BULLETIN (New Series) OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 26, Number 2, April 1992
©1992 American Mathematical Society
0273-0979/92 $1.00 + $.25 per page

Finite automata, their algebras and grammars: Towards a theory of formal expressions, by J. Richard Büchi. Dirk Siefkes, editor, Springer-Verlag, New York, 1988, 316 pp., $69.50. ISBN 3-540-96905-5

1. Finite automata and formal languages

Mathematical models of computing machines were first used in 1936 by Turing [22] in the proof that certain problems admit no algorithm for their solution. Since then, mathematical automata of various sorts have proliferated in the research literature. Today, however, the term 'finite automaton' usually refers to a very restricted version of a Turing machine. Such an automaton consists of a finite set Q, a finite alphabet Σ of input symbols, and a state-transition rule, which is a function from $Q \times \Sigma$ into Q. One state is designated as the start state, and a subset of Q is designated as the set of accepting states. The automaton, beginning in the start state, reads a finite string of input symbols. As each new symbol is read, the automaton changes its state, based on the current state and the symbol, according to the state transition rule. The input string is accepted or rejected depending on whether the machine is in an accepting state when the reading of the input is completed.

For example, let

$$\Sigma = Q = \{0, 1\},$$

and let the state transition rule be given by

$$\delta(i, j) = i + j \mod 2.$$

With 0 as both the initial state and the sole accepting state, the resulting automaton accepts a string if and only if the number of 1's in the string is even. The first important theorem concerning these automata, due to Kleene [9], asserts that a set of strings is accepted by a finite automaton if and only if it can be constructed from the letters of Σ and the empty set by repeated application of the operations of finite union, concatenation

$$(U, V) \mapsto UV = \{uv\colon u \in U,\ v \in V\},$$

and closure

$$U \mapsto U^* = \{\lambda\} \cup U \cup UU \cup \cdots .$$

(The symbol λ denotes the empty string.) In the example above, the set of strings containing an even number of 1's can be obtained as

$$(0^*10^*10^*)^*.$$

(Writing 0 and 1 instead of $\{0\}$ and $\{1\}$ is a standard abuse of notation.)

The set of strings accepted by a finite automaton is called a *regular set of strings* or *regular language*. Expressions, such as the one that appears above to denote the set of strings with an even number of 1's, are called *regular expressions*. (There appears to be no good reason for this terminology, apart from tradition. The terms *recognizable set* and *rational set*—the latter because of important connections with rational numbers and rational power series—have been used, but 'regular' has resisted replacement.) The regular languages are remarkably insensitive to changes in the definition of the underlying machine model. For example, one can allow the automaton to move back and forth on its input string or to operate 'nondeterministically'—so that from a given state reading a particular input symbol, the machine may move into any one of several states—and the set of strings accepted is still a regular language. This suggests that, as with the recursive and recursively enumerable languages (the sets of strings accepted by Turing machines), the regular languages capture a notion of fundamental importance in computation, and not just the accidental properties of one of many possible machine models. The regular languages also arise as noncommutative generalizations of rational power series [17] and as the sets of strings definable in the monadic second-order theory of strict linear order [3].

The study of finite automata and the languages they accept continues to play an important role, although no longer a central one, in theoretical computer science. It has, in addition, generated a good deal of research in other areas: The work by Krohn and Rhodes [10] on the serial decomposition of automata has led to detailed investigations in the global structure of finite semigroups, with important applications to regular languages (see Eilenberg [7] and Pin [11]) and the complexity of boolean circuits (Barrington and Thérien [2]). Questions in the temporal logic of programs (Pnueli [12]) have renewed interest in the behavior of finite automata on infinite strings (Safra and Vardi [16]). Finite automata and their genralizations (especially to automata that accept trees rather than strings as input) have been applied in formal logic to characterize the classes of finite models of certain kinds of sentences and to provide deep results on the decidability of logical theories (Büchi [3], Rabin [15], Thatcher and Wright [18]). The theory also figures in the design of computer software: Some programs that scan a text file for a given pattern take a regular expression as input and simulate a finite automaton that accepts the specified language; programs used to generate the lexical analysis phase of compilers for programming languages work on similar principles (see Kernighan and Pike [8] for descriptions of the programs *grep* and *lex* and for further references).

A closely connected area of research is the study of systems of generating rules, or *grammars*, for sets of strings. In the most general form, a grammar consists of an alphabet Σ, an auxiliary alphabet Γ, an initial string w of symbols from $\Sigma \cup \Gamma$, and a finite set of rules of the form $u \to v$, where u and v are strings over $\Sigma \cup \Gamma$. The rule allows one to replace any string of the form xuy by xvy. The language generated by the grammar is the set of all strings over Σ that can be derived from the initial string w by a sequence of such replacements.

