The Book Review Column[1]

by William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742
email: gasarch@cs.umd.edu

In this column we review the following books.

1. **Algorithms Sequential & Parallel: A Unified Approach** by R. Miller & L. Boxer. Reviewed by by Anthony Widjaja. This book is a textbook for a senior or grad course in algorithms with an emphasis on parallel algorithms.

2. **Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics** by Juraj Hromkovič. Reviewed by Hassan Masum. This book examines a variety of practical ways to solve NP-complete problems in practice.

3. **Modal and Temporal Properties of Processes** by Colin Stirling. Review by Vicky Weissman. This book is about modelling a process by modal logics. It is a good introduction to process algebras.

4. **Modal Logic** by Patrick Blackburn, Maarten de Rijke, and Yde Venema. Reviewd by P. Daniel Hestand. This book is a comprehensive treatement of Modal logic. It includes Completeness theorems and also a discussion of the complexity of some of the problems in the field.

Books I want Reviewed

If you want a FREE copy of one of these books in exchange for a review, then email me at gasarchcs.umd.edu
Reviews need to be in LaTeX, LaTeX2e, or Plaintext.

Books on Algorithms, Combinatorics, and Related Fields

1. *Genomic Perl: From Bioinformatics Basics to Working Code* by Rex Dwyer.

2. *Diophantine Equations and Power Integral Bases* by Gaal.

3. *Solving Polynomial Equation Systems I: The Kronecker-Duval Philosophy* by Teo Mora.

4. *Computational Line Geometry* by Pottmann and Wallner.

5. *Linear Optimization and Extensions: Problems and Solutions* by Alevras and Padberg.

Books on Cryptography, Complexity, and Learning

1. *Cryptanalysis of Number Theoretic Ciphers* by Wagstaff.

2. *Learning with Kernels (Support Vector Machines, Regularization, Optimization, and Beyond)* Bernard Scholkopf and Alexander Smola.

3. *Learning Kernel Classifiers* by Herbrich.

4. *Boolean Functions and Computation Models* by Clote and Kranakis.

Review of **Algorithms Sequential & Parallel: A Unified Approach** [2]
Authors of Book: R. Miller & L. Boxer
Prentice Hall 2000, 330 pages, $68.00
Reviewed by Anthony Widjaja <twidjaja@acm.org> Melbourne University

1 Introduction

Nowadays, the study of parallel algorithms has become a mainstream topic in computer science partially because of the difficult problems in fields such as computational biology and computational geometry, to name a few. There have been several different approaches to teaching this topic at university level with which computer science educators around the world have come up. The authors of this book, Miller & Boxer, took a novel approach to the design and analysis of algorithms by unifying those of sequential and parallel algorithms. More importantly, the authors suggest that several courses based on this book has been taught successfully at both undergraduate and graduate levels at the State University of New York at Buffalo.

Prerequisites *(suggested by the authors)*:

- basic knowledge on data structures (e.g. stacks, queues, lists, trees) with which the reader who has taken a CS2 course must be familiar.
- fundamental discrete maths and calculus, especially summations, integrals, and limits.

Prerequisites *(suggested by the reviewer)*:

- knowledge on fundamental algorithms (e.g. sorting and searching algorithms).

2 Summary of Contents

Chapters 1,2 & 3 contain the mathematical preliminaries required in later chapters. In particular, Miller & Boxer review some fundamentals of asymptotic analysis in chapter 1. Thereafter, in chapter 2, the authors revisit concepts of mathematical induction and recursion. As an aside, I was quite surprised to see the proof of the principle of mathematical induction in this chapter as I had initially thought that it was an axiom, which we cannot prove. In chapter 3, the master method, a cookbook-type of system for evaluating recurrence relations, is presented. The proof of the master theorem, upon which the master method is based, is also presented in gory detail.

Chapter 4 concentrates on *combinatorial circuits* and *sorting networks*. This chapter is used to motivate the study of parallel algorithms in later chapters. In particular, the *bitonic sort*, which is an example of parallel sorting algorithms, is presented in fair amount of detail. In chapter 5, Miller & Boxer introduce basic models of sequential and parallel computation. Initially, the authors introduce the RAM (Random Access Machine) model as a model of sequential computation, and the PRAM (Parallel Random Access Machine) model as a model of parallel computation. Later in the chapter some other parallel models of computation including the mesh, tree, pyramid, mesh-of-trees, and hypercube are discussed in a thorough manner.

One ubiquitous example of parallel algorithms is the operation of matrix multiplication. In chapter 6, the authors begin to draw the reader's attention to the field of scientific computing, in which many complex problems reside. In particular, the reader will see sequential and parallel algorithms for carrying out matrix multiplication and Gaussian elimination.

In chapter 7, the reader will learn the concept of *parallel prefix*, a powerful operation (especially in parallel computers), and efficient algorithms for performing this operation. Some basic applications of this operation are stipulated in this chapter, while some more advanced ones in later chapters.

Chapter 8 presents an implementation of linked lists for parallel architecture, the PRAM in particular. An interesting feature of this implementation is that it allows the computer to perform a *pointer-jumping algorithm*, which can speed up a linked-list traversal from the head to the last item of the list up to $O(\log n)$. After that, the linked-list implementation of parallel prefix, whose array implementation was introduced in the previous chapter, is presented.

In chapter 9, the reader will explore more into the design and analysis of divide-and-conquer algorithms using both sequential and parallel approaches. Some algorithms the reader will find are including mergeSort, quickSort and hyperQuickSort (a parallel implementation of quickSort).

