A Variable-Centered Intelligent Rule System

Irfan Subakti
Department of Informatics, Faculty of Information Technology
Institute Technology of Sepuluh Nopember Surabaya (ITS)
Kampus ITS, Keputih, Sukolilo, Surabaya, Indonesia
yifana@gmail.com

Abstract

A Rule-based System (RBS) is a good system to get the answer of What, How, and Why questions from the rule base (RB) during inferencing. Answers and explanations are properly provided. The problem with RBS is that it can't easily perform the knowledge acquisition process and it can't update the rules automatically. Only the expert can update them, manually, by the support of a knowledge engineer. Moreover most researches in RBS concern more about the optimization of the existing rules than about generating new rules from them. Rule optimization, however, can't change the result of the inferencing, significantly, in term of knowledge coverage.

Ripple Down Rules (RDR) came up to overcome the major problem of expert systems: experts no longer always communicate knowledge in a specific context. RDR allows for extremely rapid and simple knowledge acquisition without the help of a knowledge engineer. The user doesn't ever need to examine the RB in order to define new rules: the user only needs to define a new rule that correctly classifies a given example, and the system can determine where the rule should be placed in the hierarchy. The limitation of RDR is the lack of powerful inference. RDR seems to use Depth First Search which lacks the flexibility of question answering and explanation accrued from inference.

A Variable-Centered Intelligent Rule System (VCIRS) is our proposed method. It hybridizes RBS and RDR. The system architecture is adapted from RBS and obtains advantages from RDR. This system organizes the RB in a special structure so that easy knowledge building, powerful knowledge inferencing and evolutional improvement of system performance can be obtained at the same time. The term "Intelligent" stresses that it can "learn" to improve the system performance from the user during knowledge building (via value analysis) and refining (by rule generation).

Keywords: Rule-based Systems, Ripple Down Rules, knowledge building, knowledge inferencing, knowledge refining

1. Introduction

A knowledge-based system with knowledge structured by rules is called RBS, alternately called an expert system. The place that stores rules is called a KB. A KB can be organized in a variety of configurations to facilitate fast inferencing (or reasoning) about the knowledge. A "traditional" RBS uses forward and backward chaining during inferencing. Through the inference process we can obtain answers to such questions as: What is the result of the inference process? How does it do it? Why can it do it? It has been proposed that expert knowledge is always provided in a context and should therefore be used in the context [1]. Knowledge acquisition (KA) methodologies have been proposed to capture and use knowledge in contexts. Thus, in RBS, the expert has to communicate knowledge in a specific context.

RDR is a KA method which constrains the interactions between the expert and a shell to acquire only correct knowledge [6]. RDR overcomes the major problem of expert systems: experts no longer always communicate knowledge in a specific context. In RBS it is assumed that the context is the sequence of rules which have been evaluated to give a certain conclusion [2]. RDR allows for extremely rapid and simple KA without the help of a knowledge engineer; by providing an extensive support to the user in defining rules [1]. And this is where RDR systems gain their power compared to traditional RBS [5]. The user does not ever need to examine the RB in order to define new rules: the user only needs to be able to define a new rule that correctly classifies a given example, and the system can determine where the rule should be placed in the rule hierarchy. In contrary with RBS, the only KA task in RDR is for the expert to select from a list of conditions. The expert thus has a very restricted task and involves nothing with how knowledge base is structured. An implemented system based on the RDR approach which is now in routine use in a pathology laboratory with knowledge added by experts without the intervention of a knowledge engineer was reported by Edwards *et al.*[3].

RBS has a well-known limitation: RBS can't update its rules automatically. Only the expert can update it, manually, by the support of a knowledge engineer. However, RBS can do powerful inferencing to answer a variety of questions such as:

What if? How? Why?

On the other hand, even though an RDR system provides extensive help to the user in defining rules and maintaining the consistency of the RB, RDR has a limitation in knowledge inferencing. RDR seems to use Depth First Search (DFS) to traverse its nodes during inference. It limits the flexibility of question answering and explanation accrued from inferencing as presented in the RBS.

