A methodology for rule-base integrity in expert systems

by GEORGE STEFANEK  
Illinois Institute of Technology  
Chicago, Illinois  
and  
SHI-KUO CHANG  
University of Pittsburgh  
Pittsburgh, Pennsylvania

ABSTRACT

The incremental addition of rules over time in a rule-based system warrants the need for a system to ensure rule-base integrity. A methodology is proposed in this paper that will check the addition of new rules against the existing rule-base for conflict, redundancy, subsumption, knowledge-related limits, resource conflict, knowledge conflict, and message conflict. A relational database is used to store the rules, and relational database techniques are used to analyze the database. A directed graph is used to represent relationships between knowledge, resources, messages, and database attributes. Thus, both the relational database and the directed graph serve as unifying methodologies in the design of the system. Also, a trace mechanism is provided to show the type of conflict and the rules involved.
INTRODUCTION

This paper presents a methodology to be used along with expert system shells to check rule-based systems for rule-base integrity. The Rule Integrity Sub-System, called RISS, uses a relational DBMS to store rules making up the rule-base and to analyze them for consistency and conflict using relational database techniques. A directed graph also is used to represent interrelationships between knowledge. Together, the relational database and directed graph form the unifying approach in storing, representing, and analyzing the rule-base.

As new rules are collected from various experts and added into a rule-based system, the rule-base becomes more complex and the probability of rule-base inconsistency increases. The proposed rule-base integrity subsystem increases the integrity of the rule-base by checking for: rule conflict, redundancy, subsumption, knowledge-related limits, resource conflict, knowledge conflict, and message conflict (see Figure 1). Rule conflict, redundancy, and subsumption have been discussed by Suwa, but are formalized and expanded in this paper. Knowledge-related limits, resource consistency, knowledge consistency, and message consistency are introduced and formalized here. Also, a trace mechanism is included in the system to report the type of conflict that occurs, display the rules involved in conflict, and provide other useful information. Finally, a secretarial rule-base of 60 rules is used as an example rule-base for analysis.

KNOWLEDGE REPRESENTATION

Data organization and knowledge representation are key points in the design of RISS. These strategies form the basis for the type of analysis performed throughout the entire system.

Much of the problem solving is accomplished using meta-knowledge. Meta-knowledge is knowledge about knowledge; in our case it is knowledge about various aspects of the system, rule-base, and other domain specific information. This procedural knowledge is specified in RISS as meta-rules, templates, and knowledge tables. All these forms of meta-knowledge are stored in various forms of knowledge tables.

Meta-rules are meta-knowledge in the form of “if-then rules.” They are used in manipulating the internal knowledge or specifying knowledge about strategies. For example, meta-rules are used to activate groups of rules in the inference process:

\[ r_{011} \]

\[ \text{If: } \text{category } \text{eq. phone_call} \]
\[ \text{Then: } \text{search rules 1 thru 12} \]

Knowledge tables in the form of database relations hold mappings, function templates, and some dynamic information. Templates are in the form of list structures (object, attribute, value) where object is the name of some function such as “notify,” attribute is the first function parameter, and value is the second function parameter, usually indicating the type of action to be taken by the function. These templates offer a lot of flexibility because they can differ from function to function and primarily are used to check function specifications within rules.

Graph Theoretic Approach

The simple directed graph is the unifying representation for all rule and database paradigms. Related-resource models, related-knowledge models, and message models all use the directed graph approach to represent interrelated information. Formally, a digraph \( G \) consists of a set of vertices \( V = \{v_1, v_2, \ldots \} \), a set of edges \( E = \{e_1, e_2, \ldots \} \) and a mapping \( \psi \) that maps every edge onto some ordered pair of vertices \( (v_i, v_j) \). A vertex is represented by a circle and an edge by a line segment between \( v_i \) and \( v_j \) with an arrow directed from \( v_i \)
to $V_i$, or $V_j$ to $V_i$, or both. The digraph has no self-loops, but can have parallel edges between some nodes. This approach toward representing knowledge and its interrelationships gives a common methodology for the representation and the analysis of knowledge. Directed graphs are a mathematically formalized approach for interrelating information, and they can easily be expanded to other knowledge representations as needed. The digraph structures are stored as knowledge tables in a relational database.

