Book reviews

DAVID SKILLIKORN
Foundations of Parallel Programming. Cambridge University Press. 1994. ISBN 0-521-45511-1. £25.00. 197 pp. hardbound.

While this book contains, as the word 'foundations' in the title may suggest, much of interest to theoreticians, it aims nevertheless to provide a methodology which can be used to develop efficient parallel programs and, moreover, to do this in a context independent of particular hardware architectures. One of the main claims of the book is that it is possible to decouple parallel software from parallel hardware so that the software can be moved from one parallel architecture to another.

Based on categorical data types which provide a structured way to search for algorithms and structured programs to implement them, the key idea is that of distributing computation over a data structure so that calculations can be carried out on the components of the structure in parallel. This is done by the use of homomorphisms over the structured type constructors:

$$h(a \bowtie b) = h(a) \otimes h(b)$$

If such a homomorphism can be found, h can be distributed over a structure built by using the $\bowtie$ constructor and h can then be executed in parallel at all the leaves of the data structure. As well as developing a general theory based on these ideas, the book looks at how they apply to particular data types such as lists, trees and arrays.

The categorical data type model also facilitates the development of formal methods for traditional software engineering purposes such as reasoning about complexity, correctness and cost measures. Two chapters are devoted to software development by transformation using equational reasoning. One of the main advantages of the categorical data type approach over traditional abstract data types is claimed to be that the former guarantees that the set of transformation rules derived by the method is complete.

As well as the development of categorical data types described above, the book contains useful chapters on other approaches to parallel software, to the main parallel styles of machines and to their architectures, and to development of a cost calculus for lists. An appendix gives a brief historical background.

The book is by no means a lightweight read but careful study and rereading is rewarding. As the author readily admits, much of the material is still at a research stage, but there is also plenty in the way of software methodology which is immediately useful.

A. DAVIE
University of St Andrews

IAN PARBERRY
Circuit Complexity and Neural Networks. The MIT Press. 1994. ISBN 0-262-16148-6. £40.50. 270 pp. hardbound.

This book is aimed at researchers and practitioners working in the area of Artificial Neural Networks, one of its primary purposes being to consider this model within the framework of classical computational complexity theory. The opening chapter introduces the ideas underlying computation by neural networks and discusses the issue of scalability: quantifying how the complexity of a neural network increases with respect to increases in the size of the problem being solved. Chapter 2 reviews classical models of computation and computational complexity theory. In this chapter the use of Boolean combinational circuits as the vehicle for comparing neural network complexity with an established model of decision problem complexity is justified and a lucid discussion of the problems of uniform and non-uniform models is presented. Having established the foundations for the comparative study the remaining chapters describe increasingly sophisticated circuit models—*alternating circuits, threshold circuits, probabilistic models, Boltzmann machines*—and relate these to the issue of scalability in the neural network model.

This book is of interest not only to those working in the field of neural networks but should also be found to be of value to researchers in the area of computational complexity theory in general and the complexity theory of Boolean circuits in particular. In the latter area, Chapters 5–7 of the book provide a valuable summary of recent results in an important subfield of Boolean complexity theory. While a certain degree of mathematical awareness is required, the results and ideas presented are clearly described and easily accessible.

PAUL E. DUNNE
University of Liverpool

RANDY M. KAPLAN
Constructing Language Processors for Little Languages. John Wiley. 1994. ISBN 0-471-59754-6. £41.50. 452 pp. softbound.

This book takes a refreshingly new look at the problem of developing software and provides a neat, formal, proven solution. Language processing technology is often left to the world of systems programming to the extent that the application world would like to have none of it. Contrary to the popular opinion, it can make a world of difference even in developing large applications due to its sound mathematical basis. Technologies like parsing and lexical analysis, to name but a few, have proven tools available which make the task of a

Downloaded from https://academic.oup.com/comjnl/article/38/9/747/362970 by guest on 09 April 2024

programmer very easy. In language processing, we define the process to be a sequence of phases with each phase being catered to by a tool specifically designed for that phase. Such a clear separation of issues along with tool based development leads to the following advantages:

- Assurance of quality.
- Increased productivity.
- Increased opportunities for reuse.

