MATHEMATIQ

Der Newsletter der MathSIG
(Interessensgruppe innerhalb der Mensa Österreich)

Ausgabe 3

http://www.hugi.scene.org/adok/mensa/mathsig/

Editorial

Liebe Leserinnen und Leser!

Dies ist die dritte Ausgabe von MATHEMATIQ, dem Newsletter der MathSIG. Die MathSIG wurde gegründet, um die spezifischen Interessen mathematisch hochbegabter Menschen zu fördern. In erster Linie soll sie sich also den Themengebieten Mathematik, Informatik, Physik und Philosophie widmen. Beiträge von Lesern sind herzlich willkommen. Wenn in ihnen mathematische Sonderzeichen vorkommen, bitte ich aber, sie zwecks möglichst einfacher und fehlerfreier Formatierung im TEX-Format einzusenden. Als Vorlage ist eine Fassung des jeweils aktuellen Newsletters im TEX-Format auf Anfrage bei mir erhältlich. Außer Artikeln sind natürlich auch Illustrationen für das Titelblatt willkommen. Die Rechte an diesen müssen aber eindeutig bei euch selbst liegen, Kopieren von Bildern aus dem Internet ist nicht erlaubt.

Diese Ausgabe ist wieder den formalen Sprachen gewidmet. Dazu passend habe ich als Einführung in die Komplexitätstheorie auch meinen Artikel über das *P-NP*-Problem übernommen, der einigen Lesern bereits aus TOPIQ 365 bekannt sein dürfte. Die hier wiedergegebene Fassung ist leicht überarbeitet und mathematisch präziser formuliert. Ich werde in Zukunft meine schon in TOPIQ veröffentlichten themenrelevanten Artikel nochmals bringen, um die durch MATHEMATIQ dargestellte Sammlung zu vervollständigen.

In diesem Sinne: Viel Spaß beim Lesen und Lernen!

Claus D. Volko, cdvolko@gmail.com

Formale Sprachen, Teil 2

Auch kontextfreie Sprachen können durch Automaten definiert werden. Man verwendet dazu so genannte Pushdown-Automaten. Diese arbeiten mit einem Stack, zu deutsch Stapelspeicher. Das ist in der Informatik eine relativ weit verbreitete Datenstruktur, die im Wesentlichen eben so funktioniert, wie ihr Name es sagt. Wenn man beispielsweise einen Stapel von Büchern anlegt, dann kann man immer ein weiteres Buch auf den Stapel legen, aber wenn man ein bestimmtes Buch haben will, muss man beim jeweils obersten Buch anfangen, die Bücher schrittweise abzutragen. Anders gesagt: Ein Stapel ist ein LIFO-Speicher (*last in, first out*). Ein Pushdown-Automat macht sich nun einen Stapel als zusätzliches Kriterium zunutze, ob ein bestimmter Zustandsübergang erlaubt ist: Bei jedem Übergang ist es möglich, das oberste Speicherelement aus dem Stapel zu holen und festzulegen, dass der Übergang nur zulässig ist, wenn dieses Speicherelement einen bestimmten Wert hat. Außerdem darf bei jedem Übergang ein neuer Wert auf den Stapel gelegt werden. Mit einem solchen Automaten lässt sich entscheiden, ob ein gegebenes Wort Element der kontextfreien Sprache ist, welche dieser Automat repräsentiert.

Dass ein solcher Automat in der Lage ist, eine kontextfreie Sprache abzubilden, lässt sich leicht zeigen: Wenn auf der rechten Seite einer Regel nur Literale stehen, wird der Stack nicht benötigt. Wenn rechts auch Verweise auf andere Regeln vorkommen, dann kann der Stack genutzt werden, um, sobald ein solcher Verweis erreicht wurde, abzuspeichern, an welcher Stelle der Automat nach dem Abarbeiten dieser Regel weitermachen soll. Kommt beispielsweise innerhalb der Regel A ein Verweis auf die Regel B vor, arbeitet der Automat das Eingabewort solange nach der Regel A ab, bis dieser Verweis erreicht wird. Dann speichert er auf dem Stack, wo er weitermachen muss, sobald die Abarbeitung der Regel B fertig ist, und arbeitet das Eingabewort nach der Regel B ab. Sobald die Abarbeitung nach der Regel B erledigt ist, sieht der Automat durch Zugriff auf den Stack nach, wo er fortsetzen muss. Nach der Abarbeitung der Regel A erkennt er dann, dass der Stack leer ist, und akzeptiert das Eingabewort.

Fehlt nur der Beweis, dass ein solcher Pushdown-Automat nur kontextfreie und nicht auch in der Chomsky-Hierarchie niedriger stehende Sprachen abarbeiten kann. Natürlich lässt sich zeigen, dass ein Pushdown-Automat nicht in der Lage ist, kontextsensitive Sprachen abzuarbeiten, die nicht zugleich kontextfreie Sprachen sind. Kontextsensitive Sprachen können durch Grammatiken definiert werden, in denen Regeln wie bei kontextfreien Sprachen vorkommen, zudem aber auch Regeln, in denen auch auf der linken Seite Literale vorkommen. Also Regeln wie etwa

$a\ A\ b \rightarrow b\ C\ d$.

