**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`

I give the price as quoted on amazon; however, the reader would do well to consult www.bestbookbuys.com or bookpool.com.

In this column we review the following books.

1. **Automata and Languages: Theory and Applications** by Alexander Meduna. Review by James Rauff. This is a textbook in automata theory in the theorem-proof style.
2. **Control Flow Semantics** , by J. de Bakker and E. de Vink. Review by Riccardo Pucella. This book looks at semantics of imperative languages.
3. **External Memory Algorithms (proceedings of DIMACS Workshop, May 20-22,1998)** Edited by J.M. Abello and J.S. Vitter. Review by Gianluca Della Vedova. This is a collection of articles on external memory algorithms, where are used when there is too much memory to store in core.
4. **$\pi$ - Unleashed**. Authors: Jörg Arndt and Christoph Haenel. Review by David Marcus. This is a history of $\pi$ with an emphasis on the progress made in computing in quickly.
5. **Chaotic Elections!** by Donald Saari and **Mathematics and Politics** by Alan Taylor. Review by William Gasarch. These book looks at some of the mathematical problems that arise in elections and other political issues such as escalation.

**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 If you want more information about any of these books, again, feel free to email me. Reviewing a book is a great way to learn a field. I have personally reviewed books and then went on to use what I learned in my research.

1. *Modal Logic* by Blackburn, De Rijke, and Venema.
2. *Metamathematics of First-order Arithmetic* by Hajek and Pudlak.
3. *Algorithmic and Computational Robotics: New Directions* Edited by Donald, Lynch, and Rus.
4. *Computational Commutative Algebra* by Kreuzer and Robbiano.
5. *Computational and Geometric Aspects of Modern Algebra* Edited by Atkinson, Gilbert, Howie, Linton, and Robertson. (Proceedings of a workshop)
6. *The Clausal Theory of Types* by Wolfram.
7. *Learning Automata: Theory and Applications* by Najim and Poznyak. *Learning Automata: Theory and Applications* by Najim and Poznyak.

[1] © William Gasarch, 2001.

8. *Algorithmic Geometry* by Boissonnat and Yvinec.
9. *Parallel Processing and Parallel Algorithms: Theory and Computation* by Roosta.
10. *Mathematical Theory of Domains* by Stoltenberg-Hansen, Lindstrom, and Griffor.
11. *Derivation and Computation* by Simmons.
12. *Domains and Lambada-Calculi* by Amadio and Curien.

The following are DIMACS workshop books which are collections of articles on the topic in the title.

1. Constraint Programming and Large Scale Discrete Optimization.
2. Discrete Mathematical Problems with Medical Applications.
3. Discrete Mathematical Chemistry.
4. Randomization Methods in Algorithm Design.
5. Multichannel Optical Networks: Theory and Practice.
6. Advances in Switching Networks.
7. Mobile Networks and Computing.
8. Robust Communication Networks: Interconnection and Survivability.

**Review of**[2]
**Automata and Languages: Theory and Applications**
**Author: Alexander Meduna**
**Published by Springer-Verlag, 2000**
**\$149.00, 916 pages**
**Review by James Rauff, Millikin University, Decatur, Illinois**

## 1 Overview

Automata and Languages is a large work (915 pages) intended for use in a first course in the theory of computation. The topics in this book are developed in a strictly formal manner with a minimum of side commentary. Meduna adheres to a consistent format in his presentation. First, concepts are presented in the traditional definition-theorem-proof pattern, modified when necessary by the presentation of an algorithm. For example, in the sections discussing the various simplifications and normal forms of context-free grammars, we are first given a definition (e.g. Greibach normal form), then an algorithm for achieving the defined form. Next it is proved that the algorithm terminates and is correct. Finally, an example is presented which implements the algorithm or illustrates a theorem. The proofs are detailed and the algorithms fully specified. A unique feature of the book is a detailed development of a compiler for a new computer language, COLA. Automata and Languages contains numerous exercises ranging widely in difficulty, but no solutions are provided to any of them. Each chapter also poses a variety of programming projects building upon the theory presented. These include realizations of the algorithms given in the text.

[2]James Rauff Copyright 2001

## 2 Contents

The size of Automata and Languages results from the detail of Meduna's exposition rather than breadth of material. The book has five parts.

1. Part I Basic mathematical concepts.

Chapter 0 Sets, relations, graphs and proofs. The proof section offers a very cursory discussion of three proof techniques (contraposition, contradiction, and induction). In light of the careful detail in the rest the book, this section seems to be no more than a token addition.

Chapter 1 Basic notions of formal languages and grammars. Includes formal specification of COLA.

Chapter 2 Basic notions of automata, transducers and computability.

2. Part II Regular Languages.

Chapter 3 Regular expressions, finite state automata, equivalence of regular expressions and finite state automata.

Chapter 4 Pumping lemma for regular languages, closure properties, decidable problems.

3. Part III Context-free languages.

Chapter 5 CFGs, Chomsky and Greibach normal forms, pushdown automata.

Chapter 6 Pumping lemma, closure properties, decidable problems. Ogden's lemma appears in the exercises.

Chapter 7 Deterministic CFLs, linear and regular grammars. Regular grammars are viewed as special types of context-free grammars.

4. Part IV Beyond context-free languages.

Chapter 8 Turing machines, two-pushdown automata, linear-bounded automata, unrestricted grammars, context-sensitive grammars.

5. Part V Translations.

Chapter 9 Finite transducers, translation grammars, compilers. This chapter contains the complete construction of a compiler for COLA including scanner, parser, semantic analyzer, code generator, and optimizer

Chapter 10 computability, decidability, computational complexity, Church's thesis, undecidable problems. The undecidability of Post's correspondence problem appears here as an exercise, but is also given as an exercise in Chapter 2.

Automata and Languages also contains an index to the decision problems and the an index to the algorithms. The decision problem index does not indicate whether or not the listed problem is decidable.

## 3 Opinion

The strength of Automata and Languages lies, in my opinion, in the detailed specifications of algorithms and the detailed proofs of theorems. In supplying this level of detail, some hard content choices needed to be made. Most instructors will find a favorite result either missing (like the Myhill-Nerode Theorem) or relegated to the exercises (like Ogden's lemma). Some of the more popular topics in language theory (like Lindenmayer systems) are also missing from Automata and Languages. Also, for a book whose subtitle is Theory and Applications, there is precious little in the way of applications beyond compiler design. Nevertheless, a student who reads Automata and Languages with care will not fail to come away with a clear understanding of the algorithms or theorems presented. I stress care because Automata and Languages has a number of typographical errors that seriously impact upon the content of the text. Here is a sample:

1. p.29: the word "prefix" is mistakenly written where "suffix" is wanted.
2. p. 81: the word "uncountable" is mistakenly written where "uncomputable" is wanted.
3. p.272: in an example of a derivation, the incorrect string aaaSbbba is written where aaaSbbb is wanted.

