PROLOG AND EXPERT THREAT SIMULATION

John E. O'Reilly, Jr.
Reflectone, Inc.
5125 Tampa West Blvd.
Tampa, FL 33634-2408

ABSTRACT

Tactical missions are very demanding scenarios for both pilots and instructors. The pilot expects a flight simulator to provide a realistic simulation while the instructor requires an efficient teaching environment. Artificial intelligence (AI) provides both of these expectations: AI, in the form of an expert system, produces realism through fuzzy logic and it delivers a scenario which is easily generated and modified. The realism goal is achieved by the expert system which uses a set of protocols or rules which define the operation of threats within the tactical mission. These rules are generally English-like statements which the instructor can manipulate to form the necessary mission. This paper describes an IR&D project which combines the use of the Prolog programming language with an intelligent threat reaction system. Prolog is ideal for this project because the language furnishes a rapid development environment, and the expert system, written in Prolog, provides for easy modifications to the scenario. The paper also discusses expert systems in general and fuzzy logic processes specifically as they pertain to the main goal and design of the project.

INTRODUCTION

Instructional personnel have a heavy workload during tactical mission training. Traditionally, instructors taught and scored "over the shoulder," keeping most of their attention on the student. High quality training resulted. With the advent of sophisticated Electronic Warfare environments, measures, and counter-measures, the instructor was expected to keep one eye on the threat environment, while still instructing over the shoulder. A system that relieves instructor workload by automating the environment and one which still provides high realism would retain the high quality training seen in the past.

The key to attaining this automated environment is to use artificial intelligence techniques, especially rule-based expert systems. The rules in the expert system allow for realistic threat responses since all known enemy threat protocols can be easily entered into the system and can be easily modified. This is because rules are made up of English facts; facts such as "IF THE INTRUDER IS WITHIN RANGE OF SAM SITE, THEN FIRE A SAM." This is a very simplistic rule, one which needs to be qualified by other rules, such as "IF SAM SITE IS READY, THEN SAM MAY BE FIRED" or "IF ALTITUDE OF INTRUDER IS LESS THAN 2000 FEET, THEN DO NOT FIRE SAM." Note that the rules are with respect to the defensive commander's viewpoint. After all, the expert system is controlling the threats. Also, the rules can be easily modified since they are English statements and they may include some uncertainty (or fuzziness). For example, a realistic defender may not know the actual range of the intruder from the SAM site; he may only know approximate values. This adds a confusion factor on the part of the threat - a more realistic environment.

Reflectone has performed an IR&D project with this goal in mind - To automate the execution and control of the realistic tactical threat environment. The project involved defining a simplified, but realistic and expandable scenario which could be operated not only automatically, but intelligently. The flight simulation and instructor control can follow traditional concepts, with the singular addition of a computer-controlled expert defensive commander controlling the threat environment. The project uses the Prolog language due to its reputation of being a compact prototyping language as well as providing an efficient language to process the textual rules. Prolog supplies realism through its capacity for programming the natural expandability of rule-based systems and the incorporation of fuzzy logic techniques. Thus, the key goal of the project (to provide realism through the use of a rule-based expert system) is achievable through the use of Prolog.

PROBLEM STATEMENT

Having selected the programming language, the project continued by defining a war game. The environment to be simulated is a scaled down Electronic Warfare (EW) scenario consisting of six simultaneously active platforms threatening a single intruding aircraft. The intruder approaches the target (a runway) from the North in a 100 (down-range) by 100 (cross-range) nautical mile gaming area. The platforms consist of three Surface to Air Missile (SAM) and three Anti- Aircraft Artillery (AAA) weapons systems. The intruder has a full complement of Electronic Support Measures (ESM) and Electronic Counter Measures (ECM), while the defensive commander has surveillance and tracking radars at each platform site. The defensive commander is implemented as a rule-based expert system while the intruder is simulated by a six degree of freedom (6DOF) flight simulation algorithm with full terminal control and display.

Realism is necessary to varying levels of detail. The flight simulation runs at a real time iteration rate of ten Hertz (ten updates per second), but is required to be only of medium fidelity because the flight is secondary to the main project goal. The defensive threat environment (the expert) requires high fidelity so that a realistic game can be played.

In order to approach the necessary realism, the rule-based defensive commander must have uncertainty in its responses. Uncertainty (or "fuzziness") provides a physically and psychologically correct threat system. If degeneration of the command and control network information that is provided to the defensive player should occur, then the defense should respond (or incorrectly respond) accordingly. Degeneration can occur through various means, such as degraded weather conditions or equipment malfunctions. Additionally, the defensive expert should correctly respond to the intruder's ECM attempts, thus providing enhanced realism. Further realistic effects are provided through the use of accurate probability of kill algorithms. Kill probabilities could have been implemented using an expert, but have been hard coded under this project.