Fundamental investigations concerning these and related systems were carried out by Thue [19–21] eighty years ago. It was subsequently recognized that the sets of strings generated by such grammars are precisely the recursively enumerable languages. Thus basic problems concerning these grammars, such as the derivability of a given string in a given grammar, are algorithmically unsolvable (Post [13, 14]).

Restricting the form of the rules in the grammar restricts the class of languages that are generated. Thus, for example, the regular languages are precisely those generated by grammars in which every rule has one of the two forms $X \to Ya$, $X \to a$, where $X, Y \in \Gamma$, $a \in A$. (These grammars are often called *finite-state grammars* or *right-linear grammars*.

Grammars in which every rule has the form $X \to v$, where $X \in \Gamma$, are said to be *context-free*, and the sets of strings they generate are called *context-free languages*. Context-free grammars were introduced by Chomsky [5, 6] as one element in a mathematical model of natural language competence. (The other element is a set of rules for transforming the strings generated by the grammar to the actual sentences of the languages.) This work has been very influential in linguistics. An essentially equivalent scheme was devised by Backus and Naur for describing the syntax of programming languages (see [23]). Programs written in high-level programming languages must be parsed before they can be translated into low-level machine code. Parsing algorithms are generally constructed by producing a table of moves for the parser (essentially a finite automaton connected to a stack memory of unbounded capacity) from a context-free grammar for the programming language. This fact has been made the basis for parser-generator programs that, given a context-free grammar provided by the user, automatically produce a parser for the language generated by the grammar (see [1]).

2. Büchi's book

J. Richard Büchi made a number of important contributions to logic and the theory of automata. The book under review was left unfinished at his death in 1984. A complete manuscript of the first five chapters had been used as lecture notes in a course on automata that Büchi taught at Purdue University. The last two chapters were reconstructed by the editor, Dirk Siefkes, on the basis of assorted manuscripts and notes left by the author. While obvious gaps remain, the book presents a remarkably seamless appearance.

American universities now commonly offer, especially to computer science students, an undergraduate introductory course in the theory of computation, treating the fundamentals of finite automata, context-free languages and computability. As a result there has been a proliferation of textbooks in this area. While the present book is nominally an introductory text on finite automata (and, to a lesser degree, context-free languages) and contains many exercises, it is decidedly *not* a new entry in this growing field of textbooks. Büchi is less interested in showing the scope of the subject than in giving it a coherent development as a solid body of mathematics carried out in a very general framework.

The point of departure is to view an automaton as an algebra with a finite set of unary operations—the elements of the algebra correspond to the states of the automaton and the operations to the input symbols. This way of doing things was developed by Büchi and J. B. Wright around 1960. Their method yields an elegant treatment of the construction of the smallest (in terms of the number of states) automaton that accepts a given language: The minimal automaton appears as the minimal quotient algebra of all the automata accepting the language. This material is presented, in the early chapters, along with fairly standard expositions of the equivalence of deterministic and nondeterministic automata and of the theorem of Kleene mentioned above, as well as some interesting asides on codes and on star height. (These particular topics do not gain anything from the treatment of automata as unary algebras.)

The real novelty of the book is in the later chapters. Grammars (here called *canonical systems*) are introduced, and Büchi proves a far-reaching generalization (originally published in [4]) of the fact that regular languages are precisely those generated by finite-state grammars. In the last two chapters the unary algebras used to represent ordinary finite automata are replaced by algebras in which operations of arbitrary arity are permitted to appear. This allows tree automata, context-free grammars, pushdown automata, and even parsing algorithms of practical interest to be presented in a uniform framework. Much of the work of these later chapters remains unfinished and provides some intriguing oportunities for researchers.

Büchi had strong convictions about how things ought to be done, and the result is a highly individual book. One particularly maddening consequence of this is that his private notation and terminology, at variance with standard practice, are used without any mention of what everybody else calls the same things. (Input symbols, for example, are called 'input states,' which is very confusing. My favorite example is the 'Bunny Easter algorithm'—it's the inverse of the Easter Bunny algorithm—for the preorder traversal of a binary tree.) The book is filled with lively informal remarks intended to instruct the student and air the author's occasionally curmudgeonly opinions. Büchi gives due credit to Thue (and even to Leibniz!) for having anticipated many of the important developments in the area, but this leads him to make faintly disparaging remarks about the accomplishments of those who rediscovered and developed these ideas in more recent years.

