An Attribute-rule Dependency Matrix Method and its Java Implementation for Rule-based Expert Systems Verification

M. Ayman Al Ahmar
Deputy Dean, College of Information Technology
Ajman University of Science and Technology
United Arab Emirates

ABSTRACT
Verification of knowledge bases is an important aspect of the development procedure of rule-based expert systems. The objective of verification is to assure producing a successful intelligent computer system that reaches correct recommendations. This research introduces an attribute-rule dependency matrix verification method and its associated Java implementation program. The method can help knowledge engineers and domain experts in the automated verification process of rule-based knowledge bases for both consistency and completeness. The method can also help in the documentation of expert systems' facts and If-Then rules. A wide variety of knowledge bases has been successfully debugged and analyzed using the introduced verification method.

General Terms
Rule-Based Expert System, Knowledge Base Verification

Keywords

1. INTRODUCTION
Expert systems (ES) provide a powerful and flexible technology for solving a variety of problems, and are widely used nowadays in many academic, industrial, and commercial applications. Expert systems can be defined as "programs that attempt to emulate the behavior of human experts, usually confined to a specific field" [1]. Some authors do not support ES definitions implying that an ES is just a computer program. J. P. Ignizio, as an example, defines ES as "a model and associated procedure that exhibits, within a specific domain, a degree of expertise in problem solving that is comparable to that of a human expert" [2]. ES are usually developed using development packages known as ES shells. The most essential component of an ES is the knowledge base (KB) because it is the place where the domain's expert knowledge is modeled, saved, and processed. A successful knowledge base is the main source of obtaining expert-level high quality solutions from an ES. This fact makes the verification of the KB a highly important ES construction task. Verification is the process of assuring consistency, completeness, and syntax correctness of a knowledge base. Note that the correctness of the syntax is a task embedded within the rule editor of the most popular ES shells; therefore, the main verification tests are the tests for consistency and completeness which are usually not comprehensively covered within ES shells. Literature survey reveals many researches related to the topic of ES verification [2][7][27]. The objective of this research is to extend previously proposed rule-based ES verification methods with a method formulated in a structure that makes it ready to be implemented smoothly in high level programming languages like Java and C++. In addition (as will be shown in this paper), the proposed method's structure and Java implementation enables knowledge engineers to easily export If-Then rules from ES shells into the text file format required by the implementation program. The method is referred to as 'attribute-rule dependency matrix' and will be described with its Java implementation in Sections 5 and 6 respectively. Section 2 lists various knowledge representation schemes and Section 3 outlines rule-based ES. ES verification is addressed in Section 4 and concluding remarks and future work are presented in Section 7.

2. KNOWLEDGE REPRESENTATION IN ES
Knowledge to expert systems is as fuel to cars, and a major issue in ES development is how to represent knowledge within the digital computer. Several knowledge representation methods have been proposed including: Rule-based systems, semantic networks, frames, logic statements, and neural networks. The primary focus in this research will be on rule-based systems for knowledge representation which is a widely applied knowledge representation scheme for the development of expert systems as will be addressed in the next Section.

3. RULE-BASED EXPERT SYSTEMS (RBES)
One of the most popular methods of representing knowledge within expert systems is knowledge representation through the use of rules, or rule-based expert systems. Alternatively, such rules are called production rules, or If-Then rules. Advantages of rules include:

1. Rules have a simple syntax and are easy to understand.
2. Rules are easy to learn since they represent a natural mode of knowledge representation.
3. Rule bases can be easily modified by adding, deleting, and revising rules.
4. Rule chaining is easy to trace and debug which also results in good explanation facilities.
If-Then rules are composed of two parts: If part and Then part. The If part is called the premise or condition, and the Then part is called the conclusion or action part. Example If-Then rules are:

Rule 1:
If the instructor’s highest degree is Ph.D.
Then the instructor can supervise M.S. theses

Rule 2:
If the student's CGPA is more than 3.49 And total earned hours are more than 29
Then transfer status is accepted And new IT major title is IS

It is noted that production rules contain object-attribute-value (OAV) triplets. For example there is one OAV triplet implied in the If part of Rule 1 above as following:

Object: instructor
Attribute: Highest degree
Value: Ph.D.

and another OAV triplet in the Then part as following:
Object: instructor
Attribute: Supervision of M.S. Theses status
Value: True

In a similar logic for Rule 2, there are two OAV triplets contained in the premise portion and two OAV triplets contained in the conclusion portion.

