Fall ist.

3.1.1 Unberechenbarkeit der Kolmogorov-Komplexität

Um zu zeigen, dass die Kolmogorov-Komplexität unbeschränkt ist, zeigen wir zunächst das sogenannte Schubfachprinzip nach [AZ14].

Lemma 3.1 (Schubfachprinzip)**.**
Wenn man m Objekte auf n Fächer verteilen möchte und $m > n$ gilt, so landet in mindestens einem Fach mehr als ein Objekt.

Beweis. Angenommen das Schubfachprinzip gilt nicht, dann gibt es in jedem Fach maximal ein Objekt. Es gibt genau n Fächer, also wurden maximal n Objekte verteilt. Dies ist ein Widerspruch zur Annahme, dass $m > n$ ist. □

Satz 3.2. *Es gibt Wörter, die eine beliebig große Kolmogorov-Komplexität haben. Mit anderen Worten: Für alle $n \in \mathbb{N}$ existiert ein Wort $x \in \{0,1\}^+$ mit $K(x) \geq n$. Insbesondere ist $K(x)$ unbeschränkt.*

Beweis. Angenommen es gibt ein $m \in \mathbb{N}$, so dass es kein Wort x gibt mit $K(x) \geq m$. Dann gibt es $\sum_{i=0}^{m-1} 2^i$ Wörter mit der Länge maximal m.

$$\sum_{i=0}^{m-1} 2^i = 2^m - 1.$$

Allerdings gibt es genau 2^m Wörter der Länge m. Nach dem Schubfachprinzip wissen wir, dass es damit genau ein Wort gibt mit Kolmogorov-Komplexität größer m, da wir verlustfrei komprimieren wollen. Die Verlustfreiheit erzwingt, dass wir aus jeder komprimierten Darstellung eines Wortes nur genau ein ursprüngliches Wort erhalten können. Demnach haben wir einen Widerspruch, also ist unsere Annahme, dass es ein solches m gibt falsch. □

Jetzt haben wir gesehen, dass die Kolmogorov-Komplexität ins unermessliche steigen kann. Für die Datenkompression heißt das zunächst einmal, dass selbst die komprimierten Beschreibungen der Wörter, die wir komprimieren wollen, unendlich bzw. sehr lang werden können. Wir sehen also eine erste Grenze der Datenkompression, obwohl wir uns nur über theoretische Eigenschaften Gedanken gemacht haben und keinen konkreten Algorithmus betrachtet haben. Diese Grenze ist außerdem für alle möglichen Algorithmen fixiert, da wir bereits in Kapitel 2 gesehen haben, dass die Kolmogorov-Komplexität universell ist.

Theorem 3.3. *$K(x)$ ist nicht berechenbar.*

Beweis. (Nach [LG99])
Wir zeigen diese Aussage durch einen Widerspruchsbeweis, dazu nehmen wir zunächst an, $K(x)$ sei berechenbar.
Sei $c \in \mathbb{N}$ groß genug. Seien die Wörter in $\{0,1\}^*$ lexikographisch geordnet und sei nun $x(k)$ das k-te Wort dieser Ordnung.
Sei x_0 das kleinste Element der Ordnung, für das $K(x_0) \geq c$ gilt. Wir haben bereits gesehen, dass die Kolmogorov-Komplexität unabhängig von der Darstellung des Algorithmus ist, daher geben wir hier einen in Pseudecode an.

```
int k;
int x(int k){
...
}
int Kolmogorov(int k){
...
}

void main(){
   int k = 0;
   while (Kolmogorov(k) < c)
      k++;
   print(x(k));
}
```

Die Funktion $x(k)$ sei so implementiert, dass es das gewünschte x_k, wie es oben definiert war, zurückgibt. Die Funktion $Kolmogorov(k)$ sei so programmiert, dass sie die Kolmogorov-Komplexität von $x(k)$, für ein eingegebenes k berechnet, da wir angenommen haben, dass sie berechenbar ist.

$$\text{Kolmogorov(k)} = K(x_k) = c_{x_k}.$$

Das Programm gibt das oben gesuchte x_0 zurück.
Die Kolmogorov-Komplexität für ein bestimmtes Wort ist nach Definition konstant. Außerdem ist diese Konstante c_{x_k} von c unabhängig. Die Anzahl der Symbole ist nur $\log(c) + \mathcal{O}(1)$. Das heißt wenn wir c groß genug wählen, dann besteht das Programm aus weniger als c Wörtern und gibt x_0 zurück. Damit gilt $K(x_0) < c$ und wir haben die Aussage zu einem Widerspruch geführt.