The limitation of RBS (i.e., experts update rules manually) and the advantage of RDR (i.e., it allows for extremely rapid and simple KA without the help of a knowledge engineer) are the basic motivations for this paper. Another motivation is a desire to empower the existing rules in the KB by generating new rules, in contrast to optimizing those rules.

RBS is a good system to get the answers of What, How, and Why questions from the RB during inferencing. Answers and explanations are properly provided. The problem with RBS is that it can't update the rules automatically. Only the expert can update them, manually, by the support of a knowledge engineer.

RDR can generate rules by adding exceptions caused by misclassified cases with respect to the cornerstone cases. Rules are never deleted from an RDR RB. The KA process does not need the help of a knowledge engineer. However, it needs the expert to correct misclassified cases into exceptions. The limitation of RDR is the lack of powerful inference. Unlike RBS which is equipped with inference through forward and backward chaining, RDR seems to use Depth First Search (DFS), which reduces the flexibilities of question answering and explanation stemmed from inferencing.

Moreover most researches in RBS concern more about the optimization of the existing rules than about generating new rules from them. Rule optimization, however, can not change the result of inferencing, significantly, in terms of knowledge coverage. Updating rules to increase coverage is usually done manually by the expert.

Simplification of knowledge building is achieved by providing the user with the simple steps in the knowledge building process. The user doesn't need to consider about the knowledge base structure and can update knowledge base directly, like RDR does. The system will guide the user during the knowledge building process.

Empowering knowledge inferencing is achieved by providing the user with a guideline (i.e., the result of variable and value analysis), so that the user knows the order of importance and usage of the cases from the knowledge base. And the RBS-based inferencing is equipped too, so that the user can regain flexibility of inferencing.

Evolutionally performance improving is achieved by providing the user with the structure which supports variable and value analysis for new rule generation. Rule generation improves the coverage of domain knowledge. Moreover, value analysis also guides the user during knowledge building and inferencing. Along with rule generation, this capability can improve the system performance in terms of knowledge inferencing.

As stated before, most researches in RBS are concerned about the optimization of rules through machine learning methods such as C4.5, since they produce an optimal tree [7]. Suryanto and Compton [10] proposed Invented Predicates, a machine learning technique that could speed up KA. By generalizing the knowledge provided by the expert in order to reduce the need for later KA. This generalization is completely hidden from the expert. Forgy proposed the Rete algorithm to improve the speed of forward-chaining rule systems by limiting the effort required to recompute the conflict set after a rule is fired [4]. This method creates a decision tree (network) that combines the patterns in all the rules of the KB. The Rete algorithm is a very efficient method for pattern match problem by reorganizing production rules. Its drawback is that it has high memory space requirements.

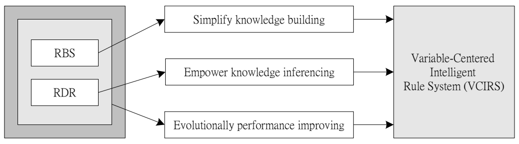
Suryanto and Compton proposed to do intermediate concept discovery in RDR knowledge bases [9]. Here, the RDR knowledge base is reorganized by converting Multiple Classification Ripple Down Rules (MCRDR) into flat rules followed by removing redundancy, to facilitate the discovery of intermediate concepts. With this type of RBs, an example (i.e., a case) may have a classification resulting from multiple conclusions in the RB. These conclusions are found by following every path in the RDR tree to the most specific node applicable to the example. The classification is then the set of conclusions from all such nodes. They use a simulated expert to rebuild the MCRDR KBS and apply Duce's intra-construction and absorption operators in each KA session [8]. It's interesting; however, they stated that it is not clear whether the compression achieved is a result of the behavior of the expert and may only occur when the expert behaves in particular ways. They found there were not many intermediate concepts to be discovered in their domain (interpreting lipid results in chemical pathology). Considered as a failure, they suggest that it may be worthwhile reconsidering the importance of intermediate concepts. In short, the paper tried to optimize (compress) KBS rules by alleviating possible repetition, redundancy and lack of intermediate concepts. Note that the expert still contributes a lot here.

