1 Introduction

Learn Prolog Now! stands out in two respects: the targeted audience and the style of the authors. The book is written for non-experts, with the aim to get them interested, even enthusiastic about programming in Prolog. The authors certainly succeed in transferring their own appreciation of the Prolog language to the reader. And they let the reader play the main role in this book: the reader is encouraged strongly to participate actively in the journey through a sequence of interesting first Prolog experiences. The pace of the book is slow so nobody gets lost during that journey, and with only 265 pages, it cannot visit all interesting Prolog places. In fact, it just scratches the surface of Prolog programming. The book is really self-contained and it feels like a self-study book: both were intended by the authors. It is freely available on-line as well, but this review is based mostly on the first edition of the paperback version. Having an on-line version is very nice, but its usefulness would increase if it were keyword searchable. It is indeed a real challenge to find out from the on-line version where e.g. infix operator is mentioned for the first time. There is also an on-line errata list, but it is certainly not complete or up to date: any discrepancies between the book, the on-line version and that errata list are annoying.

The main body of the book consists of 12 Chapters:

1. Facts, Rules, and Queries
2. Matching and Proof Search
3. Recursion
4. Lists
5. Arithmetic
6. More Lists
7. Definite Clause Grammars
8. More Definite Clause Grammars
9. A Closer Look at Terms
10. Cuts and Negation
11. Database Manipulation and Collecting Solutions
12. Working With Files.

Each chapter has Exercises, and a Practical Session. The first practical session guides the reader gently through her first interactive Prolog session. The second session introduces the debugger. That approach encourages the reader from the very start to use a real Prolog system. Was SWI-Prolog used to produce the output shown in the book? We have to guess, because the authors do not say it.
In the preface the authors thank Jan Wielemaker (of SWI-Prolog) for his encour-
agement that the authors go with ISO-Prolog. As we will see later, the ISO aspect
of the book can be improved. The Introduction might have mentioned something
about the prerequisites to this book: page xii gives the impression that you need not
know any computer programming at all, but it is hard to believe that one can use
this book without some basic skills. Page 19 in Chapter 1 does state such required
skills: nothing extraordinary is needed, but why not to point them out earlier.

2 More details about the chapters

Chapter 1 on Facts, Rules and Queries starts with the typical gentle approach to
introducing Prolog: a set of ground facts with atoms as arguments. At this point,
these facts are queried with ground queries only: queries with variables are postponed
until also ground rules have been shown. A consistent indentation is not always
followed (neither later in the book), and particularly badly indented is the clause
containing an explicit disjunction on page 7: cries for help in compl.lang.prolog show
how important a good indentation practice is.

Page 7 explains the meaning of a predicate with the two clauses

\[
\text{playsAirGuitar(butch) :- happy(butch).}
\]
\[
\text{playsAirGuitar(butch) :- listens2Music(butch).}
\]

as follows: Butch plays air guitar either if he is happy or ... That phrasing strongly
suggests some exclusiveness, i.e., if Butch is both happy and listens to music, he is
not playing air guitar. A better wording could be chosen here.

Page 12 introduces compound terms under the name complex terms: an uncommon
naming, but shared with the GNU-Prolog manual. However, saying that the functor
of \text{playsAirGuitar(jody)} is \text{playsAirGuitar} and that the functor of a complex term
has to be an atom, is unforgivable.

The explanation on consulting a file (and on debugging in a later chapter) show
that the authors mean Prolog to be used from a command line, as from an xterm.
Such an interface to a Prolog system is certainly very popular with older users, but
the new generation might be more attracted and accustomed to clicking their way
through a Prolog session.

Chapter 2 talks about Unification and Proof Search. The explanation of these topics
is quite detailed and complete. As elsewhere in the book, a lot of examples are used.
The occurs check is discussed, also in detail. The practical session makes the reader
use the debugger for the first time: there is no explanation on the debugger trace,
so the reader must figure out herself what the number 7 in \text{Fail: (7) g(a)} refers
to. Even the explanation of \text{Call}, \text{Redo}, \text{Exit} and \text{Fail} is missing. Still, the authors
recommend to use it [the trace] heavily while learning Prolog.

Chapter 3 introduces Recursion. That is a tricky subject in any course book on
programming, especially when the readers have no prior programming experience.
The authors have done a very good job here: they explain recursion with several
examples, illustrating it with pictures, with the search tree and in words. The
same chapter contains a section on rule ordering, goal ordering and termination. Clearly, termination is not an issue in programs without recursion, but rule order and goal order can determine the order of answers in non-recursive programs as well. This issue is worth a section in an earlier chapter (e.g. when the search tree is explained in Chapter 2). This would show that the order is not necessarily termination related. About termination, the book is very vague: page 63 says The change in rule ordering between descend3.pl and descend4.pl merely means that descend4.pl will terminate in some cases where descend3.pl will not. I find that very unsatisfactory. Page 63 speaks about the goal given in the head: that is not the correct terminology.

