Book reviews

doi:10.1017/S1471068405212449

Programming in Prolog. Using the ISO Standard. by William F. Clocksin, Christopher S. Mellish, Springer-Verlag, 2003, ISBN 3-540-00678-8, xiii + 299 pages.

This is the fifth and most recent edition of a legendary book whose first edition dates from 1981. It was probably the first introductory Prolog book and it still is the most gentle introduction to Prolog for everyone, including non-computer scientists. The authors make very few assumptions indeed about the computer science knowledge and programming skills of their readers. Even so, the book covers most of the Prolog language.

The first edition was based on the de facto Edinburgh Prolog standard. Obviously, with the appearance of the ISO Prolog Standard, a new edition of the book needed to adapt to ISO, and that is the reason for the subtitle *Using the ISO Standard.* Other changes were introduced as well and this review mentions some of the differences, but mainly concentrates on the current contents, independent of the history involved.

The Preface of the book sketches the rationale behind the book and the new edition. It also lists in a table differences between the new and the old editions related to switching from the de facto standard to ISO. These differences are related to syntax and built-in predicates. There are surprisingly few differences and they are small.

The book consists of eleven chapters

1. Tutorial Introduction
2. A Closer Look
3. Using Data Structures
4. Backtracking and the “Cut”
5. Input and Output
6. Built-in Predicates
7. More Example Programs
8. Debugging Prolog Programs
9. Using Prolog Grammar Rules
10. The Relation of Prolog to Logic
11. Projects in Prolog

and 4 appendices:

A Answers to Selected Exercises
B Clausal Form Program Listings
C Writing Portable Standard Prolog Programs
D Code to Support DCGs

The first chapter in any book on Prolog is the most difficult one: how to introduce gradually a whole bunch of connected concepts (and terminology) without getting sidetracked or forgetting certain essentials. The authors do a very good job here: after general remarks about objects and relationships, and the nature of programming, (ground) Prolog facts are introduced as a means to model the knowledge contained in simple English sentences. This is followed immediately by showing how such facts can be queried. Newbie questions in comp.lang.prolog show that at this point, the book would benefit from an explicit statement that facts like *likes(john,mary)*. are **not** in the Prolog database at startup (and refer to the relevant section in the book). This chapter next discusses variables, conjunctions (in queries), a first example where backtracking happens, and rules. At the end of this chapter, the reader is familiar with datalog. A set of exercises concludes this chapter.

Chapter 2 starts by taking a closer look at the syntax of terms and clauses. There is a lot of fine-print in Prolog syntax, but most programmers never need to know it, so the book justifiably does not go into all the details. That it does not mention the ISO syntax for character codes (like 0'a for the ASCII 97), seems however a more serious oversight, especially in view of the footnote on page 229 which suggests that the notation `"a"` for `[97]` should be used in some implementations: `"a"` is legal syntax but its meaning is not defined by ISO. Equality of terms (unification or matching) and arithmetic are also covered explicitly at this stage. Most importantly, chapter 2 introduces diagrams with boxes containing a goal and an arrow that illustrates the flow of control: this visualization of the execution of Prolog is extremely helpful and is later in the book used for explaining other concepts like backtracking, the cut and debugging. In the first edition, these diagrams looked like a wiggly snake in a Prolog database: the current form is much better.

Chapter 3 introduces data structures which are drawn as trees of Prolog terms and lists. The first recursive predicate now follows naturally: member/2. The chapter goes on with recursive predicates and compound data structures. At the end, we find two new sections on accumulators and difference structures.

There are two negative points about chapter 3: on page 56, the original edition contained an explanation which (rather strangely) links left-recursive rules to the order of rules. In the current edition, two code snippets were switched without adapting the text and it now makes even less sense than before. On pages 59/60, there is an albeit nice example which would have been worth reconsidering in the section on cut. Or at least in the section on cut, the reader could have been asked to reconsider the code on pages 59/60.