The rest of the paper is organized as follows. In Section 2, we describe our proposed method, VCIRS. Finally we conclude our research in Section 3.

2. Variable-Centered Intelligent Rule

System

Fig. 1 illustrates our method, which hybridizes techniques from Rule-based Systems and Ripple Down Rules to form a Variable-Centered Intelligent Rule System (VCIRS). VCIRS has a structure to organize the rule base so that easy knowledge building, powerful knowledge inferencing, and evolutional improvement of the system can be obtained at the same time.



RBS
RDR
Simplify knowledge building
Empower knowledge inferencing
Evolutionally performance improving
Variable-Centered Intelligent Rule System (VCIRS)

Figure 1. Proposed Method

First, knowledge building is simplified by the simple steps in the knowledge building process. The user needs no consideration about the knowledge base structure and can update knowledge base directly. VCIRS allows the user to refine or add node into the existing knowledge base. As in RDR, rule refining is the creation of an exception rule to correct a misclassification, while addition refers to adding a new rule at the top level of the tree. The system guides the user during the knowledge building process.

Knowledge inferencing is enhanced by the knowledge (i.e., the result of variable and value analysis) of the order of importance and usage of the cases from the knowledge base. The RBS inferencing mechanism is brought back in VCIRS, so that the user can obtain more answers and explanations from inferencing.

System performance is improved by the rule base structure which supports variable and value analysis for rule generation. Rule generation improves the result of inferencing in terms of knowledge coverage. Moreover, value analysis also guides the user during knowledge building and inferencing. Along with rule generation, this capability can improve the system performance in terms of knowledge inferencing.

We use the term “Intelligent” in VCIRS to stress that it can “learn” from the user, during knowledge building (i.e., value analysis) and refining (i.e., rule generation). Furthermore, rule generation along with the capability of the system which is able to performing RBS inferencing, can evolutionally improve the system performance.

Fig. 2 shows the architecture of VCIRS which adapts the traditional RBS architecture

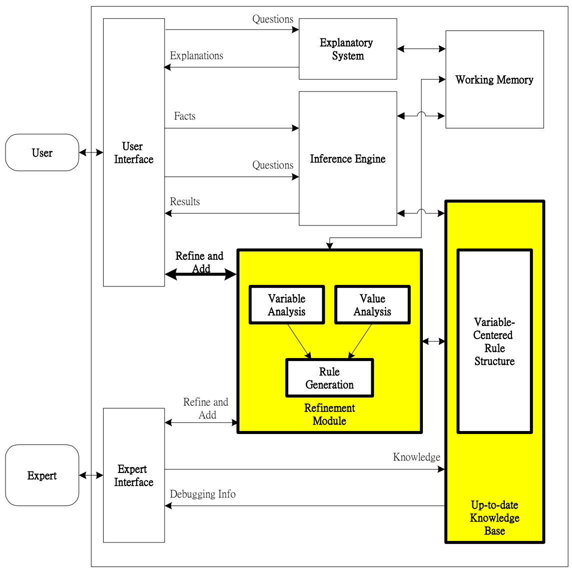


Figure 2. VCIRS architecture

A Variable-Centered Rule Structure is used to represent knowledge base and supports the Refinement Module to maintain an up-to-date knowledge base. It also records cases and their occurrence. The fundamental element of Variable-Centered Rule Structure is a variable, posted by the user. VCIRS maintain carefully the variable about its values, structure and occurrence. A sequence of variables constitutes a node, while a sequence of nodes composes a rule. Thus, the Variable-Centered Rule Structure contains a rule structure and a node structure centered on variables.

The case presented by the user goes to working memory during knowledge building, then save permanently into Variable-Centered Rule Structure while the system records the rule information and calculates the occurrence of each case. Then, the rule information recorded is used by Variable Analysis to get the important degree. In other hand the occurrence of each case is used by Value Analysis to get the usage degree. The usage degree will help the user to be a guideline during knowledge building and inferencing for deciding which variable she has to visit first. Along with the important degree, the usage degree will support Rule Generation for producing the new rule/node.

