JKAU: Sci., **vol. 13,** pp. 63-82 (1421 A.H. / 2001 A.D.)

Matrix Verification of Knowledge-Based System

SAUD M.A. MAGHRABI
Department of Mathematical Sciences,
Faculty of Applied Sciences,
Umm Al-Qura University, Makkah, Saudi Arabia

ABSTRACT. The paper describes a method, which has been designed and implemented, for the verification of rule-base as a matrix and then uses standard matrix transformation to determine the location of errors and anomalies contained within the rule-base, such as duplication, subsumption, circular rule sets, inconsistency, missing links, auxiliary rule sets, and redundancy. It has been demonstrated to be effective and simple. It has a system designed for establishing the context of the environment. The system has four main routines: the loader routine, matrix manipulation routine, and testing routine, and the output routine. The cost of using the system is measured. The method has proved the practicality of automatic computer-based errors detection using matrices. Testing routines have been clarified by examples. The efficiency of the method has been discussed. A comparison has been made between this method and other related methods.

KEY WORDS: Expert system, algorithm on rule-bases, verification of Knowledge-Based Systems, matrix manipulation.

1. Introduction

Knowledge Based Systems (KBS) are commonly being used in situations were there are erroneous function can have a major impact upon the locality as a whole. Verification and Validation of Knowledge Based Systems have been given a variety of definitions[1-12]. Verification is the process of determining whether a KBS completely satisfies its specifications, while validation is the process of determining whether a KBS satisfactorily performs the real-world tasks for which it was created[1]. In reference [1], the definitions are expressed by questions, "Are we building the product right?" (Verification), and "Are we building the right product?" (Validation). Validation has therefore become the process of checking that the system actually does what the customer wants of it. We can define verification in terms of knowledge based systems as the process of checking that the systems operation is right and was arrived at in the correct manner. The verification of knowledge based systems (KBS) is therefore of major importance to address the consequences of failure. This paper is thus concerned with the verification of rule-bases.

The aim of this paper is to investigate the use of the matrix technique as a method of statically verifying rule-bases and knowledge based systems in general. The idea is to represent a rule-base as a matrix and then to utilize standard matrix transformations. The method has been implemented as a program, to illustrate the use of this matrix technique in the verification of rule-bases and knowledge based systems in general.

A system has been designed and implemented to establish the context of the environment. It has four major routines: the loader routine, The matrix manipulation routine, tests routine, and output routine.

To be able to enter a rule-base into the system, the loader routine needs to be able to understand one of the recognizable modes of rule-base representation. This task is similar to that of interpreter but needs to give the results in terms of a matrix, as well as producing any ancillary tables required during the operation of the system. The system will receive rule-bases of the folding form:

$$a\,b \rightarrow c$$

$$f g \rightarrow i$$

At present, the system detects the following anomalies: subsumption, duplication, redundancy, missing links, inconsistency, circular rule sets, and auxiliary rule sets. Tests have been devised through many examples. The system produces the results in an easily understandable form. The results will be placed into a text file to allow the recall of the data in a semi-permanent fashion, and to allow hard copies to be made.

2. Errors Commonly Occurring within Rule-bases

There is a wide debate concerning the possible errors or anomalies that can exist in a rule-base with each author having their own "definitive"[2-12]. The system currently tests for redundancy, duplication, subsumption, circular rule sets, auxiliary rule sets, inconsistency, and missing links. It is worth describing briefly these errors and anomalies in this section.

A rule-based contains a redundancy if a rule, or group of rules, play no part in the establishment of any root proposition, or propositions. This can manifest itself in one of two ways. Firstly, if a rule makes no reference to any other rule when the rule or group of rules are not connected to the rest of the rule-bases. Secondly, redundancy can also occur if a rule exists as some kind of intermediary step in the derivation of some root proposition, but does not have any contact with any input data, or vice-versa.

Duplication, also known as absolute subsumption[3], occurs when the situation arises that two rules exist which are the same or differ only in the order of their propositions. While this only causes inefficiency, problems may arise when an update is made of the rule-base and only one copy of the rule is changed.

Complex subsumption occurs when one rule is a more specialized form of another. In other words, it appears in the following cases:

– If one rule has duplicate consequent but the antecedents of one rule form a subset of the other.
– When the reverse is true and the antecedents are the same but the consequence of one rule is a subset of the other.
– A mixture of these two states.

