speculations about the relationship between karma and backpropagation, there is a clear thread running through the argument and I found myself enjoying his thought provoking style.

In summary, the book has an enormous variety of material (making it especially difficult for a reviewer to give an objective and well-rounded assessment!), most of which is both readable, accessible and interesting. There are reservations I have outlined in some of the aims of the book, but my overall assessment is a very strong commendation of a work which genuinely attempts to be multidisciplinary. This is a pioneering approach which deserves both exposure and at the same time our indulgence, if in its enthusiasm it occasionally steps over a distant boundary a little too clumsily.

J. Shawe-Taylor
Royal Holloway
University of London

REFERENCES

Howard, R. (1990) *Dynamic Programming and Markhov Processes*. MIT Press, Cambridge, MA.
Linnainmaa, S. (1970) *The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors* (in Finnish). Master's Thesis, Department of Computer Science, University of Helsinki, Helsinki.
Minaky, M. and Papert, S. (1988) *Perceptrons, expanded edn* MIT Press, Cambridge, MA.
Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986) Learning representations by back-propagating errors. *Nature*, **323**.

John Epton (ed.)
Expert Systems and Optimisation. Avebury Technical. 1994. ISBN 0-291-39809-X. £49.50, 293 pp. hardbound.

In the process industry, chemical and process plants are becoming increasingly complex. There are moves towards increasing the flexibility of the manufacturing operations and of minimizing stocks of finished products. Simultaneously, customers are becoming increasingly demanding in terms of their requirements for rapid delivery of smaller quantities to ensure that their own stock holding is minimized. It is no surprise thererfore that many manufacturers in the process industries are looking for ways of optimizing the operation of their existing plant with minimal outlays. This book attempts to summarize the current state of the art in the application of expert and knowledge-based systems to achieve these aims.

The book consists of 21 papers organized into five sections. The sections are as follows, with a brief summary of the topics covered. Part 1 provides an overview with one paper on intelligent systems for modelling and control and a second providing an overview of AI in process control. Part 2 covers process control with three papers. The first examines the role of expert systems in control engineering, the second automatic tuning of PID regulators with the third discussing AI and future trends in process control. Part 3 covers process monitoring, with five papers covering topics such as sensor data validation, fault diagnosis, AI approaches to process supervision and the role of expert systems in time series prediction. Part 4 covers optimization with seven papers on such topics as expert process scheduling, knowledge based planning systems and the use of advanced optimization techniques in process control. Part 5 comprises four papers by industrialists presenting case studies.

The book provides a fair coverage of the application of expert systems to process planning, monitoring and control, concentrating mainly on applications rather than the latest theoretical work. As is common in such collections of papers, the papers vary considerably in both their depth and attention to detail. However, this book would be a useful addition to a university reference library, showing the state of the art in industrial applications of expert system technology.

C. Czarnecki
De Montfort University

John Elder
Compiler Construction—A Recursive Descent Model. Prentice Hall International (UK) Ltd. 1994. ISBN 0-13-291139-6. £19.95. 437 pp. softbound.

Authors of books on compilers have a dilemma: should they focus on underlying theory in the exposition or should they stress the programming exercise? Gries' early book with the same title was unique in that it blended both. Perhaps there has been considerable increase in underlying theory as well as the complexity of features in programming languages which make attempting the blend a tall order.

This book has focussed on the program construction aspect of compilers at the expense of recent developments in theory. It is a detailed exposition (extending over 19 chapters) of the design and construction of a compiler for a language called MODEL, a subset of MODULA-2, that generates code for a hypothetical general-register based machine called TARGET. An advantage of a running example through successive phases of compilation, resulting in a complete compiler, is that all pertinent aspects of compilers needed by the example are discussed; this is a general lacuna in books which focus on the underlying theory—they do not have enough glue to aid in putting it all together, leaving first-time compiler writers short on directions. A disadvantage accruing from a sharp focus on the design and construction of a single compiler is that it is limited in its coverage by the chosen example. Such a book is useful only to introduce student-programmers to the craftsmanship of compiler construction. Graduate students of Computer Science, and professional compiler-writers, both need a sufficient dose of evolving theory so as to improve it correspondingly through research, or through better engineering by deploying technology based on such theory.

