

The Book Review Column ¹

by **Nicholas Tran** (ntran@scu.edu)

Department of Mathematics & Computer Science, Santa Clara University



1 Notable New Releases

Programming-based Formal Languages and Automata Theory: Design, Implement, Validate, and Prove (Springer, 2024) by Marco T. Morazán (Seton Hall University) introduces formal languages and automata theory through programming using a Lisp-like language called FSM.

The Mathematics of Origami (Cambridge University Press, 2025) by Joseph O'Rourke (Smith College) is an intuitive and hands-on introduction to the mathematics of origami for a general audience, assuming only a high school geometry and trigonometry background.

Fundamentals of Probability and Statistics for Machine Learning (The MIT Press, 2025) by Ethem Alpaydın (Özyeğin University) is an introductory textbook for undergraduate or beginning graduate students that integrates probability and statistics with their applications in machine learning.

Sum Stories: Equations and Their Origins (Oxford University Press, 2025) by Robin Wilson (the Open University) is a collection of eighteen stories featuring famous mathematical equations and their historical development

Discrete and Computational Geometry, 2nd ed. (Princeton University Press, 2025) by Satyan Devadoss (University of San Diego) and Joseph O'Rourke (Smith College) is an updated and expanded edition of their 2011 book on the subject, previously reviewed in this column.

2 This Column

Mario Leston Rey takes an in-depth look at *Automata Theory: An Algorithmic Approach* (The MIT Press, 2023) by Javier Esparza and Michael Blondin, which presents the theory and applications of finite automata on both finite and infinite words taking the novel perspective of these machines as data structures. He particularly appreciates the book's clarity through well-chosen examples, its inclusion of unpublished material and reformulations of known results, and its large collection of exercises and detailed solutions.

David Luginbuhl reviews *Computability and Complexity* (The MIT Press, 2023) by Hubie Chen, which provides a balanced treatment of computability and complexity theory with a focus on rigorous proofs. He finds the book suitable for upper undergraduate or graduate courses in the theory of computation, praising its extensive *Exercises and notes* sections that include novel problems and explore some topics in more detail.

¹©2025 Nicholas Tran

In the mood for a mystery story? Wondering about deep connections between integers and permutations? Read Bill Gasarch's review of the graphic novel *Prime Suspects: The Anatomy of Integers and Permutations* (Princeton University Press, 2019) by master number theorist and expositor Andrew Granville and Jennifer Granville, illustrated by Robert J. Lewis, to get your fix.

3 How to Contribute

Doing anything fun for Labor Day this year? Consider writing a book review for SIGACT News. Either choose from the books listed below, or propose your own. In either case, the publisher will send you a free copy of the book. Guidelines and a LaTeX template can be found at <https://algoplexity.com/~ntran>.

BOOKS THAT NEED REVIEWERS FOR THE SIGACT NEWS COLUMN

Algorithms, Complexity, & Computability

1. Vaze, R. (2023). *Online Algorithms*. Cambridge University Press.
2. Ferragina, P. (2023). *Pearls of Algorithm Engineering*. Cambridge University Press.
3. Bosc, P., Guyomard, M., & Miclet, L. (2023). *Algorithm Design: A Methodological Approach - 150 Problems and Detailed Solutions*. Routledge.
4. Downey, R. (2024). *Computability and Complexity: Foundations and Tools for Pursuing Scientific Applications*. Springer.
5. Brody, J. (2025). *The Joy of Quantum Computing: A Concise Introduction*. Princeton University Press.
6. Dalzell, A., & McArdle, S., & Berta, M., & Bienias, P., & Chen, C.-F., & Gilyén, A., & Hann, C., & Kastoryano, M., & Khabiboulline, E., & Kubica, A., & Salton, G., & Wang, S., & Brandão, F. (2025). *Quantum Algorithms: A Survey of Applications and End-to-end Complexities*. Cambridge University Press.
7. Erciyes, K. (2025). *Guide to Distributed Algorithms: Design, Analysis and Implementation Using Python*. Springer.
8. Morazán, M. T. (2025). *Programming-based Formal Languages and Automata Theory: Design, Implement, Validate, and Prove*. Springer.

Miscellaneous Computer Science & Mathematics

1. Grechuk, B. (2019) *Theorems of the 21st Century*. Springer.
2. Nisan, N., & Schocken, S. (2021). *The Elements of Computing Systems: Building a Modern Computer from First Principles, 2nd ed.* The MIT Press.
3. Chayka, K. (2024). *Filterworld: How Algorithms Flattened Culture*. Doubleday.

4. Valiant, L. (2024). *The Importance of Being Educable: A New Theory of Human Uniqueness*. Princeton University Press.
5. Lichtman, E. (2025). *The Computer Always Wins: How Algorithms Beat Us at Our Own Games*. The MIT Press.
6. Rojas, R. (2025). *The Language of Mathematics: The Stories behind the Symbols*. Princeton University Press.
7. Wilson, R. (2025). *Sum Stories: Equations and Their Origins*. Oxford University Press.
8. O'Rourke, J. (2025). *The Mathematics of Origami*. Cambridge University Press.

Data Science

1. Alpaydın, E. (2025). *Fundamentals of Probability and Statistics for Machine Learning*. The MIT Press.

Discrete Mathematics and Computing

1. Ross, S., & Peköz, E. (2023). *A Second Course in Probability*. Cambridge University Press.
2. Devadoss, S., & O'Rourke, J. (2025). *Discrete and Computational Geometry, 2nd ed.* Princeton University Press.

Cryptography and Security

1. Garfinkel, S. (2025). *Differential Privacy*. The MIT Press.

Combinatorics and Graph Theory

1. Landman, B., Luca, F., Nathanson, M., Nešetřil, J., & Robertson, A. (Eds.). (2022). *Number Theory and Combinatorics: A Collection in Honor of the Mathematics of Ronald Graham*. De Gruyter.



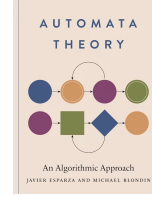
Automata Theory: An Algorithmic Approach**Javier Esparza and Michael Blondin**The MIT Press, 2023
560 pages, \$80 Hardcover

Review by

Mario Leston Rey

CMCC

Universidade Federal do ABC

**1 Overview**

Consider the following problem: suppose we are given an infinite set U , and we want to represent subsets of U . The operations of interest on these subsets are the Boolean operations from set theory, such as intersection, union, and complementation. In addition, operations on relations—subsets of $U \times U$ —such as projection and join (composition), must also be supported. The next table (taken from the book) summarizes these operations on sets and relations:

Table 1: Operations and tests for manipulation of sets and relations.