Relational Database Approach

A relational database is used for storing all data, rules, and meta-knowledge. It provides a cohesive way of manipulating all data. Standards can be enforced, data can be shared, redundancy reduced, and integrity increased. Relational database techniques such as restriction, projection, and division are used to manipulate and extract desired data and check for rule-base inconsistencies. That is, since all existing and new rules are stored in a relational database, all matching and checking is done within the context of the relational database.

Initially, new rules and other design information are accepted by the expert system front-end which outputs the gathered information in its own format. RISS uses an interface module to reformat the rules into a standard format understood by all its modules. The rules will be placed in temporary condition and action relations. The New_Conditions relation contains the attributes: NewRuleNo., Antecedent, Owner, Date, and Comment.

To analyze the rules for conflict, redundancy and subsumption against the existing rule-base, the rules in the New_Conditions and New_Actions relations are parsed and put into separate temporary relations that are used only during analysis. For example, in the New_Conditions and New_Actions relations the antecedent and consequent attributes hold all the conditions and actions of a rule. The conditions and actions must be broken up so that there is one condition or action per tuple in a relation. These relations are called Parsed_New_Conditions and Parsed_New_Actions. Parsed_New_Conditions contains the attributes: SeqNo., NewRuleNo., Antecedent, and Consequent.

The existing rule-base and meta-rule base are also stored in relational database form. The existing rules are stored in the Conditions and Actions relations and other rule information such as ownership and date of entry are stored in Conditions_Data and Actions_Data relations.

The Actions_Data relation contains one additional attribute called rule dependency which points to other rules, if any, that depend on the existing rule. This is necessary during deletion. Any rules associated with an existing rule in inferring a conclusion may also have to be removed given that the rule was the sole support. A rule supports other rules that are used before it in an inference process. If a rule is supported by more than one rule, then all of them must be erased before it is erased.

Meta-rules that specify system-related knowledge are stored in relations having identical attributes as the relations for the existing rule-base: Mr_Conditions, Mr_Conditions_Data, Mr_Actions and Mr_Actions_Data. Other meta-knowledge is stored in database relations including knowledge tables for locks and knowledge-related limits and decision tables.

SYSTEM TRACE CAPABILITIES

RISS has a trace and error handling capability. The trace mechanism keeps track of the current module in which analysis is taking place, the type of problem encountered, the existing rules involved, a display of those rules, the new rules involved, and the new rules that have passed so far. If an error is recoverable, the trace will prompt the user to decide whether to continue or to stop. An example of a trace is found in Figure 2. The following sections present the formalized methodologies used in enhancing rule-base integrity.

RULE CONSISTENCY

When knowledge is represented as production rules and new rules are added incrementally, inconsistencies may appear as:

1. Conflict: this may appear in two forms.
   a. Action conflict: two or more rules fire because they have the same conditions, but there is conflict in the action portion of the rules (that is the results are different).
   b. CF conflict: if certainty factors are used, the conditions and actions may be the same, but the associated certainty factors differ.

SYSTEM TRACE

MODULE: Consistency module
TYPE OF PROBLEM: Action Conflict

EXISTING RULE(S) INVOLVED: 036

\[036 \text{ If: category .eq. filing and filing.type .eq. letter} \]
\[\text{Then: file(letter,filing.company) and notify(sender, stat7)} \]

NEW RULE INVOLVED: 002

\[\text{new} \text{ If: category .eq. filing and filing.type .eq. letter} \]
\[\text{Then: file(letter,filing.product) and notify(sender, stat7)} \]

NEW RULES PASSED SO FAR: 001

HARDCOPY? < Y/N >

Figure 2—Example of output from RISS trace
2. **Redundancy**: two or more rules fire because they have identical conditions and actions.

3. **Subsumption**: two or more rules have the same conditions and actions, but some of the rules have more conditions that make such rules more restrictive. Also, two or more rules have identical conditions, but some of the rules have more action clauses than other rules.

