LEHRBUCH

Hans-Joachim Böckenhauer
Juraj Hromkovič

Formale Sprachen

Endliche Automaten, Grammatiken, lexikalische und syntaktische Analyse

Springer Vieweg

Formale Sprachen

Hans-Joachim Böckenhauer · Juraj Hromkovič

Formale Sprachen

Endliche Automaten, Grammatiken, lexikalische und syntaktische Analyse

Springer Vieweg

Dr. Hans-Joachim Böckenhauer
ETH Zürich
Zürich, Schweiz

Dr. Juraj Hromkovič
ETH Zürich
Zürich, Schweiz

ISBN 978-3-658-00724-9
DOI 10.1007/978-3-658-00725-6

ISBN 978-3-658-00725-6 (eBook)

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

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier.

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media
www.springer-vieweg.de

Vorwort

Dieses Lehrbuch ist der Theorie der formalen Sprachen gewidmet, die eines der klassischen Kernthemen der Informatik ist. Diese Theorie gibt uns die Möglichkeit, Probleme der Informationsverarbeitung, ihre Lösungen und die Wege zu diesen Lösungen exakt zu beschreiben. Die Theorie der formalen Sprachen bildet somit eine Basis für viele theoretische und angewandte Gebiete der Informatik.

In diesem Buch fokussieren wir hauptsächlich auf endliche Automaten und kontextfreie Grammatiken, mit denen man eine beliebige Programmiersprache exakt beschreiben kann. Damit bieten wir einen Einstieg in den Compilerbau, nämlich in die ersten zwei Phasen eines Compilers – die lexikalische und die syntaktische Analyse.

Wie auch in den anderen Lehrbüchern dieser Reihe stellen wir die Erklärung grundlegender Konzepte und Fachbegriffe in den Mittelpunkt, und nicht die Vermittlung der neuesten, oftmals zu technischen, Resultate und Errungenschaften. Dieses Lehrbuch ist in zwei Unterrichtsmodule unterteilt. Das Modul „Endliche Automaten und lexikalische Analyse“ stellt die endlichen Automaten als die einfachste Klasse von speziellen Algorithmen vor. Dabei geht es nicht nur darum, das einfachste Modell von Berechnungen zu präsentieren und zu zeigen, wie man Berechnungen darstellen und untersuchen kann. Mit Hilfe der endlichen Automaten sprechen wir fundamentale Konzepte der Informatik an, wie den modularen Entwurf komplexer Systeme, Verifikation und Testen, sowie Beweise der Nichtexistenz von speziellen Algorithmen zur Lösung konkreter Aufgaben. Neben der Anwendung endlicher Automaten zur Ampelsteuerung und Steuerung von Fahrstühlen zeigt dieses Modul im letzten Teil, wie man endliche Automaten für die lexikalische Analyse von einfachen Programmiersprachen verwenden kann. Mit diesem Teil wird eine Brücke zum zweiten Modul „Grammatiken und syntaktische Analyse“ geschlagen. In diesem Modul wird der Formalismus der Grammatiken eingeführt, mit dem man unter anderem die Programmiersprachen genau beschreiben kann. Mittels dieses Werkzeugs ist es möglich, effizient zu überprüfen, ob ein Text einem korrekt geschriebenen Programm in einer gegebenen Programmiersprache entspricht. Diese Überprüfung nennt man syntaktische Analyse, sie ist Bestandteil jedes Compilers. Die Hauptziele hier sind der Entwurf von kontextfreien Grammatiken zur Beschreibung von Programmiersprachen und die Herleitung des bekannten CYK-Algorithmus zur syntaktischen Analyse.