Circularity presents an urgent problem in knowledge based systems as a non-terminating loop can be created when the rule-base is fired. A cycle can be one of two types, it can be a direct cycle if the rule call itself in its' own definition, although it is unlikely in practice that anybody would write such a rule or an indirect cycle. An indirect cycle is essentially the same as a direct cycle but the consequent is separated from the antecedent by a number of intervening steps.

An auxiliary rule is a rule that has been included in a rule-base solely to reflect the structure of the domain, or to add structure and comprehension, or to facilitate future modification. This however introduces undesirable inefficiencies. Basically, an auxiliary rule is said to exist, if there is a rule that subsumes two or more existing rules in the rule base.

An inconsistency occurs when a rule-base allows different contradictory conclusions to be drawn from the same set of facts. Missing links happens when a rule only shares one proposition with another rule and this is not a root rule or a leave rule.

3. System Architecture

The system accepts a rule-base and then provides a commentary on any anomalies found therein. Figure 1a demonstrates the top-level architecture of our system to verify rule-bases using the matrix method. Figure 1b shows the architecture of the matrix manipulator.

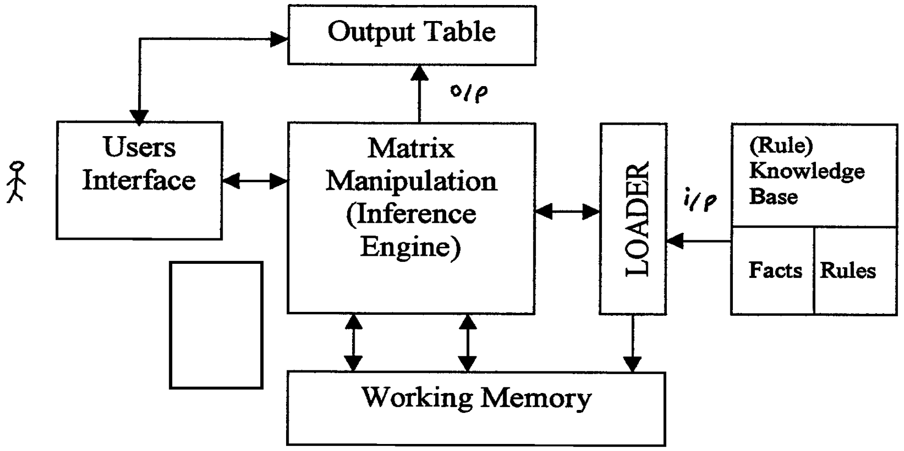

FIG. 1a. System architecture.

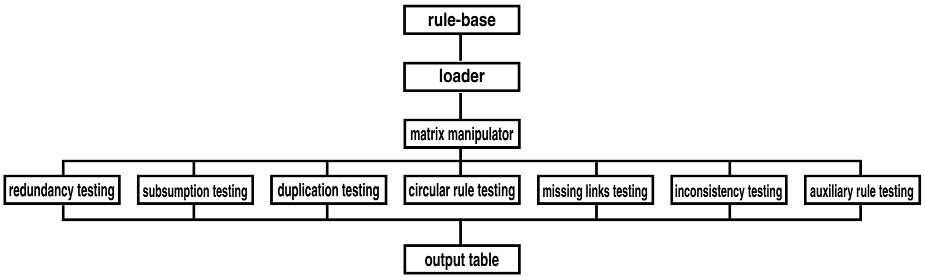

FIG. 1b. Matrix manipulator system architecture.

As can be seen from the figures, the system requires a loading routine to actually enter the rule-base and construct the matrix. The loading routine used is fairly simple, although producing a more versatile one would merely be an exercise in lexical analysis. The loader accepts rules in the *A B C* → *D E* style format, where each character or string corresponds to a proposition, the spaces between propositions to logical *AND*. The next process is then to take the matrix that has been loaded and then generating the resulting matrix. As this requires the multiplication of the matrix with its transpose a routine is required to first calculate the transpose before the multiplication can take place.

Each of the tests has difference outputs and requires their own output routine. The modular form of the system will allow addition of further tests, such as dead end rule, irrelevant propositions, and incompleteness.

4. The Loader

The task of the loader is to scan the text file containing the rule-bases, to then interpret it, and finally to place the data in its' relevant place within the matrix which represents the rule-bases. At the same time a symbol table must be derived containing all of the names of the propositions in the order they occur in the matrix, and also construct an array to illustrate those propositions in a rule that occur on the right hand side of the implied sign. Propositions in a rule that occur on the right hand side of the implied sign correspond to the head of the rule and the propositions that occur on the left hand side of the implied sign are the tail. For example, if a rule is presented as $b\ c\ d \rightarrow a$, then the a is the head and the $b\ c\ d$ corresponds to the tail.

