Rule-based Knowledge Representation Inspired from Finite Automata

Nabil M. Hewahi
Department of Computer Science
Islamic University of Gaza
Palestine

ABSTRACT
In this paper we present a method for rule based knowledge representation. The proposed approach is inspired from Finite Automata (FA). The inspired FA diagrams are easily used to represent rule-based system and make it more reliable structure in large complicated systems. One of the major additions that gives the proposed structure its significance is that it can represent the rules such that else case, the user text input or user menu input, and the relation and sequence of inputs are considered. The proposed diagram is called (RKRFA).

Keywords
Rule-Based Systems, Knowledge Representation, Finite Automata

1. INTRODUCTION
In this section, we shall give a brief definitions about rule structure and Finite Automata (FA). Our main concern is to construct rule based systems using ideas inspired from FA. This means FA is going to be used as knowledge representation. We propose this approach to help designers to simplify and clarify what is required from the software developers/programmer.

1.1 Rule Structure
The standard rule structure is very well known in the case of expert systems. The structure of standard rule is (<IF condition THEN action>). Standard rule structure is one well known method in knowledge representation. Many attempts have been tried to improve the standard rule structure to deal with Variable Precision Logic (VPL) such as Censored Production Rules (CPR) where certainty varies, while specificity stays constant and has the form IF <premise> THEN <decision> UNLESS <censors> [6]. Another form of VPL rule structure called Hierarchical Censored Production Rules (HCPRs). A HCPR is a CPR augmented with specificity and generality information, which can be made to exhibit variable precision in the reasoning such that both certainty of belief in conclusion and its specificity may be controlled by the reasoning process [1]. Hewahi [3] improved HCPR by proposing a rule called General Rule Structure (GRS) to give more flexibility in directing the system where to go if a certain rule fails to fire. The main usage of VPL is real time systems. In [4] Hewahi used Hidden Markov Model as knowledge representation for CPR and presented its usage in network management systems in [5]. The main concern was how to compute the certainty value of the CPRs using the statistical model of HMM. All the presented rule-based knowledge structures do not help much the code implementers to know the importance of the sequence of the condition parts within one rule, and also they do not illustrate whether the inputs are text inputs (entered by the users) or entered through input menu. In addition to that, such knowledge representation structures do not show the relation between one condition with another within the same rule if some input is entered incorrectly. This means what inputs need to be reentered again if a certain input is entered incorrectly. To make some of these problems clear, let us consider the following rule

IF a and b and c and d THEN k

The above rule means if the conditions a, b, c and d are true then we conclude k. Let us assume that a and c are given as default input values and b and d are user inputs. Let us further assume that if the user enters incorrect value for d, then a new value of b must be reentered. The rule itself can't explain this to the code implementer. If we assume that all the inputs a, b, c and d are user inputs, the implementer might think that the user can enter the inputs a,b,c and d in the same sequence shown in the rule, but this might not be true for a certain situation. For example a and c could be in any sequence so a can be entered before or after c but d must be entered after b. Another case is that how to notify for example the implementer that a is user text input while d is a menu user input. All the previous knowledge representation approaches explained above can't solve this problem.

1.2 Finite Automata
Finite automata or called deterministic finite automata is a diagrammatic representation of languages, it consists of the following components [2]:

a. Set of finite number of states where there is only one starting state and one or more final states (if any).
b. The set of the language alphabet Σ .
c. Transitions which are edges that connect one state to another based on a certain alphabet input.

In FA, the outgoing transition from a state must carry one alphabet. The number of outgoing transition is equal to the

number of alphabets in Σ. Because in our proposed approach we might not need to have all the inputs to be exiting from each state, we shall use None deterministic Finite Automata (NFA). NFA is similar to FA but transitions going out from a certain state might not carry all the inputs in Σ. Only the necessary inputs are considered on the transitions going out of the state. To make the NFA clear, let us consider Figure 1 and Figure 2. In Figure 1, the NFA accepts all words that start with either 0 or 1. The shortest accepted word is 01. Words such as 0010101, 101 and 10101 are accepted. In Figure 2, the shortest accepted word is abc. It is clear that all the accepted words should end by c. Words such as abcc, abcabc and abccabc are accepted.

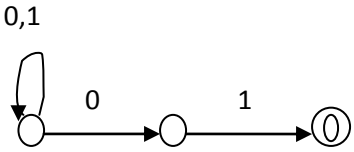

Figure 1. NFA for a language that accepts any word that ends with 01.

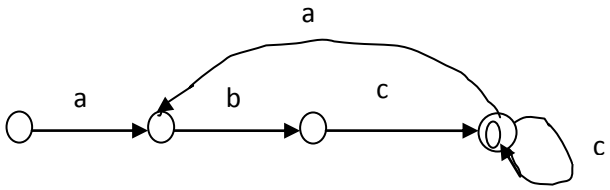

Figure 2. NFA for a language that accepts any ward starts with words end with c.

