An Integrated Rule-Base Reduction System

Kunihiko Higa Vesper Owei

Georgia Institute of Technology
School of Management
Atlanta, GA 30332-0520
Bitnet: KHIGA@GTRIOl

Abstract
Although user-friendly and rule-based expert system shells ease the implementation of expert systems, and allow for quick prototyping, their use could easily lead to ill-structured knowledgebases. They provide no methodical way to design and maintain rule-bases (RB). Thus the use of these shells does not reduce the complexity of the RB design/maintenance task. This paper addresses this problem and provides an automated RB maintenance methodology. We focus on the implementation aspect of the methodology, and formalize the RB size reduction methodology. The result of our study is an integrated rule reduction system (IRRS) which checks for and detects incompleteness and inconsistencies in RB's, and performs RB size reduction.

1. INTRODUCTION
The availability of wide ranging user-friendly and rule-based expert system shells have encouraged numerous organizations to adapt expert system technologies to their operations [3,7]. While existing rule-based shells may ease the implementation phase of expert system development, and allow for quick prototyping, their use could easily lead to ill-structured knowledgebases (KB's); they provide no methodical way to design and maintain the Rule-bases (RB's). Thus the use of those shells does not reduce the complexity of the RB design/maintenance task.

To address this problem, an automatable graph-based top-down methodology for WSR design and maintenance was developed in [4]. In this paper, we focus on the implementation aspect of that study, and formalize the rule reduction methodology developed there. The result of our study is an integrated rule reduction system (IRRS) which checks for, and detects incompleteness and inconsistencies in a rule-base system, as well as perform rule size reduction.

2. Related Works
Numerous works exist on the issue of knowledge base maintenance and enhancement. In this section, we provide a brief discussion of only a smattering of those that are more closely related to our study.

Mettrey [8] presents an evaluative study of five expert system tools: (1) The C Language Integrated Production System (CLIPS); (2) The Automated Reasoning Tool for Information Management (ART-IM); (3) Knowledge Engineering System (KES); (4) Level 5, and (5) The VAX Official Production System Version 5 (VAX OPS5). Excepting the KES Hypothesis-and-Test(KES-HT) subsystem, there is no evidence from the study that any of these systems provides any facility for enhancing the maintainability or the efficiency of a knowledge base. KES-HT comes closest to providing for knowledge base reduction through its use of a minimal covering set to describe all the possible outcomes in specialized diagnostic applications [8]. This specialization, however, also limits its suitability for more general use.
The study by Crockett and Herrera [1] deals with decision tree optimization. A technique is postulated for reducing a decision tree to an irreducible with respect to the cost criterion of decision tree preordering. The attainment of rule-base maintainability by this method is thus through code optimization. Xi Plus uses ID3 [5], which is a decision tree based methodology, for knowledge representation. Knowledge-base enhancement in this latter tool is attempted through entropy calculations which determine whether a given attribute is significant or not in a rule. Where an attribute's computed entropy is below a set threshold, the attribute is deemed insignificant, and thus dropped from the rule. Ironically, the Xi Plus approach might not support ease of future maintenance of a knowledge base. Although a decision tree might currently be entropy-significance optimum, it is possible that future expansions and evolutions of the knowledge-base might call for and require the inclusion of previously dropped attributes. Thus, the entire knowledge-base may have to be redesigned and re-optimized.

Froscher and Jacob [2] propose a methodology that, supposedly, can increase the ease of maintenance of a production knowledge-base. They employ a clustering algorithm to partition rules into groups that will enhance rule maintenance. Rule "relatedness" is used as the clustering criterion. Thus different groupings could result, depending on the semantics of each instance of relationships. It therefore follows that with the right semantic association, the clustering algorithm can be used as an adjunct to our system to partition the SOM tree into the subtrees needed by our proposed method.

3. Integrated Rule Reduction System Description

The integrated rule reduction system (IRR5) is an expert system that performs checks on data completeness and consistency, as well as rule reduction on expert systems.