Chapter 4 shows Lists for the first time. Page 73 says that [ is a special Prolog built-in operator: given the special meaning of operator in the context of Prolog syntax (and the fact that [ is not an operator!), this seems bad terminology. The same goes for referring to [ as a flexible tool (page 75). Moreover, its use for composing a list is not mentioned: it is only presented for list decomposition, while actually, it is syntactic sugar and unification does the decomposition.

The member/2 predicate is used as a case study for using lists. The version on page 77 results in a singleton variable warning in almost every Prolog system. It would be nice to warn the reader for this: page 79 talks about singleton variables, but not about the warning. Moreover, the authors condemn the (very good!) programming style that gives names to all variables (even the ones that could be replaced by an underscore). They do not mention the fact that every occurrence of an underscore represents a different variable: this is quite important and comp.lang.prolog shows that newbies get confused easily on this issue.

Chapter 5 discusses Prolog Arithmetic. It starts by making an analogy between what readers probably are accustomed to, e.g. 6+2=8, and the Prolog analogue, that (a bit to my surprise) seems to be 8 is 6+2. The use of is/2 with an instantiated left argument, is (still) allowed by ISO and was always by tradition. However, it is certainly one of the bad uses of is/2 and it bites the newbie. Two more quibbles: ?- 3 is 6/2. does not succeed in ISO; it fails because / means floating point division. // should have been used instead. And the modulo operator can be written infix. This chapter pretends that arithmetic works on integers only (page 93 and 98 are explicit about this), while floating point numbers are supported by all systems these days, as required by ISO.

When students are introduced to arithmetic in Prolog, usually the following two issues come up: operators (like the + in 3+2), and the difference between 3+2 in unification and arithmetic. The first issue is not arithmetic specific, and it would be better to treat operators earlier, in a more neutral context. In that way, the issues are kept separate: that is good, because they are orthogonal! The second issue is often explained by pointing out the difference between ?- X is 3+2, and ?- X = 3+2 (see page 92). These two queries tell only part of the story. Here are two additional queries that complete it: ?- X is a+2, and ?- X = a+2.

Page 93 reads for Prolog, it (3+2) really is the term +(3,2). The authors apparently ascribe a strong reality to Prolog internals, but this reality does not really exist:
3+2 and +(3,2) are just two syntactic forms of the same terms. The explanation of this phenomenon has no place in the arithmetic chapter; it belongs in an earlier section on operators.

Section 2 describes the use of arithmetic for computing the length of a list: the first program shown is a non-tail recursive version of length/2. That in itself is fine, but why say explicitly (page 95) that this length/2 program is efficient? It is not.

Page 98 says that \( X = b, X < 4 \) fails, but ISO requires it to throw an exception.

Page 99 offers a very good opportunity to introduce the Prolog if-then-else, which is sadly lacking from the book.

Chapter 6 has More on lists, in particular the append/3 and naive reverse programs are shown. The difference between naive reverse and a reverse with accumulator is discussed in detail. The authors give some general advice on accumulators, but the idea behind accumulators is not generalised.

It is not clear why the authors chose to present accRev/3 with the clause order as on page 114: it is good practice to write the base case(s) first, and the recursive clauses later, especially when that does not change the operational semantics (for the intended use of the predicate).

In the practical session, the authors ask the reader to flatten a list by removing all the square brackets around any list it contains as elements ... The terminology is clearly inadequate and inaccurate. The authors name this exercise the pons asinorum\(^1\) of Prolog programming. I have never seen flatten/3 referred to as such, but it is indeed an exercise that many beginners struggle with. Shouldn't the asinu get some feedback on how well they crossed the pons?

Chapters 7 and 8 are devoted to Definite Clause Grammars. Given the background of the authors, and their initial target audience, that seems natural. I do not think that DCGs form a useful topic in any Prolog text book. First of all, DCGs are not Prolog programs, they form a different language. Prolog hosts DCGs, just like Prolog hosts CHR. Next, you learn very little about Prolog while learning DCGs: what you learn is mostly about the pitfalls of DCGs. Of course I am alone holding this opinion: every book on Prolog contains at least one chapter on DCGs.

