Emulating Prolog in an IBM PC APL Environment

Maria J. Tobar and M. Alfonseca

Introduction

Since PROLOG is a highly used language in Artificial Intelligence applications, it has been our aim to test the APL capabilities to emulate it, in this way combining the properties of both languages in a single system, where APL would provide its number-crunching and file handling facilities, while PROLOG-like APL functions would make it possible to perform logical inferences in a non-procedural environment. All the work took place in a Personal Computer, using IBM PC APL as the basis system.

PROLOG operates with the so-called Horn's clauses formed by a conjunction of affirmative propositions. The answer to a certain goal is obtained by using Robinson's Resolution Principle, i.e. checking that the set formed by the premises in the negation of the goal is inconsistent. It can be proved that this process is complete and correct.

The fact that a generally accepted PROLOG standard does not exist has the consequence that almost each PROLOG system available is unique and incompatible with the others. On the one hand, this is a setback for the current applications of the language, but on the other it means that this language has not yet attained its final form, and finds itself in the process of being designed. This gives us the freedom of introducing changes in the syntax, so as to fit better with our APL environment.

The syntax we have implemented is not exactly the same as PROLOG uses, but it's quite similar and closer to the natural language. It is really an extension of Horn's clauses, since it allows conjunctions and disjunctions of both affirmative and negative propositions, making data entry and processing easier.

To illustrate this we have implemented a toy expert system, capable of answering questions on the genealogy and family relations of a certain number of the gods in the Greek mythology.

Knowledge Base Structure

Knowledge base systems contain two different types of information:

- AXIOMS (or facts) are simple propositions that the system assumes to be true. Their syntax can be defined by the following grammar in Backus Normal Form (BNF):

```
Axiom      ::= Clause
Clause     ::= Assertion|NOT Assertion
Assertion  ::= Term Verb Term
Verb       ::= word
Term       ::= variable|word
```

Variables and words are the terminal symbols of the grammar. Both are literal strings not including spaces. Variables are distinguished from words: variable names begin with an APL delta. In actual fact, variables are not allowed within axioms.

Examples of axioms are:

```
ZEUS IS MALE
GEA IS FEMALE
HYPERION IS-FATHER-OF AURORA
GEA IS-MOTHER-OF IAPETUS
```

- RULES that the system will use to deduce new facts from them. They consist of two parts premises and conclusions, joined by the IF particle. Their syntax is the following (in BNF):

```
Rule        ::= Conclusions IF Premises
Conclusions ::= Clause|
                Clause AND Conclusions|
                Clause OR Conclusions
Premises    ::= Clause|
                Clause AND Premises|
                Clause OR Premises
```

Basically the rules are formal logic implications. ***A=>B*** is equivalent to ***B IF A***,
where ***A*** and ***B*** are assertive or negative propositions containing variables capable of being replaced by one or several words.

Examples of rules are:

```
ΔX IS-SON-OF ΔY IF
   ΔX IS MALE AND
   ΔY IS-PARENT-OF ΔX

ΔX IS-PARENT-OF ΔY  IF
   ΔX IS-FATHER-OF ΔY OR
   ΔX IS-MOTHER-OF ΔY
```

As the examples show, both axioms and rules are accepted by the system in a way very similar to natural language.

Structure of the Knowledge Base

Words are kept in a literal matrix which we call word table. This table will have as many rows as there are words known by the system, and as many columns as the length of the longest word.

A reference to a word is defined by its row index position in the word table.

Axioms and rules are codified into five column numerical matrices.

An axiom is a codified into a 1 row 5 column matrix containing the following information:

The first element is **100** if the proposition is affirmative, **0** if it is negative. The next three elements are the references to the three words making up the axiom. The fifth element is always zero and is included for compatibility

with the rules.

Function *FF* converts an axiom into its internal representation.

```
     FF 'HERMES IS MALE'
100 1 2 3 0

     FF 'NOT ZEUS IS FEMALE'
0 4 2 5 0
```

The set of all axioms is stored as a matrix with 5 columns and as many rows as axioms have been defined.

A rule is internally represented as a five column matrix with as many rows as the number of clauses it contains. The first element in each row indicates whether the corresponding clause is affirmative (100) or negative (0). Words in the clauses are represented by their references to the word table. Variables are converted into negative numbers starting at -10, in decreasing order, and are considered local to the rule. Columns 2, 3 and 4 in the matrix contain the information about the three terms in each clause. Finally, the fifth column indicates the connective between each clause and the next, according to the following translation rules:

- IF becomes -1.
- AND becomes -2.
- OR becomes -3.
- A zero indicates the end of the rule.

Function *FF* will also convert a rule into its internal form.

```
FF 'ΔX IS-SISTER-OF ΔY IF
    ΔX IS FEMALE AND
    ΔX SAME-PARENTS ΔY AND
    NOT ΔX IS-SAME-AS ΔY'
```

will become the matrix:

```
100 -11 6 -12 -1
100 -11 2   5 -2
100 -11 7 -12 -2
  0 -11 8 -12  0
```

Running the System

Once the data have been introduced, the system is ready to answer questions about them by means of the inference method it contains.

The answers the system gives may be different depending on what we have requested:

- If we ask a question of a fact, the answer will be affirmative or negative:

```
     ASK 'ZEUS IS MALE'
YES
     ASK 'HELIOS IS-SON-OF OURANOS'
NO
```

- If the question contains a variable, the answer will consist of all the words with which the variable may be unified. If there is no answer, the result will be NO.