Wie auch alle vorherigen Lehrbücher dieser Reihe ist dieses Lehrbuch auf dem Konzept der Leitprogramme aufgebaut und somit zum Selbststudium geeignet. Die Idee ist dabei, jeden beliebigen Umfang der Auseinandersetzung mit dem Stoff mit individueller Geschwindigkeit zu ermöglichen, mit dem Ziel, das voll-

ständige Verständnis des Stoffs zu erreichen. Wir haben versucht, alle Wege zu neuem Wissen in kleinstmögliche Schritte aufzuteilen und jeden Schritt durch hinreichend viele Aufgaben zu festigen. Die Beispiele und Musterlösungen erklären noch einmal zusätzlich unsere Überlegungen bei der Suche nach Lösungen für die gegebene Aufgabenstellung. Damit sollte das Lehrbuch für Gymnasiasten sowie Studienanfänger an den Hochschulen geeignet sein. Besonders bestimmt ist es für die Lehramtsstudierenden, die sich mit dem Erlernen der Unterrichtsvorbereitung auseinandersetzen.

Hilfreiche Unterstützung anderer hat zur Entstehung dieses Lehrbuches wesentlich beigetragen. Besonderer Dank gilt Karin Freiermuth, Roman Gächter, Stephan Gerhard, Lucia Keller, Dennis Komm, Andre Macejko, Jela Sherlak, Jasmin Smula, Andreas Sprock, Ute Sprock und Björn Steffen für sorgfältiges Korrekturlesen, zahlreiche Verbesserungsvorschläge und umfangreiche Hilfe bei der Umsetzung des Skriptes in LaTeX. Wir möchten uns sehr bei Karin Freiermuth, Lucia Keller und Björn Steffen dafür bedanken, dass sie uns mit viel Enthusiasmus beim Testen der Unterrichtsunterlagen in der schulischen Praxis begleitet haben oder einige Lektionen selbstständig unterrichtet haben.

Genauso herzlich danken wir den Lehrpersonen Pater Paul (Hermann-Josef-Kolleg, Steinfeld), Uwe Bettscheider (INDA Gymnasium Aachen), Hansruedi Müller (Schweizerische Alpine Mittelschule Davos), Josef Vogt (Kantonsschule Sargans), Meike Akveld, Stefan Meier und Pietro Gilardi (Mathematisch-Naturwissenschaftliches Gymnasium Rämibühl, Zürich) und Harald Pierhöfer (Kantonsschule Limattal, Urdorf), die es uns ermöglicht haben, in einigen Klassen kürzere oder längere Unterrichtssequenzen zu testen oder sie sogar selbst getestet und uns ihre Erfahrungen mitgeteilt haben. Ein besonderer Dank geht auch an die Schulleitungen der Schulen, die uns für das Testen unserer Module die Türen geöffnet haben.

Für die gute Zusammenarbeit bedanken wir uns bei dem Springer Vieweg Verlag.

Wir wünschen allen Leserinnen und Lesern viel Vergnügen mit diesem Buch und viel Erfolg bei ihrer Wissenserweiterung.

Zürich, im September 2012

Hans-Joachim Böckenhauer
Juraj Hromkovič

Inhaltsverzeichnis

Modul I

Endliche Automaten und lexikalische Analyse

Zielsetzung

Endliche Automaten sind überall. Bankautomaten, Straßenkreuzungen mit Ampeln, Getränkeautomaten, Fußgängerampeln, Fahrstühle – alle werden durch endliche Automaten gesteuert. Auch ein Teil eines Compilers ist ein endlicher Automat. In diesem Modul kann man erfahren, wie solche Steuerungsmechanismen in Form von Automaten entworfen werden können. Dabei lernt man nicht nur, Automaten auf systematische Weise zu entwerfen, sondern auch die entworfenen Produkte hinsichtlich ihrer Korrektheit zu überprüfen und über das Erfüllen von Praxisanforderungen und die dadurch verursachten Produktionskosten nachzudenken.

Endliche Automaten sind das einfachste Berechnungsmodell, das man in der Informatik betrachtet. In abstrakter mathematischer Darstellung entsprechen endliche Automaten speziellen Programmen, die konkrete Entscheidungsprobleme lösen und dabei keine Variablen benutzen. Endliche Automaten arbeiten in Echtzeit, denn sie lesen entweder eine Eingabe nur einmal von links nach rechts oder empfangen externe Signale nur einmal hintereinander. Das Resultat steht sofort nach dem Lesen des letzten Buchstabens oder nach dem Empfang des letzten Signals fest.