The following sections describe each area of rule conflict in detail, starting with the presentation of the problem and followed by examples.

**Conflict Avoidance**

Rule conflict may occur as either action conflict or CF conflict. Action conflict occurs when a rule that is being added has identical conditions with an existing rule, but the actions differ. The actions may differ completely in task execution or can merely have more action clauses in one rule vs. another. A rule $r_i$ is defined as the ordered pair $(C_i, A_i)$. The set of all conditions in rule $r_i$ is denoted by $C_i$ and the set of all actions in rule $r_i$ is denoted by $A_i$. Given two rules $r_i$ and $r_j$ from the set of rules in the rule-base $R$, conflict occurs when $C_i = C_j$, $A_i \neq A_j$, and $A_i$ is not a proper subset of $A_j$, $A_i \not\subseteq A_j$. For instance, there is an existing rule in the example secretarial rule-base $R_{020}$ which specifies that a letter should be filed according to the company to which it is addressed, and that the sender should be notified that the filing has been accomplished.

$$r_{020}$$

*If:* category .eq. filing and filing.type .eq. letter

*Then:* file(letter, filing.company) and notify(sender, stat7)

Suppose the domain expert wishes to add another rule into the rule-base which has the form:

$$r_{\text{new}}$$

*If:* category .eq. filing and filing.type .eq. letter

*Then:* file(letter, filing.product) and notify(sender, stat7)

These two rules differ in that the new rule’s actions designate the letter to be filed with documents describing the product to which it pertains, as opposed to the existing rule which specifies the letter should be filed by company. This is a simple case of rule conflict: the conditions are the same, but the actions differ. The system will flag this problem and report it through the trace module shown in Figure 2. The trace will halt further analysis of additional new rules as soon as a conflict is found. From the trace, the expert adding rules will have enough information to know how to proceed. For a new rule to be added successfully, the existing rule will have to be removed first or the new rule and any related new rules may have to be modified.

The second form of conflict is CF conflict, which occurs when all conditions and actions are the same but the certainty factors (CF) associated with the conclusions differ. Given a rule $r_i$ consisting of the ordered pair $(C_i, [A_i, CF_i])$, the set of all conditions in a rule $r_i$ is denoted by $C_i$, and the set of actions having a certainty factor of $CF_i$ is denoted by $A_i$. Given two rules $r_i$ and $r_j$, CF conflict occurs when rule $r_j$ has a different certainty factor from an existing rule $r_i$ in the rule-base $R$. That is, $C_i = C_j$, $A_i = A_j$, and $CF_i \neq CF_j$. For instance, given an existing copy machine rule $R_{052}$ and a new rule $R_{\text{new}}$:

$$r_{052}$$

*If:* copy machine is on and copy button is pressed with no response and no warning light is on

*Then:* there is strongly suggestive evidence (.9) that the machine is broken

$$r_{\text{new}}$$

*If:* copy machine is on and copy button is pressed with no response and no warning light is on

*Then:* there is strongly suggestive evidence (.8) that the copy machine is broken

Both rules are the same except that the certainty factor in the existing rule is .9 and the certainty factor for the new rule is .8. Once the certainty factor parameter of the Copy Machine function is compared and found different, the trace will report the error and the program will terminate. The system will check whether $C_i = C_j, A_i = A_j$, and $CF_i \neq CF_j$, and if it is true, the system will generate a trace reporting this type of conflict and end analysis of the rule-base. After receiving the trace, the user will have to decide whether to delete the existing rule.