□

Dementsprechend können wir für ein gegebenes x die komprimierteste Beschreibung dieses Wortes nicht berechnen. Hätten wir $K(x)$ gegeben, so könnten wir (nach [FF17]) alle Wörter der Länge $K(x)$ durchgehen und prüfen, welches davon unser ursprüngliches x erzeugt. Da aber $K(x)$ nicht berechenbar ist, ist diese Methode nicht einmal auf einzelne Wörter anwendbar.

Theorem 3.4. *Es existiert eine monoton mit t steigende total rekursive Funktion $\phi(t, x)$, für die gilt*

$$\lim_{t \to \infty} \phi(t, x) = K(x).$$

Beweis. Ein Beweis findet sich unter anderem in [LV08] □

Insgesamt konnten wir also erkennen, dass $K(x)$ unbeschränkt und nicht berechenbar ist, allerdings existiert eine Funktion, die $K(x)$ annähert.

3.1.2 Weitere mathematische Eigenschaften von $K(x)$

Aus mathematischer Sicht lassen sich weitere Eigenschaften erkennen, die wir in diesem Unterkapitel zeigen wollen, die sich nach [LV08] richten.

Bemerkung 3.5. *$K(x)$ ist stetig auf $\mathbb{N}$, zumindest in dem Sinn, dass es eine Konstante c gibt, für die*

$$|K(x) - K(x \pm h)| \leq 2|h| + c$$

gilt.

Beweis. Wenn wir eine Turingmaschine gefunden haben, die x erzeugen kann, dann können wir diese Turingmaschine modifizieren, so dass sie h von der Ausgabe subtrahiert oder zur Ausgabe addiert. □

Das folgende Lemma wird benötigt um die überwiegend logarithmische Struktur von $K(x)$ zu zeigen.

Lemma 3.6. *Sei $c \in \mathbb{N}$ konstant. Für festes y und jede endliche Menge A mit $|A| = m$ gibt es mindestens $m(1 - 2^{-c}) + 1$ Elemente mit*

$$K(x \mid y) \geq \log m - c.$$

Beweis. Die Anzahl der Wörter mit weniger als $\log m - c$ Zeichen ist

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1.$$

Daher sind nach dem Schubfachprinzip mindestens $m - m \cdot 2^{-c} + 1$ Elemente in A, die länger sind als $\log m - c$ oder dies als Länge haben. □

Satz 3.7. *$K(x)$ ist überwiegend logarithmisch, in dem Sinn, dass*

$$\log x - c < K(x) \leq \log x - c$$

gilt.

Beweis. $K(x)$ ist nach Satz 2.12 von oben durch $\log x + c$ beschränkt. Andererseits gilt für alle k, dass es nur maximal 2^{n-k} verschiedene x gibt für die

$$K(x) < \log x - k$$

gilt, für $n \approx \log x$, nach Lemma 3.6. Dieses Lemma dürfen wir anwenden, da wir $K(x \mid y)$ so definiert haben, dass $K(x) = K(x \mid \varepsilon)$ gilt. □

Definition 3.8. *$K(x, y) := K(\langle x, y\rangle)$, wobei $\langle \cdot, \cdot \rangle$ die Standardkopplungsfunktion aus Definition 2.13 ist.*

Satz 3.9. *$K(x)$ ist subadditiv und es gilt*

$$K(x, y) \leq K(x) + K(y) + 2\log(\min\{K(x), K(y)\}).$$

Beweis. Ein Beweis zu diesem Satz findet sich in [LV08]. □

Aus dem Schubfachprinzip lässt sich außerdem noch folgende Eigenschaft herleiten.

Satz 3.10. *Es gibt maximal 2^n Wörter mit Kolmogorov-Komplexität n.*

Beweis. Es gibt genau 2^n verschiedene Programme der Länge n. Damit es mehr Worte mit $K(x) = n$ gäbe, müsste mindestens eins dieser Programme mehr als ein Wort erzeugen, was der verlustfreien Kompression widerspricht, da demnach eine eindeutige Zuordnung gefordert ist. □

3.2 Oberschranken der Kolmogorov-Komplexität für spezielle Wortmuster

Da die Kolmogorov-Komplexität nicht berechenbar ist, sind wir nun zumindest auf der Suche nach Oberschranken für $K(x)$. Hierfür sehen wir uns zunächst jeweils ein Beispielwort an und folgern von dort aus eine Schranke für gewisse Wortmuster.
Um eine obere Schranke für Palindrome, also Wörter die rückwärts gelesen dem vorwärtsgelesenen Wort entsprechen, zu finden, schauen wir uns zunächst das folgende Beispiel an.

