CUCS-69-83

Table-driven Rules in Expert Systems

Alexander Pasik, Marshall Schor

August 17th, 1983

Department of Computer Science
Columbia University
New York City, New York 10027

Department of Mathematical Sciences
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract

The structure and organization of expert systems can be usefully modeled after corresponding human experts. Often this modeling degrades because of insufficient expressive power in production system languages. Relational table techniques provide additional abstraction capabilities and are useful in extending the expressiveness of production system rules; the resulting systems can be easier to build, understand and debug because they can reflect more accurately human methods of reasoning. The number of superfluous rules is reduced by organizing much of the problem domain knowledge in relations in working memory. The relational table methods also provide a tool for the interfacing of knowledge bases and databases.

1. Introduction

Production Rule Systems

The knowledge base of many expert systems is composed of a set of *production rules* together with a set of *working memory elements*. The *inference engine* matches the left hand side of each rule against working memory to determine which rules are satisfied. One of the satisfied rules is selected and the actions on its right hand side are performed. These actions in general alter the working memory so that on the next inference cycle, a different set of productions is satisfied.

Using table-driven rules can ease the rule writing process in building expert systems. OPS5 [Forgy 81], a high speed production system interpreter, lends itself well to the implementation of the relational table techniques which will be described.

Elegance (and Lack of it) in Rule Writing

While building an expert system, knowledge engineers attempt to encapsulate discrete pieces of knowledge into single rules. This has several benefits. Human experts in a particular domain often can explain their expertise in small units, each of which is translated into a production rule. This strengthens the correspondence between human performance and the expert system, making the system easier to understand and debug.

Unfortunately, it is often difficult to encode a unit of knowledge into one rule. Sometimes this is inherent in the type of knowledge, but occasionally it is the result of the lack of certain structures in production system languages.

In many procedural languages, the *Case Statement* is used to unify a collection of related but conditionally exclusive actions. Intuitively, production rules can be written for each condition in a desired case statement but again this does not reflect the singularity of the unit of knowledge that must be encoded. For example, if the elements of the class **person** were to be classified into the groups **SHORT**, **MEDIUM** and **TALL**, a case statement would express this as demonstrated in Figure 1.

```
case person.height of
  (< 5.0) : person.category := SHORT
  (> 6.0) : person.category := TALL
   (else) : person.category := MEDIUM
  endcase
```

Figure 1. Case statement to categorize persons by height

Since OPS5 does not allow conditional statements in the right hand side of rules, the case statement construct in not available. Instead, three OPS5 rules can be written to encode this process, one corresponding to each clause in the case statement (see Figure 2).[1]

[1] For all rules shown in this paper, English equivalents will be supplied. Thus it is not absolutely necessary to understand OPS5 syntax.

```
Short-person:
    IF   a person has height < 5
         and is not categorized,
    THEN modify the person's category to be SHORT.

(p short-person
    ( person ^height < 5.0 ^cat ?) <the-person> )
    -->
     (modify <the-person> ^cat SHORT))

Tall-person:
    IF   a person has height > 6
         and is not categorized,
    THEN modify the person's category to be TALL.

(p tall-person
    ( person ^height > 6.0 ^cat ?) <the-person> )
    -->
     (modify <the-person> ^cat TALL))

Medium-person:
    IF   a person has height between 5 and 6
         and is not categorized,
    THEN modify the person's category to be MEDIUM.

(p medium-person
    ( person ^height (<= 6.0
                     >= 5.0) ^cat ?) <the-person> )
    -->
    (modify <the-person> ^category MEDIUM))
```

Figure 2. Three OPS5 rules to categorize persons by height

2. Using Relational Tables in Rule Writing

The Relational Table Technique

Relational tables structure information into *relations*. Each relation is made up of a collection of *tuples* each of which represents a group of objects that are related to each other. For example, the relation **party-of** can be defined over 2-tuples (i.e. doubles) in which the domain of one element is the set of all presidents and that of the other is the set of all political parties. One tuple in this relation can be expressed as **party-of**(president=Kennedy,party=Democrat). The complete relation can be expressed in a table. A small portion of the relation is shown below.