**Redundancy**

Rule redundancy exists if two or more rules have identical conditions and actions. Given two rules $r_i$ and $r_j$, redundancy exists when $C_i = C_j$ and $A_i = A_j$, thus $r_i = r_j$. If the conditions and actions are identical, but the order of either conditions or actions differs, then the redundancy check will still flag the rules as identical and go through the trace. To change the order of the conditions or actions in a rule which is redundant with a new rule, the existing rule must be deleted from the rule-base and the new rule added afterward. For example, the following two rules will be treated as being redundant:

$$r_{020}$$

*If:* category .eq. In_person

In_person.name .eq. employee

In_person.meeting .eq. 1

In_person.travel .eq. 1

free(In_person.time) .eq. t

*Then:* schedule(In_person.time, In_person.name) travel(In_person.name, In_person.time, In_person.dest)
Subsumption

Rule inconsistency can also occur in the form of subsumption. That is, a new rule can subsume an existing rule if it has a superset of conditions or actions of an existing rule. Subsumption has been discussed by Suwa1 and is expanded and formalized in this section. Subsumption can occur in two forms:

1) a superset instance
2) a subset instance

The superset instance occurs when a new rule has a superset of the conditions of an existing rule. The new rule will have more conditions, be more restrictive, and include all the conditions of an existing rule as a subset of its conditions. If r₂ is a new rule and r₁ is an existing rule, then the superset instance occurs when, C_I ⊆ C_₂ and A_I = A_₂. Another possibility is if the new rule has a superset of the actions in an existing rule, C_₂ = C_₁ and A_₃ ⊆ A_₁. Finally, both conditions and actions in a new rule may be supersets of an existing rule, C_₂ ⊆ C_₁ and A_₃ ⊆ A_₁. In the event that any of these superset instances occur, the new rule will supersede the existing rule. As an example, given an existing copy machine rule r₀₅₄ and a new rule r₉₆₄:

r₀₅₄
If: copy machine warning lights are checked and
toner light is on
Then: open machine
fill with toner

r₉₆₄
If: copy machine warning lights are checked and
toner light is on
Then: open machine and
fill with toner and
turn machine on

The new rule’s actions are a superset of the existing rule’s actions; therefore, the new rule will supersede the existing rule. This is an example of subsumption superset instance.

When certainty factors are involved, this check of logical consistency will flag the problem as if certainty factors didn’t play a role. If the conditions or actions are a subset of an existing rule but have a different certainty factor, the trace will indicate it and the existing rule will not be superseded. It must first be deleted in order for the certainty factor to change. In the superset instance, the new rule will supersede the existing rule with the new certainty factor replacing the old.

The subset instance occurs when a new rule has a subset of the conditions in an existing rule. If r₂ is a new rule and r₁ is an existing rule, then a subset instance occurs when C₁ ⊆ C₂ and A₁ = A₂. Also, the situation exists when a new rule has a subset of the actions in an existing rule, A₁ ⊆ A₂ and C₁ = C₂. Finally, both conditions and actions in a new rule can have subsets of an existing rule’s conditions and actions, C₁ ⊆ C₂ and A₁ ⊆ A₂. If any of these subset instances occur, then the new rule will not be added to the rule-base.

Knowledge-related Limits

Many higher level variables and resources may be used in a rule-base. These should be kept track of so that they are not depleted and so certain artificially set limits are not exceeded. Also, the knowledge related limits should be dynamically adjusted to accommodate changes in the rule-base. For example, suppose the following rule exists in the rule-base:

r₀₅₆
If: category .eq. stationery and
number of envelopes .lt. 10
Then: order a new box of envelopes

A new rule to be added is:

r₉₆₄
If: category .eq. stationery and
number of envelopes .lt. 8
Then: order a new box of envelopes

It can be seen initially that these are two different rules with the same conclusion. Upon closer inspection, the second condition in each rule is related. That is, the existing rule already includes the limit set by the new rule. Since the number 8 is less than 10, the new rule’s limit falls under the existing rule’s larger limit. However, it may be that the new rule is intentional and that the limit should be lowered from 10 to 8. In this case the system should flag the problem, report it in the trace, and the user should delete the existing rule before adding the new one. Knowledge tables hold constraint information of the form \{aᵢ,rₗ, ⪯ \} where aᵢ is an attribute such as the number of envelopes, rₗ is a relational operator such as “lt” or “eq,” and ⪯ is the constraint or limit. Each time an attribute is encountered, it is compared against the knowledge limits table to check if it falls within some limit already prescribed. If a rule has identical conditions to an existing rule except for the constraining condition, then the knowledge limits table is searched for the attribute with the constraining condition. If it is found, the constraint is compared and reported in the trace. If it is not found, the rule is passed to the
next test and the constraint flagged for update into the knowledge table.