I would not hand this book to a bright student seeking to learn about the theory of automata; other sources are more accessible and contain a more thorough exposition. But the reader who already knows something of the subject will find here a few deep and beautiful ideas and much food for thought.

References

1. A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, techniques and tools*, Addison-Wesly, Reading, MA, 1986.
2. D. Mix Barrington and D. Thérien, *Finite monoids and the fine structure of NC^1*, J. Assoc. Comput. Mach. **35** (1988), 941–952.
3. J. R. Büchi, *Weak second-order arithmetic and finite automata*, Z. Math. Logik Grundlag. Math. **6** (1960), 66–92.
4. ——, *Regular canonical systems and finite automata*, Arch. Math. Logik Grundlag. **6** (1964), 91–111.
5. N. Chomsky, *Three models for the description of language*, IRE Trans. Inform. Theory **2** (1956), 113–124.
6. ——, *On certain formal properties of grammars*, Inform. and Control **2** (1959), 137–167.
7. S. Eilenberg, *Automata, languages and machines*, vol. B, Academic Press, New York, 1976.
8. B. W. Kernighan and R. Pike, *The UNIX programming environment*, Prentice Hall, Englewood Cliffs, NJ, 1984.
9. S. C. Kleene, *Representations of events in nerve nets and finite automata*, Automata Studies, (C. Shannon and J. McCarthy, eds.), Princeton Univ. Press, Princeton, NJ, 1956.
10. K. Krohn and J. Rhodes, *The algebraic theory of machines* I. *Prime decomposition theorem for finite semigroups and machines*, Trans. Amer. Math. Soc. **116** (1965), 450–464.
11. J. E. Pin, *Varieties of formal languages*, Plenum, London, 1986.
12. A. Pnueli, *The temporal logic of programs*, Proc. 18th Sympos. Found. Comput. Sci. IEEE Press, New York 1977, pp. 46–57.

13. E. Post, *Recursively enumerable sets of positive integers and their decision problems*, Bull. Amer. Math. Soc. **50** (1944), 284–316.

14. ——, *Recursive unsolvability of a problem of Thue*, J. Symbolic Logic **12** (1947), 1–11.

15. M. O. Rabin, *Decidability of second-order theories and automata on infinite trees*, Trans. Amer. Math. Soc. **141** (1969), 1–35.

16. S. Safra and M. Vardi, *On ω-automata and temporal logic*, Proc. 21st Sympos. Theory of Computing, ACM Press, New York, 1989, pp. 127–137.

17. M. P. Schützenberger, *On the definition of a family of automata*, Inform. and Control **4** (1961), 245–270.

18. J. Thatcher and J. Wright, *Generalized finite automata theory with an application to a decision problem of second-order logic*, Math. Systems Theory **2** (1968), 57–81.

19. A. Thue, *Die Lösung eines spezialfalles eines generallen logischen problems*, Vidensdapssedkapet Skrifter Christiania, I. Math.-Nat. Klasse No. 8 (1910).

20. ——, *Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen*, Vidensdapssedkapet Skrister Christiania, I. Math.-Nat. Klasse No. 9 (1912).

21. ——, *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln*, Vidensdapssedkapet Skrifter Christiania, I. Math.-Nat. Klasse No. 10 (1914).

22. A. Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proc. London Math. Soc. (2) **42** (1936), 230–265.

23. R. Wexelblat, ed., *History of programming languages*, Academic Press, New York, 1981.

Howard Straubing
Boston College

BULLETIN (New Series) OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 26, Number 2, April 1992
©1992 American Mathematical Society
0273-0979/92 $1.00 + $.25 per page

Vector Lyapunov functions and stability analysis of nonlinear systems, by V. Lakshmikantham, V. M. Matrosov, and S. Sivasundaram. Kluwer Academic Publishers, Dordrecht, 1991, 172 pp., $79.00. ISBN 0-7923-1152-3

1. Introduction

One hundred years ago a Russian mathematician, A. M. Lyapunov, published a major work (printed or translated variously as [10, 11]) setting forth a method for studying stability properties of solutions of ordinary differential equations. The method is based on the construction of a function (now called a Lyapunov function) that serves as a generalized norm of a solution. Its appeal comes from the fact that properties of the solutions are derived directly from the differential equation itself (whence comes the name "Lyapunov's direct method").

This method is recognized by many investigators as the only general way of dealing effectively with stability questions of nonlinear ordinary differential equations. But for the past forty years it has also been used with marked success in the study of functional and partial differential equations. A careful look at many of these results shows that the method introduces a unifying thread