Den Beweis überlasse ich den Lesern als Übungsaufgabe. Ein Tipp: Es hat damit zu tun, dass Automaten das Eingabewort sequenziell verarbeiten, also ein Literal nach dem anderen in exakt der Reihenfolge, in der diese Literale im Eingabewort vorkommen. Wieso kann das bei kontextsensitiven Sprachen ein Problem sein? Wäre

es theoretisch möglich, mit Pushdown-Automaten auch zu Literalen, die bereits abgearbeitet worden sind, zurückzuspringen? Wieso reicht das nicht aus, um kontextsensitive Sprachen durch Pushdown-Automaten zu definieren?

Eine Möglichkeit, zu bereits abgearbeiteten Literalen zurückzuspringen und dennoch die zusätzlich benötigten (Zustands-)Informationen abzuspeichern, die man beim Verarbeiten von kontextsensitiven Sprachen braucht, ist die Verwendung von Turing-Maschinen. Eine Turing-Maschine ist deutlich mächtiger als ein Automat. Sie kann verschiedene Zustände haben, die Eingabedaten sowohl von links nach rechts als auch in umgekehrter Richtung einlesen sowie die Eingabedaten überschreiben. Konkret wird eine Turing-Maschine genauso wie ein Automat durch verschiedene Zustände und deren Übergänge repräsentiert. Genau ein Zustand ist der Ausgangszustand und mindestens ein Zustand Endzustand. Das Eingabewort wird angenommen, wenn ein Endzustand erreicht wird. Welche Übergänge möglich sind, hängt einerseits vom aktuellen Zustand ab, andererseits aber auch von dem Literal, das sich an der aktuellen Stelle des Schreib-/Lesekopfs befindet. Jeder Übergang bestimmt nicht nur den Folgezustand, sondern auch, mit welchem Wert das aktuelle Eingabedatum überschrieben werden und in welche Richtung sich der Schreib-/Lesekopf bewegen sollte.

Durch Turingmaschinen lassen sich nicht nur kontextsensitive Sprachen beschreiben, sondern allgemeinere Mengen von Sprachen. Wenn man festlegt, dass die Turingmaschine das Eingabewort in jedem Fall entweder akzeptieren oder zurückweisen muss, aber nicht in einer Endlosschleife hängenbleiben darf, nennt man diese Turingmaschine auch Turing-Entscheider. Durch Turing-Entscheider repräsentierte Sprachen heißen rekursive Sprachen. Gestattet man, dass die Turingmaschine in einer Endlosschleife hängenbleiben darf, wenn ein Wort nicht zu der durch sie beschriebenen Sprache gehört, aber andernfalls das Wort akzeptieren muss, nennt man die auf diese Weise definierbare Menge von Sprachen rekursiv aufzählbar. Es gibt auch die Menge der co-rekursiv aufzählbaren Sprachen: Diese entsprechen Turingmaschinen, die ein Wort in jedem Fall zurückweisen, wenn es nicht in der Sprache enthalten ist, und die andernfalls entweder das Wort akzeptieren oder in einer Endlosschleife hängenbleiben müssen. Logischerweise stellt die Menge der rekursiven Sprachen die Schnittmenge der Menge der rekursiv aufzählbaren Sprachen und der Menge der co-rekursiv aufzählbaren Sprachen dar. Rekursiv aufzählbaren Sprachen entsprechende Entscheidungsprobleme werden übrigens auch semi-entscheidbar genannt.

Damit hätte ich die wichtigsten Mengen von formalen Sprachen genannt und definiert. In einem möglichen dritten Teil dieser Serie könnte ich noch die Pumping-Lemmata vorstellen, mit denen man beweisen kann, dass eine gegebene Sprache nicht kontextfrei oder zumindest nicht regulär ist.

Claus D. Volko, cdvolko@gmail.com

Errata

Der erste Teil des Artikels zu den formalen Sprachen aus MATHEMATIQ Ausgabe 2 enthielt einige kleinere Schnitzer. Erstens ist es nicht ganz korrekt, von Klassen von Sprachen zu sprechen, weil Klassen ja im strengen Wortsinn disjunkte Mengen sind, die meisten der in der Chomsky-Hierarchie vorkommenden Mengen von Sprachen aber Teilmengen von anderen in der Hierarchie vorkommenden Sprachen sind. Also darf man eigentlich nur von Mengen und nicht von Klassen sprechen, wenn man ganz korrekt sein will. Weiters habe ich mindestens einmal von der in der Hierarchie "nächsthöheren" Sprache gesprochen, dabei handelte es sich aber um die nächsttiefere Sprache, weil ja die höheren Hierarchiebenen Teilmengen der niedrigeren Hierarchieebenen sind und nicht umgekehrt. Das ist mir jedenfalls jetzt beim nochmaligen Lesen des Artikels aufgefallen. Aber es handelt sich natürlich nur um Kleinigkeiten, und im Wesentlichen sollte der Artikel stimmen.

