

Logprimer: A Tutoring System For Prolog Learning

by Enrico Fischetti & Antonio Gisolfi, Dipartimento di Informatica Università di Salerno, Italy

ABSTRACT

"Logic programming" nowadays is avowedly recognized as one of the most powerful software technologies and the language Prolog is, by far, the most popular in this area. Yet learning Prolog at a working level is very often a heavy burden and several students are discouraged by the particular structure of the language, its technicisms, its inference engine, etc. LOGPRIMER (LOGic PRogramming PRIMER) is a learning environment running on a IBM — PC and devoted to teaching interactively the basic notions of Prolog. The mastery of the language is gradually achieved by intermixing theory and exercises and finally LOGPRIMER presents an expert system shell whose knowledge base can be built by the student.

INTRODUCTION

Procedural languages (i.e., Pascal, Fortran, Cobol, Basic, etc.) were for a long time the sole languages available for programming: their influence was so all-encompassing that still today many people think that no other kind of programming can exist. However, non-procedural programming, is by now established area that earns more and more appreciation for both its theoretical and practical implications. In fact, we note that the development of logic programming closely keeps pace with the increasing growth of expert systems and, in particular, of tutoring systems (intelligent or otherwise) (WEN87). Prolog (PROgramming in LOGic) is certainly the most popular language among those developed for logic programming (BRA86, STE86) and it is widely used for writing expert systems, tutoring systems and expert system shells (PY89, SCH88, WIN89, VAL89), yet learning Prolog may cause serious headaches (DUB86).

We think that there are two reasons for this to happen: first, people accustomed to languages such as Pascal or Cobol have to change their minds since logic programming requires an entirely different approach to problems; secondly, Prolog may prove to be a "false friend": in fact, the initial enthusiasm caused by attractive examples (e.g. family ties, animal recognition) may be cooled down when one realizes that Prolog is very powerful but requires deep understanding of its peculiar characteristics in order to exploit its full potential.

In other words, unless running the risk of a complete rejection, a correct approach to learning a programming language is vital to ensure, in reasonable time, a sufficient proficiency and this is specially true as regards a language like Prolog that has as many fans as phobes.

These reasons suggested to us the development of a system for Prolog learning: the system was required to be effective and friendly and had to be truly operational and maintainable. Yet it aimed at being something more than a mere introduction to the language: in fact, LOGPRIMER allows the student an insight into the inference engine of Prolog and furthermore presents the student with a simple expert system shell and asks him to build its own knowledge base.

LOGPRIMER is running on an IBM-PC with 640 Kb and is written in Arity-Prolog. The system is being field-tested in a number of classrooms and the first results of these field trials show a genuine interest in a classroom tool that facilitates the approach to logic programming.

In the following, first, the basic features of Prolog are described, then the structure of LOGPRIMER is discussed in some detail emphasizing the environment "added value" as regards the way Prolog operates and the building of a simple expert system shell.

THE PROLOG LANGUAGE

Since its beginnings in the early '70s, the Prolog language has been used by many people for applications of symbolic computation and nowadays Prolog is quickly gaining popularity throughout the world because of its usefulness in many areas of artificial intelligence.

In a conventional programming language the programmer specifies the algorithm apt to solve a particular problem instead in Prolog the programmer is concerned with the more formal relationships and objects occurring in his problem and he has to specify what relationships are "true" about the desired solution.

Thus, Prolog is to be considered as a descriptive language: the programmer describes known facts and relationships about a problem and it is the duty of Prolog to carry out the computation. The latter is determined mainly by the logical declarative semantics of Prolog and by what new facts Prolog can infer from the given ones. We note that it is right to use Prolog when we have to solve problems involving objects and their logical relationships

A Prolog program consists of a set of "clauses": each clause is either a fact about the given problem (e.g. John is parent of Paul, Paul is parent of Charles) or a rule specifying the inference that can be performed from the given facts (e.g. X is grandparent of Y if the parent of Y is the child of X). Then we can ask questions (e.g. Who is the grandparent of Charles?) and the Prolog will answer the question by means of inferences from one fact to another. Thus, we can say that programming in Prolog consists of "declaring facts", "defining rules" and "asking questions" about objects and their relationships.

ARCHITECTURE OF LOGPRIMER

LOGPRIMER is organized in four lessons, each including several topics; we will now go on to describe the main objectives of each lesson, emphasizing the relevant features of the system. We note that the figures are, in this paper, essential for a fruitful reading: in fact, although only a selection of pictures is present, there is a logical thread linking the figures shown and looking at them gives an insight into both Prolog and the structure of LOGPRIMER.