Examples:

```
      ASK 'ΔX IS-FATHER-OF AURORA'
HYPERION
      ASK 'ΔX IS-SON-OF GEA'
IAPETUS HYPERION CEO CRONOS OCEANO
      ASK 'GEA IS-MOTHER-OF ΔX'
IAPETUS HYPERION CEO CRONOS
        RHEA OCEANO TETHIS
```

Questions including two different variables are not allowed at this time.

The answer may be obtained directly (if the request is an axiom, or may be unified with one), or by inference, if it is deducible from the rules. Applicable rules will be tried successively until one of them provides one or several answers. All remaining rules are then ignored. The process is recursive, i.e. trying to apply one rule produces subgoals that may require the inference process to be applied again.

Explaining the Answers

While the inference procedure is taking place, the system builds a global numeric matrix containing information about all the steps performed to obtain the solution. This explanation may be offered to the questioner if required.

Function WHY takes this global variable generated by the last question, and produces an explanation in natural-like language to a depth chosen by the questioner. The information appears indented according to the current depth and, in this way, the level may be known.

```
  ASK 'ZEUS IS MALE'
YES
  WHY
BECAUSE IT'S AN AXIOM

  ASK 'GEA IS ΔX'
FEMALE
  WHY
BECAUSE IT'S AN AXIOM

  ASK 'HYPERION IS-SON-OF OURANOS'
YES
  WHY
HYPERION IS MALE AND
OURANOS IS-PARENT-OF HYPERION
GAVE ME THE ANSWER:
YES
DO YOU WANT MORE INFORMATION?  (YES/NO)
YES
  HYPERION IS MALE
  GAVE ME THE ANSWER:
  YES
  AND BESIDES
  OURANOS IS-PARENT-OF HYPERION
  GAVE ME THE ANSWER:
  YES
  DO YOU WANT MORE INFORMATION?
  YES
```

```
   OURANOS IS-MOTHER-OF HYPERION
   OR OURANOS IS-FATHER-OF HYPERION
   GAVE ME THE ANSWER:
   YES
   DO YOU WANT MORE INFORMATION?
   YES
OURANOS IS-MOTHER-OF HYPERION
GAVE ME THE ANSWER:
NO
AND BESIDES
OURANOS IS-FATHER-OF HYPERION
GAVE ME THE ANSWER
YES
```

Conclusion

As described before, the system is now complete and performs logic inferences according to expectations. The syntax is more compact than PROLOG's, and the number of rules and axioms for a given system is somewhat smaller. In the case of our example, a total of 24 rules and 174 axioms contain all the information. Quite complicated questions on genealogy and relationship of the Greek gods may be requested.

At the point this system has been developed, it does not include concerns as the probability of application of the rule. This is, obviously, an open field of implementation. The speed, as compared to non-emulated PROLOG interpreters, is, of course, slower by about one order of magnitude. Different techniques could be applied to make it faster, and will be the object of a further communication.

References

[1] Clocksin, W.F. and Mellish, C.S., *Programming in Prolog.* Springer-Verlag, Berlin, Heidelberg, New York (1981).

[2] Bittlestone, Robert, *XPL: An expert system framework in APL.* Vector. 1 no. 2 (October 1984), 65-72.

[3] Clark, K.L., Ennals, J.R. and McCabe, F.G., *A micro-PROLOG primer.* Logic Programming Associates Ltd., London (1982).

[4] McCabe, F.G. and Clark, K.L., *Simple PROLOG and MICRO extensions.* Logic Programming Associates Ltd., London (1983).

IBM Madrid Scientific Center
Paseo de la Castellana, 4
28046 Madrid, Spain

Trees and Networks Using APL2

Z.V. Jizba

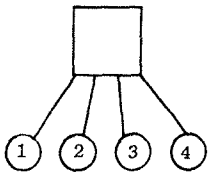
Nested arrays in APL together with other enhancements make it possible to build LISP-like (and PROLOG-like) applications. This paper describes some of the techniques for dealing with structures such as TREES and NETWORKS. Examples illustrate possible applications in the field of artificial intelligence.

TREES

A vector can be thought of as a rooted tree (for definitions of these terms, see a textbook on graph theory such as [1]). For example, the vector of integers could be represented by a tree diagram:

VECTOR: 1 2 3 4

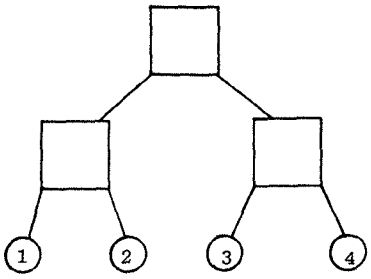
TREE:



The circles and square in the above diagram are called nodes. The lines connecting pairs of nodes are called links. The node shown as a square is called the root node. It is implied by the vector, and therefore cannot contain data. A nested vector of integers can be also represented by a tree graph. For each nested element of the vector there is an implied node as shown on the following diagram:

VECTOR: (1 2) (3 4)

TREE:



In the diagrams above, a distinction is made between terminal nodes (called leaves), and nodes of degree greater than one. The isomorphism between nested vectors and tree graphs is inadequate for many applications. There are two problems: First, implicit nodes cannot contain useful information. Second, there is no way to define a leaf node other than as a simple scalar. Consider the following diagram: