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. **Model Checking** by Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. Review by Vicky Weissman. This is both an introductory text and a reference for Model Checking.
2. **Petri Net Algebra** by Eike Best, Raymond Devillers, Maciej Koutny. Review by Ivelin Ivanov. This is a monograph on Process Algebras and Petri Nets.
3. **Combinatorial Optimization - Theory and Algorithms** by Bernhard Korte and Jens Vygen Review by Ivelin Ivanov. This is a comphrensive text on Combinatorial Optimization that includes many recent results, with full proofs.
4. **Calculated Bets** by Steven Skiena. Review by William Gasarch. This is a book about using mathematics to improve your odds when gambling on Jai-Alai. It includes material on the game of Jai-Alai as well.

I am looking for reviewers for the following books

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. *Randomized Algorithms: Approximation, Generation, and Counting* by Bubley.
2. *Algorithm Design: Foundations, Analysis, and Internet Examples* by Goodrich and Tamassia.
3. *An Introduction to Data Structures and Algorithms* by Storer.
4. *Graph Colouring and the Probabilistic Method* by Molloy and Reed.
5. *Random Graphs* by Bela Bollobas.
6. *Geometric Computing and Perception Action Systems* by Corrochano.
7. *Computer Arithmetic Algorithms* by Koren.
8. *Structured Matrices and Polynomials: Unified Superfast Algorithms* by Pan.
9. *Computational Line Geometry* by Pottmann and Wallner.
10. *Bioinformatics: The Machine Learning approach* by Baldi and Brunak.
11. *Graph Separators, with Applications* by Rosenberg and Heath.
12. *Computational Commutative Algebra* by Kreuzer and Robbiano.

[1]© William Gasarch, 2002.

13. *Algorithms in C++* (Third edition). by Sedgewick.

14. *Algorithms Sequential and Parallel* by Miller and Boxer.

15. *Computer Algorithms: Introduction to Design and Analysis* by Basse and Gelder.

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

17. *Number Theory for Computing* by Yan.

18. *Calendrical Calculation: The Millennium edition* by Reingold and Dershowitz.

19. *An Introduction to Quantum Computing Algorithms* by Pittenger.

20. Randomization Methods in Algorithm Design. (DIMACS workshop)

21. Multichannel Optical Networks: Theory and Practice. (DIMACS workshop)

22. Advances in Switching Networks. (DIMACS workshop)

Books on Cryptography

1. *Foundations of Cryptography: Basic Tools* by Goldreich.

2. *Modern Cryptography, Probabilistic Proofs and Psuedo-randomness* by Goldreich.

3. *Introduction to Crytopgraphy* by Buchmann.

4. *Secure Communicating Systems: Design, Analysis, and Implementation* by Huth.

5. *Elliptic Curves in Crytography* by Blake, Seroussi, and Smart.

6. *Coding Theory and Cryptograph: The Essentials* by Hankerson, Hoffman, Lenoard, Linder, Phelps, Rodger, and Wall. *Introduction to Cryptography with Coding Theory* by Washington and Trappe.

Books on Complexity and Logic

1. *Models of Computation: Exploring the Power of Computing* by John Savage.

2. *Introduction to Automata Theory, Languages, and Computation* by Hopcroft, Motwani, and Ullman.

3. *Complexity and Information* by Traub and Werschulz.

4. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods* by Roever, De Boer, Hannemann, Hooman, Lakhnech, Poel, and Zwiers.

5. *Domains and Lambada-Calculus* by Amadio and Curien.

6. *Derivation and Computation* by Simmons.

7. *Types and Programming Languages* by Pierce.

8. *Handbook of Logic and Proof Techniques for Computer Science* by Krantz.

Review of: **Model Checking**[2]
Authors of Book: Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled
Published in 2000 by MIT press, \$52.00, 330 pages, 2000
Author of Review: Vicky Weissman, Dept of CS, Cornell Univ.

Overview

The goal of model checking is to determine if a given property holds in a particular system. For example, we may want to know that a server never sends sensitive data to the wrong client. Model checking has been used effectively in practice for many finite-state systems, including real-time applications, and for some infinite families of finite systems. Theorem proving and testing are other approaches for system verification. A key advantage of model checking over theorem proving is that it's verification procedure can be fully automated. As for testing, the results from a model checker, as with any formal method, are obviously more conclusive.

There are three main components to model checking; describing the system, stating the property, and verifying that the property holds in the system.

The system is typically described by either a finite Kripke structure, $M = (S, R, L)$, or a slightly more general model, $M_{gen} = (S, T, L)$, where S is a finite set of states, R is a binary relation on states, T is a set of binary relations on states, and L is a function from states to atomic propositions. Often, a subset of S will be designated as the start states. For simplicity, we require that there is at least one transition out of each state (i.e. $\forall s \in S \exists s'.(s, s') \in R$) and define a path to be an infinite sequence of states, $s_1, s_2, \ldots$ in which (s_i, s_{i+1}) is in R. In symbolic model checking, the state transition graphs are viewed as boolean formulas and represented using ordered binary decision diagrams (OBDD), because there are very efficient algorithms to manipulate OBDDs. A binary decision diagram (BDD)is a graph in which each terminal node is associated with a true or false value and any path from a designated root node to a terminal node corresponds to a set of variable assignments that make the encoded formula have the terminal node's value. An OBDD is a BDD in canonical form; all redundant nodes and edges are removed and there is an ordering on the nodes such that the i^{th} node in any root-to-leaf path is associated with the same variable in the encoded formula.

The most common logics to express system properties are Computational Tree Logic (CTL), Linear Temporal Logic (LTL), CTL* , and propositional μ-calculus. CTL and LTL are sub-logics of CTL* . Any CTL formula can also be expressed in the propositional μ-calculus. Although more complicated than the others, the propositional μ-calculus is particularly interesting, because many temporal and program logics can be encoded into it and a number of efficient model checking algorithms exist for it.

