Christian Wagenknecht | Michael Hielscher

Formale Sprachen, abstrakte Automaten und Compiler

Lehr- und Arbeitsbuch für Grundstudium und Fortbildung

STUDIUM

VIEWEG+
TEUBNER

Christian Wagenknecht | Michael Hielscher

Formale Sprachen, abstrakte Automaten und Compiler

Christian Wagenknecht | Michael Hielscher

Formale Sprachen, abstrakte Automaten und Compiler

Lehr- und Arbeitsbuch für Grundstudium und Fortbildung

STUDIUM

VIEWEG+
TEUBNER

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
<http://dnb.d-nb.de> abrufbar.

Prof. Dr. rer. nat. Christian Wagenknecht
Geboren 1959 in Löbau/Sachsen. Studium der Mathematik und Physik (Lehramt) an der PH Dresden (heute in TU Dresden), Diplom in Informatik 1981. Lehrtätigkeit an Mittelschule und Gymnasium. Ab 1983 Aspirantur an der PH Dresden, Promotion in Informatik 1987. Wissenschaftlicher Mitarbeiter an der PH Dresden bis 1991 und anschließend an der PH Ludwigsburg. Seit 1993 Professor für Informatik an der Hochschule Zittau/Görlitz.
Arbeitsgebiete: Theoretische Informatik, Programmierparadigmen, Entwicklung von Web-Applikationen und XML-basierter CMS; einschl. fach- und mediendidaktischer Aspekte, E-Learning

Michael Hielscher
Geboren 1982 in Zittau/Sachsen. Studium der Informatik an der Hochschule Zittau/Görlitz, Diplom 2006, Master of Science 2008. Lehrtätigkeit an der Hochschule Zittau/Görlitz in theoretischer Informatik und Netzwerkprogrammierung. Mitarbeit in diversen Projekten und Workshops zur Lehrerfortbildung. Seit Oktober 2008 wissenschaftlicher Mitarbeiter an der PH Bern/Schweiz.
Arbeitsgebiete: Entwicklung von Desktop-/Web-Applikationen und Computerspielen, E-Learning und Web 2.0

1. Auflage 2009

Lektorat: Ulrich Sandten | Kerstin Hoffmann

Vieweg+Teubner ist Teil der Fachverlagsgruppe Springer Science+Business Media.
www.viewegteubner.de

Umschlaggestaltung: KünkelLopka Medienentwicklung, Heidelberg
Druck und buchbinderische Verarbeitung: STRAUSS GMBH, Mörlenbach
Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier.
Printed in Germany

ISBN 978-3-8348-0624-6

Lernen und (erst recht) Studieren erfordern Aktivität. Die beste Vorlesung bleibt wirkungslos, wenn sie als „Kinoeffekt“ verpufft.

Schnell sieht man ein, dass praktische Prozesse, die man beherrschen muss, *eingeübt* werden müssen. Abstrakte Denktechniken scheinen da pflegeleichter zu sein. Das Gegenteil ist der Fall! Die wirkliche Verinnerlichung abstrakter Inhalte erfordert höchste geistige *Aktivität*.

Deshalb haben wir dieses *Arbeitsbuch* zur theoretischen Informatik geschrieben. *Benutzen* Sie es! Verinnerlichen Sie die anfangs unappetitlichen Happen. Sie werden zunehmend spüren, dass abstrakte Denktechniken Freude bereiten können. Dies gilt auch für verwandte Denkaktivitäten außerhalb der theoretischen Informatik. Die Reise lohnt sich also. Aber: SIE sind der Kapitän. Wir reichen Ihnen lediglich Karte und Kompass. ...

Der Text basiert auf Vorlesungen zur theoretischen Informatik, die der erste Autor seit 1993 im Fachbereich Informatik an der Hochschule Zittau/Görlitz wiederholt gehalten hat. Sie wurden ständig modifiziert, verbessert und durch Begleitmaterial im Web ergänzt. Es wird lediglich Schulwissen aus der Mathematik vorausgesetzt.