Beispiel 3.11. *Wir nehmen uns ein Palindromwort mit gerader Länge. Sei nun*

$$x = 10011001 = x^R.$$

Wir bemerken, dass $|x| = 8 =: n$ ist. Sei M die Turingmaschine, die bei Eingabe x die erste Hälfte des Wortes erzeugt, die wir x_1 nennen.
Ist nun x ein Palindrom ungerade Länge, so schreibe zunächst das vorletzte Zeichen von x_1 hinter x_1, dann das vorvorletzte Zeichen von x_1 usw. auf das Band. Für den zweiten Teil eines Palidromwortes (welches wir x_2 nennen) gerader Länge wissen wir, dass $x_1^R = x_2$ gilt. Also spiegeln wir x_1 auf den Bandinhalt dahinter.
Die Größe der Turingmaschine bezeichnen wir mit c. Dementsprechend haben wir für die Kolmogorov-Komplexität des Wortes x der Länge 8 eine obere Schranke von $4 + c$.

Wir erhalten also für beliebige Palindrome folgende Aussage analog:

Satz 3.12. *Sei x ein Palindrom über dem Alphabet $\{0, 1\}$. Dann ist $\lfloor \frac{|x|}{2} \rfloor + c$ eine obere Schranke für $K(x)$, für ein $c \in \mathbb{N}$.*

Hierbei bezeichnet $\lfloor ... \rfloor$ die untere Gaußklammer, also die Abrundungsfunktion.

Beweis. Gehe wie im Beispiel vor. Sei $|x| = n$ und sei M die Turingmaschine, die die erste Hälfte des Wortes, bis zum $\lfloor \frac{n}{2} \rfloor$-ten Zeichen auf das Band schreibt und spiegele dann die erste Hälfte um das komplette Wort x auf dem Band zu erhalten. Hierbei wird zuerst geprüft, ob n gerade oder ungerade ist. Bei geradem n wird die erste Hälfte y auf dem Band rechts mit y^R ergänzt. Ist n ungerade, so wird y^R ebenfalls erzeugt, allerdings wird das erste Zeichen von y^R nicht auf das Band geschrieben, da das Zeichen in der Mitte eines Palindromes ungerade Länge nicht dupliziert werden muss. Mit c als Größe der Turingmaschine ergibt sich

$$K(x) \leq \lfloor \frac{|x|}{2} \rfloor + c.$$

□

Als Eingangsbeispiel (Beispiel 1.2) haben wir das Wort 0^{26} betrachtet und bereits erkannt, dass es komprimierbar ist. Dieses wollen wir hier nun wieder aufgreifen.

Beispiel 3.13. *Betrachten wir das Wort $x = 0^{26}$. Hierfür sei nun M eine Turingmaschine mit Eingabe 26 als Binärwort. Ersetze nun die Binärdarstellung von 26 in eine Unärdarstellung der Form 0^{26}.*
Wir benötigen also nur $\log 26$ des Platzes, den 0^{26} verbrauchen würde. Wenn c die Größe der Turingmaschine bezeichnet, dann gilt also

$$K(x) \leq \log 26 + c$$

als Oberschranke.
Vergleichen wir nun unsere anfangs bestimmte Komprimierung auf 18 Zeichen so sehen wir, dass $\log 26 + c \approx 4,7 + c$ *ist und damit unsere Komprimierung für kleine* c *relativ schlecht war.*

Satz 3.14. *Sei* x *ein binäres Wort der Länge* n*. Hat* x *die Form* 0^n *oder* 1^n*, so gilt*

$$K(x) \leq \log n + c.$$

Alternativ könnte man den Satz auch für eine natürliche Zahl mit Binärdarstellung der Länge n, die nur aus Nullen oder nur aus Einsen besteht formulieren, da $K(bin(x)) =: K(x)$. Insbesondere fallen, für alle natürlichen Zahlen x alle Zahlen der Form $2^x - 1$ in diese Kategorie.

Beweis. Gehe hierbei wie im Beispiel vor und ersetze jede 26 durch ein n und ggf. jede 0 durch eine 1. □

Als nächstes Beispiel wollen wir eine Oberschranke für die Kolmogorov-Komplexität der n-ten Primzahl ansehen.