This section describes the system in the algorithmic level.

2.1 Variable-Centered Rule Structure

2.1.1 Node Structure

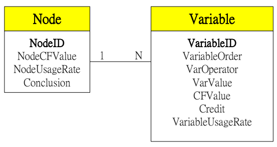


Figure 3. Node Structure

Fig. 3 describes the conceptual graph of Node

Structure. Given a new case posted by the user, from working memory VCIRS enters it in the Node Structure and then uses the Rule Structure as a piece of up-to-date knowledge base. The Node Structure also saves the occurrence of variables and nodes, for usage assignment. A case consists of a set of fields of data as depicted in the Fig. 4.

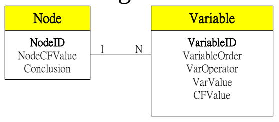


Figure 4. Case fields

Each time the user posts her cases, the Node Structure maintains the values and their positions. This information will be used in usage assignment.

In VCIRS the user is allowed to refine or add node into the existing knowledge base. Rule refining is the creation of an exception rule to correct a misclassification, while addition refers to adding a new rule at the top level of the tree. The system guides the user during the knowledge building process, to be described in detail in next section.

As in RDR, rules (nodes) in VCIRS are never modified or removed. It guarantees the consistency of the data with regard to the philosophy that such a system is the system of "if-then" rules organized in a hierarchy of rules and exceptions (exceptions to rules may themselves have exceptions, etc.) [5]. Even though rules are never deleted from a knowledge base, a rule can be "stopped," to ignore any conclusions that would normally be generated using the rule. This won't ignore those obtained from its exceptions (and their exceptions, etc.) however.

As knowledge inferencing in progress, VCIRS uses the Node Structure (and also the Rule Structure) for the structure source. The Rule Structure maintains the rule information, while the Node Structure keeps the information about the occurrence of the saved cases. The user can input facts (cases) and get the result after inferencing.

2.1.2 Rule Structure

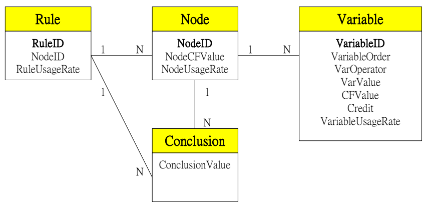


Figure 5. The conceptual graph of Rule Structure

Fig. 5 describes the conceptual graph of Rule Structure. As stated before, cases posted by the user are firstly saved in the Node Structure; it will be then used in the Rule Structure.

2.2 Knowledge Refinement

There are 3 tasks in the Refinement Module: variable analysis, value analysis and rule generation.

2.2.1 Variable Analysis

Inside the Variable-Centered Rule Structure the system knows what nodes are shared by rules, and what variables are shared by nodes. The more number of rules a node is shared by; the more important the node would likely be. The same reasoning applies to the shared variable within the nodes.

These facts reveal how important a node/variable is, while serves as a starting point for generating a new rule, along with value analysis.

2.2.2 Value Analysis

Value analysis offered by VCIRS is based on the data entered by the user. The purpose of value analysis is:

1. Give a guideline to the user/the system during knowledge building, about which rule/node/variable to visit first in rule tree traversing, e.g., the most used rule/node/variable first, followed by the second often used one and so on. This is useful in helping the user decide which cases to enter.
2. Give a guideline to the user/the system during knowledge inferencing, about which rule/node/variable to visit first in rule tree searching, as in knowledge building. This is useful in helping the user focus on most interesting rules/nodes/variables in the knowledge base.
3. Give a guideline to the system, together with variable analysis for the rule generation process. The system thus gets another perspective, in addition to the sharing occurrence from the variable analysis (i.e., node and variable) before deciding to generate a rule. By the result of value analysis the system knows the usage degree of rules/nodes/variables, while from the variable analysis the system will know the important degree of the node/variable within a rule/node.