The book introduces the concept of and the need for a little language. (I can hardly agree more!) This starts with just the basic concepts necessary for designing one's own language and suggests possible scenarios for taking this course of action. One of the highlights of the book is the way it explains the construction of a language processor for such a little language, the operative word being *construction*. It then gives a brief account of lexical analysis, syntax analysis and interpretation. It keeps away from the algorithms and discussions thereof. Wherever possible, a C implementation is provided, which urges the reader to go ahead and try using it. The book stops short of providing everything in code form and makes references to already existing tools. Most of the concepts introduced are explained through a (not so) little language IML. Implementations of a lexical analyser and parser for IML are provided in C. My only complaint would be that Lex and YACC specifications would have sufficed. Perhaps the author decided to be platform-independent.

The book is well-written and is easy to comprehend. Chapter 0 is an introduction and presents the author's viewpoint. Chapter 1 introduces the concept of a little language. Chapter 2 states the principles of language design. Chapters 3 and 4 provide the theoretical background for lexical analysis and parsing. Chapter 5 outlines the architecture of a typical language processor. Chapter 6 discusses the data structures used for implementing a language processor. Chapter 7 presents an implementation of a lexical analyser and a parser. Chapter 8 describes Lex and YACC. Chapters 9 and 10 discuss related issues such as interpretation, compilation and debugging.

Having myself have devised such little languages and engineered their processors in fields not necessarily close to language processing, the book comes as a re-affirmation of values in programming which have meant a lot to me.

VINAY KULKARNI
Tata Research Development and Design Centre
Pune, India

TIM BIENZ AND RICHARD COHN
Portable Document Format Reference Manual. Addison-Wesley. 1993. ISBN 0-201-62628-4. £20.95. 214 pp. softbound.

The PDF is Adobe's standard for multiple format documents, based on the PostScript typesetting language and designed to give even more machine independence than PostScript. It includes all the features of Level 2 PostScript, together with support for document organization as pages within a tree, annotations and hypertext support. The book defines these extensions to the language and the ways in which they are stored in the PDF files, as well as giving a basic introduction to PostScript.

The PostScript introduction is possibly not as detailed as it could be, but the book is not intended to teach the language and anyone wishing to learn PDF from scratch will probably need a PostScript textbook too.

The sections of the book on PDF itself are, of course, very detailed. The language extensions are very complex and the book discusses each of them to a level sufficient for anyone to write interpreters for the format or to produce documents in the format.

The initial chapters introduce the terminology and techniques used, followed by chapters discussing the main components of the format such as the page tree and how it is built. The chapters on optimizing PDF files are in themselves very useful since most of the techniques apply to PostScript programming in general. These chapters take up the last section of the book.

The appendices are up to Adobe's usual standard, and include the canonical 'Hello World' in PDF and another PDF example showing the updating of sections of a file. There is also a discussion of PDF's limitations (such as the file size not exceeding 10 Gbytes) and the limitations imposed by the devices the file may appear on.

The index is possibly not as large as one would have hoped, but seems to have everything needed in it, arranged in a usable fashion.

K. LUCAS
University of Warwick

JOHN R. JOSEPHSON AND SUSAN G. JOSEPHSON (eds)
Abductive Inference. Cambridge University Press. 1994. ISBN 0-521-43461 £30.00. 306 pp. hardbound.

Sherlock Holmes was wrong—his success was due not to logically valid deduction, but instead to abduction, a fallible but critical inference procedure. That he was not able to classify correctly the technique he used merely serves to illustrate how little known and how poorly represented abductive inference is in the literature. In this book, the authors forcefully argue for the centrality and pervasiveness of abductive reasoning in both science and everyday life, and describe their efforts to develop a series of computer programs capable of such reasoning.

Abduction is most appropriately described as *inference to the best explanation*. That is, given a set of data, the task of abduction is to construct hypotheses that account for or explain that data. It is often assumed that abduction is just deduction in reverse, and though this has some truth to it, the explanations that arise from

Downloaded from https://academic.oup.com/comjnl/article/38/9/747/362970 by guest on 09 April 2024