國立台灣科技大學
資訊工程系

碩士學位論文

以變數爲中心的智慧型規則系統

A Variable-Centered Intelligent Rule System

M.M. Irfan Subakti
司馬伊凡
M9215801

指導教授：何正信

中華民國 九十四 年 六 月 二十四 日

碩士學位論文指導教授推薦書

本校 資訊工程 研究所 學程 MM Irfan S 君

所提之論文 A Variable-Centered Intelligent Rule System

係由本人指導撰述，同意提付審查。

指導教授

九十四年六月二十四日

碩士學位考試委員會審定書

本校 資訊工程 系（所）＿＿＿＿＿＿學程 MM Irfan S 君

所提論文 A Variable-Centered Intelligent Rule System

經本委員會審定通過，特此證明。

學位考試委員會

委　　員：＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿

指 導 教 授：＿＿＿＿＿＿＿＿

學 程 主 任：＿＿＿＿＿＿＿＿

系主任（所長）：＿＿＿＿＿＿＿＿

中華民國　九十四　年　六　月　二十四　日

ABSTRACT

A Rule-based System (RBS) is a good system to get the answer of What, How, and Why questions from the rule base 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 not 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 does not ever need to examine the rule base 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 hierarchy. 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 lacks the flexibility of question answering and explanation accrued from powerful inference.

A Variable-Centered Intelligent Rule System (VCIRS) is proposed in this thesis. It hybridizes RBS and RDR. The system architecture is adapted from RBS and obtains advantages from RDR. This system organizes the rule base 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" in VCIRS 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

ACKNOWLEDGEMENT

First of all, I would like to appreciate sincerely to my supervisor, Professor Ho Cheng-Seen, for his support and guidance during my research.

My deepest gratitude to my Mom and Pa, who always praying for me, for them this thesis I would dedicate to.

Finally, I would like to thank to all of my lab mates in Artificial Intelligent Laboratory, Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, for their encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
Introduction

1.1 Background

A knowledge-based system with knowledge structured by rules is called Rule-based System (RBS), alternately called an expert system. The place that stores rules is called a knowledge base (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 [Comp91]. 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.

Ripple Down Rules (RDR) is a KA method which constrains the interactions between the expert and a shell to acquire only correct knowledge [Kang95]. 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 [Comp93]. 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 [Comp91]. And this is where RDR systems gain their power compared to traditional RBS [Ho74]. The user does not ever need to examine the rule base 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.* [Edw93].

1.2 Motivation

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 rule base, 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 thesis. Another motivation is a desire to empower the existing rules in the knowledge base by generating new rules, in contrast to optimizing those rules.

1.3 Problem Specification

RBS is a good system to get the answers of What, How, and Why questions from the rule base 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 rule base. 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.

1.4 Proposed Solution

In this thesis, we propose a structure to organize the rule base so that easy knowledge building, powerful knowledge inferencing, and evolutional improvement of system performance can be obtained at the same time.

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.

1.5 Related Literature

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 [Man91]. Suryanto and Compton [Sur04] 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 [For82]. 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.

In general, updating rules within RBS is not automatically done by the system itself [Sur04]. Suryanto and Compton's method could reduce KA up to 50%, but the system still needs experts during the learning process. The learning technique is based

on Duce's intra-construction and absorption operators [Mug87] and is applied to RDR as incremental KA [Comp89]. Gaines carried out similar studies using his Induct algorithm and produced slightly smaller trees [Gain89]. Another example, Garvan-ES1 is an RDR tree which can generate the data needed for rule induction [Comp89, Comp90, Call91]. One further example, RDR has now been used to build a large medical expert system, PEIRS (Pathology Expert Interpretative Reporting System), which interprets chemical pathology reports at St. Vincent's Hospital Sydney [Comp92, Edw93, Pres93]. PEIRS now contains 2300 rules and covers much of chemical pathology, making it a very large medical expert system. Knowledge in PEIRS has been added entirely by experts without any assistance of knowledge engineers, nor any knowledge engineering or programming skills.

Suryanto and Compton proposed to do intermediate concept discovery in RDR knowledge bases [Sur02]. 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 rule bases, an example (i.e., a case) may have a classification resulting from multiple conclusions in the rule base. 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 [Mug87]. 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.

Another paper talking about rule base optimization was Beydoun and Hoffmann [Bey97]. It's about nested RDR, a variant of RDR which allows the expert to define intermediate concepts explicitly. Similar to this, another variant of RDR has been proposed based on exception structures, which tend to produce compact representations [Cat92, Gain92, Kivi93, Sche96, Siro93]. However, it is not guaranteed that this secondary goal, producing more compact representations is achieved.

1.6 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we give some background about Rule-based Systems and Ripple Down Rules. In Chapter 3, we describe our proposed method, a Variable-Centered Intelligent Rule System (VCIRS). We implement VCIRS in Chapter 4, and then perform its evaluation in Chapter 5. Finally we conclude our research and provide some discussions in Chapter 6.

Chapter 2
Background

2.1 Rule-based Systems

2.1.1 Definition

A knowledge-based System is defined as a computerized advising/suggesting program which attempts to imitate or replace the reasoning process as well as the knowledge of the experts in solving specific problems. A knowledge-based system with knowledge structured by rules is called a Rule-Based System (RBS) [Igni91]. An RBS is alternatively named an expert system. Thus, an RBS contains two features: (1) Knowledge (rule-based) and (2) Inferencing.

Knowledge can refer to any of the following: clear and certain perceptions about something, understanding, learning, and all perceptions by mind, practical experiences, skills, cognizance, recognition, and organized information applied in problem solving. Inferencing is a special feature of an expert system, making it able to perform reasoning. Facts, input and saved in a knowledge-based system drive the Inferencing process. Inferencing is performed by a component called inference engine, which inspects the facts so far and invokes proper rules to derive new facts.

2.1.2 Structure

Turban explained that a rule-based system can be divided into 2 main parts: a development environment and a consultation (runtime) environment (Figure 2.1) [Tur95]. The development environment is used by the expert system builder to build system components and transfer knowledge into the knowledge-base. The consultation environment is used by non experts to obtain the expert's advice.

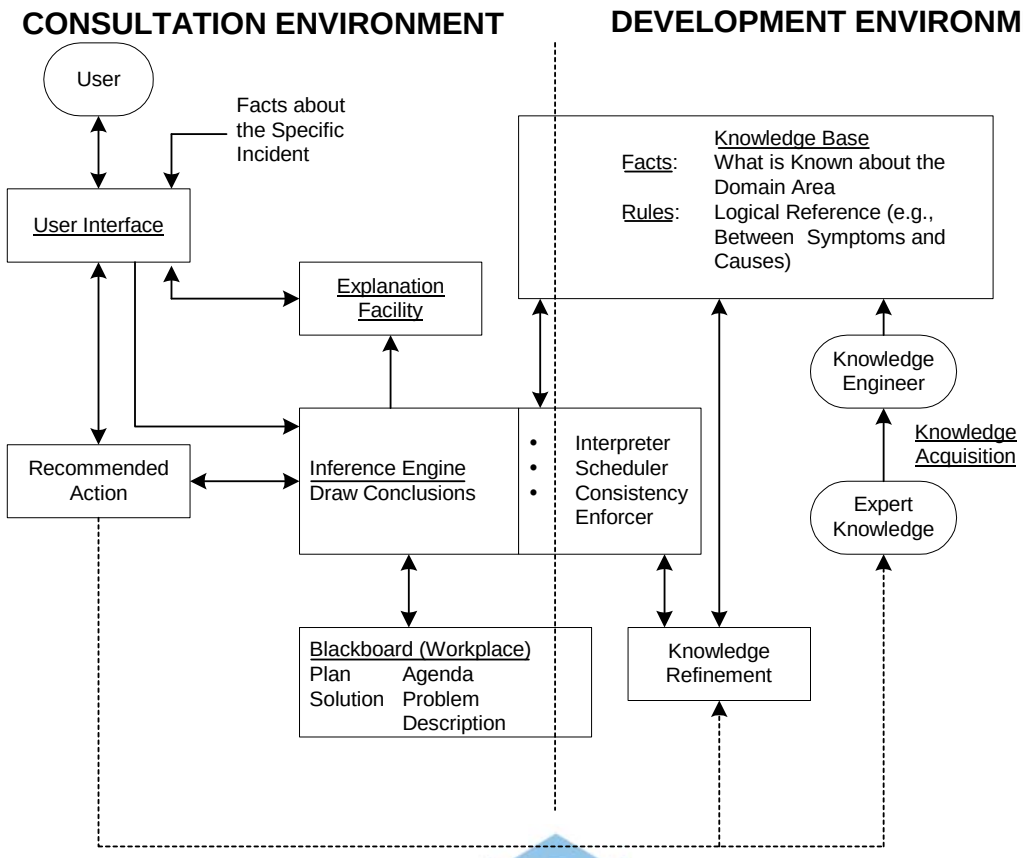


Figure 2.1 Structures diagram of a rule-based system

Typical components in the expert system are:

- Knowledge acquisition subsystem. Knowledge can be obtained from the human experts, textbooks or research reports, by the support of a knowledge engineer (an expert specialized in knowledge acquisition).
- Knowledge base. Two kinds of knowledge base are facts (i.e., situation and theory) and heuristics or rules.
- Inference engine. It's the brain of an expert system, alternately called control structure or the rule interpreter (in the rule-based expert system). It's a computer program which has a methodology for reasoning about information in the knowledge base and in the "blackboard (workplace)", and it is used for formulating the conclusion. It has three main elements: interpreter, scheduler, and consistency enforcer.

- Blackboard (workplace). It's a temporary memory for processing plan, agenda, solution, and problem description obtained from the knowledge base during consultation session.
- Users. Typical users include: (1) Clients (i.e., non experts) who want an advice. Here, the expert system behaves as a consultant or advisor. (2) Learners for learning how the expert system solves problems. Here, the expert system behaves as an instructor. (3) Expert system builder who want to improve her knowledge base. Here, the expert system behaves as a partner. (4) Experts. Here, the expert system behaves as a colleague or an assistant.
- User interface. The expert system should be user friendly and problem oriented.
- Explanation subsystem (justifier). It's the capability of truth tracing from the conclusion of its sources. It's crucial for expertise transformation and problem solving. This component is able to trace the truth and to explain the expert system behavior, interactively, answering the questions like: Why was a certain question asked by the expert system? How was a certain conclusion reached? Why was a certain alternative rejected? What is the plan to reach the solution? And what remains to be established before a final diagnosis can be determined?
- Knowledge refining system. With this component, the expert is able to analyze the performance of the expert system, learn from it, and improve it for next consultation.

The most difficult thing in developing a knowledge-based system is knowledge acquisition. Once knowledge is acquired, it needs to be organized. The place that hosts the knowledge is called a knowledge base. A knowledge base can be organized in several different configurations to facilitate fast inferencing.

The inferencing process is directed by some strategies of rule interpretation:

- Forward chaining: The premise clauses of a rule are matched against the fact data, a match causing the process to assert the conclusion.
- Backward chaining: The current goal is matched against the fact of the conclusion of some rules; a match causing the process to determine whether the premise clauses match the fact data.

In addition to inference strategy, Certainty Factors (CF) is usually added to the program to improve its harness of uncertainty in both knowledge and fact data [Baur90].

2.2 Ripple Down Rules

2.2.1 Single Classification Ripple Down Rules

Ripple Down Rules (RDR) are built as a tree with each rule as a node with 2 branches (i.e., binary tree) specifying whether the rule is satisfied or not, by the data being considered [Comp91]. Any new rule that is added in response to a wrong interpretation is attached to the branch at which the expert system terminated, thus making a new node. For example, in a one rule expert system, if the rule gives an interpretation which the expert thinks is wrong, a new rule will be attached to the satisfied branch of the first rule. On the other hand if the one rule expert system fails to give an interpretation the new rule will be attached to the unsatisfied branch. The tree then evolves over time as the expert corrects wrong interpretations. Note that each node of the tree is a rule of any desired complexity and that the interpretation produced from the tree is that from the last rule that was satisfied by the data.

For example, in Figure 2.2, the expert must choose either or both the conditions

```
NOT Dark Spot
Black Stripe
```

as conditions in the rule and can optionally choose any of the common conditions to make the rule intelligible. Such a rule is guaranteed to work on the new case but not the old case, so no further checking is required or relevant. Note that for a case that has not satisfied any rule, the difference list includes all the conditions that are true for the case and the rule can use any of these conditions. However, the rule is not evaluated until all previous rules have been evaluated.

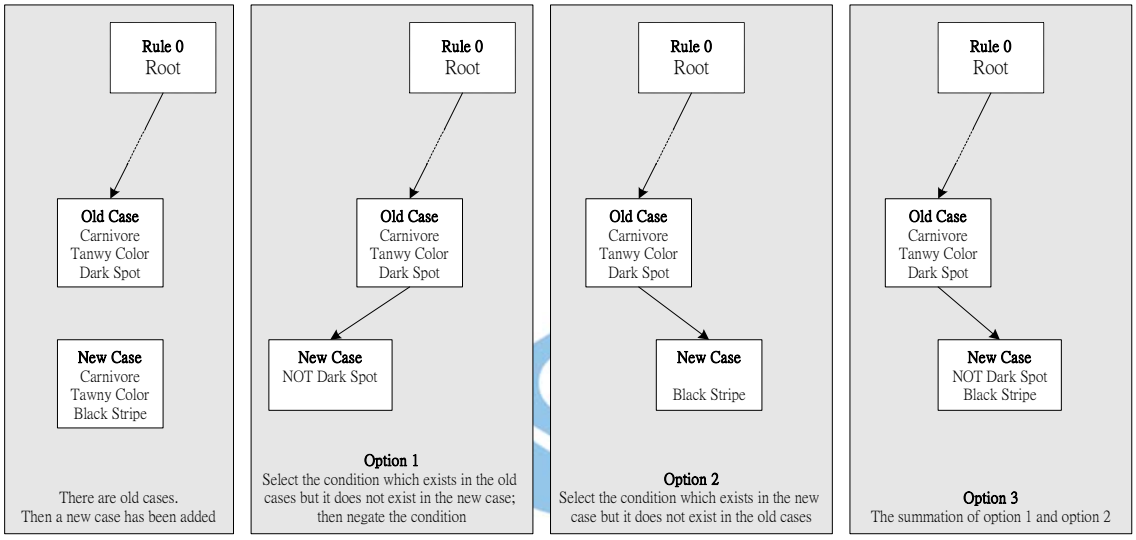


Figure 2.2 Cases and exceptions

Although RDR works well, it is only suitable for the problem which provides a single classification for a set of data. PEIRS, an application of the RDR approach (single classification), faced problem that a patient may have multiple independent diseases [Edw93]. PEIRS currently handles this problem by treating such situations as compound diseases. However, this could exponentially increase the KA task. A possible solution would be to separate domains and build an independent KB for each domain. However, in many domains it is not easy to partition domains and a clumsy work-around would be required. The Multi Classification RDR system described below deals with this problem.

2.2.2 Multiple Classification Ripple Down Rules

Kang *et al.* proposed Multiple Classification Ripple Down Rules (MCRDR) [Kang95]. The aim of MCRDR is to preserve the advantages and essential strategy of RDR in dealing with multiple independent classifications. MCRDR, like RDR, is based on the assumption that the knowledge an expert provides is essentially a justification for a conclusion in a particular context. A major component of the context is the case which has been given a wrong classification and how this differs from other cases for which the classification is correct. As we shall see, the context in MCRDR is preserved differently and only includes rules that have been satisfied by the data; the validation extends to differentiating the new case from a range of different cases.

The RDR inference is based on searching the KB represented as a decision list with each decision possibly refined again by another decision list. Once a rule is satisfied no rules below it are evaluated. In contrast, MCRDR evaluates all the rules in the first level of KB. It then evaluates the rules at the next level of refinement for each rule that is satisfied at the top level and so on. The process stops when there are no more children to evaluate or when none of these rules can be satisfied by the case in hand. It thus ends up with multiple paths, with each path representing a particular refinement sequence and hence multiple conclusions. The approach of MCRDR has been evaluated in simulation studies where the human expert is replaced by a simulated expert. MCRDR may provide a basis for building a general solver for a range of problems beyond classification.

The structure of an MCRDR knowledge base can be drawn as an n-ary tree with each node representing a rule. An example application, which employs MCRDR, was EMMA (E-Mail Management Assistant) [Ho74]. With this type of rule base, a case has a classification resulting from multiple conclusions in the rule base. These conclusions

are found by following every path in the MCRDR tree to the most specific node applicable to the example. The classification is then the set of conclusions from all such nodes. For example, consider the MCRDR trees shown in Figure 2.3. The root node is a dummy node giving no conclusion. Rules 1 to 5 are the rules in the tree that are not defined as exceptions to other rules, while Rules 6 and 7 are exceptions to Rule 1. Consider a case with features {*a, b, c, e, f, x*} (which might correspond to the sender of a message or keywords in the body of a message). At the top level of the tree, Rules 1, 3 and 5 are applicable to the example. However, Rule 7, which is an exception to Rule 1, also applies to the example. So the most specific rules applying to the example are Rules 7, 3 and 5, giving the classification {*CRC, Friends*}. Note how in this example, there are two rules applying to the message that give the conclusion *Friends*, but this makes no difference to the final classification.

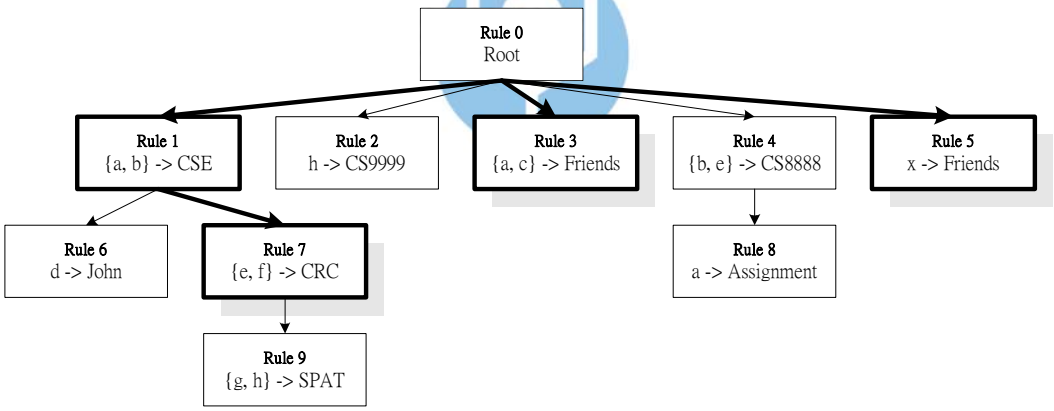


Figure 2.3 Example MCRDR rule base

2.2.3 Important Properties of RDR Systems

First, all RDR systems provide extensive support to the user in defining rules, and this is where RDR systems gain their power compared to traditional Rule-based expert systems. The user does not ever need to examine the rule base in order to define new rules: the user only needs to be able to define a new rule that correctly classifies a

given case, and the system can determine where the rule should be placed in the hierarchy. In an RDR system, rules are never modified or removed, only refined or added. Refinement 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. Rules are always defined in the context of a specific example and in relation to other "relevant" cases, and are not meant to be absolute. For RBS it is assumed that the context is the sequence of rules which have been evaluated to give a certain conclusion [Comp93]. To define a new rule "in context," the user has only to identify the features of a case that distinguish it from the other cases that should be classified differently. The RDR system can determine which cases need to be considered by checking the rule base for prior cases that are also covered by the proposed rule. To do this, the system maintains for each rule the case that was used when the user created the rule (called the "cornerstone case"). The cornerstone cases of potentially conflicting rules can be presented to the user as the new rule is being defined, prompting the user to either accept the new conclusion as also applying to the existing cornerstone case, or to identify additional features to further distinguish the current case. The process continues iteratively until the user accepts the new rule and its consequences. This makes the task of defining rules much simpler than if the user is required to define a rule base independently of the context of a specific case (as is required in traditional Rule-based expert systems and in the rule bases of current e-mail filtering systems). In this way, an RDR system can be used by the "end users" rather than only by knowledge engineers.

Consider again the example of MCRDR tree shown in Figure 2.3. A message with features {*a, c, x*} is classified as {*Friends*}. Suppose the user wishes to change the classification to {*ARC*}; she needs to determine additional features of the example that can lead to the new conclusion. By using the RDR system, the user is certain that the

selected features *y* and *z* do not result in incorrect conclusions being generated using other rules. For example, as the feature *y* of the message is selected, the user is shown any cornerstone cases that satisfy the conditions of the new rule, including *y*: note that there may be such cases even though *y* does not occur in the conditions of any existing rule. The new conclusion {*ARC*} applies to such cases, so the user is prompted to determine whether or not this is the intended behavior; if it is not, the user must identify additional features (e.g., *z*) that further discriminate the current message from these cornerstone cases. Finally, two instances of the rule 'if *y* and *z* then *ARC*' are added as exceptions to Rules 3 and 5 (so that both conclusions are overridden by more specific rules), as shown in Figure 2.4. Now, the message can be classified correctly.

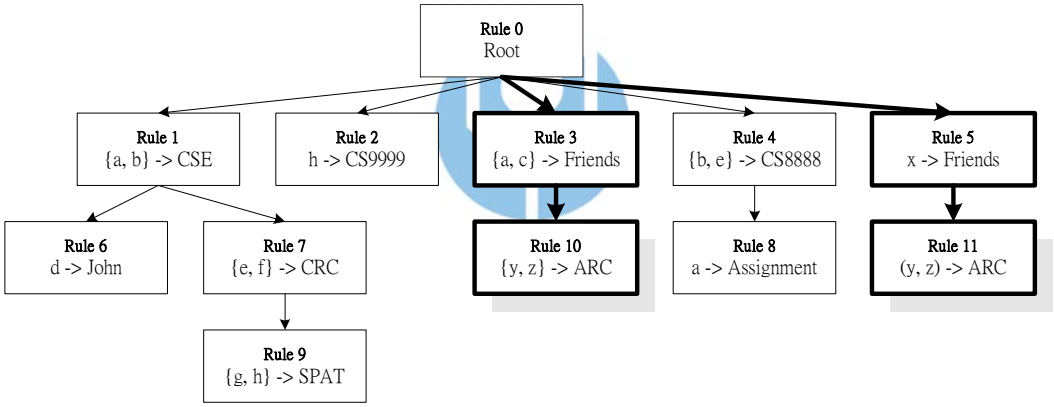


Figure 2.4 MCRDR rule base after refinement

In summary, the main advantages of RDR as a KA technique are the high degree of accuracy, the validation of conclusions that the system can provide, and the ease of constructing the rule base. This makes RDR systems suitable in domains where accuracy, explicitness of reasoning, and justification of conclusions are essential. One typical application domain where RDR systems have been successfully applied is in medical diagnosis [Comp90].

The high degree of accuracy is attained because the user refines the rule base to maintain consistency whenever necessary, so the system always classifies cases according to the user's (current) intentions (as the rule base being constructed, the user can change the classification of previously classified examples by creating exception rules). Total accuracy is not guaranteed, however, because the rules defined by the user are typically over-general, in the sense that the rules also apply to unseen cases in perhaps unforeseen ways. However, in practice, since rules apply only in specific contexts, misclassification applies to a relatively small proportion of unseen cases. More usual in practice is that the user's rule base as a whole covers too precisely the set of existing cases, and the RDR system tends to be "conservative" and fails to generalize to unseen cases. Thus in comparison to machine learning systems, RDR systems gain high accuracy (precision) at the expense of coverage (recall) of a set of examples, while typical machine learning algorithms have higher coverage but lower accuracy.

As mentioned above, rules are never deleted from an RDR rule base. To achieve the effect of deleting a rule, a rule may be "stopped," which means that any conclusions that would normally be generated using the rule are ignored, but *not* those obtained from its exceptions (and their exceptions, etc.). One potential disadvantage of this approach is that an RDR rule base can grow to contain a large number of redundant rules, especially in dynamic domains where the user may often want to stop the application of existing rules. However, with the hierarchical structure of an RDR rule base, removing redundant rules is a nontrivial operation [Wada02], and in practice, has not proven to be necessary. Table 2.1 summarizes the terminology commonly used in MCRDR.

Table 2.1 Terminology commonly used in MCRDR

Terminology	Explanation
Case	Consist of attributes and values which are first processed by predefined functions; a rule condition can be a raw attribute or the result of a function evaluation.
Conclusion is fired	When all conditions of the rule in the current node are satisfied by a case and all conditions belonging to the ancestor nodes are satisfied too.
Conclusion is given	When a conclusion is fired but none of the conclusions of its child nodes are fired.
Default conclusion	An empty conclusion; this conclusion is given only when no other conclusions are given.
Stopping rule	A rule with an empty conclusion; if this conclusion is given then no other conclusions can be given from this path through the knowledge base. It is the dummy conclusion that stops a parent conclusion from being given.
Cornerstone case	A case which prompts the addition of a rule. These cases are saved and used to ensure that any changes made are incremental improvements. The conclusion of a rule added later should not apply to an earlier cornerstone case unless the expert allows an explicit override or addition.

Chapter 3
System Description

Figure 3.1 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.

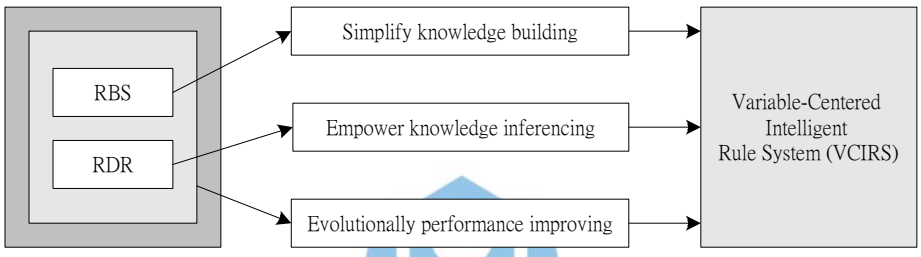


Figure 3.1 Proposed method

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.

This chapter starts from significant terms definition, to the system architecture and finally to the description of how the system operates.

3.1 Term Definition

The following terms are heavily used in the thesis. This section gives them the clear definitions to avoid misunderstanding and pave a convenience way for discussion. We present these terms in BNF. Figure 3.2 helps us captures the conceptual relationship between the terms.

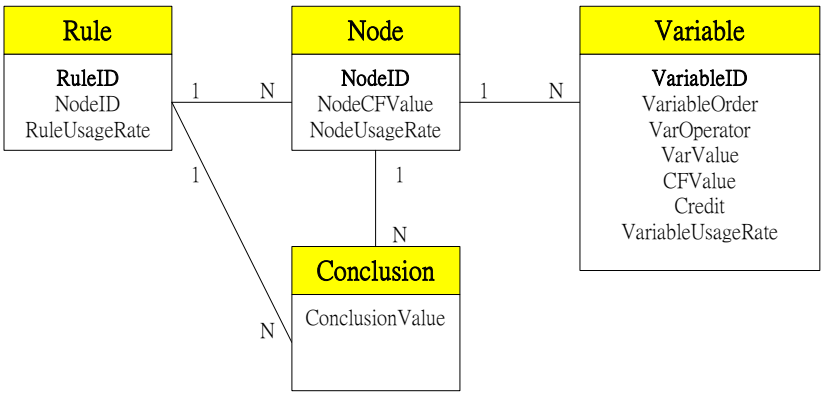


Figure 3.2 Terms relationship

A rule is the sequence of nodes. It may have one or more nodes. A node is the sequence of variables. It may have one or more variables. A variable may have varied values. A node or a rule may have one or more conclusions.

A. Rule Based Systems

/** **Rule Base**: A rule base consists of one or more rules. **/

```
<Rule Base> ::= <Rules>
<Rules> ::= <Rule> | <Rule> <Rules>
```

/** **Rule**: A rule is the fundamental element for building a rule base. **/

```
<Rule> ::= <Rule ID> <Rule CF> IF <Clauses> THEN <Conclusions>
<Rule ID> ::= ALPHANUMERIC
<Rule CF> ::= [0..100]
<Clauses> ::= <Clause> | <Clause> <Clause Operator> <Clauses>
<Clause Operator> ::= AND
<Clause> ::= <Variable> <Operator> <Value> <Clause CF>
<Variable> ::= ALPHANUMERIC
<Operator> ::= “<” | “<=” | “>” | ”>=” | “=” | “<>”
<Value> ::= ALPHANUMERIC | NUMERIC
<Clause CF> ::= [0..100]
<Conclusions> ::= <Conclusion> | <Conclusion> <Conclusion Operator>
                  <Conclusions>
<Conclusion> ::= ALPHANUMERIC
<Conclusion Operator> ::= AND
```

B. Variable-Centered Rule Structure

Remarks:

A **rule** is the sequence of one or more nodes.

A rule in VCIRS is similar to a sequence of rules in RBS with one exception that the conclusion of the rule is the conclusion belonging to the last node. A **node** is similar to a rule in RBS. It is the sequence of one or more variables and consists of one or more conclusions.

A first example of a rule in Figure 2.4 is Rule *John#1* which has 2 nodes: node *Rule 1* and node *Rule 6*. Node *Rule 1* has two variables, i.e., *a* and *b*. Node *Rule 6* has one variable *d*. Rule *John#1* only has one conclusion, i.e., *John*.

A second example is Rule *CSE#1* which has only 1 node, node *Rule 1*. Node *Rule 1* has two variables, i.e. *a* and *b*; and Rule *CSE#1* has one conclusion, i.e., *CSE*.

/** **Rule Structure** **/

```
<Rule Structure> ::= <Rules>
<Rules> ::= <Rule> | <Rule> <Rules>
<Rule> ::= <Rule ID> <NodeIDOrders>
<NodeIDOrders> ::= <Node ID> <Node Order> | <Node ID> <Node Order>
                   <NodeIDOrders>
<Rule ID> ::= ALPHANUMERIC
<Node ID> ::= ALPHANUMERIC
<Node Order> ::= NUMERIC
```

/** <Node ID> refers to the <Node ID> in the Node Structure **/

/** **Node Structure** **/

```
<Nodes> ::= <Node> | <Node> <Nodes>
<Node> ::= <Parent Node ID> <Node ID> <Node CF Value> <Variables>
           <Conclusions> <Credit> <VUR> |
           <Parent Node ID <Node ID> <Node CF Value> <Variables>
           <Conclusions> <Credit> <VUR> <Node>
<Parent Node ID> ::= <Node ID>
<Node ID> ::= ALPHANUMERIC
<Node CF Value> ::= [0..100]
<Variables> ::= <Variable> | <Variable> <Variables>
<Variable> ::= <Variable ID> <Variable Order> <Value Part>
<Credit> ::= NUMERIC
```

/** Credit is the occurrence of accessing a variable in a node **/

```
<VUR> ::= NUMERIC
```

/** VUR = Variable Usage Rate is the usage rate of a variable in a node **/

```
<Variable ID> ::= ALPHANUMERIC
<Variable Order> ::= NUMERIC
<Conclusions> ::= <Conclusion> | <Conclusion> <Conclusions>
<Conclusion> ::= ALPHANUMERIC
```

/** **Conclusion**: A conclusion consists of one or more values. **/

```
<Value Part> ::= <Operator> <Value> <CF Value>
<Operator> ::= “<” | “<=” | “>” | ”>=” | “=” | “<>”
<Value> ::= ALPHANUMERIC | NUMERIC
<CF Value> ::= [0..100]
```

3.2 System Architecture

Figure 3.4 shows the architecture of VCIRS which adapts the traditional RBS architecture (Figure 3.3). A new module called Refinement Module is added to perform 3 tasks: variable analysis, value analysis, and rule generation.

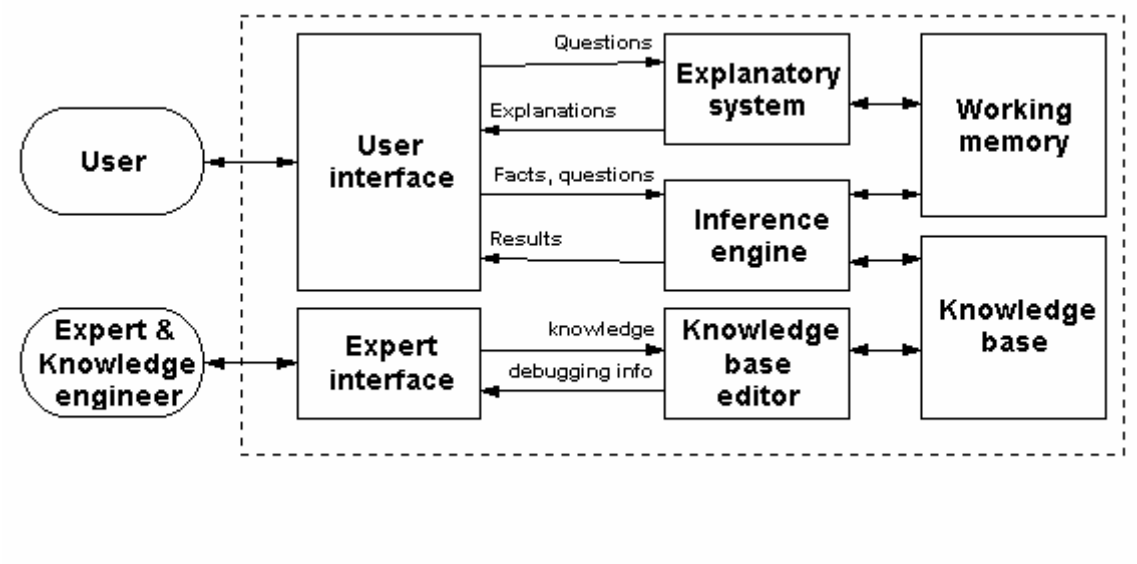


Figure 3.3 Traditional RBS architecture

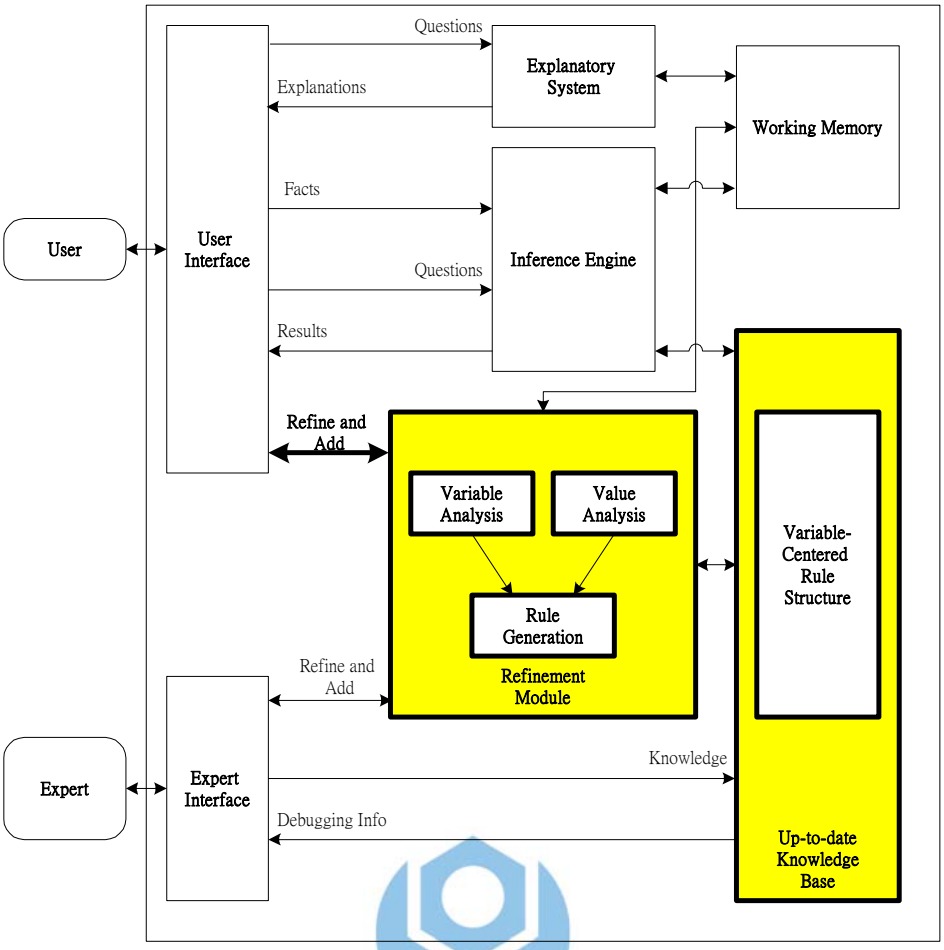


Figure 3.4 VCIRS architecture

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 chapter describes the system in the algorithmic level. For implementation details, see Appendix E.

3.3 Variable-Centered Rule Structure

This structure contains two structures, namely a Node Structure and a Rule Structure. A Node Structure records cases presented by the user and calculates the occurrence of each case. It's similar to a rule in the rule base of RBS. A Rule Structure records a sequence of nodes represented by the Node Structure. Each node in the knowledge base has it order, i.e., at least one node for a rule. Both structures are detailed below.

3.3.1 Node Structure

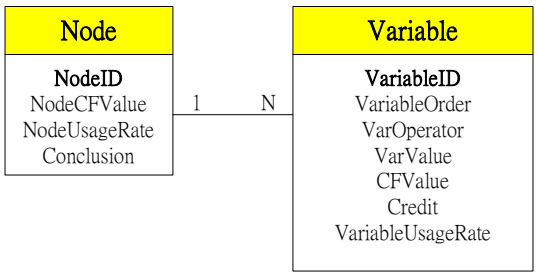


Figure 3.5 Node Structure

Figure 3.5 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 Figure 3.6.

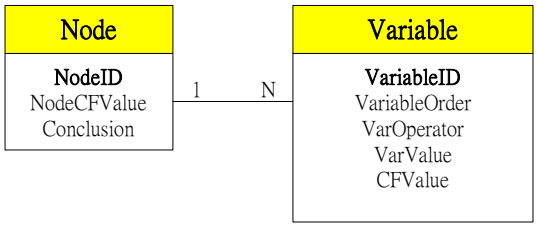


Figure 3.6 Case fields

VCIRS use a `<Variable ID>` as an ID for a variable. This ID is unique, it means once a variable ID is saved, the system will no longer save different variables with the same ID.

ID for a node is taken from the first `<Conclusion>`, posted by the user; a `<Node ID>` will be created from that `<Conclusion>`. A node is unique as long as it has the same variables and variable values. If the subsequent node has the same first conclusion name and the same variables, but its variable values are different, then its `<Node ID>` will be the same as the existing `<Node ID>` with an additional serial number.

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.) [Ho74]. 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.

3.3.2 Rule Structure

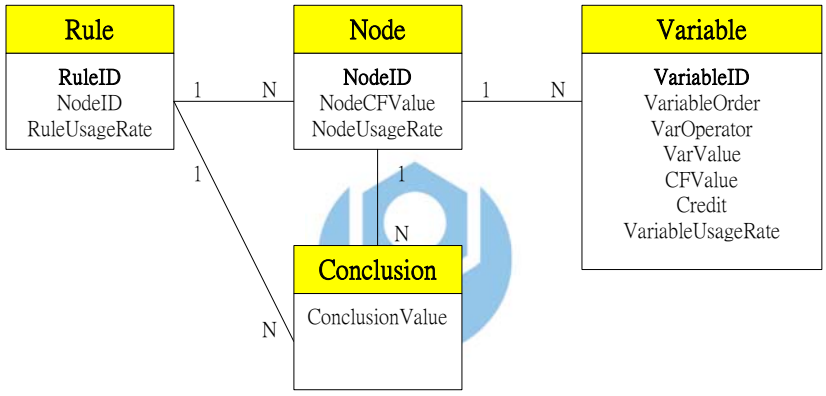


Figure 3.7 The conceptual graph of Rule Structure

Figure 3.7 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.

The ID of a rule is the same as that the lowest `<Node ID>` of each rule. If the subsequent rule has the same candidate `<Rule ID>` the system treats the naming the same like in the node naming: the name of existing rule following by the serial number (#1, #2 …).

3.4 Knowledge Refinement

There are 3 tasks in the Refinement Module: variable analysis, value analysis and rule generation. Variable analysis determines what variables/nodes are most important, while value analysis determines how often a rule/node/variable is used. Rule generation is the result of variable and value analysis. Refining rule (node) in the knowledge base is the creation of an exception rule to correct a misclassification. The system will refine the knowledge base when the user opens a knowledge base, according to the schedule or by the user demand. This design allows the expert to refine and add the knowledge base directly, in addition to invoking Refinement Module just like an ordinary user.

3.4.1 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. Since this analysis strongly depends on the implementation structure, we will postpone the demonstration of analysis until Section 4.2.

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

3.4.2 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.

$$VUR_i = Credit_i \times Weight_i \quad (1)$$

$$NUR_j = \frac{\sum_{1}^{N} VUR_{ij}}{N}, \quad VUR_{ij} \text{ for } i\text{-th variable in node } j \tag{2}$$

$$RUR_k = \frac{\sum_{1}^{N} NUR_{jk}}{N}, \quad NUR_{jk} \text{ for } j\text{-th node in rule } k \tag{3}$$

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.

- $Weight_i = NS_i \times CD_i$ (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)

- $CD_i = \frac{VO_i}{TV}$ (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.

- VO_i = Order of variable *i* in a node (8)

- TV = total variables belong to node (9)

Demonstration of how value analysis is done on a knowledge base will be demonstrated in Section 4.3

3.4.3 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.

Given a knowledge base with *m* rules, where each rule (R) is the sequence of *n* nodes (N), and each node is the sequence of *o* variables (V) and a conclusion (C). The value of *n* for the nodes belonging to a rule may vary as well as the value of *o* for the variables belonging to a node. The order of each variable in a node is meaningful as described in the value analysis above. Using these symbols, a knowledge base can be represented as Figure 3.8.

R_1: $N_{11}(V_{111}\rightarrow..\rightarrow V_{11o};C_{11})\rightarrow N_{12}(V_{121}\rightarrow..\rightarrow V_{12o};C_{12})\rightarrow..\rightarrow N_{1n}(V_{1n1}\rightarrow..\rightarrow V_{1no};C_{1n})$
R_2: $N_{21}(V_{211}\rightarrow..\rightarrow V_{21o};C_{21})\rightarrow N_{22}(V_{221}\rightarrow..\rightarrow V_{22o};C_{22})\rightarrow..\rightarrow N_{2n}(V_{2n1}\rightarrow..\rightarrow V_{2no};C_{2n})$
.
.
R_m: $N_{m1}(V_{m11}\rightarrow..\rightarrow V_{m1o};C_{m1})\rightarrow N_{m2}(V_{m21}\rightarrow..\rightarrow V_{m2o};C_{m2})\rightarrow..\rightarrow N_{mn}(V_{mn1}\rightarrow..\rightarrow V_{mno};C_{mn})$

Figure 3.8 Knowledge base presented by symbols

We generate a rule by combining the nodes in according with the order of the existing nodes in the rules. The Rule generation algorithm is depicted in Figure 3.9.

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” (Figure 3.11). 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 3.9 Rule generation algorithm

The “Computing relative node order algorithm” is shown in Figure 3.11. In the Figure 3.10, “CurrentRule” saves the RuleID being processed; starting from a rule with has the lowest RUR. “NodeOrderQueue” saves the order of nodes from rule being processed in “CurrentRule”, process from the first order. “RuleUsed” records each rule sharing a node in “NodeOrderQueue”. “PreCandidateNode” records nodes belong to rules in “RuleUsed” being compared, before save to “CandidateNode”. Node choosing in “PreCandidateNode” is according to NUR of nodes. The node has the lowest NUR will be picked first. “CandidateNode” records relative node order. “RuleStack” pushes a rule to the stack, after a rule is finished from processing in “CurrentRule”.

Step	Current Rule (RUR)	Node Order Queue (NUR)	Rule Used (RUR)	Pre Candidate Node (NUR)	Candidate Node	Rule Stack
1	...	...	...	...	...	...
2	...	...	...	...	...	...

Figure 3.10 Computing relative node order data structure

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 3.11 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).

An example of rule generation will be demonstrated in Section 4.4. As a matter of fact, Figure 3.11 also obtains the relative order of rules from the RuleStack by poping 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 Figure 3.12.

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" (Figure 3.14). 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 3.12 Node generation algorithm

The "Computing relative variable order algorithm" is shown in Figure 3.14. In the Figure 3.13, "CurrentNode" saves the NodeID being processed; starting from a node with has the lowest NUR. "VariableOrderQueue" saves the order of variables from node being processed in "CurrentNode", process from the first order. "NodeUsed" records each node sharing a variable in "VariableOrderQueue". "PreCandidateVariable" records variables belong to nodes in "NodeUsed" being compared, before save to

“CandidateVariable”. Variable choosing in “PreCandidateVariable” is according to VUR of nodes. The variable has the lowest VUR will be picked first. “CandidateVariable” records relative variable order. “NodeStack” pushes a node to the stack, after a node is finished from processing in “CurrentNode”.

Step	Current Node (NUR)	Variable Order Queue (VUR)	Node Used (NUR)	Pre Candidate Variable (VUR)	Candidate Variable	Node Stack
1	...	...	...	...	...	...
2	...	...	...	...	...	...

Figure 3.13 Computing relative variable order data structure

1. Starting from the node with the lowest NUR, pick a node and place it in CurrentNode. Obtain all the <Variable ID><VarOperator><VarValue> and enter them to VariableOrderQueue.
2. Starting from the first variable of VariableOrderQueue.

   If the variable is shared by some other node, find the node and enter it to NodeUsed if it is not already in NodeUsed or NodeStack. If more than one node exists, the variable picking will be based on VUR with the lowest one picked first. Get all variables in the previous order of the variable being processed and variable itself from all nodes in the PreCandidateVariable. Remove current variable from VariableOrderQueue. Enter the variable from PreCandidateVariable to the CandidateVariable if there's no exists such the variable and base on its order. If the order is the same, the variable of lower VUR will be picked.
3. If the node in NodeUsed has no next variables order, push it into NodeStack and remove it from NodeUsed.
4. If the node in CurrentNode which has the variable being processed has no next variables, push it into NodeStack, and remove it from CurrentNode.
5. Get a node from NodeUsed which has the lowest NUR and place it in CurrentNode. If NodeUsed is empty, but there still are nodes in the knowledge base that do not appear in NodeStack, pick a node from them with the lowest NUR. Go to step 2 and do all steps until all nodes in the knowledge base appear in NodeStack.

Figure 3.14 Computing relative variable order algorithm

Computing relative variable order process is takes a long time as it checks each node in *n* nodes of knowledge base. Also if at a variable being processed there're nodes

sharing this variable, the process has to compare each variables in each shared nodes. Thus, the time complexity of this process = O (n × number of nodes sharing a variable × number of variables in each shared nodes).

An example of node generation will be demonstrated in Section 4.4. As a matter of fact, Figure 3.14 also obtains the relative order of nodes from the NodeStack by poping the nodes from the stack.

3.5 Knowledge Building

The system architecture of VCIRS supports three operations related to the knowledge base. First, there is knowledge building which allows the user to create a knowledge base from the scratch or to update the existing knowledge base. Second, we have knowledge refining. It allows user to obtain the important and the usage degree of a variable/node/rule, or to generate the new rule. Third, we have knowledge inferencing to perform inferencing/reasoning from the knowledge base. There're two inferencing approach employed in this operation, RBS and RDR approach.

Similar to RDR, VCIRS can build a knowledge base from scratch; that's a knowledge base is not available, but the user wants to build a new knowledge base. VCIRS builds a new knowledge base on the cases provided by the user.

According to the contents of a case, VCIRS will search the knowledge base for a proper node. A node here is the similar to a rule in the RBS, which consists of one or more variables and its values (clause part), and one or more conclusions (conclusion part). Figure 3.15 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 3.15 Knowledge building algorithm

Finding a proper node in the knowledge base is done by the algorithm of Figure 3.16.

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 3.16 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 (Ref. Figure 2.2). 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 Figure 3.17.

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 3.17 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.

3.6 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.

Originated from RDR, VCIRS tends to employ the cornerstone case approach to perform an inferencing process. In RDR, once a case is entered, the system finds a similar case from the cornerstone cases. A cornerstone case is the case that was presented when the user created the rule. The user needs not to check all such cases. The only cornerstone case that can be misinterpreted is the one that is associated with the rule being fired, i.e., the rule that gives the wrong interpretation, against the new rule to be added. Rather than scrutinizing this case, the user can select proper conditions for the new rule from a list of differences between the presented cases according to which the rule is being added and the corner case. By doing this, the knowledge base is being verified at the same time it is being built. We found RDR seems to use Depth First Search (DFS) while traversing its nodes during inference.

In VCIRS the traversing process looks like DFS, but it is apparently different. With the Variable-Centered Rule Structure where each variable (the fundamental

element in VCIRS) is saved with its positions, the inferencing process can thus go very fast. VCIRS locates a variable, a node or a rule easily through their positions.

During inferencing process VCIRS treats a rule as the sequence of nodes (rules in the RBS). It disregards the content of each node, except the last node as the conclusion of that node. From this point of view the inferencing treats a rule as a (big) rule whose clause part contains all clause parts in each node of the sequence, and whose conclusion part is the conclusion of the last node. That is, the clause operator is the AND operator (of all clauses), which is also the conclusion operator if there are more than one conclusion values in a node (and then it becomes the last node in a rule).

VCIRS requires the user be consistent in using the variable name both at the clauses and the conclusion parts, and maintains the consistency of the logical rules of the entire knowledge base. This is not as restricted as it seems, because the system can provide the user with the list of values picked from the variable values in the knowledge base. The user is free to decide which value she will use or not. The inferencing process in both RDR and RBS use the knowledge base as it is.

The two inferencing mechanisms, RBS and RDR, will be described below.

3.6.1 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 Figure 3.16. Figure 3.18 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 3.18 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.

3.6.2 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. This RBS rule creation process is called Knowledge Base Transformation and will be discussed in Section 3.7 later. Here we describe how forward chaining (Figure 3.19) and backward chaining (Figure 3.20) are done in VCIRS below.

1. Identify a case

2. Until no rule produces a new assertion or the case is identified

    For each rule:

        2.1 Try to support each clause part by matching it to known facts.

        2.2 If all the clause parts are supported assert the conclusion part

        2.3 Report for all matching and instantiation alternatives.

Figure 3.19 Forward chaining in VCIRS

Repeat until all hypothesis have been tried and none have been supported or until the value is identified

For each conclusion part

For each rule whose consequent matches the current conclusion part:

1. Try to support each clause part by matching it to assertion.
2. If all the clause parts are supported, announce success and conclude that the conclusion part is true.

Note: The chaining ends unsuccessfully if any required clause assertions cannot be supported.

Figure 3.20 Backward chaining in VCIRS

With either inferencing mechanism the system can give the user the result and explanation from the posted value:

- The conclusion obtained is the answer of question: What is the result from the inferencing process according to the value being posted.
- All recorded values during inferencing process (e.g., the rule sequence being fired or failed to fire, the processed value in the clause and conclusion part) are the explanation of question: How is the result obtained.
- The calculation of CF for value being processed and the confirmation of each clause determine which rule will be fired or failed to fire are the answer of question: Why the given result is obtained.

3.6.3 Confidence Factor

Confidence Factor is used to solve the priority problem. Each clause in a rule in the rule base may have a CF.

VCIRS only allows the operator AND in the clause and conclusion parts. The range of CF (i.e., CFValue in a variable or NodeCFValue in a node) is [1...100], so the actual Confidence Factor CF for rule *j* (node *j*) is:

$$\text{CF (node } j) = \frac{Min(CFValue_{1j}, CFValue_{2j}, ..., CFValue_{Nj}) \times NodeCFValue_j}{100} \qquad (10)$$

Where:

- CF (node *j*) = the actual CF for rule *j* (node *j*)
- $CFValue_{ij}$ = CF for *i*-th variable in the node *j*
- $NodeCFValue_j$ = CF for node *j*

3.7 Knowledge Base Transformation

First, 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. Figure 3.21 describes the transformation algorithm from the Node Structure to an RBS rule, while Figure 3.22 is for transforming the Rule Structure to RBS rules.

The destination is the current rule being created, one by one.

1. From Node Structure, starting from the first <Node ID> get each <Node ID> and transform it to <Rule ID>. Then, get <Node CF Value> from the current record and transform it to <Rule CF>.
2. Put the keyword ‘IF’ after <Rule CF>.
3. Get <Variables> from the current record and transform them to RBS <Clauses> (see Section 3.1), a clause for each <Variable ID> following the <Variable Order>. Omit <Variable Order> while transforming and put the keyword ‘AND’ before the subsequent <Variable ID> if <Variables> contains more than one <Variable ID>.
4. Put the keyword ‘THEN’ after <Clauses>.
5. Base on the current <Parent Node ID> and <Node ID> being processed, find the <Parent Node ID> and <Node ID> from the Conclusion table. Starting from first conclusion value, transform <Conclusion> to RBS <Conclusions>, and put the keyword ‘AND’ before the subsequent <Conclusion> if <Conclusion> contains more than one <Conclusion>.
6. The process repeats for each node in the Node Structure until the last node.

Figure 3.21 Transforming Node Structure to rule base

The name of the rule is obtained from the last <Node ID> in the rule, and the conclusion of the rule is obtained from the <Conclusion> which belongs to the last <Node ID> in the rule.

1. From Rule Structure which has more than one node, obtain the last <Node ID>. i.e., the <Node ID> which has the largest <Node Order>, and make it the <Rule ID> of the RBS rule if <Rule ID> doesn't exist in the RBS rule base. If it exists put an additional character "R" after this <Rule ID> to make it distinct from the existing one. "R" symbolize that the rule is built from a sequence of nodes, not from a node. Then, information is obtained from the Node Structure. Starting from the first <Node ID> and base on <Node Order> to get next <Node ID> to obtain the <Node CF Value> from the current record and transform it to <Rule CF>.
2. Put the keyword 'IF' after <Rule CF>.
3. Get <Variables> from current record and transform them to RBS <Clauses>, a clause for each <Variable ID> based on <Variable Order>. Omit <Variable Order> while transforming and put the keyword 'AND' before the subsequent <Variable ID> if <Variables> contains more than one <Variable ID>. Check the <Variable ID> in the current <Rule ID>, if it already exists replace it with the newest <Variable ID> along with its <Variables>.
4. Check the next <Node ID>. If the next <Node ID> is the last node in the current rule, put the keyword 'THEN', and go to step 5. If the next <Node ID> is not the last node, start again from step 1 with the next <Node ID> in the rule.
5. Starting from the first conclusion value in the last <Node ID> of the rule <Rule ID> from the Conclusion table transform <Conclusion> to RBS <Conclusions>, and put the keyword 'AND' before the subsequent <Conclusion> if <Conclusion> contains more than one <Conclusion>.
6. The process repeats for each rule in the Rule Structure.

Figure 3.22 Transforming Rule Structure to rule base

Chapter 4
System Demonstration and Evaluation

In this chapter we demonstrate our proposed system along with its evaluation by the cases to prove that the system can be work and able to perform easy knowledge building, powerful knowledge inferencing, and evolutional improvement of the system can be obtained at the same time.

4.1 Knowledge Building

Because the VCIRS starts the knowledge building process from the scratch, we need to enter the cases, unless we already have a knowledge base. For the cases we use the rules from the Zookeeper rule base [Wins92], reproduced in Figure 4.1 below. Here, to make the representation complete, we have arbitrarily added a confidence value to each rule and each variable-value pair.

```
Z1 [RULE CF=50]   IF    x=has hair          [CF=50]
                  THEN  x=mammal
Z2 [RULE CF=50]   IF    x=give milk         [CF=50]
                  THEN  x=mammal
Z3 [RULE CF=50]   IF    x=has feather       [CF=50]
                  THEN  x=bird
Z4 [RULE CF=50]   IF    x=fly               [CF=50]     AND
                        x=lay egg           [CF=50]
                  THEN  x=bird
Z5 [RULE CF=50]   IF    x=mammal            [CF=50]     AND
                        x=eat meat          [CF=50]
                  THEN x=carnivore
Z6 [RULE CF=50]   IF    x=mammal                [CF=50]     AND
                        x=pointed teeth         [CF=50]     AND
                        x=claw                  [CF=50]     AND
                        x=forward-pointing eye  [CF=50]
                  THEN  x=carnivore
Z7 [RULE CF=50]   IF    x=mammal            [CF=50]     AND
                        x=hoof              [CF=50]
                  THEN  x=ungulate
Z8 [RULE CF=50]   IF    x=mammal            [CF=50]     AND
                        x=chew cud          [CF=50]
                  THEN  x=ungulate
Z9 [RULE CF=50]   IF    x=carnivore         [CF=50]     AND
                        x=tawny color       [CF=50]     AND
                        x=dark spot         [CF=50]
                  THEN  x=cheetah
Z10 [RULE CF=50]  IF    x=carnivore         [CF=50]     AND
                        x=tawny color       [CF=50]     AND
                        x=black stripe      [CF=50]
                  THEN  x=tiger
Z11 [RULE CF=50]  IF    x=ungulate          [CF=50]     AND
                        x=long leg          [CF=50]     AND
                        x=long neck         [CF=50]     AND
                        x=tawny color       [CF=50]     AND
                        x=dark spot         [CF=50]
                  THEN  x=giraffe
Z12 [RULE CF=50]  IF    x=ungulate          [CF=50]     AND
                        x=white color       [CF=50]     AND
                        x=black stripe      [CF=50]
                  THEN  x=zebra
Z13 [RULE CF=50]  IF    x=bird              [CF=50]     AND
                        x=does not fly      [CF=50]     AND
                        x=long leg          [CF=50]     AND
                        x=long neck         [CF=50]     AND
                        x=black and white   [CF=50]
                  THEN  x=ostrich
Z14 [RULE CF=50]  IF    x=bird              [CF=50]     AND
                        x=does not fly      [CF=50]     AND
                        x=swim              [CF=50]     AND
                        x=black and white   [CF=50]
                  THEN  x=penguin
Z15 [RULE CF=50]  IF    x=bird              [CF=50]     AND
                        x=good flyer        [CF=50]
                  THEN  x=albatross
```

Figure 4.1 Zookeeper rule base

In RBS, when we build the rule base rule by rule, we can do it directly without consider the “tree” of the rules, because RBS does not treats its rule base in the tree format. All rules in the rule base form a degenerated tree, meaning there is an imaginary “root” and each rule is the branch of that “root”. In other word, in RBS all rules are the top level tree, each rule at the same level.

In VCIRS, the “if-then” rules are organized in a hierarchy of rules and exceptions; a VCIRS knowledge base (rule base) is organized as a tree structure in which each node represents a rule and the children of a node represent exceptions to the rule. It can be viewed as classification systems where the conclusion of a rule defines a class to be assigned to an example. The use of VCIRS supports an incremental approach to knowledge acquisition: starting with an empty rule base, as new examples are encountered, the user adds a rule to correctly classify the example, and when an example is classified incorrectly, the user defines exceptions to the rules that results in the misclassification.

Before we go into the step by step construction of the VCIRS rule base, we slightly change the rule format for convenience. Formally, we change

```
Z1 [RULE CF=50]   IF    x=has hair        [CF=50]
                  THEN  x=mammal
```

to:

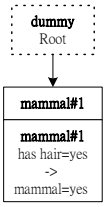
```
[RULE CF=50]      IF    has hair=yes      [CF=50]
                  THEN  mammal=yes
```

The detailed building process is in Appendix A. Here we demonstrate some important cases, i.e., case 1, 6 and 15, and the final result.

- Case 1 is taken from rule Z1.

```
[RULE CF=50]      IF    has hair=yes      [CF=50]
                  THEN  mammal=yes
```

Because no candidate nodes can be found, the user chooses to create a top level node. In the VCIRS knowledge (i.e., in terms of the tree structure) it becomes:



- Case 6 is taken from rule Z6.

```
[RULE CF=50]      IF    mammal=yes                    [CF=50]     AND
                        pointed teeth=yes             [CF=50]     AND
                        claw=yes                      [CF=50]     AND
                        forward-pointing eye=yes      [CF=50]
                  THEN  carnivore=yes
```

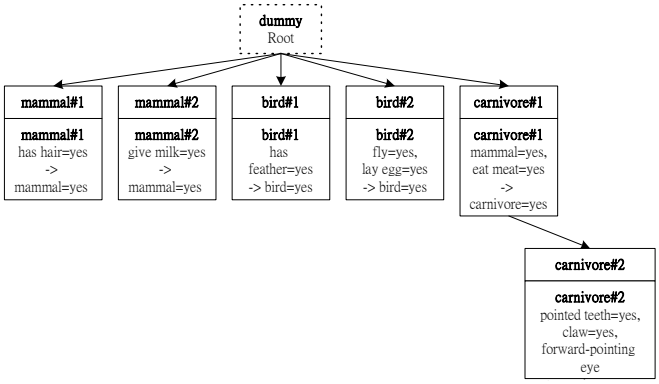
Using the Node Structure, this system knows the new case has the same conclusion value with the old case in [ParentNodeID: _root_, NodeID: carnivore#1]. Also [VariableID: mammal] is already saved at the old case in [ParentNodeID: _root_, NodeID: carnivore#1]. The user can choose either or both of the following conditions:

Condition 1	eat meat<>yes	[CF=50]
Condition 2	pointed teeth=yes claw=yes forward-pointing eye=yes	[CF=50] AND [CF=50] AND [CF=50] AND

It's trivial that the user also can choose to create a new node at the top level of the knowledge base or continue her tree traversing without doing any action. If she chooses the condition 2, with the following order of variables:

```
pointed teeth=yes             [CF=50]
claw=yes                      [CF=50]
forward-pointing eye=yes      [CF=50]
```

In terms of the tree structure, it becomes:



- Case 15 is taken from rule Z15.

```
[RULE CF=50]      IF    bird=yes           [CF=50]     AND
                        good flyer=yes     [CF=50]
                  THEN  albatross=yes
```

Suppose the user chooses to create a top level node. Figure 4.2 depicts the rule tree of the knowledge base after the last case (case 15) is entered.

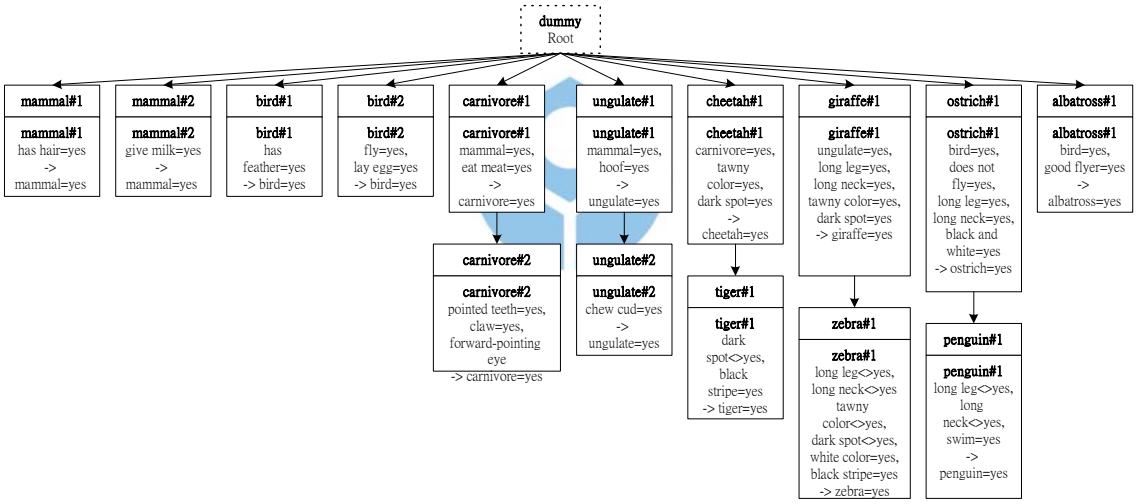


Figure 4.2 Zookeeper knowledge base in VCIRS

Every time the user enters a case; the structure of knowledge base along with VUR, NUR and RUR also changes. The next section is demonstration of variable and value analysis from the case posted by the user.

4.2 Variable Analysis

From the knowledge base obtained as demonstrated above, we got the statistic of occurrence of each variable in the nodes as depicted in Table 4.1. Thus, the most important variables obtained are long leg, long neck and dark spot, all with 4 occurrences; the next important variable is tawny color; and so on.

Table 4.1 Variable occurrences in the nodes

VariableID	NumOfNodeIDs	NodeID Used	VariableOrder
give milk	1	mammal#2	1
has feather	1	bird#1	1
fly	1	bird#2	1
lay egg	1	bird#2	2
eat meat	1	carnivore#1	2
pointed teeth	1	carnivore#2	1
claw	1	carnivore#2	2
forward-pointing eye	1	carnivore#2	3
hoof	1	ungulate#1	2
has hair	1	mammal#1	1
carnivore	1	cheetah#1	1
good flyer	1	albatross#1	2
ungulate	1	giraffe#1	1
white color	1	zebra#1	5
does not fly	1	ostrich#1	2
black and white	1	ostrich#1	5
swim	1	penguin#1	3
chew cud	1	ungulate#2	1
bird	2	ostrich#1 albatross#1	1 1
mammal	2	carnivore#1 ungulate#1	1 1
black stripe	2	tiger#1 zebra#1	2 6
tawny color	3	cheetah#1 giraffe#1 zebra#1	2 4 3
long neck	4	giraffe#1 zebra#1 ostrich#1 penguin#1	3 2 4 2
long leg	4	giraffe#1 zebra#1 ostrich#1 penguin#1	2 1 3 1
dark spot	4	cheetah#1 tiger#1 giraffe#1 zebra#1	3 1 5 4

We also obtain the occurrence of each node in the rules as depicted in Table 4.2. And we know the most important nodes obtained are carnivore#1, cheetah#1, giraffe#1, ostrich#1 and ungulate#1, the next important nodes are all rest of the nodes.

Table 4.2 Node occurrences in the rules

NodeID	NumOfRuleIDs	RuleID Used	NodeOrder
albatross#1	1	albatross#1	1
bird#1	1	bird#1	1
bird#2	1	bird#2	1
carnivore#2	1	carnivore#2	2
mammal#1	1	mammal#1	1
mammal#2	1	mammal#2	1
penguin#1	1	penguin#1	2
tiger#1	1	tiger#1	2
ungulate#2	1	ungulate#2	2
zebra#1	1	zebra#1	2
carnivore#1	2	carnivore#1 carnivore#2	1 1
cheetah#1	2	cheetah#1 tiger#1	1 1
giraffe#1	2	giraffe#1 zebra#1	1 1
ostrich#1	2	ostrich#1 penguin#1	1 1
ungulate#1	2	ungulate#1 ungulate#2	1 1

The result of variable analysis along with the result of value analysis will support rule generation to improve the coverage of the knowledge base.

4.3 Value Analysis

The calculation of VUR, NUR and RUR using the corresponding usage assignments (equations (1), (2) and (3)), from the knowledge base are listed in Table 4.3-4.5 below.

Table 4.3 Variable Usage Rates

VariableID	NodeID	VariableOrder	VUR
pointed teeth	carnivore#2	1	0.333333333333333
ungulate	giraffe#1	1	0.4
fly	bird#2	1	0.5
carnivore	cheetah#1	1	0.666666666666667
long leg	zebra#1	1	0.666666666666667
claw	carnivore#2	2	0.666666666666667
bird	ostrich#1	1	0.8
does not fly	ostrich#1	2	0.8
white color	zebra#1	5	0.833333333333333
chew cud	ungulate#2	1	1
has feather	bird#1	1	1
give milk	mammal#2	1	1
has hair	mammal#1	1	1
bird	albatross#1	1	1
lay egg	bird#2	2	1
eat meat	carnivore#1	2	1
hoof	ungulate#1	2	1
good flyer	albatross#1	2	1
forward-pointing eye	carnivore#2	3	1
swim	penguin#1	3	1
long leg	penguin#1	1	1.33333333333333
long neck	zebra#1	2	1.33333333333333
tawny color	zebra#1	3	1.5
mammal	ungulate#1	1	2
mammal	carnivore#1	1	2
dark spot	tiger#1	1	2
black stripe	tiger#1	2	2
black and white	ostrich#1	5	2
black stripe	zebra#1	6	2
long neck	penguin#1	2	2.66666666666667
dark spot	zebra#1	4	2.66666666666667
long leg	giraffe#1	2	3.2
tawny color	cheetah#1	2	4
long neck	giraffe#1	3	4.8
long leg	ostrich#1	3	4.8
tawny color	giraffe#1	4	4.8
long neck	ostrich#1	4	6.4
dark spot	cheetah#1	3	8
dark spot	giraffe#1	5	8

From Table 4.3, we discover the highest VUR refers to variable dark spot, following by long neck as the next heavily used variable, and so on.

Table 4.4 Node Usage Rates

NodeID	NumOfVariableIDs	NUR
carnivore#2	3	0.666666666666667
bird#2	2	0.75
ungulate#2	1	1
bird#1	1	1
mammal#2	1	1
mammal#1	1	1
albatross#1	2	1
ungulate#1	2	1.5
carnivore#1	2	1.5
zebra#1	6	1.5
penguin#1	3	1.66666666666667
tiger#1	2	2
ostrich#1	5	2.96
cheetah#1	3	4.22222222222222
giraffe#1	5	4.24

Similarly, from Table 4.4 the highest NUR refers to node giraffe#1, following by the next heavily used node cheetah#1, and so on.

Table 4.5 Rule Usage Rates

RuleID	RUR
bird#2	0.75
albatross#1	1
bird#1	1
mammal#2	1
mammal#1	1
carnivore#2	1.08333333333333
ungulate#2	1.25
ungulate#1	1.5
carnivore#1	1.5
penguin#1	2.31333333333334
zebra#1	2.87
ostrich#1	2.96
tiger#1	3.11111111111111
cheetah#1	4.22222222222222
giraffe#1	4.24

Finally, from Table 4.5 the highest RUR obtained is rule giraffe#1, rule cheetah#1 runs second and so on.

The result of value analysis will guide the user during the knowledge building and inferencing process by suggesting to the user those most used variables/nodes/rules in the knowledge base.

4.4 Rule Generation

We use the “Computing relative node order algorithm” (Figure 3.11) to obtain the relative order of the nodes in the rules as shown in Table 4.6. The detailed computing process is in Appendix B.

Table 4.6 The relative node order in the knowledge base

Step	CurrentRule (RUR)	NodeOrderQ (NUR)	RuleUsed (RUR)	Pre CandidateNode (NUR)	CandidateNode	RuleStack
...	...	...	...	...	...	...
21					bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1 tiger#1	cheetah#1 tiger#1 giraffe#1 zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2

Thus, the relative node order obtained is: bird#2 followed by albatross#1, bird#1 and so on. Note that the table also shows the relative rule order from the RuleStack, starting from the last saved rule, i.e., bird#2 as the first order rule, followed by albatross#1, bird#1 and so on.

Recall that the most important nodes obtained from variable analysis are carnivore#1, cheetah#1, giraffe#1, ostrich#1 and ungulate#1. According to the “Rule generation algorithm” (Figure 3.9), they become the last node of the candidate rules to be generated, which means we have five candidate rules as follows:

NodeID	NumOfRuleIDs	Conclusion	NodeOrder
carnivore#1	2	carnivore	1
cheetah#1	2	cheetah	1
giraffe#1	2	giraffe	1
ostrich#1	2	ostrich	1
ungulate#1	2	ungulate	1

The “Rule generation algorithm” then asks to compose preceding nodes according to the relative node order as we obtained earlier. Table 4.7 shows the results.

Table 4.7 Node combination based on relative node order

RuleID	Conclusion	NodeOrder
carnivore#1G	carnivore	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1
cheetah#1G	cheetah	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1
ostrich#1G	ostrich	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1
ungulate#1G	ungulate	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1

Each candidate rule is then presented to the user for confirmation. Table 4.8 shows how this is done. Nodes which are crossed out represent semantically incorrect nodes. The table also shows the percentage of correct nodes.

Table 4.8 Confirmation of generated rules

RuleID	Conclusion	NodeOrder	% Correct
carnivore#1G	carnivore	~~bird#2~~ ~~albatross#1~~ ~~bird#1~~ mammal#2 mammal#1 carnivore#1	50%
cheetah#1G	cheetah	~~bird#2~~ ~~albatross#1~~ ~~bird#1~~ mammal#2 mammal#1 carnivore#1 carnivore#2 ~~ungulate#1~~ ~~ungulate#2~~ ~~ostrich#1~~ ~~penguin#1~~ ~~giraffe#1~~ ~~zebra#1~~ cheetah#1	36%
ostrich#1G	ostrich	bird#2 ~~albatross#1~~ bird#1 ~~mammal#2~~ ~~mammal#1~~ ~~carnivore#1~~ ~~carnivore#2~~ ungulate#1 ungulate#2 ostrich#1	50%
ungulate#1G	ungulate	~~bird#2~~ ~~albatross#1~~ ~~bird#1~~ mammal#2 mammal#1 ~~carnivore#1~~ ~~carnivore#2~~ ungulate#1	38%

We find the most eligible generated rules are rule carnivore#1G and ostrich#1G. Rule carnivore#1G has nodes mammal#2, mammal#1 and carnivore#1, while rule ostrich#1G has nodes bird#2, bird#1, ungulate#1, ungulate#2 and ostrich#1. The next eligible generated rule is ungulate#1G which contain nodes mammal#2, mammal#1 and ungulate#1. Finally, rule cheetah#1G has nodes mammal#2, mammal#1, carnivore#1, carnivore#2 and cheetah#1. Note that all the generated rules can be saved because there are no such rules existents in the knowledge base.

During rule generation process the user can also perform node generation. Following the “Node generation algorithm” (Figure 3.12), we can obtain the final relative order of the variables in the nodes in Table 4.9. Detailed process can be referenced from Appendix C.

Table 4.9 The relative variable order in the knowledge base

Step	CurrentNode (NUR)	VarOrderQ (VUR)	NodeUsed (NUR)	Pre CandidateVar (VUR)	CandidateVar	NodeStack
...	...	...	...	...	...	...
25					pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes black stripe=yes swim=yes	penguin#1 tiger#1 zebra#1 carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

Now the “Node generation algorithm” wants to use the most important variables as the last variables of candidate nodes. Recall that the most important variables obtained from variable analysis include long neck, long leg and dark spot. Table 3.10 lists those NodeIDs which use these variables. In the table, we find variables long neck and long leg are used by the same NodeIDs. The VariableOrder column shows variable long neck has one higher than variable long leg in terms of order, which means variable long neck covers variable long leg (recall that in the Closeness Degree as shown in equation (7), the closer a variable is to the conclusion’s node, the better it is). So we

need only take variable long neck and omit variable long leg. It also occurs in the NodeID: giraffe#1 and zebra#1, where variable dark spot covers variable long neck. Variable long neck and dark spot thus remain the last variables in the nodes to be generated.

Table 4.10 Removal redundant important variables

VariableID	NodeID Used	VariableOrder
long neck	~~giraffe#1~~ ~~zebra#1~~ ostrich#1 penguin#1	~~3~~ ~~2~~ 4 2
~~long leg~~	~~giraffe#1~~ ~~zebra#1~~ ~~ostrich#1~~ ~~penguin#1~~	~~2~~ ~~1~~ ~~3~~ ~~1~~
dark spot	cheetah#1 tiger#1 giraffe#1 zebra#1	3 1 5 4

Now, we can compose the combination of variables to generate a complete node, according to the relative variable order we just obtained. The system will confirm to the user about the node being generated before it saves as the additional nodes knowledge base, whether a node is make sense or not. And its make sense after we observe that there are some semantically incorrect node in the generated node as shown in Appendix D.

From Appendix D, we see the most eligible generated node is node giraffe#1G which contains the sequence of variables chew cud=yes, has feather=yes, give milk=yes, has hair=yes, does not file=yes, ungulate=yes, long leg=yes, long neck=yes, tawny color=yes and dark spot=yes. Other eligible nodes include tiger#1G, ostrich#1G, zebra#1G, cheetah#1G and penguin#1G. These generated nodes can be saved because there are new in the knowledge base.

During rule generation, a generated node is useful, because they introduce primitive rules into the knowledge base, which in turn enriches the possibility of composing rules.

4.5 Knowledge Inferencing

4.5.1 RDR Inferencing

With the Zookeeper knowledge base as depicted in Figure 4.2, RDR inferencing starts with finding a proper node, supported by the proper node algorithm based upon the facts posted by the user. The implementation of RDR inferencing is described in the Figure E.13 (Appendix E).

First of all, the user may want to see the result of usage assignment to briefly check the usage degree of the knowledge base so far. Table 4.11 shows what she might see.

Table 4.11 Usage assignment result

(a) Rule Usage Rate

RuleID	RUR
giraffe#1	4.24
cheetah#1	4.22222222222222
tiger#1	3.11111111111111
ostrich#1	2.96
zebra#1	2.87
penguin#1	2.31333333333334
ungulate#1	1.5
carnivore#1	1.5
ungulate#2	1.25
carnivore#2	1.08333333333333
albatross#1	1
bird#1	1
mammal#1	1
mammal#2	1
bird#2	0.75

(b) Node Usage Rate

NodeID	NumOfVariableIDs	NUR
giraffe#1	5	4.24
cheetah#1	3	4.22222222222222
ostrich#1	5	2.96
tiger#1	2	2
penguin#1	3	1.66666666666667
ungulate#1	2	1.5
carnivore#1	2	1.5
zebra#1	6	1.5
bird#1	1	1
mammal#1	1	1
albatross#1	2	1
ungulate#2	1	1
mammal#2	1	1
bird#2	2	0.75
carnivore#2	3	0.666666666666667

(c) Variable Usage Rate

VariableID	NodeID	VariableOrder	VUR
dark spot	cheetah#1	3	8
dark spot	giraffe#1	5	8
long neck	ostrich#1	4	6.4
long leg	ostrich#1	3	4.8
long neck	giraffe#1	3	4.8
tawny color	giraffe#1	4	4.8
tawny color	cheetah#1	2	4
long leg	giraffe#1	2	3.2
dark spot	zebra#1	4	2.66666666666667
long neck	penguin#1	2	2.66666666666667
black and white	ostrich#1	5	2
black stripe	tiger#1	2	2
black stripe	zebra#1	6	2
dark spot	tiger#1	1	2
mammal	ungulate#1	1	2
mammal	carnivore#1	1	2
tawny color	zebra#1	3	1.5
long leg	penguin#1	1	1.33333333333333
long neck	zebra#1	2	1.33333333333333
bird	albatross#1	1	1
chew cud	ungulate#2	1	1
eat meat	carnivore#1	2	1
forward-pointing eye	carnivore#2	3	1
give milk	mammal#2	1	1
good flyer	albatross#1	2	1
has feather	bird#1	1	1
has hair	mammal#1	1	1
hoof	ungulate#1	2	1
lay egg	bird#2	2	1
swim	penguin#1	3	1
white color	zebra#1	5	0.833333333333333
bird	ostrich#1	1	0.8
does not fly	ostrich#1	2	0.8
carnivore	cheetah#1	1	0.666666666666667
claw	carnivore#2	2	0.666666666666667
long leg	zebra#1	1	0.666666666666667
fly	bird#2	1	0.5
ungulate	giraffe#1	1	0.4
pointed teeth	carnivore#2	1	0.333333333333333

Now, suppose she enters the following case:

```
tawny color=yes
dark spot=yes
```

The system responds by finding 2 candidate nodes:

```
cheetah#1
giraffe#1
```

Taking from here, Table 4.12 illustrates step-by-step how RDR inferencing is done on this case using the algorithm in Figure 3.18. It's clear that the inferencing results are: giraffe and zebra.

Table 4.12 RDR inferencing example

Step	CandidateNode	RuleID:NodeOrder	Unconfirmed Case	User Confirmation	Event Log	Conclusion Queue
1	**cheetah#1** giraffe#1	**cheetah#1:1** tiger#1:1	carnivore=yes	unconfirmed	cheetah#1:1 failed to fire	
2	**cheetah#1** giraffe#1	tiger#1:1	carnivore=yes	confirmed	Check next node order at tiger#1	
3	**cheetah#1** giraffe#1	tiger#1:2	dark spot<>yes black stripe=yes	unconfirmed	tiger#1:2 failed to fire Check next candidate node	
4	giraffe#1	giraffe#1:1	ungulate=yes long leg=yes long neck=yes	confirmed	giraffe#1:1 fired Check next node order if user want	giraffe
5	giraffe#1	giraffe#1:2	long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes black stripe=yes	confirmed	giraffe#1:2 fired	giraffe zebra

If at step 4, the user does not want to continue, the process will stop. The conclusion is only giraffe. Another one, if at the step 3, the user failed to confirm the fact then there's no conclusion obtained.

Specifically the answer of: "What question" of this inferencing is the values in the ConclusionQueue. The last one is the final conclusion, while the rest are intermediate conclusions. The answer of: "Why question" (e.g., why a conclusion is obtained, why no conclusion at all, why such a conclusion is obtained rather than the other) and "How question" (e.g., how a result is obtained, how come a conclusion is failed to obtain, how can only a conclusion is obtained) can be obtained from the EventLog with respect to ConclusionQueue.

Inferencing process here, starting from finding the proper node until obtaining or failing to obtain the result, is easy because our Variable-Centered Rule Structure can

remember the position of each variable and its value. Locating a variable or a node in a rule is very fast through its position.

4.5.2 Knowledge Base Transformation

Using the algorithm in Figure 3.21 and Figure 3.22, we can transform the Figure 4.2 (The Zookeeper knowledge base in VCIRS) into Figure 4.3 and Figure 4.4. The transformed two figures constitute the RBS version of the original knowledge base.

```
mammal#1    [RULE CF=50]      IF    has hair=yes      [CF=50]
                              THEN  mammal=yes
mammal#2    [RULE CF=50]      IF    give milk=yes     [CF=50]
                              THEN  mammal=yes
bird#1      [RULE CF=50]      IF    has feather=yes   [CF=50]
                              THEN  bird=yes
bird#2      [RULE CF=50]      IF    fly=yes           [CF=50]     AND
                                    lay egg=yes       [CF=50]
                              THEN  bird=yes
carnivore#1 [RULE CF=50]      IF    mammal=yes        [CF=50]     AND
                                    eat meat=yes      [CF=50]
                              THEN  carnivore=yes
carnivore#2 [RULE CF=50]      IF    pointed teeth=yes [CF=50]     AND
                                    claw=yes          [CF=50]     AND
                                forward-pointing eye=yes [CF=50] AND
                              THEN  carnivore=yes
ungulate#1  [RULE CF=50]      IF    mammal=yes        [CF=50]     AND
                                    hoof=yes          [CF=50]
                              THEN  ungulate=yes
ungulate#2  [RULE CF=50]      IF    chew cud=yes      [CF=50]
                              THEN  ungulate=yes
cheetah#1   [RULE CF=50]      IF    carnivore=yes     [CF=50]     AND
                                    tawny color=yes   [CF=50]     AND
                                    dark spot=yes     [CF=50]
                              THEN  cheetah=yes
tiger#1     [RULE CF=50]      IF    dark spot<>yes    [CF=50]     AND
                                    black stripe=yes  [CF=50]
                              THEN  tiger=yes
giraffe#1   [RULE CF=50]      IF    ungulate=yes      [CF=50]     AND
                                    long leg=yes      [CF=50]     AND
                                    long neck=yes     [CF=50]     AND
                                    tawny color=yes   [CF=50]     AND
                                    dark spot=yes     [CF=50]
                              THEN  giraffe=yes
zebra#1     [RULE CF=50]      IF    long leg<>yes     [CF=50]     AND
                                    long neck<>yes    [CF=50]     AND
                                    tawny color<>yes  [CF=50]     AND
                                    dark spot<>yes    [CF=50]     AND
                                    white color=yes   [CF=50]     AND
                                    black stripe=yes  [CF=50]
                              THEN  zebra=yes
ostrich#1   [RULE CF=50]      IF    bird=yes          [CF=50]     AND
                                    does not fly=yes  [CF=50]     AND
                                    long leg=yes      [CF=50]     AND
                                    long neck=yes     [CF=50]     AND
                                    black and white=yes [CF=50]
                              THEN  ostrich=yes
penguin#1   [RULE CF=50]      IF    long leg<>yes     [CF=50]     AND
                                    long neck<>yes    [CF=50]     AND
                                    swim=yes          [CF=50]
                              THEN  penguin=yes
albatross#1 [RULE CF=50]      IF    bird=yes          [CF=50]     AND
                                    good flyer=yes    [CF=50]
                              THEN  albatross=yes
```

Figure 4.3 Node Structure transformation result

```
carnivore#2R [RULE CF=50] IF  mammal=yes                [CF=50]     AND
                              eat meat=yes              [CF=50]     AND
                              pointed teeth=yes         [CF=50]     AND
                              claw=yes                  [CF=50]     AND
                              forward-pointing eye=yes [CF=50]
                          THEN carnivore=yes
ungulate#2R [RULE CF=50]      IF    mammal=yes          [CF=50]     AND
                                    hoof=yes            [CF=50]     AND
                                    chew cud=yes        [CF=50]
                          THEN  ungulate=yes
tiger#1R    [RULE CF=50]      IF    carnivore=yes       [CF=50]     AND
                                    tawny color=yes     [CF=50]     AND
                                    dark spot<>yes      [CF=50]     AND
                                    black stripe=yes    [CF=50]
                              THEN  tiger=yes
zebra#1R    [RULE CF=50]      IF    ungulate=yes        [CF=50]     AND
                                    long leg<>yes       [CF=50]     AND
                                    long neck<>yes      [CF=50]     AND
                                    tawny color<>yes    [CF=50]     AND
                                    dark spot<>yes      [CF=50]     AND
                                    white color=yes     [CF=50]     AND
                                    black stripe=yes    [CF=50]
                              THEN  zebra=yes
penguin#1R  [RULE CF=50]      IF    bird=yes            [CF=50]     AND
                                    does not fly=yes    [CF=50]     AND
                                    long leg<>yes       [CF=50]     AND
                                    long neck<>yes      [CF=50]     AND
                                    black and white=yes [CF=50]     AND
                                    swim=yes            [CF=50]
                              THEN  penguin=yes
```

Figure 4.4 Rule Structure transformation result

4.5.3 RBS Inferencing

Before performing forward and backward chaining, the system needs to do initialization of the transformed rule base. Here, as an example we use only rules from Node Structure transformation result, for simplicity. This is done by scanning all the rules obtained from the Zookeeper rule base. A BaseVariableList (Table 4.13) and VariableList (Table 4.14) are thus created according to Appendix E.3.2.2. With these, we can demonstrate forward and backward reasoning, to be detailed separately below.

Table 4.13 BaseVariableList

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
mammal#1	50	has hair	=	yes	50	mammal	1
mammal#2	50	give milk	=	yes	50	mammal	1
bird#1	50	has feather	=	yes	50	bird	1
bird#2	50	fly	=	yes	50	bird	1
bird#2	50	lay egg	=	yes	50	bird	2
carnivore#1	50	mammal	=	yes	50	carnivore	1
carnivore#1	50	ear meat	=	yes	50	carnivore	2
carnivore#2	50	pointed teeth	=	yes	50	carnivore	1
carnivore#2	50	claw	=	yes	50	carnivore	2
carnivore#2	50	forward-pointing eye	=	yes	50	carnivore	3
ungulate#1	50	mammal	=	yes	50	ungulate	1
ungulate#1	50	hoof	=	yes	50	ungulate	2
ungulate#2	50	chew cud	=	yes	50	ungulate	1
cheetah#1	50	carnivore	=	yes	50	cheetah	1
cheetah#1	50	tawny color	=	yes	50	cheetah	2
cheetah#1	50	dark spot	=	yes	50	cheetah	3
tiger#1	50	dark spot	<>	yes	50	tiger	1
tiger#1	50	black stripe	=	yes	50	tiger	2
giraffe#1	50	ungulate	=	yes	50	giraffe	1
giraffe#1	50	long leg	=	yes	50	giraffe	2
giraffe#1	50	long neck	=	yes	50	giraffe	3
giraffe#1	50	tawny color	=	yes	50	giraffe	4
giraffe#1	50	dark spot	=	yes	50	giraffe	5
zebra#1	50	long leg	<>	yes	50	zebra	1
zebra#1	50	long neck	<>	yes	50	zebra	2
zebra#1	50	tawny color	<>	yes	50	zebra	3
zebra#1	50	dark spot	<>	yes	50	zebra	4
zebra#1	50	white color	=	yes	50	zebra	5
zebra#1	50	black stripe	=	yes	50	zebra	6
ostrich#1	50	bird	=	yes	50	ostrich	1
ostrich#1	50	does not fly	=	yes	50	ostrich	2
ostrich#1	50	long leg	=	yes	50	ostrich	3
ostrich#1	50	long neck	=	yes	50	ostrich	4
ostrich#1	50	black and white	=	yes	50	ostrich	5
penguin#1	50	long leg	<>	yes	50	penguin	1
penguin#1	50	long neck	<>	yes	50	penguin	2
penguin#1	50	swim	=	yes	50	penguin	3
albatross#1	50	bird	=	yes	50	albatross	1
albatross#1	50	good flyer	=	yes	50	albatross	2
carnivore#2R	50	mammal	=	yes	50	carnivore	1
carnivore#2R	50	eat meat	=	yes	50	carnivore	2
carnivore#2R	50	pointed teeth	=	yes	50	carnivore	3
carnivore#2R	50	claw	=	yes	50	carnivore	4
carnivore#2R	50	forward-pointing eye	=	yes	50	carnivore	5
ungulate#2R	50	mammal	=	yes	50	ungulate	1
ungulate#2R	50	hoof	=	yes	50	ungulate	2
ungulate#2R	50	chew cud	=	yes	50	ungulate	3
tiger#1R	50	carnivore	=	yes	50	tiger	1
tiger#1R	50	tawny color	=	yes	50	tiger	2
tiger#1R	50	dark spot	=	yes	50	tiger	3
tiger#1R	50	black stripe	=	yes	50	tiger	4
zebra#1R	50	ungulate	=	yes	50	zebra	1
zebra#1R	50	long leg	<>	yes	50	zebra	2
zebra#1R	50	long neck	<>	yes	50	zebra	3
zebra#1R	50	tawny color	<>	yes	50	zebra	4
zebra#1R	50	dark spot	<>	yes	50	zebra	5
zebra#1R	50	white color	=	yes	50	zebra	6
zebra#1R	50	black stripe	=	yes	50	zebra	7
penguin#1R	50	bird	=	yes	50	penguin	1
penguin#1R	50	does not fly	=	yes	50	penguin	2
penguin#1R	50	long leg	<>	yes	50	penguin	3
penguin#1R	50	long neck	<>	yes	50	penguin	4
penguin#1R	50	black and white	=	yes	50	penguin	5
penguin#1R	50	swim	=	yes	50	penguin	6

Table 4.14 VariableList

VariableID	Sign	VarOperator	VarValue	CFValue
pointed teeth	NO			
claw	NO			
forward-pointing eye	NO			
fly	NO			
lay egg	NO			
chew cud	NO			
has feather	NO			
give milk	NO			
has hair	NO			
bird	NO			
good flyer	NO			
does not fly	NO			
ungulate	NO			
long leg	NO			
long neck	NO			
black and white	NO			
carnivore	NO			
tawny color	NO			
dark spot	NO			
mammal	NO			
hoof	NO			
eat meat	NO			
white color	NO			
black stripe	NO			
swim	NO			

4.5.3.1 Forward Chaining

- Every event is recorded into EventLog.
- The system starts by asking the user about the fact she already knows, and suppose she enters:

  tawny color=yes [CF=65]

- Then the system finds there're five RuleIDs which have tawny color in their clause: cheetah#1, giraffe#1 and zebra#1. Each rule is calculated as follows.

```
cheetah#1   [RULE CF=50]      IF    carnivore=yes     [CF=50]     AND
                                    tawny color=yes   [CF=50]     AND
                                    dark spot=yes     [CF=50]
                              THEN  cheetah=yes
```

$$CF(\text{cheetah\#1}) = \frac{Min(50,50,50)\times 50}{100} = 25$$

```
giraffe#1   [RULE CF=50]      IF    ungulate=yes      [CF=50]     AND
                                    long leg=yes      [CF=50]     AND
                                    long neck=yes     [CF=50]     AND
                                    tawny color=yes   [CF=50]     AND
                                    dark spot=yes     [CF=50]
                              THEN  giraffe=yes
```

$$\text{CF (giraffe\#1)} = \frac{Min(50,50,50,50,50)\times 50}{100} = 25$$

Because CF nodes of three RuleIDs are the same, we can arbitrarily pick one. Suppose the system picks cheetah#1.

- Record the case into VariableList.

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
carnivore	NO			
tawny color	YES	=	yes	65
dark spot	NO			
...	...			

- Record the RuleID and NumOfVariableIDs from the BaseVariableList.

RuleID	NumOfVariableIDs
cheetah#1	3

- Initialize ConclusionVariableQueue.

ConclusionVariableQueue
tawny color

- Update CurrentVarOrder.

RuleID	VarOrder
cheetah#1	1

The system goes to check the next variable: carnivore=yes

- Does carnivore have instantiation in the VariableList?

carnivore doesn't instantiate

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
carnivore	NO			
...	...			

- Instantiate variable carnivore by asking the user. Suppose the user enters:

carnivore=yes [CF=70]

The system then updates the VariableList.

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
carnivore	YES	=	yes	70
tawny color	YES	=	yes	65
dark spot	NO			
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
cheetah#1	50	carnivore	=	yes	50	cheetah	1
...	...	...	...	...	...	...	...

Yes, it is satisfied.

- Are all the clauses in the BaseVariableList checked?

No, not yet. It is just the first variable; totally there're 3 variables in the clause of RuleID: cheetah#1

RuleID	VarOrder
cheetah#1	1

RuleID	NumOfVariableIDs
cheetah#1	3

- Update CurrentVarOrder

RuleID	VarOrder
cheetah#1	2

The system now checks the second variable: tawny color=yes

- Does tawny color have instantiation in the VariableList?

tawny color is already instantiated

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
tawny color	YES	=	yes	65
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
cheetah#1	50	tawny color	=	yes	50	cheetah	1
...	...	...	...	...	...	...	...

Yes, it is satisfied.

- Are all the clauses in BaseVariableList checked?

No, not yet. It is just the second variable, totally there're 3 variables in the clause of RuleID: cheetah#1

RuleID	VarOrder
cheetah#1	2

RuleID	NumOfVariableIDs
cheetah#1	3

- Update CurrentVarOrder.

RuleID	VarOrder
cheetah#1	3

The system now checks the third variable: dark spot=yes

- Does dark spot have instantiation in the VariableList?

dark spot doesn't instantiate

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
dark spot	NO			
...	...			

- Instantiate variable dark spot by asking the user. Suppose the user enters:

dark spot=yes [CF=55]

The system thus updates the VariableList

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
carnivore	YES	=	yes	70
tawny color	YES	=	yes	65
dark spot	YES	=	yes	55
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
cheetah#1	50	dark spot	=	yes	50	cheetah	1
...	...	...	...	...	...	...	...

Yes, it is satisfied.

- Are all the clauses in the BaseVariableList checked?

Yes, it is. Totally there're 3 variables in the clause of RuleID: cheetah#1

RuleID	VarOrder
cheetah#1	3

RuleID	NumOfVariableIDs
cheetah#1	3

- Execute the Conclusion Part in the rule base.

  cheetah=yes

- Update ConclusionVariableQueue.

ConclusionVariableQueue
cheetah
tawny color

- Record cheetah=yes to the ResultQueue and in the VariableList if there exists VariableID cheetah.

ResultQueue
cheetah=yes

Because in the VariableList there exists no VariableID cheetah, so there's no change in the VariableList.

- Remove the first entry from ConclusionVariableValue after this variable finish to process (whether it success or failed to fire).

ConclusionVariableQueue
cheetah
~~tawny color~~

Now, the current variable being processed is cheetah.

- Is there any clause (i.e., VariableID) in the rest of BaseVariableList has a VariableID = cheetah?

  No, there is no VariableID = cheetah

RuleID	NumOfVariableIDs

- Variable cheetah is finished to process, i.e., RuleID and NumOfVariableIDs do not have a value. Remove the first entry from ConclusionVariableValue.

ConclusionVariableQueue
~~cheetah~~

- Check if there's any variable in the ConclusionVariableQueue, update CurrentVarOrder base on recorded RuleID and NumOfVariableIDs.

ConclusionVariableQueue

There is no variable in the ConclusionVariableQueue

- Because no variable being processed in the ConclusionVariableQueue, inferencing process is finish. Show the result of forward chaining from ResultQueue.

ResultQueue
cheetah=yes

4.5.3.2 Backward Chaining

- Every event is recorded into EventLog.
- The system asks the user about the fact she knows in the Conclusion Part of the rule base, and suppose she enters:

  carnivore

- Is there any ConclusionValue in the BaseVariableList which has the value = carnivore? If there're more than one RuleID which has ConclusionValue = carnivore, then the RuleID which has the largest CF of the node will be picked first.

Yes, there are two RuleID which have ConclusionValue = carnivore.

```
carnivore#1 [RULE CF=50]      IF    mammal=yes        [CF=50]     AND
                                    eat meat=yes      [CF=50]
                              THEN  carnivore=yes
```

$$CF\ (carnivore\#1) = \frac{Min(50,50)\times 50}{100} = 25$$

```
carnivore#2 [RULE CF=50]      IF    pointed teeth=yes [CF=50]     AND
                                    claw=yes          [CF=50]     AND
                                forward-pointing eye=yes [CF=50] AND
                              THEN  carnivore=yes
```

$$CF\ (carnivore\#2) = \frac{Min(50,50,50)\times 50}{100} = 25$$

Because CF nodes of two RuleIDs are the same, we arbitrarily pick one. Suppose the system pick carnivore#1.

- Record the RuleID and NumOfVariableIDs from the BaseVariableList.

RuleID	NumOfVariableIDs
carnivore#1	2

- Update ConclusionStack.

RuleID	VarOrder
carnivore#1	1

The system will check: mammal=yes

- Does the VariableID have instantiation in the VariableList, or, VarOrder > NumOfVariableIDs?

mammal in the VariableList doesn't instantiate and VarOrder = 1 not equal with NumOfVariableIDs

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
...	...			

- Is there any ConclusionValue = mammal in the ConclusionValue in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
mammal#1	50	has hair	=	yes	50	mammal	1
mammal#2	50	give milk	=	yes	50	mammal	1
...	...	...	...	...	...	...	...

Yes, mammal exists in the ConclusionValue in the BaseVariableList at RuleID mammal#1 and mammal#2.

- Obtain the RuleID which has the largest CF Node.

$$\text{CF (mammal\#1)} = \frac{Min(50)\times 50}{100} = 25$$

$$\text{CF (mammal\#2)} = \frac{Min(50)\times 50}{100} = 25$$

Because CF nodes of two RuleIDs are the same, we arbitrarily pick one. Suppose the system pick mammal#1.

- Record the RuleID and NumOfVariableIDs from the BaseVariableList.

RuleID	NumOfVariableIDs
mammal#1	1
carnivore#1	2

- Update ConclusionStack.

RuleID	VarOrder
mammal#1	1
carnivore#1	1

The system will check: has hair=yes in the VariableList

- Does the VariableID have instantiation in the VariableList, or, VarOrder > NumOfVariableIDs?

has hair in the VariableList doesn't instantiate and VarOrder = 1 not equal with NumOfVariableIDs

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
has hair	NO			
...	...			

- Instantiate variable has hair by asking the user. Suppose the user enters:

  has hair=no [CF=60]

The system thus updates the VariableList

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
has hair	YES	=	no	60
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
mammal#1	50	has hair	=	yes	50	mammal	1
...	...	...	...	...	...	...	...

No, it does not satisfy.

- Remove the top of stack in the ConclusionStack.

RuleID	VarOrder
~~mammal#1~~	~~1~~
carnivore#1	1

The system will check: mammal=yes in the VariableList

- Is there any ConclusionValue = mammal in the ConclusionValue in the BaseVariableList? If any, check whether it exists in the RuleID at RuleID & NumOfVariableIDs.

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
mammal#1	50	has hair	=	yes	50	mammal	1
mammal#2	50	give milk	=	yes	50	mammal	1
...	...	...	...	...	...	...	...

RuleID	NumOfVariableIDs
mammal#1	1
carnivore#1	2

Yes, mammal exists in the ConclusionValue in the BaseVariableList at RuleID mammal#2.

- Record the RuleID and NumOfVariableIDs from the BaseVariableList.

RuleID	NumOfVariableIDs
mammal#2	1
mammal#1	1
carnivore#1	2

- Update ConclusionStack.

RuleID	VarOrder
mammal#2	1
carnivore#1	1

The system goes to check the first variable of the rule: give milk=yes in the VariableList

- Does give milk have instantiation in the VariableList?

  give milk doesn't instantiate

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
has hair	YES	=	no	60
give milk	NO			
...	...			

- Instantiate variable give milk by asking the user. Suppose the user enters:

  give milk=yes [CF=80]

The system thus updates the VariableList

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
has hair	YES	=	no	60
give milk	YES	=	yes	80
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
mammal#2	50	give milk	=	yes	50	mammal	1
...	...	...	...	...	...	...	...

  Yes, it is satisfied.

- Are all the clauses in BaseVariableList checked?

  Yes, they are.

RuleID	NumOfVariableIDs
mammal#2	1
mammal#1	1
carnivore#1	2

RuleID	VarOrder
mammal#2	1
carnivore#1	1

- Execute the Conclusion Part in the rule base.

  mammal=yes

- Update VariableList.

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	YES	=	yes	
has hair	YES	=	no	60
give milk	YES	=	yes	80
...	...			

- Record mammal=yes to the ResultQueue.

ResultQueue
mammal=yes

- Remove the top of stack from ConclusionStack after this variable finish to process (whether it success or failed to fire).

RuleID	VarOrder
~~mammal#2~~	~~1~~
carnivore#1	1

- Update ConclusionStack.

RuleID	VarOrder
carnivore#1	2

The system will check: eat meat=yes in the VariableList

- Does eat meat have instantiation in the VariableList?

  eat meat doesn’t instantiate

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
has hair	YES	=	no	60
give milk	YES	=	yes	80
eat meat	NO			
...	...			

- Is there any ConclusionValue = eat meat in the ConclusionValue in the BaseVariableList? If any, check whether it exists in the RuleID at RuleID & NumOfVariableIDs.

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...

No, there is exists no ConclusionValue = eat meat. If supposedly there's a ConclusionValue, RuleID and NumOfVariableIDs also ConclusionStack will be updated.

- Instantiate variable eat meat by asking the user. Suppose the user enters:

eat meat =yes [CF=70]

The system thus updates VariableList

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	NO			
has hair	YES	=	no	60
give milk	YES	=	yes	80
eat meat	YES	=	yes	70
...	...			

- Does the case being posted satisfy with the clause part in the BaseVariableList?

RuleID	RuleCF	VariableID	Var Operator	VarValue	CFValue	Conclusion Value	VarOrder
...	...	...	...	...	...	...	...
carnivore#1	50	eat meat	=	yes	50	mammal	2
...	...	...	...	...	...	...	...

Yes, it is satisfied.

- Are all the clauses in BaseVariableList checked?

Yes, they are.

RuleID	NumOfVariableIDs
mammal#2	1
mammal#1	1
carnivore#1	2

RuleID	VarOrder
carnivore#1	2

- Execute the Conclusion Part in the rule base.

carnivore=yes

- Update VariableList.if it exists.

```
Carnivore does not exist in the VariableList, so it won't update
```

VariableID	Sign	VarOperator	VarValue	CFValue
...	...			
mammal	YES	=	yes	
has hair	YES	=	no	60
give milk	YES	=	yes	80
...	...			

- Record carnivorel=yes to the ResultQueue.

ResultQueue
carnivore=yes mammal=yes

- Remove the top of stack from ConclusionStack after this variable finish to process (whether it success or failed to fire).

RuleID	VarOrder

```
Now ConclusionStack is empty.
```

- Because ConclusionStack is already empty, inferencing processs is finish and the result of backward chaining is saved in the ResultQueue.

ResultQueue
carnivore=yes mammal=yes

4.6 System Evaluation

First, like RDR, VCIRS only pays attention to the variables (the clause part) rather than the conclusions. Conclusion does not have a role in the system; it's only the result being obtained. It means we can disregard conclusion apparently without any risk.

The unimportance of the conclusion during the rule tree traversing process has a consequence: there's no mechanism to perform forward and backward chaining like that in RBS. RDR does not mention its inferencing, because RDR was proposed mainly for rapid and simple KA, not for inferencing. Inferencing in RDR is done at the same time when the user does knowledge building by providing cases and following the system to

work. She can choose to only traverse the rule tree during the operation without creating a new rule, which means she just wants to do inferencing (i.e., the simple forward chaining). The fact that she traverses trees (inferencing) and updates the knowledge base at the same time means the verification of the knowledge base is performed on the fly. VCIRS inherits both benefits, i.e., simple and easy KA and verification-on-the-fly. In addition to the tree traversing approach like RDR to make knowledge building (updating) simple, VCIRS also provides a mechanism to perform knowledge base transformation so that the user can do powerful RBS inferencing. All VCIRS requires is that the user has to be consistent in using variable IDs both at the clause and the conclusion parts, and it is able to maintain consistency of the logical rules in the entire knowledge base. This is not a serious limitation though, because the system will provide the user with list of values picked from the variable values posted in the knowledge base. The user is free to select which value she will use or not.

Compared with RDR, VCIRS is equally powerful in knowledge building. It also save space for the rule structure doesn't save every sequence of nodes. It simply remembers the position of each node and then it's easy to reconstruct the rule as the sequence of nodes. VCIRS save every occurrence of case in the Node Structure for the node, helps the system constitutes new rules for knowledge refining, which RDR can't do this.

RDR gains his quick speed in knowledge building by providing a cornerstone case, the example which is used when the user created the rule. RDR doesn't need to check all cases, only a related case which a new case is going to correct. Cornerstone cases are saved separately from the RDR rules. It further deteriorates the space requirement of RDR and makes repetitious/redundant data worse. In contrast, VCIRS only stores the node in the Node Structure. A rule in the Rule Structure practically only

saves the order and the parent node of each node, so its space requirement can be relaxed.

Compared with RBS, VCIRS can perform knowledge verification-on-the-fly, while RBS can not. In RBS, verification has to be done by the expert manually and it is time consuming and inconsistency prone even worse. In RBS an inconsistent rule tends to impact the other rules. This is less serious in VCIRS, since inconsistent rules only affect the sharing nodes (rules) that belong to some rules, not all rules. Even though some nodes (rules) have the same values with the others, unless those nodes are in the same sequence, nothing will happen.

VCIRS needs to recalculate VUR of related variables, if a variable is entered into the Variable-Centered Rule Structure. It also needs to recalculate NUR of all the nodes that use the variable, and update RUR of all rules that use the nodes. This seems a tedious job, but it is useful for guiding user in the knowledge building and inferencing process. It's also useful to support rule generation.

While performing rule generation, VCIRS needs the user to confirm the rules/nodes being generated, because the logical (semantic) correctness of the generated rules/nodes needs the user for consideration. The system only produces alternative sequences of important variables to constitute new nodes and sequences of nodes to constitute rules according to relative order of variables and nodes, respectively. It's syntactically correct but the semantic correctness still depends on the user.

Even though VCIRS overcomes the inferencing problem of RDR, the Rete algorithm [For82] still has a better performance comparing with the cycles that has to be done before getting a result. The Rete algorithm is used in most high performance rule-based systems, because it's a very efficient method for the pattern matching problem. Its drawback is that it has a high space complexity. VCIRS which has a Variable-Centered

Rule Structure to save the position (only position) of every variable in a node and rule provides a reasonable good performance, not so excellent like the Rete algorithm, but its space complexity tends to be lower.

Chapter 5
Conclusions

5.1 Summary

VCIRS is designed centered on a variable-supported node and rule structure to implement the knowledge base of a rule system. It adopts the system architecture and the powerful knowledge inferencing process from RBS; also it adopts the power of the knowledge acquisition process from RDR. RBS suffers from the knowledge acquisition process for pushing the user to understand the structure of the knowledge base. RDR can successfully cope with such a problem, with a price of decreasing the ability of powerful inferencing. VCIRS comes in between. By adopting the features from RBS, VCIRS obtains the advantage of a familiar format of knowledge base (i.e., rule base) and powerful inferencing process and result which is able to answer to such questions as: What, How, and Why from the user. And by adopting the features from RDR, VCIRS allows for extremely rapid and simple knowledge acquisition without the help of a knowledge engineer. Like RDR, this simple knowledge building process in the same time performs verification-on-the-fly. It guarantees the knowledge base to the one the user wants it to be.

VCIRS is able to transform its knowledge base to a standard rule base, which is familiar to the common user. By this, VCIRS can support the forward and backward chaining during the inferencing process. While transforming its knowledge base into the rule base, VCIRS adds a standard rule for each sequence of rules in addition to transforming each node into a standard rule.

VCIRS is able to perform inferencing by both RDR and RBS approaches. By the RBS approach it uses both forward and backward chainings, which are better for

knowledge base information extraction and finding relations among the rules. The RDR approach is like a simple forward chaining process by traversing the rule tree using DFS.

VCIRS has a new module called Refinement Module which has three tasks: variable analysis, value analysis and rule generation. Variable analysis determines what variables/nodes are most important (i.e., the important degree), while value analysis determines how often a rule/node/variable is used (i.e., 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. Variable analysis, value analysis, and rule generation; together they help evolutional improvement of the knowledge base.

The contribution of the thesis 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.

5.2 Future Research

There are further possibilities to enhance our proposed system. Some ready improvements include.

- Remove the repetitious knowledge acquisition problem, which still appears in VCIRS, like RDR does. The Rete algorithm looks can overcome this problem.
- Include other logical operators to empower the representation capability of VCIRS. VCIRS employed only the AND operator for simplicity.

- Introduce inexact reasoning in the knowledge base to expand the applicability of VCIRS, e.g., fuzzy knowledge base.
- Ontology can be introduced to alleviate the semantic's associated with rule generation.
- Extention of the node/rule conclusion in the rule generation.

References

[Baur90] Baur, G.R. and Pigford, D.V., *Expert Systems for Business: Concepts and Applications*, Boyd & Fraser Publishing Company, Boston-USA, 1990.

[Bey97] Beydoun, G. and Hoffmann, A., "NRDR for the acquisition of search knowledge," in *Proc. of the 10th Australian Conference on Artificial Intelligence*, 1997.

[Call91] Callan, J.P., Fawcett, T.E. and Rissland, E.L., "CABOT: an adaptive approach to case-based search," in *Proc. of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Morgan Kaufmann, 1991.

[Cat92] Catlett , J., "Ripple down rules as a mediating representation in interactive induction," in *Proc. of the Second Japanese Knowledge Acquisition for Knowledge Based Systems Workshop*, Kobe, Japan, 1992.

[Comp89] Compton, P., Horn, R., Quinlan, R. and Lazarus, L., "Maintaining an Expert System," in Quinlan, J. R. (Eds.), *Applications of Expert Systems*, Addison Wesley London, 1989, pp. 366-385.

[Comp90] Compton, P. J. and Jansen, R., "A philosophical basis for knowledge acquisition," *Knowledge Acquisition 2*, 1990, pp. 241-257. (Also in Proc. of the 3rd European Knowledge Acquisition for Knowledge-Based Systems Workshop, Paris, 1989, pp. 75-89).

[Comp91] 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," in *Proc. of 6th Bannf AAAI Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada, 1991.

[Comp92] Compton, P., Edwards, G., Srinivasan, A., Malor, R., Preston, P., Kang, B. and Lazarus, L., “Ripple down rules: turning knowledge acquisition into knowledge maintenance,” *Artificial Intelligence in Medicine 4*, 1992, pp. 47-59.

[Comp93] Compton, P., Kang, B., Preston, P. and Mulholland, M., “Knowledge acquisition without analysis,” in *Proc. of European* Knowledge *Acquisition Workshop*, Springer Verlag, 1993.

[Edw93] 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.

[For82] Forgy, C.L., “Rete: a fast algorithm for the many pattern/many object pattern match problem", *Artificial Intelligence 19* (1982), 17-37.

[Gain89] Gaines, B., “Knowledge acquisition: the continuum linking machine learning and expertise transfer,” in Boose, J., Gaines, B. and Ganascia, J.G. (Eds.), in *Proc. of Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, 1989.

[Gain92] Gaines, B.R. and Compton, P.J., “Induction of ripple down rules,” in *Proc. of Fifth Australian Conference on Artificial Intelligence*, Hobart, 1992.

[Ho74] 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.), in *Proc. of IEEE/WIC International Conference on Intelligent Agent Technology*, Los Alamitos, CA, 2003, pp. 67-74.

[Igni91] Ignizio, J.P., *Introduction to Expert Systems: The Development and Implementation of Rule-based Expert Systems*, McGraw-Hill International Editions, 1991.

[Kang95] Kang, B., P. Compton and P. Preston, "Multiple classification ripple down rules: evaluation and possibilities," in *Proc. of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, University of Calgary, 1995.

[Kivi93] Kivinen, J., Mannila, H. and Ukkonen, E., "Learning rules with local exceptions," in *Proc. of European Conference on Computational Theory*, 1993.

[Man91] Mansuri, Y., Kim, J.G., Compton, P. and Sammut, C., "A comparison of a manual knowledge acquisition method and an inductive learning method," in *Proc. of the First Australian Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Sydney, University of Sydney, 1991, pp.114-132.

[Mug87] Muggleton, S.D., "An oracle-based approach to constructive induction," in *Proc. of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

[Pres93] Preston, O., Edwards, G. and Compton, P., "A 1600 rule Expert System without knowledge engineers," in Leibowitz, J. (Eds.), in *Proc. of the Second World Congress on Expert Systems*, Pergamon in press, Lisbon, 1993.

[Sche96] Scheffer, T., "Algebraic foundations and improved methods of induction or ripple-down rules," in *Proc. of 2nd Pacific Rim Knowledge Acquisition Workshop*, 1996.

[Siro93] Siromoney, A. and Siromoney, R., "Variations and local exception in inductive logic programming," in *Machine Intelligence – Applied Machine Intelligence*, S. Muggleton, (Eds.), 1993, pp. 213-234.

[Sur02] Suryanto, H. and Compton, P., "Intermediate concept discovery in ripple down rule knowledge bases," in *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.

[Sur04] Suryanto, H. and Compton, P., "Invented predicates to reduce knowledge acquisition," in *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.

[Tur95] Turban, E., *Decision Support and Expert Systems: Management Support Systems*, Fourth Edition, Prentice-Hall, Inc., United States of America, 1995.

[Wada02] Wada, T., Yoshida, T., Motoda, H., and Washio, T., "Extension of the rdr method that can adapt to environmental changes and acquires knowledge from both experts and data," in *Proc. of PRICAI 2002: Trends in Artificial Intelligence*, Springer-Verlag, Berlin, 2002, pp. 218-227.

[Wins92] Winston, P.H., *Artificial Intelligence*, Addison-Wesley, Third Edition, 1992.

Appendix A

Knowledge Building Details

For the cases we use the rules from the Zookeeper rule base [Wins92], reproduced in Figure 4.1.

Before we go into the step by step construction of the VCIRS rule base, we slightly change the rule format for convenience. Formally, we change

```
Z1 [RULE CF=50]    IF    x=has hair        [CF=50]
                   THEN  x=mammal
```

to:

```
[RULE CF=50]       IF    has hair=yes      [CF=50]
                   THEN  mammal=yes
```

- Case 1 is taken from rule Z1.

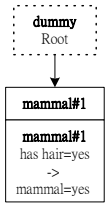
```
[RULE CF=50]       IF    has hair=yes      [CF=50]
                   THEN  mammal=yes
```

Because no candidate nodes can be found, the user chooses to create a top level node. In the VCIRS knowledge, it becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: mammal#1 NodeCFValue: 50 VariableID: has hair VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: has hair NumOfNodeIDs: 1	ConclusionValue: mammal ParentNodeID: _root_ NodeID: mammal#1

Node	Rule	Rule Structure
NodeID: mammal#1 NumOfVariableIDs: 1 NUR: 1	RuleID: mammal#1 NumOfNodeIDs: 1 RUR: 1	RuleID: mammal#1 NodeID: mammal#1 NodeOrder: 1

In terms of the tree structure:



- Case 2 is taken from rule Z2.

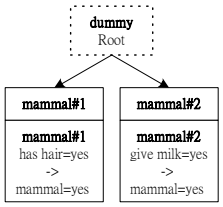
```
[RULE CF=50]       IF    give milk=yes     [CF=50]
                   THEN  mammal=yes
```

Candidate nodes are not found, so the user chooses to create a top level node. It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: mammal#2 NodeCFValue: 50 VariableID: give milk VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: give milk NumOfNodeIDs: 1	ConclusionValue: mammal ParentNodeID: _root_ NodeID: mammal#2

Node	Rule	Rule Structure
NodeID: mammal#2 NumOfVariableIDs: 1 NUR: 1	RuleID: mammal#2 NumOfNodeIDs: 1 RUR: 1	RuleID: mammal#2 NodeID: mammal#2 NodeOrder: 1

In terms of the tree structure:



- Case 3 is taken from rule Z3.

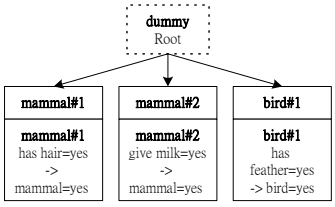
```
[RULE CF=50]      IF    has feather=yes   [CF=50]
                  THEN  bird=yes
```

Candidate nodes are not found, so the user chooses to create a top level node. It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: bird#1 NodeCFValue: 50 VariableID: has feather VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: has feather NumOfNodeIDs: 1	ConclusionValue: bird ParentNodeID: _root_ NodeID: bird#1

Node	Rule	Rule Structure
NodeID: bird#1 NumOfVariableIDs: 1 NUR: 1	RuleID: bird#1 NumOfNodeIDs: 1 RUR: 1	RuleID: bird#1 NodeID: bird#1 NodeOrder: 1

In terms of the tree structure:



- Case 4 is taken from rule Z4.

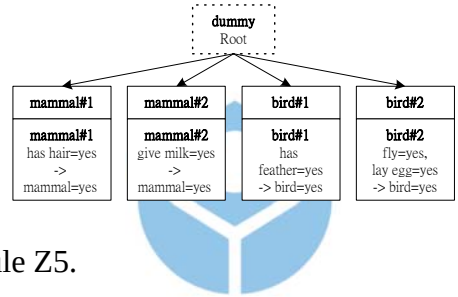
```
[RULE CF=50]      IF    fly=yes           [CF=50]     AND
                        lay egg=yes       [CF=50]
                  THEN  bird=yes
```

Candidate nodes are not found, so the user chooses to create a top level node. It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: bird#2 NodeCFValue: 50 VariableID: fly VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: lay egg VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: fly NumOfNodeIDs: 1 VariableID: lay egg NumOfNodeIDs: 1	ConclusionValue: bird ParentNodeID: _root_ NodeID: bird#2

Node	Rule	Rule Structure
NodeID: bird#2 NumOfVariableIDs: 2 NUR: 0.75	RuleID: bird#2 NumOfNodeIDs: 1 RUR: 0.75	RuleID: bird#2 NodeID: bird#2 NodeOrder: 1

In terms of the tree structure:



- Case 5 is taken from rule Z5.

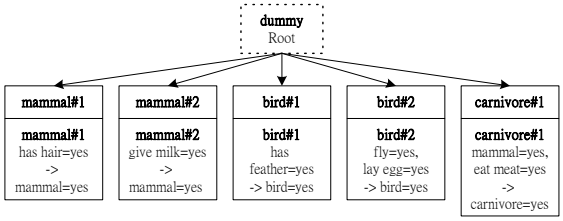
```
[RULE CF=50]      IF    mammal=yes        [CF=50]     AND
                        eat meat=yes      [CF=50]
                  THEN  carnivore=yes
```

Candidate nodes are not found, so the user chooses to create a top level node. It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: mammal NumOfNodeIDs: 1 VariableID: eat meat NumOfNodeIDs: 1	ConclusionValue: carnivore ParentNodeID: _root_ NodeID: carnivore#1

Node	Rule	Rule Structure
NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 0.75	RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 0.75	RuleID: carnivore#1 NodeID: carnivore#1 NodeOrder: 1

In terms of the tree structure:



- Case 6 is taken from rule Z6.

```
[RULE CF=50]      IF    mammal=yes                  [CF=50]     AND
                        pointed teeth=yes           [CF=50]     AND
                        claw=yes                    [CF=50]     AND
                        forward-pointing eye=yes    [CF=50]
                  THEN  carnivore=yes
```

Using the Node Structure, this system knows the new case has the same conclusion value with the old case in [ParentNodeID: _root_, NodeID: carnivore#1]. Also [VariableID: mammal] is already saved at the old case in [ParentNodeID: _root_, NodeID: carnivore#1]. The user can choose either or both of the following conditions:

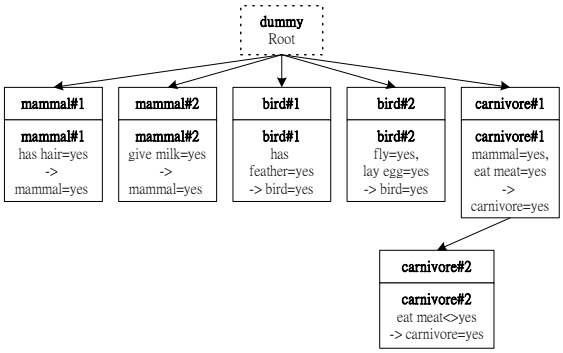
Condition 1	eat meat<>yes	[CF=50]
Condition 2	pointed teeth=yes claw=yes forward-pointing eye=yes	[CF=50] AND [CF=50] AND [CF=50] AND

It's trivial that the user also can choose to create a new node at the top level of the knowledge base or continue her tree traversing without doing any action. If she chooses condition 1, it becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: carnivore#1 NodeID: carnivore#2 NodeCFValue: 50 VariableID: eat meat VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2 ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 1 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4	VariableID: mammal NumOfNodeIDs: 1 VariableID: eat meat NumOfNodeIDs: 2	ConclusionValue: carnivore ParentNodeID: carnivore#1 NodeID: carnivore#2

Node	**Rule**	**Rule Structure**
NodeID: carnivore#2 NumOfVariableIDs: 1 NUR: 2 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 2.5	RuleID: carnivore#2 NumOfNodeIDs: 2 RUR: 2.25 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 2.5	RuleID: carnivore#2 NodeID: carnivore#1 NodeOrder: 1 NodeID: carnivore#2 NodeOrder: 2

In terms of the tree structure:



Else, if she chooses the condition 2, with the following order of variables:

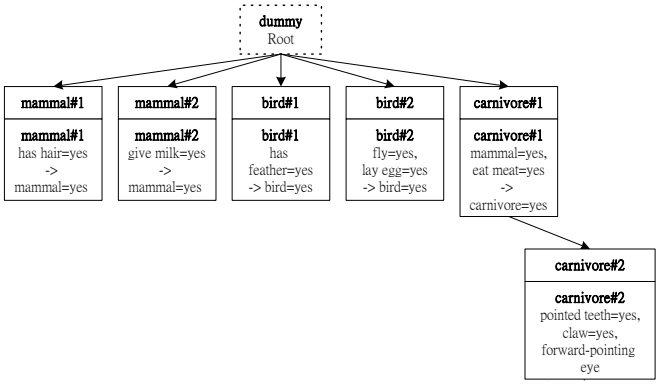
```
pointed teeth=yes              [CF=50]
claw=yes                       [CF=50]
forward-pointing eye=yes       [CF=50]
```

It becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: carnivore#1 NodeID: carnivore#2 NodeCFValue: 50 VariableID: pointed teeth VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.333 VariableID: claw VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.667 VariableID: forward-pointing eye VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 1 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: mammal NumOfNodeIDs: 1 VariableID: eat meat NumOfNodeIDs: 1	ConclusionValue: carnivore ParentNodeID: carnivore#1 NodeID: carnivore#2

Node	Rule	Rule Structure
NodeID: carnivore#2 NumOfVariableIDs: 3 NUR: 0.667 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 1	RuleID: carnivore#2 NumOfNodeIDs: 2 RUR: 1.083 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 1	RuleID: carnivore#2 NodeID: carnivore#1 NodeOrder: 1 NodeID: carnivore#2 NodeOrder: 2

In terms of the tree structure:



Finally, if she chooses both the condition 1 and 2, with the following order of variables:

```
eat meat<>yes                [CF=50]
pointed teeth=yes            [CF=50]
claw=yes                     [CF=50]
forward-pointing eye=yes     [CF=50]
```

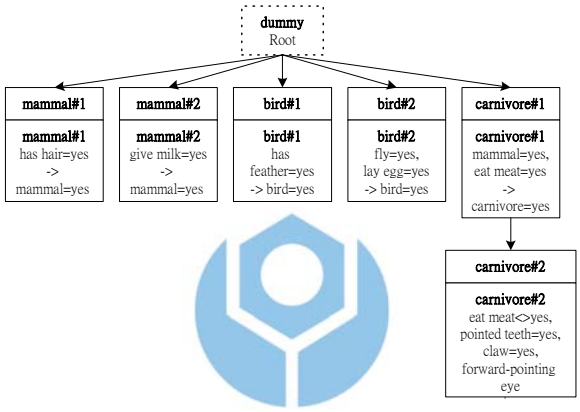
It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: carnivore#1 NodeID: carnivore#2 NodeCFValue: 50 VariableID: eat meat VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: pointed teeth VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: claw VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.75 VariableID: forward-pointing eye VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50	VariableID: mammal NumOfNodeIDs: 1 VariableID: eat meat NumOfNodeIDs: 2	ConclusionValue: carnivore ParentNodeID: carnivore#1 NodeID: carnivore#2

VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 1 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4		

Node	**Rule**	**Rule Structure**
NodeID: carnivore#2 NumOfVariableIDs: 4 NUR: 0.688 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 2.5	RuleID: carnivore#2 NumOfNodeIDs: 2 RUR: 1.594 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 2.5	RuleID: carnivore#2 NodeID: carnivore#1 NodeOrder: 1 NodeID: carnivore#2 NodeOrder: 2

In terms of the tree structure:



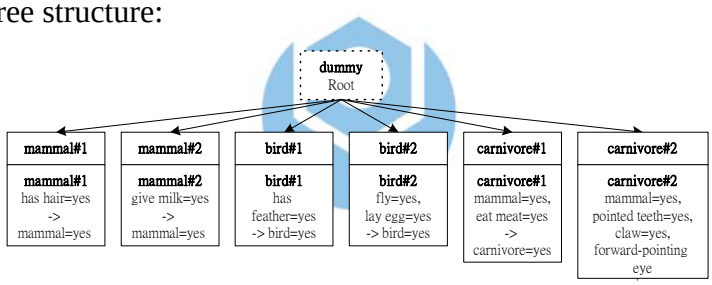
The user can also choose to create a new node at the top level of the knowledge base, it becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: carnivore#1 NodeID: carnivore#2 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: pointed teeth VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: claw VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.75 VariableID: forward-pointing eye	VariableID: mammal NumOfNodeIDs: 2 VariableID: eat meat NumOfNodeIDs: 1	ConclusionValue: carnivore ParentNodeID: carnivore#1 NodeID: carnivore#2

VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1		

Node	**Rule**	**Rule Structure**
NodeID: carnivore#2 NumOfVariableIDs: 4 NUR: 0.688 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 1	RuleID: carnivore#2 NumOfNodeIDs: 1 RUR: 0.688 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 1	RuleID: carnivore#2 NodeID: carnivore#2 NodeOrder: 1

In terms of the tree structure:



Or, the user can choose to continue her tree traversing without doing any action. For the example, we suppose the user chooses condition 2 from the 5 options above.

- Case 7 is taken from rule Z7.

```
[RULE CF=50]      IF    mammal=yes        [CF=50]     AND
                        hoof=yes          [CF=50]
                  THEN  ungulate=yes
```

The user can choose either or both of the following conditions:

Condition 1	eat meat<>yes	[CF=50]
Condition 2	hoof=yes	[CF=50]

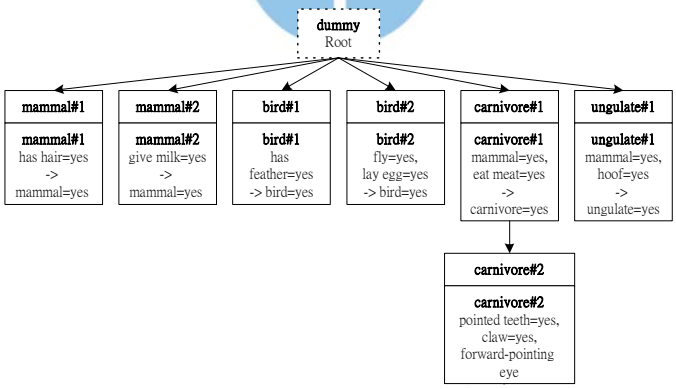
Suppose the user chooses to create a top level node, it becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: _root_ NodeID: ungulate#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: mammal NumOfNodeIDs: 2 VariableID: hoof NumOfNodeIDs: 1	ConclusionValue: ungulate ParentNodeID: _root_ NodeID: ungulate#1

VariableID: hoof VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: carnivore#1 NodeCFValue: 50 VariableID: mammal VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 2 VariableID: eat meat VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1		

Node	**Rule**	**Rule Structure**
NodeID: ungulate#1 NumOfVariableIDs: 2 NUR: 1 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 1.5 NodeID: carnivore#2 NumOfVariableIDs: 3 NUR: 0.667	RuleID: ungulate#1 NumOfNodeIDs: 1 RUR: 1 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 1.5 RuleID: carnivore#2 NumOfNodeIDs: 2 RUR: 1.083	RuleID: ungulate#1 NodeID: ungulate#1 NodeOrder: 1

In terms of the tree structure:



- Case 8 is taken from rule Z8.

```
[RULE CF=50]      IF    mammal=yes          [CF=50]     AND
                        chew cud=yes        [CF=50]
                  THEN  ungulate=yes
```

The user can choose either or both of the following conditions:

Condition 1	hoof<>yes	[CF=50]
Condition 2	chew cud=yes	[CF=50]

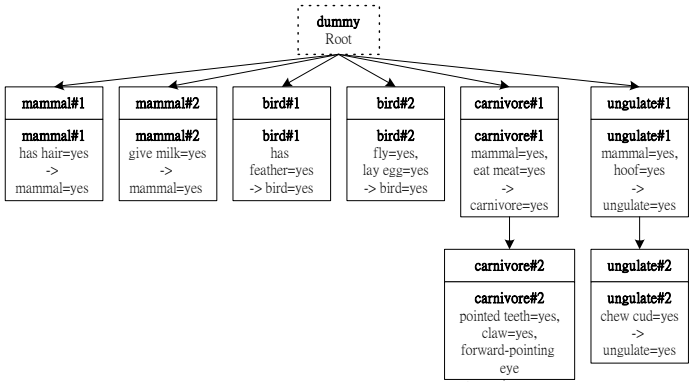
Suppose she chooses condition 2, it becomes.

Node Structure	**Variable**	**Conclusion**
ParentNodeID: ungulate#1 NodeID: ungulate#2 NodeCFValue: 50 VariableID: chew cud VariableOrder: 1	VariableID: mammal NumOfNodeIDs: 2 VariableID: hoof NumOfNodeIDs: 1	ConclusionValue: ungulate ParentNodeID: ungulate#1 NodeID: ungulate#2

```
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 1
VUR: 1

ParentNodeID: _root_
NodeID: ungulate#1
NodeCFValue: 50
VariableID: mammal
VariableOrder: 1
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 2
VariableID: hoof
VariableOrder: 2
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 1
VUR: 1

ParentNodeID: _root_
NodeID: carnivore#1
NodeCFValue: 50
VariableID: mammal
VariableOrder: 1
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 2
VariableID: eat meat
VariableOrder: 2
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 1
VUR: 1
```

Node	**Rule**	**Rule Structure**
NodeID: ungulate#2 NumOfVariableIDs: 1 NUR: 1 NodeID: ungulate#1 NumOfVariableIDs: 2 NUR: 1.5 NodeID: carnivore#1 NumOfVariableIDs: 2 NUR: 1.5	RuleID: ungulate#2 NumOfNodeIDs: 2 RUR: 1.25 RuleID: ungulate#1 NumOfNodeIDs: 1 RUR: 1.5 RuleID: carnivore#1 NumOfNodeIDs: 1 RUR: 1.5 RuleID: carnivore#2 NumOfNodeIDs: 2 RUR: 1.083	RuleID: ungulate#2 NodeID: ungulate#1 NodeOrder: 1 NodeID: ungulate#2 NodeOrder: 2

In terms of the tree structure:



- Case 9 is taken from rule Z9.

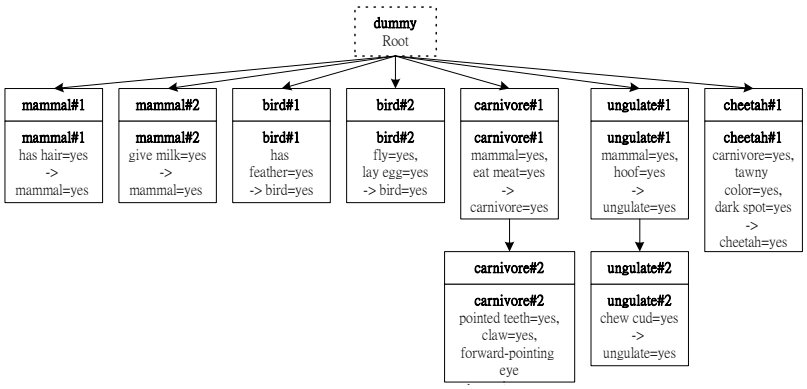
```
[RULE CF=50]      IF    carnivore=yes     [CF=50]     AND
                        tawny color=yes   [CF=50]     AND
                        dark spot=yes     [CF=50]
                  THEN  cheetah=yes
```

Candidate nodes are not found, so the user chooses to create a top level node. It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: _root_ NodeID: cheetah#1 NodeCFValue: 50 VariableID: carnivore VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.333 VariableID: tawny color VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.667 VariableID: dark spot VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1	VariableID: carnivore NumOfNodeIDs: 1 VariableID: tawny color NumOfNodeIDs: 1 VariableID: dark spot NumOfNodeIDs: 1	ConclusionValue: cheetah ParentNodeID: _root_ NodeID: cheetah#1

Node	Rule	Rule Structure
NodeID: cheetah#1 NumOfVariableIDs: 1 NUR: 0.667	RuleID: cheetah#1 NumOfNodeIDs: 1 RUR: 0.667	RuleID: cheetah#1 NodeID: cheetah#1 NodeOrder: 1

In terms of the tree structure:



- Case 10 is taken from rule Z10.

```
[RULE CF=50]      IF    carnivore=yes     [CF=50]     AND
                        tawny color=yes   [CF=50]     AND
                        black stripe=yes  [CF=50]
                  THEN  tiger=yes
```

The user can choose either or both of the following conditions:

Condition 1	dark spot<>yes	[CF=50]
Condition 2	black stripe=yes	[CF=50]

Suppose she chooses both conditions 1 and 2, with the following order of variables:

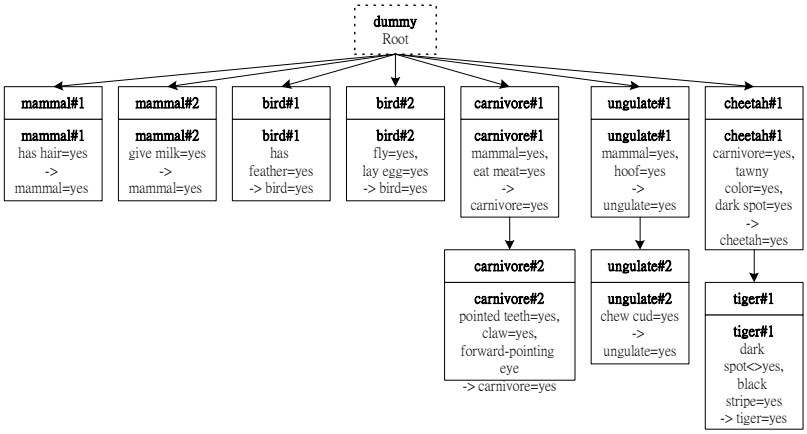
```
dark spot<>yes    [CF=50]
black stripe=yes  [CF=50]
```

It becomes:

Node Structure	Variable	Conclusion
ParentNodeID: cheetah#1 NodeID: tiger#1 NodeCFValue: 50 VariableID: dark spot VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1 VariableID: black stripe VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: cheetah#1 NodeCFValue: 50 VariableID: carnivore VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.667 VariableID: tawny color VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 1.333 VariableID: dark spot VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4	VariableID: carnivore NumOfNodeIDs: 1 VariableID: tawny color NumOfNodeIDs: 1 VariableID: dark spot NumOfNodeIDs: 2 VariableID: black stripe NumOfNodeIDs: 1	ConclusionValue: tiger ParentNodeID: cheetah#1 NodeID: tiger#1

Node	Rule	Rule Structure
NodeID: tiger#1 NumOfVariableIDs: 2 NUR: 1 NodeID: cheetah#1 NumOfVariableIDs: 3 NUR: 2	RuleID: tiger#1 NumOfNodeIDs: 2 RUR: 1.5 RuleID: cheetah#1 NumOfNodeIDs: 1 RUR: 2	RuleID: tiger#1 NodeID: cheetah#1 NodeOrder: 1 NodeID: tiger#1 NodeOrder: 2

In terms of the tree structure:



- Case 11 is taken from rule Z11.

```
[RULE CF=50]      IF    ungulate=yes      [CF=50]     AND
                        long leg=yes      [CF=50]     AND
                        long neck=yes     [CF=50]     AND
                        tawny color=yes   [CF=50]     AND
                        dark spot=yes     [CF=50]
                  THEN  giraffe=yes
```

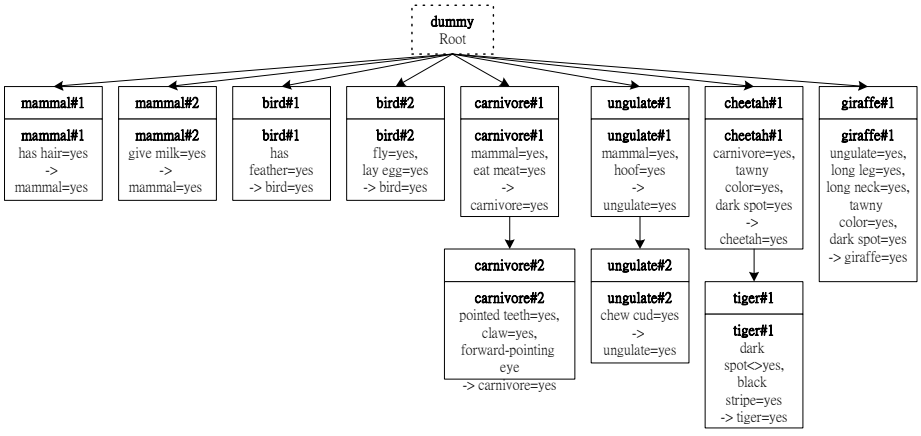
Suppose the user chooses to create a top level node, it becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: _root_ NodeID: giraffe#1 NodeCFValue: 50 VariableID: ungulate VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.2 VariableID: long leg VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.4 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.6 VariableID: tawny color VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1.6 VariableID: dark spot VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 3 ParentNodeID: _root_ NodeID: cheetah#1 NodeCFValue: 50 VariableID: carnivore	VariableID: ungulate NumOfNodeIDs: 1 VariableID: long leg NumOfNodeIDs: 1 VariableID: long neck NumOfNodeIDs: 1 VariableID: tawny color NumOfNodeIDs: 2 VariableID: dark spot NumOfNodeIDs: 3	ConclusionValue: giraffe ParentNodeID: _root_ NodeID: giraffe#1

VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.667 VariableID: tawny color VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 2.667 VariableID: dark spot VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 6 ParentNodeID: cheetah#1 NodeID: tiger#1 NodeCFValue: 50 VariableID: dark spot VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.5 VariableID: black stripe VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1		

Node	**Rule**	**Rule Structure**
NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 1.16 NodeID: cheetah#1 NumOfVariableIDs: 3 NUR: 3.111	RuleID: giraffe#1 NumOfNodeIDs: 1 RUR: 1.16 RuleID: cheetah#1 NumOfNodeIDs: 1 RUR: 3.111 RuleID: tiger#1 NumOfNodeIDs: 2 RUR: 2.181	RuleID: giraffe#1 NodeID: giraffe#1 NodeOrder: 1

In terms of the tree structure:



- Case 12 is taken from rule Z12.

```
[RULE CF=50]      IF    ungulate=yes      [CF=50]     AND
                        white color=yes   [CF=50]     AND
                        black stripe=yes  [CF=50]
                  THEN  zebra=yes
```

The user can choose either or both of the following conditions:

Condition 1	long leg<>yes	[CF=50]
	long neck<>yes	[CF=50]
	tawny color<>yes	[CF=50]
	dark spot<>yes	[CF=50]
Condition 2	white color=yes	[CF=50]
	black stripe=yes	[CF=50]

Suppose she chooses both conditions 1 and 2, with the following order of variables:

```
long leg<>yes           [CF=50]
long neck<>yes          [CF=50]
tawny color<>yes        [CF=50]
dark spot<>yes          [CF=50]
white color=yes         [CF=50]
black stripe=yes        [CF=50]
```

It becomes:

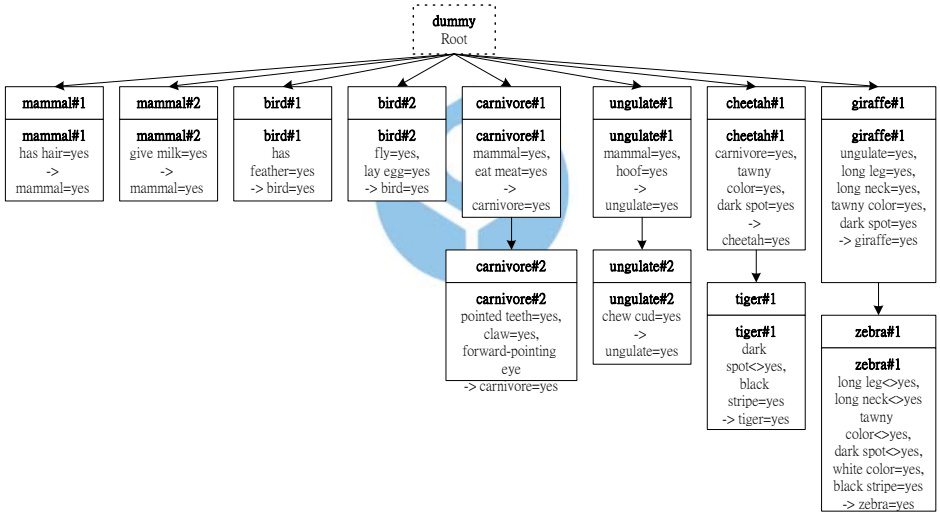
Node Structure	**Variable**	**Conclusion**
ParentNodeID: giraffe NodeID: zebra#1 NodeCFValue: 50 VariableID: long leg VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 0.333 VariableID: long neck VariableOrder: 2 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 0.667 VariableID: tawny color VariableOrder: 3 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.5 VariableID: dark spot VariableOrder: 4 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2.667 VariableID: white color VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.833 VariableID: black stripe VariableOrder: 6 VarOperator: <> VarValue: yes	VariableID: long leg NumOfNodeIDs: 2 VariableID: long neck NumOfNodeIDs: 2 VariableID: tawny color NumOfNodeIDs: 3 VariableID: dark spot NumOfNodeIDs: 4 VariableID: white color NumOfNodeIDs: 1 VariableID: black stripe NumOfNodeIDs: 2	ConclusionValue: zebra ParentNodeID: giraffe#1 NodeID: zebra#1

```
CFValue: 50
Credit: 1
VUR: 2

ParentNodeID: _root_
NodeID: giraffe#1
NodeCFValue: 50
VariableID: ungulate
VariableOrder: 1
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 0.4
VariableID: long leg
VariableOrder: 2
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 1.6
VariableID: long neck
VariableOrder: 3
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 2.4
VariableID:
tawny color
VariableOrder: 4
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 4.8
VariableID: dark spot
VariableOrder: 5
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 8

ParentNodeID: cheetah#1
NodeID: tiger#1
NodeCFValue: 50
VariableID: dark spot
VariableOrder: 1
VarOperator: <>
VarValue: yes
CFValue: 50
Credit: 1
VUR: 2
VariableID:
black stripe
VariableOrder: 2
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 1
VUR: 2

ParentNodeID: _root_
NodeID: cheetah#1
NodeCFValue: 50
VariableID: carnivore
VariableOrder: 1
VarOperator: =
VarValue: yes
CFValue: 50
Credit: 2
VUR: 0.667
VariableID:
tawny color
VariableOrder: 2
VarOperator: =
```

VarValue: yes CFValue: 50 Credit: 2 VUR: 4 VariableID: dark spot VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 8		

Node	**Rule**	**Rule Structure**
NodeID: zebra#1 NumOfVariableIDs: 6 NUR: 1.133 NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 3.44 NodeID: tiger#1 NumOfVariableIDs: 2 NUR: 2 NodeID: cheetah#1 NumOfVariableIDs: 3 NUR: 4.222	RuleID: zebra#1 NumOfNodeIDs: 2 RUR: 2.387 RuleID: giraffe#1 NumOfNodeIDs: 1 RUR: 3.44 RuleID: tiger#1 NumOfNodeIDs: 2 RUR: 3.111 RuleID: cheetah#1 NumOfNodeIDs: 3 RUR: 4.222	RuleID: zebra#1 NodeID: giraffe#1 NodeOrder: 1 NodeID: zebra#1 NodeOrder: 2

In terms of the tree structure:



- Case 13 is taken from rule Z13.

```
[RULE CF=50]      IF    bird=yes                [CF=50]     AND
                        does not fly=yes        [CF=50]     AND
                        long leg=yes            [CF=50]     AND
                        long neck=yes           [CF=50]     AND
                        black and white=yes     [CF=50]
                  THEN  ostrich=yes
```

Suppose the user chooses to create a top level node, it becomes:

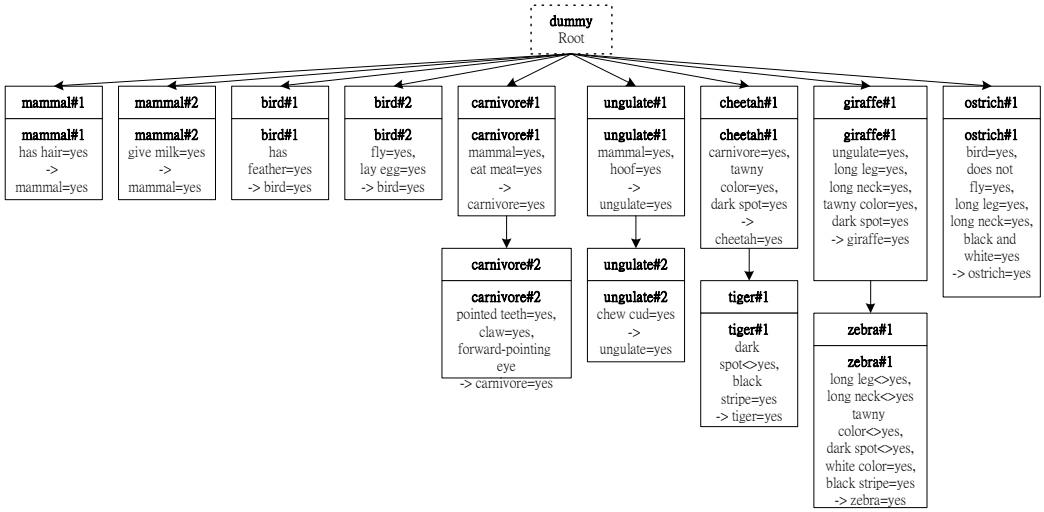
Node Structure	**Variable**	**Conclusion**
ParentNodeID: _root_ NodeID: ostrich#1 NodeCFValue: 50 VariableID: bird VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.2	VariableID: bird NumOfNodeIDs: 1 VariableID: does not fly NumOfNodeIDs: 1 VariableID: long leg NumOfNodeIDs: 3	ConclusionValue: ostrich ParentNodeID: _root_ NodeID: ostrich#1

VariableID: does not fly VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.4 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1.8 VariableID: long neck VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 2.4 VariableID: black and white VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: giraffe#1 NodeCFValue: 50 VariableID: ungulate VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.4 VariableID: long leg VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 2.4 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 3.6 VariableID: tawny color VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4.8 VariableID: dark spot VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 8 ParentNodeID: giraffe NodeID: zebra#1 NodeCFValue: 50 VariableID: long leg VariableOrder: 1 VarOperator: <>	VariableID: long neck NumOfNodeIDs: 3 VariableID: black and white NumOfNodeIDs: 1	

VarValue: yes CFValue: 50 Credit: 1 VUR: 0.5 VariableID: long neck VariableOrder: 2 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1 VariableID: tawny color VariableOrder: 3 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.5 VariableID: dark spot VariableOrder: 4 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2.667 VariableID: white color VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.833 VariableID: black stripe VariableOrder: 6 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2		

Node	**Rule**	**Rule Structure**
NodeID: ostrich#1 NumOfVariableIDs: 5 NUR: 1.16 NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 3.84 NodeID: zebra#1 NumOfVariableIDs: 2 NUR: 1.417	RuleID: ostrich#1 NumOfNodeIDs: 1 RUR: 1.16 NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 3.84 RuleID: zebra#1 NumOfNodeIDs: 2 RUR: 2.628	RuleID: ostrich#1 NodeID: ostrich#1 NodeOrder: 1

In terms of the tree structure:



- Case 14 is taken from rule Z14.

```
[RULE CF=50]      IF    bird=yes                  [CF=50]     AND
                        does not fly=yes          [CF=50]     AND
                        swim=yes                  [CF=50]     AND
                        black and white=yes       [CF=50]
                  THEN  penguin=yes
```

The user can choose either or both of the following conditions:

Condition 1	long leg<>yes [CF=50] long neck<>yes [CF=50]
Condition 2	swim=yes [CF=50]

Suppose she chooses both conditions 1 and 2, with the following order of variables:

```
long leg<>yes     [CF=50]
long neck<>yes    [CF=50]
swim=yes          [CF=50]
```

It becomes:

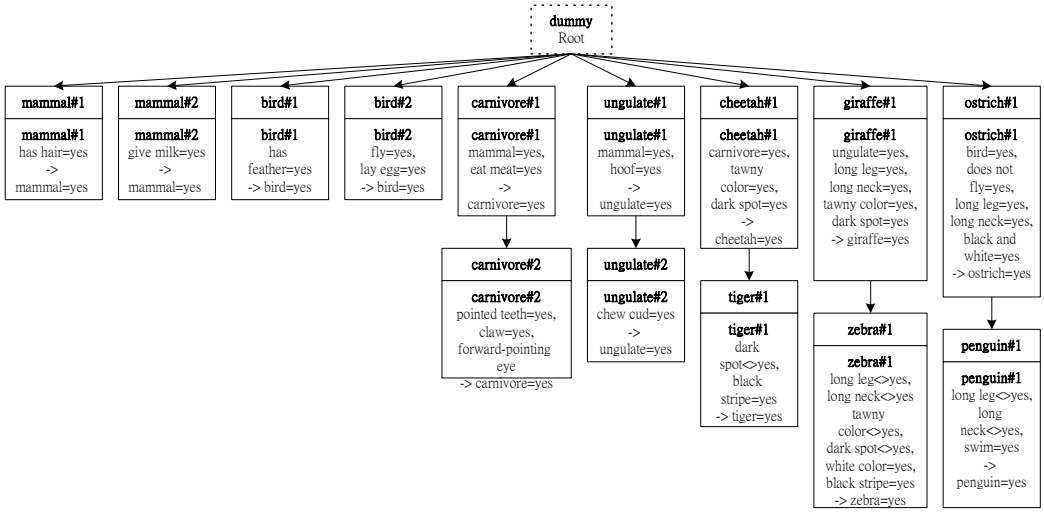
Node Structure	**Variable**	**Conclusion**
ParentNodeID: ostrich#1 NodeID: penguin#1 NodeCFValue: 50 VariableID: long leg VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.333 VariableID: long neck VariableOrder: 2 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2.667 VariableID: swim VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: ostrich#1 NodeCFValue: 50	VariableID: long leg NumOfNodeIDs: 4 VariableID: long neck NumOfNodeIDs: 4 VariableID: swim NumOfNodeIDs: 1	ConclusionValue: penguin ParentNodeID: ostrich#1 NodeID: penguin#1

VariableID: bird VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.4 VariableID: does not fly VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.8 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4.8 VariableID: long neck VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 6.4 VariableID: black and white VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 2 ParentNodeID: _root_ NodeID: giraffe#1 NodeCFValue: 50 VariableID: ungulate VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.4 VariableID: long leg VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 3.2 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4.8 VariableID: tawny color VariableOrder: 4 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4.8 VariableID: dark spot VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 8		

ParentNodeID: giraffe NodeID: zebra#1 NodeCFValue: 50 VariableID: long leg VariableOrder: 1 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 0.667 VariableID: long neck VariableOrder: 2 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.333 VariableID: tawny color VariableOrder: 3 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 1.5 VariableID: dark spot VariableOrder: 4 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2.667 VariableID: white color VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 0.833 VariableID: black stripe VariableOrder: 6 VarOperator: <> VarValue: yes CFValue: 50 Credit: 1 VUR: 2		

Node	**Rule**	**Rule Structure**
NodeID: penguin#1 NumOfVariableIDs: 5 NUR: 1.667 NodeID: ostrich#1 NumOfVariableIDs: 5 NUR: 2.88 NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 4.24 NodeID: zebra#1 NumOfVariableIDs: 2 NUR: 1.5	RuleID: penguin#1 NumOfNodeIDs: 2 RUR: 2.273 RuleID: ostrich#1 NumOfNodeIDs: 1 RUR: 2.88 NodeID: giraffe#1 NumOfVariableIDs: 5 NUR: 4.24 RuleID: zebra#1 NumOfNodeIDs: 2 RUR: 2.87	RuleID: penguin#1 NodeID: ostrich#1 NodeOrder: 1 NodeID: penguin#1 NodeOrder: 2

In terms of the tree structure:



- Case 15 is taken from rule Z15.

```
[RULE CF=50]      IF    bird=yes          [CF=50]     AND
                        good flyer=yes    [CF=50]
                  THEN  albatross=yes
```

Suppose the user chooses to create a top level node, it becomes:

Node Structure	**Variable**	**Conclusion**
ParentNodeID: _root_ NodeID: albatross#1 NodeCFValue: 50 VariableID: bird VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 VariableID: good flyer VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 1 VUR: 1 ParentNodeID: _root_ NodeID: ostrich#1 NodeCFValue: 50 VariableID: bird VariableOrder: 1 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.8 VariableID: does not fly VariableOrder: 2 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 0.8 VariableID: long neck VariableOrder: 3 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 4.8 VariableID: long neck VariableOrder: 4	VariableID: bird NumOfNodeIDs: 2 VariableID: good flyer NumOfNodeIDs: 1	ConclusionValue: albatross ParentNodeID: _root_ NodeID: albatross#1

VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 6.4 VariableID: black and white VariableOrder: 5 VarOperator: = VarValue: yes CFValue: 50 Credit: 2 VUR: 2		

Node	**Rule**	**Rule Structure**
NodeID: albatross#1 NumOfVariableIDs: 2 NUR: 1 NodeID: ostrich#1 NumOfVariableIDs: 5 NUR: 2.96	RuleID: albatross#1 NumOfNodeIDs: 1 RUR: 1 RuleID: ostrich#1 NumOfNodeIDs: 1 RUR: 2.96 RuleID: penguin#1 NumOfNodeIDs: 2 RUR: 2.313	RuleID: albatross#1 NodeID: albatross#1 NodeOrder: 1

Figure A.1 depicts the rule tree of the knowledge base after the last case (case 15) is entered.

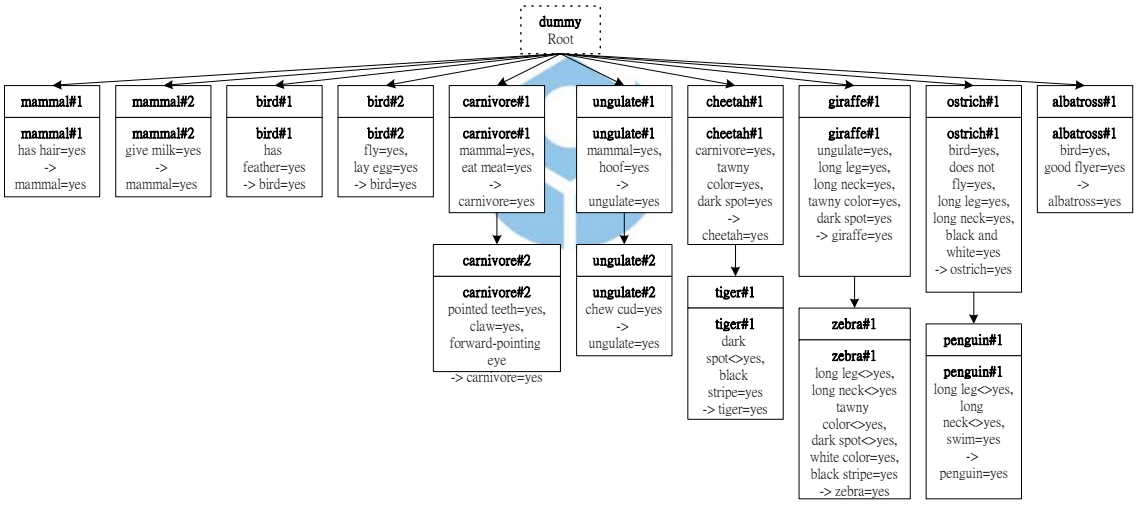


Figure A.1 Zookeeper knowledge base in VCIRS

Every time the user enters a case; the structure of knowledge base along with VUR, NUR and RUR also changes.

Appendix B

Computing Relative Node Order

We use the “Computing relative node order algorithm” (Figure 3.11) to obtain the relative order of the nodes in the rules as shown in Table B.1.

Figure B.1 The relative node order in the knowledge base

Step	CurrentRule (RUR)	NodeOrderQ (NUR)	RuleUsed (RUR)	Pre CandidateNode (NUR)	CandidateNode	RuleStack
1	bird#2 (0.75)	bird#2 (0.75)	bird#2 (0.75)	bird#2	bird#2	
2	albatross#1 (1)	albatross#1 (1)	albatross#1 (1)	albatross#1	bird#2 albatross#1	bird#2
3	bird#1 (1)	bird#1 (1)	bird#1 (1)	bird#1	bird#2 albatross#1 bird#1	albatross#1 bird#2
4	mammal#2 (1)	mammal#2 (1)	mammal#2 (1)	mammal#2	bird#2 albatross#1 bird#1 mammal#2	bird#1 albatross#1 bird#2
5	mammal#1 (1)	mammal#1 (1)	mammal#1 (1)	mammal#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1	mammal#2 bird#1 albatross#1 bird#2
6	carnivore#2 (1.08)	carnivore#1 (1.5) carnivore#2 (1.67)	carnivore#2 (1.08) carnivore#1 (1.5)	carnivore#1 (1.5) carnivore#1 (1.5)	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1	mammal#1 mammal#2 bird#1 albatross#1 bird#2
7	carnivore#2 (1.08)	carnivore#2 (1.67)	carnivore#2 (1.08) carnivore#1 (1.5)	carnivore#2	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2	mammal#1 mammal#2 bird#1 albatross#1 bird#2
8	carnivore#1 (1.5)	carnivore#1 (1.5)	carnivore#1 (1.5)	carnivore#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2	carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
9	ungulate#2 (1.25)	ungulate#1 (1.5) ungulate#2 (1)	ungulate#2 (1.25) ungulate#1 (1.5)	ungulate#1 (1.5) ungulate#1 (1.5)	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1	carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
10	ungulate#2 (1.25)	ungulate#2 (1)	ungulate#2 (1.25) ungulate#1 (1.5)	ungulate#2	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2	carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
11	ungulate#1 (1.5)	ungulate#1 (1.5)	ungulate#1 (1.5)	ungulate#1	bird#2 albatross#1	ungulate#2 carnivore#1

					bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2	carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
12	penguin#1 (2.31)	ostrich#1 (2.96) penguin#1 (1.67)	penguin#1 (2.31) ostrich#1 (2.96)	ostrich#1 (2.96) ostrich#1 (2.96)	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1	ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
13	penguin#1 (2.31)	penguin#1 (1.67)	ostrich#1 (2.31) ostrich#1 (2.96)	penguin#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1	ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
14	ostrich#1 (2.96)	ostrich#1 (2.96)	ostrich#1 (2.96)	ostrich#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1	penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
15	zebra#1 (2.87)	giraffe#1 (4.24) zebra#1 (1.57)	zebra#1 (2.87) giraffe#1 (4.24)	giraffe#1 (4.24) giraffe#1 (4.24)	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1	ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
16	zebra#1 (2.87)	zebra#1 (1.57)	zebra#1 (2.87) giraffe#1 (4.24)	zebra#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1	ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
17	giraffe#1 (4.24)	giraffe#1 (4.24)	giraffe#1 (4.24)	giraffe#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1	zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2

					zebra#1	
18	tiger#1 (3.11)	cheetah#1 (4.22) tiger#1 (2)	tiger#1 (3.11) cheetah#1 (4.22)	cheetah#1 (4.22) cheetah#1 (4.22)	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1	giraffe#1 zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
19	tiger#1 (3.11)	tiger#1 (2)	tiger#1 (3.11) cheetah#1 (4.22)	tiger#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1 tiger#1	giraffe#1 zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
20	cheetah#1 (4.22)	cheetah#1 (4.22)	cheetah#1 (4.22)	cheetah#1	bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1 tiger#1	tiger#1 giraffe#1 zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2
21					bird#2 albatross#1 bird#1 mammal#2 mammal#1 carnivore#1 carnivore#2 ungulate#1 ungulate#2 ostrich#1 penguin#1 giraffe#1 zebra#1 cheetah#1 tiger#1	cheetah#1 tiger#1 giraffe#1 zebra#1 ostrich#1 penguin#1 ungulate#1 ungulate#2 carnivore#1 carnivore#2 mammal#1 mammal#2 bird#1 albatross#1 bird#2

Appendix C

Computing Relative Variable Order

We use the "Computing relative variable order algorithm" (Figure 3.14) to obtain the relative order of the variables in the nodes as shown in Table C.1.

Table C.1 The relative variable order in the knowledge base

Step	CurrentNode (NUR)	VarOrderQ (VUR)	NodeUsed (NUR)	Pre CandidateVar (VUR)	CandidateVar	NodeStack
1	carnivore#2 (0.67)	pointed teeth=yes (0.33) claw=yes (0.67) forward-pointing eye=yes (1)	carnivore#2 (0.67)	pointed teeth=yes claw=yes forward-pointing eye=yes	pointed teeth=yes claw=yes forward-pointing eye=yes	
2	bird#2 (0.75)	fly=yes (0.5) lay egg=yes (1)	bird#2 (0.75)	fly=yes lay egg=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes	carnivore#2
3	ungulate#2 (1)	chew cud=yes (1)	ungulate#2 (1)	chew cud=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes	bird#2 carnivore#2
4	bird#1 (1)	has feather=yes (1)	bird#1 (1)	has feather=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes	ungulate#2 bird#2 carnivore#2
5	mammal#2 (1)	give milk=yes (1)	mammal#2 (1)	give milk=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes	bird#1 ungulate#2 bird#2 carnivore#2
6	mammal#1 (1)	has hair=yes (1)	mammal#1 (1)	has hair=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes	mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
7	albatross#1 (1)	bird=yes (1) good flyer=yes (1)	albatross#1 (1) ostrich#1 (2.96)	bird=yes (1) bird=yes (0.8)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes	mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

					has hair=yes bird=yes	
8	albatross#1 (1)	good flyer=yes (1)	albatross#1 (1) ostrich#1 (2.96)	good flyer=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes	mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
9	ostrich#1 (2.96)	does not fly=yes (0.8) long leg=yes (4.8) long neck=yes (6.4) black and white=yes (2)	ostrich#1 (2.96)	does not fly=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes	albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
10	ostrich#1 (2.96)	long leg=yes (4.8) long neck=yes (6.4) black and white=yes (2)	ostrich#1 (2.96) giraffe#1 (4.24)	long leg=yes (4.8) ungulate=yes (0.4) long leg=yes (3.2)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes	albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
11	ostrich#1 (2.96)	long neck=yes (6.4) black and white=yes (2)	ostrich#1 (2.96) giraffe#1 (4.24)	long neck=yes (6.4) long neck=yes (4.8)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes	albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
12	ostrich#1 (2.96)	black and white=yes (2)	ostrich#1 (2.96) giraffe#1 (4.24)	black and white=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes	albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

13	giraffe#1 (4.24)	tawny color=yes (4.8) dark spot=yes (8)	giraffe#1 (4.24) cheetah#1 (4.22)	tawny color=yes (4.8) carnivore=yes (0.67) tawny color=yes (4)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes	ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
14	giraffe#1 (4.24)	dark spot=yes (8)	giraffe#1 (4.24) cheetah#1 (4.22)	dark spot=yes (8) dark spot=yes (8)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes	ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
15	ungulate#1 (1.5)	mammal=yes (2) hoof=yes (1)	ungulate#1 (1.5) carnivore#1 (1.5)	mammal=yes (2) mammal=yes (2)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes	giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
16	ungulate#1 (1.5)	hoof=yes (1)	ungulate#1 (1.5) carnivore#1 (1.5)	hoof=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes	giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

					ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes	
17	carnivore#1 (1.5)	eat meat=yes (1)	carnivore#1 (1.5)	eat meat=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes	ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
18	zebra#1 (1.5)	long leg<>yes (0.67) long neck<>yes (1.33) tawny color<>yes (1.5) dark spot<>yes (2.67) white color=yes (0.83) black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67)	long leg<>yes (0.67) long leg<>yes (1.33)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
19	zebra#1 (1.5)	long neck<>yes (1.33) tawny color<>yes (1.5) dark spot<>yes (2.67) white color=yes (0.83) black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67)	long neck<>yes (1.33) long neck<>yes (2.67)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

					white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes	
20	zebra#1 (1.5)	tawny color<>yes (1.5) dark spot<>yes (2.67) white color=yes (0.83) black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67)	tawny color<>yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
21	zebra#1 (1.5)	dark spot<>yes (2.67) white color=yes (0.83) black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67) tiger#1 (3.11)	dark spot<>yes (2.67) dark spot<>yes (2)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
22	zebra#1 (1.5)	white color=yes (0.83) black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67) tiger#1 (3.11)	white color=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

					does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes	
23	zebra#1 (1.5)	black stripe=yes (2)	zebra#1 (1.5) penguin#1 (1.67) tiger#1 (3.11)	black stripe=yes (2) black stripe=yes (2)	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes black stripe=yes	carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2
24	penguin#1 (2.31)	swim=yes (1)	penguin#1 (2.31)	swim=yes	pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes black stripe=yes	tiger#1 zebra#1 carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

					swim=yes	
25					pointed teeth=yes claw=yes forward-pointing eye=yes fly=yes lay egg=yes chew cud=yes has feather=yes give milk=yes has hair=yes bird=yes good flyer=yes does not fly=yes ungulate=yes long leg=yes long neck=yes black and white=yes carnivore=yes tawny color=yes dark spot=yes mammal=yes hoof=yes eat meat=yes long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes white color=yes black stripe=yes swim=yes	penguin#1 tiger#1 zebra#1 carnivore#1 ungulate#1 giraffe#1 cheetah#1 ostrich#1 albatross#1 mammal#1 mammal#2 bird#1 ungulate#2 bird#2 carnivore#2

Appendix D

Confirmation of Generated Node

We compose the combination of variables to generate a complete node, according to the relative variable order obtained in Appendix C. The system will confirm to the user about the node being generated before it saves as the additional nodes knowledge base, whether a node is make sense or not. And its make sense after we observe that there are some incorrectness in the generated node as depicted in the Table D.1 below.

Table D.1 Variable combination based on relative variable order followed by confirmation

Last VariableID	NodeID	VariableOrder	% Correct
dark spot	giraffe#1G	~~pointed teeth=yes~~ ~~claw=yes~~ ~~forward-pointing eye=yes~~ ~~fly=yes~~ ~~lay egg=yes~~ chew cud=yes has feather=yes give milk=yes has hair=yes ~~bird=yes~~ ~~good flyer=yes~~ does not fly=yes ungulate=yes long leg=yes long neck=yes ~~black and white=yes~~ ~~carnivore=yes~~ tawny color=yes dark spot=yes	53%
dark spot	tiger#1G	pointed teeth=yes claw=yes forward-pointing eye=yes ~~fly=yes~~ ~~lay egg=yes~~ ~~chew cud=yes~~ ~~has feather=yes~~ give milk=yes has hair=yes ~~bird=yes~~ ~~good flyer=yes~~ does not fly=yes ~~ungulate=yes~~ ~~long leg=yes~~ ~~long neck=yes~~ ~~black and white=yes~~ carnivore=yes tawny color=yes ~~dark spot=yes~~ mammal=yes ~~hoof=yes~~ eat meat=yes long leg<>yes long neck<>yes ~~tawny color<>yes~~ dark spot<>yes	50%
long neck	ostrich#1G	~~pointed teeth=yes~~	47%

		~~claw=yes~~ ~~forward-pointing eye=yes~~ ~~fly=yes~~ lay egg=yes chew cud=yes has feather=yes ~~give milk=yes~~ ~~has hair=yes~~ bird=yes ~~good flyer=yes~~ does not fly=yes ~~ungulate=yes~~ long leg=yes long neck=yes	
dark spot	zebra#1G	~~pointed teeth=yes~~ ~~claw=yes~~ ~~forward-pointing eye=yes~~ ~~fly=yes~~ ~~lay egg=yes~~ chew cud=yes ~~has feather=yes~~ give milk=yes has hair=yes ~~bird=yes~~ ~~good flyer=yes~~ does not fly=yes ungulate=yes ~~long leg=yes~~ ~~long neck=yes~~ ~~black and white=yes~~ ~~carnivore=yes~~ ~~tawny color=yes~~ ~~dark spot=yes~~ mammal=yes hoof=yes ~~eat meat=yes~~ long leg<>yes long neck<>yes tawny color<>yes dark spot<>yes	42%
dark spot	cheetah#1G	pointed teeth=yes claw=yes forward-pointing eye=yes ~~fly=yes~~ ~~lay egg=yes~~ ~~chew cud=yes~~ ~~has feather=yes~~ give milk=yes has hair=yes ~~bird=yes~~ ~~good flyer=yes~~ does not fly=yes ~~ungulate=yes~~ ~~long leg=yes~~ ~~long neck=yes~~ ~~black and white=yes~~ ~~carnivore=yes~~ tawny color=yes dark spot=yes	40%
long neck	penguin#1G	~~pointed teeth=yes~~ ~~claw=yes~~ ~~forward-pointing eye=yes~~ ~~fly=yes~~ lay egg=yes ~~chew cud=yes~~ has feather=yes ~~give milk=yes~~ ~~has hair=yes~~ bird=yes ~~good flyer=yes~~ does not fly=yes ~~ungulate=yes~~ ~~long leg=yes~~ ~~long neck=yes~~ black and white=yes ~~carnivore=yes~~	29%

		~~tawny color=yes~~ ~~dark spot=yes~~ ~~mammal=yes~~ ~~hoof=yes~~ ~~eat meat=yes~~ long leg<>yes long neck<>yes	

Appendix E
Implementation Details

E.1 System Specifications

We implement our proposed method, VCIRS, under the system specification below.

Computer

- Processor: Intel® Celeron™ CPU 1000 MHz
- 998 MHz, 512 MB of RAM

Operating System

- Microsoft® Windows™ XP Professional Version 2002 Service Pack 2

Compiler

- Borland® Delphi™ Professional Version 7.0

Database

- Microsoft® Access™ 2002

E.2 Database Structure

The system architecture is implemented in the MS Access database consisting one main table and several supporting tables. The main table is the NodeStructure table to implement the Node Structure which is of course based on the variables.

The names (ID) of variables, nodes and rules are implemented by separate Variable, Node and Rule tables respectively. The conclusion of a node/rule is implemented by Conclusion table.

The up-to-date structure of the node is maintained by the NodeStructure table and then it constitutes the rules in the RuleStructure table. The occurrence of variables during knowledge building is saved into the NodeStructure table in the Credit field. Calculation of information updating for the Refinement Module is achieved by the NodeStructure table along with the Rule, RuleStructure and Node tables.

In the following we will describe how the tables are related and the tables themselves.

E.2.1 Relationships Diagram

Figure E.1 describes the relationships between the tables in the system, under the Microsoft Access relationship diagram specification.

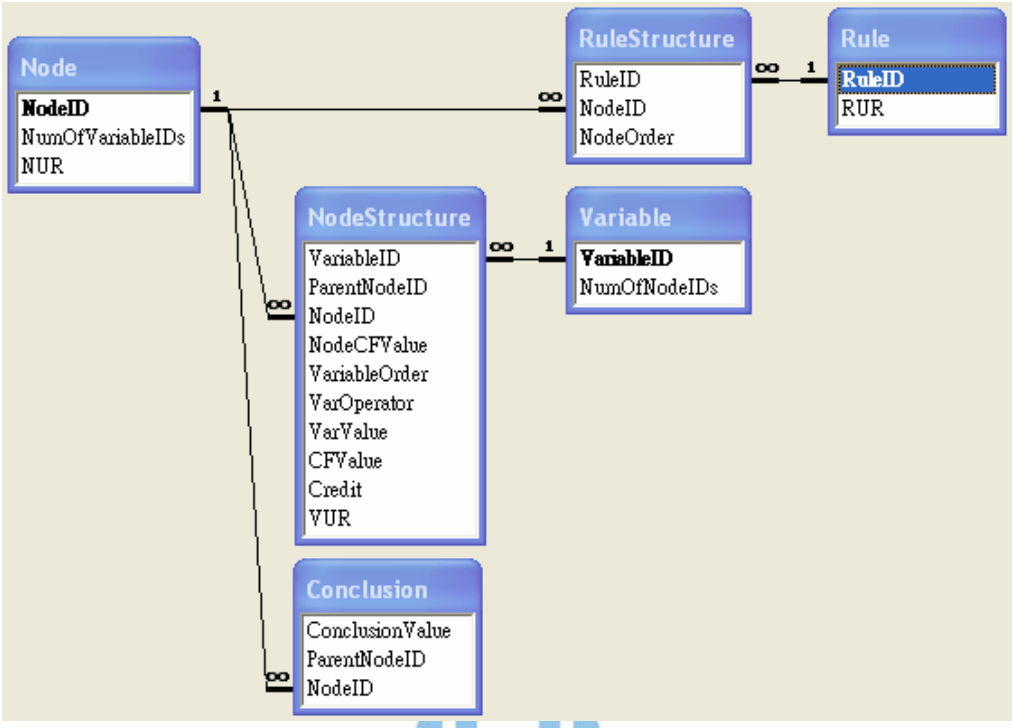


Figure E.1 Relationships diagram

We strictly enforced the referential integrity to guarantee that each value in the tables have the relation with others and then can be process further. From the figure, we see the Rule Structure is implemented by both the RuleStructure table and the Rule table. The Node Structure is implemented by the NodeStructure table together with the Node table, which calculates the NUR. The calculation of the RUR is supported by the RuleStructure and Node tables and is saved in the Rule table.

E.2.2 Node Structure

Figure E.2 describes the fields of the NodeStructure table.

Field Name	Data Type
VariableID	Text
ParentNodeID	Text
NodeID	Text
NodeCFValue	Number
VariableOrder	Number
VarOperator	Text
VarValue	Text
CFValue	Number
Credit	Number
VUR	Number

Figure E.2 NodeStructure table fields

VariableID, ParentNodeID, NodeID, VarValue are alphanumeric so their types are Text, while VariableOrder is integer. ParentNodeID value is the same as NodeID, only referring to the parent node of some node. VarOperator's type is text which has 2 character at the most, i.e., "<" | "<=" | ">" | ">=" | "=" | "<>". NodeCFValue and CFValue types are the integer type with the range [1...100]. One is both the default and the lowest CF to get the certain result while inferencing.

Credit is the field to record the occurrence of a variable during knowledge building under the condition that the variable has the same value in both the old and new cases. VUR is a real number, which is updated when a node is created during knowledge building.

E.2.3 Rule Structure

Figure E.3 describes the fields of the RuleStructure table.

	Field Name	Data Type
▶	RuleID	Text
	NodeID	Text
	NodeOrder	Text

Figure E.3 RuleStructure table fields

RuleID and NodeID are alphanumeric. The value for RuleID is the same as the value for NodeID. NodeOrder is the order of the node in the rule. It starts from 0 for the root and one for the child, two for the child of the child (grandchild) and so on. NodeOrder is integer.

E.2.4 Rule

Figure E.4 describes the fields of the Rule table.

	Field Name	Data Type
🔑▶	RuleID	Text
	RUR	Number

Figure E.4 Rule table fields

RuleID is the primary key, i.e., has the unique value. This unique value guarantees SQL (Syntax Query Language) gets the result when it executes. A rule means a sequence of nodes, and the name and conclusion of the rule is obtained from the last node in a path of nodes.

RUR (Rule Usage Rate) is a real number. This field is updated while a node is created during knowledge building.

E.2.5 Node

Figure E.5 describes the fields of the Node table.

Field Name	Data Type
NodeID	Text
NumOfVariableIDs	Number
NUR	Number

Figure E.5 Node table fields

NodeID is the primary key. The field of NumOfVariableIDs is integer and used to save the number of variables in a node. NUR (Node Usage Rate) is real, which is updated while a node is created during knowledge building.

E.2.6 Variable

Figure E.6 describes the fields of the Variable table.

Field Name	Data Type
VariableID	Text
NumOfNodeIDs	Text

Figure E.6 Variable table fields

VariableID is the primary key. It is used in the NodeStructure table. The field of NumOfNodeIDs is integer and used to save the number of nodes which share the variable.

E.2.7 Conclusion

Figure E.7 describes the fields of the Conclusion table.

Field Name	Data Type
ConclusionValue	Text
ParentNodeID	Text
NodeID	Text

Figure E.7 Conclusion table fields

This table saves the conclusion part from the case posted by the user. A node (rule) may have more than one conclusion value. ParentNodeID and NodeID are alphanumeric so it types are Text.

E.3 Program Modules

This section implements the three operations, namely, knowledge refinement, knowledge building and knowledge inferencing as described in the previous chapter.

E.3.1 Knowledge Building

The knowledge building process is similar to RDR, i.e., it can build knowledge base from scratch, when knowledge base is not available, but the user wants to build a new knowledge base. The system builds a new knowledge base on the cases provided by the user. The cases will be saved in the Variable-Centered Rule Structure. It becomes the sources for knowledge refining.

Input from the user is the case and it has a certain fields. A case looks like a rule in RBS, which contains a clause part and a conclusion part. In VCIRS, the clause part is represented by the variables and the conclusion part by the conclusions. For knowledge inferencing purpose, in addition, the user put a CF value for her case (like the CF value for a rule in the RBS) as well as a CF value for each single variable she entered. The calculation of CF will be described in the next section, i.e., knowledge inferencing.

All algorithms involved in building the knowledge base are described in Section 3.5. The tables for recording the cases posted by the user are tables of Variable, Node, NodeStructure, Conclusion, Rule and RuleStructure.

Figure E.8 describes the initialization of variables and nodes from the case posted by the user. A variable name becomes a VariableID and the first conclusion value becomes a NodeID. Both VariableID and NodeID are unique (i.e., primary key). Node CF Value is saved in NodeCFValue's field. Once VariableID and NodeID are initialized, it can be used in the tables. For each variable input, the system saves its ID, order, variable operator, value and CF value into VariableID, VariableOrder, VarOperator, VarValue, and CFValue in the NodeStructure, respectively.

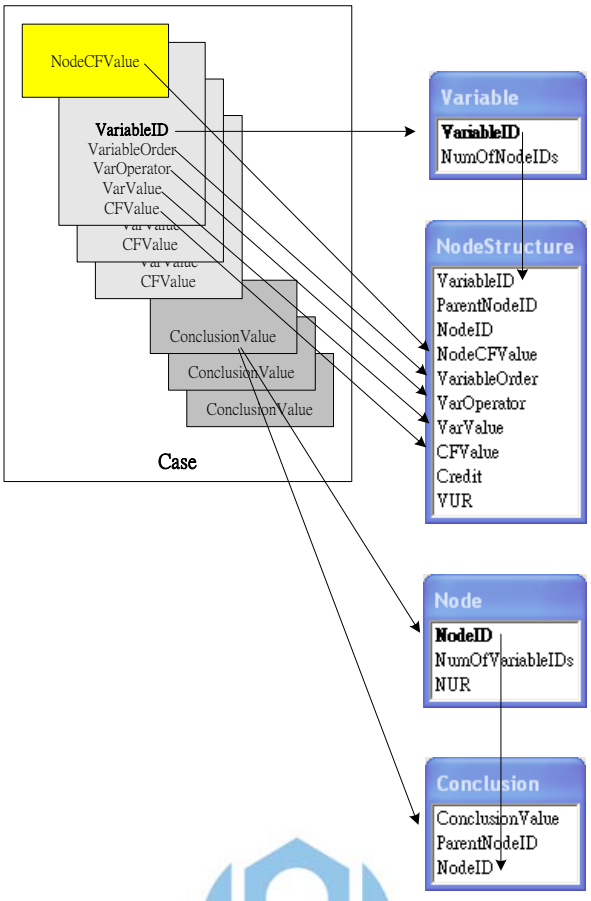


Figure E.8 Initialization of variables and nodes

The conclusion value goes to ConclusionValue along with the new NodeID in the Conclusion table. In both NodeStructure and Conclusion tables, the ParentNodeID is the root if the new NodeID is a top level node; otherwise the ParentNodeID is the NodeID of the higher level node that the new node refers to.

Figure E.9 describes the next step, which initializes credit and VUR for the new node. Credit is set to 1 and VUR = variable order/total variables for a new node.

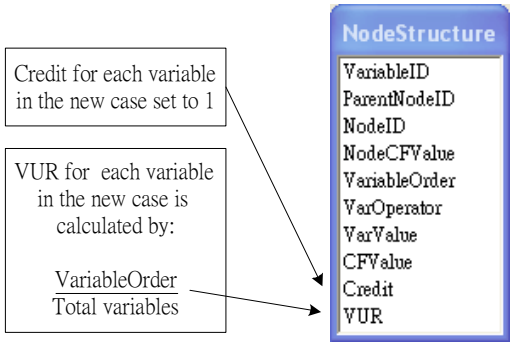


Figure E.9 Credit and VUR initialization

Then system checks the parent node of the new node, if the parent node is not the root. The credit and VUR of the parent node will be updated accordingly. Figure E.10 describes these events.

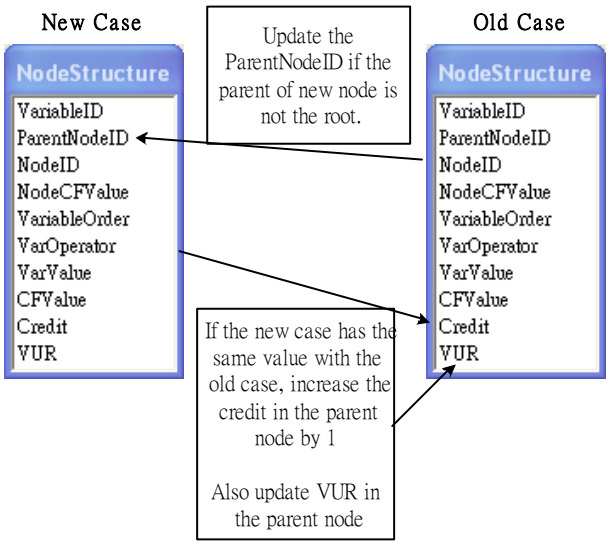


Figure E.10 Credit and VUR updating for the parent node

Figure E.11 shows the next step, where the number of variables from the case is saved in the NumOfVariableIDs' field of the Node table. NUR is obtained by averaging VURs from all variables a node has.

NodeID from the new node become a RuleID in the Rule table. The number of nodes a rule has is updated, increased by one. RUR is obtained by averaging NURs from all nodes a rule has. In the RuleStructure table, NodeID from a new node is saved along with its order in the rule.

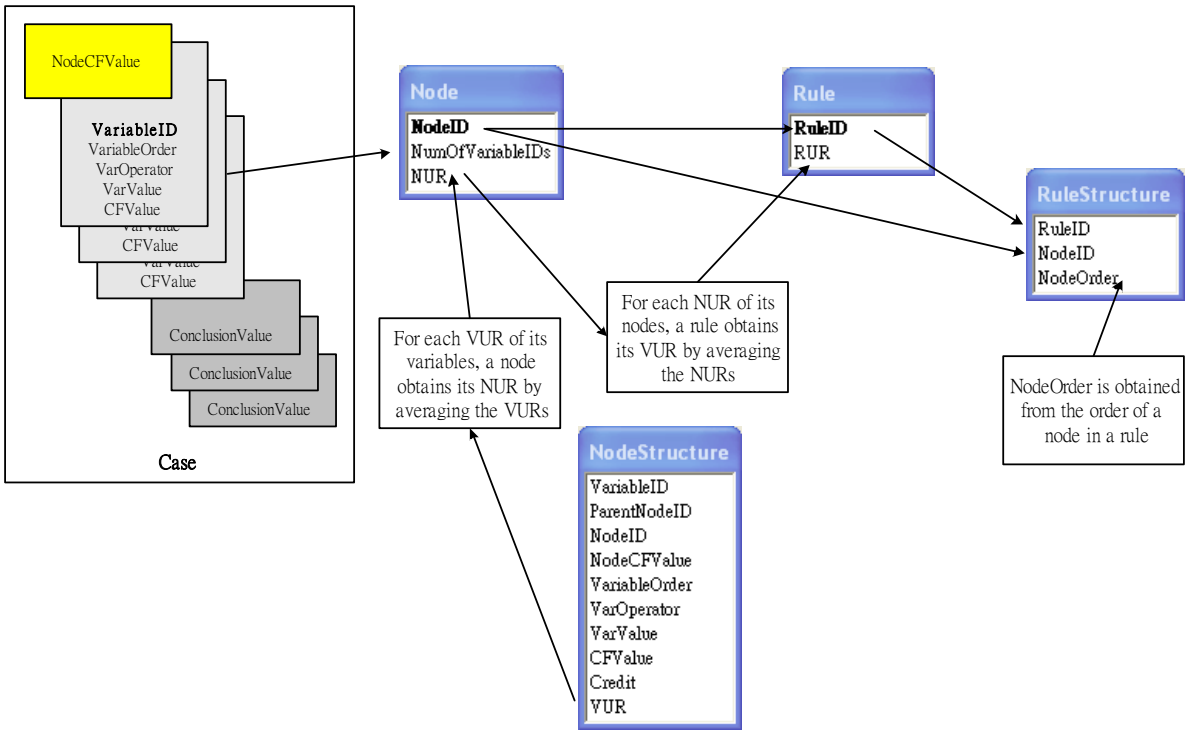


Figure E.11 Rule initialization and value updating

Knowledge building process will repeat the above each steps until the user stops the process. VUR, NUR and RUR obtained at any time point help the user to see which variables/nodes/rules are the most used so far. It can be retrieved anytime, because the calculation is in the same time as the knowledge building process goes on. It also can be used in the knowledge inferencing process for guiding the user to focus on most used variables/nodes/rules.

E.3.2 Knowledge Inferencing

VCIRS provides RDR-like reasoning (i.e., a simple forward chaining). It also supports RBS-based inferencing, by transforming the knowledge base into the rule base. Inferencing in RBS involves forward and backward chaining. A rule in the rule base uses Confidence Factor (CF). CF supports both forward and backward chaining to improve precision and value. In the following we will describe the implementation of RDR and RBS inferencing approach.

E.3.2.1 RDR Inferencing

VCIRS will search the knowledge base for the proper node. The implementation of the proper node algorithm of Figure 3.16 is shown in Figure E.12.

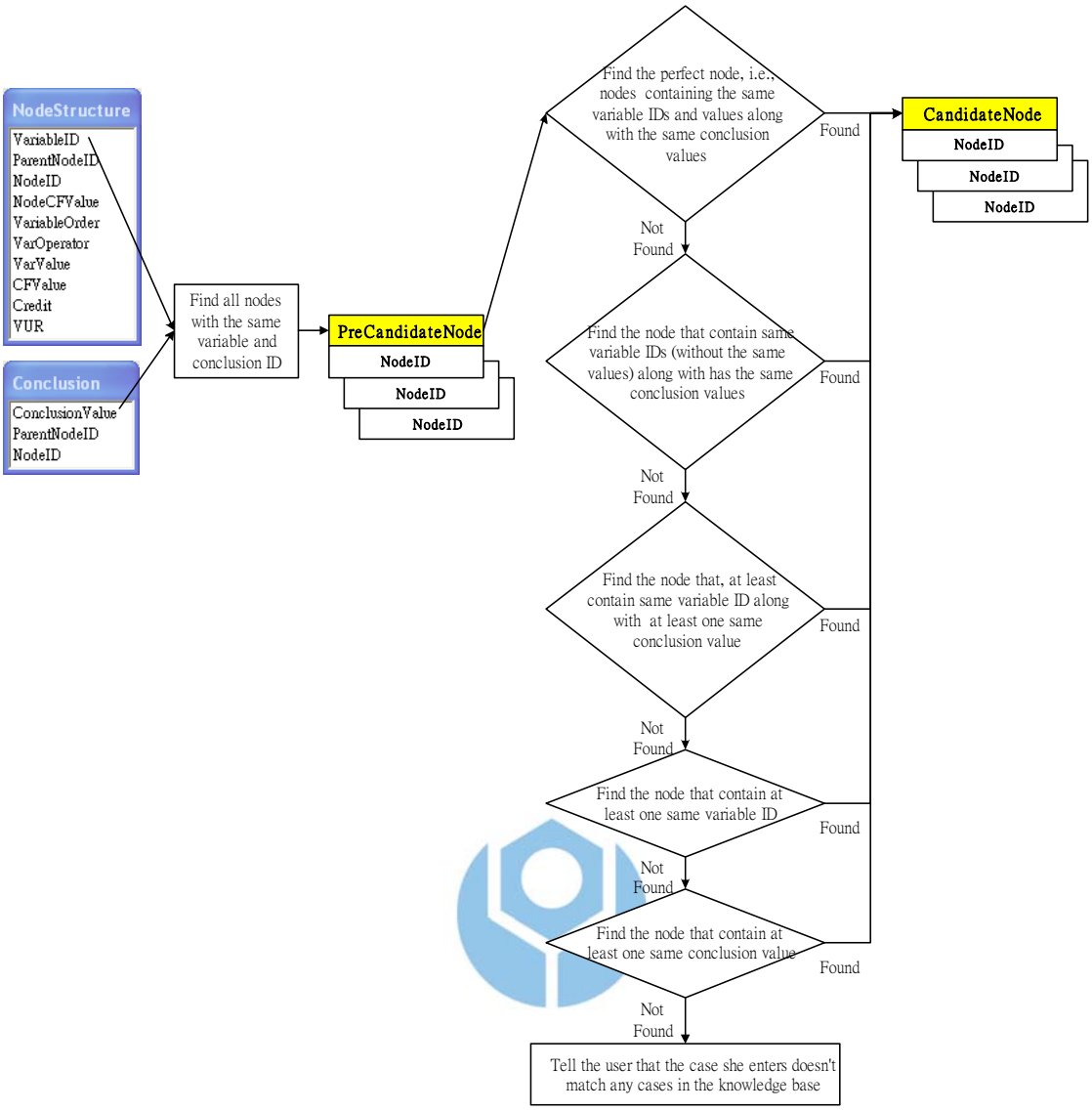


Figure E.12 Proper node finding implementation

The implementation of RDR inferencing mechanism is shown in Figure E.13.

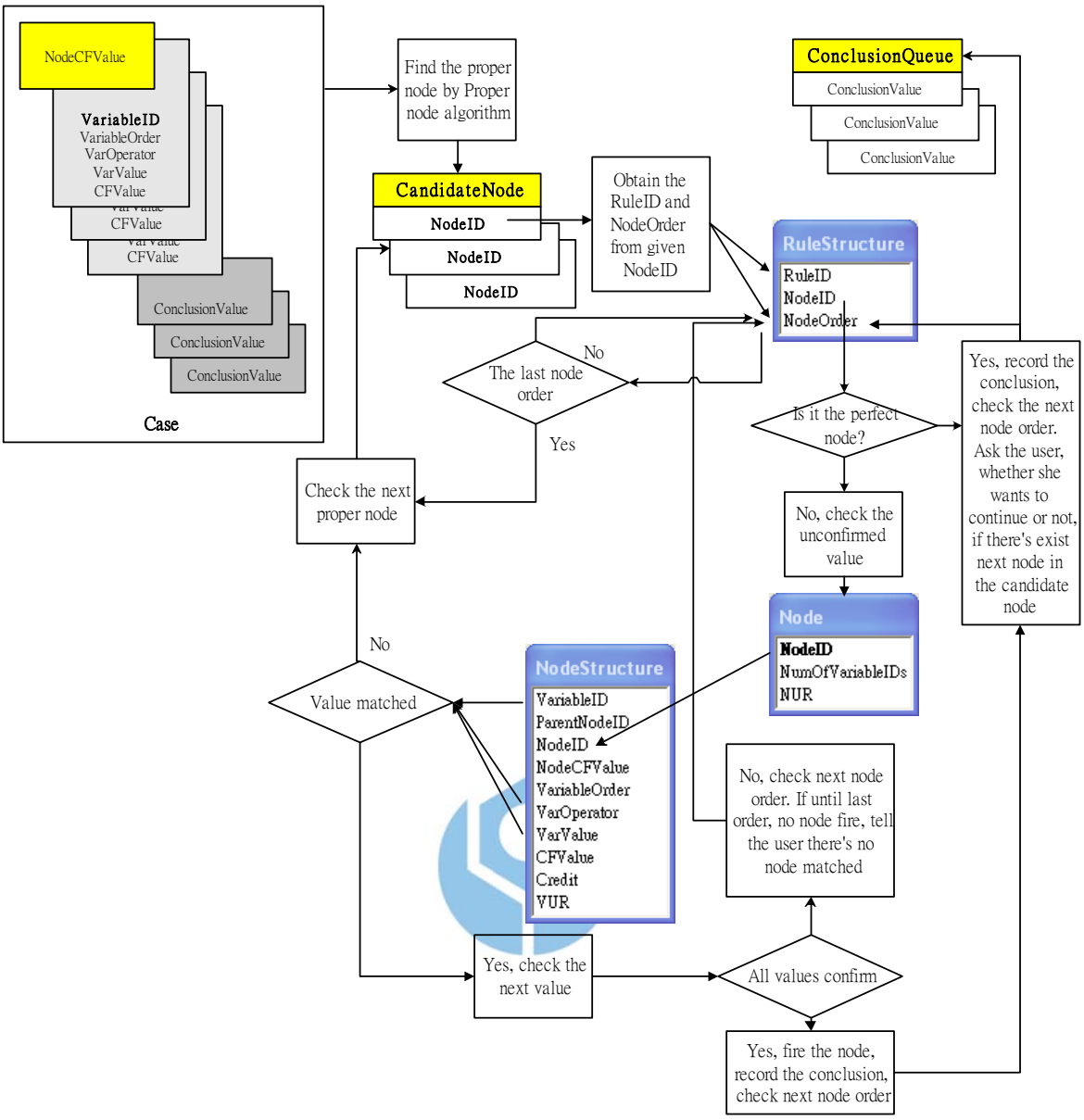


Figure E.13 RDR inferencing implementation

E.3.2.2 RBS Inferencing

Before performing forward and backward chaining, the system needs to do initialization of the structures from the rule base (i.e., the result of knowledge base after rule base transformation). The initialization step is used in both forward and backward chaining process. Implementation of forward chaining is shown in the Figure E.16, and backward chaining in Figure E.17.

Initializations

The system scans all the rules obtained from the transformed rule base. Then it builds a BaseVariableList as described in the Figure E.14. The purpose of this structure is the store the RBS rules for comparing the variables from VariableList (i.e., it shows below) to the clause and conclusion part of BaseVariableList, when the inferencing in progress. BaseVariableList represents the existing conditions of rules, while VariableList represents the conditions from the user confirmation or the result of firing rule.

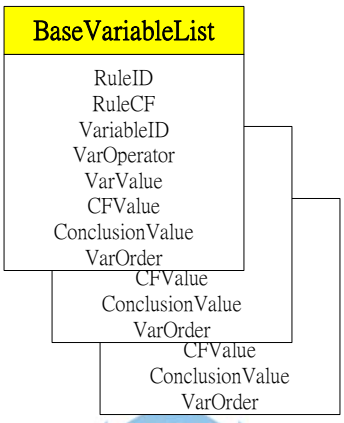


Figure E.14 BaseVariableList structure

Then system then creates a VariableList to contain only the clause part of the rule, as depicted in Figure E.15.

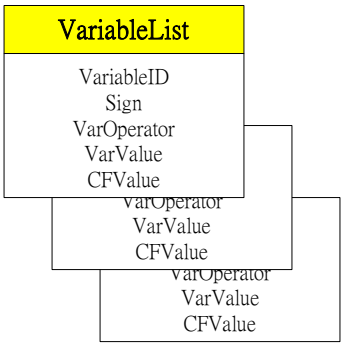


Figure E.15 VariableList structure

In VariableList the value of VariableID is the same as the value of VariableID in the BaseVariableList. “Sign” has a default value “NO”, meaning that VariableID does not have an instantiation. VarOperator, VarValue and CFValue are all set to “empty”.

Forward Chaining

1. Clause part identification. The user is asked by the system about the facts she knows. A fact may contain VariableID, VarOperator, VarValue and CFValue.
2. VariableID is entered into the queue (ConclusionVariableQueue) while VariableID, VarOperator and VarValue are saved into the VariableList.
3. A search is conducted to find a VariableID in the BaseVariableList with the same name as the VariableID in the first entry of the queue. If a VariableID found, the RuleID and NumOfVariableIDs of the variable are saved. Initialize the CurrentVarOrder. If no VariableID found then jump to step 6.
4. For each variable found in the clause part of the rule, check whether a VariableID is instantiated or not (i.e., Sign = “NO” means not instantiate, “YES” means instantiated). It has to have instantiation with VarOperator, VarValue and CFValue either from the user or the rules itself.
5. Next, matching process is conducted to check if the fact entered by the user is equal to the clause part of the rule. If yes add a new entry in the ConclusionVariableQueue and the ResultQueue with the value of the conclusion part of the rule (i.e., ConclusionValue); else jump to step 6. Record every event into EventLog.
6. If there are no more variables in the clause part that has the same VariableID in the ConclusionVariableQueue’s first entry, then entry is deleted. If there is more VariableID in the clause part, jump to step 3.
7. If there are no more entries in the ConclusionVariableQueue, the search is finished. Else, jump to step 3.

Figure E.16 Forward chaining process

Backward Chaining

1. Conclusion part identification. The user is asked by the system about the facts of the conclusion part she knows. It will be filled into the ConclusionValue.
2. There are two possibilities:

   2a. A search is conducted to find a variable in the clause part of the rules in the BaseVariableList with the same name as the one input by the user. If a VariableID found, RuleID and NumOfVariableIDs of the variable are saved to the ConclusionStack. If no VariableID found the system tell the user that no result can be found.

   2b. A search is conducted to find a variable in the conclusion part of the rules in the BaseVariableList with the same name as the one in the first entry of the ConclusionStack. If a VariableID found, the RuleID and NumOfVaribleIDs of the variable are saved to the ConclusionStack. If no VariableID found then the entry will be deleted.
3. Each variable in the clause part of the rule must be filled with the value. The values can be taken from the input of the user or from the rules themselves.
4. If a variable is a conclusion variable (i.e., ConclusionValue) then adds a new entry in the ConclusionStack with the RuleID of its variable, then go back to step 3.
5. Next, matching process is conducted to check if the fact input by the user is equal to the clause part of the rule. If equal, add a new entry in the ResultQueue with the value of the conclusion part of the rule, and also delete the entry from the ConclusionStack. If not equal, go back to step 2b. Record every event into EventLog.
6. If the variable of the last result is a conclusion variable (i.e., ConclusionValue) then follow step 4.
7. If there are no more entries in the ConclusionStack, the search is finished. Else, jump to step 2b.

Figure E.17 Backward chaining process

E.3.3 Knowledge Refinement

Every time the user posts her case, the structure and occurrence of its case will be changed in the Variable-Centered Rule Structure.

There are 3 tasks in the Refinement Module: variable analysis, value analysis and rule generation. Each will be described in detail as follows.

E.3.3.1 Variable Analysis

Automatically, every time the user posts her case, the Rule Structure, which uses the Node Structure, maintains the structure of the knowledge base. The occurrence of a node in the rule is obtained from the Rule Structure while the occurrence of a variable is obtained form the Node Structure. Figure E.18 depicts the relation between the shared variables and nodes in the knowledge base.

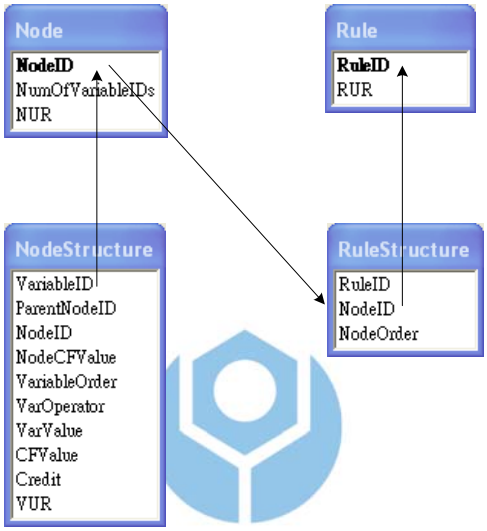


Figure E.18 Shared variable and node relation in the knowledge base

E.3.3.2 Value Analysis

The process of value analysis or usage assignment is to determine the usage degree of rules/nodes/variables in the knowledge base. Table E.1 describes the fields of the tables that are used in the usage assignment.

Table E.1 Fields of table that used in the usage assignment

Tables	Fields
Rule	RuleID, RUR
Node	NodeID, NumOfVariableIDs, NUR
RuleStructure	RuleID, NodeID, NodeOrder
NodeStructure	VariableID, VariableOrder, Credit, VUR

Table E.2 describes the tables and their fields and their relation with the symbols that are used in usage assignment.

Table E.2 Table and its field and symbol in the usage assignment

Table:Field	**Symbols in equations (Ref. Section 3.4.2)**
NodeStructure: VUR NodeStructure: Credit	VUR_i $Credit_i$ Note: $VUR_i = Credit_i \times Weight_i$
NodeStructure: VariableOrder Node: NumOfVariableIDs	VO_i TV Note: $Weight_i = NS_i \times CD_i$ $CD_i = \frac{VO_i}{TV}$ NS_i is obtained by calculate the count of NodeID for the given VariableID
Node: NUR Node: NumOfVariableIDs	NUR_j N Note: $NUR_j = \frac{\sum_{1}^{N} VUR_{ij}}{N}$
Rule: RUR RuleStructure: Number Of NodeIDs of each RuleID	RUR_k N Note: $RUR_k = \frac{\sum_{1}^{N} NUR_{jk}}{N}$

E.3.3.3 Rule Generation

The result of variable and value analysis becomes the source of the system while performing rule generation.

The variable combination produces new nodes, while the node combination produces new rules. These combinations occur as long as the order of variables/nodes being produced doesn't break the existing order of variables/nodes in the knowledge base. The implementation of rule generation is based on the rule generation algorithm of Figure 3.9.

Curriculum Vitae

Name : M.M. Irfan Subakti (司馬伊凡)

Birthday : 9th February, 1974

Birthplace : Magetan, East Java, Indonesia

Education : Master of Science in Engineering (M.Sc.Eng.) from National Taiwan University of Science and Technology (NTUST) – Graduate Program at Department of Computer Science and Information Engineering, Taipei, Taiwan, Republic of China.

Bachelor Science in Computer Engineering (B.Sc.) from Sepuluh Nopember Institute of Technology (ITS) – Undergraduate Program at Department of Informatics, Surabaya, Indonesia.