First lesson

In this lesson the important concepts of predicate and argument are illustrated by means of some examples (Fig. 1). The aim is to approach the peculiar "thread of argument" of Prolog in a natural way so that possible idiosyncrasies can be removed. A Prolog program is shown to be a sort of data base including facts and rules that can be used to answer questions and perform inferences. The correct learning is soon verified by means of suitable exercises (Fig. 2). Obviously, LOGPRIMER is very friendly, the student will never get into trouble and he will never feel "stupid". If the answer is wrong, suitable hints are provided and the student is asked to try again. The concept of rule is similarly first exemplified (Fig. 3) and then the system asks the student to solve some exercises (Fig. 4) and furnishes the "trace" of the solution proposed by the student. In such a way the meaning of "goal satisfaction" is explained and the backtrack logic of Prolog becomes apparent (Fig. 5).

The first lesson, in short, aims at "breaking the ice" so that the student becomes aware Prolog is not for "super-humans" and he or she is able to work with this new software tool, quite far from conventional languages, yet so useful to solve logical problems.

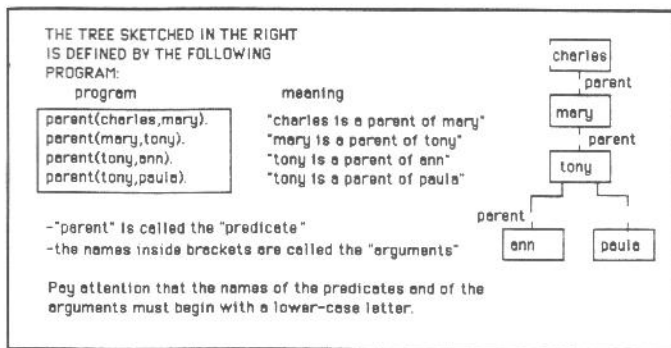


Figure 1.

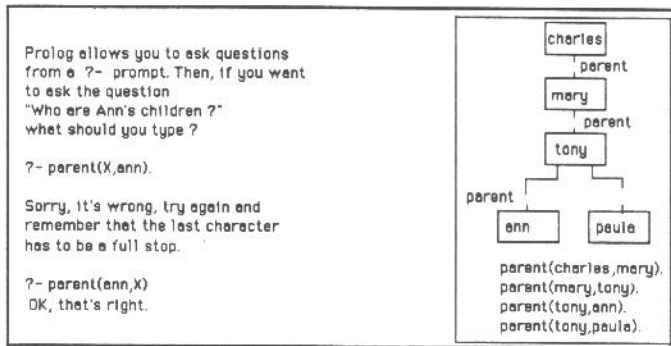


Figure 2.

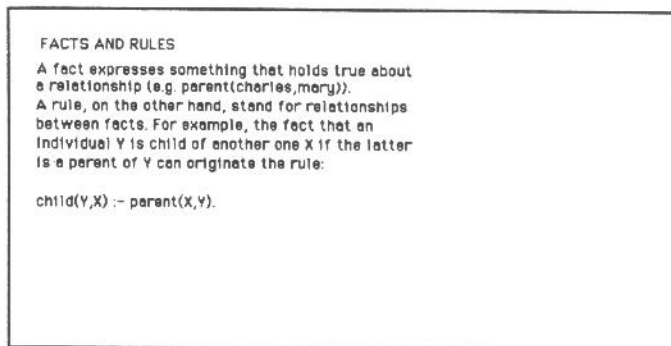


Figure 3.

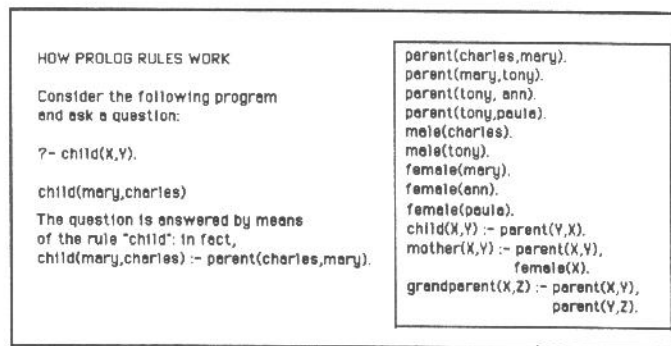


Figure 4.