Chapter 10, 11, 12, & 13 are a selection of (more advanced) topics in which the techniques introduced in previous chapters are extensively used. In specific, chapter 10 concerns with problems and solutions in the field of computational geometry. In chapter 11, Miller & Boxer delve into the field of image processing. Chapter 12 & 13 cover, respectively, topics of graph algorithms and numerical problems. All these topics are presented in .

3 Opinions

First of all, I'm happy to report that the "story" told by this book is easy to follow. In addition, this book is self-contained in the sense that every theorem presented in this book is proved. Also, a fair number of examples are worked out in great detail using a variety of different methods. That is, for example, given a problem, the authors first present a solution with a sequential algorithm and, in turn, introduce the parallel counterpart which often can solve the problem in a shorter amount of time. Finally, it is important to note the second printing of the book (which I have) has undergone quite a big change because of the relatively large number of small (mostly typographyical) errors found in the first printing. The website which contains errata, corrections, and supplementary materials for this book is available at `<http://www.prenhall.com/millerbox>` and updated regularly.

This book can serve as a prescribed textbook for an advanced algorithms course, which focus on the design and analysis of sequential and parallel algorithms. If this is the case, some suggestions as to the structure of the course can be found on the companion website. Also, I think this book is suitable to be read cover-to-cover since almost always the contents of subsequent chapters are based on those of the previous ones. Some chapters, though, I think are more important than others. For example, chapter 4 to 9 cover the essence of the design and analysis of sequential & parallel algorithms, while chapter 10 to 13 provide some applications of them.

However, this book is not about parallel programming such as "cluster computing in Linux". In fact, the authors emphasized that most of the parallel models of computation introduced in this book are not yet physically realizable due to current technological limitations in connecting processors and memory [p.82]. Furthermore, the authors use pseudocodes when presenting algorithms, which make things slightly worse for those only interested in the practical side of parallel algorithms. Nevertheless, this book should be appropriate for everyone with a theoretical taste of computer science, who is new to and interested in parallel algorithms.

Review of **Algorithmics for Hard Problems:** [3]
Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics
Juraj Hromkovič
Publisher: Springer, 2001
ISBN: 3-540-66860-8

Review by:
Hassan Masum
Carleton University, Ottawa, Canada
hmasum.com

1 Overview

Algorithmics for Hard Problems is an intriguing new book that attempts to fill an underserved niche: practical methods for attacking those pesky problems that are NP-hard but must nevertheless be dealt with.

On the whole, the book is well-written and forms a useful introduction to a wide variety of algorithmic methods. Many readers of this column may find it a handy collection, collecting important ideas that might otherwise be scattered across a variety of sources. While not an advanced text or comprehensive reference, it may prove a handy refresher even for those who already know most of the design methods.

2 Book Features and Contents

Since the chapters are relatively self-contained, I will intersperse chapter-specific comments with a chapter-by-chapter contents listing of the book:

1) Introduction. *Setting the stage via previewing the topics covered in the rest of the book and discussing motivation and aims.*

There is an amusing "input-constraints-cost-objective" formulation of the goals of the book. Most of the goals are achieved, with some caveats. In the introductory notes, a target audience of senior undergraduate and graduate students is mentioned for the rough versions of the book; however, the relevance for the classroom could be improved by including more exercises (and some solutions).

2) Basic Concepts. *Fundamentals of math (linear algebra, combinatorics, Boolean functions, number theory, probability) and algorithms (languages, problem specification, complexity theory, design paradigms).*

132 of 460 non-index pages are devoted to background mathematics and complexity theory. While largely superfluous for readers of this column, this could be helpful for students and application-oriented users without a strong theory background. Although the section is well-written, it does seem like an excessive amount of space to devote to elementary knowledge. I would suggest that the author needs to focus on his target audience a little better – if it is students, more exercises and a broader selection of proofs are needed, while if it is practitioners and theorists, the introductory section could be skipped and more material added on heuristic methods, statistical methods (very important for empirically characterizing truly hard problems), and so forth.

3) Deterministic Approaches. *Pseudo-polynomial-time algorithms, parameterized complexity, branch-and-bound, reducing exponential constants, local search, and relaxation to linear programming.*

A variety of practical methods are concisely explained, with attention in each case to both the fundamental idea and the potential utility. In section 3.6, it would be nice to see a description of the kinds of spaces

[3]

and problems that admit good local search solutions, and more discussion of the connection with matroids and greedy algorithms. There is perhaps too much focus on the slightly pathological TSP-nonlocality proof – this proof could have been condensed or simply referenced.

4) Approximation Algorithms. *Fundamentals (concepts and classification, stability, dual approximation algorithms), design methods (applied to several problems including cover problems, max cut, knapsack, TSP, and bin-packing), and inapproximability.*

The author does a good job in covering many different approaches comprehensibly, going far beyond the classic TSP reduction to Hamiltonian cycle. Section 4.3.5 has an interesting illustrations of TSP-approximation and stability...the discussion does however seem too TSP-specific, and I would have liked links to where else these methods have been or could be used.

5) Randomized Algorithms. *Fundamentals and classification, design methods, derandomization.*

Clear explanation of the basics of randomized algorithms. I particularly liked 5.3.6 on a "portfolio" approach combining random sampling and relaxation to linear programming for MAX-SAT; it is a good example to clarify the basic concepts.

6) Heuristics. *Simulated annealing and genetic algorithms.*

It is nice to see this section included with the other methods, and the author made the right choice in sticking to theoretical results (since an empirical study would have been much too long). However, the section is very small relative to the importance of heuristic methods – especially considering that section 7.2 mentions that heuristics are often the most practical solution method. Also, the Schema Theorem result in 6.3 has been extensively criticized, and I would suggest the author look at more recent theoretical work on evolutionary computation (and update some references).