Chapter 4 explains in detail backtracking and the cut. Understanding backtracking is central to programming in Prolog and the explanation of backtracking is repeated in the context of several concrete programming examples later in the book as well. This is nice work: it makes use of the diagrams introduced in chapter 2, this time with nested boxes. The cut is also explained very well and with the same diagrams. This is also the natural place to introduce the Prolog negation `\+/1`. The authors devote some space to the common uses and the pitfalls of the cut. In particular they show that a cut should be placed before the output unification (an issue related to steadfastness), but too many of the example predicates in the book are not steadfast! Another critique related to cuts in the programs in the book, is that there are so many of them. The book would have gained a lot if the Prolog if-then-else construct would have been introduced in this very same chapter, and if subsequently a series of quite ugly programs had been cleaned up. One more negative point: on page 89, the text seems to argue that placing a cut in the body of the first clause (the fact) of append/3, will increase space and time efficiency. This is not true. It is understandable that the authors have chosen not to talk about first argument indexing, but that does not justify wrong statements. Another Example of that kind (related to the silence about indexing) can be found on page 146 where it is strongly suggested that Prolog must do a linear search in a database of ground facts, even if the first argument is instantiated.

A chapter about input and output is unavoidable in an introductory textbook on any programming language: it is never exciting and the current book makes no exception, but the example programs are OK. This chapter is adapted to the ISO standard, albeit in a minimalistic way: the authors stick to the DEC-10 *see/seeing* style which changes the current streams using the ISO built-ins current_input/1 and set_input/1, warning for the pitfalls, but failing to point out the more safe I/O predicates that take the stream as their first argument. At the end of this chapter, there is a list of *most important ISO Prolog operators*: for some reason the quoting of the operators is inconsistent and backslashes in quoted atoms were not doubled. Also, the ISO style of op/3 with a list of atoms in the third argument should have been followed. Still related to operators: on page 178 it is written that *we shall have to declare a "^" operator*; however, "^" is already an ISO operator. The subsequent declaration in the book gives a non-ISO precedence and associativity for "^"...

Chapter 6 goes into more detail about some built-in predicates some of which were covered already earlier in the book. To name just some new ones: var/1 (and friends), clause/2 (and friends), the functor/arg/univ triple, a series of predicates to transform atomic objects to a list of characters, term comparison...The section on dynamic predicates should have mentioned the logical database update view adopted by ISO Prolog. Page 132 says that the query `?- number_chars(23,X).` could deliver the result `X = ['2','3','.','0']`: that is not true in ISO Prolog. The section named Constructing Compound Goals (which deals with conjunction, disjunction, call/1 and `\+/1`) should have contained the Prolog if-then-else. The section on arithmetic lists some arithmetic operators and concludes on page 140 with the note: *"Particular Prolog implementations may include more arithmetic operations such as exponentiation and trigonometric functions"*. This is misleading as ISO Prolog defines both exponentiation and a whole lot of trigonometric functions, so every ISO Prolog must have them.

Chapter 7 contains a selection of classic programs: a sorted tree, searching in a maze, the towers of Hanoi, dealing with graphs...Both the problems and the solution programs are nicely laid out: this chapter can be digested by a novice and contains lots of interesting stuff. It is not clear why the authors choose to introduce (the implementation of) findall/3 as an example program, instead of as an extremely useful built-in predicate. Page 147 contains a figure that was adapted from the first version and it looks nicer now, but unfortunately, cut&paste has introduced errors, and made the figure inconsistent with the text. The programs are mostly OK, but cut is used too often and inconsistently. Examples of this are on pages 148 and 165. Many of these programs would benefit from a rewrite in an if-then-else style.

Chapter 8 is about debugging Prolog programs: ISO says nothing about it, but actual implementations do not differ much in their simple interface to the Byrd Box Model. This model is explained very nicely and example traces show how to use it. It includes spy-points and leashing. This chapter contains advice on program layout (which the book regularly does not follow), a section on Common Errors (whose first point is wrong: there is no need to have a return after the last dot in a program - EOF is enough), and a section on Fixing Bugs. The update of the latter section is already 10 years overdue: one cannot seriously describe the use of `?- [user].` (and its pitfalls) as the main method for fixing a program, given the current user interfaces.

Definite Clause Grammars are explained in chapter 9, starting from a simple English sentence analysis. It is a classic and satisfying explanation. The section about Translating Language to Logic is particularly nice, because through DCGs it shows the relation between natural language sentences and logic formulas: this is what logic programming is about!

Chapter 10 explains how Prolog relates to (pure) Logic. A reader who just wants to have a practical grasp on Prolog could skip this chapter, but nobody should be afraid of tackling it: the authors have done a very good job starting from an explanation of Predicate Calculus, going to Clausal Form, covering Resolution and Theorem Provers, finally arriving at Horn Clauses and ending with Prolog. This chapter should be reread regularly by old Prolog programmers.

Chapter 11 lists some small and medium-sized projects to be done by the interested reader: the selection shows some of the areas Prolog is good at.