Das erste Hauptziel dieses Moduls ist das Erlernen einiger grundlegender Methoden für den Entwurf endlicher Automaten in zwei Schritten. Zuerst lernen wir, relativ einfache Automaten zu erzeugen, indem jedem Zustand eines Automaten eine Bedeutung zugeordnet wird. Für Aufgaben, die durch komplexere Bedingungen formuliert werden, entwickeln wir einen modularen Ansatz mit strukturiertem Vorgehen. Er benutzt einfache Automaten als Bausteine, um einen komplexeren endlichen Automaten mit den gewünschten Eigenschaften zu erzeugen.

Der Grund dafür, endliche Automaten zu betrachten, ist nicht nur der Automatenentwurf für Aufgabenstellungen aus der Praxis und der Einstieg in die Automatentheorie. Wir nutzten endliche Automaten auch für didaktische Zwecke, um auf einfache und anschauliche Weise die Modellierung von Berechnungen zu erläutern. Dazu führen wir einige Grundbegriffe der Informatik wie Konfiguration, Berechnungsschritt, Berechnung, Verifikation und Simulation ein.

Wir lernen in diesem Modul, wie man eine Teilklasse von Algorithmen (Programmen) formal und dabei anschaulich modellieren und untersuchen kann. Wir

werden damit verstehen, wie man einen Automaten auf Korrektheit überprüfen kann. Gleichzeitig üben wir dabei Induktionsbeweise. Neben dem ersten Kontakt mit den oben erwähnten Grundbegriffen lernen wir auch, einen Beweis zu führen, der zeigt, dass eine konkrete Aufgabe mit einer gegebenen Teilklasse von Algorithmen nicht lösbar ist. Gezeigt wird hier zum Beispiel, dass gewisse Aufgabenstellungen von keinem endlichen Automaten gelöst werden können und dass jeder Automat, der eine bestimmte Aufgabe löst, eine gewisse Mindestgröße haben muss.

Das zweite Hauptziel dieses Unterrichtsmoduls ist es, zu zeigen, wie man endliche Automaten für die lexikalische Analyse von Programmen verwenden kann. Damit lernt man, die erste Phase des Compilerbaus zu verstehen. Zu diesem Zweck werden die endlichen Automaten zu sogenannten Transducern, also Automaten mit Ausgabe, verallgemeinert. Die Transducer werden dann verwendet, um Algorithmen für die lexikalische Analyse für einfache Programmiersprachen zu entwerfen.

Hinweis für die Lehrperson Das Minimalziel dieses Moduls sollte sein, mit ihm den Automatenentwurf zu unterrichten und die Anwendung der Automatentheorie für die lexikalische Analyse im Compilerbau zu erläutern. Verwenden Sie dazu Kapitel 1 bis 3, 6 und 8. Kapitel 1 ist eine terminologische Vorbereitung. Kapitel 2 zeigt, wie man endliche Automaten modellieren und darstellen kann. Kapitel 3 und Kapitel 6 beinhalten zwei systematische Methoden zum Automatenentwurf. Kapitel 8 ist der lexikalischen Analyse gewidmet. Dieser Teil ist auch sehr konkret und konstruktiv und sollte der ganzen Klasse gut zugänglich sein. Kapitel 4 ist eine gute, optionale Erweiterung des minimalen Ziels. Die Besonderheit liegt hier darin, dass hiermit auch das Modellieren von realen Problemsituationen geübt wird und weniger mathematische Formalismen benötigt werden. So wirkt dieser Teil entspannend und erfrischend. Zudem wechselt man die Unterrichtsmethode und geht zu selbstständigeren Projekten und ihrer Präsentation über.

Eine weitere optionale Erweiterung ist Kapitel 5. Die Vorteile liegen einerseits in der Vertiefung der Fähigkeit, Induktionsbeweise zu führen und dadurch die mathematische Denkweise zu stärken. Andererseits wird deutlich, wie wichtig es ist, die entworfenen technischen Produkte auf ihre korrekte Arbeitsweise zu überprüfen und die entsprechende Methodik dafür kann erlernt werden.

Kapitel 7 ist der schwierigste Teil und macht einen weiteren Vertiefungsschritt in Richtung korrekter, mathematischer Argumentation. Hier geht es um Nichtexistenzbeweise. Zuerst zeigen wir, dass für manche Aufgaben keine kleinen, sondern nur große Automaten existieren. Danach stellen wir eine Beweismethode vor, die uns hilft zu zeigen, dass konkrete Aufgaben mit keinem Automaten lösbar sind. Zielsetzungen dieser Art gehören selten zum Gymnasialstoff. Die endlichen Automaten sind aber relativ einfach und bieten uns damit einen verständlichen Einstieg in die Beweisführung über die Existenz von Objekten mit gegebenen Eigenschaften. Dabei werden wieder direkte und indirekte Beweise geführt die Argumentation über allgemeine Aussagen im Sinne von unendlich vielen Fällen lässt sich nicht umgehen. Dieser Teil ist für hochmotivierte Schüler gut zu

bewältigen. Es besteht auch die Möglichkeit, nur den ersten Teil der Kapitel 7 zu benutzen, mit dem man die Mindestgröße von Automaten für gegebene Sprachen beweisen kann. Dort betrachtet man nicht das Unendliche und erfahrungsgemäß ist der Lernstoff für die Klasse nicht viel schwieriger als der Automatenentwurf.

Kapitel 1

Alphabete, Wörter und Sprachen

Im Allgemeinen verarbeitet jeder Rechner Texte, die man als Folge von Buchstaben aus einem bestimmten Alphabet ansieht. Die Eingaben sind Texte, die konkrete Probleminstanzen (Aufgabenstellungen) beschreiben. Auch die Ausgaben werden als Texte dargestellt. Ein Rechner arbeitet ebenfalls mit Texten, weil alle Speicherinhalte (Registerinhalte) als Folgen der Buchstaben 0 und 1 angesehen werden können und alle Rechenbefehle diese Bitfolgen in neue Bitfolgen umwandeln. Das Ziel dieses Abschnitts ist es, die Fachterminologie für die textliche (symbolische) Darstellung der Information festzulegen, um am Ende die Informationsverarbeitung als eine Form der Textverarbeitung anzusehen.

Wir wollen die Daten und die betrachteten Objekte durch Symbolfolgen darstellen. Genau wie bei der Entwicklung einer natürlichen Schriftsprache fängt man mit der Festlegung von Symbolen an, die die Atome unserer Datenrepräsentation sind. Im Folgenden bezeichnet $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ die Menge der natürlichen Zahlen.

Definition 1.1 *Eine endliche, nichtleere Menge Σ heißt **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben (Zeichen, Symbole)** genannt.*

Definition 1.1 besagt nichts anderes, als dass man das Alphabet frei wählen darf. Die einzigen Restriktionen sind, dass man mindestens einen Buchstaben im Alphabet haben muss und man nur endlich viele unterschiedliche Buchstaben nehmen darf. Dies entspricht auch der Situation in einer natürlichen Schriftsprache. Daher ist zum Beispiel die einelementige Menge $\{0\}$ ein Alphabet, aber die Menge $\mathbb{N}$ ist wegen ihrer unendlichen Mächtigkeit kein Alphabet. Deswegen sind für uns Zahlen keine Symbole, sondern Folgen von Symbolen, die wir Ziffern nennen.

Einige der hier am häufigsten benutzten Alphabete sind:

- $\Sigma_{\text{Bool}} = \{0, 1\}$ ist das Boolesche Alphabet (benannt nach George Boole (1815 – 1864), dem Begründer der modernen mathematischen Logik), mit dem die Rechner arbeiten.

H.-J. Böckenhauer, J. Hromkovič, *Formale Sprachen*,
DOI 10.1007/978-3-658-00725-6_1, © Springer Fachmedien Wiesbaden 2013

- $\Sigma_{\text{lat}} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$ ist das Alphabet der kleinen lateinischen Buchstaben.
- $\Sigma_{\text{dez}} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist das Alphabet der Ziffern, die wir zur dezimalen Darstellung von Zahlen verwenden können.
- $\Sigma_{\text{griech}} = \{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda, \mu, \nu, \xi, o, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi, \omega\}$ ist das Alphabet der kleinen griechischen Buchstaben.
- $\Sigma_{\text{Klam}} = \{\langle, \rangle, [,], (,)\}$ ist das Alphabet der Klammern.
- $\Sigma_{\text{Geo}} = \{\triangle, \square, \circ\}$.

Aufgabe 1.1 Welche der folgenden Mengen sind Alphabete? Begründe deine Antwort.

(a) $\mathbb{N}_{\text{ger}} = \{0, 2, 4, \ldots\}$ als die Menge der geraden natürlichen Zahlen,

(b) $\{a, b, c, A, B, C\}$,

(c) $\{1, 2, 3, a, b, \triangle, \heartsuit\}$,

(d) $\emptyset$.

Im Folgenden definieren wir den Begriff „Wort“ als eine Folge von Buchstaben aus einem gegebenen Alphabet. Der Begriff „Wort“ entspricht in der Fachsprache der Informatik einem beliebigen Text und nicht nur der Bedeutung von „Wort“ in der natürlichen Sprache.

Definition 1.2 *Sei Σ ein Alphabet. Ein* ***Wort über Σ*** *ist eine endliche (eventuell leere) Folge von Buchstaben aus Σ. Das* ***leere Wort λ*** *ist die leere Buchstabenfolge.*

Halten wir fest: Der Fachbegriff „Wort“ hat ohne Bezug zu einem Alphabet keine Bedeutung. Was wollen wir damit sagen? Wenn man über Wörter spricht, muss man zuerst das Alphabet festlegen und dann darf man von „Wörtern über dem Alphabet“ sprechen. Zum Beispiel ist $01ab1a$ ein Wort über $\{0, 1, a, b\}$, aber kein Wort über $\{0, 1\}$. Das leere Wort λ ist ein Wort über jedem möglichen Alphabet.

In der Mathematik schreibt man eine Folge von Elementen einer Menge üblicherweise, indem man die Elemente mit Kommata trennt, also bezeichnet zum Beispiel $0, 1, 0, 0, a, b, 1$ ein Wort über dem Alphabet $\{0, 1, a, b\}$. Hier verzichten wir auf die Kommata und schreiben anstatt $0, 1, 0, 0, 1, 1$ einfach nur 010011, wie wir es auch mit Wörtern einer natürlichen Sprache machen. Ein weiterer Grund

für diese verkürzte Schreibweise ist auch die häufige Verwendung des Kommasymbols „ , “ als Alphabetsymbol, denn das könnte zu Missverständnissen führen. Zum Beispiel ist $01, 10, 0$ ein Wort über $\{0, 1\} \cup \{,\}$.

Aufgabe 1.2 Schreibe zu den folgenden Symbolfolgen jeweils das kleinste Alphabet Σ auf, so dass die Symbolfolge ein Wort über Σ ist.

(a) *abbabb*

(b) $01000, (00)!$

(c) 1111111

(d) $aXYabuvRS$

Beachte: Wörter sind immer endliche Folgen von Buchstaben. Somit ist die unendliche Folge $1111\ldots$ kein Wort über dem Alphabet $\{1\}$.

Die **Länge $|w|$** eines Wortes w über einem Alphabet Σ ist die Länge des Wortes als Folge, das heißt die Anzahl der Buchstaben der Folge. Somit ist $|01a2b1| = 6$ für das Wort $01a2b1$ über dem Alphabet $\{0, 1, 2, a, b\}$. Für das leere Wort λ gilt $|\lambda| = 0$, weil λ keinen Buchstaben enthält.