Kernstück des Arbeitsbuches ist eine Lernumgebung, die wir seit 2004 entwickelt, erprobt, verbessert und erweitert haben. Sie heißt AtoCC (from automaton to compiler construction), `http://www.atocc.de`, und bietet die Möglichkeit, Grundlagen der Theorie formaler Sprachen und Automaten mit Bezug auf sehr praxisbezogene Anwendungen im automatisierten Übersetzerbau (compiler compiler/construction) am Computer einzuüben. Andere Teilgebiete der theoretischen Informatik, wie Berechenbarkeitstheorie und Komplexitätstheorie, werden hier nicht betrachtet.

Der Dank der Autoren gilt allen Kollegen und Studierenden, die in irgendeiner Form – durch individuelle Beiträge, gezielte Erprobungen und vielfältige Hinweise – direkt oder indirekt an der Entstehung dieses Materials beteiligt waren. Die kritischen Bemerkungen haben maßgeblich dazu beigetragen, dass der Text in den fast 15 Jahren seiner Entstehung zahlreiche didaktische Erfahrungen aufnehmen konnte.

Die trotz großer Sorgfalt nicht auszuschließenden Fehler bzw. Ungeschicklichkeiten sind allein den Autoren anzulasten. Bitte schreiben Sie uns eine E-Mail mit Ihren Anmerkungen. Dafür sind wir sehr dankbar!

`http://www.inf.hs-zigr.de/~wagenkn`
`http://www.michael-hielscher.de/`

Den Professoren
Immo O. Kerner und Herbert Löthe
in großer Dankbarkeit gewidmet

Inhaltsverzeichnis

1 Einleitung

1.1 Theoretische Informatik und ihre Anwendungen

Zur Weihnachtszeit sind Flughäfen oft überfüllt und restlos ausgelastet. Jeder möchte rasch nach Hause in den Kreis der Familie. Weihnachten 2004 ging dieser Wunsch für ca. 30000 Reisende nicht in Erfüllung: Eine US-amerikanische Fluggesellschaft musste über 1100 Flüge streichen. Grund war ein Softwarefehler. Ein enormer Image-Schaden für diese Firma war die Folge.

Der Zwischenfall führte dazu, dass in den Medien über die Nachteile von Automatisierung und Sicherheit im Informationszeitalter debattiert wurde. Wie war es aber zu dieser Katastrophe gekommen? Ein Mitarbeiter trug in ein Formular (vermutlich ein typisches Web-Formular) einen Wert ein, der dort nicht hingehörte. Dies führte zu einer Kettenreaktion, die sich sogar noch auf eine weitere bekannte Fluglinie auswirkte, da die Server der beiden Fluggesellschaften von ein und der selben Firma betrieben wurden. Es gab kein Sicherheitssystem für einen solchen Fall. Mit sehr geringem Aufwand hätten die Entwickler des Systems einen solchen Fehler ausschließen können, nämlich durch eine *Eingabevalidierung*. Hierfür sind Kenntnisse über *formale Sprachen und abstrakte Automaten* – ein Teilgebiet der theoretischen Informatik – notwendig. Zulässige Eingaben werden dabei (durch formale Grammatiken oder abstrakte Automaten bestimmten Typs) exakt beschrieben, unzulässige werden ausgefiltert bzw. abgewiesen.

Eingabevalidierung

formale Sprachen und abstrakte Automaten

Auch leistungsfähige Text-Editoren erfordern Theorie-Wissen, das sich nicht selten über Begrifflichkeiten darstellt. Sie arbeiten nämlich mit *regulären Ausdrücken* und damit nicht nur Zeichen-, sondern Struktur-orientiert. Der Begriff „regulärer Ausdruck“ wird hier mit der Erwartung benutzt, dass der Nutzer das zugrunde liegende theoretische Konzept kennt.

reguläre Ausdrücke

Theoriewissen ist also eine wichtige Voraussetzung für erfolgreiche praktische Tätigkeit auf hohem Niveau - ganz im Sinne des von dem deutsch-amerikanischen (Sozial-)Psychologen Kurt Lewin[1] (1890-1947) stammenden Ausspruchs "There is nothing as practical as a good theory". Theoretische Informatik ist heute mehr denn je eine Voraussetzung, um die Herausforderungen moderner IT-Anwendungen bewältigen zu können.