Appendix A contains the answers to selected problems: Prolog code plus explanation. Not bad, but it should be cleaned up: indentation, layout and cuts are the main issues. Page 269 and 270 introduce some extra typos compared to the first edition, and it is suggested that the notation `{X}` is not ISO, while it is (page 290 states it correctly though).

Appendix B contains the code for translating a logic formula into clausal form: it follows the explanation of chapter 10, but now we see the full Prolog code.

Appendix C is about Writing Portable Standard Prolog Programs. The only sensible advice would have been to eschew anything that is not defined by the standard, and in particular the issues that ISO has left implementation dependent or implementation defined. Instead, the authors warn the reader that non-conforming implementations might exhibit non-ISO behavior, which is like the empty statement. A section which describes more clearly the differences between (say) DEC-10 Prolog and ISO Prolog would have been more helpful. The programs given in this chapter mimic ISO built-ins using predicates that were popular in the de facto standard: it seems better for the individual programmer not to use these programs.

Appendix D contains the full code for DCG translation. It is not clear why the authors wanted it in the book. Moreover the code and its comments still try to make it work in a non-ISO system: it is one of the examples of the ambiguity of the book which has not resolutely chosen for ISO.

Overall, the book is as great as ever as an introductory text for Prolog. When a newbie asks for an introduction to Prolog, the best advice is still Clocksin & Mellish. Leaving the ISO issue aside, the book would benefit from correcting (sometimes newly introduced) errors, cleaning up some programs (especially the cuts in them) and introducing (and using) if-then-else.

People attracted by the subtitle *Using the ISO Standard* could feel left in the cold by the actual ISO content of the book. There are many small errors related to ISO in the book. One more example: the footnote on page 182 suggests that an implementation that limits the applicability of clause/2 to dynamic predicates, is not ISO conforming. Since dynamic predicates are public, and since an implementation can limit public predicates to the dynamic ones, it follows that this is not true. There are too many such points in the book.

The book also often fails in *Using the ISO Standard* in some of its programs. As an example: the preface mentions explicitly that the arithmetic operators =:= and =\= are introduced (they were not in the first edition), but the programs on page 175 and 268 do not use these operators while that would have been the natural thing to do.

On top of that, there are at least two major areas in ISO that the book does not even mention: the first one is the issue of modules. On page 189, the book advises to split a large program over several files, but the concept of modules, neither the fact that ISO Prolog has also a part that standardized modules is mentioned. The second issue is the one of the ISO error mechanism: ISO Prolog prescribes it in detail and beginning programmers are confronted with it from the start because badly called built-in predicates throw an exception. The book should have explained the mechanism: it is not prohibitively difficult. Moreover, on pages 8 and 119, the book mentions only failure, failure+warning or error message as the ISO possibilities for calling a non-existing predicate or calling arithmetic on non-numbers.

In conclusion: this is a great introduction to Prolog, but it is weak on the ISO aspect.

Bart Demoen
K.U. Leuven, Belgium

doi:10.1017/S1471068405222445

Term Rewriting Systems by "Terese" (Marc Bezem, Jan Willem Klop, and Roel de Vrijer, eds.), Cambridge University Press, *Cambridge Tracts in Theoretical Computer Science* **55**, 2003, hard cover: ISBN 0-521-39115-6, xxii + 884 pages.

The formal study of rewriting and its properties began in 1910 with the pioneering work of Axel Thue, and has since developed into a major area of research. Broadly speaking, *rewriting* is the study of normal forms, their existence, their uniqueness, and their computation. As a discipline of computer science, term rewriting has two main application areas: functional languages and their semantics; and equational reasoning and mechanical inference. *Term Rewriting Systems* provides a very comprehensive treatment of the subject, mainly from the former point of view.

The rhetorical question, "Who is Terese?" the ostensible author of the volume, is answered in the prefatory pages. "Terese" is the pseudonym of the contributing editors, Marc Bezem, Jan Willem Klop, and Roel de Vrijer, and their colleagues, Erik Barendsen, Inge Bethke, Jan Heering, Richard Kennaway, Paul Klint, Vincent van Oostrom, Femke van Raamsdonk, Fer-Jan de Vries, and Hans Zantema, who all shared in the writing of this massive work. The authors were all, at one time or another, members of the TeReSe ("*Te*rm *Re*writing *Se*minar") group, which held regular meetings during the years 1988–2000 at the Vrije University in Amsterdam.