Beispiel 3.15. *Sei* $x = 1011$ *also die Binärdarstellung der Zahl 11 und damit der fünften Primzahl. Sei* M *eine Turingmaschine, die eine fünf in Binärdarstellung, also* 101 *als Eingabe erhält. Nun zählt man alle Primzahlen, die man zum Beispiel über das Sieb des Eratosthenes erhalten kann und lässt dabei einen Zähler laufen. Ist dieser Zähler bei fünf angekommen, so unterbricht man das Programm und gibt die aktuelle Primzahl aus. Wenn wir die Größe der Turingmaschine, die die Primzahlen berechnete, mit* c *bezeichet, so hat man eine obere Schranke von* $\log 5 + c$*.*

Analog erhält man so für ein Wort x, das der Binärdarstellung einer Primzahl entspricht folgende Aussage:

Satz 3.16. *Sei* x *die Binärdarstellung der* n*-ten Primzahl. Dann ist*

$$K(x) \leq \log n + c$$

eine Oberschranke für die Kolmogorov-Komplexität von x*.*

Beweis. Der Beweis funktioniert analog zum obigen Beispiel. Man ersetze dabei jede fünf durch ein n. □

Bemerkung 3.17. *Vergleicht man dieses Ergebnis einmal mit der Aussage von Beispiel 2.25, so sieht man, dass die Primzahlen ab einem gewissen* n *(abhängig*

von der Konstanten c), zu der Minderheit der Zahlen gehören, die 7-komprimierbar sind. Dies ist der Fall da

$$\log n \leq n - 7,$$

für alle $n \geq 11$ *gilt, außerdem* n *schneller als* $\log n$ *wächst und* $\log(11) \approx 3,46$.

Beispiel 3.18. *Ähnlich zur fünften Primzahl kann man auch die fünfte Fibonnacci-Zahl, die den Wert 8 hat, mit einer Turingmaschine berechnen, die 5 in Binärdarstellung, also* $x = 101$ *als Eingabe erhält. Die Turingmaschine hat zu Beginn zwei Einsen auf dem Arbeitsband und addiert diese insgesamt* $5 - 1 = 4$ *Mal, in dem sie gleichzeitig bei jedem Durchlauf runterzählt und schließlich bei 0 stoppt.*
Damit erhalten wir für $K(8_{10}) = K(1000_2)$ *eine obere Schranke von* $\log 5 + c$.

Satz 3.19. *Ist* x *die* n*-te Fibonacci-Zahl, so gilt* $K(x) \leq \log n + c$.

Beweis. Ersetze jede 5 im Beispiel durch ein n und gehe analog vor. □

Wir können dadurch direkt folgern, dass wir die Oberschranke

$$K(x) \leq \log n + c,$$

für alle Wörter x setzen, die einer gewissen Sprache angehören. Die Wörter der Sprache müssen lediglich durch eine Turingmaschine berechnet bzw. erzeugt werden, wie es bei den Primzahlen durch das Sieb des Eratosthenes oder wie bei den Fibanacci-Zahlen durch reine Addition der Startwerte der Fall war. Allerdings ist zu beachten, dass die Konstante c, die sich dabei jedesmal ergibt, jedesmal eine andere ist. Die Konstante ist immer abhängig von der Berechnungsvorschrift der Turingmaschine, die die Sprache erzeugt.

Als nächstes Untersuchen wir periodische Wörter. Diese hatten wir auch bereits in Kapitel 1 als komprimierbar vermutet.

Beispiel 3.20. *Sei* $x = 10010111001011$, *also* $x = yy$ *für* $y = 1001011$. *Um aus* y *das Wort* x *herzustellen, muss eine Turingmaschine bei Eingabe* y *nur das Wort rechts auf das Band direkt neben* y *noch einmal schreiben. Somit gilt*

$$K(x) \leq K(y) + c.$$

Satz 3.21. *Sei* x *ein Wort der Form* $x = yy$ *dann gilt*

$$K(x) \leq K(y) + c.$$

Beweis. Analog zum Beispiel. □

Nun folgt eine Obergrenze, für beliebige Wörter der Länge n von denen die Anzahl der Einsen bekannt ist.