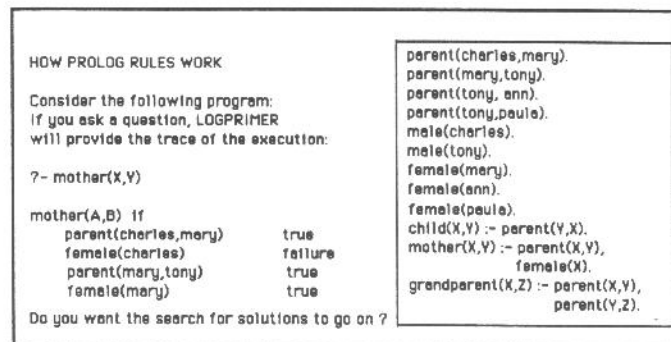


Figure 5.

Second lesson

The task of this lesson is rather exacting: in fact, the self-confidence reached by the student during the first lesson has to be maintained in this second one: the fundamentals of Prolog syntax and semantics are described and it is very important that the unavoidable technicisms are supported by suitable examples so that a high level of interest can be kept. For instance, the notion of "atom" is first defined and then several examples are shown and many questions are asked about it. (Fig. 6).

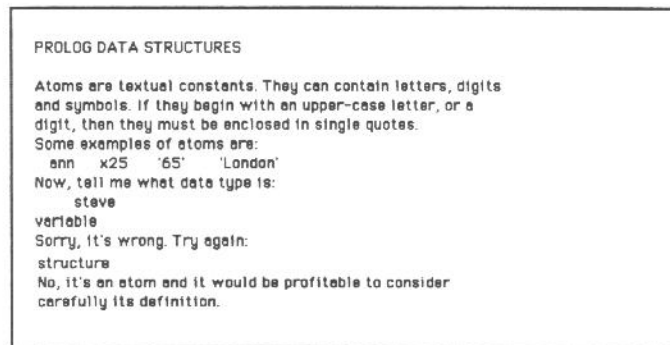


Figure 6.

The following topic discusses the problem of comparing two terms in order to decide whether they are identical or not. Several examples and exercises are presented (Fig. 7), aiming at illustrating the meaning, both descriptive and prescriptive, of a Prolog program. Finally the lesson tackles the problem of endless loops that can happen in spite of the formal correctness of the program (Fig 8).

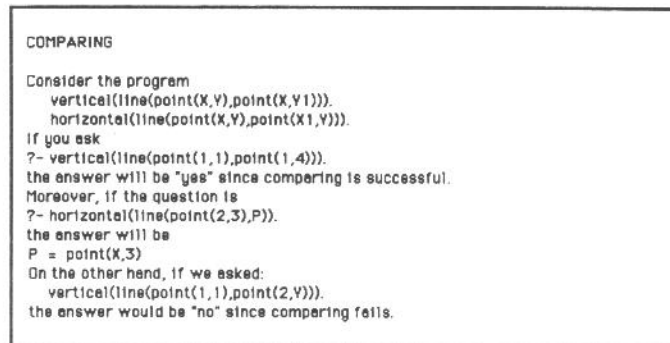


Figure 7.

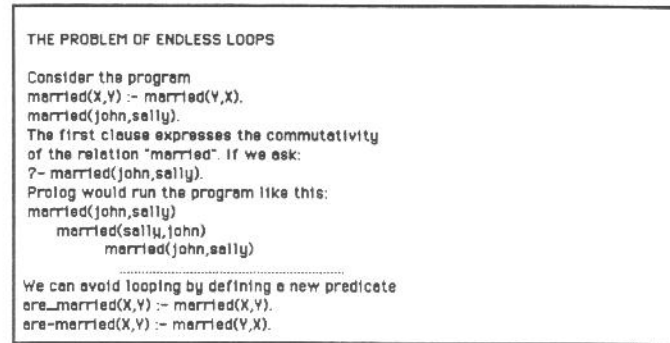


Figure 8.

Third lesson

So far the student has seen some interesting features of Prolog, yet to approach more "real" problems he needs additional information about the language so that he can exploit the full power of Prolog.

The concept of "list" (an ordered sequence of items) is defined and then it is suitably exemplified (Fig. 9).

```
LIST MANAGEMENT

Please, type a list:
?- [lion, tiger, cat].

Good. What is the head ? lion

Very good. And what is the tail ? cat

Sorry, it's wrong. Do you remember how
a list is defined ? (y,n)
```

Figure 9.