The values listed in a rule’s If part are compared (or tested) with any provided value, whereas, the values appearing in the Then part are assigned to the mentioned attributes.

4. VERIFICATION OF RBES

It is important to note that “knowledge base problems can only be detected if the rule syntax is restrictive enough to allow one to examine two rules and determine whether situations exist in which both can succeed and whether the results of applying the two rules are the same, conflicting, or unrelated” [26]. If unrestricted rule syntax is allowed, it is difficult or impossible to propose and implement verification algorithms. For the present version of the proposed method in this paper, we shall assume that If-Then rules are deterministic (that is, we assume that there is no uncertainty factors attached to rules) and all attributes are single valued (no multi-valued attributes). We also assume that all rules are backward chaining rules grouped into rule-sets and that all rule premise clauses are conjunctive (connected by And operator). In fact it is a common practice in rule-based ES design to avoid disjunctive premise clauses (connected by Or operator) by replacing them with multiple rules. For example the rule:

If A = M Or B = N Then C = P

can be replaced by the two equivalent rules:

If A = M Then C = P
If B = N Then C = P

For generality (i.e., not referring to a particular ES domain) rules in this paper will be written in a symbolic format as shown above (i.e., A, B, C, M, P, etc.).

In the verification of rule bases there are mainly two types of checks: Checks for consistency and checks for completeness. Consistency and completeness conditions can be used for the systematic examination of integrity and logic of the knowledge base. Much of the material that immediately follows is based on Ref. [2] and Ref. [26] and describes the types of consistency and completeness included in the current version of the proposed method.

4.1 Consistency

The following four types of inconsistency are currently considered.

4.1.1 Conflicting Rules

Two rules that succeed under identical conditions and give conflicting conclusions are said to be conflicting rules. Consider the two rules below. For the same set of If clauses, different conclusions are given (C = P in Rule 1 and C = Q in Rule 2).

R1: If A = M And B = N
Then C = P
R2: If A = M And B = N
Then C = Q

4.1.2 Redundant Rules

Consider Rule 1 and Rule 2 given below. Both rules have equivalent If parts while the conclusion of Rule 1 is a subset of the conclusions of Rule 2. This indicates that Rule 1 is made redundant by Rule 2 and may be removed from the rule-set under consideration. Since ‘any set is a subset of itself’, this case is also applicable for two rules having the same If parts and the same Then parts (i.e., two identical rules, so one of them is redundant and should be removed).

R1: If A = M And B = N
Then C = P
R2: If A = M And B = N
Then C = P And D = Q

4.1.3 Unnecessary IF Conditions

Two rules contain unnecessary IF conditions if they have the same conclusions, an IF condition in one rule is in conflict with an IF condition in the other rule, and all other IF conditions in the two rules are the same. Consider the rules shown below.

R1: If A = M And B = N
Then C = P
R2: If A= M And B = Not N

18
Then C = P

Note that the same conclusion is reached regardless of the value of the attribute B. Thus, the premise clause containing B is unnecessary, and may be removed to form an efficient rule. Specifically, Rules 1 and 2 should be replaced with a single rule:

If A = M Then C = P.

### 4.1.4 Subsumed Rules

One rule is subsumed by another if both have the same conclusions, but one of them contains additional if conditions. Such additional conditions are thus unnecessary. In the rules below, Rule 1 is subsumed by Rule 2 (so Rule 1 may be removed from the knowledge base).

R1: If A = M And B = N
Then C = P
R2: If A = M
Then C = P

### 4.2 Completeness

There are five types of incompleteness considered currently in the proposed method as following:

#### 4.2.1 Unreferenced Attribute Values

The case of unreferenced attribute values occurs if one or more legal values of an attribute (which is not a final goal) are not covered by any rule's premise clauses. For example, consider an attribute 'Degree' with three legal values: Ph.D., M.S., and B.S. and assume that the only related rule premises are "If Degree is Ph.D." and "If Degree is B.S.". Thus, the value M.S. does not appear in any premise clause and is 'unreferenced attribute value'. The case of unreferenced attribute value indicates that either a rule is missing, or a premise clause is missing, or the missing value should be removed from the set of legal values.