The module works using one pass taking in the rule-base character by character using spaces as delimiters. Once a word has been entered, it is compared with a table of reserved words and characters and ignored if a match is made. Any word, which does not match, is then assumed to be a proposition and is then compared with those entries within the symbol table that contains the list of propositions. If a match is found, an entry is placed in the matrix in the sample place as that in the array as each position in the symbol table corresponds to its' position in the matrix. However, if a match is not found for a word, the word is placed in the symbol table after the last previous entry. An entry is then made in the matrix array at the position corresponding to the new entry. The number of names in the symbol table corresponds to the number of different propositions in the rule-base, which in turn corresponds to the number of columns in the matrix. Therefore, whenever an entry is made to the symbol table, a variable keeping score of the number of different propositions must also be incremented.

When the end of a line is reached, a variable keeping score of the number of rules is incremented. This variable also corresponds to the number of rows in the matrix. The matrix exists as a two-dimensional array of one hundred by one hundred elements of type integer. This matrix has been initialized by being filled with zeroes. Therefore, when an entry is made, a one is added to that position with the active part of the array defined by those variables keeping score of the number of rules and the number of propositions. The symbol table is an array of one hundred elements of type string with a maximum length of twenty-five characters. The twenty-five characters limit is also, therefore, the limits in the lengths of any proposition name.

In short, the module accepts characters from a text file and returns three arrays. The first array contains the matrix. The second array consists of the list of propositions, and the third holds the locations of all the heads to the rules.

5. The Use of Matrices

The approach to be used in the rule-base verification system requires the use of matrices. These matrices will be used to represent the rule-base in question. When these matrices are manipulated using standard matrix operations, they can be used to highlight any anomalies that may be presented in the rule-base.

A rule is represented by a one-row matrix, where every column in the matrix corresponds to a proposition in the rule. We can then think of a rule-base as a (NxM) matrix where each row represents separate rule and each column represents each possible proposition within the rule-base. For example, if we consider a rule as below:

Rule 1:

Someone is a Tiger if
Someone has stripes
Someone eats meat and
Someone is a mammal

We can now consider this to be a rule with four propositions, which are:

a) Someone is a Tiger
b) Someone has stripes
c) Someone eats meat
d) Someone is a mammal.

Therefore, if we represent each of these propositions by their corresponding letters in turn, we obtain a proposition that can be expressed in the following way:

$$a \text{ if } b \text{ and } c \text{ and } d$$

or to be even more concise:

$$b\ c\ d \rightarrow a$$

Where the symbol '$\rightarrow$' means ‘implies’. This last notation will be used in the rest of this paper and is also used for the formatting of the data representing the rule-base which is to be entered into the program that implements the system. However, it should be noted that only conjunctive proposition could be implemented using this method with any disjunction having first to be converted into a conjunctive form, using standard logical equivalencies, before they can be implemented using this method.

A matrix has a table of entries, each of which belongs to one row and one column respectively. As it has already been stated, we can consider a row to represent a rule and a column to represent a proposition. This allows us to represent the previous example $b\ c\ d \rightarrow a$ as (1111). If we now consider two more rules such as:

$$ef \rightarrow c$$

$$g\,h \rightarrow d$$

This can now be combined with our original rule to give us a single matrix with three rows and eight columns, thus:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

This matrix can then be manipulated using the transpose of the matrix. A transpose of a matrix contains the same entries as the original matrix but the columns in the old matrix become the rows in the transpose and vice-versa. The procedure that finds the transpose of the matrix is as follows:

If we then take the original matrix and multiply it by the transpose that we have just constructed, we obtain a matrix so:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 3 \end{pmatrix}$$

From this resultant matrix, we can see that the number of propositions the rule shares with itself has derived by the leading diagonal. The entries of the leading diagonal therefore highlight how many propositions there are within a corresponding rule.

The resultant matrix can now be used to find out several facts about the rule-base, which can then be used in the detection of errors and anomalies. For instance, we can note that each entry in the resultant matrix corresponds to the number of propositions, which are shared by the two rows and therefore the two rules. This is because each row and column represent the prerequisite row in the original matrix and therefore corresponds to the relevant rules in the rule-base. In the above resultant, it can be seen that the entry in row 1 column 2 equals one therefore it can be said that the first and second rules share a single proposition.

These matrices are the main tool used in the finding of the set of errors and anomalies.

In the next section, the detection procedure for each anomaly will be discussed in details.