Die (immer noch) junge Wissenschaft „Informatik“ entwickelt sich mit rasan-

[1] Oft wird dieses Zitat auch Albert Einstein zugeschrieben.

ter Geschwindigkeit. Äußerlich betrifft dies vor allem den technologischen Fortschritt. Für jedermann sichtbar und oftmals spürbar haben sich viele Lebensbereiche durch den wachsenden IT-Einsatz fundamental verändert. Daten stehen unabhängig von Ort und Zeit zu akzeptablen Kosten zur Verfügung. Kommunikationsprozesse via Chat, Telefon, E-Mail, Web, Blog, Wiki usw. verlaufen synchron bzw. asynchron, oft „auf mehreren Kanälen“ gleichzeitig. Tiefgreifende Anwendungen der Informatik in Ingenieurdisziplinen, wie etwa dem Fahrzeugbau, führen zur Veränderung traditioneller Berufsfelder.

zunehmende Abstraktion

Fragt man nach dem Kern dieser Innovationen, der allen Ausprägungen gemein ist, stößt man zwangsläufig auf die *zunehmende Abstraktion*. Das Absehen von nicht-charakteristischen Aspekten des Betrachtungsgegenstandes (Dinge und Prozesse) ermöglicht den Übergang zu etwas Allgemeinerem und in gewissem Sinne Einfacherem. Wenn man sich nicht mehr um jede Einzelheit selbst kümmern muss, vereinfacht sich der Umgang damit. Noch bequemer wird es, wenn man den notwendigen Umgang mit dem Abstraktum an Dritte übertragen kann. Letzteres erfordert jedoch zusätzlich, den Arbeitsauftrag so zu spezifizieren, dass er von Dritten verstanden und (mit deren Fähigkeiten) umgesetzt werden kann.

deskriptives Arbeiten

Spätestens an dieser Stelle wird klar, dass *deskriptives Arbeiten* auf hoher Abstraktionsstufe für den Menschen wesentlich anspruchsvoller ist, als konkret-operationale Tätigkeiten auszuführen. Das Profil des Informatikers wandelt sich *vom Problemlöser zum Manager*, der die gewünschten Ergebnisse auf abstrakter Ebene *beschreibt* und einen Lösungsprozess *veranlasst*. Die auf Ausführungsebene erforderlichen Operationen, wie etwa das Sortieren von Listen oder Suchprozesse, sind längst zum Systembestandteil geworden. Sie stehen zur Verfügung und treten uns mit ihren Eigenschaften, wie etwa einer durch den Nutzer zu interpretierenden Effizienzangabe, also einem Qualitätsmerkmal, entgegen.

Komplexitätstheorie
Effizienz praktisch unlösbar

Die *Komplexitätstheorie*, ein weiteres Teilgebiet der theoretischen Informatik, befasst sich mit algorithmischer *Effizienz* (Ressourcen: Zeit und Speicher) und lotet objektiv die *praktischen* Grenzen algorithmischen Problemlösens aus. Leider sind gerade die aus wirtschaftlicher Sicht relevantesten Algorithmen, etwa in der Logistik, bestenfalls für „Baby-Anwendungen“ brauchbar. In echtem Anwendungskontext sind sie unbrauchbar. Man versucht dann (bisher meist erfolglos) effizientere Algorithmen zu finden bzw. geeignete Näherungsverfahren einzusetzen. Die Klassifizierung der praktisch schwersten Probleme wurde inzwischen so weit getrieben, dass aus der Angabe einer effizienten Lösung irgendeines Problems aus der schwersten Klasse, effiziente Lösungen für sämtliche (prinzipiell lösbare) Probleme resultieren. Die Suche nach einer solchen „Schlüssellösung“ verlief bisher ohne Erfolg.

Berechenbarkeitstheorie
absolut unlösbar

Auch die *Berechenbarkeitstheorie* gehört zur theoretischen Informatik. Sie vermittelt die Einsicht, dass es (sogar unendlich viele) formal definierbare Probleme gibt, die *absolut unlösbar* sind. Der Umgang mit unendlichen Mengen verschiedenster Art ist Gegenstand dieses Gebiets. Abstraktes Denken wird hier an Gegenständen

herausgebildet, die in der Praxis eher selten vorkommen, wohl aber in anderem Kontext angewandt werden.