Perhaps this section could be expanded to include other theoretical results on these and other heuristic methods, along with more discussion of where they are useful. Two specific methods to discuss would be i) support vector machines and related heuristics, which are intuitive, mathematically sophisticated, and powerful in practice; ii) simulation in combination with statistical analysis, since it's a key method for really complex problems (and since we almost always get intuition about a problem from "trying it out"). Both have a solid body of theory to draw from.

7) A Guide to Solving Hard Problems. *Economics, combining and comparing approaches, parallelization, future technologies, glossary.*

Section 7.2 is a humorous yet accurate page-and-a-half on the cost equation for actually using algorithmics, noting that the simple and easily applicable methods are most often used due to lower economic and cognitive costs. I'd like to see this expanded as it's extremely important in practice. It's nice to get a brief description in 7.6 of DNA and quantum computing, that accurately states their potential and limitations. In 7.6.3, perhaps the author meant "quantum interference" instead of "quantum inference".

References

There is an extensive set of references to most of the material, including many recent important books and papers. However, I noted that the references are too old in some fast-moving fields like molecular biology; some also have incorrect years, e.g. 1979 instead of 1997.

3 Opinion

Some pros and cons that I came away with after finishing the book:

Pros: The introductory material is well written and gives a good sense of what is in the book. Similarly, each chapter has an introduction which accurately presages the material in the rest of the chapter. — Each section has a short review at the end which briefly describes the main concepts introduced in the section. Terms introduced are also reviewed. — A reader new to the material but with some perseverance would

come away well-equipped to delve further into many different areas of algorithmics (which sometimes seem to be a little scattered in practice). — At the end of each chapter is a historical and bibliographical discussion. The glossary (with links to the section where each term is discussed) is also a useful touch.

Cons: The selection of some of the proofs seems slightly unbalanced, with many pages sometimes devoted to a perhaps not entirely central result (although this choice is partly a matter of personal taste). For some of the more specialized proofs, a condensed version could be given. (On the positive side, the author does often have some informal discussion of the goal behind a proof.) — A summary chart of complexity classes and solution methods would be useful. — The English is sometimes a little idiomatically unusual; while this usually just makes for an occasionally offbeat and perhaps interesting writing style, there were a couple of spots where the intent of a statement was not entirely clear.

This book is a worthwhile first step for those interested in "the theory of practical algorithms". To really get into depth, a reader would need to add supplementary reading, perhaps one text per chapter (e.g. *Complexity and Approximation* for Chapter 4, *How to Solve It: Modern Heuristics* for Chapter 6). However, this is a good book for a first look at several areas.

Algorithmics for Hard Problems will be valuable to many learners. I look forward to seeing more books that give a balanced overview of "algorithms for hard problems"!

Review of **Modal and Temporal Properties of Processes**[4]
Author of Book: Colin Stirling
Springer Verlag, Hardcover, 250 pages, 2001
Review by Vicky Weissman, Dept of CS, Cornell University

Overview

At a very high level, this book presents languages for describing processes and properties of processes. It then discusses techniques for checking if a process, or family of processes, has a given property.

What is a process? In essence, a process is something that can perform actions. By doing actions, processes may change. Examples of processes include a clock that can tick and a counter with value zero that can do an increment action to become a counter with value one. The following syntax is used to describe processes where E, possibly subscripted, is a process, both a and b are actions, and I is an indexing set.

$$E ::= a.E \mid E[a/b] \mid \sum_{i \in I} E_i$$

A process of the form $a.E$ can do action a to become process E. A process of the form $E[a/b]$ is identical to the process E, except that every occurrence of action b is replaced by action a. A process of the form $\sum_{i \in I} E_i$ refers to some process E_j where $j \in I$.

We would like to extend the language to describe interactions between processes, but first we need to explain what we mean by process interaction. Two processes interact when one sends information and the other receives it. We formalize this notion, by creating pairs of actions. Each pair consists of two actions with the same name, except that one has an overhead bar and is associated with out-going communication, while the other does not have a bar and is associated with in-coming communication. For example, a process E_1 may tell a process E_2 that it's idle by doing an action $\overline{a}$. E_2 receives this information by doing the action a. We say that a is the *co-action* of $\bar{a}$ and vice-versa.

To capture interactions between processes, we add the syntax $E_1|E_2$ and $E\backslash J$ to the language where E_1, E_2, and E are processes and J is a set of actions. A process of the form $E_1|E_2$ is the concurrent

[4] ©To Vicky Weissman, 2003

composition of processes E_1 and E_2. In other words, if E_1 can do an action a to become E_1' and E_2 can do the co-action $\bar{a}$ to become E_2', then $E_1|E_2$ can do a to become $E_1'|E_2$, can do $\bar{a}$ to become $E_1|E_2'$, and can do the internal action τ to become $E_1'|E_2'$. (We always use the symbol τ to refer to an internal action, which is one process doing an action and another process doing the co-action. The set of actions may not include $\bar{\tau}$.) The restriction operator $\backslash$ is used to synchronize processes, by forbidding actions to be performed unless the co-action is also done. More specifically, a process of the form $E\backslash J$ is identical to the process E except that it cannot do any action that is in J and it cannot do any action whose co-action is in J. For example, $(E_1|E_2)\backslash a$ synchronizes processes E_1 and E_2 with respect to action a, by forbidding the processes to do a or $\bar{a}$. If one of the processes does a (or $\bar{a}$), then it cannot do another action until the other process has done $\bar{a}$ (or a) and, thus, $(E_1|E_2)\backslash a$ has done τ instead of a or $\bar{a}$.