RESOURCE CONSISTENCY

A rule-based system may make use of a database. If a database is to be updated by functions invoked by a rule which fires, then one of the considerations that should be addressed is the possibility of resource conflict. A resource is defined as a quantity of objects or allocation of time. Typically, a resource is an item quantity (e.g., the number of nuts or bolts) or a time related attribute (e.g., for scheduling or reserving dates). Resource conflict will occur when a rule fires and: (1) requires access to a resource which was recently updated, (2) rules in an inference path have access to the same database attribute, (3) rules update or access an attribute which is related to other attributes, or (4) individual rules are incomplete and therefore may contribute to a resource conflict.

To maintain resource consistency by avoiding the problem of resource conflict, the following strategies are used:

1. **lock table**: used to lock a resource on either a time dependent basis or by user ID
2. **multi-rule resource conflict model**: used to check that the same resource isn’t accessed during an inference process
3. **related resource model**: used to show the relationships between various related resources.

**Lock Table**

A resource such as reserving a room or scheduling an appointment should be checked against being updated shortly after it has been set. A rule may fire which invokes a function that allocates a resource. Subsequent access to this resource by other rules or other functions within a rule should be restricted on a time- or ID-dependent basis. The restriction on a resource will be in the form of a lock, which will be specified in a lock table. The lock table is a knowledge table in the form of a database relation and consists of the following attributes:

1. **DB relation**: specifies the database relation containing the resource to be locked.
2. **DB attribute**: specifies the attribute in the relation representing the resource to be locked.
3. **RR function**: specifies the function in a rule that requires access to the resource.
4. **Tuple Num.:** the number of the tuple in the relation holding the resource to be locked.
5. **Lock type**: specifies the type of lock to be put on the resource.
   - The lock type may be either time dependent or ID dependent. The time dependent lock is set on an attribute for a length of time \( t \). The time length is set ahead of time by updating the lock table. The ID dependent lock is used to lock a resource by user ID. Only the original updater can unlock the resource. Both of these types of locks are set ahead of time in the lock table.
6. **Time**: specifies the length of time \( t \) that a resource attribute may be locked.
7. **ID**: specifies the ID of the user who updated the resource last.

The lock types are preset in the lock table and may be changed anytime. To avoid resource conflict when a rule fires, the lock table is checked by keying off the DB relation and DB attribute, which is the resource, and by checking the lock type. If the lock type is time, the time field is then checked to see if the resource can be accessed. If the lock type is ID, then the ID attribute is checked against the current updating user. Most of the problems of resource conflict can be handled by either the knowledge tables or appropriate database relations (e.g., include attributes such as “appointee,” “visitor” to identify the person for which the appointment is made).

**Multi-rule Resource Conflict Model**

The multi-rule resource conflict model is used to check rules in an inference path for access to a single resource. A resource should not be updated more than once in a single inference since the second update will cancel out the first and make it meaningless. Therefore, a set of rules involved in an inference should be checked for functions which update the same database attribute.

To check for this potential problem, a model in the form of a directed graph \( G_{mm} \) is constructed. All conclusions in a rule represented as functions which update the database are assigned as vertices \( V_{mm} = \{v_{mm1}, v_{mm2}, \ldots \} \) in the graph \( G_{mm} \). The direction of the arc \( e \), from a vertex \( v \) to a vertex \( v \), indicates that vertex \( v \), representing a database updating function, precedes vertex \( v \) which represents another database function. The graph \( G_{mm} \) contains all database updating rules involved in an inference path. The functions are linked by directed arcs in the sequence they fire during an inference. If a vertex \( v \) has more than one arc \( a_1, a_2, \ldots \) pointing to it, then there are duplicate functions or conclusions in the inference path. If this function updates a database, then it can cause resource conflict.

