After a short introduction, the book opens with a comprehensive account of the elements of fuzzy set theory and fuzzy logic. The exposition is clear, precise, nicely paced and comprehensive. For ease of reading, references are collected together in a final section which includes some supplementary remarks. These cover both historical material, an example application treated in depth, and pointers to research issues which arise from the earlier material. This final section concludes with some exercises which are designed to clarify and deepen understanding of the technical material presented in the chapter.

The same format is followed for the remaining two chapters. These are on possibilistic reasoning, including details on knowledge representation and efficient implementation, and fuzzy controllers, one of the major application domains.

The sum total is a very high quality text which advanced students, researchers and practitioners with an interest in the theory and application of fuzzy sets will find invaluable. The authors also address one of the remaining criticisms of the fuzzy set approach by including detailed analyses of the semantics of the underlying concepts as they are introduced. So, those that are still sceptical of the "fuzzy paradigm" might also benefit from reading this book!

Reviewed by Paul J. Krause, Advanced Computation Laboratory, Imperial Cancer Research Fund, 61 Lincoln's Inn Fields, London WC2A 3PX, UK

The Gödel programming language by Patricia Hill and John W. Lloyd, The MIT Press, 1994, pp 337, £40.50/$60.75, ISBN 0-262-08229-2.

For more than 20 years, the logic programming world has been dominated by the Prolog programming language and its many more specialized offspring. While this has, in some ways, led to a usefully focused community of users and researchers, it has also given rise to some false assumptions and conclusions both within and outside the logic programming community.

The most insidious of these assumptions is that Prolog and its siblings are, somehow, the be-all and end-all of logic programming languages, which is surely not the case. While Prolog is very well suited for many styles of programming work, such as rapid prototyping, knowledge-rich applications, and so on, it has drawbacks which cannot be ignored and which should not be thought of as representative of logic programming in the large. One obvious such drawback is floundering under negation, where a variable becomes bound during a program execution, but the binding is hidden from the rest of the program, leading to incorrect answers. Another is the lack of a declared language for logic programs: I dread to think how many hours I have wasted debugging trivial spelling mistakes in the names of (undeclared) Prolog atoms.

A more subtle problem, particularly for users not familiar with the ethos of logic programming, is the fixed left-right computation rule, which allows us to view the procedural, rather than the declarative, interpretation of programs as paramount. Reliance on the fixed computation strategy often results in Prolog programs which might as well have been implemented in C. This is, of course, missing the point of Kowalski's original equation "Algorithm = Logic + Control" (Kowalski, 1979); in logic programming, we want to think of the logic only, and let the control be looked after as automatically as possible.

There is clearly a long way to go before we reach Kowalski's vision of a programming system where issues of efficiency can be left entirely to the computer and the task of programming becomes that of simply stating the problem to be solved as clearly as possible. Even so, the current state of the art in programming technology means that we can do a lot better than Prolog, and the Gödel language, described in this new volume from MIT Press, is one example of how to do so.

Gödel is aimed at superceding the "core" versions of Prolog. That is to say, it is a mainstream, non-specialized programming language, which can and (I do not doubt) will be specialized in various ways in future. The main difference between Prolog and Gödel is that the Gödel system is significantly more carefully thought out in theoretical terms. Any program written in Prolog can be

rewritten in Gödel with little more effort than declaring the symbols used. More specifically, the major changes are as follows.

Types The logic implemented in Gödel is a first-order, *many-sorted* one. Sorts—or rather, in Gödel nomenclature, types—are parametric and user-definable, allowing symbol overloading, and typing is strict, with the requirement that all symbols appearing in a program be declared. The type system is based on the Hindley–Milner system (Milner, 1978), and well-typedness of any program call with respect to its program is checkable statically, so we don't lose run-time efficiency. The consequence of the introduction of types is that the Gödel programmer is forced to think a little more carefully and in advance than her Prolog counterpart, with the result that rapid prototyping becomes more so—Prolog's inscrutable "no" in response to a mis-spelled atom name is replaced in Gödel by a clear error message from the type-checker. Another advantage of Gödel is that some types are implemented according to particular (high) standards, such as ANSI integer arithmetic. There's no reason why this could not be done in Prolog, but, as far as I know, it hasn't.

Control The specification of the Gödel language does not define a computation rule. Instead, `DELAY` declarations, like the `WAIT` declarations of Nu-Prolog (Naish, 1986), are given to determine, in terms of the contents of parameters, when to execute parts of a program. It is possible using this system to arrange that Gödel version of incorrectly non-terminating Prolog programs will be trapped. The main advantage of all this is that it leads the programmer to think in a more declarative style: it is impossible to rely on one call being executed after another, and so the program must be thought of purely as a relation between its inputs and outputs.

Meta-programming The biggest advance over Prolog comes in Gödel's meta-programming facilities. Gödel includes (among others) an abstract data specifying Gödel syntax, and defining all the predicates needed to construct, manipulate and execute any Gödel program. The use of strong typing means that the mixture of meta and object level work which was (often confusingly) possible in Prolog is no longer possible in Gödel—the distinction between code and representation of code is absolute. There is not space to explain all the advantages of the Gödel meta-programming system here; the interested reader must refer to the book!