#### 4.2.2 Illegal Attribute Values

Consider again the 'Degree' attribute given above and consider the following rule premise: "If Degree is B.A." The value B.A. is an illegal attribute value since it is not contained in the set of legal values of the attribute 'Degree' and the rule needs to be corrected (or this might be an indication that B.A. should be added to the list of legal values).

#### 4.2.3 Unachievable Intermediate Conclusions

An intermediate conclusion is unachievable if it does not exist in the premise of any rule. Consider the following rule: If A = M Then C = P. The intermediate conclusion C = P would be considered unachievable if the clause C = P does not appear in the premise of any other rule. Such a rule is either incorrectly formulated or unnecessary extra rule that may be removed from the rule-set under testing.

#### 4.2.4 Unachievable Goals

A goal (or a final conclusion) is unachievable whenever there is no query for the premise of the goal and the premise cannot be deduced from any other rule (i.e., it does not appear in the Then part of another rule).

### 4.2.5 Unachievable Premises

Consider the following rule-set.

R1: If A = M Then B = N
R2: If B = N Then C = P
R3: If D = Q Then Z = true (Z is the goal)

Note that the premise of Rule 3 is unachievable if there is no query associated with it. In general, a rule premise is considered unachievable if there is no query for the premise and the premise cannot be concluded from any other rule. Note that the above example shows also an unachievable goal (the goal Z is unachievable).

## 5. ATTRIBUTE-RULE DEPENDANCY MATRIX METHOD

Table 1 below displays a sample attribute-rule dependency matrix. The matrix may be most easily explained by a simple example. The five rules listed below will be used to illustrate the procedure. We shall assume that user queries for this rule-set are associated with the attributes: W and Y. The associated attribute-rule dependency matrix is presented in Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>W = high</th>
<th>W = moderate</th>
<th>W = low</th>
<th>Y = low</th>
<th>Y = high</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>X = satisfactory</td>
<td>XX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>X = satisfactory</td>
<td>XX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>X = unsatisfactory</td>
<td>XX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>X = satisfactory</td>
<td>XX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>X = unsatisfactory</td>
<td>XX</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.1 Unachievable Premises

Consider the following rule-set.

R1: If A = M Then B = N
R2: If B = N Then C = P
R3: If D = Q Then Z = true (Z is the goal)

Note that the premise of Rule 3 is unachievable if there is no query associated with it. In general, a rule premise is considered unachievable if there is no query for the premise and the premise cannot be concluded from any other rule. Note that the above example shows also an unachievable goal (the goal Z is unachievable).

### 5.2 Completeness

There are five types of incompleteness considered currently in the proposed method as following:

#### 5.2.1 Unreferenced Attribute Values

The case of unreferenced attribute values occurs if one or more legal values of an attribute (which is not a final goal) are not covered by any rule's premise clauses. For example, consider an attribute 'Degree' with three legal values: Ph.D., M.S., and B.S. and assume that the only related rule premises are "If Degree is Ph.D." and "If Degree is B.S.". Thus, the value M.S. does not appear in any premise clause and is 'unreferenced attribute value'. The case of unreferenced attribute value indicates that either a rule is missing, or a premise clause is missing, or the missing value should be removed from the set of legal values.

#### 5.2.2 Illegal Attribute Values

Consider again the 'Degree' attribute given above and consider the following rule premise: "If Degree is B.A." The value B.A. is an illegal attribute value since it is not contained in the set of legal values of the attribute 'Degree' and the rule needs to be corrected (or this might be an indication that B.A. should be added to the list of legal values).

#### 5.2.3 Unachievable Intermediate Conclusions

An intermediate conclusion is unachievable if it does not exist in the premise of any rule. Consider the following rule: If A = M Then C = P. The intermediate conclusion C = P would be considered unachievable if the clause C = P does not appear in the premise of any other rule. Such a rule is either incorrectly formulated or unnecessary extra rule that may be removed from the rule-set under testing.

#### 5.2.4 Unachievable Goals

A goal (or a final conclusion) is unachievable whenever there is no query for the premise of the goal and the premise cannot be deduced from any other rule (i.e., it does not appear in the Then part of another rule).