**Related-resource Model**

Resource conflict can also occur if resources that are related to one another are not updated simultaneously. That is, a resource such as a nut is related to a resource such as a bolt. For every nut there should be a bolt, or every time a trip is scheduled for a period of time, all appointments during that period should be rescheduled. These relationships or associations between resources in the database are represented by a related resource model.

The related-resource model is in the form of a directed graph \( G_r \) where the vertices \( V = \{v_{rr1}, v_{rr2}, \ldots \} \) correspond to a resource attribute and each arc from the set \( E = \{e_{rr1}, e_{rr2}, \ldots \} \) connects two related attributes \( v_r \) and \( v_r \). The associations between resources are predefined as world knowledge. Any new associations should be updated manually in this model. The functions in the actions portion of a new rule \( r \) are
checked against a knowledge table holding all functions which update a database. If the new rule’s function matches and therefore updates a database, then the functions’ parameters are checked for the attribute being updated and this attribute is matched against the graph $G_r$. If the resource attribute $v_i$ is associated with another resource $v_j$, then the resource $v_j$ is checked further for any other associations and so on. Last, the list of all associated resource attributes, which are found by searching the graph $G_r$, are compared with functions updating the database. The actions portion of a rule must contain functions which update all these related resource attributes in the list.

**KNOWLEDGE CONSISTENCY**

An extension of resource conflict in a database domain is knowledge conflict. Knowledge is defined as an item of information or as an object of information. Knowledge conflict can occur when related pieces of knowledge are not updated simultaneously.

Some attributes in a database such as a company, its phone numbers, street address and city address may be related to one another. If a company changes its location, not only does the street address have to be changed, but probably the zip code and possibly city, state, and other attributes have to be changed as well. These are related pieces of knowledge. Also, a semantic network consists of related pieces of knowledge that may have to be updated simultaneously when a change is made.

A related-knowledge model in the form of a digraph $G_{km}$ is set up to represent related pieces of knowledge. Each vertex $v_j$ in the graph represents either an object in the form of a fact or a database attribute. Related pieces of knowledge are connected by arcs. Each time a piece of knowledge is referenced, it is checked against the graph $G_{km}$ to see if other pieces of knowledge should be updated simultaneously.

**MESSAGE CONSISTENCY**

Some rule actions may include messages that are sent if a rule fires. If multiple rules are in an inference path, then many messages may be sent. Following is a list of some of the potential problems that may be encountered when many rules fire:

1. Identical or redundant messages may be sent by several actions invoked within a single rule.
2. Conflicting messages may be sent from more than one rule in a single inference.
3. Identical or redundant messages may be sent from more than one rule in a single inference.
4. Conflicting messages may be sent by actions invoked within a single rule.

All these message problems make up a category of conflict called message conflict. Message models are proposed as a method of increasing message consistency by checking for message conflict.

**Intra-rule Message Conflict**

Rules may contain a number of actions that may fire resulting in more than one message being sent from a single rule. For instance, two messages which will be sent when rule $r_{001}$ fires:

$$r_{001}: \text{If: } \begin{cases} \text{category} .eq. \text{phone\_call} \\
\text{phone\_call\_name} .ne. \text{null} \\
\text{location(boss)} .eq. \text{"out"} \end{cases} \text{Then: } \begin{cases} \text{notify(sender,statl)} \\
\text{inquire(phone\_number)} \\
\text{generateform(type.memo,memo.info)} \\
\text{mailbox(memo,receiver,stat14,sender)} \end{cases}$$

First, the sender is notified with a message $\text{stat1}$ that the person he wishes to talk to is out, $\text{notify(sender,stat1)}$. Then the sender is notified that the intended receiver of his phone call has been notified by a mail message, $\text{mailbox(memo,receiver,stat14,sender)}$.

