A more consistent attitude would have been to discard knowledge of the lambda calculus as a prerequisite in favour of an in-depth knowledge of ML. But then it would be hard to justify the minimalistic attitude.

Jørgen Steensgaard-Madsen
Technical University of Denmark
Lyngby, Denmark


The book, *The Practice of Prolog*, edited by Leon Sterling comprises a collection of eight papers reporting on practising Prolog, originally A. Colmerauer’s language. The market for this book are people with a nonnumerical problem who want a computer solution, and are willing to first spend some time with a Prolog text book, have they not done so already. In fact, I think it would be a good idea to publish a few more books on this subject. “The raison d’etre of this book”, writes the editor, “is to encourage programmers to use Prolog in their day-to-day work. Personally, I also find the language exciting to use, and wish to share the excitement with others.” I share this opinion on the better Prolog systems, e.g., D. Warren’s, especially where they play the role as a logic programming language following R. Kowalski. Unfortunately, Prolog sometimes plays less celebrated roles that can be disappointing for its champions. Let us now turn briefly to the individual chapters.

Chapter 1: Prototyping Databases in Prolog, by T. Kaziac, E. Lusk, R. Olson, R. Overbeek, and S. Tuecke. This is a chapter on an area that probably soon will find a manifold of applications. There are several good illustrations by programs how to build up a database, and the authors make the point that when you have your prototype you might just discover your production system.

Chapter 2: Preditor: A Prolog-based VLSI Editor, by B. Reintjes. In this chapter we can find several practical Prolog programs for a VLSI editor. One of the reported advantages with Prolog is a “tenfold reduction in source code size with considerable expanded capability when compared to traditional CAD tool implementations.”

Chapter 3: Assisting Register Transfer Level Hardware Design: A Prolog Application, by P.J. Drongowski. Here is a Prolog implementation compared favourably with C for hardware design.

Chapter 4: Developing a Portable Parallelizing Pascal Compiler in Prolog, by E. Gabber. It is generally known that Prolog is a good language for writing compilers. The author’s main point is that this is still true for a compiler of Pascal to a virtual machine for multiprocessors running in C. The main parallelization of the Pascal program is to find suitable loops that can be run in a parallel version called single program multiple data mode.

Chapter 5: Promix a Prolog Partial Evaluation System, by A. Lakhotia and L. Sterling, is on metaprogramming, in particular partial evaluation. This is still an
active research area. Many Prolog programmers have found themselves being metaprogrammers, who are unfortunately missing the proper metalogic concepts in Prolog. The authors write a pure interpreter for pure Prolog that is not all that pure metalogic. This interpreter, although expanded significantly, is the central idea of this paper. Metaprogramming is useful, as argued in this paper, on partial evaluation of Prolog, realizing Futamura’s original idea of 1971. Successful applications will, hopefully stimulate developments of metalogic notions, so Prolog could get a proper metalogic part in metaprogramming.

Chapter 6: Generating Natural Language Explanations from Plans, by C. Mellish. The author warns that this paper uses a somewhat old-fashioned notion of planning, and I would like to add ontology. However, it contains crisp Prolog programs that are state of the art. The idea of explaining plans in natural language is also interesting.

Chapter 7: A Simple Learning Program, by R. O’Keefe. Also this chapter is full of crisp Prolog programs that are state of the art, and thus worth studying although the reader might not be too keen on the subject itself.

Chapter 8: Stream Data Analysis in Prolog, by D. Scott Parker. Here is the flexibility of Prolog shown by a simple integration of Narian’s Log(F), giving a functional language as well and again we see some good Prolog programs.

Just as good literature is good for authors; good programs are good for programmers. This book contains a fine collection of practical Prolog programs, new and old, that good programmers could also enjoy. How many languages could show a similar collection of programs?

References


Carnegie-Mellon University is one of the big places in Computer Science in the United States of America. But it started in Pittsburgh like everywhere else; in small steps. What the result is depends on how many of these small steps have been done