Formulas in CTL* are composed of atomic propositions, propositional logic symbols ($\wedge$, $\vee$, and $\neg$), path quantifiers (**A**and **E**), and temporal operators (**X**, **F**, **G**, **U**, and **R**) as follows:

- Every atomic proposition is a formula.
- If f and g are formulas, then $f \wedge g$, $f \vee g$, $\neg f$, $\mathbf{A}f$, $\mathbf{E}f$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are formulas.

Any CTL* formula can be expressed using only atomic propositions, $\wedge$, $\neg$, **X**, **U**, and **E**. We can define inductively what it means for a formula f to hold in a state s or a path $\pi = s_1, s_2, \ldots$ in a

[2]©2002 Vicky Weissman

model, M, (written $M, s \models f$ or $M, \pi \models f$) as follows (omitting the standard logical symbols $\wedge$, $\vee$, and $\neg$):

- $M, s \models \mathbf{A}f$ if f holds on every path that begins at state s.
- $M, s \models \mathbf{E}f$ if f holds on at least one path that begins at state s.
- $M, \pi \models \mathbf{X}f$ if $M, s_2 \models f$
- $M, \pi \models \mathbf{F}f$ if f holds in some state in π.
- $M, \pi \models \mathbf{G}f$ if f holds in every state in π.
- $M, \pi \models f\mathbf{U}g$ if g holds in some state in π and, in all previous states, f holds.
- $M, \pi \models f\mathbf{R}g$ if g holds in either every state in π or until, and including the state where, f holds.

A CTL* formula is true in a model if it is true in all the start states (or in every state if no initial states are specified). CTL restricts the set of CTL* formulas to those in which each temporal operator is immediately preceeded by a path quantifier. The CTL semantics are often modified slightly to express properties along 'fair paths', ones in which a state from each designated subset appears infinitely often, rather than all the paths in a system. An LTL formula is a CTL* formula of the form **A**f where f does not contain a path quantifier.

Given a model, $M = (S, T, L)$, and a set of relational variables $Var = \{Q, Q_1, Q_2, \ldots\}$ where each Q_i ranges over sets of states, the μ-calculus formulas are constructed as follows:

- Every atomic proposition is a formula.
- Every relational variable is a formula.
- If f is a formula and a is a binary relation in T, then $[a]f$ and $\langle a \rangle f$ are formulas.
- If f is a formula and Q is a variable then $\mu Q.f$ and $\nu Q.f$ are formulas, provided that all occurrences of Q in f fall under an even number of negations.
- If f and g are formulas, then $\neg f$ and $f \wedge g$ are also formulas.

Given an environment e that maps random variables to sets of states, we can define inductively (omitting standard symbols $\neg$ and $\wedge$) what it means for a formula f to hold in a state s in M (written $M, s, e \models f$):

- $M, s, e \models p$ if p is in $L(s)$.
- $M, s, e \models Q$ if s is in $e(Q)$.
- $M, s, e \models [a]f$ if for every state s' such that $(s, s') \in T(a)$, $M, s', e \models f$.
- $M, s, e \models \langle a \rangle f$ if there is a state s' such that $(s, s') \in T(a)$ and $M, s', e \models f$.
- $M, s, e \models \mu Q.f (\nu Q.f)$ if the least (greatest) fixed point holds.

The naive model checking algorithms for the languages are fairly straight-forward. The key to CTL formulas is to evaluate which states satisfy the smallest sub-formulas first, then use these results to find the states that satisfy the next largest and then the next until the entire formula has been evaluated. A CTL* checker basically uses the same approach, except that some of the sub-formulas are LTL and must be evaluated using LTL checking techniques. The interesting part of the μ-calculus checker is the evaluation of fixed points. It turns out that, because S is finite and any μ-calculus formula is monotonic, the fixed points can be found through a simple, iterative process. Specifically, the least fixed point, $\mu Q.f$, can be determined through a series of approximations, $Q_1, Q_2, \ldots$, in which Q_1 is the empty set, Q_{i+1} is the evaluation of f when $Q = Q_i$, and the least fixed point is the first approximation that equals its predecessor. The greatest fix point can be found through the same procedure with $Q_1 = S$. The LTL checkers take a different approach, either by using automata theory or a tableau construction. For brevity, I'll only discuss the tableau strategy here.

For an LTL formula $\mathbf{A}f$ that is logically equivalent to a formula $\neg\mathbf{E}\neg f = \neg\mathbf{E}g$, the tableau approach constructs a graph to determine if $\mathbf{E}g$ is satisfied and then uses this information to decide if $\mathbf{A}f$ holds. More specifically, the checker first constructs the closure of g, $cl(g)$, which is the set of formulas whose truth value effects the truth of g. Then, the checker associates with each state, s, a set k_s that contains $L(s)$ and the maximal consistent set of formulas in $cl(g)$ that is also consistent with $L(s)$. A new graph, G, is constructed where the nodes are the (s, k_s) pairs and the edges correspond to the transitions (s, s') in R such that if a formula $\mathbf{X}h$ is in k_s then h is in $k_{s'}$. Finally, a state s satisfies $\mathbf{E}g$ iff g is in k_s and there exists a path in G from (s, k_s) to a self-fulfilling strongly connected component (a single node with a self-loop or a set of nodes where every node is reachable from every other node and if any node has a k that contains a formula $h_1 U h_2$ then some node in the component has a k that contains h_2).