Das größte der häufig verwendeten Alphabete ist Σ_{Tast}, das alle Symbole der Rechnertastatur enthält. Somit gehören alle kleinen und großen Buchstaben des lateinischen Alphabets und alle möglichen Sonderzeichen wie @, $, +, -, :, !, ? usw. dazu. Zu Σ_{Tast} gehört auch das Leerzeichen, das wir als $\sqcup$ oder durch einen leeren Abstand darstellen können. Wenn man ein Leerzeichen $\sqcup$ im Alphabet Σ_{Tast} hat, dann kann man jeden Satz wie zum Beispiel

Kryptographie ist faszinierend.

als ein Wort über Σ_{Tast} betrachten. Die zwei Leerzeichen und der Punkt in diesem Satz zählen als Symbole und somit gilt

$$|\text{Kryptographie ist faszinierend.}| = 31.$$

So gesehen ist jeder deutschsprachige Text ein Wort über Σ_{Tast}. Somit ist der Textinhalt eines Buches auch nur als ein Wort über Σ_{Tast} zu betrachten. Wir sehen also, dass in der Informatik ein „Wort über einem Alphabet“ dem entspricht, was man in der natürlichen Sprache unter „Wort“, „Satz“ und „Text“ versteht.

Um dies zu verdeutlichen, betrachten wir zuerst eine natürliche Sprache wie Deutsch, die auf dem lateinischen Alphabet basiert. Wenn wir nicht gerade griechische Buchstaben in mathematischen Texten verwenden, reicht uns Σ_{Tast} zur

Herstellung aller möglichen Texte auf Deutsch aus. Wir können also mit dem festen Alphabet Σ_{Tast} ewig auskommen, auch wenn sich die Sprache weiterentwickelt. Wenn man Bedarf nach neuen Begriffen hat (und dieser Bedarf ist ständig vorhanden), dann führt man neue Wörter in das Vokabular ein, die aus lateinischen Buchstaben zusammengesetzt sind. Ein festes Alphabet ist auf Dauer daher keine Behinderung für die Entwicklung einer natürlichen Sprache. Bei den Bildsprachen, wie zum Beispiel dem Chinesischen, ist dies anders. Da erzeugt man für jeden neuen Begriff (jedes neue Wort im Wörterbuch) ein neues Zeichen. Auch wenn dies meist aus Teilen zusammengesetzt ist, die bereits bestehenden Symbolen entsprechen, versteht man das Zeichen dabei als ein neues Symbol des Alphabets. Deswegen wächst das Alphabet mit der Entwicklung der Bildsprache immer weiter. Zu einem festen Zeitpunkt ist das Alphabet aber immer endlich und kann zum Schreiben beliebiger Texte über dem bestehenden Alphabet verwendet werden.

Aufgabe 1.3 Welches Alphabet verwendet man zur dezimalen Darstellung der natürlichen Zahlen? Wie hängt die Größe einer Zahl mit der Länge ihrer Dezimaldarstellung zusammen?

In der Informatik arbeiten wir immer mit festen Alphabeten, typischerweise mit Σ_{Bool} oder Σ_{Tast}. Die Bedeutung (die Semantik) der einzelnen Symbole ist nicht festgelegt. Was ein Symbol für uns unter gegebenen Umständen bedeutet, ist unsere Entscheidung. Beispiel: In einer Situation kann das Wort 11010 die binäre Darstellung der Zahl 26 sein, in einer anderen die Beschreibung eines Objekts oder die Kodierung einer Rechenoperation, wie etwa +.

Für endliche Automaten, die bei Lift- und Kreuzungssteuerungen eingesetzt werden, nutzen wir Symbole eines selbstgewählten Alphabets, um mit der Außenwelt zu „kommunizieren“. Kommunizieren bedeutet hier meistens, Signale zu empfangen. Für die Steuerung einer Ampel können wir an einem Fußgängerüberweg beispielsweise das Symbol „a“ verwenden, um dem Automaten mitzuteilen, dass Fußgänger die Straße überqueren möchten. Das Symbol „b“ könnte hingegen verwendet werden, um dem Automaten mitzuteilen, dass kein Fußgänger über die Straße gehen möchte. Genauso gut können wir für diesen Zweck die Symbole „0“ und „1“ oder beliebige andere verwenden.

Beispiel 1.1 An der T-Kreuzung in Abbildung 1.1 wären drei Ampeln A_1, A_2 und A_3 für Autos denkbar. Jede könnte mit einer Kamera ausgestattet sein, die signalisiert, ob auf der entsprechenden Seite Fahrzeuge über die Kreuzung fahren wollen. Jetzt könnte jemand sagen: „Mit dem Symbol (Signal) ‘a’ bezeichne