EXPRS
A Prototype Expert System Using Prolog for Data Fusion

Vincent J. Pecora, Jr.
Lockheed 52-56/205
Palo Alto Research Laboratory
3251 Hanover St.
Palo Alto, CA 94304

Abstract

During the past year, a prototype expert system foı tactical data fusion has been under development This computeı progıam combines vaıious messages concerning electronic intelligence (ELINT) to aid in decision making conceıning enemy actions and intentions The prototype system is written in Prolog, a language that has proved to be veıy powerful and easy to use for problem/rule development The resulting prototype system (called EXPRS - EXpert PRolog System) uses English-like ıule constructs of Pıolog code This approach enables the system to generate answers automatically to "why" a rule fiıed, and "how" that rule fired In addition, a rule clause construct is provided which allows direct access to Prolog code routines This paper descıibes the structure of the rules used and provides typical useı interactions

IN THE MODERN MILITARY ENVIRONMENT, fusion of intelligence data from different sources is becoming increasingly important. Multiple sensor inputs need to be interpreted in a timely manner to assess developing battlefield conditions. The high volume of data from such sensor systems, as well as their high rate of data transfer, make this timely interpretation difficult and very demanding of human resources. A system that could process routine messages automatically would free the human intelligence analyst to concentrate on more difficult data interpretation tasks

As part of an effort to implement a computer expert system to aid in tactical data fusion, a project was initiated at the Palo Alto Research Laboratory to study potential languages and architecture that could contribute to the development of this system. This effort, which is still in its initial stages, led to the development of a small prototype expert system program written in PROLOG Most existing expert systems have been written in LISP The language PROLOG is described in Clocksin and Mellish (1981), and its use for writing expert systems is described in Clark and McCabe (1980) and in Mizoguchi (1983). The prototype system, called EXPRS (EXpert PRolog System), has a number of interesting and powerful features. This system was implemented using the DEC-10 version of PROLOG (Pereira et al 1979), and is being translated to the University of Sussex POPLOG version of PROLOG on the VAX 11/780 (Hardy 1983). This POPLOG implementation will allow calls to external routines written in other languages, such as FORTRAN, which can be used for tasks such as "number crunching."

I wish to thank Herbert E Rauch and Oscar Firschein of Lockheed Palo Alto Research Laboratory who helped in the formulation of this problem

Problem Characteristics

Automating the process of tactical data fusion is inherently a difficult problem. The system must integrate a large amount of input data from multiple sources, including sensor data as well as direct observation data. In addition, input messages can be received out of time order, or information can contradict earlier information. Thus the process is inherently probabilistic as well as time varying and nonmonotonic.

The fusion process can also require numerical analysis to be done on the raw sensor data. This "number crunching" analysis is best done (and is currently being done) with languages such as FORTRAN. For this reason, it is desirable for the final system to make use of existing FORTRAN analysis software. A more detailed description of the tactical data fusion problem is given in Rauch *et al.* (1982), which also references other work on expert systems relevant to tactical data fusion.

From the beginning of this project, it was recognized that the prototype system was a first step, and most of the difficult aspects of tactical data fusion would be solved later during construction of the operational system. The prototype effort therefore centered on providing a flexible architecture for that future effort.

Knowledge Representation Scheme

EXPRS uses a general Attribute-Object-Value knowledge representation scheme coupled with an English-like rule format. This form of representation is very general, offering good future growth potential for the system. The English-like rule format makes it easy to add complex rules and simplifies the task of automatically generating answers to "why" a rule was fired, and "how" that rule came to be satisfied.

The basic format of a rule is as follows:

```
rule123:

if
   (attribute1) of-the (object1) has-value (value1) and
   (attribute2) of-the (object2) has-value (value2) and
   .


then
   create-new (attribute) of-the (object)
   with-value (value) and
   print- (print-list)
```

The `(attribute)`, `(object)`, and `(value)` fields in the rule clause are replaced with either a token name or a variable. Variables are represented using the notation `v(variable name)`. A sample clause might be:

```
emitter-members of-the battery1 has-value v(e-member1)
```

This form is equivalent to the PROLOG term

```
emitter-members(battery1,E-member1)
```

where `E-member1` is a variable. This dual representation has the advantage that internal bookkeeping is easily done in the translation step. For example, the `create-new` clause in the `then` part of the rule basically corresponds to the PROLOG term:

```
asserta(attribute(object,value))
```

However, in making this translation, the system automatically keeps a list of valid attributes for each new object that is created. This allows the "describe" mechanism to provide a description of any object as detailed below. This translation step between the English-like clause format and the actual PROLOG representation will make it relatively simple to add a more complete inheritance mechanism to the `create-new` clause in the future.