6. Errors Detection Procedures

This section provides all of those mathematical functions utilized in the manipulation of matrices, as well as the identities used to detect each error. The tests included in this paper are redundancy, missing links, duplication, subsumption, inconsistency, circular rule sets, and auxiliary tests. Examples have been given for clarification.

6.1 Redundancy

This test is concerned with the identification of rules that share one proposition with other rules. The procedure containing the test works by taking each rule in turn and seeing which rules it shares a proposition with. When the total number is found then the number is applied to a conditional statement, which will assign that rule the value of redundant.

As has been stated, the resultant matrix has many features. These features can be used to identify anomalies within the rule-base. If we consider a rule-base that is disconnected, the disconnected section of the rule-base will not share any propositions with any other rule in the rule-base. For example, if we consider the following rule-bases:

$$\begin{aligned} b\,c &\to a \\ e\,f &\to g \\ g\,h &\to i \end{aligned}$$

Which gives us the matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Which in turn gives us the resultant:

$$\begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 1 \\ 0 & 1 & 3 \end{pmatrix}$$

We can see that the first rule does not share a proposition with any other rule, since the entries on its' rows and columns are zero except on the leading diagonal where it was multiplied by itself. The first rule is therefore disconnected and it is redundant.

6.2 Missing Links

There are some ideas that need to be described before we show how the missing links problem can be detected by the matrix technique. A root rule is a top-level rule designed to establish some end proposition. A leaf rule is a rule, which does not obtain data from the rules but from some other source, for example, user input. A body rule is any other rule, which is either a root rule or a body rule.

This test is concerned with the identification of those body rules that only share one proposition with one other rule. An error or anomaly can exist where a rule only shares one proposition with another rule and this is not a root rule or a leaf rule. In this case, there must be a missing link. If we now consider a connected rule-bases such as:

$$\begin{array}{l} b \rightarrow a \\ c \rightarrow b \\ d \rightarrow c \\ e \rightarrow d \end{array}$$

By our procedure, we obtain the following resultant.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

From the resultant matrix, given that the first rule is the root rule, forth rule is a leaf rule and all other rules are body rules, we can see that all the rules share at least one proposition with another rule and they are therefore connected free from missing links and redundancy. However, if we consider the following rule-bases, a different picture will emerge.

$$\begin{array}{l} b \rightarrow a \\ c \rightarrow b \\ d \rightarrow f \\ e \rightarrow d \end{array}$$

Which gives us the following multiplication:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

As can be seen in this example, if rule 1 is a root rule and rule 4 is a leaf rule then rule 2 and rule 3 are body rules and therefore need to share propositions with at least two other rules. An anomaly exists since rules 2 and 3 only shares one rule with another rule and so a missing link exists.

6.3 Duplication and Subsumption

In the test for duplication the aim is to find those rules which may be exact duplicates of each other. This is done by first finding the resultant matrix and then scanning through the resultant matrix using a pair of nested loops and highlighting those rules which have the same number of propositions and also share this same number of propositions in the duplicate column. As it has been already stated, the leading diagonal of the resultant matrix highlights how many propositions there are within each rule. We can demonstrate a duplicate exists by use of the following identity:

if R(i, i) = R(j, j) = R(i, j) then duplication exists

Where, *R(i, j)* refers to the resultant matrix and will be used in the rest of this paper. For example, if we consider the following rule-bases with an obvious duplication.

$$\begin{array}{c} b\ c \rightarrow a \\ d\,e \rightarrow b \\ f\,g \rightarrow d \\ f\,g \rightarrow d \end{array}$$

Which gives us the following:

$$\begin{pmatrix} 1110000 \\ 0101100 \\ 0001011 \\ 0001011 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 1 & 1 \\ 1 & 3 & 0 & 0 \\ 1 & 0 & 3 & 3 \\ 1 & 0 & 3 & 3 \end{pmatrix}$$

From the resultant matrix *R,* if we compare rule 3 with rule 4, we find that both rules contain three propositions and that they share three propositions. So,

R(3, 3) = R(4, 4) = R(3, 4) = 3, Therefore, rule 4 is a duplicate of rule 3.

The subsumption test is slightly more complicated with the aim being to detect those rules, which are a specialized case of another. This is achieved by scanning through the resultant in a similar fashion to the test for duplication, and highlighting those rules that fail the conditions if the numbers of propositions is greater in one rule than another and the smaller rule shares all of its propositions with the original rule then it has been subsumed by the original rule, or vice-versa.