We write $E_1 \xrightarrow{a} E_2$ to say that process E_1 can do action a to become process E_2. For example, $a.E \xrightarrow{a} E$ is always true and $\sum_{i\in I} E_i \xrightarrow{a} F$ is true if there is some $j \in I$ such that $E_j \xrightarrow{a} F$. We use the symbol $\Rightarrow$ to discuss *observable actions* where an observable action is any action other than τ. If a is an observable action, then $E_1 \stackrel{a}{\Rightarrow} E_2$ means that process E_1 can become process E_2 by doing a sequence of actions in which the only observable action is a. The notation can be extended in a straightforward way to allow an action sequence in place of a single action. For example, if a and b are actions, then $a.b.E \xrightarrow{ab} E$ is always true and, if ϵ is the empty sequence, then $E_1 \stackrel{\epsilon}{\Rightarrow} E_2$ is true if E_1 can become E_2 without doing any observable actions.

Given a process, we would like to know if it has certain properties. For example, we may want to know if the process is deadlocked or if it could become deadlocked without doing any observable action. To discover if a property holds for a process, we need a language for properties and we need a formal definition of what it means for a process to have a property. The syntax for a simple property language is given below where Φ, possibly with a subscript, is a property and K is a set of actions. As an aside, we refer to the set of all actions except those in an action set K as the set $-K$. Similarly, the set of all actions is $-$ (for $-\emptyset$).

$$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K\rangle\Phi$$

tt stands for true; it is the property that every process has. Similarly, ff stands for false; it is the property that no process has. Conjunction and disjunction have their standard meanings. A process has the property $[K]\Phi$ if, after doing any action in K, it becomes a process that has the property Φ. A process has the property $\langle K\rangle\Phi$ if, after doing some action in K, it can become a process that has the property Φ. Formally, we define the following satisfiability relation where $E \models \Phi$ means that process E has (satisfies) property Φ and $E \not\models \Phi$ means that it does not.

- $E \models tt$ and $E \not\models ff$
- $E \models \Phi_1 \wedge \Phi_2$ iff $E \models \Phi_1$ and $E \models \Phi_2$
- $E \models \Phi_1 \vee \Phi_2$ iff $E \models \Phi_1$ or $E \models \Phi_2$
- $E \models [K]\Phi$ iff $\forall F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.F \models \Phi$
- $E \models \langle K\rangle\Phi$ iff $\exists F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.F \models \Phi$

In this language, a process E is deadlocked if it has the property $[-]ff$. Unfortunately, it may not be clear if E has this property when E contains the restriction operator $\backslash$, the concurrency operator $|$, or renaming. To simplify our analysis, we note that formulas written in our language (properties) adhere to the following:

- If $a \notin K$ then $a.E \models [K]\Phi$ and $a.E \not\models \langle K\rangle\Phi$.

- If $a \in K$ then $a.E \models [K]\Phi$ iff $a.E \models \langle K\rangle\Phi$ iff $E \models \Phi$.
- $\Sigma\{E_i : i \in I\} \models [k]\Phi$ iff for all $j \in I.E_j \models [k]\Phi$.
- $\Sigma\{E_i : i \in I\} \models \langle k\rangle\Phi$ iff for some $j \in I.E_j \models \langle k\rangle\Phi$.

These facts can be used to show that $E\backslash J \models \Phi$ iff $E \models \Phi\backslash J$ where $\Phi\backslash J$ is obtained easily from Φ. The concurrency operator and renaming can be handled similarly, although the solution for concurrency is not entirely satisfactory.

While the above property language is sufficiently expressive for the first deadlock example, it cannot capture the second one, namely a process may become deadlocked after some sequence of unobservable actions. To write such properties, we extend the logic to include the syntax $[\![]\!]\Phi$ and $\langle\langle\rangle\rangle\Phi$ where Φ is a property. If a process has the property $[\![]\!]\Phi$, then doing any internal action sequence results in a process that has the property Φ. (Formally, $E \models [\![]\!]\Phi$ iff $\forall F \in \{E' : E \stackrel{\varsigma}{\Rightarrow} E'\}.F \models \Phi$.) If a process has the property $\langle\langle\rangle\rangle\Phi$, then doing some internal action sequence, possibly the empty sequence, results in a process that has the property Φ. (Formally, $E \models \langle\langle\rangle\rangle\Phi$ iff $\exists F \in \{E' : E \stackrel{\varsigma}{\Rightarrow} E'\}.F \models \Phi$.) Using the extended logic, the ability to evolve silently into a deadlocked process can be written as $\langle\langle\rangle\rangle[-]ff$.

We may want to extend the logic further to handle convergence and divergence. A process converges ($\downarrow$) if it cannot perform internal actions indefinitely, otherwise it diverges ($\uparrow$). We write $[\![\downarrow]\!]\Phi$ to say that the process converges and has the property $[\![]\!]\Phi$. We write $\langle\langle\uparrow\rangle\rangle\Phi$ to say that the process either diverges or has the property $\langle\langle\rangle\rangle\Phi$. For example, the property that an action a must (and will) happen next is written as $[\![\downarrow]\!]\langle\langle-\rangle\rangle tt \wedge [\![-a]\!]ff$ where $[\![]\!][K][\![]\!]\Phi$ is abbreviated as $[\![K]\!]\Phi$ and $\langle\langle\rangle\rangle\langle K\rangle\langle\langle\rangle\rangle\Phi$ is abbreviated as $\langle\langle K\rangle\rangle\Phi$ for any action set K.

In the text, the simplest language consisting of $tt, ff, \wedge, \vee, [K]\Phi,$ and$\langle K\rangle\Phi$ is called M for modal logic. The language consisting of $tt, ff, \wedge, \vee, [\![K]\!]\Phi, [\![]\!]\Phi, \langle\langle K\rangle\rangle\Phi,$ and $\langle\langle\rangle\rangle\Phi$ is called M^O for observable modal logic. Finally, M^O with $[\![\downarrow]\!]\Phi$ and $\langle\langle\uparrow\rangle\rangle\Phi$ is the language $M^{O\downarrow}$.

