Computer Science and Systems Analysis

Computer Science and Systems Analysis

Technical Reports

Miami University *Year* 1988

A Language for Rule-based Systems

James Kiper
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1988-006

A Language for Rule-based Systems

James D. Kiper

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

A Language for Rule-based Systems

by

James D. Kiper
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #88-006 09/88

Appendix: A Language for Rule-based Systems

1. The Problem

Expert systems are proliferating in many situations in which it is important to capture expertise in a computer system. This type of system is useful in situations in which human expertise is expensive or difficult to obtain or in which the operating environment is too dangerous for a person. Expert systems are used to address the following categories of problems: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, and control. [Hayes-Roth]

Expert system have now moved out of the laboratory and are being used in production environments. Herein lies the problem addressed by this research. Expert systems have traditionally been used in a research environment in which the software engineering of the product is not particularly important. Production environments are much more demanding. The quality necessary for continual use and abuse is not generally built into research quality expert systems. The problem is further exacerbated when an expert system is to be embedded in an autonomous system for which human interaction is difficult. (For example, an expert system could be used to drive a robot in a hazardous environment. If the expert system fails, it may not be easy for a human to reach the robot for repair.) Quality in these situations is vital.

2. Expert System Background

There are several techniques used to implement expert systems. One of the most commonly used is a rule-based system. This type of expert system is the subject of this research. In particular, this research has used the CLIPS expert system shell as a test bed. [Giarratano] An expert system shell may be conceptualized as an expert system without its expert knowledge. That is , it consists of inference mechanisms, and explanation support facilities. [Subrahmanyam]

Typical rule-based system architecture consists of three major components: a rule-base, an inference engine, and a global fact base. The rule base is composed of conditional statements. Such a conditional statement, or rule, is composed of a set of antecedent conditions and a set of actions. The inference engine is a computer program which determines which conditional statement will be executed, or fired. This inference engine evaluates the antecedent of each rule. Any rule for which the antecedent condition set is satisfied is added to the agenda of rules which are eligible to be executed. The most important task of the inference algorithm is to decide which rule to execute from among all those which are eligible to be executed. Two common inference engine strategies are "backward chaining" and "forward chaining".

The global fact base is the set of all facts which a particular expert system has determined to be true. These facts can arise from information provide by the user, from conclusions asserted by rules which are fired, or from initial conditions of the system. This fact base is consulted each time that the inference engine needs to determine the truth or falsehood of antecedent conditions.

The expert system shell CLIPS ('C' Language Production System) was developed at the Johnston Space Center by Gary Riley and Barry Cameron [Culbert]. It is being used by a growing number of NASA and Air Force sites and among many of their contractors. This product is in the public domain and is available from Cosmic, the NASA Software Distribution Center. (The C language source code will also be supplied.) Expert systems written in Clips can be run in stand alone mode, or can be embedded in another C program. The syntax of Clips closely resembles that of Lisp with its parenthesized list.

3. The Language Problem

Languages typically used to develop expert systems have been chiefly characterized by their flexibility. This property has been important to the rapid prototyping of research quality expert systems. It permits researchers to quickly develop a working system to demonstrate the feasibility of an idea. If the idea is not successful, the researcher has not invested many resources into the expert system development.

With the move to production quality systems, flexibility is no longer the most important quality in a development language. Expert system developed in such languages, typified by Clips, have many properties which make quality development difficult.

An examination of an expert system rule base developed in one of these languages reveals its monolithic nature. There is no division of the rule base into readable portions. Rules bases of 200 rules are quite common with larger systems approaching 1000 rules. Commonly used expert system development languages provide no syntactic components to help organize this rule base into smaller, more understandable units. Many developers try to use comments and groupings of related rules to control this monolithic rule base, but this is an arbitrary solution which depends upon the discipline of the developers. More typically, developers of prototype expert system mis-use this freedom. Rules are not logically grouped, but are added at the most convenient location. The resulting disorganized rule base functions adequately when rigorous operation and maintenance of the system are not high priorities, but is considerably less than desirable for a system in which quality is paramount.

These languages typically are not typed. A single variables can be used for multiple kinds of data in a single rule. This is another area of flexibility which researchers have appreciated. However, in developing a large system to be used in a production environment, this flexibility in typing can be a source of errors which are quite difficult to find and repair.