Duplication and subsumption share a number of properties and their methods of identification are similar. This is especially true when we consider that duplication is also known as absolute subsumption[3], and complex subsumption involves a rule being duplicated as a part of another rule. Complex subsumption occurs when one rule is a more specialized version of another. A subsumed rule can be thought of as either a clause which is generated from a rule that is wholly subsumed, or as a clause which is generated from a rule that is not wholly subsumed, but which need never play a role in establishing a proposition. Complex subsumption can therefore be detected using the following two identities:

if R(i, i) > R(j, j) and (Ri, j) = R(j, j) then rule (j) subsumes rule (i), or
if R(j, j) > R(i, i) and R(i, j) = R(i, i) then rule (i) subsumes rule (j).

For example, if we consider the following rule-bases:

$$\begin{array}{c} d\ e \rightarrow b \\ b\,c \rightarrow a \\ d \rightarrow b \end{array}$$

Giving us the following matrices:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 2 \end{pmatrix}$$

From the resultant matrix, if we apply the identities to the first and third rules we obtain:

if R(1, 1) > R(3, 3) and R(1, 3) = R(3, 3), then, rule 1 subsumes rule 3.

6.4 Inconsistency

The procedure for inconsistency is to highlight those rules that share common tails and have different heads. For example, if the number of antecedents of two rules is the same, and the two rules share the same number of propositions then an inconsistency exists.

Consider the following rule-bases that are deliberately flawed:

$$\begin{gathered} a\,b \rightarrow c \\ a\,b \rightarrow d \\ e\,f \rightarrow b \end{gathered}$$

From these rules, we can see that rule 1 and rule 2 have the same antecedents but they have different consequence. If this is correct than *c* and *d* must be true at the same time, but if this is not the case then we have an obvious contradiction and an inconsistency exists. If we take the example rule-base mentioned above and construct the relevant matrices, we produce the following:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix}$$

If we now consider a variable *A(i)* which corresponds to the number of antecedent propositions in a rule *i,* we can define an identity to detect inconsistencies as follows:

If A(i) = A(j) = R(i, j), then, an inconsistency exists.

If we apply this identity to the resultant matrix *R* derived above, we find that:

A(1) = A(2) = R(1, 2) = 2, therefore, an inconsistency exists between rule 1 and rule 2.

6.5 Circular Rule Sets

The test for circular rule sets is completely different from the other tests as it does not use the resultant of the multiplication of the matrix with its' transpose, but utilizes a matrix which has been processed by substituting any proposition defined with the rule-base by its' constituent parts.

Once this matrix has been produced, the test procedure merely needs to look at the heads of the rules. If any of them is greater than one, it is involved in a circular rule set.

Circularity represents an urgent problem in backward chaining systems. When a knowledge-based system (KBS) contain a set of rules such that a non-terminating loop can occur when the rules are fired, these rules are a circular rule set. For example, consider the following circular rule set:

$$\begin{array}{l} a \rightarrow b \\ b \rightarrow a \end{array}$$

In this example, when the rule-base is fired, a implies b which in turn implies a which implies b and so on in a non-terminating loop. Two types of circular rules sets are categorized in reference [3]. Firstly, we can have a direct cycle where the rule calls itself i.e. a $d \rightarrow d$. Secondly, there is the indirect cycle which is essentially the same as a direct cycle, but the consequent and antecedent that are the same are divided by a number of steps. The example above is an indirect cycle. We remove the intervening steps on any indirect cycle. This will reduce it to a direct cycle and it will then be easily detected.

6.6 Auxiliary Rule Test

An auxiliary rule may be defined as the situation when there is a rule, which subsumes two or more existing rules in the rule-base. This can lead to inefficiencies and can make the search for other types of error of anomaly more difficult. An example of an auxiliary rule is shown below:

$$\begin{array}{l} a \rightarrow b \\ b \rightarrow c \end{array}$$

It is possible to replace these two rules by the rule $a \rightarrow c$, which effectively subsumes the two previous rules. As we see the task of simplifying indirect cycles is in effect similar to that of removing auxiliary rule sets.

We can therefore think of indirect cycles as direct cycles separated by one or a number of auxiliary rules. As it is easy to define a test for a direct cycle, then the only requirement to test for a circular rule set is that it be free of auxiliary rules. We can eliminate these rules by substituting in the left-hand side of a rule. For example, consider the following rule-bases:

Rule 1: $b\ c \rightarrow a$
Rule 2: $a\ d \rightarrow e$
Rule 3: $e f \rightarrow g$

If we now substitute b and c for a in the second rule then we get the following rule-bases:

Rule 1: $b\ c \rightarrow a$
Rule 2: $b\ c\ d \rightarrow e$
Rule 3: $e f \rightarrow g$

Rule 1 has been superceded and is now in fact redundant. If we now do the same for *e* in rule 3 then we get.

