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. **Types and Programming Languages** by Benjamin C. Pierce. This is a book about implementing type systems in programming languages. It uses λ-calculus.

2. **Information Theory, Inference, and Learning Algorithms** by David MacKay. This is a book on Information Theory. It covers probability, inference, coding of data, and neural nets.

3. **Verification of Reactive Systems : Formal Methods and Algorithms** by Klaus Schneider. This is a book about verification of real time systems.

4. **Algorithmic Learning in a Random World** by Vovk, Gammerman, and Shafer. This is a book about Machine Learning that takes that randomness of the world into account.

5. **The Random Projection Method** by Santosh Vempala. The method in the title involves taking a high-dimensional problem and randomly projecting it to one-dimensional. This technique is used in combinatorial optimization, learning, and information retrieval.

Books I want Reviewed

If you want a FREE copy of one of these books in exchange for a review, then email me at gasarchcs.umd.edu

Reviews need to be in LaTeX, LaTeX2e, or Plaintext.

Books on Algorithms and Data Structures

1. *Combinatorial Optimization: Packing and Covering* by Cornuejols.

2. *An Introduction to Data Structures and Algorithms* by Storer.

3. *Nonlinear Integer Programming* by Li and Sun.

4. *Biologically Inpsired Algorithms for Financial Modelling* Brabazon and O'Neill.

5. *Planning Algorithms* LaValle.

Books on Cryptography, Number Theory, and Security

1. *Concurrent Zero-Knowledge* by Rosen

2. *Algebraic Aspects of the Advanced Encryption Standard* by Cid, Murphy, Robshaw.

3. *Complexity and Cryptography: An Introduction* by by Talbot and Welsh.
4. *Complexity Theory and Crytography: An Introduction to Cryptocomplexity* by Rothe.
5. *Crytographic Applications of Analytic Number Theory: Complexity Lower Bounds and Pseudorandomness* by Shparlinski.
6. *Cryptography and Computational Number Theory* edited by Lam, Shparlinski, Wang, Xing.
7. *Coding, Cryptography, and Combinatorics* edited by Feng, Niederreiter, Xing.
8. *Computer Viruses and Malware* by Aycock.
9. *An introduction to Sieve Methods and their Applications* by Cojocaru and Murty.
10. *Diophantine Analysis* by Steuding.

Books on Coding Theory

1. *Block Error-Correcting Codes: A Computational Primer* by Xambo-Descamps.
2. *Authentication Codes and Combinatorial Designs* by Dingyi Pei.
3. *Algebraic Coding Theory and Information Theory: DIMACS Workshop* Edited by Ashikhmin and Barg.

Combinatorics Books

1. *A Beginners Guide to Graph Theory* by Wallis.
2. *Graphs and Discovery: A DIMACS Workshop* Edited by Fajilowicz, Fowler, Hansen, Janowitz, Roberts. (About using software to find conjectures in graph theory.)
3. *Combinatorial Designs: Constructions and Analysis* by Stinson.
4. *Computationally Oriented Matroids* by Bokowski

Logic and Verification Books

1. *Formal Models of Communication systems: Languages, Automata, and Monadic Second Order Logic* by Bollig.
2. *Classical Mathematical Logic: The Semantic Foundations of Logic* by Epstein.
3. *Logicism Renewed: Logical Foundations for Mathematics and Computer Science* by Gillmore.
4. *Essay on Constructive Mathematics* by Edwards.
5. *Elements of Finite Model Theory* by Libkin.
6. *Mathematical Approaches to Software Quality* by O'Regan.
7. *Software Abstractions: Logic, Language, and Analysis* by Jackson.

Misc Books

1. *An Introduction to Game-Theoretic Modelling* by Mestertson-Gibbons.
2. *Computation Engineering: Applied Automata Theory and Logic* by Gopalakrishnan. (Looks like an automata theory textbooks.)
3. *Combinatorial Auctions*. Edited by Cramton, Shoham, and Steinberg.
4. *Small Worlds: The Dynamics of Networks Between Orders and Randomness* by Duncan J. Watts.
5. *Handbook of Computational Methods of Integration* by Kyther and Schaferkotter.
6. *Polynomials* by Prasolov.
7. *An Introduction to Difference Equations* by Elaydi.
8. *Difference Equations: From Rabbits to Chaos* by Cull, Flahive, Robson.
9. *Fundamentals of Algebraic Graph Transformations* by Ehrig, Ehrig, Prange, Taentzer.
10. *Informationa and Self-Organization: A Macroscopic Approach to Complex Systems* by Haken.

Types and Programming Languages
Author: Benjamin C. Pierce
MIT Press, 2002, 623 pages
Review written by Mats Kindahl[2]

Overview

Type systems and type checking are playing an increasingly important role in the development of programming languages. Almost every programming language developed today are developed with type systems as an integral part of the language.

The goals of this book are to cover all important core topics of and give useful examples of how to use and implement type systems for programming languages. The book is separated into several parts, each part with coverage of one important core topic. After the introduction of lambda-calculus and some basic formalism for describing languages, the book presents a simple type system for lambda-calculus. In the following parts, the type system is then extended to cover recursive types, subtypes, polymorphic types, and higher-order polymorphic types (of the form that the purely functional language Haskell uses).