The process of value analysis, called usage assignment, is to determine the usage degree of rules/nodes/variables in the knowledge base. The usage assignment uses the information stored in the Variable-Centered Rule Structure.

We have three usage degrees. First, Variable Usage Rate (VUR) is used to measure the usage of a variable within a node being posted and used. Second, we use Node Usage Rate (NUR) to measure the usage of a node on firing. The last is Rule Usage Rate (RUR) which measures the usage of a rule on firing. The larger the usage indicator is, the more usage the value is, and vice versa.

Equation (1) calculates VUR for the *i*-th variable, (2) gives NUR for the *j*-th node, while (3) defines RUR for the *k*-th rule.

$$\text{VUR}_i = \text{Credit}_i \times \text{Weight}_i \quad (1)$$

$$\text{NUR}_j = \frac{\sum_1^N VUR_{ij}}{N}, \text{ VUR}_{ij} \text{ for } i\text{-th variable in node } j \quad (2)$$

$$\text{RUR}_k = \frac{\sum_1^N NUR_{jk}}{N}, \text{ NUR}_{jk} \text{ for } j\text{-th node in rule } k \quad (3)$$

Where:

- Credit_i = the occurrence of variable *i* in the Node Structure (4)

Credit is obtained from the Node Structure. It increases when the user creates a node which agrees with the values of the old case.

- $\text{Weight}_i = \text{NS}_i \times \text{CD}_i$ (5)

Weight calculates the weight of a variable to the node which it belongs to. There are 2 factors which contribute the weight of a variable. The first is the number of nodes sharing a variable, and the second is CD, the degree of closeness of a variable upon a node.

- NS_i = number of nodes sharing variable *i* (6)
- $\text{CD}_i = \frac{VO_i}{TV}$ (7)

CD stands for Closeness Degree. CD_i in node *j*, calculates the degree of closeness of variable *i* in node *j*. The closer a variable is to the conclusion's node, the better it is. Thus, it based on the order of a variable in a node (note: a node is the sequence of variables). CD is calculated by the order of variable VO, divide by total variables TV, belonging to a node.

- VOi = Order of variable *i* in a node (8)
- TV = total variables belong to node (9)

2.2.3 Rule Generation

Rule generation works according to the result of variable and value analysis. Note that from variable analysis we calculate the important degree of a node/variable, while from value analysis we obtain the usage degree of rules/nodes/variables.

Information about shared node/variable from variable analysis is useful to choose a good candidate to make a combination. The most shared node/variable means it is the most important node/variable in the existing knowledge base, because it is used in many places in the recent structure. The most usage degree produced by value analysis means this node/variable has the largest occurrence inside the structure.

Variable combination combines variables to produce a new node, while node combination combines nodes to produce a new rule. These combinations can be done as long as the order of variables/nodes being produced doesn't break the existing order of variables/nodes in the knowledge base. Because there are many combination possibilities, we present a new approach to perform variable and node combination below.

We generate a rule by combining the nodes in according with the order of the existing nodes in the rules. It's depicted in Fig. 6.

1. For each of the most important nodes, produced by variable analysis, select one as the conclusion of a candidate rule. This node becomes the last node of the candidate rule.
2. Compose the preceding nodes according to the relative order of nodes computed by "Computing relative node order algorithm" (Fig. 7). Concatenate "G" into the <Rule ID> to make it distinct from existing rules, "G' symbolizing system-generated rule.
3. Present the rule to the user for confirmation before it is saved as an additional rule in the knowledge base.

Figure 6. Rule generation algorithm

The "Computing relative node order algorithm" is shown in Fig. 7. The structure of this process describes in the Table 1 below.

Table 1. Computing relative node order structure

Step	Current Rule (RUR)	Node Order Queue (NUR)	Rule Used (RUR)	Pre Candidate Node (NUR)	Candidate Node	Rule Stack
1	...	...	...	...	...	...