Moreover, some examples refer to problems having a recursive nature. For instance, the problem of deciding whether an element belongs to a list can be solved by writing a recursive program (Fig. 10). Subsequently, the student can ask questions about the items of a list and LOGPRIMER provides the trace performed to answer the question (Fig. 11). In the second part of this lesson the relational operators are illustrated and some exercises are suggested (Fig. 12). We note that, when a wrong clause is written by the student, suitable hints are provided by the system and eventually the correct program is achieved. Finally, the control structures offered by Prolog are presented and the main built-in predicates which provide convenient facilities just to save each programmer from having to define them himself are illustrated.

```
LIST MANAGEMENT

The problem of deciding whether an element belongs to a list
can be solved by a recursive program (the vertical bar
separates the head from the tail):
element(X,[X|Tail])
element(X,[Head|Tail]) :- element(X,Tail).

The meaning of the program is:
X belong to the list L if
    X is the head of L
    X belongs to the tail of L

For example, "element(b,[a,b,c])" is true,
"element(b,[a,[b,c]])" is false.
```

Figure 10.

```
LIST MANAGEMENT

Consider the program
element(X,[X|Tail]).
element(X,[Head|Tail]) :- element(X,Tail).
and ask a question:
?- element(2,[3,1,2]).

The trace of the execution is the following:
element(2,[3,1,2]) is obtained by the second clause
element(2,[3,1,2]) :- element(2,[1,2])
element(2,[1,2]), in turn, is obtained by the second clause
element(2,[1,2]) :- element(2,[2])
element(2,[2]) is finally obtained by the first clause.
Thus the answer is affirmative.
```

Figure 11.

```
Consider the predicate max(X,Y,Max) such that Max is the
greatest between the numbers X and Y and type the correct
Prolog clauses (two clauses are enough):
max(X,Y) :- X < Y.
Pay attention: the predicate max needs three arguments.
Try again the first clause:
max(X,Y,Y) :- X < Y.
Good, type now the second clause:
max(X,Y,X) :- X > Y.
Look out! The case X = Y is omitted. Try again:
max(X,Y,X) :- X >= Y.
Very good!
```

Figure 12.

Fourth lesson

This lesson is the most exacting among those present in LOGPRIMER since it tries to explain the fundamentals about the way an expert system shell operates: we observe that the two basic uses of a shell are consultation, that is, having students consult a pre-built knowledge base, and building where the student constructs his own knowledge base.

First, the structure of an expert system is sketched (Fig. 13) and then the operation of a shell is exemplified in relation to some simple knowledge bases (e.g. animal classification, medical diagnosis, engine breakdown, etc).

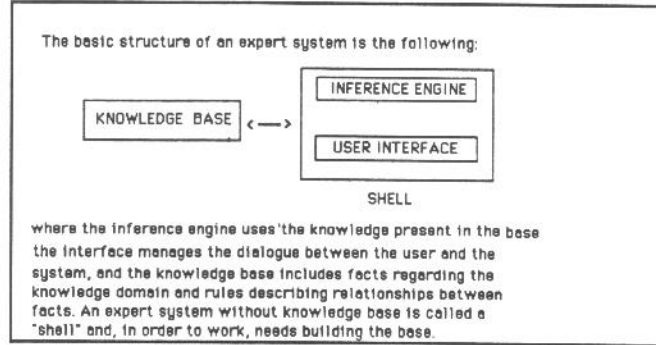


Figure 13.

The student can ask questions about the knowledge base and the shell can operate in two ways:

a) backward chaining: for instance, starting from the assertion "animal(zebra)" the shell goes back (Fig. 14) to verify whether in the knowledge base there are the facts and rules that justify the assertion.

```
BACKWARD CHAINING

To activate the shell press a key:
Ask a question:
animal(zebra)
mammal ? (y,n,why) y
hoof ? (y,n,why) y
black_bands ? (y,n,why) why
animal(zebra) if it has hoofs and has black_bands
black_bands ? (y,n,why) y

animal(zebra) the answer is affirmative
```

Figure 14.

b) forward chaining: for example, we give some initial conditions (Fig. 15) and the shell infers some conclusion by activating the rules of the base.

```
FORWARD CHAINING

The engine uses the initial data and the rules of the knowledge
base in order to attain some conclusions.
To activate the shell press a key at will:
hair
carnivore
yellow_fur
black_bands

The following inferences are carried out:
hair
carnivore
yellow_fur
black_bands
is(mammal)
is(carnivore)
animal(tiger)
```

Figure 15.