Operation on sets	Returns
$\text{Complement}(X)$	$U \setminus X$
$\text{Intersection}(X, Y)$	$X \cap Y$
$\text{Union}(X, Y)$	$X \cup Y$
Test on sets	Returns
$\text{Member}(x, X)$	true if $x \in X$, false otherwise
$\text{Empty}(X)$	true if $X = \emptyset$, false otherwise
$\text{Universal}(X)$	true if $X = U$, false otherwise
$\text{Included}(X, Y)$	true if $X \subseteq Y$, false otherwise
$\text{Equal}(X, Y)$	true if $X = Y$, false otherwise
Operation on relations	Returns
$\text{Projection}_1(R)$	$\pi_1(R) = \{x : \exists y (x, y) \in R\}$
$\text{Projection}_2(R)$	$\pi_2(R) = \{y : \exists x (x, y) \in R\}$
$\text{Join}(R, S)$	$R \circ S = \{(x, z) : \exists y \in X (x, y) \in R \wedge (y, z) \in S\}$
$\text{Post}(X, R)$	$\text{post}_R(X) = \{y \in U : \exists x \in X (x, y) \in R\}$
$\text{Pre}(X, R)$	$\text{pre}_R(X) = \{y \in U : \exists x \in X (y, x) \in R\}$

An infinite set has an uncountable number of subsets. The set of possible data structures, however, is countable, since each one is a finite object. As a consequence, there exist subsets that cannot be represented by any data structure. Thus, any data structure for manipulating infinite sets

¹©2025 Mario Leston Rey

must strike a balance between expressiveness (which sets can be represented) and manipulability (which operations can be efficiently implemented).

The book presents a different perspective on the theory and applications of finite automata, both on finite and infinite words, highlighting their role as data structures capable of solving certain instances of this problem.

To give a taste of the technical content and style of the book, a few formal definitions are included throughout this review.

To begin the story of the book, we introduce its main character. Informally, a *semi-automaton* is an entity consisting of *states* and *transitions*. A distinguished nonempty subset of states is declared *initial*. Each transition has an *origin* and a *destination*, which are states, and a *label*, which is a symbol taken from some alphabet Σ —a nonempty and finite set of symbols. A *word* over Σ is a finite sequence of symbols taken from Σ , and a *language* over Σ is simply a subset of Σ^* , the set of all words over Σ . An ω -*word* over Σ is a function from $\mathbb{N}$ to Σ . The set of all such functions is denoted by Σ^ω , and an ω -*language* is a subset of Σ^ω .

The evolution of a semi-automaton is captured by the notion of a *run* (or a *computation*), which is simply an alternating sequence of states and transitions that begins in an initial state (and, when the run is finite, it ends in a state), such that if p, t, q are consecutive elements of the run, and p, q are states and t is a transition, then p is the origin and q is the destination of t . Each run c spells out a word, which is obtained by concatenating the labels of the transitions that make up c .

Thus, a semi-automaton can be defined as a tuple (Q, Σ, δ, Q_0) , where

- Q is a finite set of states;
- Σ is a finite input alphabet;
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function; and
- $\emptyset \neq Q_0 \subseteq Q$ is the set of *initial states*.

A semi-automaton is said to be *nondeterministic* in general, and *deterministic* whenever Q_0 is a singleton set and $\delta(q, a)$ is a singleton set for each $q \in Q$ and $a \in \Sigma$. In this case, both sets are identified with their unique element. These two notions play a fundamental role throughout the text.

Specializations of the notion of semi-automaton yield data structures capable of representing (possibly infinite) subsets of an infinite set.

2 Summary of Contents

The first part of the book, consisting of Chapters 1 to 9, is dedicated to finite automata over finite words. The second part, comprising Chapters 10 to 14, deals with ω -automata—automata designed to handle infinite words.

Chapter 1 introduces regular expressions and variants of finite automata, and shows that these formalisms constitute what the authors refer to as a *trinity*. Each provides a finite mechanism for representing languages, and they are equally expressive in the sense that they define the same class of languages. We now turn to define the main entities of the chapter.

The discussion begins with the usual suspects: alphabets, words, and languages. In addition to the standard set-theoretic operations on languages, the chapter introduces language concatenation

and iteration (Kleene star). It then moves to regular expressions, which are syntactic forms used to generate languages. A *regular expression* over an alphabet Σ is defined by the following grammar:

$$r ::= \emptyset \mid \varepsilon \mid a \mid (r r) \mid (r + r) \mid r^*,$$

where $a \in \Sigma$. The language $\mathcal{L}(r)$ generated by a regular expression r is defined inductively as follows:

$$\begin{aligned}\mathcal{L}(\emptyset) &= \emptyset \\ \mathcal{L}(\varepsilon) &= \{\varepsilon\} \\ \mathcal{L}(a) &= \{a\} \\ \mathcal{L}(r_1 + r_2) &= \mathcal{L}(r_1) \cup \mathcal{L}(r_2) \\ \mathcal{L}(r_1 r_2) &= \mathcal{L}(r_1) \cdot \mathcal{L}(r_2) \\ \mathcal{L}(r^*) &= \mathcal{L}(r)^*\end{aligned}$$

A language over Σ is said to be *regular* if it is generated by a regular expression over Σ .

A *nondeterministic automaton* is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- (Q, Σ, δ, Q_0) is a finite semi-automaton; and
- $F \subseteq Q$ is the set of *final* states.

The automaton A is said to be *deterministic* when the underlying finite semi-automaton is deterministic.

Another fundamental notion is that of acceptance, which allows one to associate a language with each finite automaton. A word w is *accepted* by A if there is a run that spells w and ends in a state inhabiting F . The *language accepted* by A is the set $\mathcal{L}(A)$ of all words accepted by A .

A graph-theoretical perspective on nondeterministic automata reveals that, for the purpose of language acceptance, only the states reachable from some initial state are relevant. Such an automaton is said to be in *normal form*.

The central result of the chapter establishes the equivalence of the three core formalisms. For all languages L over Σ :

- (i) L is a regular language;
- (ii) L is accepted by a nondeterministic finite automaton;
- (iii) L is accepted by a deterministic finite automaton.

To prove this equivalence, the chapter introduces two intermediate models: NFA- ϵ , which are NFAs allowing ϵ -transitions (i.e., transitions that consume no input symbol), and NFA-reg, in which transitions may be labeled by regular expressions rather than single symbols.

The proof that (ii) implies (iii) employs the well-known subset construction. This construction transforms a nondeterministic automaton A with n states into a deterministic automaton D with 2^n states such that $\mathcal{L}(A) = \mathcal{L}(D)$. The chapter also presents an algorithm that constructs D directly in normal form, helping mitigate the exponential state blowup in practice.

Chapter 2 deals with the minimization of DFA and the reduction of the number of states of an NFA. These are critical tasks when viewing automata as data structures for representing

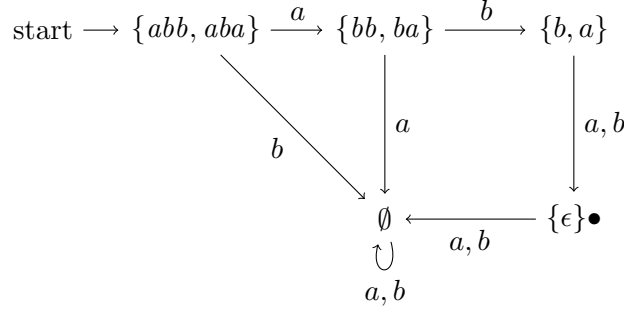


Figure 1: The figure illustrates the canonical automaton for $\{abb, aba\}$.

languages. It shows that the uniqueness of the minimal DFA provides a canonical representation of a regular language. The chapter also presents a heuristic for reducing the size of nondeterministic finite automata (NFA) and proves that computing a minimal NFA is a PSPACE-complete problem. Moreover, it shows that there is no canonical minimal NFA.

For a regular language $L \subseteq \Sigma^*$, a DFA A that accepts L is said to be *minimal* if every DFA that recognizes L has at least as many states as A .