Claus D. Volko, cdvolko@gmail.com

Das P-NP-Problem

Dies ist eines der schwierigsten noch ungelösten Probleme der Informatik. Es zählt zu den Millennium-Prize-Problemen, und die erste Person, die eine Lösung publiziert, wird vom Clay Mathematics Institute 1 Million Dollar bekommen.

P und *NP* sind Mengen von Entscheidungsproblemen, die eine bestimmte Komplexität haben. Umgangssprachlich nennt man *P* und *NP*, wenn auch nicht ganz korrekt, deswegen auch Komplexitätsklassen. Dies bedeutet, dass es einen Algorithmus gibt, der das jeweilige Entscheidungsproblem löst und dessen Laufzeit selbst im schlechtesten Fall ein gegebenes Limit nicht überschreiten wird. Egal mit welchen Eingabedaten man den Algorithmus ausführt, benötigt der Algorithmus, wenn das zu ihm gehörende Entscheidungsproblem in die Klasse *P* fällt, nur eine polynomielle Anzahl von Schritten in Bezug auf die Größe der Eingabedaten (also eine Konstante multipliziert mit der Größe der Eingabedaten hoch eine andere Konstante), um die Ausgabedaten zu berechnen. Im Gegensatz dazu bedeutet *NP*, dass es möglich ist, eine Kandidatenlösung in einer polynomiellen Zeit in Bezug auf die Größe der Lösung auf Korrektheit zu überprüfen.

Jeder Algorithmus, der in *P* ist, ist auch in *NP*. Hat man die Lösung für ein gegebenes Problem in *P*, kann man diese Lösung auch in einer polynomiellen Anzahl von Schritten verifizieren. Die Frage ist, ob es auch andersrum möglich sei. Die meisten Informatiker denken, nein. Sollte sich aber herausstellen, dass jedes Problem in *NP* auch in *P* ist, wäre das eine wahre Revolution, weil sie die Berechnung vieler Probleme deutlich beschleunigen würde.

Um zu beweisen, dass *P* und *NP* gleich sind, würde es reichen, es für ein einziges *NP*-vollständiges Problem zu beweisen. *NP*-vollständige Probleme stellen eine Teilmenge von *NP* dar, mit der Eigenschaft, dass sie zumindest so schwierig wie jedes andere Problem in NP sind. Ein Problem, für das diese Eigenschaft bewiesen wurde, ist das Erfüllbarkeitsproblem der Aussagenlogik (SAT, Satz von Cook und Levin). Um für ein anderes Problem zu zeigen, dass dieses ebenfalls *NP*-vollständig ist, reicht es aus zu beweisen, dass es in *NP* ist, und dass es zumindest so schwierig wie SAT ist. Letzteres kann erreicht werden, indem man einen Weg findet, SAT zu diesem Problem zu reduzieren, also einen Weg zu finden, wie ein Algorithmus für dieses neue Problem Instanzen von SAT lösen könnte. Bis jetzt wurde noch kein einziger polynomieller Algorithmus für ein *NP*-vollständiges Problem gefunden.

Es scheint realistischer zu sein, einen Beweis dafür zu finden, dass *P* ungleich *NP* ist, aber auch diese Suche gestaltet sich sehr schwierig. Eine Möglichkeit wäre es zu zeigen, dass es für ein bestimmtes *NP*-vollständiges Problem gar keinen Algorithmus mit polynomieller Laufzeit geben kann, aber wie sollte man dies bewerkstelligen? Zu beweisen, dass etwas nicht möglich ist, scheint viel schwieriger zu sein als das Gegenteil zu beweisen, und aus diesem Grund wird das *P*-*NP*-Problem immer noch als ungelöst betrachtet.

Claus D. Volko, cdvolko@gmail.com

Ressourcen

Beim Surfen im Internet sind mir einige Links untergekommen, die für Mathematikinteressierte wertvoll sein könnten.

- The 12 Most Controversial Facts In Mathematics
  `http://www.businessinsider.com/the-most-controversial-math-problems-2013-3`
- 9 More Super-Controversial Math Facts
  `http://www.businessinsider.com/controversial-math-problems-markov-chain-cantor-coin-flip-2013-5`
- Beal-Vermutung (Preisgeld von 1 Million Dollar!)
  `http://www.spiegel.de/wissenschaft/mensch/andrew-beal-lobt-eine-million-fuer-loesender-beal-vermutung-aus-a-904122.html`

Ich habe mir heute auch ein neues Buch zugelegt. Es heißt "After Gödel: Platonism and Rationalism in Mathematics and Logic". Der Autor ist Richard Tieszen, Professor der Philosophie an der San José State University in California. Das Ziel des Buchs ist es, "eine vertretbare Position zum platonischen Rationalismus in Mathematik und Logik zu finden". Sobald ich mit dem Lesen fertig sein werde, werde ich wohl den Inhalt dieses Buchs in MATHEMATIQ zusammenfassen.

Claus D. Volko, cdvolko@gmail.com