3.1. Algorithm

Since the IRR5 is motivated by the work in [4], its current form requires that the rule-base be a well-structured rules system. The study in [4] uses categorization trees for knowledge representation. In the categorization tree, the root of a tree is the object of interest, while the nodes of a tree are the attributes of the object. Each path on a tree terminates at the leaf-node that represents the antecedent of a rule. Each leaf node is directly connected to a conclusion, or goal. Since we are implementing the knowledge base in [4], we decided to maintain the tree notations and terminologies in much of what follows. An IRR5 consists of the following three modules:

1) Module-1 (Completeness-check Module)

This module uses the concept of Domain Covering to check the completeness of the domain space. For the domain space of a rule-base to be complete, the following conditions must hold: (i) the set of sibling attributes for each parent attribute must cover the domain to which the siblings belong. This condition is needed to ensure that all possible conditions are accounted for, and (2) each attribute value or class in each rule must be in the domain space. This condition ensures that each attribute takes on values only in its permissible domain-space.

Let a(k) represent the k-th attribute of attribute-type a, A be the domain of attribute-type a, x be an attribute, s(a) be the set of children attribute of a(k), and S be the domain of s(a). Then, the
following algorithm forms the basis for the implementation of Module-1.

1-1] If $s(a) \in S$, then write a message of A-incompleteness.
1-2] Repeat [1-1] for the domain of each attribute-type.

II) Module-2 (Consistency-check Module)

Two rules, $R(i)$ and $R(k)$, are inconsistent if $R(i)$ and $R(k)$ are semantically equivalent (SE) or one is a subset of the other but they conclude differently. Inconsistent rules are said to be conflicting. This module checks for rule inconsistencies in the rule-base, and thus could be used to enforce rule integrity. Rule inconsistency could result from three possible causes. These are discussed below.

Case (1): $a(i) \equiv a(k)$

By this notation we mean that attribute $a(i)$ implies attribute $a(k)$, and vice versa. When this condition of mutual implication exists between $a(i)$ and $a(k)$, they are said to be semantically equivalent (SE) or equal. Semantic equivalence between $a(i)$ and $a(k)$ may be expressed more simply as: $a(i) = a(k)$

Rule-set 1: Generic Subgroup of a Rule-base

R1: IF $a(1)$ And $b(1)$, THEN $G(1)$
R2: IF $b(1)$ And $c(2)$, THEN $G(2)$

To make more clear the discussion that follows, we illustrate the concepts with Rule-set 1. Let Rule-set 1 be a subgroup of a rule-base. As shown in Rule-set 1, suppose that attributes $a(1)$ and $c(2)$ are on paths that conclude in goals $G(1)$ and $G(2)$, respectively, and that $a(1)$ and $c(2)$ are semantically equivalent. Since the corresponding attributes on the two paths, e.g., $a(1)$ and $b(1)$, and $b(1)$ and $c(2)$ are SE, but the two paths conclude differently, i.e., $G(1)$ and $G(2)$ are not SE, denoted as: $G(1) \not= G(2)$, then the two paths or, equivalently, the SE rules $R1<a1.b1>$ and $R3<b1.c2>$ are inconsistent. A corollary statement is that if the two rules $R1<a1.b1>$ and $R3<b1.c2>$ are SE, then the corresponding attributes in both rules must also be SE.

It therefore follows that if $a(i)$ in rule $R(i)$ and $a(k)$ in rule $R(k)$ are SE, but the goals $G(i)$ of $R(i)$ and $G(k)$ of $R(k)$ are not, then a possible inconsistency between $R(i)$ and $R(k)$ could result from these corresponding pair of attributes.

Case (2): $a(i) \subseteq a(k)$

In this case $a(i)$ is a subset of $a(k)$, as shown in Rule-set 2. Assume that $a(i)$ and $a(k)$ respectively lead to goals $G(i)$ and $G(k)$, in the sense described in case (1). This is portrayed as: $a(i) \rightarrow G(i)$ and $a(k) \rightarrow G(k)$.

Rule-set 2: Subset of Attributes

R3: IF $a(i)$ THEN $G(i)$
R4: IF $a(k)$ THEN $G(k)$

Since $a(i) \subseteq a(k)$, it thus follows that if $G(i) \subseteq G(k)$, no inconsistency results. If this is not the case, then $\exists x(i) \subseteq a(i)$: $x(i)$ leads to goal $G(i)$: $x(i) \rightarrow G(i)$, where $G(l) \subseteq G(i)$ but $G(l) \not= G(k)$. A possible inconsistency could result from this condition since for the same $x$ value and SE rules, two different conclusions occur. If, however, $G(i) \subseteq G(k)$ or $G(i) = G(k)$ holds, then the possibility of this inconsistency occurring does not exist. In that case, $a(i) \rightarrow G(i)$ is seen to be redundant and should therefore be dropped.