The `print-` clause in the `then` part of the rule above is a convenient way of printing a list of any combination of PROLOG goals, strings, and rule variables. Any PROLOG goal may be performed in this list by using the format

```
do(goal-name)
```

as a list element A `print-` clause typically looks something like this:

```
print- [do(nl), v(battery1), ' is associated with ',
v(communication-node2), ' (rule24)', do(nl)]
```

In the above example the `nl` is the built-in PROLOG goal that generates a new print line; however any goal could be used with the `do` construct.

PROLOG code can also be accessed using a rule clause form. This provides an additional degree of flexibility to the system architecture. The rule clause format for this function is

```
the-relation (relation-name) is-true-for
(list-of-arguments)
```

An example might be

```
the-relation append is-true-for
  [v(old-battery-members),
   v(current-emitter),
   v(new-battery-members)]
```

This would translate into the PROLOG clause:

```
append(Old-battery-members,
       Current-emitter,
       New-battery-members)
```

Automatic Description of Objects

Whenever an object receives a new attribute as a result of a rule running, the system automatically adds that new attribute to the list of valid attributes for that object. This is done as part of the definition of the clause form `create-new` (attribute) `of-the` (object) `with-value (value)`. Then, when the user enters

```
describe battery1
```

EXPRS looks through the list of valid attributes for battery1, and for each attribute that it finds, a PROLOG term of the form

```
attribute(battery1,Value)
```

is generated and matched. If this particular attribute is a result of a rule's being run (instead of an attribute that is in place because of an input message) then a term of the form

```
how-true(attribute(battery1,Value),Rule-number)
```

will also be in the data base. This term is put into the data base every time a new attribute is added by the "create-new ..." rule clause. The "describe" mechanism uses this term to pick up the rule number that was responsible for the attribute-object-value term. The "describe" function thus also provides pointers to the rules that are responsible for each value of each attribute for any object. A simple output would look like:

```
describe emitter5.                            (user command)
emitter5 description:
belongs-to-battery:            battery5   (rule12)
detection-time:                13:52
emitter-designation:           landroll
radio-communication-node:      node3      (rule21)
```

This scheme is very general; it will be replaced with a more efficient inheritance network in future systems.

Automatic Generation of "Why" and "How"

One of the objectives of the EXPRS system was the automatic generation of answers to questions from the user as to "why" a particular message or result was obtained (that is, which rule was responsible for the result) and "how" the conditions for a particular rule were satisfied. The use of an English-like rule format simplifies this task. As described above, whenever any rule fires, the appropriate `how-true` PROLOG term is generated. In addition, most rules, as a part of their firing, print out a message that includes the rule number that was responsible for the message. For example Rule 12 might state that, if the location of the emitter in the current message is within 10 kilometers of the location of an emitter in a previous message, then the current emitter should be assigned to the same battery as the previous emitter. Then, when a user enters:

```
why rule12.
```

the system responds with the text of rule 12 which could look like:

```
rule12:
if
  designation of-the current-emitter has-value
    C-emitter *(1)* and
  position of-the C-emitter has-value
    Position1 *(2)* and
  position of-the Previous-emitter has-value
    Position2 *(3)* and
  the-relation distance is-true-for
    [Position1,Position2,Distance] *(4)* and
  Distance < 10 0 *(5)* and
  belongs-to-battery of-the Previous-emitter
    has-value Battery1 *(6)*
then
  create-new belongs-to-battery of-the C-emitter
    with-value Battery1 *(7)* and
  print- [do(nl), C-emitter, belongs-to-battery,
  Battery1, (12), do(nl)] *(8)*
```

Note that the variable notation of `v(variable-name)` has been converted to the standard PROLOG variable notation of starting variables with a capital letter. The first rule clause thus corresponds to a rule text of the form:

```
designation of-the current-emitter
    has-value v(c-emitter)
```

and which corresponds to the PROLOG term:

```
designation(current-emitter,c-emitter),
```

The "why" mechanism thus provides the user with the rule text that is responsible for the output of the system. Note that each clause in the rule is numbered when that rule is printed. These numbers are used to reference each clause separately for the "how" mechanism as described below.

In order to understand how the conditions of a particular rule were satisfied, the "how" mechanism is used to show the user the actual instantiations of the rule variables when the rule fired. EXPRS attempts to model the data driven environment of a tactical data fusion system, and therefore can process more than one input message before interacting with the user. Each time a single input message is processed by the system, the rules are run in a predetermined order to process that message. If any of the rules fire for an input message, then the entire sequence of rules is retried, until no more rules fire, and the system is in a stable state. Then, if there are one or more messages in the input queue waiting to be processed, the rule running sequence is repeated for each of these messages one at a time Thus, any given rule can run any number of times as it processes different input messages. The "how" mechanism which shows the user the instantiations of the variables in a rule, needs to know which instance of the rule's running that the user wants to know about.