In the course of defining and extending the type systems, the author gives several detailed examples of how to use them. The examples are interesting and to-the-point. Each important topic contain at least one in-depth study of an either an application or an example type system. In the book, there are case studies of both functional and imperative versions for objects (as in object-oriented programming) and also an introduction to *Featherweight Java*, which is a minimal core system for modeling Java's type system.

[2] ©Mats Kindahl, 2006

Coverage

Untyped Systems

The first part of the book introduces a simple language to express untyped arithmetic expressions. The purpose of this part is to introduce fundamental concepts for the description of syntax and semantics of programs.

The subjects introduced are basic and can be skipped by anybody familiar with the subjects lambda-calculus, functional programming, and basic program theory. I would, however, recommend reading this part: there are several interesting tidbits that you might not be familiar with, unless you are already have experience in the theory and implementation of functional languages.

The first chapter, *Untyped Arithmetic Expressions*, starts off by presenting notations and methods for describing and reasoning about languages. The first notation introduced is the traditional BNF notation. It continues by explaining the nature of BNF as a *metalanguage* for describing languages and also presents some other notations—inductive definitions, definitions by inference rules, and concrete definitions as sets—but do not delve too deep in them. Here is also where the first encounter with structural induction, used to handle the terms of the language.

Several notions for the description of semantics are presented: operational, denotational, and axiomatic. In the sequel, the operational semantics are used to describe the evaluation of the terms of the language. In the next chapter, *An ML implementation of Arithmetic Expressions*, an implementation of the untyped arithmetic is demonstrated.

The part continues with *The Untyped Lambda-Calculus*, introducing the traditional lambda-calculus, including beta-reduction using different evaluation strategies (full, normal order, call by name, and call by value). The chapter continues with introducing Church encodings of booleans and numerals. Finally, the formal definition of the syntax and evaluation of the untyped lambda calculus (called λ) is introduced.

In preparation for the ML implementation of the untyped lambda-calculus, the chapter *Nameless Representation of Terms* introduces *de Bruijn terms*, which is a representation of lambda-terms that eliminates the need for names (and therefore also eliminates scoping problems). The chapter is followed by *An ML Implementation of the Lambda-Calculus* using the above methods for implementing evaluation of lambda-terms.

Simple Types

This part gives reasons for using types and introduces a basic typing systems for lambda-calculus terms. An interesting theoretical results is presented: well-typed terms are *normalizable*—i.e., that every well-typed terms is guaranteed to halt in a finite number of steps.

The part gives a sound and thourogh introduction to the theoretical and practical issues on typing by presenting a simple type system and demonstrating how to implement it in actual code. In addition to giving an informal explanation the description of each extension and/or alteration is given as a scheme with syntax, evaluation, and typing rules; where each change or addition is clearly marked. By starting with simple typing issues, the author manage to give a clear presentation of type systems and their implementation; in addition this gives the reader a gentle introduction to the notation used.

In the chapter *Typed Arithmetic Expressions*, the idea of using typing as a method to achieve safety is presented and Harper's *Safety = Progress + Preservation* slogan is presented and formal-

ized. In the chapter on the *Simply Typed Lambda-Calculus* types and typing rules are added to the previously introduced lambda-calculus, giving a typed lambda-calculus (called $\lambda_{\rightarrow}$). This chapter also presents the *Curry-Howard Correspondence*, demonstrating the correspondence between logical propositions and types. After discussion of the implementation of $\lambda_{\rightarrow}$ in *An ML Implementation of Simple Types* several simple extensions to $\lambda_{\rightarrow}$ are given in *Simple Extensions*. This chapter demonstrates how to add several interesting extensions to $\lambda_{\rightarrow}$: *base types* and the *unit type*, *ascription*, *let bindings* and its complications, *pairs*, *tuples*, *records*, *sums* and *variants* (as in *variant records*), *general recursion*, and (finally) *lists*.

In the chapter *Normalization* the author proves that every well-typed term is normalizable. After that, some more complicated extensions are demonstrated: in *References* it is shown how to extend the typing system with reference types and in *Exceptions* the effect of adding exceptions are discussed.

Subtyping

With the introduction of various object-oriented languages, the need for type systems able to handle subtyping became vital. In this part, the author describes the intricacies of creating a type system for subtyping. Although many languages use some form of *nominal* subtyping systems—with Ada 95 perhaps having the most rigorous type system—the author have chosen to describe *structural* subtyping systems. In a nominal type system, every type have to be defined prior to use. The definition includes the name and subtype relation, effectively meaning that the *name* of the type defines the type. In a structural type system, the structure of data determines the type. For example (using Pascal notation), in the definitions below, `a` and `b` have different types in nominal type systems—a new type name is created for each structure—but they have the same type in structural type system.

a : **record** x:Integer; y:Real **end**;
b : **record** x:Integer; y:Real **end**;

In the first chapter, *Subtyping*, the author describes the subtyping relation as a *preorder* on the types and demonstrates how to extend $\lambda_{\rightarrow}$ with subtyping $\lambda_{<:}$. The basic $\lambda_{<:}$ system is not very interesting since it does not have any subtype structures. To create an more interesting language, the author defines a subtype relation on records and adds these to the $\lambda_{<:}$. He continues by explaining that the subtype relation can be extended to a lattice by adding a bottom element (the top element is already present in $\lambda_{<:}$), but also demonstrates how this complicates the problem of building a type checker for the language. He the explains how to extend $\lambda_{<:}$ with ascriptions and casting, base types, variants, lists, references, and the new types *arrays* and *channels*. He concludes the chapter by giving an alternative to subset semantics used for describing subtyping above: namely *coercion semantics*, where the compiler subtyping is replaced with run-time coercions changing the representation of types. This method is used in, for example, modern C and C++ compilers.