I personally would not adopt this book as a text. The absence of any solutions to the exercises reduces its value to a student. Missing connections to modern computer science also reduce its potential interest. Also, I am not confident that one could expect a second- or third-year student to work carefully through the entire 900+ pages of Automata and Languages in a single term or semester. Finally, the typographical errors, and other minor lapses of detail (including a reference to a book which is not listed in the bibliography) would require an instructor using Automata and Languages to be especially vigilant. I do feel that a student would benefit from working through the detailed proofs and from the discussion of the COLA compiler, but neither of these positives would tilt the balance in favor of using the book in a regular theory of computation class. It would be well-suited , however, for a seminar-type class that was structured along the lines of the Moore method of mathematics instruction where the students are lead to discover and prove major results. Here the important results in the exercises could be brought out and proven during the normal flow of the development of the material

**Review of[3]**
**Control Flow Semantics**
**Published by MIT press, 1996**
**$75.00, 608 pages**
**Author: J. de Bakker and E. de Vink**
**Review by Riccardo Pucella Dept. of C.S., Cornell University**

I have to admit, I was looking forward to reviewing this book. It answered what was for me a 6-year old question. Six years ago, I was pursuing a Master's degree at McGill University. working on programming language semantics. My training in semantics was rather traditional: lambda calculus, functional programming languages, denotational semantics based on complete partial orders, etc. At the time, Franck van Breugel was visiting McGill, and I came across the fact

[3]copyright Riccardo Pucella 2001

that Franck was also working on semantics of programming languages, but on semantics based on metric spaces. For someone with an undergraduate background in mathematics, this was seriously intriguing. Unfortunately, I never got around to asking Franck about his work. This book is an answer to that question that never was.

The purpose of the book is to describe an approach to provide semantics to imperative languages with various types of control flow models, including concurrency. The approach handles both operational semantics and denotational semantics, all in a topological setting. (We will come back to this later.) The issue of relating the two semantics for any given language is a central theme of the approach.

I will provide in the next section an introduction to topological semantics. For now, let me say a word on the applicability of the approach. As stated, the main interest is in providing semantics to *imperative* languages.

Imperative programs can be thought of, for our purposes, as sequences of actions performed on a state. Typically, states are sets of variables, and actions include modifying the value of a variable in a state.

An important characteristic of imperative programs is that they embody the notion of a *computation step*: a program being a sequence of actions, it forces a sequence of intermediates states. Typically, the intermediate states are *observable*, meaning that one can observe something about that intermediate state, either by looking up the value of a variable, by witnessing an output operation, etc. When the intermediate states of a computation are observable, it becomes reasonable to talk about infinite (nonterminating) computations. (The classical example of this is of course an operating system, which at least theoretically is an infinite process; the main motivation for the topological approach, as we shall see, is to make sense of such infinite computations.)

Contrast this with functional languages, which are often used as motivating examples for the study of semantics. In a pure functional language, all infinite programs are equivalent, and in a traditional denotational semantics based on partial orders (à la Scott and Strachey [SS71]), every nonterminating program is mapped to $\bot$, the least element of the appropriate partial order. This is not helpful in a setting where we want to discuss observably different infinite behaviors.

We distinguish two kinds of imperative languages. The distinction between them is akin to the distinction between propositional and first-order logic.[4] *Uniform languages* are based on a primitive, abstract notion of action, combined with appropriate operators. For instance, a program may have the form $a; b; c; (d + a)$ where ; represents sequential composition and + a nondeterministic choice operator. The primitive actions $a$,$b$,$c$,$d$ are uninterpreted. The state is implicitly defined by the actions that have been performed. On the other hand, *nonuniform languages* have an interpretation associated with the actions; typically, as we mentioned, the state is a set of variables, and actions include modifying the value of a variable in a state.

To showcase the versatility of their approach, the authors study different languages. The main difference between the various languages, aside from the question of uniformity, is the *program composition operators* considered. The following groups of related operators are studied:

- The first group consists of operators including *sequential composition* and *choice*. The latter introduces nondeterminism in the framework, with suitable complications. Many versions of choice are investigated, including backtracking choice.

- The second group of operators consists of *recursion* and *iteration*. Such operators are required to get universality (in the computability theory sense).

[4]This analogy can be made precise when looking at dynamic logics, a family of logics for reasoning about programs in such languages [HKT00].

- The third group of operators includes *parallel composition* operators. Modeling such operators forces one to deal with issues such as deadlock, synchronization and communication. Languages with such operators include CSP [Hoa85] and CCS [Mil89].
- Related to the last group of operators, one may distinguish between *static* and *dynamic* configuration of processes.
- Finally, we can investigate issues of *locality* and *scope* of variables.

All in all, 27 languages are studied in the book, encompassing various features described above (and others, such as the kernel of a logic programming language). For each language, an operational semantics is given, along with a denotational semantics based on topological spaces. The relationship between each semantics is investigated. Here are the chapter titles, to give an idea of the breakdown of content: 1. Recursion and Iteration, 2. Nondeterminacy, 3. Variations, 4. Uniform Parallelism, 5. Unbounded Nondeterminism, 6. Locality, 7. Nonuniform parallelism, 8. Recursion Revisited, 9. Nested Resumptions, 10., Domain Equations and Bisimulation, 11. Branching Domains at Work, 12. Extensions of Nonuniform Parallelism, 13. Concurrent Object-oriented Programming, 14. Atomization, Commit, and Action Refinement, 15. The Control Flow Kernel of Logic Programming, 16. True Concurrency, 17. Full Abstractness, 18. Second-order Assignment.

In the next section, I summarize the first chapter of the book, to give a feel for the approach.

## Overview of topological semantics

Given $\mathcal{L}$ a collection of programs in a language, a *semantics* for $\mathcal{L}$ is a mapping $\mathcal{M} : \mathcal{L} \to \mathcal{P}$ taking a program $p$ to an element $\mathcal{M}(p)$ from a domain of meanings $\mathcal{P}$. The domain $\mathcal{P}$ should have enough mathematical structure to capture what we want to model. The study of semantics centers around the development of methods to specify $\mathcal{M}$ and associated $\mathcal{P}$ for a range of languages $\mathcal{L}$. We can distinguish essentially two ways of specifying $\mathcal{M}$:

**Operational** $\mathcal{O} : \mathcal{L} \to \mathcal{P}_O$, which captures the operational intuition about programs by using a transition system (axioms and rules) describing the actions of an abstract machine. This is the structural approach to operational semantics (SOS) advocated by Plotkin [Plo81].

**Denotational** $\mathcal{D} : \mathcal{L} \to \mathcal{P}_D$, which is *compositional*; the meaning of a composite program is given by the meaning of its parts. This is helpful to derive program logics, to reason about correctness, termination and equivalence. Also, in general, denotational semantics are less "sensitive" to changes in the presentation of a language.