When should we consider two processes to be equivalent? A popular definition is to say that two processes E and F are equivalent if for any action a,

- if $E \stackrel{a}{\rightarrow} E'$, then $F \stackrel{a}{\rightarrow} F'$ for some F' such that E' and F' are equivalent.
- if $F \stackrel{a}{\rightarrow} F'$, then $E \stackrel{a}{\rightarrow} E'$ for some E' such that E' and F' are equivalent.

Two processes that meet the above criteria are said to be *bisimilar*. A binary relation B is a *bisimulation* if for any process pair (E, F) in B, the processes E and F are bisimilar.

If E and F are bisimilar processes, then for any process G, for any set of actions K, for any action a, and for any renaming function r, the following hold where $X \sim Y$ means that processes X and Y are bisimilar.

1. $a.E \sim a.F$
2. $E + G \sim F + G$
3. $E|G \sim F|G$
4. $E[r] \sim F[r]$
5. $E\backslash K \sim F\backslash K$

Because of this, bisimilar processes can replace one another in process descriptions. Furthermore, bisimilar processes either both have or both fail to have any property that is expressible in $M^{O\downarrow}$.

Two processes that share the same $M^{O\downarrow}$ properties may or may not be bisimilar. To characterize the class that are, let a process E be immediately image-finite if for all actions a the set $\{E' : E \xrightarrow{a} E'\}$ is finite. A process is image-finite if it can only become an immediately image-finite process, regardless of which action sequence it does. Two image-finite processes that have the same $M^{O\downarrow}$ properties are bisimilar.

We can demonstrate that two processes are bisimilar by constructing a bisimulation that contains them. Alternatively, we can write a proof that uses algebraic and semi-algebraic theorems (such as $(a.x)\backslash K \sim a.(x)\backslash K$ if $\{a, \bar{a}\} \cap K = \emptyset$) to show the equivalence.

The definitions for bisimulation and image-finite processes can be modified in a straightforward way to handle observable actions. The relationship between bisimulations and $M^{O\downarrow}$ occurs between observable bisimulations and M^O. Many other properties hold for both bisimulations and observable bisimulations.

The property language $M^{O\downarrow}$ is sufficiently expressive to distinguish one image-finite process from another, provided that the two processes are not bisimilar. This does not mean, however, that all interesting properties can be written in $M^{O\downarrow}$. An example of an inexpressible property is 'the process can never become deadlocked'. More generally, $M^{O\downarrow}$ lacks the expressive power to state long term properties, such as safety properties (something never happens), liveness properties (something eventually happens), and repeating properties (something happens again and again, forever).

We can discuss these events, by adding propositional variables to M. (As we shall see, $M^{O\downarrow}$ is equivalent to a fragment of this logic.) To complement the new syntax, valuations are added to the semantics. The valuation V assigns each variable z to a set of processes $V(z)$. If a process is in $V(z)$, then it is said to satisfy the formula z relative to the valuation V.

We use formulas in the extended logic to assign variables to sets of processes. More specifically, we write $z = \Phi$ if the variable z is assigned to the set of processes that satisfy formula Φ. For example, if $z = [-]ff$, then z is the set of processes that are deadlocked. If the variable is assigned to a formula that contains it (e.g. $z = [-]z$), then there may be a number of process sets that satisfy the equation. Roughly speaking, the smallest set is the least fixed point, written $\mu z.\Phi$, and the largest is the greatest fixed point, $\nu z.\Phi$.

Least and greatest fixed points are found through an iterative process. The number of iterations needed, in general, is an open problem. However, if the variables do not occur under an alternation of fixed points, then the number of iterations is, at most, $k * n$ where k is the number of fixed points and n is the number of processes.

The extended logic is called Modal Mu-Calculus (or μM) and its syntax is:

$$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K\rangle\Phi \mid \mu z.\Phi \mid \nu z.\Phi$$

where Φ, Φ_1, and Φ_2 are formulas, K is a set of actions, and z is a variable. The meanings of tt, ff, $\Phi_1 \wedge \Phi_2$, $\Phi_1 \vee \Phi_2$, $[K]\Phi$, and $\langle K\rangle\Phi$ are similar to the ones for the modal logic M; the only difference is that the valuation must be considered. For example, a process has the μM property $\Phi_1 \wedge \Phi_2$ relative to a valuation V, if it has the property Φ_1 relative to V and the property Φ_2 relative to V. The meaning of fixed points is given below where the notation $V[S/z]$ refers to the valuation that is identical to V, except that variable z is assigned to the process set S. (Formally, $V[S/z](z) = S$ and for all variables $y \neq z$, $V[S/z](y) = V(y)$.)

- $E \models_V \mu z.\Phi$ iff for all sets of processes S, if $E \notin S$ then there exists a process $F \notin S$ such that $F \models_{V[S/z]} \Phi$
- $E \models_V \nu z.\Phi$ iff there exists a set of processes S that contains E and for all $F \in S, F \models_{V[S/z]} \Phi$.

μM is very expressive. It can capture any $M^{O\downarrow}$ formula. In particular, assuming the variable z is not in Φ, $\langle\langle\rangle\rangle\Phi = \mu z.\Phi \vee \langle\tau\rangle z$, $[\![]\!]\Phi = \nu z.\Phi \wedge [\tau]z$, $\langle\langle\uparrow\rangle\rangle\Phi = \nu z.\Phi \vee \langle\tau\rangle z$, and $[\![\downarrow]\!]\Phi = \mu z.\Phi \wedge [\tau]z$. Also,

many safety, liveness, and repeating properties that are inexpressible in $M^{O\downarrow}$ can be written in μM. For example, $\nu z.[K]ff \wedge [-]z$ says that no action in K ever happens, $\mu z.\langle - \rangle tt \wedge [-K]z$ says that some action in K eventually happens, and $\nu z.\langle K \rangle z$ says that actions in K can be done over-and-over again, forever.