### 5.3 Completeness}

#### 5.3.1 Unreferenced Attribute Values

The case of unreferenced attribute values occurs if one or more legal values of an attribute (which is not a final goal) are not covered by any rule's premise clauses. For example, consider an attribute 'Degree' with three legal values: Ph.D., M.S., and B.S. and assume that the only related rule premises are "If Degree is Ph.D." and "If Degree is B.S.". Thus, the value M.S. does not appear in any premise clause and is 'unreferenced attribute value'. The case of unreferenced attribute value indicates that either a rule is missing, or a premise clause is missing, or the missing value should be removed from the set of legal values.

#### 5.3.2 Illegal Attribute Values

Consider again the 'Degree' attribute given above and consider the following rule premise: "If Degree is B.A." The value B.A. is an illegal attribute value since it is not contained in the set of legal values of the attribute 'Degree' and the rule needs to be corrected (or this might be an indication that B.A. should be added to the list of legal values).

#### 5.3.3 Unachievable Intermediate Conclusions

An intermediate conclusion is unachievable if it does not exist in the premise of any rule. Consider the following rule: If A = M Then C = P. The intermediate conclusion C = P would be considered unachievable if the clause C = P does not appear in the premise of any other rule. Such a rule is either incorrectly formulated or unnecessary extra rule that may be removed from the rule-set under testing.

#### 5.3.4 Unachievable Goals

A goal (or a final conclusion) is unachievable whenever there is no query for the premise of the goal and the premise cannot be deduced from any other rule (i.e., it does not appear in the Then part of another rule).
3. An 'if' appears at the intersection of an attribute-row and a rule-column if that attribute (with its specified value) appears in the If part of the rule.

4. A 'then' appears at the intersection of an attribute-row and a rule-column if that attribute (with its specified value) is an intermediate conclusion that appears in the conclusion part of the rule.

5. A 'then-goal' appears at the intersection of an attribute-row and a rule-column if that attribute (with its specified value) is a goal that appears in the conclusion of the rule.

6. Multiple 'yes' appear at the intersection of an attribute-row (with all its legal values) and the query-column if there is a user query for that attribute. Note that multiple 'yes' appear here since the user can select any legal value as the answer for the query.

7. 'If-rows' for a rule are the rows that contain all the premise clauses of that rule, whereas its 'then-rows' are the rows that contain rule's conclusion clauses (including goals).

As noted in the matrix description above, the proposed method uses explicit 'if', 'then', 'then-goal', and 'yes' in the matrix cells (unlike many previous researches that use just an asterisk (*) or other symbols to indicate dependencies [2], [25], [26], [27]). Advantages of using the strings 'if', 'then', 'then-goal', and 'yes' include: 1. It allows enhanced logic using the advanced string processing capabilities in modern high-level programming languages, 2. Differentiating between intermediate (then) and final conclusions (then-goal), and 3. The resulted matrix is a straightforward, easy to be included in a program, and more readable documentation for the knowledge base.

The rules for the detection of inconsistency and incompleteness from the attribute-rule dependency matrix may be summarized as following and related examples are illustrated in Tables 2 and 3 which show only partial (incomplete) matrices with example rules different than the rules presented in Table 1.

1. If two rules have identical if-rows but contradicting then-rows, then the rules are conflicting. Example: R1 and R2 in Table 2.

2. If two rules have identical if-rows, and the then-rows of one of them is a subset of the then-rows of the other, then the 'subset' rule is redundant. Example: R1 and R3 in Table 2.

3. If two rules have identical then-rows and their if-rows are the same except a row related to an attribute having conflicting values in each rule, then that If condition is unnecessary. Example: R4 and R5 in Table 2.

4. If two rules have the same then-rows, and coincide with common if-rows, but one of them has additional if-rows, then the rule with the additional if-rows is subsumed by the other rule and may be removed from the rule-set. Example: R5 and R6 in Table 2.

5. If there are no 'if's (or 'then-goal's in case of a goal) in any non-goal attribute-row, the associated attribute value is unreferenced (or that might be an indication of the existence of rules with illegal attribute values, see next point). Example: The value 'moderate' of the attribute W in Table 3.