Case (3): $a(i) \not\subseteq a(k)$ and $a(i) \cap a(k) \not= \emptyset$

In this case, $a(i)$ is not a subset of $a(k)$, neither are $a(i)$ and $a(k)$ disjoint, i.e., $\exists x(i) \subseteq a(i)$: $x(i) \not\subseteq a(k)$, as shown in Rule-set 3. As in case (2), $a(i)$ leads to goal $G(i)$ and $a(k)$ leads to goal $G(k)$. If $G(i)$ is a subset of $G(k)$, i.e., $G(i) \subseteq G(k)$
or \( G(i) \supseteq G(k) \), then no inconsistencies or conflicts result from the \( a \)-type attributes. Suppose, however, that \( G(i) \nsubseteq G(k) \) or \( G(k) \nsubseteq G(i) \). Then,
\[
\exists y \in [a(i) \cap a(k)]: y \rightarrow G(i) \text{ and } y \rightarrow G(k).
\]
This implies an inconsistency of the case (2) type.

**Rule-set 3: Non-disjoint Attributes**

R5: IF \( a(k) \) THEN \( G(k) \)
R6: IF \( a(i) \) THEN \( G(i) \)
R7: IF \( a(ik) \) THEN \( G(ik) \)

\{where \( a(ik) = a(i) \cap a(k) \), and \( G(ik) = G(i) \cap G(k) \)\}

A way of resolving inconsistencies arising from case (3) type situations is suggested by the realization that \( a(i) = a(ik) + (a(i) - a(ik)) \). Since \( a(ik) \subseteq a(k) \), it follows that \( G(ik) \subseteq G(k) \) or \( G(ik) = G(k) \). Since \( (a(i) - a(ik)) \not\subseteq a(k) \), it is not necessary for \( G(i) \subseteq G(k) \) or for \( G(i) = G(k) \), i.e., \( G(i) \not\subseteq G(k) \) or \( G(i) \not\text{Eqv.} G(k) \) does not imply an inconsistency. Thus \( a(i) \) could be split into the disjoint parts \( a(ik) \) and \( (a(i) - a(ik)) \), as shown in Rule-set 4, with each disjoint concluding as follows:
\[
a(ik) \rightarrow G(ik), \text{ and } (a(i) - a(ik)) \rightarrow G(i) - G(ik)
\]

**Rule-set 4: Inconsistency Handling of non-disjoint Attributes**

R8: IF \( (a(k) - a(ik)) \) THEN \( G(k) - G(ik) \)
R9: IF \( (a(i) - a(ik)) \) THEN \( G(i) - G(ik) \)
R10: IF \( a(ik) \) THEN \( G(ik) \)

From the preceding treatments in cases (1) to (3), the following algorithm emerges for detecting conditions in a well-structured rule-base (WSRB) that could lead to inconsistent, or conflicting rules:

1. Starting from the root of the tree, identify all the immediate (or next-level) subtrees.
2. Compare in a pair-wise manner the subtree roots (these are the children of the root in (1).) for semantic equivalence:
   
   (2-1) If any pair are SE, flag the possibility of an inconsistency or conflict resulting from the pair.
   (2-2) If no pairs are SE, then no inconsistencies or conflicts can result from this level of the tree.

3. For each of the subtrees, re-iterate steps (1) and (2).

4. If all the subtrees in the tree have been accessed and processed as in (1), (2) and (3), then terminate.

While the above algorithm detects the possibility of an inconsistency between pairs of rules occurring, it does not categorically say when this pathological condition occurs. The following step, when added to the algorithm, achieves this goal:

If for rules \( R(i) \) and \( R(j) \), where \( R(i) \neq R(j) \), a possibility for inconsistency occurs at each level, then \( R(i) \) and \( R(j) \) are inconsistent.

(III) Module-3 (Rule Reduction Module)

The actual reduction of the number of rules in the RB is performed in this module. Intuition would seem to suggest that completeness and consistency checks be performed prior to rule reduction. Where this suggestion is not adhered to, and the RB is subjected to rule reduction without at first performing completeness and consistency checks, it may happen that we only succeed in producing a pathological system of rules. Incompleteness and inconsistencies in an RB are instances of pathology. We restate our intuitive sensing more formally:
Rule-size reduction of a pathological WSRB does not necessarily eliminate the pathology, but instead produces a pathological set of rules that is either incomplete or semantically confusing.