The genesis of this book went something like this: In the beginning, in 1985 C.E., Klop delivered a short course on term rewriting at a seminar in Naples. He reworked the unpublished course notes into a tutorial for a 1987 issue of the *Bulletin of the European Association of Theoretical Computer Science*. Then, in 1990, he gave a lecture on rewriting at the International Colloquium on Automata, Languages and Programming, held at Warwick University. This grew into a large chapter in the *Handbook of Logic in Computer Science* (Oxford Univ. Press, 1992), which was the twelfth most frequently cited paper in CiteSeer's gigantic database of online papers, circa 2000–2001, and remains high up on their list. Determined to expand this material into a full-length book, the Terese team of contributors was assembled.

Terese labored many years to bestow upon the scientific community a large and appealing volume, comprising sixteen chapters, plus an appendix. This is not a gentle introduction to the field (there are handbook chapters for that), but a text that is both broad and deep. One or more of the editors authored or coauthored ten of the chapters. Some are virtually full-length monographs; most are comprehensive discourses; a few are bare-bones sketches. This is the first book to provide rigorous coverage of topics such as standardization, neededness, and infinitary rewriting. Algorithmic issues are played down. The chapters are complemented by an encyclopedic bibliography of some 500 references, and an index.

Term Rewriting Systems starts off with an introductory Chapter 0, presenting three examples of rewriting, well-chosen for their heterogeneity: a functional program for integer division; Reidemeister moves for unravelling knots; and Jieh Hsiang's nondeterministic rules for computing the Zhegalkin (Boolean-ring) normal form of a propositional formula.

As has become customary, as many of the basic concepts as possible are introduced within the abstract framework of arbitrary binary relations. Thus, Chapter 1, by Bezem and Klop, defines (and fixes notations for) the fundamental notions of confluence, including the Church-Rosser property (CR), and of normalization – in both its weak (WN, that is, existence of normal forms) and strong (SN, or, uniform termination) forms. Then it goes on to enumerate the relations between the notions, most notably the 1942 result of Max Newman that local confluence (WCR) and strong normalization imply the Church-Rosser property. (Three proofs of Newman's Lemma are given *ad loc.* and one more in Chapter 14.)

Chapter 2, by Klop and de Vrijer, turns to the central theme of the volume: rewriting first-order terms. After defining the basic notions of term, context, substitution, (syntactic) unification, reduction (i.e. rewriting), and many more, it uses artful diagrams to explain the concepts of overlap and critical pair, and prove Don Knuth's famous Critical Pair Lemma.

Two important, early examples of term rewriting, namely Herbrand and Gödel's general recursive functions and Curry's Combinatory Logic, are explained in Chapter 3. These are followed by short sections on ground rewriting, recursive program schemes, many-sorted rewriting, string rewriting (a.k.a. semi-Thue systems), conditional rewriting (with equational Horn clauses), and the $\lambda\sigma$-calculus (the earliest of numerous recent proposals of a lambda calculus with explicit operations for performing substitutions).

The pivotal chapter is the fourth. In it, Klop, van Oostrom, and de Vrijer define *orthogonal* (left-linear and overlap-free) term rewriting systems and prove (again, in more than one fashion) that they are confluent, hence, have at most one normal form per term. The core ideas of descendants, residuals, developments, and reduction diagrams are clearly explained, and the central issue of normalizing strategy is introduced. (Regrettably, most of the proofs about normalization are deferred to a later chapter.)

Chapter 5, by Klop and de Vrijer, is in reality two chapters: The first proves that most interesting properties of term rewriting are – no surprise here – undecidable. The second part gathers many results on the question of the extent to which the union of two systems that share no function symbols or constants, but share a property like confluence or strong normalization, also enjoys that property. The two central results are a theorem by Yoshihito Toyama to the effect that confluence is preserved, proved in detail, and another that confluence plus strong normalization are preserved for left-linear systems, which is stated without proof. The book does not deal with cases in which constructors or other symbols are shared (early on recognized as important for practical modularity of programming considerations).

The termination chapter, by Zantema, is a long, somewhat haphazard survey of syntactic, semantic, and transformation-based methods of proving strong normalization (termination), including the recent dependency-pair method (*sans* proof).

Chapter 7, by Bethke, contains a very brief treatment – only 40 pages long – of equational inference, including the all-important Knuth-Bendix completion procedure, presented in the style of inference rules and proof orderings (as developed by Leo Bachmair), proof by consistency (invented by Dave Musser, who should have been cited), and equation solving (semantic E-unification). Many topics are not addressed, including "unfailing" completion and associative-commutative completion, despite their importance for automated deduction. Also, the contributions of Gérard Huet to our understanding of the rôle of completion as theorem prover and of Jieh Hsiang to the use of Boolean rings for first-order reasoning go unmentioned.