The flight simulation is executed in a host minicomputer, with an external processor supplying the expert defense command and control. This separation of computational functions provides a secure processing environment as well as a functionally independent system. All secure data are held in the external processor, allowing the host to execute in a clear mode.

EW INTERFACE

Using an external processor for the expert defense implementation injects requirements for both hardware and software interface controls with the host flight simulation. The external processor requires the current flight status from the host. This information is made up of that data which is normally "seen" by defensive radars and communications systems. In general, this means that aircraft positions and attitudes need to be passed to the expert defender along with the statuses of ECM (on/off control, timing, and level of ECM injection). Also, the instructor may elect to move, remove, or add platforms during the game. The statuses, conditions, and positions of these controls must be included as part of the interface. The external processor also needs the status of the simulation (e.g., freeze flags) and current simulation timing data for synchronization purposes.

The aircraft's responding ESM needs data from the expert defender in return. The external processor passes signal strength and bearing data for active platform radars as well as positions of active missiles (for pilot display of infrared information). Although not a necessity, missile flyouts are computed and controlled in the external processor to maintain functional independence.

All interfaces are passed between the host and external processors by way of a standard serial link. This project has shown that a serial path running at 9600 bits per second is adequate to provide control of up to 100 platforms. The large number of platforms can easily be handled at this rate because not much changing activity occurs between transmissions of interface data. Only the data that does change on any given pass (or query) through the expert system need be transmitted; the host assumes that platforms with no new transmitted information retain their past statuses.

KNOWLEDGE BASE

An expert system requires a data base (called the rule base) made up of conditions which imply conclusions. Rules can thus be thought of as IF/THEN statements - IF a set of conditions are true, THEN the conclusion is true. For example, in an EW environment, a rule to tell a platform to come to alert could be:

```
RULE 1:
     IF   the aggressor aircraft is in
          range of the early warning radar

     AND  the platform is in standby mode

     THEN set the command of the platform
          to alert mode.
```

Expert systems can operate on the rule base in one of two modes: forward or backward chaining. In forward chaining, the expert has no indication of what conclusions are available. It simply checks the conditions for truthfulness, and provides the user with all conclusions. In backward chaining, the expert assumes a conclusion (sometimes called a goal), and checks the truthfulness of conditions which lead to this goal. For the rule above, forward chaining would entail checking the conditions - Is the aggressor aircraft in range of the early warning radar? If so, is the platform in question currently in standby mode? If so, then tell the platform to go to alert mode. Conversely, in backward chaining, the expert would assume that the platform should be commanded to alert mode. The expert must then determine if the conditions are true.

Using this background and the fact

that Prolog (the language of choice for this project) implements an efficient backward chaining technique, we chose to use backward chaining. Thus, the expert system queries each rule, attempting to prove it (or "fire" it) by checking whether all of the conditions are true. If any condition is false, then there is no need to continue looking at further conditions. (This introduces the concept of intelligent rule base ordering, which is discussed later.) The expert checks the truthfulness of conditions (asserts them) by querying the simulation's information. For example, the example rule above has a condition requiring the aggressor to be within range of an early warning radar. The expert knows the location (Defined by the scenario as X-Y coordinates within the gaming area) of the radar and it obtains the aggressor location from the host simulation. A root sum of squares of axis deltas is performed on these locations and compared with the radar's maximum range. If within the maximum range, the condition is asserted; if not, the condition is not asserted.

This short description of the rule base (or knowledge base) gives only a cursory look at defensive platform protocol. The implemented rule base is not much more complex than this. In fact, all six platforms of the problem contain a rule of the form of RULE 1 as well as one other rule. The other rule defines conditions which imply the conclusion to launch or fire the platform's weaponry.

Additionally, the rule base includes a "fuzzy factor" for each condition. This factor is a number between zero and one, and indicates the probability that the condition is true. Fuzzy logic is described further in the next section. The rule base is built up in a single text file and is accessed through Prolog's "consult" predicate.