EXPRS has an automatic bookkeeping system which allows it to handle this situation in a simple way for the user Whenever a rule runs and creates or updates an attribute for an object, a PROLOG term is formed that links the rule name, a counter, and a list of objects together (one object per attribute changed by the rule). The counter referred to is simply incremented every time a rule is run, so as to make each rule firing a unique event. For example, if the sample rule (rule12) referred to above fired, the PROLOG rule instantiation term that would be created might look like:

```
rule-instan-objects(rule12,132,[emitter5]).
```

If Rule 12 later fired again for emitter6, another term of this form would also be created, *i.e.*

```
rule-instan-objects(rule12,153,[emitter6])
```

When the "why" procedure is run, a PROLOG term is asserted that keeps track of the current rule that is being examined Subsequently when the "how" procedure is next run for one of the numbered clauses printed by the "why" procedure, the current rule name is used to examine the relevant `rule-instan-objects` terms. If there is only one `rule-instan-objects` term for the rule that is being examined, then the value of the instantiation counter to use for variable value retrieval is obvious. If there is more than one rule-instan-objects term, then the user is asked the question:

```
For which (object class)?
```

In the above example this would be:

```
For which emitter?
```

This question is repeated for each object in the object list of the first of the `rule-instan-objects` terms that correspond to the rule that is being examined. Note that any of the `rule-instan-objects` terms would do, as the object classes that are asserted by a rule are constant, but the values of the objects are unique. As keeping track of the valid answers could get confusing to the user, EXPRS can provide a list of all of the valid responses to this question. For example:

(user responses in capitals)

```
For which emitter?
  HELP
potential emitter
  names are:
                       [emitter5]
                       [emitter6]
For which emitter?
  EMITTER5
```

Once the proper instantiation counter value is determined, no further questions are asked for subsequent user "how" requests, until another "why" request is processed, which resets the value of the current rule. An example of using "how" on Rule 12 above might look like:

```
HOW 7                (the user asks how clause 7 ran)
For which emitter?   (EXPRS attempts to determine which
HELP.                instance of the rule running the
                     user wants to know about)

potential emitter
  names are:         [emitter5]
                     [emitter6]

For which emitter?   EMITTER5

create-new belongs-to-battery of-the C-emitter:emitter5
with-value Battery1:battery3 *(7)*

HOW 5                (the user also wants to know
                     the exact value of the
                     distance involved)
Distance:            (EXPRS now knows to which
  7 5 < 10 0 *(5)*   instance of the rule
                     running the user is referring)
```

Conclusion

The EXpert PRolog System [EXPRS] described in this article uses the ability of PROLOG to declare arbitrary word strings as operators to implement an English-like rule format. This makes the generation of rules more natural, and has the added advantage that the rule text itself can be used to generate text for the "why" and "how" mechanisms automatically This first implementation of EXPRS is data driven by messages that are input to the system, each message input triggering a running of the rules in a predetermined order. The implementation of EXPRS thus demonstrates the ability of PROLOG to build control structures that are not backward chaining in nature. EXPRS also provides a rule clause format that allows access to arbitrary Prolog procedures. Because the DEC-10 PROLOG implementation does not contain the capability to access other executable modules, such as FORTRAN subroutines, EXPRS is being converted to the POPLOG system containing a version of PROLOG which provides this feature

References

Clark, K L and McCabe, F.G (1980) PROLOG; A language for implementing expert systems In J E Hayes & D Michie (Eds), *Machine Intelligence*, Volume 10. New York: Halsted Press

Clocksin, W.F. and Mellish, C S (1981) *Programming in Prolog* New York: Springer-Verlag.

Hardy, S (1983) A new software environment for list-processing and Prolog programming In M Eisenstadt & O'Shea (Eds), *Artificial Intelligence: Tools, Techniques, and Applications* New York: Harper & Row

Mizoguchi, F (1983) PROLOG based expert system *New Generation Computing*, Volume 1, Number 1, 99-104

Pereira, L M., Pereira, F., and Warren, D. H D (1979) User's guide to DECsystem-10 PROLOG Occasional Paper 15, Department of Artificial Intelligence, University of Edinburgh

Rauch, H E , Firschein, O , Perkins, W A , and Pecora, V J (1982) An expert system for tactical data fusion *Proceedings of the Sixteenth Asilomar Conference on Circuits, Systems & Computers*, 235-238

APEX

Now in our second year supplying knowledge-based expert systems to the financial services community.

Dr. Fred L. Luconi
President

Dr. Norton R. Greenfeld
Vice President—Software Engineering

Richard I. Karash
Vice President—Product Development

Kenneth P. Morse
Vice President—Market Development

Dr. William A. Woods
Chief Scientist

Applied Expert Systems, Inc.
Five Cambridge Center
Cambridge MA 02142
(617) 492-7322