Consider the following simple example, to highlight the difference between the two styles of semantics. Let $A$ be an alphabet, and $W$ the set of structured words over $A$, given by the following BNF grammar:

$$w ::= a \mid (w_1 \cdot w_2)$$

where $a$ is an identifier ranging over the elements of $A$. If $A = \{a, b, c\}$, then $(a\cdot(b\cdot a))$, $((a\cdot b)\cdot(c\cdot b))$, $(((a \cdot b) \cdot a) \cdot b)$ are structured words over $A$. We choose to assign, as the meaning of an element of $W$, its *length*. We derive both an operational and a denotational semantics to assign such a meaning to elements of $W$. We take $\mathcal{P}_O = \mathcal{P}_D = \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers). To define the operational semantics, we consider the slightly extended language $V = W \cup \{E\}$, where intuitively $E$ stands for the empty word. We define a transition system with transitions of the form

$(v, n) \longrightarrow (v', n')$ where $v, v' \in V$ and $n, n' \in \mathbb{N}$. (Such a transition "counts" one letter of the word $v$.) Let $\longrightarrow$ be the least relation satisfying the following inference rules:

$$\frac{}{(a, n) \longrightarrow (E, n+1)}$$

$$\frac{(v_1, n) \longrightarrow (v_1', n')}{((v_1 \cdot v_2), n) \longrightarrow ((v_1' \cdot v_2), n')}$$

$$\frac{(v_1, n) \longrightarrow (E, n')}{((v_1 \cdot v_2), n) \longrightarrow (v_2, n')}$$

We can define the operational semantics $\mathcal{O}$ by:

$$\mathcal{O}(w) = n \quad \text{if and only if} \quad (w, 0) \longrightarrow (v_1, 1) \longrightarrow \cdots \longrightarrow (E, n)$$

The denotational semantics $\mathcal{D}$ is much easier to define:

$$\begin{array}{rcl} \mathcal{D}(a) & = & 1 \\ \mathcal{D}(w_1 \cdot w_2) & = & \mathcal{D}(w_1) + \mathcal{D}(w_2) \end{array}$$

It is straightforward to show, by structural induction, that in this case the operational and denotational semantics agree (that is, they give the same result for every word $w \in W$).

Let us now turn to a somewhat more realistic example. Recall that there are two kinds of imperative languages we consider, uniform and nonuniform. Let's define a simple uniform language with a recursive operator. This example is taken straight from Chapter 1 of the book. The language, $\mathcal{L}_{rec}$, is defined over an alphabet $A$ of primitive actions. We assume a set of program variables $PVar$.

$$\begin{array}{lccl} (\mathit{Stat}) & s & ::= & a \mid x \mid (s_1; s_2) \\ (\mathit{GStat}) & g & ::= & a \mid (g; s) \end{array}$$

A *statement* $s$ is simply a sequence of actions; variables are bound to *guarded statements* $g$, which are simply statements that are forced to initially perform an action. When a variable is encountered during execution, the corresponding guarded statement is executed. A declaration is a binding of variables to guarded statements, and the space of all declarations is defined as $\mathit{Decl} = PVar \to \mathit{GStat}$. The language $\mathcal{L}_{rec}$ is defined as $\mathcal{L}_{rec} = \mathit{Decl} \times \mathit{Stat}$. We write an element of $\mathcal{L}_{rec}$ as $(x_1 \Leftarrow g_1, \ldots, x_n \Leftarrow g_n \mid s)$, representing the statement $s$ in a context where $x_1, \ldots, x_n$ are bound to $g_1, \ldots, g_n$, respectively.

The operational semantics is defined by a transition system over $\mathit{Decl} \times \mathit{Res}$, where $\mathit{Res} = \mathit{Stat} \cup \{E\}$; the intuition is that $E$ denotes a statement that has finished executing. We notationally identify the sequence $E; s$ with the statement $s$. This will simplify the presentation of the reduction rules. The transitions of the system take the form $s \xrightarrow{a}_D r$ where $s \in \mathit{Stat}$, $r \in \mathit{Res}$, $a \in A$, and $D \in \mathit{Decl}$ ; this transition should be interpreted as the program statement $s$ rewriting into the statement $r$, along with a computational effect $a$. (For simplicity the computational effect is taken to be the action performed.) Again, the $\xrightarrow{a}_D$ relation is the least relation satisfying the following inference rules:

$$\frac{}{a \xrightarrow{a}_D E}$$

$$\frac{g \xrightarrow{a}_D r}{x \xrightarrow{a}_D r} \qquad \text{if } D(x) = g$$

$$\frac{s_1 \xrightarrow{a}_D r_1}{s_1; s_2 \xrightarrow{a}_D r_1; s_2}$$

We take the domain $\mathcal{P}_O$ of operational meanings to be the set of finite and infinite sequences of actions, $\mathcal{P}_O = A^\infty = A^* \cup A^\omega$. We define the operational semantics $\mathcal{O} : \mathit{Decl} \times \mathit{Res} \to \mathcal{P}_O$ as:

$$\mathcal{O}(D \mid r) = \begin{cases} a_1 a_2 \cdots a_n & \text{if } r \xrightarrow{a_1}_D r_1 \xrightarrow{a_2}_D \cdots \xrightarrow{a_n}_D r_n = E \\ a_1 a_2 \cdots & \text{if } r \xrightarrow{a_1}_D r_1 \xrightarrow{a_2}_D \cdots \end{cases}$$

For instance, we have $\mathcal{O}(D \mid a_1; (a_2; a_3)) = a_1 a_2 a_3$, and $\mathcal{O}(x \Leftarrow (a; y), y \Leftarrow (b; x) \mid x) = (ab)^\omega$.
Deriving a denotational semantics is slightly more complicated. A program in $\mathcal{L}_{rec}$ may describe infinite computations. To make sense of those, we need the notion of the limit of a computation. In mathematical analysis, limits are usually studied in the context of metric spaces [Rud76]. This is the setting in which we will derive our semantics.

A *metric space* is a pair $(M, d)$ with $M$ a nonempty set and $d : M \times M \to \mathbb{R}_{\geq 0}$ (where $\mathbb{R}_{\geq 0}$ is the set of nonnegative real numbers) satisfying: $d(x, y) = 0$ iff $x = y$, $d(x, y) = d(y, x)$, and $d(x, y) \leq d(x, z) + d(z, y)$. A metric space $(M, d)$ is *$\alpha$-bounded* (for $\alpha < \infty$) if $d(x, y) \leq \alpha$ for all $x$ and $y$ in $M$. We can define a metric on $A^\infty$ as follows. For any $w \in A^\infty$, let $w[n]$ be the prefix of $w$ of length at most $n$. The Baire-distance metric $d_B : A^\infty \times A^\infty \to \mathbb{R}_{\geq 0}$ is defined by

$$d_B = \begin{cases} 0 & \text{if } v = w \\ 2^{-n} & \text{if } v \neq w \text{ and } n = \max\{k \ : \ v[k] = w[k]\} \end{cases}$$

We say a sequence $\{x_n\}_{n=1}^{\infty}$ is *Cauchy* if for all $\epsilon > 0$ there exists an $i$ such that for all $j, k \geq i$, $d(x_j, x_k) \leq \epsilon$. In other words, the elements of a Cauchy sequence get arbitrary close with respect to the metric. A metric space $(M, d)$ is *complete* if every Cauchy sequence converges in $M$. It is easy to check that the metric space $(A^\infty, d_B)$ is complete.

If $(M, d)$ is $\alpha$-bounded for some $\alpha$, and $X$ is any set, let $(X \to M, d_F)$ be the *function space* metric space defined as follows: $X \to M$ is the set of all functions from $X$ to $M$, and $d_F(f, g) = \sup\{d(f(x), g(x)) \ : \ x \in X\}$ ($\alpha$-boundedness on $M$ guarantees that this is well-defined).