Satz 3.22. *(Nach [LG99])*
Sei x ein binäres Wort der Länge n mit k Einsen. Dann gilt:

$$K(x) \leq \log_2 \binom{n}{k} + \log_2(n) + \log_2(k) + c$$

Beweis. Das Wort x kann als t-tes Wort in einer lexikographischen Ordnung aller Wörter der Länge n mit genau k Einsen bezeichnet werden. Die Anzahl aller Wörter der Länge n mit exakt k Einsen kann mittels Binomialkoeffizient $\binom{n}{k}$ ermittelt werden ('Man suche von n Stellen k Stellen heraus, die aus 1 gesetzt werden').
Die Beschreibung der Zahlen t, n und k benötigt lediglich

$$\log_2 \binom{n}{m} + 2\log_2(n) + 2\log_2(k)$$

Bits.
Ein Programm, dass das geeignete Wort berechnet, benötigt nur eine Konstante Anzahl an Bits ($\rightarrow +c$). □

Da es im binären Alphabet nur zwei Zeichen gibt und die Rolle der Null und der Eins austauschbar ist, gilt die Aussage von Satz 3.22 auch für Wörter mit einer speziellen Anzahl Nullen.

4 Zusammenhang zu Kompressionsfunktionen

Dieses Kapitel beschäftigt sich mit den Resultaten aus [FF17].
In diesem Kapitel geht es um Datenkompression. Die anderen Kapitel handeln im eigentlichen Sinne von Datendekompression, da man aus einem Wort x erstmal keine komprimierte Variante ebendieses Wortes ableiten kann, bzw. nicht einmal weiß, ob man das Wort überhaupt komprimieren kann. Wir haben in den Beweisen immer den Ansatz verfolgt, dass man eine komprimierte Variante einer bestimmten Länge hat und daraus das Wort wiederherstellen kann.
Dieses Kapitel betrachtet eine andere Herangehensweise. Es beschäftigt sich mit der Existenz von Kompressionsfunktionen mit gewissen Eigenschaften.

Im Folgenden sei U eine universelle Turingmaschine.

Definition 4.1. *Sei $n \in \mathbb{N}$. Eine berechenbare Funktion q heißt Kompressionsfunktion der Länge n, wenn*

$$U(q(w)) = w,$$

für alle Wörter w der Länge n gilt.
Eine berechenbare Funktion q heißt Kompressionsfunktion, wenn q eine Kompressionsfunktion für alle Längen ist.

Eine Kompressionsfunktion komprimiert also ein Wort w auf eine andere Form (bspw. eine kodierte Turingmaschine oder ein Programm, dass w kürzer darstellt), so dass eine universelle Turingmaschine dies bei Eingabe $q(w)$ umkehren kann und man wieder das Wort w erhält. Dementsprechend spielen Kompressionsfunktionen auch in der Analyse verlustfreier Datenkompression, wie es in dieser Arbeit der Fall ist, eine Rolle.

Beispiel 4.2. *Die triviale Kompressionsfunktion für ein Wort w gibt lediglich das Programm 'Print w' zurück.*

Bemerkung 4.3. *Jede Kompressionsfunktion q ist total. Außerdem gilt für alle Wörter x*

$$|q(x)| \geq K(x).$$

Dies kann man sich leicht vor Augen führen. Die Kolmogorov-Komplexität $K(x)$ ist genau das Minimum aller Programme, die ein spezielles Wort x komprimieren. Dann kann ein Programm, das alle Wörter der Länge $|x|$ komprimiert, das Wort x ebenfalls maximal so stark komprimieren, bzw. auf minimal so viele Zeichen reduzieren.

Bisher haben wir keine Untersuchungen bzgl. $K(f(x))$ für eine berechenbare Funktion f gemacht, daher wollen wir zuerst ein Resultat zeigen, dass uns $K(f(x))$ durch $K(x)$ abschätzen lässt.

Satz 4.4. *(Nach [LV08])*
Für jede berechenbare Funktion $f(x)$ gibt es eine Konstante c, sodass

$$K(f(x)) \leq K(x) + c$$

gilt.

Gilt diese Aussage, so gilt sie insbesondere auch für jede Kompressionsfunktion q.