Theorie der formalen Sprachen, Automatentheorie automatisierte Compiler-Konstruktion

Wir befassen uns im Folgenden mit der Theorie der formalen Sprachen und der Automatentheorie. Die entsprechenden Inhalte geben uns das Rüstzeug, um Sprachen – meist *unendliche* Mengen von Wörtern bzw. Sätzen – beschreiben zu können. Wie bei Programmiersprachen spielt hier die *Syntaxanalyse* eine besondere Rolle. Deshalb ist es auch sinnvoll, einen kleinen Ausflug auf das Gebiet der praktischen Informatik zu wagen, um zu sehen, wie man das Wissen zur *automatisierten Compiler-Konstruktion* erfolgreich einsetzen kann. Im Gegensatz zur Situation bei Programmiersprachen wird die Semantik sprachlicher Ausdrücke in der theoretischen Informatik nicht betrachtet.

Compilergenerator

Ende der 70er Jahre gehörten *Compilergeneratoren* (*compiler compiler*), die man häufig zur Herstellung von Übersetzern (Compilern) für Fachsprachen benutzt, zum Arbeitsgebiet einer geringen Zahl von Spezialisten. Dies lag vor allem daran, dass Compilergeneratoren ausschließlich auf Großrechnern zur Verfügung standen und sich auf diese Weise dem Tagesgeschäft entzogen. Seit der Verbreitung von Linux-Betriebssystem-Distribution für Heim- und Personalcomputer gehören Compilergeneratoren, wie etwa *yacc* (yet another compiler compiler) in Verbindung mit *lex* (lexical analysis), zum Softwarebasispaket und können für die verbreiteten Betriebssysteme aus dem Internet beschafft werden. Mit wenigen Handgriffen steht damit ein Werkzeug zur täglichen unmittelbaren Verfügung, das wesentlich abstraktere Anforderungen an den Informatiker stellt, als es die „Compilerprogrammierung vom leeren Blatt“ vermag. Die Sprache, für die der Compiler hergestellt werden soll, muss geeignet *beschrieben* werden. Die eigentliche Arbeit erledigt dann das Generator-Programm. So soll es sein! Ohne Kenntnisse aus der Theorie der formalen Sprachen und der Automatentheorie sind solche Werkzeuge nicht zielführend nutzbar.

yacc

lex

Beschreibung der Zielsprache

Abstrakte Automaten Computerspiele

Abstrakte Automaten finden außerdem vielfältige Anwendungen zur Modellierung von Zustandsübergangsprozessen in diversen Bereichen. So kann man beispielsweise Computerspiele (adventure games) mit Automaten bestimmten Typs modellieren. Auch die Robotik, Neuroinformatik und Künstliche Intelligenz verwenden Automatenmodelle. Denkt man an Getränkeautomaten oder ähnliches, werden hingegen Automaten benötigt, die eine Ausgabe[2] liefern. Auch zur Modellierung des Interaktionsverhaltens bei modernen Web-Anwendungen können Automaten (mit gewissen Einschränkungen) verwendet werden.

Automaten mit Ausgabe; Web-Anwendungen

Bei allen Anwendungen der theoretischen Informatik, die wir auch in einigen der folgenden Kapitel ganz hoch halten wollen, bleibt es ein Wissengebiet, das der Mathematik sehr nahe steht. Insofern ist es auch notwendig, abstrakten Denktechniken anwendungsfrei zu folgen und die mathematische Fachsprache gepflegt einzusetzen.

[2] ggf. zur Geldrückgabe, nicht zur Kaffeeausgabe

1.2 AtoCC - unsere Lernumgebung

AtoCC

AtoCC steht für „vom Automaten zur Compilerkonstruktion“ (from automaton to compiler construction). Seit 2004 arbeiten wir intensiv an dieser Software, die nicht nur für Windows-, sondern auch für Linux- und Mac-Betriebssysteme zur Verfügung steht und kostenlos aus dem Internet bezogen werden kann. Die Adresse lautet

`http://www.atocc.de`.

Neben der Software und den in diesem Buch an diversen Stellen verwendeten Arbeitsdateien finden sich hier zahlreiche didaktische Materialien und Tutorials, auch Videos.