1. Starting from the rule with the lowest RUR, pick a rule and place it in CurrentRule. Obtain all the nodes and enter them to NodeOrderQueue.
2. Starting from the first node of NodeOrderQueue. If the node is shared by some other rule, find the rule and enter it to RuleUsed if it is not already in RuleUsed or RuleStack. If more than one rule exists, the node picking will be based on NUR with the lowest one picked first. Get all nodes in the previous order of the node being processed and node itself from all rules in the PreCandidateNode. Remove current node from NodeOrderQueue. Enter the node from PreCandidateNode to the CandidateNode if there's no exists such the node and base on its order. If the order is the same, the node of lower NUR will be picked.
3. If the rule in RuleUsed has no next nodes, push it into RuleStack and remove it from RuleUsed.
4. If the rule in CurrentRule which has the node being processed has no next nodes, push it into RuleStack, and remove it from CurrentRule.

5. Get a rule from RuleUsed which has the lowest RUR and place it in CurrentRule. If RuleUsed is empty, but there still are rules in the knowledge base that do not appear in RuleStack, pick a rule from them with the lowest RUR. Go to step 2 and do all steps until all rules in the knowledge base appear in RuleStack.

Figure 7. Computing relative node order algorithm

Computing relative node order process is takes a long time as it checks each rule in *n* rules of knowledge base. Also if at a node being processed there're rules sharing this node, the process has to compare each nodes in each shared rules. Thus, the time complexity of this process = O (n × number of rules sharing a node × number of nodes in each shared rules).

As a matter of fact, Fig. 7 also obtains the relative order of rules from the RuleStack by popping the rules from the stack.

During rule generation process we can perform node generation as well. The last variable of a candidate node is obtained from the most important variable. We generate a node by combining the variables according to the relative order of the existing variables in the nodes. The node generations algorithm describe in the Fig. 8.

1. For each of the most important variable, produced by variable analysis select one as the last variable of a candidate node.
2. Compose the preceding variables according to the relative order of computed by "Computing relative variable order algorithm". Concatenate "G" into the <Node ID> to make it distinct from existing variables, "G" symbolizing system-generated variable.
3. Present the rule to the user for confirmation before it is saved as an additional node in the generated rule in the knowledge base.

Figure 8. Node generation algorithm

The "Computing relative variable order algorithm" is similar with "Computing relative node order algorithm", except it's for the node.

2.3 Knowledge Building

Fig. 9 describes the algorithm of knowledge building.

I. If a proper node found, the system will give the user one of the following choices.
   1. If she disagrees with the current node, she can create a new node at the top level of the knowledge base. Automatically the system will create a new rule for this new node. The disagreed variables from the case posted by the user will be saved under the new node position.
   2. If she agrees with one or more variables in the current node, but disagrees with others; she can create an exception node which containing the disagreed ones. The agreed variables from the case posted by the user will be saved in the Credit field of the Node Structure under the old node position (the parent node), while the disagreed variables will be saved under the new node position.
   3. She can choose to continue traversing the knowledge base without any changes. Here, Variable-Centered Rule Structure won't get anything from the user. The system works like an inferencing process in this case and allows the user to verify the knowledge base, a verification-on-the-flay mechanism.

II. If no nodes are found, the system will ask the user to create a new node at the top level of knowledge base and the case posted by the user will be saved under the new node position.

Figure 9. Knowledge building algorithm

Finding a proper node in the knowledge base is done by the algorithm of Fig. 10.

1. Find all nodes with the same variable ID and conclusion ID as the cases provided by the user from Node Structure and Conclusion tables. Remove redundant IDs. Set them as pre-candidate nodes. If more than one node found in the pre-candidate nodes, then follow the priority as follows to find the candidate nodes.
2. First priority, find perfect nodes, which are nodes containing the same variable IDs and values along with the same conclusion values. If a perfect node found, save it into the candidate nodes.
3. Second priority; find the nodes that contain same variable IDs (without the same values) along with the same conclusion values. If such a node found, save it into the candidate nodes.
4. Third priority; find the nodes that, at least contain same variable ID along with at least one same conclusion value. If a node likes this found, save it into the candidate nodes.
5. Fourth priority; find the nodes that contain at least one same variable ID. If such a node found, save it into the candidate nodes.
6. Fifth; find the nodes that contain at least one same conclusion value. If a node likes this found, save it into the candidate nodes.