EXPERT SYSTEM SHELL

The problem definition requires the use of an expert system, however, the utility of a rule-based paradigm extends beyond the requirements. The project's main goal is to decrease the instructor's workload while providing a realistic environment. An automatic system provides this flexibility with the added advantage of very easy and straightforward modification capabilities. If a scenario doesn't operate correctly, then the rule base contains insufficient or incorrect rules. By simply modifying or extending the text oriented rule base, the instructor can efficiently tailor the expert. For example, if the instructor notes that an AAA system is firing (i.e., the command of the platform is fire as opposed to its normal alert command) when the intruder is above the range of the AAA's guns (1000 feet), then the instructor would simply add a rule:

```
RULE 2:
     IF   platform is type "AAA"

    AND   platform is commanded to fire

    AND   altitude of aggressor is greater
          than 1000 feet

   THEN   set command of platform to
          alert.
```

The roundabout definition of this rule is due to Prolog's rule base search techniques. Prolog always searches rules from the top down. The first rule in the rule base is the first to be queried. If a conclusion to fire the AAA platform has been made, then RULE 2 fires if the intruder is within altitude range. Otherwise, the platform's fire command is overridden (command is set back to alert). This rule is placed _after_ any other rules which command the platform to fire. This discussion brings up another aspect of intelligent rule base ordering which is described in the next section.

Expert Shell Structure

Figure 1 shows a flow diagram for a generic expert system shell. This shell is perfectly usable for this project and was implemented with few changes. The shell loops on all of the rules and on all of the conditions within each rule. Conditions are checked by either querying the operator for a true or false reply, or by using the surrounding application's knowledge (Ranges are computed and checked against minima, etc.). If a condition passes (is found to be true), its text string is placed in a temporary list, YES. If the condition fails, its string is put in another temporary list, NO. These temporary lists can be checked prior to any conditional checking, thus speeding up the overall query response throughput. Conditions which have already been checked appear in one or the other of the lists and need not be re-computed. If any condition fails, the inner (condition) loop is exited. If all conditions pass, then the rule fires, that is, it is executed (via a call to a "FIRE" routine), and the conclusion text is placed in the YES list (Conclusions can also be conditions. See RULE 2.). The next rule is then processed until all rules have been checked at which time a full pass or query of the expert is complete. For real time operation, one query of the expert is made per pass and the YES and NO lists are emptied prior to beginning the query.

The flow chart can obviously be implemented in any programming language. We have done so in three languages in order to show development costs as well as execution benchmarks. Development cost estimates are very difficult to show,