Despite the greater expressive power, the relationship between μM and bisimulation is similar to that between M and bisimulation. Specifically, two bisimilar processes share the same μM properties and a weakened version of image-finiteness is needed for processes that share the same μM properties to be bisimilar.

A disadvantage of using μM is that the satisfiability problem for it is undecidable. In general, we cannot determine if a process satisfies a μM formula relative to a valuation. For the other modal logics (M, M^O, and $M^{O\downarrow}$), we can grind through the satisfiability definition in a straightforward manner, but handling fixed points in μM is more challenging. One approach is to view answering a satisfiability question in terms of playing a game.

We want to play a game whose outcome will tell us if a process E satisfies a μM formula Φ relative to a valuation V. The game has two players, a verifier V who wants to show that the formula is satisfied and a refuter R who wants to show that it is not. A play of the game is a sequence of the form $(E_0, \Phi_0) \ldots (E_i, \Phi_i) \ldots$ where each subscripted E is a process and each subscripted Φ is a formula in μM. The rules of the game are:

- assuming we want to know if E satisfies Φ, E_0 is E and Φ_0 is Φ.
- if $\Phi_i = \Phi_j \wedge \Phi_k$ then player R chooses one of the conjuncts to be the formula in the next pair. In other words, E_{i+1} is E and R decides if Φ_{i+1} is Φ_j or Φ_k.
- if $\Phi_i = \Phi_j \vee \Phi_k$ then E_{i+1} is E and V decides if Φ_{i+1} is Φ_j or Φ_k.
- if $\Phi_i = [k]\Psi$, then player R chooses a transition $E_i \xrightarrow{a} E_j$ where $a \in k$. The next process, E_{i+1}, is E_j and the next formula, Φ_{i+1}, is Ψ.
- if $\Phi_i = \langle k \rangle \Psi$, then player V chooses a transition $E_i \xrightarrow{a} E_j$ where $a \in k$. The next process, E_{i+1}, is E_j and the next formula, Φ_{i+1}, is Ψ.
- if $\Phi_i = \sigma z.\Psi$ where $\sigma \in \{\mu, \nu\}$, then E_{i+1} is E_i and Φ_{i+1} is Ψ. Whenever z is encountered in the future, it will be replaced by Ψ. For example, if $\Phi_i = \mu z.\langle tick \rangle z$ then $\Phi_{i+1} = \langle tick \rangle z$, $\Phi_{i+2} = z$, and $\Phi_{i+3} = \langle tick \rangle z$. (For simplicity, we're assuming that each fixed point in Φ_i bounds a different variable and none of the bound variables are free elsewhere in the formula.)

Player R wins the game if the play contains a pair (E_i, Φ_i) such that $E_i \models_V \Phi_i$ is clearly false. Similarly, player V wins the game if the play contains a pair (E_i, Φ_i) such that $E_i \models_V \Phi_i$ is clearly true. If the play doesn't contain such a pair, then there must be some variables that are replaced according to the last rule infinitely often. If the outermost variable is bound to a least fixed point operator, then player R wins. Otherwise, player V wins.

If player R can always win the game, regardless of player V's moves, then E does not satisfy Φ relative to V. Otherwise, player V must be able to win the game, regardless of player R's moves, and E satisfies Φ relative to V.

Variations of this game have been developed to improve efficiency, particularly for fragments of μM. To prove that every process in a set satisfies some formula, we can play one of these games multiple (possibly infinitely-many) times. Alternatively, we can construct a proof tree, according to rules that are based on the satisfiability relation.

Contents

The book has seven chapters and is 181 pages. The chapter summaries are given below:

Chapter 1, **Processes**, introduces the notion of processes and gives a language in which to describe them.

Chapter 2, **Modalities and Capabilities**, presents the modal logics M, M^O, and $M^{O\downarrow}$ for discussing the properties of processes.

Chapter 3, **Bisimulations**, gives a formal definition for when two processes should be considered equivalent, argues that this choice matches our intuition, and examines the relationship between equivalent processes and those that are indistinguishable using the modal logics presented in Chapter 2.

Chapter 4, **Temporal Properties**, shows that the modal logics cannot be used to write safety and liveness properties. A variant of CTL is proposed as a more expressive logic. Fixed points are then introduced.

Chapter 5, **Modal Mu-Calculus**, defines the Modal Mu-Calculus (μM) and gives algorithms for finding fixed points using iterative methods.

Chapter 6, **Verifying Temporal Properties**, uses game theory to address the satisfiability problem for μM and the variant of CTL given in Chapter 4.

Chapter 7, **Exposing Structure**, uses proof trees to address the satisfiability problem for sets of processes.

Opinion

I recommend this book as a good introduction to process algebra, although an additional text will be needed for a solid understanding. Many examples are given throughout the text and exercises are provided at the end of each section. The examples complement the text very well and the exercises are given in increasing difficulty. This allows readers to test their general understanding first and then try to challenge themselves. The book also uses game theory as a way of presenting the same material from different perspectives without being overly redundant. I found this approach to be both effective and fun. Finally, the book does not assume much (if any) prior knowledge of the field.

I have only two negative comments. First, Chapters 4 and 5, while still good, are not as well written as the rest of the book. Second, the author occasionally refers to examples and definitions without telling the reader where the ideas are introduced. As a result, I had to flip through the book looking for text that I remembered reading, but whose details I couldn't recall. This situation may be helped by adding a glossary to future editions.