7. If no nodes are found in the pre-candidate nodes, the system will tell the user that the case she enters doesn't match any cases in the knowledge base.
8. The above process stops where the user is satisfied with a node or the user decides to stop/start the process again.

Figure 10. Finding proper node algorithm

If any candidate nodes found, the system presents to the user with the options. She then can choose either or both the exception cases occurred as the new case for the new node. The condition being created in the new node is the condition that is unavailable in the old case. The system will process these unavailable conditions to create a new node by the algorithm of Fig. 11.

1. If a condition is available in the old case, but unavailable in the new case then the new node being created will has the negation of such condition.
2. If a condition is unavailable in the old case, but available in the new case then the new node being created will has such condition.
3. If no candidate nodes found, or the user wants to create a new node at the top level; then the new node being created will has all conditions in the new case.

Figure 11. Node creating algorithm

Every new node creates update information of the occurrence of values in the Node Structure. While a new node is being created, the variable value exists in both old case and new case will be saved in the Node Structure under the old case position (i.e., Credit field). If it's a new top level node, the value from the case being posted by the user is only saves under the new case position.

When the user creates a node, it means she creates a new rule, because that new node is the node which does not have a child. The name of the new rule is obtained from the last node, so that new node becomes the name of the new rule.

The building process finishes after the user is satisfied with a node as the final conclusion, or the user decides to stop/start the process again. During the process, the user can repetitively perform the above actions to make improve the existing knowledge base or traverse the nodes to verify the inferencing process. The latter implies that our system also performs verification-on-the-fly, like RDR does, while RBS does not. This will guarantee the knowledge base to be the one that the user wants it to be.

To empower the knowledge building process, when the user is entering her case, the system will guide her about the usage degree of the rules/nodes/variables in the knowledge base. With this guideline, the user knows easily the status of each variable/conclusion in her case being entered: whether these values exist in the knowledge base or the status of the most usage variable. This information will help her to decide from which variable/node/rules she can go through and update the knowledge base.

2.4 Knowledge Inferencing

Knowledge inferencing is simply a knowledge building process without actions done by the user. The user inputs cases and the system goes through the traversing process, when VCIRS already performs a simple forward chaining.

2.4.1 RDR Inferencing Mechanism

When the user provides a case as asking the system to obtain the result, the inferencing process starts to find a proper node. Then, if a proper node found, the system will continue to traverse the rule tree and ask the user for confirmation of each value like the forward chaining approach until a conclusion is fired when the value is matched or no result obtained when the value unmatched. This proper node finding algorithm is the same as described in Fig. 10. Fig. 12 describes the RDR inferencing mechanism.

1. Process the case posted by the user to find a proper node by the proper node finding algorithm. The result is saved in the candidate nodes. If there is no result from this step, process stop here. Step 2 will continue only if there's at least one node in the candidate nodes.
2. Starting from the first proper node in the candidate nodes, obtain the <Rule ID> and <Node Order> belong to this <Node ID> node.
3. If the proper node is the perfect node, fire the proper node, record the conclusion in the ConclusionQueue and continue to step 4. If there's a value unconfirmed, ask the user to confirm. If the value posted by the user is unmatched, start again from step 2 with the next proper node. Record every event into EventLog.
4. Check the next node based on the next <Node Order> of the current <Rule ID>, and ask the user to confirm the values in that node. If all values confirmed, fire that node, and record the conclusion at that node in the ConclusionQueue. Repeat this step, until there's no next node at <Rule ID>, as long as the value of node is matched with the value posted by the user. If unmatched values occurred, start again from step 2 with the next proper node. If the last node of

<Rule ID> is successful to fire, present the last conclusion to the user as the final conclusion along with all conclusions from ConclusionQueue as the intermediate conclusions. Ask the user if she wants to continue to find more conclusions, if there are other nodes in the candidate nodes. If the user agrees, start again from step 2 with the next proper node to find more conclusions. If the user does not want, inferencing process can stop here. Record every event into EventLog.
5. If no node fired, tell the user that there's no node matched with her case.