Rule 1: $bc \rightarrow a$
Rule 2: $b\ c\ d \rightarrow e$
Rule 2: $b\ c\ d f \rightarrow g$

Now, Rule 3 contains the simplified rules with rule 1 and rule 2 being redundant. It can be seen in this rule that no circularity exists. However, if we substitute *b* for *g,* the rule-bases would be as follows:

Rule 1: $bc \rightarrow a$
Rule 2: $b\ c\ d \rightarrow e$
Rule 2: $b\ c\ d f \rightarrow b$

We can see that a circular rule set exists as rule 3 now has *b* on both sides of the implied sign. The other rules are still redundant as they played the same role as in the original rule-bases. This role can be simplified by ensuring that the rules that are left are as simple as possible. Any propositions occurring twice on one side of the rule are replaced with a single proposition as anything ended with itself gives the original value. If this is done in a "correct" rule-bases, there will only be one of each proposition in any rule. There will only be more than one if the two propositions occur on different sides of the implied sign and are therefore a part of a circular rule set.

When using matrices, we must first also construct an array, which shows us those propositions that occur on the right hand side of the rule. For example, using the previous example would give us (*a, e, g*). Below is the algorithm for constructing this array.

```
Input    :  symbol-table, proposition
Output   :  matrix
Method   :
            Read entry in symbol-table
            Column : = 0
            While match not found
            {

               if proposition = symbol-table-entry
               {
                  matrix(row, column)
               }
               increment column
               read next entry
            }
```

However, there is one refinement to this process in each case were a row is added. It is best to set the position, which we are substituting for, to zero as this has been replaced. This means that when the process is finished, we examined those propositions corresponding to those held in the array. The number will only be greater than zero, if a circular rule set exists and runs through that rule. As an example, let us use the circular rule set:

Rule 1: $b\ c \to a$
Rule 2: $a\ d \to e$
Rule 3: $e f \to b$

Giving us the following matrix and array containing the positions of the right-hand side propositions:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Now, if we perform the method on each rule in turn, we get the matrices as follows from left to right:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

As can be seen, the first two rules fail the criteria set and so a circular rule set does exist within these two rules.

7. An Example

In this section, a comprehensive example is given to show how matrix verification can be applied when various types of anomalies are included at the same time. This example is as follows:

Rule 1: $a \to b$
Rule 2: $d \to e$
Rule 3: $d \to e$
Rule 4: $df \to e$
Rule 5: $df \to g$
Rule 6: $g \to f$

Which gives us the following:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 & 1 & 0 \\ 0 & 2 & 2 & 2 & 1 & 0 \\ 0 & 2 & 2 & 3 & 2 & 1 \\ 0 & 1 & 1 & 2 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \end{pmatrix} = R$$

We can apply the previous error detection procedures to find the rules that have anomalies. From the resultant matrix, we can see the following:

1. The first rule does not share a proposition with any other rule, and the entries on its' rows and columns are zero except on the leading diagonal. Therefore, the first rule is redundant and it is missing link with other rules.
2. If the duplication rule described in section 6.3 is applied to the resultant matrix, then *R(2,2) - R(3, 3) = R(2, 3).* Therefore, rule 3 is a duplicate of rule 2.
3. If the subsumption rule described in section 6.3 is applied to the resultant matrix, then (*R(4, 4) > R(2,2), and R(2,4) = R(2,2)*) and (*R(4, 4) > R(3, 3), and R(3, 4) = R(3,3)*). Therefore, rule 4 subsumes rule 2 and it also subsumes rule 3.
4. We can see from rules 4 and 5 that the number of antecedent propositions between these rules is equal 2. If the inconsistency rule described in section 6.4 is applied to the resultant matrix, we can see than *R(4,5) = 2.* Therefore, an inconsistency exists between rules 4 and 5.

By applying the circular rule sets described in section 6.5 and the auxiliary rule described in region 6.6 to the rules 5 and 6, we obtain the following matrices from left to right:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 2 & 0 \end{pmatrix}$$

As can be seen, the second line of the second matrix fail the criteria set, and so a circular rule set does exist within rules 5 and 6.

8. The Output

The output collected from all of the tests, and will be assembled by the output routines and placed in a table. This table contains the numbered rules down one side and the tests involved across the top. The format of each test will be as follows:

(i) Redundancy and Missing links will put beside each rule the corresponding finding from redundant, leaf or root, or body. This will be contained within an array of type string.