Die Installation der Software geschieht vollautomatisch in wenigen Sekunden[3]. Bei Internetzugang stehen eine Online-update- und eine Online-Hilfefunktion zur Verfügung.

Während der Name AtoCC mit o.g. Langform den ursprünglichen Themenkreis reflektiert, hat sich dessen Funktionsumfang inzwischen auf den Bereich der formalen Sprachen ausgeweitet. Es ist nun möglich, die für das Gebiet der formalen Sprachen und abstrakten Automaten typische Themenbreite und ausgewählte Inhalte des Compilerbaus mit AtoCC zu behandeln. Damit steht eine konsistente Lernumgebung zur Verfügung, die den Lernenden (Studierende der Informatik aller Hochschulformen im Grundlagenstudium, Studierende von Informatik-Anwendungsfächern, Lehrpersonen und Schüler der gymnasialen Oberstufe) bei intuitivem Bedienkontext in aufeinander abgestimmten Modulen agieren lassen. Auch in der Hand des Lehrenden kann AtoCC wertvolle Unterstützung bei der Wissensvermittlung und bei der Herstellung von Lehrmaterialien bzw. einschlägiger Publikationen leisten.

Lernende

Immer wenn in diesem Buch das folgende Symbol am Rand auftaucht, handelt es sich um eine Computerübung, die unter Verwendung von AtoCC durchgeführt werden soll.

AtoCC wurde auf nationalen und internationen Tagungen sowie in teilweise mehrtägigen Workshops für Lehrpersonen vorgestellt und in mehrjährigen Erprobungen mit Studierenden kontinuierlich überarbeitet. Die Beurteilung des Resultats überlassen wir Ihnen.

[3] Das ist keines der üblichen hohlen Versprechen!

2 Struktur von Programmen

2.1 Sprache, Syntax, Semantik und Pragmatik

Stößt man auf das Wort „Sprache“, wie im Titel dieses Buches, so denkt man wohl zuerst an die „natürliche Sprache“, d.h. an unsere Muttersprache oder eine Fremdsprache. Eine solche Sprache benutzen wir täglich. Wir verfügen über einen mehr oder weniger breiten Wortschatz und bilden *Sätze* durch *Aneinanderreihung* von *Worten*.

Regeln
Grammatik
Syntax
Satzanalyse

Für den *Satzbau* gibt es *Regeln*, die durch die *Grammatik* der entsprechenden Sprache definiert sind. Man nennt das die *Syntax* einer Sprache. Im Allgemeinen wenden wir diese Regeln intuitiv an. Wenigstens bei Fremdsprachen kann das allerdings schon mal schief gehen: Dann fördert die *Satzanalyse* Regelverstöße ans Tageslicht.

Semantik

Die Kommunikation, zu deren Zweck wir die natürliche Sprache im Allgemeinen einsetzen, wird durch einen fehlerhafte Satzbau gestört. Sätze mit geringen Syntaxfehlern können für den Kommunikationspartner dennoch verständlich bleiben. Wovon hängt das ab? Wenn es um die Verständlichkeit geht, kommt die *Semantik* ins Spiel. Sie steht für die *Bedeutung* eines Wortes bzw. Satzes. Sie kann unter anderem von Stimmungen, sozialen Komponenten, kulturellen Einflüssen oder weltanschaulichen Positionen abhängen. Die Semantik des „gesprochenen Wortes“ kann nachhaltig durch Mimik und Gestik beeinflusst werden. Die moderne Neurobiologie untersucht u.a. die Beeinflussung des Gehirns durch Stimmmodulation: Wenn Sie jemanden mit dem Satz „Ich liebe Dich.“ beschimpfen, wird der Empfänger starke Zweifel am eigentlichen Bedeutungsgehalt dieser Aussage haben. Was man mit Sprache bei Menschen bewirken kann, ist ein Untersuchungsgegenstand der *Pragmatik*.