As shown in [4], rule size reduction of a WSRB is possible if, and only if, certain conditions hold. We re-state those conditions here without proof. The interested reader is referred to [4] for detailed discussion and proof of these conditions.

Condition (1): The same categorization appears at all siblings of the same branch.

Condition (2): The same categorization object reaches the same conclusion.

The common categorization can then be placed as a super class of the siblings, i.e., the positions of the current super class and the common categorization can be swapped. We formally outline the swapping algorithm below.

### Swapping Algorithm

Step 0. Start from the lowest leaf level.

Step 1. Find out if an identical categorization occurs at all siblings of the same branch and if any particular categorization object (i.e., subclass) reaches the same conclusion.

Step 2. If no such categorization object exists at the current branch, then go to Step 4.

Step 3. Swap the position of the current sibling level categorization with the common categorization that is identified at Step 1. If a common categorization reaches the same conclusion within a branch, the conclusion is independent of the siblings. In other words, the same conclusion can be reached within the branch without knowledge of the value of the sibling. Therefore, this swapping process does not alter the semantics of the branch.

Step 4. Move to the next branch which contains the equal leaf level to the current branch or to the next lowest leaf level. If no other branch is left on which to conduct this process, terminate the process. Otherwise, go to Step 1.

As seen from the rule reduction algorithm below, rule size reduction requires the use of the swapping algorithm.

### Rule Reduction Algorithm

1. Identify all the subtrees of the tree.
2. For each of the smallest subtrees, group all the rules with SE goals.
3. Use Swapping Algorithm and perform swapping for each of the subtrees in (2).
4. Repeat (1) to (3) for the next higher level of subtrees.
5. Repeat (1) to (4) until all the subtrees of the tree have been processed and no further reduction of rules is possible.

### 3.2. Redundancy and Rule-set Efficiency Measures

As an adjunct to its main function of reducing the rule size of a rule-base, Module-3 also determines the redundancy measure and the efficiency of the rule-set. These measures can be used to compare competing RB's. The terms redundancy and efficiency are used in the sense described in [4]. Redundant rules are those that do not add value to the rulebase. In reducing the rule-size of a rulebase, one must ensure that the reduction process does not degrade its semantic clarity. Clearly, the higher the level of redundancy in a rulebase,
the lower its efficiency is, and vice versa. We define these two measures thus:

Redundancy ($r$) = \[ \frac{\text{Original number of rules} - \text{New number of rules}}{\text{Original number of rules}} \]
Efficiency ($\eta$) = $1 - r$.

Redundancy and efficiency can also be expressed as percentages by multiplying $r$ and $\eta$ by 100%, respectively.

4. Implementation

The IRRS is realized with VP-Expert, an expert system shell developed by Paperback Software International. Our choice of VP-Expert is primarily predicated by its provision for interface with already existing external datafiles. Database files in dBASE II and dBASE III, Lotus 1-2-3, and selected spreadsheets (or worksheets) are easily imported into the VP-Expert work environment and directly processed by VP-Expert, without the need for any modification or tailoring. VP-Expert, however, does not possess any rule-base optimization facility that we are aware of. Indeed, the comparative analysis in [5] would seem to suggest that this is the case. We, therefore, believe that the IRRS, if incorporated into VP-Expert, would result in an enhanced-power expert system shell.

Each of the subgroups of Rule-set 5 is stored as a separate dBASE file and used by VP-Expert as called for by the already described algorithm. These subgroup files contain the rules and their corresponding goals for the respective subgroups. Another dBASE file, termed Positiondb.dbf, is used to record the relative positions of the attributes in the rules. In the rule R(a4, a3, a2, a1), attributes a1, a2, a3 and a4 occupy positions 1, 2, 3 and 4, respectively. The address field in Positiondb.dbf gives the positions of the attributes in the rules. Thus, the swapping of attributes is reflected and portrayed only in Positiondb.dbf. A third dBASE file contains all the rules, including the attributes and the goals. This latter file is termed Ruledb.dbf. An attribute value of X in any of the files means a "don't care", i.e., the particular value assigned to that attribute has no influence on the conclusion of the rule, thus we "don't care" about that attribute in the rule containing X.