Beweis. Sei U die universelle Turingmaschine, für die

$$K_U(x) = K(x)$$

gilt. Sei außerdem f berechenbar. Dann existiert eine Beschreibung $\langle f \rangle$ der Funktion f, so dass

$$U(\langle f \rangle x) = f(x).$$

Sei nun p das kürzeste Wort mit $U(p) = x$, also sei

$$K(x) = |p|.$$

Nun simmulieren wir U mit der Eingabe $\langle f \rangle p$. Dann gibt es eine Turingmaschine M, die bei der Eingabe $\langle f \rangle$ zuerst das Wort p in x überführt. Dann gilt

$$K(f(x)) = K_U(f(x)) \leq K_M(f(x)) + c_1 \leq |p| + |\langle f \rangle| + c_1 = K(x) + |\langle f \rangle| + c_1.$$

Für

$$c = c_1 + |\langle f \rangle|$$

ergibt sich das gewünschte Resultat. □

Wir wollen im Folgenden zeigen, dass es für jedes m eine Kompressionsfunktion q der Länge m gibt, die jedes Wort mit Kolmogorov-Komplexität maximal m

vollständig komprimiert. Wir brauchen lediglich m Bits für q, obwohl es q damit in m exponentiell viele Wörter komprimiert.
Außerdem werden wir sehen, dass für jedes $n > m$ ein Wort x der Länge $|x| = n$ und Kolmogorov-Komplexität $K(x) = m$ exisiert, so dass jede Kompressionsfunktion der Größe kleiner als m das Wort x nicht komprimieren kann.
Um diese Aussagen zu beweisen, brauchen wir jedoch vorher noch die Definition des Busybeavers nach [Rad61].

Definition 4.5. *Ein Busybeaver (engl. für fleißiger Bieber) ist eine Turingmaschine M, die folgende Regeln erfüllt:*

- *M bewegt den Kopf bei jedem Schritt entweder nach links oder rechts.*
- *Das Band von M ist zu Beginn mit Nullen gefüllt.*
- *M hält nach endlich vielen Schritten.*
- *M hat nach dem Halten eine maximale Anzahl von Einsen geschrieben.*

Daraus kann man die folgende Funktion herleiten:

Definition 4.6. *Die Busybeaver Funktion $BB(n)$ entspricht der Anzahl von Einsen, die ein Busybeaver mit n Zuständen schreibt.*

Nun wollen wir die erste Hauptaussage dieses Kapitel damit zeigen.

Theorem 4.7. *Für alle m existiert eine Kompressionsfunktion q mit $|q| \leq m + c$, so dass für alle Wörter x,*

- *$|q(x)| = K(x)$, falls $K(x) \leq m$*
- *$|q(x)| \leq |x| + c$, sonst.*

Das Theorem sagt also aus, dass es eine Kompressionsfunktion q gibt, die alle Wörter mit Kolmogorov-Komplexität kleiner als ein m optimal komprimiert. Alle Wörter, die eine größere Kolmogorov-Komplexität haben, werden mittels q auf eine Variante komprimiert, die kürzer ist als die Länge des Wortes addiert mit c.

Beweis. Wir nehmen an, m sei gegeben. Nun wollen wir für dieses m ein zugehöriges q erzeugen. Sei $t = BB(m)$ der Wert der Busybeaver Funktion. Sei p_m das lexikographisch letzte Programm, dass in t Schritten hält.
Nun suchen wir das lexikographisch erste Programm p, der Länge maximal $|z|$, mit

$$U(p) = z,$$

wobei U bei Eingabe p in t Schritten hält. Wenn p gefunden wird, so gebe es aus, wenn nicht gebe 'Print z' zurück.
Die Kompressionsfunktion q hängt von p_m ab und es gilt $|q| \leq m + c$.
Sei x ein festes Wort. Nun unterscheiden wir die Fälle $K(x) \leq x$ und $K(x) > x$:

- $K(x) \leq m$.
  Sei p das lexikographisch erste Programm der Länge $K(x)$ (in Zeichen: $|p| = K(x)$), mit
  $$U(p) = x.$$
  Nach der Definition von BB hält $U(p)$ in maximal t Schritten. Es gilt also $q(x) = p$ und damit $|q(x)| = K(x)$.

- $K(x) > m$.
  Entweder gibt $q(x)$ die triviale Kompressionsfunktion 'Print x' oder ein Programm der Länge maximal n zurück. In beiden Fällen erhalten wir
  $$|q| \leq m + c.$$

□

Damit folgt letztendlich, dass c eine genügend große Konstante ist, die sowohl unabhängig von x als auch von m ist.

Korollar 4.8. *Für alle m existiert eine Kompressionsfunktion q mit*

$$|q| \leq m + c,$$

so dass für alle x mit $|x| \geq m$ gilt

$$|q(x)| - K(x) \leq |x| - m + c$$

Das Korollar liefert also eine obere Schranke für die Abweichung der Länge der Komprimierung eines Wortes von seiner Kolmogorov-Komplexität. Je kleiner diese Abweichung ist, desto optimaler ist die Kompressionsfunktion für dieses Wort.