Modules Finally, there is a module system. Regrettably, it is not parametric, but it works as well as the module system of any other logic programming language I have seen, and it admits the nice implementation of type standards I mentioned above.

One very important result of the design of Gödel is that automated program manipulation becomes very much easier than in Prolog. Therefore, it is possible to overcome one of the major barriers to "efficient" logic programming. We can automatically convert a declarative program written with little regard for operational behaviour into a less readable, but more efficiently executable, program before we execute it. One way of doing this is with the SAGE partial evaluator (Gurr, 1994)—the first fully self-applicable partial evaluator for a logic programming language. A version of SAGE is bundled as part of the Gödel implementation available from the University of Bristol, in England (Gödel).

So, having said all this about the subject, what about the book? *The Gödel Programming Language* is written by two of the designers of the language, as a reference manual. It has evolved over a period of time from the original specification of the language, and therefore contains a really complete coverage. It is divided into two main sections, the first being a friendly, reasonably verbose explanation of how to use the language—not as a programming tutorial, but as a showcase of the available facilities. There are plenty of examples, all of which are supplied with the Bristol implementation. The second part is a fully formal, terse description, covering all aspects of the language, in precisely the way one would want from a language reference book.

Of course, this division of the information into two levels, rather than into subjects, means that there is repetition. Even so, I feel the idea is good: the reader may enter at her own level for each topic, and I found it easy to switch between the two sections to clarify informally the things I didn't

immediately pick up from the formal section. Also, for those less familiar with using logic as a programming language, there is a short appendix covering some useful issues.

The description in the book satisfyingly follows its own espoused methodology, and sticks to declarative readings and explanations wherever possible; this, surprisingly, is not always the case with logic programming books. The text is not too formal, and is comfortable to read, with, I find, the pace being judged just about right. Also, once one becomes a bit more familiar with the topic, there is a reasonably good index, for quick access—though I would have liked the "system index" to have been organized by predicate function rather than name (to be fair, the naming convention is sufficiently sensible that these are often the same thing).

In short, I think the book succeeds pretty much completely in being what it is intended to be: an accessible but detailed, digestible but complete, guide to the facilities of an exciting new programming language. It will certainly be indispensable to the users of the Gödel system—it is *considerably* more useful than many programming reference books I have had the misfortune to use in the past. It will also be useful to teachers and students, not just for Gödel itself, but for logic programming as a whole. Here, at last, is a logic programming language which is designed from the bottom up, with the features that most programmers view as necessary for efficient and correct programming; and with it a book to do it credit.

If I have one serious misgiving, it is not with the book itself, but with its publisher. The book is published only in hardback, and is correspondingly expensive. To do the logic programming community, and indeed itself, a service, MIT Press should be producing this kind of volume in softback, so it becomes more easily accessible *en masse* to students and other users. This sort of reference book is of little use in a library—you need your own copy next to you as you write programs. Gödel needs the book, and as Gödel use increases, more books will be sold. But at these prices, there's a real hindrance to book sales and, indirectly, to Gödel uptake. This is really a shame.

References

The Gödel programming language. Available by ftp; details from goedel©compsci.bristol.ac.uk.

Gurr, CA, 1994. "A Self-Applicable Partial Evaluator for the Logic Programming Language Gödel." Unpublished PhD thesis, Department of Computer Sciences, University of Bristol.

Kowalski, R, 1979. "Algorithm = Logic + Control." *Communications of ACM* **22** 424–436.

Milner, R, 1978. "A theory of type polymorphism in programming." *Journal of Computer and System Sciences* **17** (3) 348–375.

Naish, L, 1986. *Negation and Control in Prolog, Vol. 238, Lecture Notes in Computer Science.* Springer-Verlag.

Reviewed by Geraint A. Wiggins, Department of Artificial Intelligence, University of Edinburgh

Advanced methods in neural computing by Philip Wasserman, International Thomson Publishing (Van Nostrand Reinhold), USA, 1993, ISBN 0-442-00461-3.

The upsurge of interest in neural networks literature over the last decade is a reliable sign of the increasing popularity of a field that promises to yield answers to questions resisting satisfactory solutions when attacked by cybernetics, finite-machines theory and artificial intelligence, to name just a few of the attempts to build "smart" machines.

The crucial feature that makes this discipline attract the attention of undergraduate students as well as of established scientists is probably the seemingly inexhaustible reservoir of novel ideas and techniques that it offers. Indeed, the number of journals and conferences dedicated to this paradigm grows so fast that it is sometimes difficult to keep pace with the forefront of the current research.

The simplest way to find one's way through the labyrinth of new articles and products is to pick a moderately sized textbook that attempts to establish some order in the material and to emphasize the essential approaches that a beginner should master before probing further.

But here the problem begins. Those who have lectured this subject know how difficult it is to select a proper monograph. Some are too introductory and tend to be superficial, others are so