One can check that if $(M, d)$ is complete, then so is $(X \to M, d_F)$. A central theorem of the theory of metric spaces, which is used heavily in the book, is Banach's fixed point theorem. Essentially, this theorem says that every function $f$ from a metric space to itself that decreases the distance between any two points must have a fixed point (a point $x$ such that $f(x) = x$). We need more definitions to make this precise. Define a function $f : (M_1, d_1) \to (M_2, d_2)$ to be contractive if there exists an $\alpha$ between 0 and 1 such that $d_2(f(x), f(y)) \leq \alpha d_1(x, y)$. For example, the function $f : (A^\infty, d_B) \to (A^\infty, d_B)$ defined by $f(x) = a \cdot x$ is $\frac{1}{2}$-contractive.

**Theorem (Banach)**: Let $(M, d)$ be a complete metric space, $f : (M, d) \to (M, d)$ a contractive function. Then

1. there exists an $x$ in $M$ such that $f(x) = x$,
2. this $x$ is unique (written $\mathit{fix}(f)$), and
3. $\mathit{fix}(f) = \lim f^n(x_0)$ for an arbitrary $x_0 \in M$, where $f^{n+1}(x_0) = f(f^n(x_0))$.

This is the basic metric space machinery needed to get a simple denotational semantics going. Returning to our sample language $\mathcal{L}_{rec}$, we take as a target of our denotational semantics the domain $\mathcal{P}_D = A^\infty - \{\epsilon\}$. (We do not allow the empty string for technical reasons. Notice that the empty string cannot be expressed by the language in any case.) What we want is a function $\mathcal{D}$ defined as follows:

$$\begin{aligned}\mathcal{D}(D \mid a) &= a\\ \mathcal{D}(D \mid x) &= \mathcal{D}(D \mid D(x))\\ \mathcal{D}(D \mid s_1; s_2) &= \mathbf{;}(\mathcal{D}(D \mid s_1), \mathcal{D}(D \mid s_2))\end{aligned}$$

(for some function **;** defined over $A^\infty$, meant to represent sequential composition, and to be defined shortly.) Notice that this definition of $\mathcal{D}$ is not inductive. We will use Banach's theorem to define the function **;** over $A^\infty$, and to define the function $\mathcal{D}$. Let us concentrate on **;**. Intuitively, we want **;** to be a function $A^\infty \times A^\infty \to A^\infty$ satisfying

$$\begin{aligned}\mathbf{;}(a, p) &= a \cdot p\\ \mathbf{;}(a \cdot p', p) &= a \cdot \mathbf{;}(p', p)\end{aligned}$$

Note that the above properties do not form an inductive definition of **;** due to the presence of infinite words in $A^\infty$. Instead, we will define **;** as the fixed point of the appropriate higher-order operator. Let $Op = A^\infty \times A^\infty \to A^\infty$ be the complete metric space of functions.[5] Define the following operator $\Omega_; : Op \to Op$:

$$\begin{aligned}\Omega_;(\phi)(a, p) &= a \cdot p\\ \Omega_;(\phi)(a \cdot p', p) &= a \cdot \phi(p', p)\end{aligned}$$

Note that the above equations *do* define a function $\Omega_;$. One can check that $\Omega_;$ is in fact a $\frac{1}{2}$-contractive mapping from $Op$ to $Op$. Therefore, by Banach's theorem, there exists a unique fixed point (call it **;**) such that $\Omega_;(\mathbf{;}) = \mathbf{;}$. It is easy to see that this **;** satisfies the original equations we were aiming for.

Now that we have such a function **;**, let us turn to the problem of actually defining $\mathcal{D}$. We proceed similarly, by defining $\mathcal{D}$ as the fixed point of the appropriate higher-order operator, through an application of Banach's theorem. Consider the metric space $Sem_D = \mathcal{L}_{rec} \to A^\infty$, which is complete since $A^\infty$ is complete. Define the following function $\Psi : Sem_D \to Sem_D$ by:

$$\begin{aligned}\Psi(S)(D \mid a) &= a\\ \Psi(S)(D \mid x) &= \Psi(S)(D \mid D(x))\\ \Psi(S)(D \mid s_1; s_2) &= \mathbf{;}(\Psi(S)(D \mid s_1), S(D \mid s_2))\end{aligned}$$

(There is some subtlety in coming up with the last equation; as you'll notice from looking at the righthand side, there is recursion over $\Psi$ in only one of the two cases. The book explains this.) Once again, we can show that $\Psi$ is a $\frac{1}{2}$-contractive function (in $S$), and thus by Banach's theorem there is a unique fixed point of $\Psi$ (call if $\mathcal{D}$) such that $\Psi(\mathcal{D}) = \mathcal{D}$. It is straightforward to check that this $\mathcal{D}$ satisfies our requirements for the denotational semantics function.

A final result of interest after all of these developments is the relationship between $\mathcal{O}$, the operational semantics based on an intuitive notion of computation, and $\mathcal{D}$, the denotational semantics with its compositional properties. It turns out that in this case, $\mathcal{O} = \mathcal{D}$, and moreover this result can be derived from a third application of Banach's theorem. The details can be found in the book.

[5]Strictly speaking, we need to consider the space of bounded functions to ensure that the space is complete. This will be irrelevant at our level of discussion.

## Opinion

As a technical book, aimed at describing an approach to provide semantics to a wide variety of imperative language control flow structures, this book is complete. All the examples are worked out with enough details to grasp the subtleties arising. Let the reader be warned, however, that the book is dense—both in terms of the technical material, and in terms of the presentation. The first few chapters should be read slowly and with pencil in hand.

The book does not require as much background knowledge of topology as one may expect. Prior exposure is of course beneficial, but in fact, only the basics of topology and metric spaces are actually used, and whatever is needed is presented in the first few chapters. On the other hand, the presentation does assume what may best be called mathematical maturity.

In the grand scheme of things, a problem with this book is one of motivation and followup. This is hardly new in the field of semantics. Specifically, the use of denotational semantics is hardly motivated, considering that most of the machinery in the book is aimed at coming up with denotational semantics and showing that it agrees with the intuitive operational semantics. There is a throw-away line about the fact that denotational semantics can help in developing logics for reasoning about programs, but most of the interesting developments are buried in the bibliographical notes at the end of the chapters. This will not deter the hardcore semanticist, but may have other readers go: "so what?". Sad, since denotational semantics *is* useful. And for the curious, Franck's actual work can be found in [vB98].

## References

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* The MIT Press, Cambridge, Massachusetts, 2000.

[Hoa85] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[Rud76] W. Rudin. *Principles of Mathematical Analysis.* McGraw-Hill, third edition, 1976.

[SS71] D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of the Symposium on Compute= rs and Automata*, New York, 1971. Polytechnic Institute of Brooklyn Press.

[vB98] F. van Breugel. *Comparative Metric Semantics of Programming Languages: Nondeterminism and Recursion.* Birkhäuser, 1998.

**Review of [6]**
**External Memory Algorithms (DIMAC Workshop, May 20-22,1998)**
**Edited by J.M. Abello and J.S. Vitter**
**Published by American Math Society, 1999, $75.00**
**Review by Gianluca Della Vedova, Dept. of Statistics, Univ. Milano-Bicocca**

## 1 Overview

This book is a collection of articles presented at the DIMACS Workshop on External Memory Algorithms and Visualization, held in 1998. All modern computer architectures are based on a hierarchy of storage devices, usually on at least three layers (cache memory, main memory and disk). While general-purpose operating systems have introduced the notion of virtual memory in order to allow the ordinary programmer to avoid dealing with such hierarchy, taking into account the different characteristics of each kind of memory may help to design faster algorithms.