The chapter *Metatheory of Subtyping* discusses the algorithmic aspects of creating a type-checking algorithm for a type system with subtypes, which the succeeding chapter *An ML Implementation of Subtyping* uses to implement a type-checking system for into $\lambda_{<:}$.

In the chapter *Case Study: Imperative Objects* the author demonstrates how to create and use the type system to implement a functional version of objects with both methods and instance

variables and the chapter *Case Study: Featherweight Java* presents and describes a minimal core calculus for modeling Java's type system that was proposed by Igarashi, Pierce, and Wadler in 1999.

Recursive Types

Simple types are sufficient for many purposes and also have the nice property that every well-typed term is normalizable. Unfortunately, a simple (non-recursive) type system fails for some quite trivial examples, such as trying to express the type for a binary tree (in fictious Pascal notation):

```
type Tree = record
    case leaf: (data:String);
    case node: (left,right:Tree)
end;
```

The remedy is to extend the type system with operators to express *recursive types*, which this part explains in detail.

In the first chapter, *Recursive Types*, the advantages of recursive types are discussed and several examples of useful recursive types are presented: lists, streams, processes, and objects. The chapter continues by explaining the difference between iso-recursive and equi-recursive types, but unfortunately a simple, intuitive explanation of the differences is missing. The next chapter, *Metatheory of Recursive Types*, deals with several subjects with regard to recursive types. It starts by introducing the theoretical foundations for recursive types: induction and coinduction in fixed point theory. It continues by introducing the domain of *tree types*, which are then used to represent the subtype relation. The last part of the chapter deals with implementation of iso- and equi-recursive type systems for a recursive type system with subtypes.

Polymorphism

The term *polymorphism* refer to language constructions that allow parts of a program to work with several different types in different contexts. For example, the following procedure 'maparray'[3] (in a fictious Pascal) is an example of using *universal types*.

```
procedure maparray[type T](var a:array[0..10] of T;
                           procedure p(var x:T));
var i:0..10;
begin
    for i := 0 to 10 do p(a[i])
end;
```

The first chapter, *Type Reconstruction*, develop a more powerful *type reconstruction algorithm*. In contrast to the *type checking algorithms* used prior in the book, this algorithm is capable of computing the *principal type* of a term. The algorithm is built around a constraint-based typing algorithm and uses unification to compute solutions to constraints. The chapter also discusses a simpler form of polymorphism known as *let-polymorphism*. In the *Universal Types* chapter, a more

[3] having the universal type $\forall T.\,(\text{Ref }(\text{Array }T) \times (\text{Ref }T \to \text{Unit})) \to \text{Unit}$.

general form of polymorphism, known as *System F*, is developed. System F permits abstracting out *types* out of terms in the same fashion as the lambda-calculus permits abstraction of terms out of terms. Since this means that the term is universally defined for any type, the term is typed with a *universal type*, written $\forall A.T$ (where T is a type containing the free *type variable* A). Since terms can have universal types—that is, it is well-typed for *any* supplied type—it is natural to ask if it is meaningful to have terms that are well-typed for *some* type. The chapter *Existential Types* considers this and shows that it is quite meaningful for use in data abstraction and information hiding. The chapter *An ML Implementation of System F* the author demonstrate how to extend $\lambda_{\rightarrow}$ with universal and existential types.

When combining subtyping and polymorphism, several problems surface. In the chapter *Bounded Quantification* the author demonstrates how to combine subtyping and polymorphism into the calculus $F_{<:}$ and in the chapter *Case Study: Imperative Objects, Redux* he extends on the previous chapter on imperative objects, demonstrating the role of polymorphism and bounded quantification.

The part is concluded with *Metatheory of Bounded Quantification* where the type checking algorithm for $F_{<:}$ is developed.

Higher-Order Systems

Using the polymorphic system above we can express, for example, lists of elements of type X as the type $\forall R.(X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R$ (on page 351)—which is not very readable. This part introduces *type operators* as operators from types to types. For instance, we can define the type operator `List` by

$$\texttt{List}\ X = \forall R.(X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R.$$

In the first chapter, *Type Operators and Kinding*, type operators are discussed in depth. The notion of *kinds* as "types for type operators" are introduced. Both type operators and kinds are added to $\lambda_{\rightarrow}$ giving a system named λ_{ω}. In the next chapter, *Higher-Order Polymorphism*, λ_{ω} and System F is combined into System F_{ω}, which is then, in chapter *Higher-Order Subtyping*, combined with subtyping to yield System $\mathrm{F}^{\omega}_{<:}$. The last chapter of the part, *Case Study: Purely Functional Objects*, is a in-depth case study of typing purely functional objects in System $\mathrm{F}^{\omega}_{<:}$. The objects permits using all the features described in the book—especially considering subtyping, universal and existential types, and type operators—and also includes *polymorphic update* to handle different representations of the object while retaining the objects original behavior.