6. If an attribute value in a rule cannot find a matching value in any attribute-row, then the value is illegal. Example: The premise: If W = large (large is not a legal value of the attribute W).

7. In any row containing 'then' (for an intermediate conclusion), if there is no 'if on the same row, then the associated rule's intermediate conclusion is unachievable. Example: R1 in Table 3.

8. If there is a 'then-goal' in any rule-column, then every corresponding rule's 'if' should have (in its same row) either 'yes' or 'then', otherwise the associated goal is unachievable. Example: R1 in Table 3.

9. In any row containing 'if', if there is no 'yes' or 'then' on the same row, then the associated rule's premise is unachievable. Example: R2 in Table 3.

**Table 1. Sample attribute-rule dependency matrix**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>high</td>
<td>if</td>
<td></td>
<td>if</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>W</td>
<td>moderate</td>
<td>if</td>
<td></td>
<td></td>
<td>if</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>W</td>
<td>low</td>
<td>if</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>X</td>
<td>satisfactory</td>
<td>then</td>
<td></td>
<td>then</td>
<td>if</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>unsatisfactory</td>
<td>then</td>
<td></td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&gt; 1000</td>
<td>if</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&lt; 1000</td>
<td>if</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>true</td>
<td></td>
<td>then-goal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>false</td>
<td></td>
<td></td>
<td>then-goal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. Sample partial matrix showing various cases of inconsistency among example rules**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>high</td>
<td>if</td>
<td>if</td>
<td>if</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>moderate</td>
<td>if</td>
<td>if</td>
<td>if</td>
<td>if</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>low</td>
<td>if</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>satisfactory</td>
<td>then</td>
<td>then</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>unsatisfactory</td>
<td>then</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&gt; 1000</td>
<td>if</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&lt; 1000</td>
<td>then</td>
<td>then</td>
<td>then</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>true</td>
<td></td>
<td>then-goal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3. Sample partial matrix showing various cases of incompleteness among example rules**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>high</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>moderate</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>low</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>satisfactory</td>
<td>then</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>unsatisfactory</td>
<td>then</td>
<td>if</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&gt; 1000</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>&lt; 1000</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>true</td>
<td></td>
<td>then-goal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>false</td>
<td></td>
<td>then-goal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20
6. THE JAVA IMPLEMENTATION PROGRAM AND ITS TESTING

In order to automate the verification of rule-sets using the method introduced in this paper, a Java program is developed and tested as will be described here. Java programming language was selected because it is ranked at the top of the most popular programming languages according to many references [28]-[29]. The TIOBE programming community index ranks Java as the most popular language since 2006. The chart shown in Figure 1 below is based on TIOBE index for December 2011 [28]. Java is also a highly platform independent programming language.

![Most Popular Programming Languages](image)

**Fig. 1: The 10 Most Popular Programming Languages [28]**

The program’s high level algorithm steps are as following:

1. **Matrix formulation:** Formulating the attribute-rule dependency matrix by reading the attributes and the rules from external files and prompting the user for some required data and parameters.

2. **Verification:** Verification of the consistency and completeness of the rules by checking the matrix as described in Sec. 5 above.

3. **Report generation:** Generating the verification report which is displayed on the screen and also saved in a text file.

Step 1: The matrix formulation step is performed by reading text files that contain the required data in addition to prompting the user for some parameters. Figures 2 and 3 below show the format of sample text files that can be read (using string processing) by the Java program and used to generate the required verification matrix implemented in Java as a two-dimensional string array.

**Rules**

```java
if ( W == high )
Then ( X = satisfactory )
if ( W == moderate & Y <= 1000 )
Then ( X = satisfactory )
if ( W == low & Y <= 1000 )
Then ( X = unsatisfactory )
if ( X == satisfactory )
Then ( Z = true )
if ( X == unsatisfactory )
Then ( Z = false )
```

**Fig. 2:** A sample input text file containing an example IF-Then rule-set

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>W high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W moderate</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>Y &lt;= 1000</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>X satisfactory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X unsatisfactory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 3:** A sample input text file containing the attributes and their legal values