Intra-rule message conflict is defined as conflict between messages sent from a single rule. Message conflict is defined as separate messages appearing together and having conflicting or contradictory information. A message model is used to designate what is conflicting or contradictory between messages. The message model consists of a set of vertices $V_{mm} = \{v_{mm1}, v_{mm2}, \ldots\}$, a set of arcs $E_{mm} = \{e_{mm1}, e_{mm2}, \ldots\}$ where each vertex $v_i$ corresponds to a message $m$, and each arc $e_i$ has a flag $F$ designating a positive or negative association $[e_i,F]$. A negative association, $F = -1$, from vertex $v_i$ to $v_j$ indicates that message $m_i$ should not be associated with message $m_j$. A positive flag, $F = +1$, indicates that the association is valid. That is, both messages $m_i$ and $m_j$ can appear together. All messages in the graph are linked to all other messages and each link is assigned an association flag.

Intra-rule message conflict is resolved by using the message model to check whether messages in the action portion of the rule conflict or are redundant. The action functions in the rule to be analyzed are compared to a knowledge table containing templates matching the correct message to a particular function (see Table I).

The message associated with a particular action function is then matched against the message model. If additional functions in the rule send messages, they also are compared to the message model. If these messages are connected by arcs, the flag $F$ associated with the arc $e$, connecting the two messages $m_i$ and $m_j$, is checked to verify the validity of the association of the two messages, $m_i$ and $m_j$. If the flag equals $-1$, then the

<table>
<thead>
<tr>
<th>TABLE I—Sample knowledge table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context</strong></td>
</tr>
<tr>
<td>mail</td>
</tr>
<tr>
<td>file</td>
</tr>
</tbody>
</table>
grouping of these messages in a single rule is not allowed, and RISS reports the problem through the trace. If the messages are the same, then this problem is also reported through the trace. Otherwise, the messages may be grouped together and RISS continues on without interruption.

Inter-rule Message Conflict

Rules may fire one after another in an inference path before coming to a conclusion. As the actions are invoked, messages may be sent that are conflicting or redundant. To avoid this problem, new rules are checked to see if they are standalone or if they are involved in an inference path. This is done by checking a rule dependency model for all rules involved in an inference. The details of this model will not be discussed here. Once all the rules in an inference path are found, the action clauses of these rules are searched for functions containing messages. Those functions are compared to the knowledge table containing templates which match the correct message to the function in the action portion of the rule. All messages involved are gathered and each message is matched to the message model to see if any messages in that inference path should not be grouped together. If any messages should not be grouped together as indicated by the association flag, then this problem is reported through the trace module.

Database Message Conflict

In a database environment, access or update of certain attributes in various relations may be monitored. Specific accesses or updates trigger messages that are sent by the monitor on the database relation. The monitor kernel may be rule-based in the form of alerter rules, which designate actions that should be taken upon any access or update of database attributes. Therefore, the techniques described in the last two sections would apply.

CONCLUDING COMMENTS

Developing and formalizing a methodology that tries to increase rule-base integrity is helpful during the ongoing addition of rules into an expanding rule-base. This paper introduces some techniques for achieving rule-base integrity and opens up some new possibilities and questions. Future work will include the development of rule models, rule-set models, and knowledge belief maintenance models. Rule models will be used to check rules for completeness (e.g., whether certain conditions or actions are missing for a rule). Rule-set models will attempt to check for missing rules, and belief maintenance models will attempt to check new rules against an internal view of the world (belief) as based on existing rules and previous world knowledge. This will include the ability of a system to learn about itself in all modules including the ability to learn about the world through the addition of new rules to the rule-base and thereby expand its knowledge dynamically. Currently, world-view knowledge is predefined and added manually. Also, a friendly administrative interface should be developed to make changes to domain specific knowledge, templates, etc. Additional modules could be added to handle specifics about backward chaining and other inference mechanisms as well as analysis of other knowledge structures and the inferences that are made from them.

Although this paper focuses on rule-base integrity, the concepts presented can apply to other knowledge representations. These methodologies could easily be extended or tailored to encompass other knowledge representations. These additional extensions based on the original concepts could make this a more general knowledge integrity methodology that would be more applicable to systems which make use of several representations in their design.

RISS is implemented on a VAX 11/780 running VMS, and uses the ORACLE database to store rules, and ORACLE's SQL language to handle database analysis. C programs call embedded SQL procedures to handle rule-base analysis.

REFERENCES