General impression

On the general level, the book is well-written and well-structured. The text is easy to follow and, as an extra help, there are small exercises interleaved with the text so that you can test your understanding of the subject. Although the subjects can at times be complicated, no previous experience of type-theory is assumed—you are provided with an understanding of the issues involved before the complicated subjects are introduced.

The intended audience of the book are mature undergraduate and graduate students and also researchers specializing in programming languages and type theory. It is my opinion that the author has managed keep the intended focus. The book is suitable for mature compute scientists but does not require any previous understanding of type theory.

There are not really any bad sides to the book, but there are a few issues that should be mentioned. The ML implementations contain a lot of extraneous pieces of code that can obscure the issue at hand. For example: (1) there are error handling code needed in a real implementation but which makes the code hard to follow and (2) variables used in the actual implementation but which serves no purpose in the code shown. Another issues is that the book is almost entirely in a functional setting (in contrast to an imperative setting). The advantage is that problems with type systems are "out in the open", but the notation is sometimes unwieldy and very hard to follow.

Information Theory, Inference, and Learning Algorithms[4]
Series: Texts in Theoretical Computer Science : An EATCS Series
Author of book: David J. C. MacKay
Publisher: Cambridge University Press, 2003. $39.50, Hardcover

Reviewer: Maulik A. Dave

1 Introduction

This is a text book on information theory. The major areas, covered by the book, are probability, inference, coding of data getting communicated, and neural networks. According to the preface of the book, the book is aimed at senior undergraduates and graduate students in Engineering, Science, Mathematics, and Computing. Familiarity with calculus, probability theory, and linear algebra is expected as a prerequisite. According to the author, the information theory, and machine learning are two sides of the same coin, and hence the inclusion of machine learning is in this book on information theory. The book is also connected with a course taught by the author at Cambridge, under the title *Information Theory, pattern recognition, and neural networks.* The resources on the book on internet can be found at the web site

`http://www.inference.phy.cam.ac.uk/mackay/itila.`

2 Information Theory as in the Book

The data in computers, or digital communications consists of sequence of bits. To make the data informative, the data is organized, which is called coding. The information is coded in data by the sender of the information, or writer of the information. The reader, or receiver of the information retrieves the data with the knowledge of coding. The theory behind the information contents in the data is the information theory. The noisy channel is the channel, where the data received is slightly different then the data sent. Retrieving such a data is a problem to be solved using probability, and inference theories. Hence, these theories have close relation with information theory. In general, the information theory provides the foundation to solve the problems of information contents of data in various forms, and codes.

[4]

3 Organization, and Contents of the Book

The book is organized by dividing it in 7 parts spreading over 50 chapters, and 3 appendicitis. Before each chapter, a page or two of motivation for the chapter is found. The main bodies of the chapters have explanations, examples, illustrations, figures, and exercises. The preface has 5 pages of the chapter dependency diagram guiding the orders in which the book can be read. The printing format is a two columns format for each page. The left hand side bigger column has main text flow, and the right hand side smaller column has supplements. Very important statements, or small notes are enclosed in rectangular boxes. One can scan through the book by reading texts in these boxes.

The first three chapters introduce the information theory, and give a flavor of things to come in the other chapters of the book. Topics covered are various types of probabilities, Hamming code, ensembles, entropy, and inference. The first part is on data compression consisting of four chapters. The part begins with a discussion on information content of an outcome of a random experiment. The data compression for source codes, symbol codes, stream codes, and integer codes is explained. The major theorems presented are Shannon's source coding theorem, and source coding theorem for symbol codes. The second part is on noisy channels consisting of four chapters. The part begins with background mathematics in chapters 8, and 9. The chapter 9, and 10 contains a detailed discussion on the noisy channel coding theorem. The chapter 11 has discussion on the Gaussian channel.

The part 3 of the book consisting of 8 chapters, presents further topics in information theory. The chapter 12 is on hash codes for information retrieval. The chapter 13 discusses distance properties of codes. The chapter 14 presents a simultaneous proof of the source coding and noisy channel coding theorems. The chapter 15 is a collection of exercises. The chapter 16 is on message passing. The chapter 17 discusses constrained channels. The chapter 18 introduces the crosswords. The chapter 19 introduces evolution, and its relation to information acquisition.

The part 4 consists of as many as 18 chapters. This part is mostly mathematical in nature revolving around the inference. The chapter 20, and 22 introduce clustering, and the K means algorithm. The K-means algorithm is an algorithm for putting N data points in an I-dimensional space into K clusters. Each cluster is parameterized by a vector as its mean. The maximum likelihood parameters are also discussed. The chapter 23 presents some common probability distributions over integers, over real numbers, periodic variables, and probabilities. The chapter 21 is on exact inference obtained by complete enumerations. The exact marginalization is discussed in chapters 24 to 26. The laplace's method, and occam's razor are introduced in next 2 chapters. The chapters 29 to 32 are on the monte carlo methods. The major topics in monte carlo methods covered are importance sampling, rejection sampling, the metropolis method, Gibbs sampling, slice sampling, Hamiltonian monte carlo, the multi-state leapfrog method, ising models, and exact sampling concepts. The chapter 33 to the chapter 36 covers topics like variational methods (especially the variational free energy minimization), latent variable models, component analysis, the Luria Delbruck distribution, and an introduction to the decision theory. The last chapter of the part 4 compares the Bayesian inference, and the sampling theory.