So we are skipping to Chapter 9: it is about A closer Look at Terms. This time, the operator issue goes under the header of Terms with a special notation. That is misleading: the special notation is related to operator declarations (built-in or not) and is not tied to the term itself. Page 163 contains a funny sentence: Prolog finds the query \( 2 \leftarrow 3 \leftarrow \leftarrow (2,3) \) confusing. No it doesn't: at most it finds a syntax error! Moreover, what is shown is not a query. Queries start with \(?\).

Page 164 refers to a recursive definition: it is better to name it an inductive definition. Page 172 introduces the univ built-in (=./2) in the form \( =./(\text{arg1},\text{arg2}) \). There is no need for the quotes. At last, Section 4 mentions operators with more generality, but quite inappropriately, operator notation is mentioned as being more user-friendly than its [Prolog's] own internal representation. This seems based on

\(^1\) The bridge of fools.
the authors' misunderstanding about what an internal representation is, and indeed Section 4 explicitly says *Internally, Prolog uses the expression is(11,+(-2,*((3,3))))*. As a seasoned Prolog implementor, I swear: this is not true. The same mistake is repeated on page 177 and it even distinguishes between *user-friendly* and *real Prolog* notation. One must be careful with such statements, because soon we might see hackers writing everything in the real Prolog prefix-notation.

Page 180 uses display/1: this is not an ISO predicate.

*Chapter 10* has the title *Cuts and negation*. This would have been another perfect opportunity to introduce the Prolog if-then-else; alas this construct is not mentioned. On the other hand, the reader is lead to believe that the following two definitions of max/3 are the same (page 193):

\[
\begin{align*}
\text{max}(X,Y,Y) & : - X < Y. & \text{max}(X,Y,Y) & : - X =< Y, !. \\
\text{max}(X,Y,X) & : - X > Y. & \text{max}(X,Y,X) & : - X > Y. 
\end{align*}
\]

Just ask the query \(?- \text{max}(1,A,1).\) and observe what happens on backtracking.

The max/3 example is elaborated on for explaining red and green cuts, and even the principle of steadfastness, which is good. It would have been even better to follow that principle in code later in the book. Negation by failure is explained with its pitfalls on an example.

*Chapter 11* is about *Database Manipulation and Collecting Solutions*. In the context of this book, those are advanced topics. Lots of things are not mentioned about the database built-ins or are plainly wrong. For instance: page 204 says that *assert/1 always succeeds*: page 205 says that dynamic predicates must be declared as such; the explanation why \(?- \text{assert( (naive)(X) :- happy(X))}\) has the extra pair of parentheses is missing; page 206 suggests that the goal \text{retract}(\text{happy}(\text{vincent}))\) removes only one fact even if \text{happy}(\text{vincent})\) is in the database twice \(^2\); can \text{retract}/1 also retract clauses? ; page 208 suggests that \text{retractall}/1 only retracts facts; and on the same page, it is suggested that one needs to declare a predicate as dynamic if one intends to change it. However, in the context of page 208, closer to the truth is: you must declare it dynamic if you intend to call it before it has any definitions. This section is really too superficial. On top of that, \text{assert/1} is not an ISO predicate.

One of the uses of dynamic predicates is memoization: this is explained with an example. I made exercise 11.3 (related to memoization) and then compared my solution to the one on page 254 ... the result of the \(?- \text{listing}\) query at the bottom of page 215 must also include \text{signares}(0,0)\) and \text{signares}(1,1). And the code shown in the *Answers to Exercises* should have been structured better, so that it would be clear to the reader that there is a systematic approach to introducing memoization for a (deterministic) predicate.

The explanation of \text{findall}/3 and its friends is always challenging: the authors do a good job in Section 2. Page 212 (using nested \text{findall}/\text{bagof} calls with \text{\sim}) might be a bit over the top for many readers.

\(^2\) The confusion probably comes from the silly habit of Prolog systems to make a ground top level query not exhaust the search space.
Page 210 names the \texttt{fromMartha/1} in the goal \texttt{findall(fromMartha(X),......)} a \textit{predicate}: this is bad terminology to say the least. Some fine-print about \texttt{\^/2} and free variables in a template is avoided. That is a good choice given the objectives of the book, but a warning about unexplained issues would be appropriate. It is impossible to talk about \texttt{setof/3} without talking about an order: the authors limited themselves to the order on atoms, and the order on numbers. Again, just mentioning that Prolog has an order on (almost) all terms would have been nice.

Chapter 12 is about \textit{Working with files}. Up to this point in the book, the idea was to have one program in one file. Here, the authors explain how a larger project with several files can be managed by using consult, commands in a file, \texttt{ensure loaded}, libraries and the basics of modules. An extensive treatment of these topics would be utterly confusing, and the authors have avoided that successfully.