Nowadays data set in large applications are easily too large to fit completely inside the computer's internal memory, consequently they must reside on disk, making accesses to data the actual bottleneck, as a single disk access is often 10000 times slower than a memory access. This gap is likely to widen, as (according to Moore's Law) CPUs double their speed every 1.5 years, while in the same amount of time disks increase their speed at most by 20%.

This situation calls for specific algorithmic solutions for a number of problems; this book presents a glimpse of the current state of research in that field.

## 2 Contents

The first Chapter is an extensive (38 pages) introduction to the subject of external memory algorithms, beginning with the parallel disk model, which is a simple and appropriate mathematical abstraction of an internal vs. disk memory hierarchy. Other alternative abstractions are briefly presented. Successively a number of problems and algorithms are presented, pointing out that a problem which can be solved efficiently in main memory can be harder in this new framework. The first problem examined is sorting, for which two different approaches (sorting by distribution and sorting by merging) are presented, moreover a lower bound on the number of disk accesses that are necessary is shown.

The Fast Fourier Transform, and various problems in more specialized frameworks, such as Computational Geometry, Graph Theory, Spatial Data and String Processing, are presented. The chapter is completed with the presentation of a free External Memory Programming Environment (TPIE) and the experimental analysis of a number of External Memory algorithms.

Chapter 2 is devoted to Synopsis Data Structures, that is data structures that use very little space and provide fast (approximate) answers to queries. A central problem in this chapter is that of computing the frequency moments of a set of values (e.g. number of distinct elements, cardinality of the set, maximum element). Hot lists (storing only the best "m" elements) queries and histograms are also studied in the chapter.

In the one-dimensional case computing the median can be done quite easily, but this is not the case for higher dimensions. In the third chapter two alternative measures for two-dimensional data sets (i.e. regression depth and half-space depth) are studied, and fast (that is only a few passes over the data set are made) approximate algorithms for both measures are proposed.

[6]copyright Gianluca Della 2001

The next chapter deals with a new data structure (i.e. Cross-Trees) for dynamically maintaining multidimensional data sets in external memory, where the data sets are simultaneously sorted under various orderings, and the operations allowed are set splits and concatenations (w.r.t. an ordering) and the insertion/deletion of an element. Moreover the data structure can be exploited for enquiring about data contained in a given range. Some applications of the results are pointed out, most notably the implementation of multidimensional priority queues and $2-3$ trees.

Algorithms that scan only once (or at most a small number of times) the input data is the topic of Chapter 5, most specifically lower and upper bounds on the space used for solving some fundamental problems in graph theory (such as finding two vertices in the graph which are connected by the largest number of paths). Computing large cliques in very large graphs is the topic on the successive chapter.

Chapters 7-8 are focused on Computational Geometry, more precisely on the trapezoidal decomposition problem, and some fundamental problems in the field, such as computing the convex hull and the nearest neighbor of all elements (for these fundamental problems they give lower bounds on the I/O complexity).

The following chapter is a survey of External Memory algorithms for numerical linear algebra, that is solving linear equations or computing eigenvalues, Fast Fourier Transform and $n$-body computations.

Chapters 10-11 present two (freely available) libraries. One is for efficient handling of dynamic memory (replacement of `malloc`) and of I/O (replacement of the while `stdio.h` library). The second one is for maintaining external dynamic dictionaries with variable length keys, which has applications in file system implementations.

Adaptive online scheduling is the subject of chapter 12, where an algorithm for prefetching and I/O scheduling for read-once reference strings is presented. This algorithm needs an number of I/O operations which is only a small times that of the optimal algorithm.

The problem of distributing data on parallel memory systems is dealt with in the successive chapter, where fairly simple efficient schemes are presented.

The last two chapters are devoted to scientific visualization; notably the problems of isosurface extraction and retrieval of unstructured volume data are investigated. Two efficient solutions are proposed and studied on real-world cases.

## 3 Opinion

This book gives the current state of research of the field of External Memory algorithms, as such it is not suitable for teaching and requires a good general-purpose knowledge of algorithms (such as a one-term course on algorithms and another one on models of computation).

The book is well organized, the first chapters are introductory (especially the first one) and give all the fundamental notions that are required to be able to fully understand the subsequent chapters. I have found the problems and the algorithms discussed in chapter two very intriguing.

The core of the book (from chapter 3 to the end of the book) are focused on a specific problem and present technical results, so I guess that the average reader will not find all the chapters equally interesting, but the variety of the topics are so large that almost everybody in the algorithms field should find some chapters very intriguing.

An editing note: I would really like a unified bibliography (or at least a unique bibliography style). The editors have put a great effort in making an index (which can be very useful), and it is quite a nuisance to find some citations not up-to-date.

**Review of**
**$\pi$ - Unleashed**
**Authors: Jörg Arndt and Christoph Haenel**
**Publisher by Springer, 2001**
**$29.95, 250 pages**
**Review by: David J. Marcus, CEO, M5 Consulting, Inc. djmarcus@m5inc.com**

## 1 Overview

The authors combine very effectively the various elements of story telling, mathematical reference work (equations), and supplemental material (CD, and web). The book is in its second edition (a translation from German by Catriona and David Lischka). It is accompanied by a CD full of source code (C/C++), web references, miscellaneous documentation, and tutorials. It even includes the first 400 million digits of $\pi$ in 10,000,000-digit chunks (they needed to fill the CD somehow!).

The hardest part of reviewing this book is deciding where to start. The book 'logically' covers the history of $\pi$, from antiquity, to paper and pencil computations, to the computer age. I might as well mention the two complaints I have about this book and get them out of the way, so I can laud the rest. My first (minor) issue is the actual organization of the book. The order of the material is somewhat jumbled. The authors jump from topic to topic seemingly in a random manner. Perhaps the randomness of the digits of $\pi$ affected the authors' sense of organization! They present a short history of $\pi$ then jump to a discussion of its randomness, then to a short cut to the value of $\pi$. This is followed by various computational algorithms, discussions of Gauss, Ramanujan, and the Browein brothers, only to return to long arithmetic, then jump to a detailed history of $\pi$, and so on. It almost seems as if the book is a collection of previously independent papers. I consider this a minor issue that is soon forgotten once the reader becomes immersed in the beauty and richness of the content.

My second but more serious complaint is the lack of derivation for almost all of fascinating formulas for $\pi$ presented in the chapters and the exquisite collection of formulas at the end of the book (Chapter 16, page 223). While the book is brimming with infinite series and continued fractions which result in some simple equation involving $\pi$ (such as $\pi/4$, $1/\pi$, $\pi/6$ $\sqrt{\pi}$, etc), the reader is left to wonder at the magic spell engulfing the originator of each of these formulas. It would have made the book immeasurably better to include the derivation of each of the formulas.