The antecedent portion of rules are generally a sequence of conditions which are combined by logical operators. It is frequently the case that a particular rule will share several of these conditions with another rule. In this situation, these rules must duplicate the condition. There is no syntactic mechanism for stating that a given condition applies to several conditions. This duplication is a potential source of error and confusion. Errors can occur because the duplicate conditions may be stated accurately in one rule, but mis-stated elsewhere. The testing and debugging of one rule should simplify the verification of the other related rules which use the same conditions, but does not. Confusion can arise when the same condition is tested in multiple rules, but different variable names are used in the statement of the conditions. Two conditions which have an identical affect may not appear alike syntactically.

The complexity inherent in a monolithic, unorganized rule base means that rules frequently interact in unanticipated ways. Changing assertions about facts in the conclusions of a particular rule affect all the rules which use these facts in their antecedent. Without some syntactic organization of that information, a change to one rule may require an exhaustive search of the entire rule base to find affected rules. Since these rules may in turn have to be modified, the potential effect of a change in one rule may ripple throughout the rule base.

All data which is to be maintained after a rule fires must be stored in the global fact base. This single repository of facts gives the developer the ability to retrieve any required information without concern over access rights to that data. This flexibility is at the price of giving each rule access to all facts. This has the potential of permitting a rule to produce a side effect which implicitly modifies the functioning of another rule. This sort of error is difficult to repair in a large system. Furthermore, subsequent modification to the rule base that frequently occurs during the maintenance phase may have unanticipated interactions with other rules through the global fact base.

We have mentioned several times that modifications or additions to the rule base may require an examination of many rules to understand which are affected by modifications or additions. The difficulty of this search is increased by the fact that it is necessary to examine all the details of the rule to determine how it will be impacted. There is no syntactic mechanism for denoting those components of the global fact base which are used or modified.

In summary some of the problems with rule bases implemented in current expert system development languages are:

1. monolithic, complex rule bases.
2. no typing of variables.
3. implicit sharing of conditions among rules.
4. interactions among rules in implicit, but sophisticated ways.
5. all persistent data is global.
6. global data used or modified by a rule is apparent only from a detailed reading of the rule.

Each of these qualities can be seen as a benefit if the criterion used for the language is flexibility. Conversely, each of these is detrimental when the criteria is readability, modifiability, and maintainability.

3. Errors in Rule Bases.

The recent research literature has demonstrated an increasing concern with the problems inherent in developing expert systems for use in environments where rigor and quality are vital. The following categories of of errors have been identified [Bellman, Stachowitz]:

1. conflict - two rule have the same antecedents, but conflicting conclusions.
2. redundancy - two rule with the same antecedents, and the same conclusions.
3. subsumption - one rule subsumes the conditions of the other.
4. missing rules - one possible choice of the antecedents has not been mentioned in a rule.
5. deadends - rules whose conclusions have no effect on other rules or on the global fact base.
6. insufficient activation - a rule whose conditions can never be satisfied by firing of other rules.

This list is by no means exhaustive. However, an examination of each of these reveals that these are at least partially caused by lack of readability and understandability of current rule base description languages.

4. Applying Software Engineering Expertise to Expert Systems Development.

In the past few years there some papers have been published regarding the impact that Artificial Intelligence (AI) research can have on software engineering. [Simon, Mostow] Others have debated the benefits of Lisp, Prolog, and expert system shells for the development of expert systems. [Bobrow, Subrahmanyam] The focus of our research is that software engineering can have an important impact upon AI software, especially when that software is put into a production-quality product which demands rigor and dependable quality. Other researchers have indicated a need for an improved development environment which supports software engineering techniques. [Johnson]

Our research has identified the software engineering properties that an expert system development language should possess. We have ignored, in this research, any consideration of alternatives to rule bases as representations of expert system. In addition, we have chosen to ignore the inference engine and the global fact base. The description of the rule base by means of a language which incorporates many important software engineering principles is the focus of this paper.

The understandability of expert systems can be improved substantially by modularizing the rule base. An improved language should provide syntactic components to divide the rule base into smaller, logically related pieces. Many expert system developers currently attempt such an organization through a disciplined organization of the rule base. The development language should provide a syntactic component similar to Pascal's procedures for this purpose.