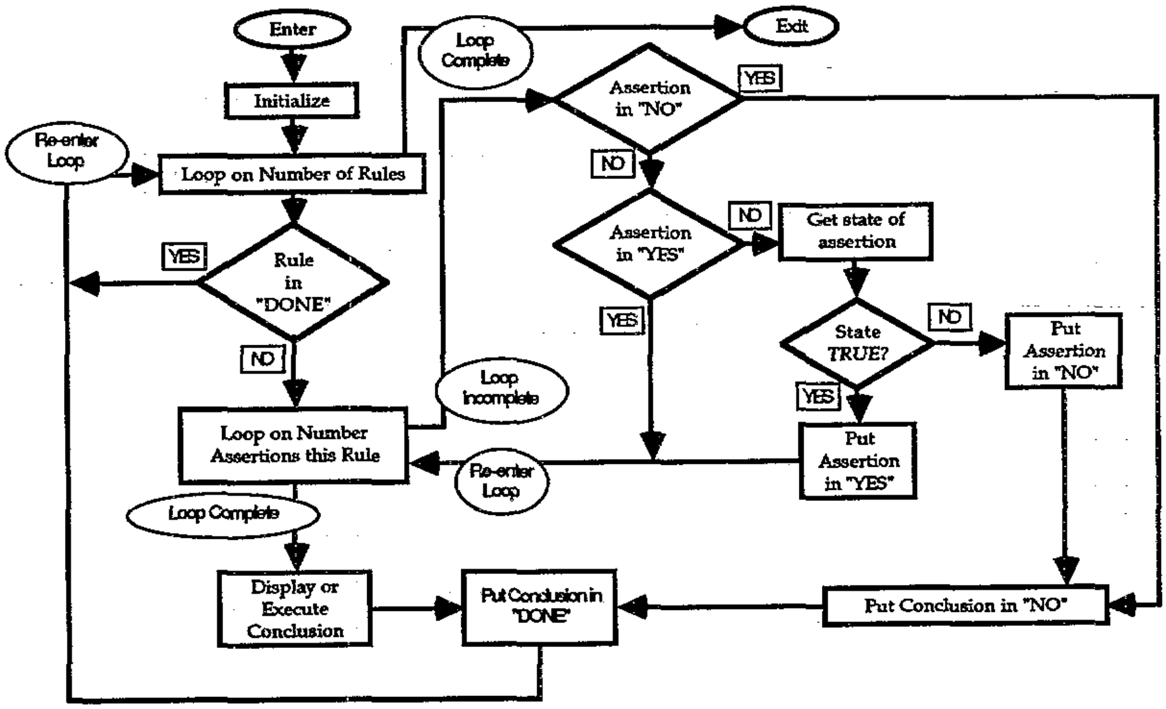

Figure 1. FLOW DIAGRAM FOR A GENERIC EXPERT SYSTEM SHELL

however, we have found that the Prolog environment supplies a rich set of tools allowing for rapid prototyping of not only expert systems but most any algorithm. Once the language is understood (which requires, compared to most languages, a relatively short learning curve), software development is rapid using high level constructs for string manipulation and list processing. Prolog's recursion properties provide efficient resultant code because compilers can easily perform optimizations. This is mainly due to the logic programming paradigm employed during the early definition of the language. It was designed with compiler optimization in mind. Prolog does execute somewhat slower in practice, though. The benchmarks are further discussed in the next section.

Fuzzy Logic

To provide further realism to the game, fuzzy logic can directly be inserted into the generic expert shell. We must be careful as to how fuzziness (or uncertainty) is introduced into the system. Fuzzy rules imply a probability that the defensive commander will actually know the answer to a conditional check. For example, the commander may respond that an aircraft is indeed in range of his weapons because he can read the range scale on his PPI. The problem is that the commander may well not be able to discern correct ranges, especially on a long range surveillance radar. He may only have a 60 percent probability of knowing a range accurately because of spot blooming and dither or because of equipment degradation. This 60 percent then becomes fuzzy information. The expert must handle this and can do so in several statistical ways. To illustrate the methods, an example will suffice:

```
RULE 3:
     IF    aggressor is in range of SAM 1
           site with probability 0.40

     AND   altitude of aggressor is greater
           then 2000 feet with probability
           0.60

     AND   SAM 2 has not fired with probab-
           ility 0.80

     AND   SAM 3 has not fired with probab-
           ility 0.80

     THEN  set command of SAM 1 to fire.

RULE 4:
     IF    aggressor is in range of SAM 2
           site with probability 0.40

     AND   altitude of aggressor is greater
           then 2000 feet with probability
           0.60

     AND   SAM 1 has not fired with probab-
           ility 0.80

     AND   SAM 3 has not fired with probab-
           ility 0.80
```

THEN set command of SAM 2 to fire.

RULE 5:
IF aggressor is in range of GUN 1 site with probability 0.40

AND altitude of aggressor is less than 2000 feet with probability 0.90

THEN set command of GUN 1 to fire.

The probabilities are the chances that a true answer is really true. That is, the commander only knows ranges to a probability of 0.60 (60%) and so on. Classical statistics says that probability of something occurring is the product of the probabilities of independent events occurring which lead to the result. Thus, if the expert asks the following questions and the responses are as shown:

IS aggressor in range of SAM 1 site? YES

IS altitude of aggressor greater than 2000 feet? YES

IS SAM 2 not fired? YES

IS SAM 3 not fired? YES

IS aggressor in range of SAM 2 site? YES

IS SAM 1 not fired? YES [Currently this is true - maybe not later]

then probabilities can be computed for each of the conclusions occurring. Because the second question (altitude greater than 2000 feet) was answered YES, the second condition of RULE 5 is false (Altitude is NOT less than 2000 feet.). The expert realizes this and computes probabilities on the statistical inverse of the 0.90 probability (That is, 1.0-0.-9). Classical theory says:

Probability of SAM 1 firing = 0.4 x 0.6 x 0.8 x 0.8 = 0.1536

Probability of SAM 2 firing = 0.4 x 0.6 x 0.8 x 0.8 = 0.1536

Probability of GUN 1 firing = 0.4 x (1.0 - 0.9) = 0.04

Note that the probability of the aggressor being below 2000 feet in RULE 5 is one minus the probability of it being above 2000 feet (since the first response was, YES). What do these probabilities mean? They supply suggestions to the commander as to whether or not to fire a given weapon. The probabilities are very small implying a small chance of hitting the target. But, this is incorrect. The commander should fire the SAMs but not the GUN. Classical theory falls apart in this context because there are some dependencies linking the conditions.

Other theories must be applied. The weakest link theory says to take the smallest probability found as the dominant feature. This is somewhat akin to a chain being only as strong as its weakest link. The probabilities then come out to 0.4, 0.4, and 0.1 for SAM 1, SAM 2, and GUN 1, respectively. These results are somewhat more meaningful, but still indicate chances are less than fifty-fifty. Finally, the strongest link theory says to use the highest probability, not the product nor the smallest. This implies that there is an overwhelming condition which indicates the weapon should be fired. Probabilities thus become 0.8, 0.8, and 0.4 - believable values. So the strongest link theory is most correct and should be used, but with great care taken in assigning the fuzzinesses.