Now that the complaints have been dispensed with, lets look the the good stuff. The book starts with a review of the "State of Pie Art". This is really a brief history of the Pied Pipers of history ranging from the Babylonians and Egyptians (who crudely approximated $\pi$), to *arctan* formulas, to appearances of infinite series and continued fractions as representations of $\pi$, and culminating with Professor Kamada who announced the calculation of $\pi$ to over 206 billion digits. It nicely summarizes the computational triumphs of the various generations (from the traditional $\frac{22}{7}$ to the aforementioned 206+ billion digits) and sets the stage for the rest of the book. As noted above, the order of the material is somewhat ad hoc. For example, the fascinating detailed history of $\pi$ is relegated to the later part of the book (Chapter 13, page 165) when in fact it should have been in the beginning.

The book is sprinkled with many gems. Some are numeric, some are infinite series and continued fractions, and others are mnemonics for remembering $\pi$. Here are some examples. As you read them, imagine yourself having to *derive* these from scratch, you will surely be in awe of their authors:

$\Rightarrow$ By John Wallis (1616 - 1703)

$$\frac{3 \times 3 \times 5 \times 5 \times 7 \times 7 \times \cdots}{2 \times 4 \times 4 \times 6 \times 6 \times 8 \times \cdots} = \frac{4}{\pi}$$

$\Rightarrow$ Due to Leonhard Euler (1707 - 1783)

$$\pi = \lim_{n \to \infty} \frac{4}{n^2} \sum_{k=0}^{n} \sqrt{n^2 - k^2}$$

$\Rightarrow$ By François Viète (1540 - 1603)

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \times \frac{\sqrt{2+\sqrt{2}}}{2} \times \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \times \cdots$$

$\Rightarrow$ Due to Lord William Brouncker (1620 - 1684)

$$\frac{4}{\pi} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

$\Rightarrow$ Developed by John Machin (1680 - 1752)

$$\frac{\pi}{4} = \arctan\frac{1}{5} - \arctan\frac{1}{239}$$

$\Rightarrow$ Due to Jörg Arndt, a mesmerizing connection between $\pi$ and the Fibonacci numbers:

$$\frac{\pi}{4} = \sum_{n=1}^{\infty} \arctan\frac{1}{F_{2n+1}}$$

$\Rightarrow$ Magically presented by Srinivasa Ramanujan (1877 - 1920). Every term in this series generates 8 accurate digits of $\pi$

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!}{(n!)^4} \frac{(1103 + 26390n)}{396^{4n}}$$

$\Rightarrow$ My personal favorite by Euler (it incorporates the 5 fundamental constants)

$$e^{i\pi} + 1 = 0$$

$\Rightarrow$ Also due to Euler (relationship between $e$ and $\pi$ which does not use $i$)

$$\frac{e+1}{e-1} = 2 + 4\sum_{r=1}^{\infty} \frac{1}{(2\pi r)^2 + 1}$$

⇒ BBP formula (David Bailey, Peter Borwein, Simon Plouffe, 1995) It remarkably generates the individual digits of $\pi$ in *hex*.

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n}\Big(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\Big)$$

⇒ By Collin Percival (1981 - ) Incredibly, used 700 years of computer time via the screen-saver deadtime on 1,700 computers on the Internet, to conclude that: *The quadrillionth bit of $\pi$ is 0.*

⇒ By Scottish mathematician Alexander Aitken (1895 - 1967) who showed that the following is within $10^{-2}$ of being an integer.

$$e^{\pi\sqrt{163}} = 262,537,412,640,768,743.9999999999992\cdots$$

⇒ An extension of the above shows that the following is within $10^{-24}$ of being an integer.

$$\sqrt[3]{e^{\pi\sqrt{163}} - 744} = 640319.9999999999999999999999993\cdots$$

⇒ Small C program (133 characters) to calculate $\pi$ to 15,000 decimal places:

```
a[52514],b,c=52514,d,e,f=1e4,g,h;
main(){for(;b=c-=14;h=printf("%04d",e+d/f))
for(e=d%=f;g=–b*2;d/=g)
d=d*b+f*(h?a[b]:f/5),a[b]=d%–g;}
```

⇒ A poem by English astrophysicist Sir James Jeans (1877 - 1946) where the count of letters in each word follows the digits of $\pi$:

How I want a drink, alcoholic of course, after the heavy chapters involving quantum mechanics. All of thy geometry, Herr Planck, is fairly hard...

Similarly, from Michael Keith, a Cadaeic cadenza which produces an awe-inspiring 740 decimal places for $\pi$. The first of 18 stanzas:

Poe, E. Near A Raven

Midnights so dreary, tired and weary. Silently pondering volumes extolling all by-now obsolete lore. During my rather long nap - the weirdest tap! An ominous vibrating sound disturbing my chamber's antedoor. "This". I whispered quietly, "I ignore".

⇒ Since 1995, the world record holder in recitation of the digits of $\pi$ has been Hiryuku Goto (then aged 21) of Japan who recited an incredible 42,000 digits of $\pi$ in 9 hours.

⇒ Using the estimated diameter of our Universe, one would need only the leading 39 digits of $\pi$ to calculate the circumference of the universe to within the radius of a proton!

⇒ Proof that $\pi = 2$. Start with a semi-circle with diameter as 1. Divide diameter into two and replace the semi-circle with 2 semi-circles (in an S-shape) each with the radius of $\frac{1}{2}$. Repeat this process ad infinitum. In the limit, the wavy S-shape curves become indistinguishable from the original diameter (a straight line). The combined length of the semi-circles, after $k$ divisions of the diameter is:

$$\lim_{k\to\infty} 2^k \frac{\pi}{2^{k+1}} = \frac{\pi}{2}$$

But the original diameter was 1, therefore:

$$\frac{\pi}{2} = 1 \Rightarrow \pi = 2$$

These "diamonds" are everywhere in the book. As I mentioned above, I could only wish for the authors to have included the derivations of each and every one of these (sigh). However, moving on, the book covers in details several algorithms for $\pi$ and related optimization of arithmetic operations. We will visit these briefly.

## 2 A Detailed Look

### 2.1 Spigot Algorithm

Developed by Stanley Rabinowitz and Stanley Wagon (1995), this algorithm has the remarkable property of churning out the digits of $\pi$ as the algorithm is executed. This is in sharp contrast to earlier algorithms that generated the results once the algorithm was complete. This makes the algorithm ideal for running on a personal computer. It is surprisingly fast, though the dependency on time is quadratic which means it is not suited for the really "monster" calculations of billions of digits. The mathematics behind it are fairly simple. This algorithm can be written in just a few lines of code (evidence the short C program above).

The starting point is the series:

$$\pi = 2 + \frac{1}{3}(2 + \frac{2}{5}(2 + \frac{3}{7}(2 + \cdots)))$$

The Spigot algorithm is based on a radix conversion to the equivalent "base 10" series:

$$\pi = 3.1415\cdots = 3 + \frac{1}{10}(1 + \frac{1}{10}(4 + \frac{1}{10}(1 + \frac{1}{10}(5 + \cdots))))$$

The first series can be considered a fraction using a mixed-radix representation where the bases are: $\frac{3}{1}$, $\frac{5}{2}$, $\frac{7}{3}$, $\frac{2i+1}{i}$, $\cdots$. The second series is clearly in base-10. The computation essentially generates 1 digit per calculation (ignoring the rare case where an overflow of 1 into a range of consecutive 9's converts them to 0's). The repeated steps consist of mainly dividing (from the right) by the mixed radix base, $\frac{2i+1}{i}$. On every division, the remainder is retained and the integer quotient is carried over to the next decimal place. The most recently calculated carry-over is the new decimal place of $\pi$.

This elegant method can be further improved by calculating $1000\pi$ starting from the series (multiplied by 1000):

$$1000\pi = 2000 + \frac{1}{3}(20000 + \frac{2}{5}(2000 + \frac{3}{7}(2000 + \cdots)))$$

This allows for conversion to be performed in base 10,000 rather than base 10. This means that 4 digits are generated at every iteration rather than just 1. Interestingly, this spigot algorithm can be applied to the transcendental number $e$:

$$e = 1 + \frac{1}{1}(1 + \frac{1}{2}(1 + \frac{1}{3}(1 + \cdots)))$$

## 2.2 Gauss, $\pi$, and the AGM formula

One of the fastest method of calculating $\pi$ was invented by Carl Friedrich Gauss around 1809. It was subsequently forgotten and only resurrected 25 years ago and transformed into the basis of super fast $\pi$ calculations. The arithmetic geometric mean (AGM) formula discovered by Gauss:

$$\pi = \frac{2AGM^2(1, \frac{1}{\sqrt{2}})}{\frac{1}{2} - \sum_{k=1}^{\infty} 2^k c_k^2}$$

where the $AGM(a, b)$ is the arithmetic-geometric mean of its two operands. The $c_k$ constants are directly linked with the AGM. The AGM rule iterates the arithmetic and geometric means until they "converge". In particular:

$$a_0 = a, b_0 = b$$
$$a_{k+1} = \frac{a_k + b_k}{2}$$
$$b_{k+1} = \sqrt{a_k \cdot b_k}$$
$$c_{k+1} = \frac{1}{2}(a_k - b_k)$$

The power of this algorithm is due to the rapid (quadratic) convergence of the AGM function. In essence each iteration generates twice as many digits as the previous iteration. Thus with relatively few iterations, incredible precision can be achieved.

## 2.3 The BBP Algorithm

One of the more remarkable achievements of modern times is the discovery by Bailey, Borwein, and Plouffe of an algorithm for generating any middle digit of $\pi$ (albeit in hexadecimal) without first generating all the previous digits. Furthermore, this algorithm scales almost linearly with the order of the digit desired. Even more remarkable is that the underlying equation was derived serendipitously by a computer. The formula:

$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n} \Big( \frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \Big)$$

Its distinctive power arises from the $16^n$ term in the denominator.

## 2.4 Karatsuba and FFT Multiplication

The principal obstacle to computation of $\pi$ is simple arithmetic operations on multiple-precision numbers. Multiplication time, for example, grows quadratically as the number of digits increase. To multiply two 2-digit numbers requires just 4 multiplies and 3 additions. However, when multiplying two 1,000-digit numbers, the number of multiplications skyrocket to 1,000,000. The Karatsuba multiplication method, is named after the Russian scientist Anatolli Karatsuba, though the true author is unknown. In its simplest form, consider the multiplication of two numbers $u$ and $v$, each a $N = 2^k$ digit number. Then let $u = 10^{N/2}u_1 + u_0$, and $v = 10^{N/2}v_1 + v_0$. Then their product becomes:

$$u \cdot v = 10^N u_1 v_1 + 10^{N/2}(u_0 v_1 + u_1 v_0) + u_0 v_0$$

This makes it clear that there are 4 partial multiplies (the multiplications by a power of 10 can be ignored since they are "just" a shift in the decimal point, that is, actual multiplication is not required). The Karatsuba method rearranges the terms differently to get the equivalent product:

$$u \cdot v = (10^N + 10^{N/2})u_1 v_1 + 10^{N/2}(u_1 - u_0)(v_0 - v_1) + (10^{N/2} + 1)u_0 v_0$$

By inspection, we can see that we have only 3 partial multiplications, though we incur 3 additional additions. This saving of a partial product becomes more significant when it is applied recursively to the partial products. Thus multiplying two $N = 2^k$ digit numbers using traditional multiplication requires $4^k$ multiplications. The Karatsuba method reduces the number to just $3^k$ (and some additions). As a result, the total time is reduced from a quadratic order $N^2$ to order $N^{\log_2 3}$. To put this advantage into perspective, where multiplying two 1-million digit numbers may take a day's worth of computing time, the Karatsuba method reduces the time to a mere 5 minutes.

The next significant advance is the FFT multiplication. This is based on the Fast Fourier Transforms methods. Prior to multiplication, the multiplicands are first transformed. Then, an operation which is equivalent to normal multiplication but runs much more quickly is performed on the transformed multiplicands. The results are subsequently transformed back, producing the desired product. The classic example of such transformations is adding logarithms in lieu of normal multiplications and converting the resultant logarithm back after the addition. The FFT reduces the computational time to order $N \log_2(N)$. This is indeed dramatic when $N$ is large compared to normal multiplication (order $N^2$), and even when compared with the Karatsuba method. In the case of 1-million digit numbers, the number of individual multiplications is reduced from 1 billion (normal multiplication) to approximately 20 million ($= N \log_2 N$), and reduces the computational time from roughly a day down to 3 seconds. It improves over the Karatsuba method by a factor of around 100. This makes it easy to see why it is the backbone of modern large-precision implementations. Note, by the way, that division can be treated as multiplication by the reciprocal. This allows FFT to be used in wide ranging large-precision numerical applications. While the book does not dwell on this, the next barrier to calculations with ever larger precision of $\pi$ is storage (RAM and disk). For the current record of 206+ billion digits, the result no longer fits in the computer's memory. Modern algorithms have to account for spooling data to disk. This means that the fastest algorithms are no longer necessarily the ones who reduce the number of multiplications to a minimum but rather ones that manage CPU and RAM and disk most efficiently.

## 3 Conclusion

The book is a treasure of information and is fun to read. It includes a chapter on $\pi$ formulas, continued fractions, products, iterative algorithms, etc. It even includes tables of various constants to 100 digits of precision. Of course, since this is a book about $\pi$, the book also includes tables of $\pi$ to 5,000 digits, in base 10 and base 16. To top it off, a table of the first 2,000 terms of the continued fractions representation of $\pi$ is also included. There is also extensive list of references (including web URL's where available). The code on the CD-ROM is not as well documented as I would have liked but it is in ready-to-run format (you can start using it right out of the box). Its inclusion alone is worth much more than the price of the book.

In short, I loved this book. In fact, I've pretty much run out of superlatives. I only wish the derivations of all these wonderful formulas was included. I would recommend it to anyone who is just curious about *pi*, or about large-precision arithmetic in general, as well as to the professional who is looking to break the next $N$-digit barrier. Who knows, perhaps a reader will be inspired to invent a faster yet method for such algorithms.

**Review of** [7]
**Chaotic Elections! A mathematician looks at Voting**
**Author: Donald Saari**
**Publisher: AMS, 2001**
**$23.00, 159 pages**
**and**
**Mathematics and Politics**
**Author: Alan Taylor**
**Publisher: Springer-Verlag, 1995**
**$39.95, 284 pages**
**Reviewed by William Gasarch, Univ. of MD, Dept of CS, gasarch@cs.umd.edu**

"... we mathematicians have tended to make two errors in teaching nonscience students: we have overestimated their comfort with computational methods, and we have underestimated their ability to handle conceptual material."
from "Mathematics and Politics"

## 1 Review of "Chaotic Elections"

### 1.1 Overview

If two people are running for office then a straightforward vote suffices to determine who the voters want. If three (or more) people run for office then many problems arise.