The part 5 consisting of 9 chapters is devoted to neural networks. The neural networks are presented in the context of inference. The neural networks are learning machines. The learning as inference demonstrates a very good application of the inference theory. This is established in chapter 41 after describing single neurons in chapters 38 to 40. A feedback network called Hopfield

network is discussed in chapter 42. This is followed by a short chapter on boltzman machines. The feedforeward multilayer networks are found in chapter 44, and chapter 45. The last chapter of the part presents the deconvolution in image reconstruction.

Although graphs are dealt in earlier chapters, the last part of the book is on sparse graphs. The family of graphs discussed are low-density parity-check codes, turbo codes, repeat accumulate codes, and digital fountain codes. The book also has few appendixes followed by a 7 page bibliography.

4 Comments of the Reviewer

Most of the theories are accompanied by motivations, and explanations with the corresponding examples. The policy for organization of the book is to keep the chapter sizes small, and to present a large number (50) of chapters supplied with the chapter dependency diagrams. This policy can be of great help to students studying from the book. The division of the chapters in first 4 parts could have been different. For many topics, there are indicators for further readings.

On the whole, the book achieves its goal of being a good textbook on information theory.

Verification of Reactive Systems : Formal Methods and Algorithms
Springer-Verlag Berlin Heidelberg, 2004
Series: Texts in Theoretical Computer Science : An EATCS Series
Author: Klaus Schneider
Publisher: Springer - Verlag Berlin Heidelberg New York, 2004
Reviewer: Maulik A. Dave[5]

1 Introduction, and Background

This book is in the series on theoretical computer science. One of the areas, where results of theoretical computer science is successfully applied, is verification of computing information processing systems. The book classifies systems into transformational systems, interactive systems, and reactive systems. The reactive systems are real time systems. In these systems, the reactions to the environment of the systems, have to be in real times. It is not clear from the book, the connection between verification of reactive systems, and finite state formalisms. However, it can be observed that publicly available verification tools like spin, and PVS use the finite state theories for their purposes. There are a very large number of finite state formalisms developed. The author has chosen μ-calculus, omega automata, temporal logics, and predicate logics for the book. Clearly, the choice is for the finite state formalisms, which are often used in "verification of reactive systems".

Apart from presenting the formalisms, the book also has comparison of these formalisms with each other, and existing translations from each other. It has a common language for better understanding of the concepts in these formalisms, and their relations to each other.

2 Contents, and Structure of the Book

The book has 7 chapters, and 3 appendices.

[5] ©2006, Maulik Dave

The chapter 1 has introduction of the book. It starts with taxonomy, and classification of the formal methods. Then it explains classification of the systems. The main contribution of the chapter is a few pages devoted to the history of verification of the systems, and related subjects like automated theorem proving.

In second chapter, Kripke structures are projected as formal models of reactive systems. After some theories on Kripke structures, a unified specification language is described. Normally, systems are verified with respect to their specifications. A unified specification language, which covers all the logics described in the book, acts as a good vehicle for describing the formalisms, and their interrelations. The semantics of the language, involving kirpke structures, is also described. At the end of the chapter, various normal forms for terms in the language are described.

The third chapter is on mew calculus. The vectorized mew calculus is described in details. The chapter also describes various theories, and algorithms for model checking associated with mew calculus. At the end of the chapter, its relationship with omega tree automata is described. The forth chapter has theories on omega automata. After introducing regular expressions, and finite state automata, the chapter describes in details the theories, algorithms, hierarchies of the omega automata. The chapter ends with the translation procedures from omega automata to vectorized mew calculus. If your priority is to read practical theories first, then the third, and forth chapters can be read later than reading fifth, and sixth chapters.

The fifth chapter is on temporal logic, and its variants. Major portion of the chapter is devoted to the translation of temporal logics to mew calculus, and the translation of temporal logics to omega automata. The chapter ends with presenting methods of involved reductions, and simulations. The sixth chapter is on predicate logics, mainly monadic. The chapter includes Buchi's decision procedure, translation from monadic second order logic of linear order to omega automata. If you are building a verification tool, or designing a verification process for a system, then fifth, and sixth chapters can be useful.

The seventh chapter concludes. The appendix A is on data structures, called binary decision diagrams. These diagrams can be of great help in implementing some of the theories above. Appendix B describes procedures for model checking mew calculus. The appendix C contains some fundamental concepts from mathematics, used in the book.

The book has a huge list of 527 references. It has 149 figures. The author is from Germany.

3 Opinion

Undoubtedly, the book can be kept as a reference for theories on verification of computing systems, especially finite state formalisms. The book is complete with respect to its concepts, and explanations. The research oriented readers can find the book useful, not only because of a good description of theories, but also because of the references to related works, and historical perspectives. For theory oriented readers, the flow of chapters is smooth. For implementation oriented readers, I think, one of the useful reading flows can be chapter 5, chapter 6, appendix A, appendix B, and referring other chapters when necessary On the whole, the book is well written, and appropriately a part of a series in theoretical computer science.