At each step the student can ask "why" the shell demands a particular information and then the inference carried out by the system are shown.

Finally, the student is asked to write the rules corresponding to some facts so that at last he can build his own knowledge base (Fig. 16) and start operating the shell.

BUILDING A KNOWLEDGE BASE

Type the name of the new base: mybase

Now, type the clauses grouping those having the same predicate. Remember the steps to be carried out:

- Typing the clauses
- Loading the base in memory
- Examining and editing
- Backward chaining
- Forward chaining

Figure 16.

INTERNAL ARCHITECTURE

We focus now our attention on some of the structural features of LOGPRIMER: this short section can be safely skipped by the reader not interested in technical details.

First, we note that, for the success of the project, it was vital to have the ability to write "meta — programs", i.e., programs whose input data are other Prolog programs. Moreover, Arity Prolog offered us powerful debugging tools and we were able to use, in theoretical parts of the lessons, strings instead of atoms.

When the student is asked to translate some knowledge, e.g. family ties, in corresponding Prolog rules, the analysis of his answer is devoted to some clauses that check accurately its correctness: the predicate "cut" plays an essential role in order to prevent wasteful backtracking.

In order to get the trace of any program execution, suitable meta — interpreters were written: a proof tree is generated and from this the trace can be visualized. Other meta — interpreters were constructed to give the student a range of answers wider than the mere "yes — no". Each query of the student, undergoes the necessary processing by the predicates "structure analysis" and meta — logical verification": this is adequate for simple programs. Furthermore, other tools are used to stop endless loops caused by queries of the student: the loop is stopped and the stack used for overflow is shown.

Finally we examine some characteristics of the expert system shell. In the case of backward chaining it is the duty of an interactive interpreter, beginning from the goal written by the student, to construct a stack of the rules used which represents the trace of the inferences performed by LOGPRIMER. Accessing the stack and using a proof tree, the student can see the steps carried out by the shell. The forward chaining is managed by a different procedure that takes the initial conditions and considers them as arguments of a suitable predicate. Subsequently the non — deterministic choice of the right rules is simulated and then another predicate verifies the correctness of the inference.

CONCLUSIONS

We think that a privileged area for using tutoring systems is the sea of programming languages (FIS89): their unambiguous syntactic and semantic structure is particularly fit for effective computer aided teaching. This is specially true for the languages nowadays used in artificial intelligence since their non — conventional behaviour can be best grasped as shown in an interactive environment provided by the computer. The main objectives behind LOGPRIMER are to produce a tutoring system which will suit users with wide ranging levels of computing skills, which enables safe exploration of the Prolog language in a flexible and friendly way, with a view to helping students learn something about the intimate structure of the language, and, finally, which furnishes a consultation environment of pre-built knowledge bases as well as a building environment to facilitate knowledge representation.

LOGPRIMER has now been implemented and although, at the time of writing, the prototype has yet to be fully evaluated, the first classroom experiences are rather

encouraging and we express the opinion that LOGPRIMER can be regarded as a way in which the potential of Prolog and of expert system shells can be fully exploited as helpful classroom tools.

REFERENCES

1. BRA86: Bratko, I.: Prolog programming for artificial intelligence, Addison — Wesley, 1986
2. DUB86: Du Boulay, B. & Taylor, J: Studying novice programmers: why they may find learning Prolog hard, Internal draft 6 Sussex University, 1986
3. FIS89: Fischetti, E. & Gisolfi, A.: Logoworld: a learning environment for LOGO language, Tech. Rep. DIA89 — 8, Dip. Informatica, University Salerno, 1989
4. PY89: Py, D.: Mentoniez: an intelligent tutoring system in geometry, Proc. 4th Int. Conf. AI & ED, Amsterdam, p.202, 1989
5. SCH88: Scherz, Z., Weinberg, B., Pontch, M. & Goldberg, D.: EESS: Education expert system shells, Pegboard, 3, 1, p.100, 1988
6. STE86: Sterling, L. & Shapiro, E.: The art of Prolog: advanced programming techniques, The MIT Press, 1986
7. VAL89: Valley, K.: Realising the potential of expert system shells in education, Proc. 4th Int. Conf. AI & ED, Amsterdam, p.307, 1989
8. WEN87: Wenger, E.: Artificial intelligence and tutoring systems, Los Altos, Morgan Kaufmann, 1987
9. WIN89: Winkels, R. & Achthoven, W.: Methodology and modularity in ITS design, Proc. 4th Int. Conf. AI & ED, Amsterdam, p.314, 1989