PROLOG DEVELOPMENT

Prolog has, as its mainstay, an efficient unification algorithm. This means that Prolog interpreters and compilers are designed to allow for pattern matching as the single most important detail. For use in rule-based systems such as that described here, Prolog has a distinct advantage over other languages in that it efficiently matches textual strings. Other languages (procedural languages, like Ada, Pascal, and FORTRAN, as opposed to logic programming languages, like Prolog), are efficient computationally and lack Prolog's unification efficiency. Furthermore, Prolog is not overly concerned with data flow with external devices. It is meant for strict use within the computer. This reduces Prolog's interface capacity over other languages, but maintains the capacity to easily produce workable software - an environment known as rapid prototyping. Algorithms can be taken from design into Prolog code much more quickly than with languages which must have large initialization and setup code sections. Prolog needs nothing more than a "write" subroutine (or "predicate") to supply informative responses. Other languages need strong typing introductions and file opens and closes to communicate the most simple results.

Prolog Benchmarks

We ran some benchmarks, comparing Prolog, Pascal, and interpreted LISP on the example scenario. All systems used a rule base composed of twelve rules (two rules for each of the six platforms), with a total of 54 conditions. The benchmarks used a method that responded to all conditional checks with different "answers". That is, a list of 54 "YES" answers was generated with the first element being the answer to the first conditional check, the second to the second, and so on. A full pass through the expert was made, the number of logical inferences (number of conditional checks) was totaled, and the amount of real time used was obtained.

For the next expert query, the element in the "YES" list which corresponded to the number of inferences in that past query was changed to a "NO" (or if it was already a "NO", it was changed to a "YES" and the previous element was changed to a "NO", unless it was already a "NO", etc.). In this way, all possible combinations of results were tested because all possible sets of answers to the conditional checks were supplied. With this method, a reasonable determination of expert system speed could be obtained. The total number of inferences was divided by the total real time used giving a value for logical inferences per second (LIPS), a standard for measuring expert system rates. Some undistinguished numbers appeared from these benchmarks, but a more important result surfaced. (Prolog, although compiled to machine code, ran about 20 times slower than Pascal and about the same rate as interpreted LISP. These rates are not conclusive, however, and are really beside the point.)

While running the benchmarks, we attempted a more "intelligent", off-line ordering of the rules. That is, we attempted to move any conditions that were checked quite often to the beginning of rules, so that they would be checked early. If these conditions failed or they appeared in the "NO" list, fewer other conditions would need to be checked. This should increase the real time response of the expert. To our surprise, this re-ordering decreased the LIPS for that run by a significant percentage. However, the amount of time between each query was quite low. This paradox can be explained because Prolog needs some significant time to "set up" for its pass through a rule. If the rule has many conditions, Prolog executes the average condition (inference) faster. Thus, few inferences per pass imply low LIPS. But, of course, few inferences per pass imply little time between passes - an advantageous situation in real time operation.

CONCLUSIONS

The tactical threat simulation environment, in all of its complexity, does not have to be unrealistic nor demanding on instructional manpower. A rule-based expert system can operate the threat environment automatically and realistically through the use of English-like textual rules governing the threatening protocols. Fuzzy logic techniques add to the realism. All of this power can be attained through the Prolog programming language at relatively small development costs.

The IR&D project, upon which this paper is based, has been successful. Although only a simplified mission has been implemented, the project has shown that realistic, automated systems can provide a quality training environment while reducing instructor workloads. The implementation of the automated system in Prolog has proven to be worthwhile in operation, in decreased development effort, and in producing a realistic environment. We foresee the use of other AI techniques in the tactical simulation field to produce the same quality of results. These fields include an extension of the threat scenario, automated scoring and check riding, and mission generation.

REFERENCES

Sterling, L. and Shapiro, E., The Art of Prolog, The MIT Press, Cambridge, MA, 1986.

Levine, R. et al, A Comprehensive Guide to AI and Expert Systems, McGraw-Hill, NY, 1986.

O'Shea, T. and Eisenstadt, M., Artificial Intelligence Tools, Techniques, and Applications, Harper & Row, NY, 1984.

Townsend, C., Mastering Expert Systems with Turbo Prolog, Sams, Indianapolis, 1987.

BIOGRAPHY

Mr. O'Reilly is a Senior Staff Engineer with Reflectone in Tampa, FL. He was formerly with the BDM Corp. and AB Associates working in computer aided design and network analysis. He holds B.S. and M.S. degrees in Electrical Engineering from the University of South Florida. He is currently working under IR&D at Reflectone in the Electronic Warfare and AI areas.