1. If $A$ gets 40%, B gets 30%, C gets 30%, and all the people who voted for B or C detest A (so 60% of the voters detest A) then does A reflect the peoples choice?
2. If people know that C cannot win so they vote for second choice B instead, and B wins for that reason, does B reflect the peoples choice?

The book under review (1) gives concrete examples (some real, some fictional) of these and other problems that arise with voting, and (2) gives several voting schemes that attempt to solve these problems. No scheme avoids all the problems; however, the Borda-count (described below) seems to avoid many of them.

[7]William Gasarch Copyright 2001

## 1.2 Chapter one: A mess of an election

Chapter one discusses the 2000 US presidential election, the electoral college, several voting schemes, and Arrows Theorem. Power indices, which measure how influential your vote is, are defined and discussed. The following voting schemes are defined.

1. *Positional Schemes:* A voter ranks all $n$ candidates and then points are allocated to the candidates using weights. For example, the *Borda-count* allocates $n$ points to the first choice, $n-1$ to the second choice, etc. The candidate with the most points wins. Different weights give rise to different schemes. It is a (non-rigorous) question of some interest to determine the weights that are the best in terms of discerning the people's choice.

2. *Condorcet Method:* A voter compares each pair of candidates. The candidate that wins the most pair-wise contests wins. If a candidate beats *all* other candidates in pair-wise contests, then she is called the *Condorcet Winner*.

3. *Approval voting:* A voter submits an unranked list of candidates she finds acceptable. Whoever gets the most approval wins.

The chapter ends with a discussion of Arrows theorem which says that, under some reasonable assumptions, it is impossible to design a perfect voting system. The author challenges the reasonableness of one of the assumptions.

## 1.3 Chapter two: Voter preference, or the procedure

## 1.4 Chapter three: Chaotic election outcomes

Chapters two and three discuss scenarios where, given how the voters voted, you can derive (non-contrived) voting scheme that could make any of the candidates win. A lengthy discussion of the 1860 US presidential election (Lincoln, Douglas, Bell, Breckinridge) describes (non-contrived) schemes under which Douglas or Bell could have won. No such scheme yields victory for Breckinridge. A nice geometric way of representing voter preference is introduced and used throughout the book. A theorem is stated which indicates that the Borda-count avoids many of these problems.

## 1.5 Chapter four: How to be strategic

Chapter four discusses voting strategically (e.g., voting for Bush instead of your first choice Libertarian Browne). Given a voting scheme and how the votes went, the book determines if a single voter (or bloc of voters) change their minds (in a rational way) so that someone else wins. This is close to what really does happen since, although voters do not know the outcome ahead of time, polls can give the voters a very good idea.

## 1.6 Chapter five: What *do* voters want?

Chapter five discusses scenarios where the Condorcet winner might not be the people's real choice. The author also reveals the method by which he has come up with the examples in the book.

## 1.7 Chapter six: Other procedures; other assumptions

Chapter six is a collection of related topics. Included are apportionment (what do do with the fractions?), non-parametric statistics (ranking machines based on measuring performance is similar to voting), non-transitive dice, and others.

### 1.8 Opinion

Because of the chaos that marked the 2000 US presidential election this is a good time to offer a course on the mathematics of voting. This book has nothing on dangled chads or butterfly ballots; however, it has a great deal about the problems of having three people in a race, such as strategic voting. These problems are clearly relevant.

This book is well written and would be suitable for such a course. The book does not have that much math in it, so students with a weak math background could take such a course. Students with a strong math background could be referred to *Basic Geometry of Voting* by Donald Saari (this book will be reviewed in a later column).

The book does not contain a discussion of the Hare voting scheme which is somewhat surprising as it seems to be a good one. It is discussed in "Mathematics and Politics" which is reviewed later in this column.

I suspect that this branch of math could inspire research in theoretical computer science. A cursory glance of the theory database (at http://ls2-www.informatik.uni-dortmund.de/cgi-bin/Paper) lists around 20 papers with the word "Election" in them that that seem relevant.

## 2 Review of "Mathematics and Politics"

### 2.1 Overview

As we have seen in the above review, elections can lead to mathematics of interest. However, other political issues also lead to mathematics of interest. The book "Mathematics and Politics" investigates a wide range of such issues.

The first five chapters of the book introduce some political issues, introduces some mathematics to explain it, and states some theorems. The next five chapters revisit these issues and proves these theorems. For readers of this column this will involve flipping back and forth many times; however, for a student this may be fine.

### 2.2 Chapters 1 and 6: Escalation

If one country manufactures 10 bombs then typically its enemy will then manufacture 11 bombs. And then the first country . . .. This is called an arms race. This can be modeled by an unusual auction. The prize is (say) one dollar. The two players bid back and forth or pass (and hence lose). The winner gets the dollar for his bid. However, the loser also has to forfeit his last bid. It is easy to see that the players may end up bidding more than one dollar.

What is the best strategy for the players to play? How does this kind of auction compare to other auctions? These and other questions are addressed.

### 2.3 Chapters 2 and 7: Conflict

Country A has the option of obeying its peace treaty with country B or attacking. Country B has the option of obeying its peace treaty with country A or attacking. If both obey the treaty then both are better off. But if one attacks and one does not, the attacker is better off. Should a country obey or attack? This leads to a discussion of game theory and the prisoners dilemma.

### 2.4 Chapters 3 and 8: Yes-No voting

There are many settings which do not follow the one-person one-vote system. For example, in the UN any member of the security council has veto power. These chapters discuss voting rules that exist and how they compare to each other. Even though there are only two options, YES and NO, there is still interest here. This did not come up in the book "Chaotic Elections!" because in that book one-person one-vote was assumed.

### 2.5 Chapters 4 and 8: Political Power

Who has more power, a voter in a small state that usually votes democrat, or a voter in a large state that is borderline? How to measure? These chapters introduce several ways to measure how influential a vote is, and compare them to each other.

### 2.6 Chapters 5 and 10: Social Choice

This is essentially the topic of chapters 2 and 3 of "Chaotic Elections."

### 2.7 Opinion

This book is well written and has much math of interest. While it is pitched at a non-math audience there is material here that will be new and interesting to the readers of this column.

The models are simplistic. For example, their is some question as to how well auctions model escalation. The examples seem out of date. Even though the book was written in 1995 most of the examples seem to be from the Cold War. These two points do not distract from the interest of the math; however, if teaching a course to non-math people these points might need to be addressed.

## 3 Comparing the two books

Both books requires little math background. Both could be used in a freshman interdisciplinary course addressed to both math-types and political-types.

Chaotic Elections is either only deals with elections that are one-person one-vote and involve three or more candidates. However, it covers this domain quite well. Plus it is very much up-to-date and seems to actually have things to say about the last election. Math and Politics has a much wider scope and has more math of interest.

The books (surprisingly) do not overlap that much. Chapters 5 and 10 of "Math and Politics" cover much the same material as chapters 2 and 3 of "Chaotic Elections" but aside from that there is very little overlap.