When all the attributes and the conclusion of a rule are assigned the value X, the rule becomes a degenerate one; this effectively means that such a rule ceases to exist, and is therefore deleted. Thus a degenerate rule is one that is found by the algorithm to be redundant and is therefore driven into a state of deletion. We employ this scheme to indicate the deletion of redundant rules by the rule reduction module.

Rule-set 5: Example Rule-base for Implementation

Rule 1: IF personality = not-very-good And GMAT < 600 And document = complete And GPA < 3.25 THEN recom = Reject
Rule 2: IF personality = not-very-good And GMAT < 600 And document = complete And GPA > 3.25 THEN recom = Discuss
Rule 3: IF personality = not-very-good And GMAT < 600 And document = not-complete THEN recom = Pending
Rule 4: IF personality = not-very-good And GMAT > 600 And document = complete And GPA < 3.25 THEN recom = Reject
Rule 5: IF personality = not-very-good And GMAT > 600 And document = complete And GPA >= 3.25 THEN recom = Accept
Rule 6: IF personality = not-very-good
And GMAT >= 600 And document = not-complete THEN recom = Pending

Rule 7: IF personality = very-good And
GMAT < 600 And document = complete
And GPA < 3.25 THEN recom = Reject

Rule 8: IF personality = very-good And
GMAT < 600 And document = complete
And GPA >= 3.25 THEN recom = Discuss

Rule 9: IF personality = very-good And
GMAT < 600 And document = not-complete THEN recom = Pending

Rule 10: IF personality = very-good And
GMAT >= 600 And document = complete And GPA >= 3.25 THEN recom = Accept
with Graduate Assistantship

Rule 12: IF personality = very-good And
GMAT >= 600 And document = not-complete THEN recom = Pending

4.1. Results

Table 6(a) shows the initial, i.e., starting state of Ruledb.dbf. It is noted that this table is the upper symmetric half of the entire rule-base, i.e., Rule-set 5. We elected to implement the IRRS on only this half since both halves are symmetrically structured. On these tables, a GMAT value of "low" is assigned to scores that are less than 600 and "high" for those that are not less than 600. Similarly, GPA's that are less than 3.25 are assigned the value "low", otherwise they take on the value "high". On Table 6(b) is shown the initial Positiondb.dbf. The states of Ruledb.dbf and of Positiondb.dbf after the first three swaps are shown in Tables 7(a) and (b).

Redundancy $= 0.1667$
Efficiency $= 0.8333$

The effect of the rule reduction algorithm can be seen from comparing Table 6 and Table 7. The ordering of attributes in Table 6(b) is: R(personality, gmat, document, gpa). The resulting ordering in Table 7(b) is: R(gpa, personality, gmat, document). From Table 7(a), it is also seen that all the attributes and the goal of rule R4 have the value X. R4 is thus found by the algorithm to be redundant, and thus deleted from the rule-base accordingly. The rule-size is thus reduced from 6 to 5, thereby indicating (see Table 7c) that: $\gamma(3) = 17\%$, and $\eta(3) = 83\%$. Care, however, must be
taken in interpreting these values. As can be gathered from module-3, the rule reduction algorithm compares the current state, \( k \), of the rule base to its initial state, i.e., state 0, to compute \( r(k) \) and \( \eta(k) \). These are \( k \)-th state values relative to state 0. The overall redundancy and efficiency of the rule base are obtained when no further reduction in rule size is possible, and the algorithm terminates. These values are given by \( \gamma(L) \) and \( \eta(L) \), where \( L \) is the terminal state of the rule base.

5. Conclusion

In this study, we presented the implementation algorithm and results of an integrated rule reduction system for expert systems that are based on well structured production rules. We showed, through our implementation scheme, that the IRRS is composed of three independent modules, each of which addresses one of three types of pathologies that occur in knowledge bases. We suggested that in order to avoid production of semantically confusing and meaningless knowledgebases, the modules in the IRRS be implemented in the sequence in which they are given in this study. For already existing expert systems, the IRRS could be used as a front-end interface; for systems that are being newly built, the IRRS could be incorporated as an integral part of such systems.

Further research is needed to extend the capabilities of the IRRS to other forms of knowledge representation. Another area of future research deals with processor efficiency. Further studies are also needed to ascertain and develop IRRS machine-level representational forms that will support efficient processor implementation schemes. We also believe that the issues of incompleteness and inconsistencies are sufficiently important to warrant further research.

References