Review of
Modal Logic [5]
Cambridge Tracts in Theoretical Computer Science #53
Patrick Blackburn, Maarten de Rijke, and Yde Venema
Cambridge University Press, 2001

Reviewer: P. Daniel Hestand

1 Overview

Most people, upon hearing the phrase "modal logic," envision the logic of possibilities or contingent truth. The phrase also brings to mind the notion of truth in all "possible worlds" and even the typical modal operators, $\Diamond$ and $\Box$. These views, while correct, are somewhat limited in scope. This book attempts (and

[5]

succeeds, in my opinion) to update that view. The topic of the book, while obvious from the title, goes far beyond merely talking about "possibility" and "necessity." Instead, the authors present us with the notion that modal logics are tools for exploring and exploiting relational structures in ways that provide insight into areas other than logic. Frames and models are distinguished as differing structures upon which we can investigate various aspects of truth. Methods for extending the modal languages also give us insight into the limits of these structures and possible ways to circumvent some of these limitations. In the end, we are left with the impression that modal logic used as an analysis tool provides rich results beyond mere contingencies. This is the subject of the book, "Modal Logic."

2 Summary of Contents

The book is organized very nicely. Each chapter starts with a short transition section connecting the previous chapter with the current, followed by a chapter guide which indicates which sections of the chapter are on a basic track and which are on an advanced track. Presumably the concepts in the basic track are indispensable for an understanding of modal logic while those on the advanced track are for gaining a deeper understanding of those concepts introduced in the advanced track. Following the chapter guide, the chapter contents are launched into. Each section is ended by a set of exercises and the end of each chapter concludes with notes that generally contain historical perspective as well as names of note in the field. The notes are extremely useful because they are linked to the bibliography and provide a nice jumping off place for reading further work on a particular aspect of the concepts in the chapter. What follows is a summing up of each of the chapters.

- Preface — the preface is important because it lays out very explicitly, the assumptions used by the authors in preparing the text. Three "slogans" are introduced that are to be used as guiding principles in understanding modal logic. They are (directly from the book)
  - Modal languages are simple yet expressive languages for talking about relational structures.
  - Modal languages provide an internal, local perspective on relational structures.
  - Modal languages are not isolated formal systems.
- Basic Concepts — this chapter introduces those tools and techniques that are to be used in subsequent chapters. In particular, since modal languages can be used for discussing relational structures, those structures are introduced along with the notion of modal language. To connect the two, models and frames are introduced as different levels for understanding relational structures and the concepts of satisfaction and validity are introduced with respect to models and frames. Following this discussion, the authors introduce general frames which provide an intermediate level for understanding relational structures. After we understand the structures, languages, and interpretations that we can use, the authors then raise and answer the question of what consequence relations are important for modal languages. Up to this point, the discussion has been semantic, but now the authors intend to demonstrate that there are syntactic definitions that are useful since with them we can discuss proof theory and reasoning in modal logic. The chapter closes with a lengthy historical review of modal logic with respect to the topics covered in the chapter.
- Models — this chapter opens with a discussion of three ways of constructing new models from old ones: disjoint unions, generated submodels, and bounded morphisms. All of these are important because the new models they generate preserve theories associated with states in the old models. The next section then discusses bisimulation and shows that the three model construction methods previously introduced are only special cases of bisimulation. The next section shows how modal

languages have the finite model property, that is, if we can satisfy a formula on an arbitrary model, then we can also satisfy it on a finite model. However, we must be able to choose the "correct" finite model and this section also presents methods for doing so in the form of a selection method and a filtration method. The next section introduces the standard translation which is the link between modal languages and first-order languages. Two questions are raised here that show the limitations of the standard translation:

1. What part of first-order logic does modal logic correspond to?
2. Which properties of models are definable by modal means?

The next section then introduces ultrafilter extensions as a means of not only constructing new models, but also as a part of the answer to the two questions above. The next section is an in-depth look at the two major results of the whole chapter, namely van Benthem's Correspondence Theorem and the connection between modally definable classes of models and bisimulation and ultraproducts. The final section is an advanced track section discussing invariance under simulation and first-order definable operations respecting bisimilarity.

- Frames — the chapter opens with a discussion of frame definability, frames having already been introduced in chapter one. To aid in illustrating definability, several examples are provided. The next section discusses frame definability and its connection to second-order logic and in particular answers the question of why frame definability must be second-order. Following the format in chapter two on models, the authors proceed to demonstrate construction techniques on frames that preserve validity (namely disjoint unions, generated subframes, and bound morphic images) and show further that under ultrafilter extension validity is not preserved. The major result of this section if the Goldblatt-Thomason Theorem which defines the conditions under which a class of frames is modally definable. The next section then introduces finite frames and discusses a finite frame property connecting it to the finit model property through normal modal logic. The next section then discusses the first-order correspondence theorem applicable to frames — the Sahlqvist Correspondence Theorem and how it is possible to eliminate second-order quantifiers from a formula via the use of positive and negative formulae. The next section then proceeds to prove the Sahlqvist Correspondence Theorem (basic track) and the following section explores the limitations of the Sahlqvist Correspondence Theorem and introduces Kracht's Theorem. The final section provides a proof of the Goldblatt-Thomason Theorem.