Chapters 8 and 9, both by van Oostrom and de Vrijer, comprise a very extensive and beautiful study (at almost 250 pages, nearly one third of the text!) of properties of derivations of left-linear systems and reduction strategies for orthogonal systems, including much new material. Labelling methods, tracing methods, various forms of reduction (derivation) equivalence, standardization, reduction strategies, sequentiality, "family" rewriting, and much more are covered in great depth.

Two digressions are included next: the basics of untyped and typed lambda calculi in a chapter by Bethke, followed by an introduction to higher-order rewriting and combinatory reduction systems by van Raamsdonk. Of course, full treatment of either would require a whole book unto itself. Still, they serve nicely to link rewriting to sibling fields.

Properties of infinitely-long reduction sequences for orthogonal systems are the subject of Chapter 12 by Kennaway and de Vries. It is refreshing to see this relatively new topic in a textbook. The less interesting, and more problematic, case of Cauchy-converging sequences is not elaborated.

Term-graph rewriting, important for implementations, is briefly discussed by Barendsen in Chapter 13. Some advanced topics, like de Bruijn and van Oostrom's method of decreasing diagrams, are presented in Chapter 14 by Bezem, Klop, and van Oostrom. Chapter 15, by Heering and Klint, is a welcome, though incomplete, list of twenty-seven implementations of varied flavors of rewriting.

Last, but by no means least, the appendix by Bezem contains a masterful review of mathematical prerequisites, including Cantor's ordinal numbers, Kruskal's Tree Theorem, and Ramsey's Theorem. The book closes with a pretty exercise on the evolution of "amœbæ colonies."

In fact, all chapters (save, naturally, the first and last) include exercises, some more generously, and others less so. Keeping up with the times, the editors maintain a web site, `www.cs.vu.nl/~terese`, with solutions to many exercises (but, so far, only for half the chapters).

The editors are also to be commended for what was obviously a gargantuan effort expended in maintaining uniformity of form across the writings of a dozen people. Abundant use is made of a few abbreviations: "ARS" for "abstract reduction system" (already used on p. 7, just prior to its definition); "TRS" for "term rewriting system," "HRS" for higher order, etc. As of January 2005, the web site lists four errata; there are relatively few typos. Cambridge should have provided better copyediting: though always clear, the English is occasionally stilted. The notation is good overall, despite some minor inconsistencies of usage. For example, the letter $\mathscr{R}$ is used for rewriting systems (signature plus rules) in Chapters 2–5, for reductions in 8–9, and for sets of redexes in 12.

The copious references given in all chapters are very helpful, though, here and there, there are lapses. I think I have decoded the non-standard order of the references; I leave that as a challenging exercise to the reader! The name and subject indices are quite comprehensive and useful, but a bit inconsistent in coverage. For example, Newman's Lemma and Toyama's Theorem (about which much ado is made in the introduction) are not indexed. The notation index is harder to utilize, and a few symbols are missing from the list. All in all, my students found the cross-references between chapters very handy, and the examples well-placed and helpful. They were particularly happy with the multiple insights, perspectives, and proof methods afforded throughout.

For an introductory course in rewriting, I would choose Chapter 0, as much as is needed from Appendix A, Chapters 1, 2 and 4, Sections 9.3, 5.1–3, 6.1–2, and 6.4, and Chapter 7. For an in-depth treatment of orthogonal rewriting, one could choose Chapters 1, 2, 4, 8, and 9, with Chapters 10 and 12 as optional extras. Anyone interested in a more exhaustive discourse on equational theorem proving will need to supplement Chapter 7 with additional material. For example, the older book, *Term Rewriting and All That,* by Franz Baader and Tobias Nipkow (Cambridge University Press, 1998), has a more detailed discussion of associative-commutative unification, but also only a brief account of unfailing and associative-commutative completion. For students of logic programming, the book, *Advanced Topics in Term Rewriting*, by Enno Ohlebusch, reviewed in these pages (vol. 4, pp. 539–541), has extensive material on conditional rewriting and termination of logic programs, which could be of interest.

Suffice it to say that this book is indispensable for any serious student of rewriting.

Nachum Dershowitz
Tel Aviv University, Tel Aviv