(ii) Duplication, Subsumption and Inconsistency will list the numbers of the rules, which are duplicated, subsumed or inconsistent. These results will be contained in a two-dimensional array corresponding to the number of the rule and containing the numbers of the failed rules.

(iii) Those rules taking part in circular rule sets and auxiliary rule sets will be listed at the bottom of the table with an explanatory sentence. This will be presented in a simple array of integers.

The data will be presented to the procedure in arrays with the entries for the places in the table being produced in turn. The results will be read into a text file to allow the easy retention and the obtaining of hard copies. The output is intended however to be easily read and assimilated in a coherent manner.

9. The Efficiency

The system consists of four main modules: a loading routine, matrix manipulation routine, testing routine, and the output routine as described in Figure 1. Each module has its efficiency. The modules have been implemented, and the code is written in *C* language. The version of *C* used is ANSI. The reason for this is because it is a defined standard and most *C* compilers have an ANSI switch. The efficiency of each module and ultimately the system as a whole can be measured in terms of number of rules and number of propositions. The worst case for the system as a whole is always the same, as all of the processes must cover the whole matrix. The system as a whole therefore takes time, the number of rules times the number of possible propositions.

10. Related Systems

A variety of methods have previously been described in the literature for detecting errors and anomalies in rule-bases[1-12]. Three common types of tool can be classified: coherence checkers (*e.g.* RCP[6], CHECK[7], EVA[8]), completeness checkers (*e.g.* PEIRS[9], MCRDR[10], COVER[3]), correctness checkers (*e.g.* SYCOJET[11], SACOO[11], Bias[12]).

RCP[6] checks for a number of anomalies using decision table techniques and were used successfully on the ONCOCIN system, an EMYCIN-like KBS (for oncology) that used attribute-value rules. Out of RCP[6] work came CHECK[7] which is an automated rule verifier operating on a knowledge-based system developed on the LES shell. CHECK[7] is also based on ideas about decision tables. This employs a combination of goal driven and data driven rules and allows rules to contain variables that may be instantiated at run-time. EVA (Expert systems validation Associate)[8] was, in turn, developed out of CHECK[7]. The long-range goals of the EVA[8] project are to build an integrated set of generic tools to “validate” any KBS written in any expert system shell. EVA[8] is written in PROLOG and consists of a wide range of validation tools that enables the user to check for a number of anomalies including those identified in this paper.

In PEIRS[9], a KBS was built by applying Ripple-down Rules (RDR) algorithm to the entire data set. A manual RDR KBS was then built by incrementally working through the data. MCRDR[10] uses an *n-ary* tree rather than a binary tree. A rule can be added in a variety of locations depending on the difference list conditions chosen. COVER[3] was shown to detect genuine and potentially errors by using generate-and-test algorithm in two ways. The first is be generating only those environments that contain certain combinations of data items. The second is by generating small environment, and then generating larger ones.

SACCO[11] assumes that the knowledge base is split into three parts: a factual part describing the domain, a factual part describing the problem that the KBS has to solve, and a deductive part describing the expertise required to solve problems. It generates consistent fact bases that lead to the deduction of inconsistencies. SYCOJET[11] consists of three modules: a test data generation module, a test execution module, and run-time tracing module. The test data generation module has two parts: a label calculator, and a value generator. The label calculator is based on a backward chaining process from output to input attributes. The value generator checks if the value proposed verify the conjunction of the predicates associated with the environment. The test execution module supplies the test data values to the attributes of the object. The tracing module provides output of the form: initial test data, the set of rules fired during the run, and a set of deduced facts. Bios[12] was developed out of SACCO[11] and SYCOJET[11]. The method of this paper use correctness and coherence checkers, and therefore can be considered as correctness and coherence checking method.

These systems demonstrate another approach to the problem, however, some of these systems are complex. The matrix method used here is simple and highly effective tool in the implementation of rule-bases and knowledge based systems in general.

11. Conclusion and Future Work

In this paper, the use of matrices in the verification of rule-bases has been investigated. A method, has been demonstrated to be effective, and it has been implemented as a program. The tests that have been implemented do seem to act in the fashion anticipated and do work effectively in the finding of errors and anomalies. The project of this paper has proved the practicality of automatic computer based errors detection using matrices. The system does provide a simple and mathematically elegant method of verifying rule-bases.