The efficiency of the basic model checking algorithms can be improved in a number of ways. As previously stated, the computation time can be decreased by changing the algorithms to use OBDDs. One can also try to find an equivalent, smaller model. This can be done by exploiting any symmetry in the system and removing data that is not relevant to the checked property. It may also be possible to replace actual system data by abstract values and collapse the model accordingly. For example, a model of a traffic light may make a distinction between a yellow and red light, where the property may not need to differentiate between the two 'stop' signals. Often concurrent systems are represented by a model in which every possible interleaving is considered. If the property does not distinguish between some or all of the different orderings, then only one 'representative' ordering must be checked. In addition to finding a smaller model, a natural strategy is to split the model into independent parts. For example, we may be able to verify that a server is operating correctly without considering the network or the client. When interdependencies between sections of the model exist, we may make assumptions about one component, that we later verify, to check a property of another.

Contents

Chapter 1, **Introduction**, is a general overview of what model checking is, how it compares to other verification techniques, and how it can be optimized.

Chapter 2, **Modeling Systems**, explains how systems can be represented by formulas or Kripke structures and reminds us that a poor match between the model and the real system can lead to incorrect conclusions.

Chapter 3, **Temporal Logics**, defines the CTL, LTL, and CTL* logics, as well as the modifications to CTL for handling fair paths.

Chapter 4, **Model Checking**, presents basic algorithms for checking formulas in CTL, CTL with fairness, LTL, and CTL* .

Chapter 5, **Binary Decision Diagram**, defines BDDs, gives an algorithm for obtaining an OBDD from a BDD, and explains how Kripke structures can be expressed using OBDDs. The size of an OBDD can depend critically on the variable ordering used to construct it. Although some heuristics are mentioned (e.g. try to keep variables in the same subcircuit close together), simply checking that an ordering is optimal is NP-complete.

Chapter 6, **Symbolic Model Checking**, gives symbolic model checking algorithms (ones based on the manipulation of boolean formulas/OBDDs) for CTL, CTL with fair paths, and LTL. The CTL algorithm very roughly corresponds to converting the CTL formula to μ-calculus and doing the basic evaluation technique for that logic (as given in the **Overview** section). The LTL approach presented here still constructs a tableau, but fewer formulas are used. For example, the entire closure is not taken. An algorithm for finding witnesses and counterexamples (a path that proves a property is or is not satisfied) is presented. In addition, strategies are discussed for improving computational efficiency by handling the OBDDs 'in chunks', rather than as single, large structures.

Chapter 7, **Model Checking for the μ-calculus**, defines the propositional μ-calculus and presents a model checking algorithm for it.

Chapter 8, **Model Checking in Practice**, gives a brief overview of the Symbolic Model Verifier (SMV) tool. It then discusses how SMV was used to uncover errors and ambiguities in the cache coherency protocol for the IEEE Futerbus+ standard.

Chapter 9, **Model Checking and Automata Theory**, explains how both the model and the negation of an LTL formula can be represented as generalized Buchi automata (finite automata over infinite words), how two generalized Buchi automata (gba) can be combined to create a third that accepts their intersection, and how to determine if a gba does not accept any input. Putting these pieces together, an LTL formula holds in a model if taking the intersection of the model and the negation of the formula produces a gba that accepts the empty set. By using the property gba to guide the construction of the model gba, a counter-example (showing that the intersection is non-empty) can often be found before the entire model is constructed.

Chapter 10, **Partial Order Reduction**, shows that an LTL formula can be expressed without using the next time operator, **X**, iff the property cannot distinguish between two stuttering equivalent models (ones that only differ in their sequences of identically labeled states). Using this fact, the chapter presents algorithms that reduce the time needed for verification, by reducing the size of the model.

Chapter 11, **Equivalences and Preorders between Structures**, gives algorithms for determining if two structures are bisimulation equivalent (same CTL* formulas hold in both) or if one is a simulation (abstraction/generalization) of the other. If a model M' is a simulation of a model M, then any ACTL* formula (CTL* restricted to universal quantification) that holds in M', holds in M. This fact is used as the foundation for an ACTL* model checking algorithm which is in the same spirit as the LTL checker given in this review and refined in Chapter 6.

Chapter 12, **Compositional Reasoning**, shows how to decompose a model into inter-dependent parts. Each component is checked separately, based on assumptions about the rest of the structure. Once these assumptions are verified, we can infer if the property holds in the entire model without ever having to construct the (large) global, state-transition graph.

Chapter 13, **Abstraction**, presents two approaches to reducing the model's size. The first determines which variables effect the truth of the property and removes the others from the model.

The second finds a mapping from the actual data to a smaller set of abstract values (e.g. map x = 7 and y = 9 to x = y = odd).

Chapter 14, **Symmetry**, uses group theory to derive an equivalence relation on the model's state space. This relation is used to construct a bisimulation equivalent model, called a quotient model, that is usually smaller, and hence easier to check, than the original.

Chapter 15, **Infinite Families of Finite-State Systems**, looks at the problem of verifying a property for an infinite set of finite systems. Although the problem is, in general, undecidable, a number of techniques have been developed for particular families. Most of these rely on finding an invariant such as a model that is bisimulation equivalent to all the models in the family.

Chapter 16, **Discrete Real-Time and Quantitative Temporal Analysis**, presents an extension of CTL, called RTCL, which associates the operators **EU**and **EG**with time intervals. Also, naive algorithms are given for determining the minimum and maximum delays in a system. The chapter concludes with a discussion on how these algorithms and the model checking techniques discussed thus far were used to verify the correctness, both functional and timing, of an aircraft controller.

Chapter 17, **Continuous Real Time**, shows how continuous real time systems can be represented as timed automaton and references a number of papers that propose model checking algorithms for these structures. (A timed automaton is a finite automaton augmented with a set of real-valued clocks. Each state has a set of clock constraints that must be met for the automaton to be in that state. Time can elapse 'in a state', but transitions occur instantaneously. Finally, some, or all, of the clocks can be reset when a transition occurs.)