The canonical automaton of a language plays a fundamental role. Given a language $L \subseteq \Sigma^*$ and a word $w \in \Sigma^*$, the *residual* of L with respect to w is the language

$$L^w = \{u \in \Sigma^* \mid wu \in L\}.$$

The *canonical* automaton of L is the automaton

$$C_L = (Q_L, \Sigma, \delta_L, q_{0L}, F_L),$$

where:

- $Q_L = \{L^w \mid w \in \Sigma^*\}$;
- $\delta_L(K, a) = K^a$, for each $K \in Q_L$ and $a \in \Sigma$;
- $q_{0L} = L$;
- $F_L = \{K \in Q_L \mid \varepsilon \in K\}$.

It is shown that for each regular language L the automaton C_L is the unique minimal DFA up to isomorphism that recognizes L (while the notion of isomorphism is not formally defined in the text, its meaning is made clear through the structure of the proof).

The text presents Hopcroft's version of Moore's algorithm [3], which is a partition refinement algorithm whose main idea—to partition a block into two disjoint blocks—is also used to reduce the number of states of an NFA.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton (DFA). *Moore's algorithm* consists of the following iterative procedure. Start with a partition $\mathcal{P}$ of Q such that every block $P \in \mathcal{P}$ satisfies either $P \subseteq F$ or $P \cap F = \emptyset$. That is, the initial partition is $\{F, Q \setminus F\}$. At each step, the algorithm checks whether the automaton obtained by collapsing each block $P \in \mathcal{P}$ into a single state yields a DFA. If so, the algorithm terminates. Otherwise, it identifies a block $P \in \mathcal{P}$ that is

non-deterministic with respect to the current partition – meaning that there exist states $p, p' \in P$ and a symbol $a \in \Sigma$ such that $\delta(p, a)$ and $\delta(p', a)$ lie in different blocks of $\mathcal{P}$. In that case, the algorithm refines the partition by splitting P into two subsets:

$$P_1 = \{q \in P \mid \delta(q, a) = \delta(p, a)\} \quad \text{and} \quad P \setminus P_1.$$

The next iteration begins with the updated partition $(\mathcal{P} \setminus \{P\}) \cup \{P_1, P \setminus P_1\}$.

At this point, it is clear that finite automata provide a finite representation of a language. **Chapter 3** shows how to implement the set-theoretic operations listed in Table 1 under the following assumption: given a universe U , there exists an injection $s : U \rightarrow \Sigma^*$, called an *encoding*, such that the set $\{s(x) \mid x \in U\}$ forms a regular language over Σ . The chapter focuses on implementing operations on *regular subsets* of U – that is, subsets $X \subseteq U$ such that $\{s(x) \mid x \in X\}$ is regular – via corresponding constructions on finite automata. For instance, given two finite automata A_1 and A_2 , it shows how to construct a new automaton that accepts $\mathcal{L}(A_1) \diamond \mathcal{L}(A_2)$, where $\diamond$ is a binary operation on languages. Methods are presented for computing the union, intersection, and complement of regular languages by manipulating the structure of deterministic or nondeterministic automata.

The implementation of binary operations involving two DFAs $A_i = (Q_i, \Sigma, \delta_i, q_{0i}, F_i)$ for $i \in \{1, 2\}$ can be achieved by taking the DFA induced by the set of states reachable from (q_{01}, q_{02}) of the *pairing* (or product) of the automata, i.e., the DFA $A = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F)$, where $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ for each $q_1, q_2 \in Q_1 \times Q_2$ and $a \in \Sigma$, and appropriately selecting the set F of final states. The text shows an algorithm that given two automata returns a product automaton that is in normal form. It also emphasizes that this construction does not, in general, preserve minimality, and so the resulting product automaton is not necessarily minimal.

Beyond these operations, the chapter addresses the following decision problems related to regular languages:

- *Membership*: whether a given word belongs to the language accepted by a given finite automaton;
- *Emptiness*: whether the language accepted by a finite automaton is empty;
- *Universality*: whether a finite automaton accepts all words over the alphabet;
- *Equality*: whether two finite automata accept the same language;
- *Containment*: whether the language of one finite automaton is contained in that of another finite automata.

All these problems are shown to be decidable, and efficient algorithms are presented in the case of DFAs. While membership and emptiness remain tractable for NFAs, the chapter closes by proving that universality and language inclusion are both PSPACE-complete problems. However, for both universality and inclusion, a refined *subsumption-based* algorithm, which is variant of subset construction, is introduced to control the explosion of explored subsets in practice.

The development throughout the text provides a clear picture of the view of finite automata as a data structure with its emphasis on constructions and algorithms. The PSPACE-completeness proof showcases the limits of this paradigm when nondeterminism is involved.

Chapter 4, the first chapter devoted to applications, focuses on two instances of the *pattern-matching problem*. In the first, given a nonempty word $t \in \Sigma^*$ (the *text*) and a regular expression

p (the *pattern*) over Σ , the goal is to find the length j of the shortest prefix u of t – if one exists – such that a word in $\mathcal{L}(p)$ is a suffix of u . The second instance is the special case where p is itself a word in Σ^* .

For the first variant, the chapter presents NFA and DFA-based algorithms. These work by constructing a finite automaton A that accepts the language $\mathcal{L}(\Sigma^*p)$ and checking whether there exists a prefix of t that belongs to $\mathcal{L}(\Sigma^*p)$ using A . The trade-offs between the NFA and DFA constructions are discussed, particularly the potential state explosion when converting the regular expression into a DFA.

Next, the chapter turns to the second instance, the *word case*, in which the pattern p is itself a word. This restriction makes the construction of a minimal DFA recognizing $\mathcal{L}(\Sigma^*p)$ straightforward. To improve the time complexity of the resulting algorithm, the notion of *lazy DFAs* (a restricted case of a two-way automata of Rabin and Scott [1]) is introduced. The chapter concludes with a novel derivation of the celebrated Knuth–Morris–Pratt algorithm.

Chapter 5 returns to Table 1 to show how operations on relations can be implemented. The conditions for an encoding are relaxed as follows. Given a universe U and an alphabet Σ that includes a padding symbol, say $\#$, a *partial encoding* is an encoding $s : U \rightarrow \Sigma^*$ (as in Chapter 3) such that for every $x \in U$, the last letter of $s(x)$ is different from $\#$. An *encoding* of U over Σ^* derived from s then maps each $x \in U$ to the (infinite) set $\{s(x)\#^n \mid n \geq 0\}$. In this approach, each object x is encoded as a (regular) language over Σ .

Given such an encoding of U over Σ^* and an NFA A , an element $x \in U$ is:

- *accepted* by A if A accepts all encodings of x , and
- *rejected* by A if A accepts none of them.

Moreover, A is said to recognize a subset $X \subseteq U$ if

$$\mathcal{L}(A) = \{w \mid \exists x \in X : w \text{ is an encoding of } x\}.$$

To handle relations, the chapter introduces the notion of a *transducer* over an alphabet Σ —that is, an NFA over the alphabet $\Sigma \times \Sigma$. Given words $u = a_1a_2 \cdots a_n$ and $v = b_1b_2 \cdots b_n$ over Σ , a transducer T accepts the pair (u, v) if it accepts the word $(a_1, b_1)(a_2, b_2) \cdots (a_n, b_n)$. This definition motivates the use of the aforementioned encoding scheme in this context, since transducers, as defined, are length-preserving. The notion of acceptance must then be extended to transducers in a way analogous to the extension made for NFAs.

The chapter then nicely presents algorithms for efficiently implementing the relational operations over regular relations – subsets of $U \times U$ which can be recognized by transducers – listed in the table, including projection, join, post-image, and pre-image. The presentation contains a carefully chosen set of examples that illustrate the constructions in a clear way.

Now, we turn to **Chapter 6**. Chapter 3 presented implementations of the operations of union, intersection, and complementation for DFAs. However, even when these algorithms are applied to minimal DFAs, the resulting automata are not necessarily minimal—except in the case of complementation. For operations involving relations, the situation is even more critical, as determinism may not be preserved, particularly in the case of projection and join operations. This chapter presents implementations that produce minimal DFAs as output, with no need for an additional minimization step, under the assumption that the universe is finite. In this setting, all elements of the universe can be encoded using words of fixed length over some alphabet.

It is possible to view the universe of regular languages over Σ as an automaton, called the *master automaton*, introduced in Chapter 3, whose states are the regular languages, and where there is a transition from a regular language L_1 to a regular language L_2 labeled by $a \in \Sigma$ if and only if $L_2 = L_1^a$. Moreover, the set of final states is the set of all regular languages that contain the empty word. For a regular language L , the automaton induced by the set of states reachable from L , called an *L -induced automaton*, is the minimal automaton that recognizes L . With this, it is possible to obtain directly a minimal automaton for representing any finite language (see Figure 2).

Let Σ^n denote the set of all words of length n , and consider a finite set $\mathcal{L}$ whose elements are subsets of Σ^n . A *multi-DFA* for $\mathcal{L}$ is simply the union of all L -induced automata, where the union is taken over each $L \in \mathcal{L}$. The chapter then proceeds with a detailed exposition of how to implement the operations listed in Table 1, both for sets and relations, in this special case of multi-DFAs. These constructions reuse the states of the master automaton, allowing efficient implementations of set and relation operations without duplicating structure unnecessarily.

The chapter closes by showing how binary decision diagrams can be used as minimal automata of a certain kind.

Chapter 7 is the second one of the first part dedicated to applications, and turns to the problem of verification of safety properties of sequential and concurrent programs with bounded-range variables.

Consider programs whose semantics transform an entity, called a *configuration*, into another configuration. With this setup, we can, for instance, construct an automaton whose states are all reachable configurations, plus an initial state, say i . All of its states are final. There is a transition $c \xrightarrow{c'} c'$ from a reachable configuration c to a configuration c' if the semantics of the program allow the transformation of c into c' . Moreover, for each initial configuration c , there is a transition $i \xrightarrow{c} c$. We are interested in the case where the set of configurations, say C , is finite. An execution is thus a run of this automaton, and can be seen as a word over the alphabet C —that is, as a language over C .

To test whether the set of executions $E \subseteq C^*$ of a program satisfies a certain property, which could be given by a regular expression over C , we can construct the automaton representing the set of executions, as well as the automaton representing the subset P of C^* that satisfy the property, and then check whether their intersection, $E \cap P$, is empty. This is the idea behind automata-based verification.

Network automata, discussed next, constitute an alternative way of modeling a program. Roughly, the idea is to model the behavior of each component of a configuration using a non-deterministic finite automaton (NFA). Thus, a network automaton is a tuple $\mathcal{A} = \langle A_1, \dots, A_n \rangle$ of NFAs, possibly over different alphabets.

The text presents an algorithm that constructs an equivalent NFA, the *asynchronous product*, from a given network automaton. This construction is related to pairing and in the worst case, grows exponentially with respect to the size of the network. It also provides a modification of this algorithm that computes an NFA – called the *system automaton* – which represents the set of all executions of a program modeled by the network automaton.

Next, it shows how network automata can model concurrent programs. To this end, the text uses Lamport and Burns’s mutual exclusion algorithm as an illustrative example.

The last section of the chapter is dedicated to the state explosion problem. As previously discussed, the verification task essentially reduces to deciding whether the intersection $\mathcal{L}(A) \cap \mathcal{L}(r)$ is empty, where A is an automaton that recognizes the set of program executions over the configuration

space C , and r is a regular expression that generates the subset of C^* consisting of traces that violate the desired property. Notice that when the system is modeled as a network automaton, A corresponds to the asynchronous product of the individual components, and thus its state space may grow exponentially with the size of the network. The verification problem is shown to be PSPACE-complete. An algorithm is then presented to check whether a property is violated, which, in a sense, constructs the asynchronous product on the fly.

The chapter concludes with a discussion of compositional verification and symbolic state-space exploration, followed by a short treatment of safety and liveness properties.

In **Chapter 8**, the relationship between automata and logic is explored. Consider, for instance, the following declarative definition of a language: the set of words over a, b that contain an even number of a s and an even number of b s. It is straightforward to construct a finite automaton that recognizes this language. Producing a regular expression that generates it, however, is less immediate. Of course, the equivalence between both models allows one to convert from one representation to the other. As this example illustrates, for certain languages, a declarative definition is more natural. This motivates the study of logical formalisms that enable languages to be defined declaratively.

The chapter begins by introducing a predicate logic over words, essentially a first-order logic in which atomic formulas express properties of word positions, and quantification ranges over those positions. It then presents a fundamental result showing that this logic is not expressive enough to define all regular languages. Specifically, it is shown that the language $\{a^{2n} \mid n \geq 0\}$ is not definable in this logic.

Next, the chapter introduces monadic second-order logic (MSO) on words, which extends first-order logic with two key features:

1. variables that range over sets of positions, and
2. the ability to express that a given position belongs to such a set.

Thus, in addition to quantification over positions, one can also quantify over sets of positions. The chapter presents a proof of the Büchi–Elgot–Trakhtenbrot theorem, which states that a language is regular if and only if it is definable by a formula in monadic second-order logic on words. As in previous chapters, several examples are provided to illustrate the constructions used in the proof, and they are particularly effective in clarifying the key ideas.

Chapter 9 discusses Presburger arithmetic and constitutes the third chapter on applications. Presburger arithmetic (PA) is a first-order language used to express numerical properties involving addition and comparison. The language of PA includes the constants 0 and 1, and features a single binary function symbol, namely $+$, which is written in infix form as usual. It also includes a single binary predicate symbol, $\leq$, likewise written in infix form. This syntax allows one to write formulas such as $\exists x (x \leq y + 1 \wedge y \leq x + x)$.

Let φ be a formula in PA. The chapter shows how to construct a transducer A_φ such that the language recognized by A_φ corresponds to the set of natural numbers (or integers) that satisfy φ . In line with the book’s algorithmic perspective, the construction is clearly described through a careful exposition of the underlying algorithms. The techniques from Chapter 5 make it possible to reduce this problem to that of constructing a transducer for atomic formulas of the form

$$a_1x_1 + \cdots + a_nx_n \leq b,$$

for which the chapter provides an algorithm along with a proof of correctness. It also addresses the special case of equations and presents an algorithm to handle that case as well. As in earlier chapters, the presentation is supported by clear and instructive examples that significantly aid comprehension.

Chapter 10 marks the beginning of the second part of the book, which focuses on the study of automata over infinite words. In line with Chapter 1, it begins by introducing the notion of an ω -regular expression over an alphabet Σ , which are syntactic forms generated by the following grammar:

$$s ::= r^\omega \mid rs \mid s + s,$$

where r is a regular expression over Σ . The meaning of an ω -regular expression s (defined recursively) is a subset $\mathcal{L}_\omega(s)$ of Σ^ω . This leads to the notion of an ω -regular language: a subset L of Σ^ω for which there exists an ω -regular expression s such that $L = \mathcal{L}_\omega(s)$. With this definition in hand, the chapter begins the search for data structures (automata) capable of representing such languages. The notions introduced here are more subtle, and the quest not only for a *trinity* among the various models presented, but also for closure under Boolean operations, becomes more elaborate and intricate.

The main character of the chapter is an ω -*automaton*, defined as a pair $A = (S, \alpha)$, where $S = (Q, \Sigma, \delta)$ is a finite semi-automaton and $\alpha : 2^Q \rightarrow \{0, 1\}$ is an *acceptance condition*. This abstract formulation elegantly subsumes all types of ω -automata presented in the chapter.

Given an infinite run ρ of S , let $\text{inf}(\rho)$ denote the set of states that occur infinitely often in ρ . The run ρ is said to be *accepting* if $\alpha(\text{inf}(\rho)) = 1$, and a word $w \in \Sigma^\omega$ is *accepted* by A if there exists an accepting run ρ of S that spells w . The language *recognized* by A is defined as

$$\mathcal{L}_\omega(A) = \{w \in \Sigma^\omega \mid A \text{ accepts } w\}.$$

The chapter first introduces Büchi automata and proves constructively that ω -regular expressions and Büchi automata are equivalent models with respect to ω -regular languages. A (non-deterministic) *Büchi automaton* (NBA) is an ω -automaton (S, α) such that α is a *Büchi condition*, meaning that there exists a set $F \subseteq Q$ such that $\alpha(Q') = 1$ if and only if $Q' \cap F \neq \emptyset$ for all $Q' \subseteq Q$. If S is deterministic, then A is called a *deterministic Büchi automaton* (DBA).

The chapter then shows that NBAs and DBAs are not equally expressive, and so DBAs cannot recognize all ω -regular languages. This is a manifestation that the notion of a *trinity* becomes more subtle in the setting of infinite words.

The next model introduced is the *co-Büchi automaton*, defined by an acceptance condition $\alpha : 2^Q \rightarrow \{0, 1\}$, called a *co-Büchi condition*, such that there exists a subset $F \subseteq Q$ with $\alpha(Q') = 1$ if and only if $Q' \cap F = \emptyset$ for all $Q' \subseteq Q$. The chapter presents a determinization algorithm that, given a nondeterministic co-Büchi automaton (NCA) A , returns a deterministic co-Büchi automaton D such that $\mathcal{L}_\omega(A) = \mathcal{L}_\omega(D)$ —that is, both ω -automata recognize the same ω -language. However, it is also shown that NCAs do not recognize all ω -regular languages—yet another manifestation of the subtlety involved in establishing a trinity for infinite-word automata.

The search for the trinity culminates with the introduction of Rabin automata, the next model presented in the text. A *Rabin automaton* (NRA) is an ω -automaton $A = (S, \alpha)$ such that α is a *Rabin acceptance condition*; that is, there exists a finite set $\mathcal{R} \subseteq 2^Q \times 2^Q$ such that for each $Q' \subseteq Q$, we have:

$$\alpha(Q') = 1 \iff \exists (F, G) \in \mathcal{R} \text{ such that } Q' \cap F \neq \emptyset \text{ and } Q' \cap G = \emptyset.$$

In other words, a run ρ is accepting if there exists a pair $(F, G) \in \mathcal{R}$ such that ρ visits states in F infinitely often and states in G only finitely often.

The book provides a constructive proof of the equivalence between NRAs and ω -regular expressions, but refrains from proving the equivalence between nondeterministic and deterministic Rabin automata (DRAs). The complexity of union, intersection, and complementation is analyzed for DRAs. It is shown that union can be implemented by pairing, whereas intersection cannot. Moreover, complementation requires a modification of the underlying semi-automaton. Thus, unlike the case of deterministic finite automata, not all Boolean operations can be implemented with ease.

The chapter then delves into Streett, Parity, and Muller automata. Table 2, extracted from the text, summarizes the results involving these models.

Table 2: Summary of results of automata on infinite words.

Automaton Type	Expr.	Det.	Union	Inters.	Comp.
NFA/DFA	<u>Y</u>	Y	<u>Y</u>	<u>Y</u>	<u>Y</u>
NBA/DBA (Büchi)	<u>Y</u>	N	<u>Y</u>	N	N
NCA/DCA (Co-Büchi)	N	Y	N	<u>Y</u>	N
NRA/DRA (Rabin)	<u>Y</u>	Y	<u>Y</u>	N	N
NSA/DSA (Streett)	<u>Y</u>	<u>Y</u>	N	<u>Y</u>	N
NPA/DPA (Parity)	<u>Y</u>	Y	N	N	<u>Y</u>
NMA/DMA (Muller)	Y	Y	Y	Y	Y

- **Expr.** Every ω -regular expression (for the row NFA/DFA, every regular expression) can be converted into an automaton of this type.
- **Det.** For every nondeterministic automaton of this type, there is an equivalent deterministic automaton of the same type.
- **Union.** Union of deterministic automata of this type can be implemented using the pairing construction.
- **Inters.** Intersection of deterministic automata of this type can be implemented using the pairing construction.
- **Comp.** Complementation of deterministic automata of this type can be implemented without changing the semi-automaton or the type of the acceptance condition.

The entries of the table are as follows: **N** (the property does not hold), **Y** (the property holds, but the underlying conversion or algorithm requires exponential time), and **Y** (the property holds and the underlying conversion or algorithm only requires polynomial time). In particular, **Y** indicates that the resulting automaton has polynomial size in the input.

Chapter 11 shows how to implement binary operations on sets under the assumption that objects are encoded as ω -words, and thus the set of encodings defines an ω -regular language, making automata on infinite words a suitable data structure for representing these sets. Table 2 indicates that deterministic Muller automata can be used to implement union, intersection, and complementation, although all three operations incur a worst-case exponential blow-up. The table also shows that Rabin, Streett, and parity automata exhibit the same behavior with respect to two out of these three operations.

The chapter then moves from deterministic to nondeterministic automata. To this end, it introduces *nondeterministic generalized Büchi automata* (NGAs), which differ from Büchi automata by having a set $\mathcal{G}$ of accepting sets. A run is considered *accepting* if it visits each set in $\mathcal{G}$ infinitely often. The chapter provides an algorithm that, given a GBA, constructs an equivalent NBA. Moreover, it presents solutions to the union and intersection problems involving NGAs. The union

construction, not surprisingly, resembles the one for NFAs, whereas the intersection relies on a pairing technique, supported by an algorithm that produces an NGA in normal form.

The complementation problem, however, cannot be addressed using the approach employed for NFAs, since there exist NGAs for which no equivalent deterministic generalized Büchi automaton exists. While the use of Muller automata could solve this issue, the corresponding construction is rather complex. Instead, the chapter follows a different strategy: it first transforms the NGA into an NBA and then applies a complementation algorithm for NBAs, which is presented in full detail throughout the remainder of the chapter.