Algorithmic Learning in a Random World
Authors: Vovk, Gammerman, and Shafer
Published by Springer, 2005
Hardcover, 316 pages, $69.95
ISBN: 0-387-00152-2

Review by James Law[6]
The Software Maintainer's Network
http://www.softwaremaintainers.net
San Diego, California, USA

1 Overview

Machine learning is an area of computer science concerned with developing algorithms that learn from experience. For example, handwriting recognition is an important problem to solve, but it is not amenable to direct mathematical analysis. Machine learning algorithms attempt to learn how to distinguish handwritten letters by experience rather than by performing direct computations. This experience comes through training, which consists of showing the algorithm examples whose information can be integrated into the recognition of future examples.

The methods used to conduct training vary widely. One important division in training methods is *batch* versus *on-line*. During batch training a group of examples is shown to the algorithm, then new examples are classified based on the training. In on-line training, training and classification start simultaneously from the first example. Batch training requires a set of known examples beforehand, while on-line training can work with examples as they become available. The authors call on-line algorithms "predictive" algorithms since they use previous experience to predict a classification for each new example as it arrives.

An additional feature of some machine learning techniques is *hedging*. Hedging is the ability of an algorithm or machine learning technique to give a valid indication of the reliability and accuracy of a given classification.

Most current machine learning algorithms can be adapted for use in an on-line training environment, and hedging may be added if the original algorithm does not directly support it. *Algorithmic Learning in a Random World* is concerned only with machine learning in on-line, hedging applications.

The other machine learning distinction that is important in understanding this book is the assumptions made about the examples used in training. Generally it is assumed that the examples come from some underlying distribution, and that the examples are presented to the algorithm in a random order (that they are independent of each other). If the examples are not actually independent, we would like to assume that they are exchangable (that changing the order of presentation will not effect the classification accuracy).

Algorithmic Learning in a Random World is devoted to an examination of the performance of hedging in predictive algorithms (*conformal prediction*, in the authors' terms), and how hedging is affected by the assumptions of exchangability and randomness machine learning makes about the data. This narrowly limits the scope of the book, reflecting the authors' longstanding academic interests.

2 Summary of Contents

Algorithmic Learning in a Random World has ten chapters, three appendices, and extensive references. Each chapter ends with a section containing comments, historical discussion, and bibliograhical remarks. These remarks sections generally contain interesting discussion. The authors have clearly devoted a refreshing amount of effort to them.

The first chapter consists of short discussions of each of the book's chapters. These provide quite readable thumbnail sketches of each chapter. The chapters of the book are roughly divided into groups on the following topics: machine learning under various assumptions of randomness, the problems with existing machine learning theory that the book intends to address, conformal prediction, probabilistic prediction under unconstrained randomness, and on-line compression modeling.

3 Chapter Highlights and Historical Notes

The first part of Chapter 2 introduces formal details of the author's *conformal predictors* and defines the significance level used in hedging. Since the underlying machine learning algorithm is not specified, the analysis is in terms of two potentially infinite spaces. The object space contains the examples and the label space contains the classifications. Hedging specifies a subset of the label space such that the correct label is a member of the subset with some given significance level. The size of this subset must shrink as the significance level rises.

The second part of Chapter 2 shows how to apply conformal predictors using ridge regression confidence machines and nearest neighbors regression as the underlying machine learning algorithm. The empirical data presented is confusing. The author's intent is to "test the validity and evaluate the efficiency of some of the conformal predictors introduced". Other than showing that errors decrease during on-line training, I am not clear they show either. Perhaps readers who are familiar with the USPS data set of handwritten digits and the Boston Housing data set will find it easier to understand.

Chapter 3 concentrates on the classification problem using conformal predictors. The examples given in the chapter use support vector machines as the underlying algorithm. The empirical results are explained and labelled considerably better than in Chapter 2. The last part of the chapter discusses details of the confidence prediction mechanism. The proofs are lengthly.

Chapter 4 presents further modifications of conformal predictors to accommodate various desireable properties: validity, asymptotic efficiency, and flexibility (the ability to incorporate a wide range of machine learning methods). They also examine the ability to adapt to different teaching schedules. For instance, the algorithm may be given the correct label (the answer) before classification, immediately after classification, or after some delay. Finally, they examine conditionality; the answer may not be given at all with some probability.

Impossiblity results are presented in Chapter 5. These results show that under certain conditions it is not possible to find valid and efficient conformal predictors. These results are used in the next chapters to design new predictors (multiprobability predictors) that address these shortcomings.

Chapter 6 develops multiprobability predictors, or Venn predictors. Venn predictors output several probablility distributions for each new label rather than a single one. These probablility distributions tend to become identical as the number of old examples grows large.

Chapter 7 explores the relaxation of some of the standard assumptions of randomness and exchangeability on the authors' techniques. Although this chapter is short, it is technical and

dense.

On-line compression models are the subject of Chapter 8. The authors' relax the exchangeability assumption to the assumption that the examples follow an "on-line compression model". Examples of on-line compression models could be Gaussian or Markovian. The authors have included a large amount of material on how these compression models have been developed. The end-of-chapter notes contain extensive discussion of the relationship of this chapter to the authors' earlier investigations of Kolmogorov complexity. These notes are slightly less than eight pages, and mostly an historical discussion.

The next to last chapter explores a new class of on-line compression models that the authors call "hypergraphical models" for the classification problem with an infinite label space. Hypergraphical models are analogous to causal networks in machine learning and contingency tables in mathematical statistics. They also extend the Venn predictors introduced in Chapter 6.