Chapter 18, **Conclusion**, gives several directions for future research including making model checking easier for engineers to use, finding more concise representations for the models and properties, exploring probabilistic verification, determining performance measurements rather than only checking if a property holds, and combining model checking techniques with theorem proving to get the best of both worlds.

Opinion

The book was designed to be both an introductory text and a reference for researchers. To accommodate the novice, no previous knowledge about model checking is assumed. In fact, foundational ideas such as automaton and group theory are briefly reviewed. If the reader is unfamiliar with an area, however, the review will not be enough to fully understand the relevant chapter(s). For the theoretician, the book contains most of the theorems and proofs needed to convince him/her that the various approaches are correct. The systems researcher and the practitioner are not ignored either. There is a significant amount of discussion on the protocols used in model-checkers and the results of applying these to practical, verification problems. When a proof is omitted or the details of a topic are not fully discussed, the book recommends additional reading, namely the original research papers. The reference section, which contains 253 entries, makes this book an excellent 'first-stop' for people who want a thorough understanding of a specific aspect of model checking.

I have only two recommendations for future editions. First, although examples are worked-out in the text, adding exercises at the end of the chapters would help students test their understanding of the presented techniques. Second, the 'flow' of the book is not smooth. Specifically, some of the chapters cover much more information in much more depth than others. An attempt is given to have every chapter begin with an overview, but for some chapters this is a paragraph and for others it is several pages including definitions and proofs. Also, the book continually jumps from one logic to another and back again, rather than discussing all the techniques for one logic and then switching to another.

I strongly recommend this book to researchers looking to begin or further their understanding of model checking.

Review of
Petri Net Algebra [3]
by Eike Best, Raymond Devillers, Maciej Koutny
Springer Verlag, 2002, 378 pages
EATCS Series: Monographs in TCS

Reviewer: Boleslaw Mikolajczak (`bmikolajczak@umassd.edu`)

1 Introduction

Many contemporary computing systems are concurrent and distributed. Many approaches to concurrency formalization have been introduced over last forty years. This monograph combines two theories of concurrency: process algebras and Petri nets. One can hope that combining several competing theories that are mutually complementary can produce a formalism that will be practically useful. This is the primary motivation of the book. The book investigates structural and behavioral aspects of these two models. It also shows strong equivalence between these two models. Process algebras have the following advantages: allow study of connectives directly related to actual programming languages, are compositional, come with a variety of concomitant and derived logics that support reasoning about important properties of systems, and come with variety of algebraic laws that help in systems' refinement and in proving corrctness. Petri nets clearly distinguish local states and local activities of a system (places and transitions, respectively), global states and activities can be derived from basic concepts, Petri nets are graphical, and as bipartite graphs have strong links to graph theory and linear algebra. These last two facts can be used for the verification of systems.

2 Contents

Chapter 1, Introduction

It presents motivation to study jointly process algebras and Petri nets. It also overviews the content of each chapter.

Chapter 2, The Petri Net Calculus

This chapter introduces process algebra, also called as Petri Net Calculus (PBC), in an informal way. PBC combines a number of features taken from other process algebras, such as COSY (COncurrent SYstems), CCS (Calculus of Communicating Systems), SCCS (Synchronous CCS), CSP (Communicating Sequential Processes), TCSP (Theoretical CSP), and ACP (Algebra of Communicating Processes). PBC has been designed with two objectives: to support a compositional Petri net semantics and to provide a basis for developing a compositional semantics of high-level concurrent specification and programming languages. Main topics, developed formally later in the text, are presented using simple examples. In particular the following issues are explained: the operations of PBC (sequential composition, parallel composition, synchronization), PBC semantics of synchronization, and how PBC can model a concurrent programming language.

[3]© Ivelin Ivanov

Chapter 3, Syntax and Operational Semantics

This chapter introduces the syntax and operational semantics of the Petri Box Calculus. The basic version and extensions of PBC are discussed. Standard PBC syntax includes static expressions, dynamic expressions, and recursion. Structured operational semantics (SOS) is an approach to defining the set of possible moves of process expressions called also an operational semantics of process algebra. A SOS consists of a set of axioms and derivation rules from which behaviors of its expressions can be derived. The question of whether two concurrent systems can be regarded as behaviorally equivalent is present in many places and formulated in several different ways. Behavioral congruence is used instead of behavioral equivalence, i.e. as a behavioral equivalence that is preserved for each PBC operator. Isomorphism is used as the strongest version of transition systems behavioral equivalence. Another form of strong equivalence of transition systems is called a strong bi-simulation that works for non-isomorphic systems. Parallel composition, choice composition, sequential composition, and several forms of synchronization (standard PBC synchronization, auto-synchronization and multilink-synchronization, step synchronization) are also introduced in a formal manner. Basic relabeling, restriction, scoping, iteration, and recursion are another operations discussed in this chapter. Three kinds of extensions of the PBC syntax have been defined: several generalized iterations, data variables, and generalized control flow and communication interface operators.