President	Party
Reagan	Republican
Carter	Democrat
Ford	Republican
Nixon	Republican
Johnson	Democrat
Kennedy	Democrat
Eisenhower	Republican
Truman	Democrat
Roosevelt	Democrat

There is a natural analogy between relational table constructs and OPS5 working memory. A relation and its domains can be represented by a class and its attributes respectively, and the values of a tuple by values in a working memory element. The tuple **party-of**(president=Truman, party=Democrat) could appear in working memory as (party-of ^president Truman ^party Democrat).

Working Memory as Long Term Memory

Production rules are often referred to as long term memory, and working memory elements are considered short term. In other words, knowledge of the problem domain resides in the rule base, whereas the state of the particular problem being solved is described in working memory.

The relational table technique uses working memory as long term memory by encoding much of the problem domain knowledge in relations. Much of the inference process is then table driven according to the relations in working memory. This has the effect of decreasing the number of rules which then can more accurately reflect the processing of the human expert.

Rules Driven by Relations

Using the example of the categorization of persons by their height, the relational table technique improves the clarity of that portion of the expert system (see Figure 3). The relation **height-cat** would be composed of the the three tuples in working memory. The knowledge involved in this categorization is expressed in one simple rule, driven by the relation between heights and categories. This structure supports additional levels of abstraction allowing greater expressive power.

```
(height-cat ↑low 0.0 ↑high 5.0 ↑cat SHORT)
(height-cat ↑low 5.0 ↑high 6.0 ↑cat MEDIUM)
(height-cat ↑low 6.0 ↑high 9.0 ↑cat TALL)


Categorize-heights:
    IF   a person has height <H>
         and is not categorized,
         AND
         <H> is within height-cat <C>,
    THEN modify the person's category to be <C>.

(p categorize-heights
    { (person ↑height <H> ↑cat ?) <the-person> }
    (height-cat ↑low <= <H> ↑high > <H> ↑cat <C>)
    -->
    (modify <the-person> ↑cat <C>))
```

Figure 3. The HEIGHT-CAT relation improves the system

Table-driven Attribute Selection

In OPS5, relations can be used to encode information about attributes also. For example, if one working memory element is to keep track of how many persons of each height exist, the relation and rule shown in Figure 4 could be used for the categorization. The OPS5 function **substr** is used to reference the old value of the attribute <C> of the counter.[2] At a given time, the counter element might be (counter ↑SHORT 27 ↑MEDIUM 152 ↑TALL 112). Each time the rule fires, a person gets categorized and one of the counter attributes gets incremented.

---

[2] OPS5's **substr** has the following format and function. (substr WME ATT1 ATT2) returns the values of working memory element WME from the position designated by attribute ATT1 through the position designated by attribute ATT2. Thus (substr <the-counter> <C> <C>) returns the value of the counter at the attribute <C>.

```
(height-cat ^low 0.0 ^high 5.0 ^cat SHORT)
(height-cat ^low 5.0 ^high 6.0 ^cat MEDIUM)
(height-cat ^low 6.0 ^high 9.0 ^cat TALL)


Categorize-heights:
    IF   a person has height <H>
         and is not categorized,
         AND
         a counter exists
         AND
         <H> is within height-cat <C>,
    THEN modify the person's category to be <C>,
         AND
         increment the <C> attribute
         of the counter by 1.

(p categorize-heights
   { (person ^height <H> ^cat 0) <the-person> }
   { (counter) <the-counter> }
    (height-cat ^low <= <H> ^high > <H> ^cat <C>)
   -->
   (modify <the-person> ^cat <C>)
   (bind <V> (substr <the-counter> <C> <C>))
   (modify <the-counter> ^<C> (compute <V> + 1)))
```

Figure 4. Values in tuples can be attribute names

Table-driven State Transitions

Another important use of this technique is the management of problem solving stages within the system. Expert systems using the *Match* problem-solving strategy [Newell 69] are composed of groups of rules, each group designed to solve a sub-problem within the system. When one sub-problem is solved, a transition rule fires which results in making the next set of rules active. This transition may need to be accompanied by the addition or reorganization of relevant information in working memory before the next stage begins. These *transition values* can be organized in a relation in working memory.