The work can continue on the project of this paper is of course to implement tests for those errors and anomalies that are not mentioned. The anomalies that have not been handled in this paper include compound subsumption, compound inconsistency, irrelevancy, dead end rules (missing leaves), incompleteness, and soundness. Some of these anomalies appear to be consequences of some of the errors that have been handled in this paper. For example, a missing link can lead to a dead end rule.

References

[1] **Guida, G.** and **Tasso, C.,** *Design and Development of Knowledge-Based System,* John Wiley & Son (1994).

[2] **Preece, A.D., Shinghal, R.** and **Batarekh, A., "**Principles and Practice in Verifying Rule-Based Systems", *Knowledge Engineering Review,* **7:** 115-141 (1992).

[3] **Preece, A.,** "A New Approach to Detecting Missing Knowledge in Expert System Rule Bases", *Int. J. Human-Computer Studies,* **38:** 661-688 (1993).

[4] **Preece, A.D.** and **Shinghal, R.,** "Practical Approach to Knowledge Based Verification", In: **M. Trivedi, (**Ed.), *Applications of Artificial Intelligence,* Orlando, FL, USA, 608-619 (1991).

[5] **Preece, A.,** "Towards a Methodology for Evaluating Expert System", *Expert Systems,* **7:** 215-223 (1990).

[6] **Suwa, M., Scott, A.C.** and **Shortliffe, E.H.,** "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System", *AI Magazine,* **3**(4): 16-21 (1982).

[7] **Nguyen, T.A., Perkins, W.A., Laffey, T.J.** and **Pecora, D.,** "Checking an Expert Systems Knowledge Base for Consistency and Completeness", *Proceedings of the 9th International Joint Conference on Artificial Intelligence,* Los Angeles, CA, USA, vol. **1:** 375-378 (1985).

[8] **Chang, C., Combs, J.** and **Stachowitz, R.,** "A Report on the Expert Systems Validation Associate (EVA)", *Expert System with Applications,* **1:** 217-230 (1990).

[9] **Kang, B., Gambetta, W.** and **Compton, P.,** "Verification and Validation with Ripple-Down Rules", *Int. J. Human-Computer Studies,* **44:** 257-269 (1996).

[10] **Kang, B.H., Gambetta, W.** and **Compton, P.,** "Validation and Verification with Ripple Down Rules", *AAAI Workshop on Validation and Verification,* Seattle, WA, USA, 64-69 (1994).

[11] **Ayel, M.** and **Vignollet, L.,** "SYCOJET and SACCO, Two Tools for Verifying Expert System", *International Journal of Expert Systems: Research and Applications,* **6:** 357-382 (1993).

[12] **Preece, A.D., Talbot, S.** and **Vignollet, L.,** "Evaluation of Verification Tools for Knowledge-Based Systems", *Int. J. Human-Computer Studies,* **47:** 629-658 (1997).

طريقـة استخـدام المصفوفـات للتحقق من صحـة أنظمـة وقواعـد المعرفـة

سعود محمد مغربي

قسم العلوم الرياضية ، كلية العلوم التطبيقية ، جامعة أم القري
مكة المكرمة – المملكة العربية السعودية

المستخلص . يوصف هذا البحث طريقة تستعمل المصفوفات للتحقق من صحة أنظمة وقواعد المعرفة الحاسوبية بشكل عام . تقوم هذه الطريقة بتمثيل أي قاعدة معرفة كمصفوفة ، وبعد ذلك تستعمل التحويلات القياسية في المصفوفات من أجل تحديد موضع الأخطاء والشواذ الموجودة في القاعدة مثل : التكرار ، الازدواجية ، التناقض ، الحلقات المفقودة ، الاستدارة ، فقدان الاتصال بين أجزاء القاعدة ، الأجزاء الزائدة في القاعدة .

طريقة هذا البحث فعالة وبسيطة . وتحتوي طريقة البحث نظام صمم لتأسيس سياق البيئة المشكلة ، ويحتوي هذا النظام على أربع روتينات رئيسة :

١) روتين إدخال قاعدة المعرفة للحاسب الآلي .
٢) روتين معالجة المصفوفات .
٣) روتين اختيار أجزاء قاعدة المعرفة .
٤) روتين إظهار النتائج .

لقد برهنت طريقة البحث على إمكانية تطبيق نظام أتوماتيكي حاسوبي باستعمال المصفوفات للكشف عن الأخطاء في قواعد المعرفة . وينتهي البحث بمناقشة النتائج المستخلصة من البحث . بالإضافة إلى القيام بمقارنة نظام هذا البحث مع أنظمة البحوث المشابهة له .