Chapter 4, Petri Net Semantics

The aim in this book is to obtain a model in which it is possible to apply both Petri nets (S-invariant-based structural analysis of nets) and process algebra specific analysis techniques (structural operational semantics together with related behavioral equivalencies). In order to take advantage of both kinds of techniques an expressive framework of composing Petri nets based on transition refinement has been adopted. This means that each operator on nets would be based on a Petri net Ω whose transitions $t_1, t_2, \ldots$ are refined by corresponding nets $\Sigma_1, \Sigma_2, \ldots$ in the process of forming a new net $\Omega(\Sigma_1, \Sigma_2, \ldots)$. Being able to compose nets is not enough; we need to preserve semantic properties. To be able to use S-invariant analysis technique, it ought to be the case that $\Omega(\Sigma_1, \Sigma_2, \ldots)$ is covered by S-invariants provided that the nets $\Sigma_1, \Sigma_2, \ldots$ were. It happens that the net operators described in chapters 2 and 3 preserve S-coverability and admit structured operational semantics. Extension beyond this set of operators may not lead to these desired properties. As a result of this observation the book essentially undertakes two separate studies of Petri net compositionality: S-compositionality (oriented toward S-invariant analysis) and SOS-compositionality (oriented toward structured operational semantics). However, S-compositionality and SOS-compositionality share a significant number of properties. These common definitions and properties are presented in chapters 4 and 5, and are called as general compositionality. These nets whose interfaces are expressed by the labeling of places and transitions are called labeled nets and boxes. Equivalencies of these nets are defined in terms of step reachability graphs. Net refinement is defined in terms of transition refinement and interface change defined by relabelings. The operation of net refinement is carried out by taking two nets Ω and Σ and creating a new labeled net, $\Omega(\Sigma)$ by first applying to every transition of Σ an interface change specified by relabeling of the corresponding transition of Ω. Net refinement is then applied to associate plain boxes with PBC expressions discussed in chapters 2 and 3.

Chapter 5, Adding Recursion

This chapter continues the discussion of Petri net semantics of PBC-like algebras by developing a theory of associating nets with process variables defined by systems of recursive equations. The standard methods of using fixpoints and limit construction are applied to obtain solutions of such equations. It is also shown that if they are applied to Petri nets it is necessary to use the tree device for naming places and transitions in net refinement. First an ordering relation on labeled nets is introduced creating a domain in which solutions are sought. Subsequently a theory is developed for solving the general kind of recursive equations in the domain of labeled nets. Two special instances of this theory are considered: finitary equations and the case when operator box is finite. Several general examples are presented such as unbounded parallel composition, rear-unguardedness, concurrency within unbounded choice and extreme unguardedness. Finally, the complete development of the solvability of systems of equations is presented.

Chapter 6, S - Invariants

This chapter deals exclusively with S-compositionality. First the S-invariants are defined and their relation to safeness, boundedness, absence of concurrency and auto-concurrency, cleanness and isomorphism-based net equivalences, are discussed. Then the synthesis problem for net refinement is considered, i.e. the construction of an S-invariant of a refined net from S-invariants of the operator and operand boxes. Conditions under which the result of net refinement is static or dynamic box are formulated. Similar issues are considered for nets obtained as maximal solutions of systems of recursive expressions. Finally partial order semantics of Petri nets is introduced and infinite occurrence nets of boxes defined through net refinement and recursive systems of net equations are investigated.

Chapter 7, The Box Algebra

This chapter deals exclusively with SOS-compositionality. Due to the generality of the developed framework and the way in which refinements and solutions of recursive systems in the box domain were treated, it is possible to: a) define a general scheme to construct an algebra of boxes b) define a process algebra for which the denotational semantics is given homomorphically by the box algebra c) define a concurrent operational semantics equivalent to the denotational one.

First a class of operator boxes which yield plain boxes with step sequence semantics obeying SOS operational rules is identified. Then it is shown that SOS-operator boxes give rise to compositional nets for which a variant of the SOS semantics can be both defined and proved to be fully consistent with the standard net semantics. Finally a development of algebra of process expressions whose operational semantics is derived from that satisfied by the SOS-operator boxes, is presented. The resulting model, called the box algebra, allows introduction of the standard structured operational semantics and a denotational semantics in terms of plain boxes.

Chapter 8, PBC and Other Process Algebras

The theory developed in chapter 7 can be applied to various specific algebras such as CCS, TCSP, and COSY, especially in terms of semantic properties of the above algebras. This chapter relates PBC to other process algebras such as CCS, TCSP, and COSY. It is argued that PBC can be seen as an instance of box algebra.

Chapter 9, A Concurrent Programming Language

In this chapter the Petri Box Calculus (PBC) is used to give semantics of a concurrent programming language. An example language called Razor is used. This is imperative language that supports both shared data parallelism (concurrent processes operate on a common set of variables) and buffered communication (processes communicate through shared buffers of arbitrary capacity). Razor is an extension of Occam and Dijkstra's guarded command language. First syntax of Razor is introduced. Later its semantics is defined in terms of PBC, and implicitly in terms of Petri nets. In fact a mapping is defined that associates with every Razor program a PBC expression and a net. This mapping is compositional, i.e. it possess the homomorphism property of mapping operators of the programming language to operations on PBC expressions and operator nets. This mapping is illustrated by several examples. Later an extended version of Razor with procedures is introduced.

Finally the general theory developed in previous chapters is used to infer properties of the semantics of Razor. An important goal of compositional translation is to create a usable framework for making Petri net specific methods readily available. Correctness proofs of three distributed mutual exclusion algorithms (Peterson, Dekker, and Morris) are discussed as examples. In addition, an automatic verification of the mutual exclusion property using PEP partial order-based model checker (see http://theoretica.informatik.uni-oldenburg.de/ pep), has been performed. The model checker translates a program automatically into a net, calculate the MacMillan finite prefix of that net, and executes Esparza's model checker with given input formula of the associated temporal logic.

Chapter 10, Conclusion

The contents of this book have been developed by several authors during nineties as part of two European Union funded Basic Research Action Project DEMON (Design Methods based on Nets) and CALIBAN (Causal Calculi based on Nets). These investigations were triggered by many factors such as the needs in the translation of a realistic concurrent programming language and the concept of compositionality. The last idea has developed into matching the operators of process algebra with operator nets, thus resulting in using nets both as objects and as functions. The compositionality appears in the book in a form of general methods for deriving S-invariants, S-components, and S-aggregates for nets which are constructed modularly from the corresponding invariants of their components. This has been done even after allowing infinite nets, unguarded recursion, and non-pure operator boxes. Another example of compositionality is that of behavior, i.e., whether or not the behavior of a composed object is the composition of the behavior of its components. Finally, it has been shown in the book that the denotational semantics of a full programming language (including shared variables, channels, parallelism, and recursion) can be given compositionally.