- Completeness — this chapter is completely devoted to developing techniques for proving completeness and soundness (to a lesser degree). The opening section defines soundness and completeness in the context of normal modal logics. The next section introduces canonical models and frames, that is models that are built from a maximal consistent set of formulae with a canonical binary relation $R^{\Lambda}wu$ defined such that if $\psi \in u$ then $\diamond\psi \in w$ and a natural or canonical valuation defined by $V^{\Lambda}(p) = \{w \in W^{\Lambda} \mid p \in w\}$. The major result of this section is the Canonical Model Theorem which relates completeness of a normal modal logic and its canonical model. The next section explores canonicity and makes use of the implications of the Canonical Model Theorem to prove completeness results for several modal and temporal logics. The next section then discusses limitations to the use of canonical models by showing that there are normal logics that are not canonical and that some normal logics cannot be characterized by a class of frames. The next two sections deal with what to do when there exist properties for which no canonical formula exists. The methods introduced allow us to transform the canonical model with the failure into one without a failure by transforming the model or by inductively building up a model with very specific properties. The next section then

shows how certain proof rules allow us to construct canonical models containing submodels that express undefinable properties. The final two sections deal with finitary models (finite canonical models) and filtrations and how they can be used to prove weak completeness results for non-compact logics (for example, propositional dynamic logic).

- Algebras and General Frames — this chapter deals with how to express modal logics with algrebras versus using the models and frames of the previous chapters. The basic tenet here is that algebras provide a richer set of tools for dealing with modal logic problems. The first section introduces algebraic logic by exploring how propositional logic and boolean algebras are related. The next section then uses this foundation to begin to algebraize modal logic. This is done by introducing boolean algebras with operators. The boolean algebras will capture the underlying propositional logic and the operators will be used to capture the modalities. Two approaches are used: a semantic approach in which complex algebras are used and a syntactic approach in which Lindenbaum-Tarski algebras are used to obtain BAOs from normal modal logics. The next section provides a proof of the Jónsson-Tarski Theorem which is the basis for approaching modal completeness theory algebraically. The next two sections proceed to duality theory where the connection between the frame construction methods is made to construction methods in algebras, namely homomorphisms, subalgebras, and direct products. A major result in the first of the two sections is an algebraic proof of the Goldblatt-Thomason Theorem. The second of the two focuses on the use of general frames. The final section introduces a notion called persistence which is generalization of the canonicity concept and the uses this notion to prove the Sahlqvist Completeness Theorem.

- Computability and Complexity — this next chapter provides a connection of modal logic to the field of computer science by examining the computability and complexity issues surrounding satisfiability and validity problems in normal modal logics. The first section introduces satisfiability and validity problems in modal logic and how to express their computation using Turing machines. The next section then looks at decidability and shows how decidability may be proved using finite models. Three theorems are given and proven that related decidability to the finite model property and these results are then used to show that most of the logics introduced as examples in chapter 4 are actually decidable. The next section introduces interpretation in other decidable theories as a means of showing decidability for logics without the finite model property. The next section discusses how to arrive at decidability using quasi-models (a frame with distinguished points and a labeling function mapping states in the frame to subsets of the closure of the modal formula under discussion) and mosaics (a collection of pairs of Hintikka sets that can be used in a step-wise fashion to construct a satisfiable model). The next section detours into a discussion of undecidability results and shows how easy it is to create undecidable modal logics. The major result of this section is the introduction of tiling arguments as a proof method for determining undecidability. The remaining sections discuss the various types of complexity classes to which modal languages belong. The first of these sections discusses NP, the second shows that PSPACE is a key complexity class and shows that a group of normal logics between **K** and **S4** are PSPACE-hard, and the final section shows that the satisfiability problem for propositional dynamic logic is EXPTIME-complete.

- Extended Modal Logic — this chapter shows how modal logics may be extended. The authors indicate that the topics discussed in the chapter constitute six of their favorite topics. The first section introduces logical modalities, in particular the global modality (remember that modal languages are inherently local, so we need a way to step back and look at the bigger picture) and the difference operator (an operator that scans the whole model looking for different states to satisfy a particular formula). The next section then introduces the since and until operators that are used in temporal

logic. These operators are then used to arrive at expressive completeness and then finally to deductive completeness. The next section discusses hybrid logic which have terms that can be used to refer to worlds. The basic hybrid language is given and the hybrid logic completeness theory is discussed. The next section explores further the concept of the guarded fragment which is the result of generalizing the notion that modal operators are a form of guarded quantification over states in first-order logic. The next section discusses multi-dimensional modal logic which treats assignments as possible worlds and quantifiers as diamond operators. The beauty of this extension is that we can now approach first-order logic as a modal language. The final section introduces a Lindström Theorem for Modal Logic. The section shows that there is a modal logic equivalent to Lindström's Theorem which states that first-order logic is the strongest logic possessing compactness and Löwenheim-Skolem properties.

- Appendices — — the appendices provide "toolkits" to assist the reader in understanding the foundational concepts applied to the study of modal logic.
  - Appendix A — this appendix is concerned with logical concepts including first-order logic, model-theoretic approaches, ultraproducts, and first-order logic extensions. An important result of this appendix is Łös's Theorem.
  - Appendix B — this appendix deals with algebraic concepts such as universal algebra, homomorphisms, congruences, algebraic model theory, and equational logic.
  - Appendix C — this appendix deals with computability notions such as Turing machines, Church's thesis, complexity theory (the complexity classes), and big-O notation.

3 Opinion

This book has found a permanent home in my library. I find myself repeatedly going back to a section here or there and rereading. The authors have made a fantastic effort at exploring modal logic in all of its aspects and make it interesting with detailed discussion of results, numerous simple examples, and numerous extended examples. The book is well-organized and clearly distinguishes between those sections that a newcomer to the topic should explore and those sections which will provide a little more meat for the more experienced reader to explore. As a textbook, it would be difficult to cover all of the material in a single semester, but a survey course could hit the high points using the basic track material and follow up in a second semester with a more advanced approach. I recommend this book for anyone who has any interest at all in modal logic. It certainly expanded my view of just what modal logic can be.