It is worth to point out the use of the notion of the directed acyclic graph of a ω -word w – a construction that captures all possible runs of the automaton on an input word. This notion, introduced in Chapter 10 in the proof of equivalence between NCAs and DCAs, is employed again in proving the correctness of the complementation algorithm. Given a semi-automaton (Q, Σ, δ, Q_0) and a word $w = w_0w_1w_2 \cdots \in \Sigma^\omega$, the *directed acyclic graph* of w is the graph (V, E) , where V and E are the smallest sets satisfying:

- $\langle q, 0 \rangle \in V$ for every $q \in Q_0$;
- if $\langle q, i \rangle \in V$ and $q' \in \delta(q, w_i)$, then $\langle q', i + 1 \rangle \in V$ and $\langle q, i \rangle \xrightarrow{w_i} \langle q', i + 1 \rangle \in E$.

Chapter 12 is dedicated to the emptiness problem for NGAs, since problems such as membership, containment, and equivalence can all be reduced to it. For the purposes of the emptiness problem, transition labels are irrelevant, so δ is identified with the set

$$\{(q, q') \in Q \times Q \mid \exists a \in \Sigma \text{ such that } (q, a, q') \in \delta\}.$$

The following assumptions are made throughout the chapter regarding the primitive operations available to the algorithms. For an NGA A with generalized Büchi acceptance condition $\mathcal{G}$, the algorithms are allowed access to:

- (i) the set Q_0 of initial states;
- (ii) the set $\delta(q)$ of successors of each $q \in Q$;
- (iii) the collection $\{F \in \mathcal{G} \mid q \in F\}$ for each $q \in Q$.

Note that, because of (ii), the algorithms are restricted to forward exploration only. Such algorithms are called *on-the-fly*. This assumption aligns with other algorithms that operate in a similar fashion. For instance, algorithms for computing the intersection of two NGAs also construct the intersection in the forward direction. Therefore, to solve the emptiness problem for the intersection of two NGAs, one can compose both algorithms in a way that avoids explicitly constructing the intersection automaton.

The solution to the emptiness problem is presented next, based on two classical graph traversal algorithms: depth-first search (DFS) and breadth-first search (BFS). The problem is first addressed for nondeterministic Büchi automata (NBAs), where it reduces to deciding whether at least one state in the Büchi acceptance set F belongs to a cycle.

Although naive algorithmic solutions exist for this problem, the text develops a nested depth-first search strategy, which is described in detail along with a small optimization. It is observed that the nested DFS algorithm cannot be extended to NGAs, and therefore a conversion from an NGA to an NBA is required.

The text also explores a solution based on the notion of strongly connected components (SCCs) of a directed graph. A strongly connected component is a maximal subset $S \subseteq Q$ such that, for every pair of states $s, t \in S$, state t is reachable from s . It is well known that the directed graph obtained by collapsing each strongly connected component into a single node is acyclic. The text presents an algorithm, based on Tarjan’s algorithm for computing SCCs, to decide the emptiness problem, which takes $O(|Q| + |\delta|)$ time. The advantage of this algorithm is that it can be easily extended to deal with NGAs.

The chapter closes with a description of breadth-first search algorithms for the emptiness problem in the special case of NBAs. It discusses the Emerson-Lei algorithm for NBAs and its generalization to NGAs.

Chapter 13 is devoted to the verification of liveness properties of programs, while **Chapter 14** addresses monadic second-order logic (MSO) over ω -words. The latter presents a construction of a Büchi automaton that recognizes the set of ω -words satisfying a given MSO formula. These two chapters were not read with the care they deserve and thus are not reviewed in detail here.

3 Opinion

The book presents a consistent and coherent account of finite automata operating on both finite and infinite words. It brings to light a new perspective – finite automata as data structures – which reflects the influence of key research areas in computer science, particularly verification, where algorithmic and data-structural concerns are central.

The book offers a clear and accessible exposition of its topics. The chapters are carefully organized, and each subject is developed in a steady and logical progression. Particular care is devoted to the choice of examples, which are well-crafted to illustrate the key ideas. Algorithms are presented in a style that is comfortable for computer scientists, although I personally favor a more functional approach that avoids side effects. A particularly strong point of the book is the inclusion of exercises along with detailed solutions. They complement and enhance the value of the text. Finally, the book includes previously unpublished material and reformulations of known results, and delivers what it promises – a distinct perspective on automata theory – making it a valuable contribution to the literature.

Textbook presentations of topics related to ω -automata are scarce. For instance, the book by Khoussainov and Nerode [2], which is not cited in the bibliography, offers a presentation of Büchi, Muller, and Rabin automata with a distinct flavour that complements the present work. Connections to logic are also explored in Schneider [4] and Straubing [5], both likewise absent from the book’s references. Schneider’s text, in particular, provides a comprehensive and detailed treatment of verification. Finally, I would like to mention the book by Sakarovitch [3], which, in its first part, presents a beautiful account of classical automata theory (on finite words), and is also not cited in the bibliography.

As for prerequisites, a solid background in discrete mathematics is essential. Moreover, the book assumes what is often referred to – though never defined – “mathematical maturity”, which includes comfort with formal definitions, inductive proofs, and abstract reasoning. While not strictly necessary, a familiarity with basic notions of automata theory and grammars would certainly be beneficial.

The book can be used at the undergraduate level, possibly as a second course in automata theory, or even as a first course, provided the students already have a solid mathematical and

computer science background. Naturally, it also can be used as a textbook for an introductory graduate-level course.

References

- [1] Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, First Edition, Reading (MA), 1979.
- [2] B. Khoussainov and A. Nerode. *Automata Theory and its Applications*. Progress in Computer Science and Applied Logic, vol. 21. Birkhäuser, 2001.
- [3] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- [4] F. B. Schneider. *Verification of Reactive Systems*. Springer, 2004.
- [5] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

Review of ¹

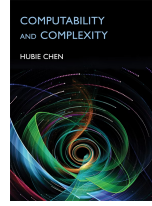
Computability and Complexity

Hubie Chen

The MIT Press, 2023
416 pages, \$65 Hardcover

Review by

David Luginbuhl (dluginbuhl@fit.edu)
Electrical Engineering and Computer Science
Florida Institute of Technology



1 Overview

Theory of computation and computational complexity have been well-studied and written about extensively over the past several decades, much of it chronicled in this publication. We are all familiar with the many textbooks that have captured the themes in a way that is accessible to college students (and we all have our favorites, I am sure). While many of these texts cover topics in both computability and complexity, they tend to emphasize the former to establish the fundamentals (and my experience has been that focusing on computation usually takes up most, if not all, of a semester-long formal languages course). Hubie Chen, in his new book *Computability and Complexity*, strives to achieve more of a balance between these two main topics. Through his highly mathematical treatment and in-depth proofs, he also builds a strong connection among the major themes of the text.

2 Summary of Contents

In the *Preface*, Chen describes the audience for this book, stating that it is written for a course “at the upper undergraduate level.” He also maintains that it would be useful for “students, researchers and workers in disciplines that draw on or depend on this theory” or as a reference for a theory-related course.

In the *Introduction*, Chen says that he is “motivated by two questions:

- What is *computable*?
- What is *efficiently computable*?”

Chapters 1 and 2 answer the first question, with a focus on finite state machines (Chapter 1) and Turing computability (Chapter 2). Chapters 3 and 4 comprise a detailed response to the second question, exploring issues of time-bounded computation (Chapter 3) and several additional complexity-related topics (Chapter 4).