3 Opinion

The book can serve researches and practitioners working in concurrency theory or in formalization of parallel and distributed systems. It can also be used in an advanced graduate course as a textbook or as a reference. The book contains numerous examples and exercises included in the text immediately following the relevant material. Some of them are solved in the appendix at the end of the book. Many concepts, theorems, counterexamples, and ideas are illustrated by figures that facilitate understanding and improve readability. Some of them are solved in the appendix. Three researchers have written the book. However, it is well organized and coherent both in structure and content. A reader has a good sense of text continuity. Notation and terminology is unified and coherent. References to literature are included at the end of each chapter with some commentary. This is a theoretical book concerning two concurrency models. Do not expect hints with direct practical implications and usefulness. A design of concurrent programming language in chapter 9 is a culmination of the book.

Review of

Combinatorial Optimization - Theory and Algorithms [4]

by Bernhard Korte and Jens Vygen

Published in 2000 by Springer Verlag, $45.00, 540 pages

Reviewer: Ivelin Ivanov

1 Overview

Rarely do mathematical disciplines have so direct practical relevance as combinatorial optimizations, which is why it is one of the most active areas of discrete mathematics. It became a subject in its own right only about 50 years ago, which makes it one of the youngest also.

This book covers most of the important results and algorithms achieved in the field to date. Most of the problems are formulated in terms of graphs and linear programs. The book starts with reviewing basic graph theory and linear and integer programming. Next, the classical topics in the field are studied: minimum spanning trees, shortest paths, network flows, matching and matroids.

Most of the problems in Chapters 6-14 have polynomial time (efficient) algorithms, while most of the problems studied in Chapters 15-21 are NP-hard, i.e. polynomial time algorithm is unlikely to exist. Although in many cases approximation algorithms are offered which at least have guaranteed performance.

Some of the topics include areas which have developed very recently, and which have not appeared in a book before. Examples are algorithms for multicommodity flows, network design problems and the traveling salesman problem. The book also contains some new results and new proofs for previously known results.

The authors have an unique approach in the presentation of the topics. They provided detailed proofs for almost all results, including deep classical theorems (e.g. weighted matching algorithm and Karmarkar-Karp bin-packing algorithm) whose proofs are usually sketched in previous works.

2 Content Summary

The first 10 chapters are on elementary material. Chapters 1 introduces the reader to the field of Combinatorial Optimization and defines some basic terminology. Chapter 2 is an introduction to Graph theory which provides basic definitions used throughout the rest of the book. Chapters 3-5 introduce the field of Linear and Integer Programming. Chapters 6-10 discuss classical problems like spanning trees, shortest paths, network flows, minimum cost flows and maximum matchings.

Chapter 11 studies the topic of weighted matching resulting in efficient solution $O(n^3)$ of the optimal job assignment problem and Chapter 12 looks at its generalization. The problem is informally described as: There is a set of jobs each of which having a certain processing time. There is also a set of employees each being able to complete a subset of the jobs. All employees are equally efficient. Several employees can work on a job at the same time and an employee can work on several jobs (although not at the same time). The goal is to assign tasks to the employees so that the jobs get done soonest. This is a cornerstone problem in combinatorial optimizations with numerous applications. Worth mentioning is its potential help in an area which is becoming increasingly popular recently computing clusters. Having a number of computers, which can perform tasks with certain efficiency, how to optimally assign a set of processes scheduled for execution.

[4]© Ivelin Ivanov

The weighted matching algorithm is particularly interesting in the case when the computers participating in a cluster are servers dedicated to repeatedly performing a well known set of processes such as business transactions. Additionally Chapter 11 offers an attractive result allowing for minor recalculations $O(n)$, when there are partial changes in the cluster configuration (e.g. server failure or adding/replacing servers).

Chapter 13 introduces the concept of Matroids and provides generalization of results discussed in earlier chapters. Matroids are taken into special consideration mainly because a simple greedy algorithm can be used for optimization over matroids.

Chapter 15 discusses the subject of NP-Complete problems. Even though for many combinatorial optimization problems polynomial time algorithms are found, there are also many important problems for which no polynomial-time algorithm is known. A classic NP-hard problem is the traveling salesman problem. It states that given a set of cities and the distances between each two, a salesman should start from a given city and visit all the other exactly once before returning to the city of origin. The salesman should do so choosing the shortest path possible. Although Ch. 15 does not prove that no efficient algorithm exists for NP-hard problems, it shows that one efficient algorithm would imply a polynomial time algorithm for all NP-easy problems. The authors use the Turing machine model to define precisely the concept of polynomial time algorithm and formally defend their claims.

Chapter 16 introduces approximation algorithms which offer polynomial time algorithms for NP- hard problems with known deviation from the optimal solution. More precisely the authors define the term performance ratio. An algorithm is of performance ration k if it provides solutions which are no more than k times worse than the minimum solution, in the case of minimization problems, and no more than k times less than the maximum, in case of maximization problems. Two major examples are studied edge coloring and vertex coloring. The authors prove that NP-hard problems have efficient algorithms only if P=NP. Additionally they give the reader tools to cope with NP-hard problems, by showing how one can reduce a given problem to a known NP-hard problem for which an approximation algorithm is available.

Chapter 18 opens the area of the bin-packing problem. It asks that a number of objects each with given size are assigned to the least possible number of bins all of which are of an equal known size. Chapter 21 describes the best known to date approximation algorithm for the traveling salesman problem solving instances with several thousand cities.