Beweis. Wir wollen das Korollar mithilfe des vorigen Theorems zeigen. Da das Theorem zwei verschiedene Aussagen, einmal für $K(x) \leq m$ und einmal für $K(x) > m$, trifft, müssen wir in diesem Korollar ebendiese Fälle unterscheiden.

- Für $K(x) \leq m$ erhalten wir mithilfe des Theorems, dass

$$|q(x)| = K(x)$$

gilt. Demnach ist die linke Seite der geforderten Ungleichung gleich Null. Bleibt also zu zeigen, dass

$$|q(x)| - K(x) = 0 \leq |x| - m + c$$

gilt. Nach Voraussetzung ist $|x| \geq m$. Dementsprechend ist $|x| - m \geq 0$. Danach ist die geforderte Ungleichung für alle Konstanten c erfüllt.

- Im zweiten Fall gelten

$$|q(x)| \leq |x| + c \text{ und } K(x) > m.$$

Durch Subtraktion mit $K(x)$ auf der linken Seite und m auf der rechten Seite erhalten wir

$$|q(x)| - K(x) < |x| + c - m.$$

Dies ist zulässig, da $K(x) > m$ ist und damit das $\leq$ zu $<$ wird. Wenn $<$ gilt, gilt also insbesondere auch

$$|q(x)| - K(x) \leq |x| - m + c.$$

□

Theorem 4.9. *Für alle m, n mit $0 \leq m \leq n$ existiert ein x sodass*

1. $|x| = n$
2. $K(x) \leq m + c\log(n)$
3. *Für alle Kompressionsfunktionen q der Länge n*

$$|q| \leq m - c\log(n) \text{ und } |q(x)| \geq n$$

4. *Für alle Kompressionsfunktionen q der Länge n*

$$|q| \leq m - c\log(n) \text{ und } |q(x)| - K(x) \geq n - m - c\log(n)$$

gilt.

Mit anderen Worten: Für alle Längen n gibt es Wörter x mit $K(x) \leq m + c\log(n)$, die durch keine Kompressionsfunktion der Länge n komprimiert werden können.

Beweis. Die vierte Aussage folgt aus der zweiten und der dritten Aussage. Aus der dritten Aussage erhalten wir direkt die erste Ungleichung

$$|q| \leq m - c\log(n).$$

Um die zweite Ungleichung zu erhalten, subtrahieren wir die Ungleichung aus der Aussage 2 von der zweiten Ungleichung der Aussage 3. Wir erhalten

$$|q(x)| - K(x) \geq n - (m + c\log(n)) = n - m - c\log(n).$$

Es bleibt noch zu zeigen, dass die ersten drei Aussagen gelten.
Sei A_s^l die Menge der Wörter y der Länge $|y| = l$, für die es kein Programm p mit der Länge $|p| < l$ gibt, so dass $U(p) = y$ und U dabei in s Schritten hält. Man kann einen kanonischen Index für A_s^l berechnen, mit s und l als Input.
A_s^l ist nicht die leere Menge, da auf jeden Fall die unkomprimierbaren Wörter der Länge l enthalten sind. Ist $s \geq BB(l)$, dann enthält A_s^l nur die unkomprimierbaren Wörter der Länge l.
Sei $m \leq n$ gegeben. Seien weiterhin $t = BB(m)$ und x das lexikographisch erste Wort in der Menge A_t^n. Es gilt

$$K(x) \leq m + c\log(n),$$

da wir x mit p_m erzeugen können. Hierbei benutzen wir p_m, m und n um t zu bestimmen.
Angenommen es gibt eine Kompressionsfunktion q der Länge n, so dass

$$|q| \leq m - c\log(n) \text{ und } |q(x)| < n.$$

Sei z das lexikographisch erste unkomprimierbare Wort der Länge m. Wir zeigen nun, wie man q, n und m benutzt, um z zu finden und werden das anschließend zu einem Widerspruch führen.
Sei nun t' das Maximum der Anzahl der Schritte die U braucht, bis sie bei Eingabe $q(y)$ hält, über alle Wörter y der Länge n. Dieses Maximum existiert, da nach Definition einer Kompressionsfunktion $U(q(y)) = y$ gilt, für alle Wörter y der Länge $|y| = n$.
Da $|q(x)| < n$ und $U(q(x)) = x$ gelten, ist die Anzahl $\bar{t}$ der benötigten Schritte von

U bei Eingabe $q(x)$ größer als $t = BB(m)$. Es gilt also

$$t' \geq \bar{t} > t = BB(m).$$

Wir können weder t noch $\bar{t}$ mithilfe von q, n und m berechnen, aber wir können t' berechnen.
Damit können wir auch die Menge $A_{t'}^m$ bestimmen. Diese Menge besteht genau aus den unkomprimierbaren Wörtern der Länge m. Das Wort z ist dann das lexikographisch letzte Wort dieser Menge.