¹©2025 David Luginbuhl

The *Agreements* section in the front matter provides a very short introduction (three pages) to the mathematical foundations necessary to understand computation, particularly, formal language theory, set theory, and relations and functions. The treatment is terse, but this knowledge is assumed coming in, and Chen is only ensuring we are all on the same page in terms of notation. I would add that readers will want to have a high comfort level with mathematical proofs, because they are the bread and butter of this book, especially in the latter part.

Each of the four chapters ends with an extensive *Exercises and notes* section, which explores some concepts from the chapter in more detail, introduces interesting related topics, and provides a large number of exercises.

3 Chapter Highlights

Chapter 1 Automata Theory – This chapter might be more appropriately titled *Automata Theory and Regular Languages*. DFAs, NFAs, and regular expressions are all introduced, but unlike many texts in this area, Chen does not address any language classes between those represented by DFAs and those represented by Turing Machines. This is just an observation and is not meant as a critique; Chen does not need the intermediate languages or automata to support his overall discussion of computation.

One interesting aspect of Chen’s approach to exploring finite automata is that he eschews extending the δ function (δ^*) in favor of *configurations* of the machine (indicating the current state and the rest of the string yet to be consumed). Chen uses this construct not only to walk through a computation, but also to define acceptance, the language of a machine, and as a result, whether or not a language is regular. Most theory texts introduce students to both the extended δ function and configurations, but Chen emphasizes the configuration approach.

The rest of the chapter is a fairly straightforward treatment of the various representations of regular languages - NFAs, ϵ -NFAs, and regular expressions. Closure properties are covered.

Chen introduces *pairwise separability* of strings as a way to discuss non-regularity of strings and minimization. The Pumping Lemma is found in the *Exercises and notes* section.

Chapter 2 Computability Theory – In Chapter 2, Chen turns to the concept of computation and to the Turing machine. As he says, “the *halting deterministic Turing machine* will be presented as a formalization of the intuitive notion of algorithms.” So a definition of algorithms is in order, and I like Chen’s: “a finite list of instructions; each instruction is finite and unambiguously describes an action performable mechanically, without recourse to judgment or creativity. An algorithm operates deterministically, and in discrete time steps. When executed on an input, an algorithm terminates after a finite number of steps, producing the desired output” (p. 71-72). Plenty of definitions of algorithms exist, but this one provides a nice connection to Turing machines.

For classes of languages, Chen prefers the terms *computable* and *computably enumerable* to *recursive* and *recursively enumerable*.

Chen introduces encodings to present concepts like Universal Turing machines and to prepare for the distinctions between computably enumerable, computable, and non-computably enumerable.

The chapter contains standard Turing machine topics such as closure properties and alternatives to deterministic Turing machines, in particular, nondeterministic Turing machines, multi-tape machines, and Random Access Machines. Reductions from one language to another are discussed near the end of the chapter to set up for further non-computability results.

Chapter 3 Complexity Theory – Having taken us through the basics of computation, Chen moves to complexity results, which I would say form the heart of this book. The first two chapters really set the stage for what follows here, and although the book (as indicated by the title) is about both computation and complexity, the coverage of complexity in this chapter is especially comprehensive and detailed. Because of this, I will provide a bit more detail on this chapter.

The first section provides some background in graph theory and focuses in particular on Eulerian cycles and graph coloring as basic algorithms for investigating complexity. Graphs are explained in detail, and associated problems are formalized as languages using the string encoding introduced in the previous chapter. The point here is to demonstrate their computability and also to use them as a basis for discussing time-efficient computation.

Chen then introduces time-bounded computation for both deterministic and nondeterministic machines and uses this to define PTIME and NP. Eulerian cycle detection and 2-colorings are provided as examples of PTIME languages - detailed proofs are provided. A non-graph example, determining whether an integer is a proper divisor of another integer, is also noted as a PTIME language.

Next up are NP languages, with Chen presenting some fundamentals and also establishing the basic relationship between NP and PTIME. Of course, he addresses “The P vs NP question.” He gives some attention to the concepts of *solving* and *verifying* as a way to address this question, characterizing it as follows: “Throughout human experience, the act of *creating* has been felt to be distinctly more challenging than the act of *evaluating* a creation; whether or not this perceived discrepancy is genuine, in the milieu of polynomial-time computation, can be seen as the heart of the P versus NP question!” (p. 159)

Several languages are examined that are in NP, such as HAMILTONIAN CYCLE and 3-COLORING. Chen provides an alternative formalism for considering whether a language is in NP by defining the concept of a polynomial time deterministic verifier and proving in great detail that a language is in NP if there is a verifier for the language.

Chen describes closure properties of PTIME languages, with a focus on complementation in order to introduce coNP. He then describes and provides fundamental relationships between NP and coNP.

In order to discuss NP-completeness, we must have an understanding of reductions. To exemplify the concept of reducibility, Chen again makes use of graph-theoretic concepts, this time with the languages of INDEPENDENT SET and CLIQUE. In this case, he shows that INDEPENDENT SET reduces in polynomial time to CLIQUE and (for good measure) to VERTEX COVER. Chen also discusses basic properties of reducibility.

Chen introduces NP-completeness and NP-hardness using the notion of polynomial time reducibility, which sets us up for the first NP-completeness result. While some texts begin with BOOLEAN SATISFIABILITY (essentially, Cook’s theorem), Chen chooses instead CIRCUIT SATISFIABILITY, showing that any NP language can be reduced to CIRCUIT SATISFIABILITY. He also examines coNP-hardness and coNP-completeness using languages related to CIRCUIT SATISFIABILITY.

Chen presents several additional NP-completeness proofs, moving from CIRCUIT SATISFIABILITY to BOOLEAN SATISFIABILITY and a number of variants. From there, Chen uses reductions to show NP-completeness for various graph problems, including INDEPENDENT SET, VERTEX COVER, DOMINATING SET, CLIQUE, HAMILTONIAN PATH and HAMILTONIAN CYCLE and related problems, as well as TRAVELING SALESPERSON.

This chapter can serve as a reference for NP-completeness results. The proofs are set up in

great detail and are quite thorough. Early on, Chen even provides a strategy to approaching these proofs for those not wishing to examine them all in detail.

Chapter 4 *Further Complexity Theory* – The final chapter introduces a number of additional topics in complexity. It begins with space complexity, providing some definitions and fundamental relationships between time and space. Chen defines PSPACE and discusses relationships between PTIME and PSPACE. He introduces quantified propositional formulas to prove QUANTIFIED SATISFIABILITY as a PSPACE-complete language. He then provides a detailed discussion and proof of Savitch’s theorem, showing that space bounds in NTMs and DTMs are polynomially related.

The next section of this chapter covers space and time hierarchy theorems. As Chen notes, these theorems show that “for each of the resources of space and time, *one can do more, with more of the resource*” (p. 291, emphasis in the original).

Chen next moves to fixed-parameter tractability and parameterized complexity, which were new concepts to me. They appear to provide a way to examine some types of problems (certain graph-theoretic and database query problems, for example) with more granularity than is allowed by the standard time complexity framework. Chen summarizes this as follows: “The notion of fixed-parameter tractability, in essence, allows one to identify problems where a seemingly unavoidable combinatorial explosion can be confined to a parameter, and thus to finely pinpoint sources of combinatorial hardness” (pp. 298-299). It is a fascinating topic that Chen addresses in sufficient detail and with motivating discussion for those interested in exploring further.