3 Conclusion

This comprehensive textbook on combinatorial optimization, with rigorous style, puts special emphasis on theoretical results and algorithms with provably good performance, in contrast to heuristics. It is known to be the basis of several courses on combinatorial optimization at graduate level. To mention some: MIT, TU Berlin, ETH Zurich, West Virginia University and University of Bonn among others. The authors suggest that the complete book contains enough material for at least four semesters (4 hours a week). The book contains complete (but concise) proofs, also for many deep results, some of which did not appear in a book before. Many very recent topics are covered as well, and many references are provided. This book represents the state-of-the-art of combinatorial optimization.

I would like to thank the co-author Jens Vygen for his sincere cooperation during my work on this review.

Review of **Introduction to Distributed Algorithms** [5]
Author of Book: Gerard Tel
Cambridge University Press, 2001, 608 pages
Review by Marios Mavronicolas, University of Cyprus,mavroni@cs.ucy.ac.cy

1 Overview

This book presents a view of several important topics in the modern field of distributed algorithms. The book is intended to provide useful background and reference information for professional engineers dealing with distributed systems, and for entering researchers who wish to develop a sufficient background in distributed algorithms. In addition, the book is suitable for providing a textbook-style introduction to distributed algorithms for advanced undergraduates or early graduate students who are taking a relevant course or consider pursuing research in the field.

The reference sources for this book incluude mainly research papers on the modern theory of Distributed Computing that routinely appear in conferences such as the ACM Symposium on Principles of Distributed Computing (PODC), or the International Symposium on DIStributed Computing (DISC). Fundamental milestones of this theory include also research papers from other major conferences of Theoretical Computer Science that are of wider interest, such as the ACM Symposium on Theory of Computing (STOC), or from leading theory journals, such as the Journal of the ACM or SIAM Journal on Computing.

2 Book Contents and Features

The book is divided into four major parts. Preceding these parts is an introductory chapter, namely Chapter 1 (Introduction: Distributed Systems), that talks about distributed systems, a communication and computation platform over which distributed algorithms are running, and prepares the reader to want to hear more about such systems.

Part One, called Protocols, consists of Chapters 2 (The Model), 3 (Communication Protocols), 4 (Routing Algorithms), and 5 (Deadlock-free Packet Switching). Overall, these four chapters cover fundamental concepts and primitives that are used in the remainder of the book. Particularly significant for conveniently reading the rest of the book is Chapter 2 that discusses (among other things) the formalization of a message-passing system as a transition model, additional assumptions on network topology, communication channels, and real-time behavior of processes, and the concepts of safety and liveness. Chapter 3 (Communication Protocols) discusses two particular protocols that are used for the reliable exchange of information between two computer hosts, a sender and a receiver. Most importantly, this chapter demonstrates the use of invariants for proving correctness of a distributed algorithm and techniques for reasoning about and analyzing the (time or message) complexities of such algorithms. Chapter 4 (Routing Algorithms) discusses how routing is possible by use of routing tables, and according to which criteria one may analyze the performance of a routing algorithm. A nice feature of this chapter is the discussion of hierarchical routing algorithms, which partition a distributed network into connected clusters and distinguish between routing within the same cluster, and routing between two different clusters; the chapter advocates that such algorithms may reduce the number of routing decisions necessary. Chapter 5 (Deadlock-free Packet Switching) considers a situation in store-and-forward routing, where a group of packets can never reach their destination in a distributed network because they are all waiting

[5]©Marios Mavronicolas, 2002

for the use of a buffer currently occupied by a packet in the group; the chapter treats the use of the so called controllers as a mechanism for preventing such deadlocks from occurring.

Part Two, called Fundamental Algorithms, includes Chapters 6 (Wave and Traversal Algorithms), 7 (Election Algorithms), 8 (Termination Detection), 9 (Anonymous Networks), 10 (Snapshots), 11 (Sense of Direction and Orientation), and 12 (Synchrony in Networks). Chapter 6 (Wave and Traversal Algorithms) is devoted to message-passing algorithms for dissemination of information over a network; the importance of this chapter lies in the fact that more advanced tasks to be solved in a distributed system often reduce themselves to appropriate tasks of information dissemination and discovery The problem of electing a leader in a network is considered in Chapter 7 (Election Algorithms), where lower and upper bounds are presented (among other things) on the number of messages needed to solve such a task. Another significant problem of Distributed Computing is Termination Detection, namely how to enable processes in a distributed system figure out that a particular distributed computation has entirely terminated; this is discussed in Chapter 8 (Termination Detection). Chapter 9 (Anonymous Networks) considers anonymous networks, that is networks over which processors and/or processes do not bear any distinct feature (such as process id) that may uniquely identify it. The main question addressed in this chapter is which problems solvable on named networks are also solvable on anonymous networks. Chapter 10 (Snapshots) considers the very basic problem of taking snapshots in a distributed system, namely computing and storing a single configuration of a distributed system that is generated during the course of a particular distributed computation; as Chapter 10 explains, this turns out to be a very hard task. Chapter 11 (Sense of Direction and Orientation) is one of the most interesting chapters of the book; it discusses how labeling the links of a distributed network with their direction may affect the possibility and the complexities of solving particular problems, such as broadcasting and leader election (see Chapter 7). Finally, Chapter 12 (Synchrony in Networks) investigates how the theory of Distributed Computing is affected by the assumption that there exists a global time available to all processes of a distributed system. Of particular interest are the descriptions for network synchronizers, which are abstract mechanisms used to implement the illusion of global time over networks that are completely asynchronous and do not provide to processors access to (any sort of) global time.