Pragmatik

Semantische Mehrdeutigkeiten

Selbst bei syntaktisch korrektem Gebrauch einer natürlichen Sprache können semantische *Mehrdeutigkeiten* auftreten. Es gibt Personen, die diese Eigenschaft gezielt einsetzen, um etwas „durch die Blume“ zu sagen oder um liebevolle sprachliche Seitenhiebe auszuteilen: „Verena kaufte noch eine Vase. Sie war wieder einmal blau.“ Worauf bezieht sich der zweite Satz?

Semiotik
Linguistik

Syntax (bzw. Syntaktik), Semantik und Pragmatik sind (überlappende) Teilbereiche der *Semiotik*, die als die allgemeine Lehre von den Zeichen, Zeichensystemen und Zeichenprozessen gilt. Die *Linguistik* ist die Wissenschaft, die sich mit Sprache beschäftigt.

Programme

Auch für die Informatik ist der Sprachbegriff fundamental: Es gibt eine riesige Anzahl von Programmier- und Fachsprachen, in denen *Programme* im Allgemeinen als Texte geschrieben werden, deren Abarbeitung einen vorgegebenen Zweck erfüllen oder eben fehlerhaft sind. Bei genauerem Hinsehen stellt man fest, dass Programme eine weitere sprachbezogene Aufgabe wahrnehmen können: Sie verarbeiten Eingaben zu Ausgaben und reichen letztere ggf. an einen oder mehrere Empfänger weiter. Im Allgemeinen kommen sie weder mit semantischen noch mit syntaktischen Mehrdeutigkeiten zurecht.

Kommunikation zwischen Programmen

Protokoll

Es geht aber nicht nur um Programmiersprachen und *Übersetzungen* entsprechender Texte in lauffähige Programme, sondern eben auch um die Kommunikation zwischen Programmen. In der Tat können sich zwei Programme „unterhalten“: Der Browser „spricht“ mit dem Webserver, indem er z.B. ein `html`-Dokument anfordert (request). Der Server antwortet (response) entsprechend. Dabei wird eine sehr einfache Sprache verwendet, die man als ein *Protokoll* bezeichnet. Auf höherem Niveau kommunizieren Systeme mit *XML-Sprachen*. Auf diese Weise lässt sich sogar die traditionelle Softwarelandschaft zu einer Service-orientierten Architektur (SOA) umgestalten, was im Moment ein sehr populäres Thema ist. Sprache ist also ein tragender Begriff in der Informatik.

formale Sprache

Die oben herausgestellten Aspekte (Syntax, Semantik, Pragmatik) von Sprache sind Gegenstand verschiedener Teilgebiete der Informatik. Während sich die praktische Informatik (*Compilerbau*) mit Programmiersprachen (sowie deren Übersetzung und Interpretation) beschäftigt, beschränkt sich die theoretische Informatik auf die Untersuchung *formaler Sprachen*.

Programmiersprachen sind künstliche, also keine natürlichen, Sprachen. Das ist klar. Aber worin besteht der Unterschied zwischen Programmiersprachen und formalen Sprachen? Die folgende „Gleichung“ bringt es auf den Punkt:

Programmiersprache = Syntax + Semantik + Pragmatik.

Die Theorie der formalen Sprachen befasst sich also nur mit der Syntax und stellt Fragen zur Syntaxanalyse für Sprachen und Sprachklassen, um festzustellen, ob zur Erzeugung eines vorgelegten Satzes (hier Wort genannt) ausschließlich die grammatikalischen Regeln der betrachteten Sprache angewandt wurden. Sprachübersetzung (Compilerbau) greift einige Ergebnisse der theoretischen Informatik auf und thematisiert zusätzlich Semantik, Pragmatik und die Effizienz der Analyseverfahren. Damit kein falscher Eindruck entsteht: Auch zur Definition der Semantik von Programmiersprachen wird in der praktischen Informatik sehr viel Mathematik eingesetzt.

Didaktischer Hinweis 2.1
Damit haben wir ein intuitives Verständnis für die Bedeutung des Sprachbegriffs und seiner Hauptaspekte geschaffen. Alltagserfahrungen im Umgang mit natürlichen Sprachen wurden begrenzt auf künstliche Sprachen übertragen und den entsprechenden Teilgebieten der Informatik zugeordnet. Der Untersuchungsgegenstand der Theorie der formalen Sprachen wurde herausgearbeitet.