□

5 Ausblick

Dieses Kapitel soll einen Ausblick geben, wofür man die Kolmogorov-Komplexität noch verwenden kann und verweist auf Anwendungen aus [LV08]

Die Kolmogorov-Komplexität als Maß für die Komprimierbarkeit eines binären Wortes ist nicht die einzige Anwendung, die die Kolmogorov-Komplexität besitzt. Ähnlich zu der in Kapitel 2 definierten c-Unkomprimierbarkeit kann man zufällige Folgen definieren und sogenannte Zufallszahlen untersuchen. Wie bereits in der Einleitung dieser Arbeit erwähnt, entsprechen hierbei die Zufallszahlen nicht dem Begriff des Zufalls aus der mathematischen Stochastik. Für den Zufallsbegriff den die Kolmogorov-Komplexität induziert, gibt es sogenannte Martin-Löf-Tests. Kommt durch einen solchen Test heraus, dass eine Zahl zufällig ist, so ist sie nicht komprimierbar.

Außerdem kann man mithilfe der Kolmogorov-Komplexität einige Sätze beweisen. So lässt sich auch ein alternativer Beweis für die Unendscheidbarkeit des Halteproblems finden, oder des Primzahlsatzes der aussagt, dass die Anzahl der Primzahlen in etwa so schnell wächst wie die Funktion $\frac{n}{\log n}$.

Für induktives Schließen in Bereichen außerhalb der Mathematik gibt es 'Ockhams Rasiermesser'. Dies sagt aus, dass wenn mehrere Theorien den gleichen Sachverhalt erklären, die einfachste Theorie auszuwählen ist. Es lässt sich auf die Kolmogorov-Komplexität zurückführen, in dem man

$$K(DATEN) + K(DATEN|THEORIE)$$

betrachtet und das Minimum über alle Theorien auswählt. Hieraus leitet sich die sogenannte 'Minimum Description Length' ab.

Des Weiteren kann man die Kolmogorov-Komplexität in Verbindng mit dem Entropiebegriff sowie dem Begriff des Informationsgehalts setzen. Dabei kann man dann entdecken, dass es für die Kolmogorov-Komplexität ein Analogon zur

Kettenregel der Entropie gibt. Für die Entropie gilt

$$H(x, y) = H(x) + H(y \mid x),$$

wohingegen für die Kolmogorov-Komplexität

$$K(x, y) = K(x) + K(y \mid x) + \mathcal{O}(\log(K(x, y)))$$

gilt.

Literaturverzeichnis

[AZ14] Aigner, Martin ; Ziegler, Günter M.: *Das BUCH der Beweise*. Springer Spektrum, 2014. – ISBN 3662444569

[FF17] Fenner, Stephen A. ; Fortnow, Lance: Compression Complexity. In: *CoRR* abs/1702.04779 (2017). http://arxiv.org/abs/1702.04779

[Goo97] Goos, Gerhard: *Vorlesungen über Informatik*. Springer Spektrum, 1997. – ISBN 978–3–642–59140–2

[Grü14] Grübel, Rudolf: *Stochastik*. Vorlesungsskript, 2014

[LG99] Lovász, László ; Gács, Peter: *Complexity of algorithms*. Lecture Notes, 1999

[LV08] Li, Ming ; Vitányi, Paul M.: *An Introduction to Kolmogorov Complexity and Its Applications (Texts in Computer Science)*. Springer, 2008. – ISBN 0387339981

[Rad61] Radó, Tibor: On Non-Computable Functions. (1961)

[Vol13] Vollmer, Heribert: *Grundlagen der theoretischen Informatik*. Vorlesungsskript, 2013

[Wat11] Watanabe, Osamu: *Kolmogorov Complexity and Computational Complexity* -. Wiesbaden : Springer Berlin Heidelberg, 2011. – ISBN 978–3–642–77737–0