Part Three, called Fault Tolerance, includes Chapters 13 (Fault Tolerance in Distributed Systems), 14 (Fault Tolerance in Asynchronous Systems), 15 (Fault Tolerance in Synchronous Systems), 16 (Failure Detection) and 17 (Stabilization). Chapter 13 (Fault Tolerance in Distributed Systems) motivates the need for using fault-tolerant distributed algorithms, and introduces two particular types of fault-tolerant distributed algorithms, namely robust algorithms and stabilizing algorithms. The material in Chapter 14 (Fault Tolerance in Asynchronous Systems) is arranged around the fundamental FLP impossibility result, due to Fischer, Lynch and Paterson, stating that (roughly speaking) it is impossible to get all processes to agree on a common value in a distributed system if even a single process in the system may fail by crashing in the middle of an execution (and not taking computation steps any further). Chapter 15 (Fault Tolerance in Synchronous Systems) considers a synchronous distributed system, where impossibility results holding for asynchronous systems may no longer apply. The natural question that comes into mind then concerns the inherent costs to solve tasks that are solvable in such systems (but probably not in the asynchronous systems), and also to see if the synchrony assumption permits the occurrence of faults more severe than crash faults (e.g., Byzantine faults) without limiting the class of solvable problems. Chapter 15 surveys some very elegant answers to these natural questions in the context (mainly) of the broadcasting, consensus (getting all processes to agree) and clock synchronization problems. The very short Chapter 16 (Failure Detection) provides a glimpse at the very important problem of detecting the occurrence of failures in a distributed system; it turns out that the accuracy by which it is possible to detect such failures affects critically the possibility or impossibility of solving consensus in such systems (cf. Chapter 14 for the general impossibility of solving consensus if failure detectors are not available and the system is completely asynchronous). Finally, Chapter 17 (Stabilization) discusses the design of algorithms that are guaranteed to work correctly even if their execution starts from any arbitrary configuration of the distributed system.

Finally, Part Four (Appendices) of the book contains two useful appendices on pseudocode conventions, and on graphs and networks, respectively.

Each chapter is concluded with a list of interesting exercises and small projects. A list of answers (some partial) to most of them is available from the authors to instructors using this book as a textbook.

3 Opinion

The book concentrates on the message-passing (network) model of Distributed Computing, and makes a very good selection of topics related to this model to present. Some of these topics (e.g., sense of direction and orientation, compact routing, etc.) appear in textbook form for the first time (to the best of my knowledge) and in a very stylized way. This feature is definitely a pros for the book; another pros is the interleaving of formal discussion with prose English that describes the concepts and motivates the subsequent definitions in a remarkably nice and natural way along the text. For these reasons, I would recommend the book to all those who wish to specialize in the topics treated, or wish to acquire a broad and solid range of knowledge in the theory of Distributed Computing. However, a possible cons of the book is that it almost completely neglects the standard asynchronous shared memory model of distributed computation; as a consequence, several important topics that have been studied within the framework of this model (e.g., mutual exclusion algorithms for asynchronous shared memery), and have even contributed algorithmic paradigms to the entire field of Distributed Computing, could not fit into the book.

In conclusion, the book provides a very decent introduction to significant topics of the theory of Distributed Computing, and it is especially suitable as both a reference book and a textbook.

Review of
Calculated Bets [6]
by Steven Skiena
Published in 2001 by Cambridge University Press, $12.60, 262 pages

Reviewer: William Gasarch

Very few of us can turn our non-mathematical hobbies into papers or books. I doubt that Steven Rudich will get a paper out of his ability to do magic tricks, even though he is quite good at it (I've seen him at CRYPTO 01 and some DIMACS workshops. Catch his act!) I doubt I will get a paper from collecting Novelty Songs (Check out my website of *Satires of The 12 Days of Christmas* which gets far more hits then my website of *Applications of Ramsey Theory to Computer Science*). Steven Skiena has managed to get a book out of his hobby. His hobby is watching and gambling on the sport of Jai-Alai.

The book has three threads of information (1) the sport of Jai-Alai and how one gambles on it, (2) the mathematics underlying sports and gambling, (3) the story of Steven Skiena's interest in mathematics and in sports and how they have come together at various times in his life. All

[6]© William Gasarch

three are interesting, though of course the mathematics in it is why it is getting reviewed in this column. The book is well written and interesting. Even the non-math parts should interest anyone who picks it up. The math is well explained.

The scoring system in Jai-Alai is unusual. Initially 8 players are ordered in a queue $(p_1, \ldots, p_8)$. The top two in the queue play. The winner gets a point and stays at the head of the queue. The loser goes to the end of the queue. The first player to get 7 points wins. A player's chances of winning is determined partly by his initial post position. His abilities also play a part. Which is the bigger factor? Can we use the inequity in post position to help us gamble on the game?

These types of questions, and others, are raised and answered. While raising them he takes the reader through a tour of much mathematics of interest, mostly within probability, statistics, mathematical modeling, and computer programming. He also tries (with moderate success) to actually use his system to bet and win. His attempt will wake people up to how real world constraints can get in the way of an elegant mathematical theory.

This book *should* have a wide audience. A high school student who likes mathematics and sports will find much here of interest, though some of the math may be over her head. An undergraduate who likes math should like it even if she doesn't like sports. Even a grad student or professor would benefit from it since, even though the math is easy, some of it is not well known. In addition the information on Jai-Alai and other non-math topics will be of interest.

If you find yourself teaching a math-for-non-majors course of some sort, this book might be ideal. The key is that all the math here is motivated by solving a real world problem, and that has a certain appeal.

Semantic Scholar profile for Edmund M. Clarke, with 4278 highly influential citations and 464 scientific research papers.Â Model Checking: MIT Press. Edmund M. Clarke, Orna Grumberg, Doron A. Peled. Computer Science. 1999. 452. 63. Cite. Save. Automatic verification of finite-state concurrent systems using temporal logic specifications. Edmund M. Clarke, E. Allen Emerson, A. Prasad Sistla.