The final chapter of the book attempts to discuss the historical and philosophical relation of these new methods in the field of machine learning by comparing them to inductive and Bayesian techniques. Although long, this is an excellent summary chapter.

Appendix A give some definitions in probability theory that are used in the book, but is not a tutorial. Appendix B discusses the USPS and Boston Housing data sets. Appendix C is a short (six question) FAQ.

4 Opinion

The material is developed well and reasonably easy to follow considering it has such a specialist's focus. The discussion in the text is good enough to follow even if you can't understand the proofs. Other than a rather large number of awkward (but still understandable) phrases, the text is very readable. No typos were detected in the proofs, but time permitted only a little checking.

The publisher's blurb for this book (which can be seen on Springer.com and Amazon.com) claims that the book's results are due to "theoretical and experimental developments in building computable approximations to Kolmogorov's algorithmic notion of randomness". This is puzzling because Kolmogorov's work is only mentioned once outside the end-of-chapter notes. In these end-of-chapter notes, the authors make it clear that some of their initial thinking in this area was influenced by attempts to understand Kolmogorov's work, but that nothing in the book relies on Kolmogorov's work, nor do they extend it in any way. In fact, in the end-of-chapter notes for Chapter 5, page 140, they state, "the notion of algorithmic randomness has nothing to do with the assumption of randomness." The authors' own website for the book (http://www.alrw.net/) only mentions "connections with Kolmogorov's algorithmic randomness". So, if you are interested Kolmogorov's algorithmic randomness, this book probably won't be helpful.

If this is your research area, *Algorithmic Learning in a Random World* is doubtless an important reference summarizing a large body of work by the authors and their graduate students. Academics involved with new implementations and empirical studies of machine learning techniques may find it useful too. However, it is probably quite a bit too theoretical for general machine learning practitioners. The lack of examples or exercises would make it difficult to use as a graduate text. The number and density of proofs alone probably rules out undergraduate use. While the inclusion of empirical results is nice, better explanations and direct comparison to a variety of widely used machine learning techniques would be welcome.

The Random Projection Method
by Santosh Vempala

Review by Aravind Srinivasan[7]
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland at College Park
College Park, MD 20742, USA
textttsrin@cs.umd.edu

1 Introduction

This book is an exposition of the *Random Projection Method*, wherein one projects a high-dimensional set of points or distribution to a "random" low-dimensional space using a careful choice of distribution. The book describes this method, which has seen active research recently, in the context of three major areas: combinatorial optimization (especially approximation algorithms), learning, and information retrieval. It is especially easy to imagine why dimension-reduction would help in these last two areas.

2 Summary of Contents

Chapter 1 starts with the fundamental observation of Johnson-Lindenstrauss and others that the pairwise Euclidean distances between n points in Euclidean space get distorted by at most $(1 \pm \epsilon)$ with high probability, when projected "randomly" to a space of dimension $O((\log n)/\epsilon^2)$. There are many natural choices for the random mapping here.

Part 1 of the book deals with combinatorial optimization. Chapter 2 begins with the seminal work of Goemans & Williamson on MAX CUT, where the "random projection" is used in the rounding step via a random hyperplane. The subsequent semidefinite-programming-based breakthroughs on MAX k-CUT and coloring are also presented in detail. The author then presents Borgain's theorem for projecting points in an arbitrary metric space into Euclidean space, followed by the max-flow min-cut ratio for multicommodity flows. This part of the book then concludes with Feige's seminal volume-respecting embeddings, preserving point-to-subset distances, and applications to layout problems.

Part 2 is about (PAC) learning, and is primarily about learning functions of half-spaces; in particular, as mentioned in Chapter 6, intersections of half-spaces can be used in principle to approximate any convex set. Chapter 5 deals with the question of how to learn complex, information-rich concepts using a relatively few examples, as the human brain seems to do. The argument is made that the human brain, say, can learn such concepts when they are *robust* to noise: informally, when each example is binary – labeled "positive" or "negative" – modifying the attributes of an example by an amount bounded by a parameter ℓ should not change the label of the example. Now suppose we want to learn a half-space when the distribution on example points is robust: i.e., the example points given to the learning algorithm are bounded away from the boundary of the halfspace. Random projection is used to reduce the dimensionality of the problem, and then the classical "Perceptron Algorithm" is used for the low-dimensional problem. One of the main efforts

undertaken is to make the random projection "neuronal" – computable by simple neural networks – in an attempt to go back to the original motivation for modeling how our brains may plausibly conduct such tasks. Chapter 6 is on learning the intersection of k half-spaces. To get around the hardness of the problem, two allowances are made: (i) the example points can come from any distribution that is "non-concentrated" – i.e., the density function lies in the range $[1/poly(n),\ poly(n)]$ everywhere on the unit ball in n dimensions; and (ii) the output hypothesis is allowed to be an intersection of $O(k)$ half-spaces, instead of exactly k. Even the case of a special non-concentrated distribution – the uniform distribution – had been a major challenge, until its resolution by Blum and Kannan. This chapter was to me the technically most challenging part of the book.