An expert system designed to prepare gourmet meals would be built in terms of solving various sub-problems. The stages would include cooking the entree, appetizers and dessert as well as buying the ingredients and serving the meal. A relation for the organization of the transition values can reside in working memory as in Figure 5. By using this relation, transition rules between each stage all collapse to the one rule shown.

```
(trans ^from start      ^to buy        ^need money)
(trans ^from buy        ^to appetizers ^need a-ingr)
(trans ^from appetizers ^to dessert    ^need d-ingr)
(trans ^from dessert    ^to entree     ^need e-ingr)
(trans ^from entree     ^to serve      ^need places)
(trans ^from serve      ^to done       ^need none)

Change-stages
   IF   the goal to solve problem <A> is satisfied,
        AND
        there is a transition from <A> to <B>
        which needs material <C>,
   THEN remove the goal to solve <A>
        AND
        create a new goal to solve <B> with a
        prerequisite to obtain material <C>.

(p change-stages
   { (goal ^name <A> ^status SATISFIED) <old> }
    (trans ^from <A> ^to <B> ^need <C>)
   -->
   (remove <old>)
   (make goal ^name <B> ^prerequisite <C>))
```

Figure 5. The transition states in a gourmet expert system

3. Applications of the Technique

In the examples presented, the relational tables were of only a few tuples. The relational table technique has a greater impact on the clarity of a system when the relations are large. An expert system to suggest travel routes using the Manhattan subway and bus systems was built at Columbia University as a project for a course in expert systems [Lerner and Cheng 83]. The system used several relational tables including one of over 500 tuples to determine intersections of the rapid transit system. Use of this technique dramatically reduced the number of rules and succeeded in separating the true procedural knowledge from the database which described the transit map of the city.

Besides the structural benefits of this method, the relational table technique provides a tool that knowledge engineers can use in another emerging application: the connection of expert systems and databases [Walker 83]. As databases grow and require maintenance, analysts are hired whose responsibility it is to examine large quantities of data and make management decisions based on the information gathered and computed daily by the machines. This data analysis is performed by trained human experts. With the current expert system technology, intelligent systems can be (and have been) built to analyze these large databases [Stolfo and Vesonder 82]. The relational table techniques can provide knowledge engineers with a tool for the manipulation of portions of the databases in working memory. Information structured in a relational database can be mapped into working memory elements in a natural way (one tuple results in one working memory element). The working memory elements are then used by the rule base.

The relational table technique described can be used in almost any expert system. Knowledge in a particular domain is characterized by rules that solve problems. Expert systems have been constructed with a multitude of additional rules which, though syntactically separate, encode the same knowledge as the original group of rules, varying only in the values of certain parameters [McDermott 1982]. By building relational tables in working memory consisting of related values and attributes, the superfluous rules can be eliminated. Benefits of these techniques include increased ease of preparation and improved readability. Systems are easier to maintain because the procedural knowledge will be condensed. An erroneous result due to a portion of the system can be repaired by the inspection of one rule and a uniform table rather than relying on the complete understanding of a large number of rules that are only slightly different from each other. It has already been demonstrated that relational tables can improve expert systems in general. With the upcoming demand for expert data analysis, the importance of these methods will continue to increase.

References

Forgy, C. L. "OPS5 User's Manual" Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

Lerner, M. and Cheng, J. "The Manhattan Mapper" Expert Systems Course Project, Columbia University, Department of Computer Science, 1983. (unpublished)

McDermott, J. "R1: A Rule Based Configurer of Computer Systems" Artificial Intelligence, Volume 19(1), pp. 39-88, September 1982.

Newell, A. "Heuristic Programming: Ill-structured Problems" In J. S. Aronofsky (Ed.), *Progress in Operations Research*, John Wiley and Sons, pp. 361-414, 1969.

Stolfo, S. J. and Vesonder, G. "ACE: An Expert System Supporting Analysis and Management Decision Making" Technical Report, Columbia University, Department of Computer Science, 1982.

Walker, A. "Data Bases, Expert Systems, and Prolog" Technical Report RJ 3870, IBM Research Laboratory San Jose, 1983.