The last major topic in this chapter is compilability theory, which is related to the preceding parameterization discussion. As Chen states in introducing this concept, “When trying to solve instances of a problem, if it is the case that multiple instances of relevance share a feature in common, it may be fruitful to *compile* this feature into a format that allows for more efficient solution, even if the compilation process is relatively expensive” (p. 332). This section sets out to formalize that compilation process and show how it can be used.

The *Exercises and notes* section of this chapter introduces two additional topics worthy of consideration: logarithmic space complexity and treewidth. The latter is a graph-theoretic concept that Chen uses to provide more results on parameterized complexity.

I would treat this last chapter as more of a “for further reading” chapter. For those who wish to dig deeper into computational complexity, there is more than enough here to whet the appetite.

4 Summary

What stands out to me in Chen’s book above all else is the rigor and detail in his proofs throughout the text. And there are a **lot** of proofs. But Chen is very good about including motivation and high-level introductions to proofs where appropriate. Often, he includes intermediate results to move the discussion forward and keep the attention of the reader. It is possible to examine many of the proofs at multiple levels of detail. With respect to the proofs, I believe he is true to his intent (noted in the Preface) to provide a “uniform treatment of core concepts and topics.” There are many common threads here, and Chen does a good job of ensuring the reader is able to see how earlier results lead us to more advanced theorems.

Referring back to Chen’s discussion of intended audiences, I would say that the first two chapters are undoubtedly written at the upper undergraduate level. Interested undergraduate students with the requisite mathematical maturity will be challenged by, but benefit from, the complexity

discussion in the latter half of the text. As a whole, the book is appropriate for a graduate course in theory of computation and could be considered for a similar undergraduate course. I would certainly recommend it as a candidate text for a graduate research seminar in this area.

I agree that the extensive proofs provided, especially those in chapters 2 and 3, make this book useful as reference for those interested in these topics. It is deserving of a place on the bookshelf of anyone planning to do research in theory of computation.

Review of ¹

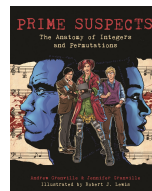
**Prime Suspects:
The Anatomy of Integers and Permutations
A Graphic Novel**

Andrew Granville and Jennifer Granville
Illustrated by **Robert J. Lewis**

Princeton University Press, 2019
232 pages, \$29.95 Paperback, \$24.50 eBook

Review by

William Gasarch (gasarch@cs.umd.edu)



Gasarch: Hardy, Ramanujan, what did you think of the graphic novel *Prime Suspects*?

Ramanujan: I thought it was great! It was intuitive! It discussed deep concepts in number theory and permutations and avoided proofs which would not have been appropriate for a graphic novel. And intuitions are more important than proofs anyway.

Hardy: I didn't like it! I found the lack of rigor disturbing. This, this, ... graphic novel is too informal! The math was correct but there were no proofs!

Ramanujan: The appendix, which is prose, gives more information and references, so that a reader can learn more if they want to.

Gasarch: So, Srinivasa, tell us about the main story. But don't give too much away.

Ramanujan: The story opens with two cops, Tao and Greene, finding a body: Arnie Int is dead! This seems related to another murder: Daisy Permutation is dead! In a normal mystery the question would be 'Who committed these murders?'. However, in this story they are more interested in the similarities of the two victims. Arnie Int's autopsy involves numbers and primes, whereas Daisy Permutation's autopsy involves permutations and cycles. You may wonder how an autopsy can involve math concepts. Well... the world of *Prime Suspects* is very different from our world.

Gasarch: A crime that involves math! So Godfrey, how does it compare to the TV show *Numb3rs*? (I reviewed *Numb3rs* in this column: <https://www.cs.umd.edu/~gasarch/bookrev/37-3.pdf>.)

Hardy:

1. The crime and solutions in *Numb3rs* make more sense.
2. The math in *Numb3rs* makes less sense and is often made up.

Gasarch: Did you like the story?

Ramanujan: The authors portrayed academia pretty well, though I have never seen a murder (note from Gasarch: I have taught for 40 years and have also never seen a murder). The world of *Prime Suspects* is different from ours in interesting ways.

¹©2025 William Gasarch

Hardy: But the mystery was unsatisfying.

Gasarch: It would be hard to have a story with both correct advanced math (which this story does) and a mystery worthy of Agatha Christie.

Hardy: Uh, Bill, recall that Ramanujan died in 1920, and Agatha Christie published her first novel in 1920.

Gasarch: Since I portrayed you as having an opinion about *Numb3rs*, I can have Srinivasa knowing who Agatha Christie is.

Ramanujan: I do. And yes, the mystery in *Prime Suspects* is not as good as a work by Agatha Christie. Though that is a high bar.

Gasarch: Godfrey, is the math that is discussed already known to my readers (as was the case for *Logicomix*, a graphic novel about logic that was reviewed in SIGACT News, Vol. 41, No. 2, 2010 (see <https://www.cs.umd.edu/~gasarch/bookrev/41-2.pdf>), or will it be new to them?

Hardy: Some of it was new to me!

Ramanujan: I had already intuited it.

Gasarch: One of you, describe the math.

Ramanujan: Numbers can be factored into primes. In 1919 Godfrey and I proved that a *typical* number n can be factored into $\sim \log \log n$ primes. There are many more theorems about factoring a number into primes. On the other hand, permutations can be factored into cycles. In 1942 Goncharov proved that a *typical* permutation on n letters can be factored into $\sim \log n$ cycles. There are many more theorems about factoring a permutation into cycles.

What is amazing is that many theorems about factoring a number into primes have an analog in factoring a permutation into cycles. I suspect your readers did not know that. Now they do!

Gasarch: So let's wrap this up. Both of you, say something you like and dislike about the book that you have not already mentioned.

Hardy:

LIKE: The art is great. The characters are well-drawn, and the math is well-illustrated. Most of the character's names were either those of mathematicians (e.g., Gauss), variants of names of mathematicians (e.g., Langer for Lang). Oh, and there is one more curious case: one of the main characters is a female math grad student whose name is Emmy Germain.

DISLIKE: The mystery and its resolution are not that interesting.

Ramanujan:

LIKE: In the background there are some math in-jokes.

DISLIKE: There is no character named after Godfrey Hardy or Srinivasa Ramanujan.

Hardy: Bill, how about you? Give us something you liked and something you didn't like, and you can't use any of our answers. And you can't use ChatGPT.

Gasarch: I'll give two LIKES and a DISLIKE.

LIKE: The authors discuss math I did not know that is interesting and deep.

LIKE: They didn't have every sentence end with an exclamation point, which is a common device for comic books! And a stupid one - not every sentence is that exciting!

DISLIKE: I have no criticism of the book, but I will give one of this review. Recall that *The New Mathematical Coloring Book* was written in the form of a memoir (*When I first met Paul Erdős ...*). Hence my review (see <https://www.cs.umd.edu/~gasarch/bookrev/NICK/soifer.pdf>) was in the form of a memoir (*When I first read this book ...*). Since *Prime Suspects* is a graphic novel, I would have wanted the review to be in that format. But alas, I have neither the time, money, nor expertise to pull that off! I hope that writing the review as a conversation suffices.

Ramanujan: Bill, you always end your reviews with saying who should buy this book.

Gasarch: A non-math person could understand some of the math in *Prime Suspects* but might not appreciate it. Hence I think that anyone who knows and likes math on the college level would like it.