As seen from Figure 2 the text file containing the If-Then rules uses syntax similar to the well known C-based languages’ syntax (Java, C++, etc.) and can be easily generated by many ES shells. This feature makes the method a general purpose verification tool and this feature is very important since knowledge engineers can automatically generate the required If-Then text file using the appropriate option within the used ES shell. Note that even if the ES shell does not generate the If-Then text file exactly as shown in Figure 2, the generated file can be easily converted to the required format using find/replace features that are available in most word processing applications (changing all the occurrences of ‘And’ into ‘&&’ as an example). Note that rule names are read from a different ‘parallel’ text file. Figure 3 shows the required input text file that lists the attributes with their legal values. The following is a sample Java code that utilizes this text file (named: attributesValues.txt) in order to fill the first and second columns in the attribute-rule dependency matrix (a two-dimensional string array named matrix in the developed Java program).

```
Scanner inFile2 = new Scanner (new FileReader("attributesValues.txt"));
int i=0; k=attributesValues.length; ++
for (i=1; i<legalValues; i++)
|
| matrix[i][0] = attributes[j];
| matrix[i][1] = inFile2.next();
| if (i=k) |
| j++; if (j==attributesValues.length) break;
| k = k + attributesValues[j];
| }
| inFile2.close();
```

At the end of Step 1 the generated matrix will be similar to the matrix shown in Table 1. This feature makes the proposed method also useful in the documentation of expert systems’ attributes, attributes’ legal values, and If-Then rules.
Step 2: The verification step applies the verification rules described in Section 5 by comparing rules' rows and columns along with the query column in order to detect any case matching the described cases of inconsistency and incompleteness. The Java code of this step is a thorough code utilizing string processing capabilities of Java merged with repetition structures (for, while, and do-while loops) for processing the two-dimensional string array. As an illustrating sample segment of this code consider the following self-explanatory Java code:

```java
boolean unreferencedValue;
for (i=1; i<=legalValues; i++)
{
    unreferencedValue = true;
    for (j=2; j<=rules+1;j++)
    {
        if (matrix[i][j].equals("if") || matrix[i][j].equals("then"))
            unreferencedValue = false;
    }
    if (unreferencedValue)
        reportString = reportString + "\n\nUnreferenced attribute value case:\n\n\n\nThe value " + matrix[i][1] + " of the attribute " + matrix[i][0] + " is unreferenced value;"
}
```

Step 3: In this stage the verification results for the rule-set under consideration are reported as a display on the screen and also saved in a text file as a permanent reference. Figure 4 below shows a sample screen verification report. Note the use of the graphical user interface (GUI) components for input/output using the Java class JOptionPane contained in the package javax.swing.

The Java program was tested, debugged and continuously improved by the author for a long period of time during various expert systems projects and researches (e.g., [31], [32], [33]). The screen of Figure 4 is the report screen related to the verification of a rule-set within the knowledge base of the ES developed for the diagnosis of fungal diseases of date palm [33]. In that rule-set there were an unreferenced attribute value (value: FusariumMoniliforme of the attribute: FungusName) and a rule named PossibleDisease2 which was subsumed by another rule named PossibleDisease1. The rules were as following (given here in English-like syntax):

PossibleDisease1: If (StrangeColorType = Brown &
    StrangeColorPosition = ExternalSurfaceOfUnopenedSpathes)
    Then (PossibleDisease = Khamedj)
PossibleDisease2: If (StrangeColorType = Brown &
    StrangeColorPosition = ExternalSurfaceOfUnopenedSpathes &
    & FungusName = MauguiellaScatet44)
    Then (PossibleDisease = Khamedj)

The Java program (and the method behind it) still needs modifications and improvement as will be addressed in the Conclusion Section below.

7. CONCLUSION
In this research, an attribute-rule dependency matrix method and its Java implementation are described and illustrated with sample examples. The method is used in the verification of rule-based expert systems for both consistency and completeness. The use of the method in the verification of various expert systems proved that the method is successful and promising. The method and its Java implementation program can be improved in various ways. Future work includes:

- Making the method more general by releasing the assumptions listed in Section 4 (e.g., verification of rules under uncertainty by allowing the presence of confidence or uncertainty factors attached to rules).
- More types of inconsistency and incompleteness can be added to the method (e.g., circular rules).
- Integration of the method implementation in various ES shells so that the knowledge engineer can consult the method from within the ES shell being used.

8. REFERENCES