As MODULA 2 is the chosen compiler implementation language, the presentation of the design and implementation exhibits better delineation of information pertinent to the different phases of compilation. This is a decided improvement over books which have presented similar designs expressed in PASCAL or C, as they do not have even first-level support for abstract data types. The use of MODULA 2 allows locality in modelling of information pertinent to different compilation phases and activities, such as symbol table definition and use, checks for well-formed MODEL program structures in semantic analysis, and code generation in the separate chapters on these topics. Integration effected through IMPORT of these modules is relatively clean, compared to the distribution of such information in older recursive descent based compilers for PASCAL (which led to a stepwise enrichment of phase-based information structures in all of the recursive-descent parsing procedures in these compilers).

An important part of the book is the collection of exercises at the end of each chapter. These exercises are themselves a source of learning, as they extend the discussion of compilation to aspects of programming languages that are not a part of MODEL.

The presentation in terms of content and style is a hop-step-and-jump from the structured programming movement that started over 25 years ago. The sprint for this event started with the author's involvement in porting Ammann's CDC 6600 Pascal Compiler (developed by stepwise refinement based on recursive descent parsing) to the ICL 1900 series, with two other books in this series co-authored by his senior collaborator, Jim Welsh, as the hop and the step. It is in this regard that the book falls short.

The proponents of structured programming have steadily moved on to increasingly formalized descriptions of the programming process. The directions they have set for themselves is to establish a science of programming. Taking this spirit forward requires that even complex programming problems, such as compiler construction, be addressed formally. Hence, there is no escape from attempting to use underlying programming language theory in any discussion on compiler construction, regardless of the current state-of-the-art of this theory, or the universal acceptance of any particular formalisation. Such a discussion shows up the scientific versus intuitive aspects of compilation. It is relative to their theory that the compilation function as a syntactic transformation under semantic invariance should be modelled. Such a discussion could also provide a basis for expressing engineering design concerns in the efficiency, robustness, and fault tolerance (error recovery!) in compilers. All these properties are clearly enjoyed by the developed compiler, for it presents a stable architecture that has evolved over time. Making this formally explicit is a responsibility of proponents of good programming practices.

Another point of note: by perpetuating an old style of handcrafting of programs, the programmers of the future will get rooted into history. Automatic generation of compiler components through use of compiler–compiler tools is widely accepted in practice and readily available in standard environments such as UNIX. Indeed, it is the most pragmatic fallout from improvements in the theory of programming languages. Expressing compiler design and construction through use of tools allow the designer to concentrate on programming language specifications, rather than on programming. After all, formal specification of programming languages is motherhood to several formal specification approaches. Positive experience in use of this style can only rub off on other programming exercises as well.

Till the arrival of the first solutions to retargetable code generation methods, the structure of compilers was completely dominated by the structure of source languages. Today, the idea that code generation can be modelled as a tree-pattern-matching based covering-problem is widely accepted. Systematic approaches to code selection can be explored—even through use of parsing technology. Locally optimal code selection is possible without resorting to complex flow analyses, all through modelling of target machines, and using tool-based approaches to deriving the code selection part of code generation. Posing some aspects of code generation to be local optimisation problems, and others (such as global register allocation or instruction scheduling for a pipe-lined RISC processor) as global optimisation problems, introduces a sharper focus on modelling of engineering properties of generated code.

The use of tool based approaches in teaching compiler construction provides a path for scale-up from classroom toy exercises to full-blown programming languages and industrially available target machines.

Today's compilers interface to a comprehensive programming environment, consisting of editors, browsers, debuggers and program composition facilities. Some of these aspects have been mildly dealt with in this book. The code generated by today's compilers also need to interface with a wide variety of extra-linguistic platforms, such as operating systems through run-time support libraries, database systems, graphical user interface management systems, etc. They also need to accommodate use of program fragments written in other programming languages. As pragmatic matters in compiler construction, they too need to be addressed.

Kesav V. Nori
Tata Research Development and Design Centre

Julian Ullmann
Compiling in MODULA-2. Prentice Hall. 1994. ISBN 0-13-088741-2. £19.95. 425 pp. softbound.