Figure 12. RDR inferencing in VCIRS

From the ConclusionQueue we can answer: What are the results of the inferencing. The answer of: How and Why can be obtained from the EventLog.

2.4.2 RBS Inferencing Mechanism

This inferencing process is based on the structures in the Rule Structure and Node Structure. As stated before, a node in the Node Structure is similar to an RBS rule, which contains a clause and a conclusion part. A rule in the Rule Structure is also similar to an RBS rule, whereas big RBS rule. An RBS rule created from a node is obtained by obtaining the node name, variable names and their values from the Node Structure, and the conclusion from the corresponding Conclusion Node. An RBS rule also can be created from the Rule Structure. It is obtained similarly to rule creation from the Node Structure, except the name and the conclusion of each rule is obtained from the last node of a given rule. The RBS rules created from both Node and Rule Structures are combined together to do RBS inferencing.

2.5 Knowledge Base Transformation

The Node Structure in VCIRS has the similarity with the structures of the rule base in an RBS. We thus can transform the VCIRS knowledge base into an RBS rule base. The structure in the Rule Structure uses the Node Structure. Thus, it also can be transformed into an RBS rule base, except the name and the conclusion of each rule is obtained from the last node of a given rule.

3. Conclusions

The contribution of the paper thus can be summarized as: we have proposed and implemented a Variable-Centered Intelligent Rule System (VCIRS), which uses a variable-centered node and rule structure to organize the rule base so that easy knowledge building, powerful knowledge inferencing, and evolutional improvement of the system performance can be obtained at the same time.

4. References

[1] Compton, P., G. Edwards, B. Kang, L. Lazarus, R. Malor, T. Menzies, P. Preston, A. Srinivasan and C. Sammut, "Ripple down rules: possibilities and limitations," *Proc. of 6th Bannf AAAI Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada, 1991.

[2] Compton, P., Kang, B., Preston, P. and Mulholland, M., "Knowledge acquisition without analysis," *Proc. of European Knowledge Acquisition Workshop*, Springer Verlag, 1993.

[3] Edwards, G., Compton, P., Malor, R., Srinivasan and A., Lazarus, L., "PEIRS: a pathologist maintained Expert System for the interpretation of chemical pathology repots," *Pathology 25*, 1993, pp. 27-34.

[4] Forgy, C.L., "Rete: a fast algorithm for the many pattern/many object pattern match problem", *Artificial Intelligence 19* (1982), 17-37.

[5] Ho, V., Wobcke, W. and Compton, P., "EMMA: an e-mail management assistant," in Liu, J., Faltings, B., Zhong, N., Lu, R., Nishida, T. (Eds.), *Proc. of IEEE/WIC International Conference on Intelligent Agent Technology*, Los Alamitos, CA, 2003, pp. 67-74.

[6] Kang, B., P. Compton and P. Preston, "Multiple classification ripple down rules: evaluation and possibilities," *Proc. of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, University of Calgary, 1995.

[7] Mansuri, Y., Kim, J.G., Compton, P. and Sammut, C., "A comparison of a manual knowledge acquisition method and an inductive learning method," *Proc. of the First Australian Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Sydney, University of Sydney, 1991, pp.114-132.

[8] Muggleton, S.D., "An oracle-based approach to constructive induction," *Proc. of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

[9] Suryanto, H. and Compton, P., "Intermediate concept discovery in ripple down rule knowledge bases," *Proc. of the 2002 Pacific Rim Knowledge Acquisition Workshop (PKAW 2002)*, in conjunction with the Seventh Pacific Rim International Conference on Artificial Intelligence (PRICAI 2002), Tokyo, Japan, pp. 233-245.

[10] Suryanto, H. and Compton, P., "Invented predicates to reduce knowledge acquisition," *Proc. of the 14th International Conference on Knowledge Engineering and Knowledge*

Management (EKAW04), 5-8th October 2004 - Whittlebury Hall, Northamptonshire, UK, 2004, pp. 293-306.