The standard method of containing complexity has become the use of abstraction. The language of choice should support these by providing for the nesting of the procedure-like syntactic units described above. This nesting has the effect of presenting rules in a layer manner. At the highest level broad categories of rules are presented. At lower levels, more details of the rules are revealed until the specification of rules is complete. Not only does this nesting make the rule base more understandable, it allows an easier development of a rule base by a team.

This nesting of rule groups has the added SE benefit of information hiding. The details of rules are hidden in the deeply nested rule groups. Distinct portions of the rule base are not tightly coupled if the rule details are hidden. The language should enforce this separation and protection in a syntax similar to that of Ada's packages.

The problem of implicit interactions among rules should be solved by making these interaction explicit. Corresponding to the parameters of a Pascal procedure, each grouping of rules should explicitly declare the global data which is to be used and that which potentially may be modified by this grouping. Having a syntactic component of the language to describe this information makes the maintenance task much easier. The maintenance programmer does not have

to read all the details of a rule to understand potential interaction with other rules. It also gives rise to the possibility of a tool to check this interaction.

Strong data typing has long been recognized as a property which allows for improved compilation diagnostics and which makes a program more readable. Typing of data in an rule base development language would have similar benefits. The reader or modifier of a rule base would then find the variables to be used in a "declaration" section which precedes the rule. This makes explicit some information about the name and type of variables which would otherwise be retrieved only via a study of the rule's details. Furthermore, it enforces a discipline upon the developer which can result in a higher quality product.

This typing and procedural-like packaging of rules give the possible of scoping of variables. Rather than forcing all permanent data to be globally visible, information should be made available in the packages which is hidden from other packages of rules. The language can easily check that this scoping is not violated through the type checking mechanism. (However, this feature of the language may also require a change in the structure of the global fact base.)

5. Future Research

We are currently in the process of defining a language called SEES (Software Engineering of Expert Systems) which possesses the properties described above. Subsequent to the complete definition, we plan to write a translator which will translate programs in SEES to equivalent programs in Clips. This will serve two purposes. First, it will provide equivalent SEES and CLIPS programs which can be compared for readability and easy of modification. Second, it will give SEES some added viability among the group of developers which already use Clips. They will be more easily convinced of the quality of the expert systems developed in SEES if they can examine the equivalent and more familiar Clips version. In the long term, we plan to build a compiler for SEES.

6. References

[Bellman] Bellman, Kristie, "Testing and Correcting Rule-Based Expert Systems", Computer Science Laboratory, Aerospace Corporation, presented at the Space Quality Conference, April 20, 1988.

[Bobrow] Bobrow, Daniel G. , "If Prolog is the Answer, What is the Question? or What it Takes to Support AI Programming Paradigms", IEEE Transactions on Software Engineering, SE-11, 11, November 1985, pp. 1401-1408.

[Culbert] Culbert, Chris, Clips Reference Manual, Artificial Intelligence Section, Lyndon B. Johnson Space Center, April, 1988.

[Giarratano] Giarratano, Joseph C., **Clips User's Guide**, Artificial Intelligence Section, Lyndon B. Johnson Space Center, July 3, 1988.

[Hayes-Roth] Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat, Building Expert Systems, Addison-Wesley Publishing Company, Inc., 1983.

[Johnson] Johnson, Sally C., "Validation of Highly Reliable Real-Time Knowledge-Based Systems", preprint from Proceedings SOAR 88 Workshop of Automation and Robotics, Wright State University, Dayton, Ohio, July 20-23, 1988.

[Mostow] Mostow, Jack, "What is AI? And What Does IT Have to Do with Software Engineering?", IEEE Transactions on Software Engineering, SE-11, 11, November 1985, pp. 1253-1256.

[Simon] Simon, Herbert, "Whether Software Engineering Needs to Be Artificially Intelligent", IEEE Transactions on Software Engineering, SE-12, 7, July 1986, pp. 726-732.

[Stachowitz] Stachowitz, R.A., J.B. Combs, and C.L. Chang, "Validation of Knowledge-Based Systems", Second AIAA/NASA/USAF symposium on Automation, Robotics and Advanced Computing for the National Space Program, March 9-11, 1987, Arlington, Va.

[Subrahmanyam] Subrahmanyam, P.A., "The 'Software Engineering' of Expert Systems: Is Prolog Appropriate?", IEEE Transactions on Software Engineering, SE-11, 11, November 1985, pp. 1391-1400.