Page 220 uses \texttt{tab/1}: this is not ISO. Page 222 tells you that \texttt{append/3} and \texttt{member/2} are built-ins in SWI-Prolog and not in SICStus Prolog ... but beware! The (current) truth is: in SWI-Prolog, those predicates are in a library that is loaded transparently by default. And in the most recent SICStus Prolog version, they are true built-ins (as a forthcoming revision of ISO Prolog intends them to be).

Page 225 has a program that on backtracking checks for at \texttt{end.of.stream} of a stream that was closed already: this can easily be prevented by introducing a cut, and removing the negation in the second clause of \texttt{read_houses/2}. That would also conform to the advice given by the authors on pages 198–199. Using an \texttt{if-then-else} would have been even better.

Page 225 also says that \texttt{get_code/1} returns \texttt{97} if the character \texttt{a} is next in the input stream. But ISO says it returns \texttt{0'a}. The use of explicit character codes as in the \texttt{checkCharAndReadRest/3} code on page 226 (which is not steadfast either) should be discouraged.

At this point, I became aware of differences between the paperback version and the on-line version: I have not bothered checking the other chapters, but if chapter 12 is symptomatic, the difference is huge. There is even an example in the on-line version (in section 12.2) that is posed as exercise 12.1 in the book ... very confusing.

3 How the book ends

After chapter 12, there is a section with \textit{Answers to Exercises}: an absolute must for a self-study book of course. The on-line version does not contain this section. There is also a short \textit{Further Reading} section: the usual books are pointed at, and unsurprisingly, the early Clocksin & Mellish book is mentioned as \textit{it won't take you far beyond Learn Prolog Now!}. Indeed, these two books are close in spirit and level, but C&M is rather oriented towards students in computer science while the LPN book in the first place addresses (and probably appeals to) a different crowd.

At the end there is a very short list of \textit{Prolog Environments}: listed are SWI-Prolog, SICStus Prolog, YAP Prolog and Ciao Prolog. It is impossible to have an exhaustive list here of course, but one might expect some other freely available systems like
XSB, B-Prolog, ECLiPSe and GNU-Prolog. A pointer to the comp.lang.prolog FAQ would also be useful.

4 Weakness and strength of Learn Prolog Now!

The terminology could be improved in several places and made consistent. I found rather few errors and typos: these can be easily fixed, but it is a tedious job to spot them all. Please always show steadfast code. More ISO compliance would be nice. Some missing elementary topics or constructs could be added without losing touch with the audience, e.g. the Prolog if-then-else or the meta-call.

There is one area in which the authors seem to overstep their competence, or at least get carried away by their enthusiasm: Prolog implementation. At several places in the book, there are disturbing remarks about performance and about internal representation. For instance on page 8 we find: the semicolon is more efficient than using separate rules] as Prolog only has to deal with one rule. That is nonsense. Or on page 29: [Infinite terms ... ] No matter how hard Prolog tries, it can never build one. Still, most Prolog systems these days happily represent infinite terms. Page 33 tells us: when a program is written that makes heavy use of unification, it is likely to be extremely efficient. This is very misleading: naive reverse uses only unification but is extremely inefficient. The same holds for programs using successor arithmetic: lots of unification, very inefficient. The authors might have had a valuable message, but why wrap it in a false slogan? Pages 164 and 170 say weird things about lists and ./.2. Please reconsider carefully all statements related to performance and internal representation.

The book is not written for academics who seek to understand from the start the relation of Prolog with logic (or other paradigms), and if you read it with an academic attitude, the book will probably get on your nerves. My review certainly reflects that.

The level of the book is spot-on for the intended audience: the beginner. This includes the stumped computer science undergrad in need of a really basic introduction because his professor has jumped to the middle of Sterling & Shapiro in lesson 2, the hobbyist who learns new languages just for fun and the scientist from the human science department who wants to find out whether Prolog might be useful for her research. It is not a surprise that the book — or the on-line version — is used by many people, and in particular by people who need or want to learn Prolog without a teacher in flesh and blood. For people with a more profound interest in programming languages, the book has little to offer.

5 Conclusion

Learn Prolog Now! is a great first book on Prolog programming, in particular for people who are easily forgotten by academic writers: people with little knowledge about computers, little background in programming and little interest in logical foundations. It certainly lowers the threshold for getting into Prolog: you just
download SWI-Prolog, work your way to the first practical session on page 16 of Learn Prolog Now! and your first Prolog queries give their answers. This is a fantastic way to learn a piece of Prolog.

Bart Demoen