Part 3 is about information retrieval, an area of obvious current interest, and where one naturally expects dimension-reduction to be useful. The approximate-nearest-neighbors problem is first studied: one of the main tools of interest developed here is random projection for vertices of the *hypercube*, which then leads to approximate-nearest-neighbors algorithms for the hypercube (which is a natural domain in this context). It is also shown how one can reduce the Euclidean-space version of the problem to the "Hamming distance and hypercubes" setting. The idea that works here (a variant of an idea due to Kushilevitz, Ostrovsky, and Rabani) is to do several randomized embeddings of various subsets (balls) of the given database points. Low-rank approximations of matrices, a fundamental tool used, e.g., in Latent Semantic Indexing, are considered next. Rather than apply a Singular Value Decomposition (SVD) directly to the given matrix, one first applies a random projection to reduce the number of rows, and then applies SVD to the resultant (normalized) matrix. However, further work (e.g., of Frieze, Kannan and Vempala using random sampling) has led to further speedups here. The last topic considered is geometric p-median, where we aim to find the p "best" centers for a geometric point-set in high dimension.

3 Recommendation

I would recommend this book highly to any researcher in theoretical computer science, probability and statistics, discrete mathematics, and related areas. It provides good coverage of an important topic, and can be used in many graduate-level courses (e.g., randomized algorithms, approximation algorithms, or learning theory in Computer Science departments). A few weeks spent absorbing the material would be highly rewarding.

The author mentions a possible future edition of this book, at the end of Chapter 3. I have the following comments which I believe will improve the book's readability. My main comment is about high-dimensional-geometry background. The typical reader of the book is likely to be someone working in Theoretical Computer Science; I believe most of us do not get a formal grounding in high-dimensional geometry (this is certainly true of me). With this in mind, it would help if some appropriate proof approaches and reading material can be suggested. As an example, I wouldn't have known the background for the inequality involving "$Vol(P - P)$" in page 45 (apart from the easy special case of centrally-symmetric bodies). Also, the geometric arguments of Section 6.3.1 took a while to digest. More importantly, I think several readers would not even have an idea of how to approach, say, extremal inequalities in geometry; this is because the typical Theoretical Computer Scientist's math background is often restricted to discrete math, logic, probability, linear algebra, and elementary algebra and number theory. While there is a bit of geometric background given in the appendix, I think a web-based companion to the book – even a few pages which give some intuition into relevant proof approaches in high-dimensional geometry – will be valuable.

I also have some minor comments. First, an Index would help. Also, Chapters 5 and 6 can be integrated better: e.g., "homogeneous halfspaces" are repeatedly defined in these chapters. Finally, in terms of references: (i) The discussion on derandomization in Section 2.3.3: the paper "Algorithmic derandomization via complexity theory" by D. Sivakumar in STOC 2002, shows how such derandomizations can almost be read off from Nisan's generator. (ii) Part (2) of Theorem 5.3 can be strengthened – one only needs good bounds on moments up to a certain value. See *Randomness-efficient oblivious sampling* by M. Bellare and J. Rompel (FOCS 1994) and *Chernoff-Hoeffding Bounds for Applications with Limited Independence* by J. P. Schmidt, A. Siegel, and A. Srinivasan (*SIAM J. Discrete Math.*, 1995). (iii) Reference [92] needs a year.

PDF | On Jan 1, 2004, Benjamin C. Pierce published Advanced Topics in Types and Programming Languages | Find, read and cite all the research you need on ResearchGate.Â All content in this area was uploaded by Benjamin C. Pierce on Aug 20, 2014. Content may be subject to copyright. Advanced Topics in. Types and Programm ing Languages. Benjamin C. Pierce, editor. The MIT Press.Â As explained in TAPL, Chapter 24, type systems involving existentially quan-. tiï¬ed type variables provide a useful foundation for explaining and relating. various features of programming languages to do with information hiding. To establish the properties of such type-theoretic interpretations of infor-. mation hiding requires a theory of semantic equivalence for expressi ons of. Types and Programming Languages book. Read 17 reviews from the world's largest community for readers. A comprehensive introduction to type systems and pr...Â This text provides a comprehensive introduction both to type systems in computer science and to the basic theory of programming languages. The approach is pragmatic and operational; each new concept is motivated by programming examples and the more theoretical sections are driven by the needs of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web.Â That is why I picked up the Benjamin Pierceâ€™s book after many years of random embarrassments. I am glad that I did it. The book is really a fun read. This excellent book uses types to navigate the rich variety of programming languages, bringing a new kind of unity to their usage, theory, and implementation. Its author writes with the authority of experience in all three of these aspects." - -Robin Milner, Computer Laboratory, University of Cambridge.Â This text provides a comprehensive introduction both to type systems in computer science and to the basic theory of programming languages. The approach is pragmatic and operational; each new concept is motivated by programming examples and the more theoretical sections are driven by the needs of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web. This item:Types and Programming Languages by Benjamin C. Pierce Hardcover CDN$ 102.12. In Stock. Ships from and sold by Amazon.ca. FREE Shipping. Details.Â Types and Programming Languages is carefully written with a well-balanced choice of topics. It focusses on pragmatics, with the right level of necessary theory.Â This text is perhaps the most accessible yet thorough introduction to type systems I've encountered. On the one hand, it offers excellent grounding: practical motivation is provided, numerous examples illustrate the concepts, and implementations are provided which can be used to typecheck and evaluate these examples. At various points, extended demonstrations of the type systems under consideration are given (e.g. showing how objects may be encoded).