2. THE PROPOSED KNOWLEDGE REPRESENTATION

Our rule based knowledge representation mechanism is based on the general concepts of NFA but not with the exact meaning as given in NFA. Our proposed diagrammatic rule based representation is called Rule-Based as Knowledge Representation inspired from Finite Automata (RKRFA) and has the following:

a. Σ is the system inputs and sub goals.
b. States are given in three different shapes where each has a specific meaning Δ, O, □, ◊, ■. When Δ is used between two transitions, it means the sequence of the inputs on top of transitions is also valid if reversed. O means that the sequence of inputs on top of the transition is must. □ means this state represents a sub goal. ◊ means "or" relation in a rule having also "and" relation. ■ is used when a sub goal is an input among other inputs considering the sequence of a certain input with the sub goal.
c. We have transitions that transfer the situation from state to another based on the input.

Let us consider the following case
Rule 1: IF a and k THEN z
Rule 2: IF e THEN m
Rule 3: IF n or b THEN f
Based on our proposed scheme the above rules are given in Figure 3.

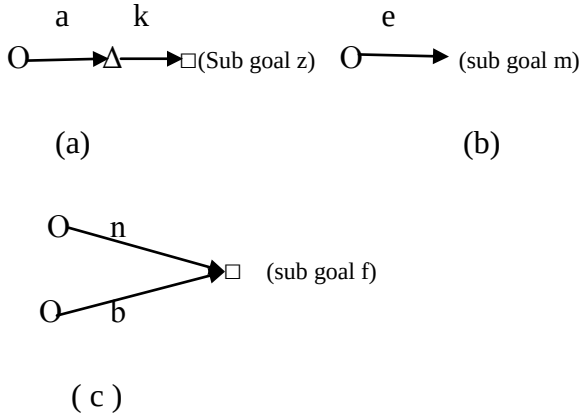

Figure 3. (a) Representation of Rule 1. (b) Representation of Rule 2 (c) Representation of Rule 3.

In Figure 3(a), the Δ is used to inform the developer that the input a is independent of input k and they can be inserted to the system regardless of their order, but still both are necessary to achieve the sub goal z. This implies that IF k and a THEN z is exactly the same in execution as Rule 1. If in Figure 3(a) O is used instead of Δ, this means k can only be inserted after input a. In Figure 3(b), only the input e is required to achieve the sub goal m. in Figure 3(c) f sub goal is only achieved when either n or b is entered or given to the system.

Based on the explanation given above, we consider the following cases:

Case 1: IF a and b THEN k
ELSE IF g and c and d THEN g

This case can be represented as shown in Figure 4. The rule checks if a and b are true irrespective of their input sequence, k is achieved. If any of the inputs is wrong then else rule is considered.

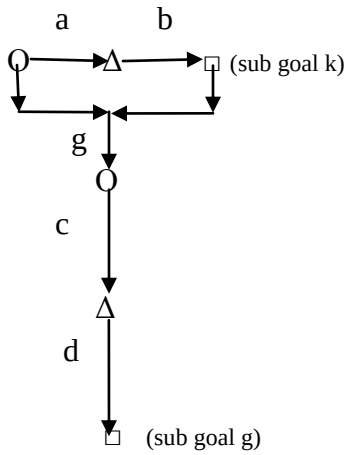

Figure 4. Illustration of RKRFA for case 1.

It is also to be noted from Figure 4 that if a or b is false then we should go directly to the inputs for the ELSE case.

Case 2: If a and b and (c or d) THEN e

This case can be represented as shown in Figure 5.

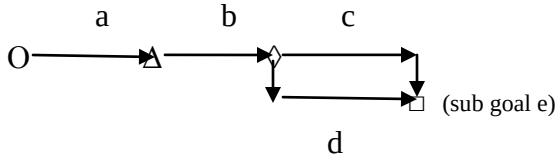

Figure 5. Illustration of RKRFA for case 3.

From Figure 5, ◊ state is used to illustrate that we have "or" relation in the rule. This state also means that the "or" input components can be in any order of inputs with the other inputs as a and b. ◊ state also gives equal opportunity for the inputs of c and d. If the order of the "or" relation component for any reason must be checked before any other input, it can be located at the beginning of RKRFA and then connected with O state.

Case 3: IF a THEN e
IF a THEN x
IF a THEN n

In this case if the input a is given, then we can conclude e, x and n. To depict this case, we need to use the term □ (known as empty string in FA). Figure 6 illustrates this case.

It is clear from Figure 6, that if the input a is given then the system can achieve e, x and n.

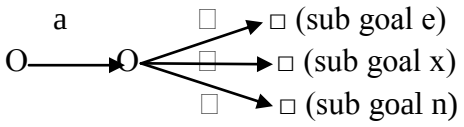

Figure 6. Illustration of RKRFA for case 3.

Case 4: In some cases, to help the system developer, whether the input should be entered by the user or is selected from menu among several choices. In all the previous cases the inputs are assumed to be entered by the user, the question is how to represent the input if it is selected from a given menu. Let us consider the following rule
IF m THEN g
We want to further assume that the value of m should be selected from a given menu. Figure 7 shows this case where the shape of the loop on the top of the state means this value is entered through a menu.

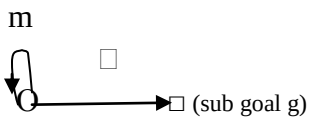

Figure 7. Illustration of Case 4.

We may also have for example a rule as below:
IF a,b THEN F
Let us assume a is input to be provided by the user and b is a menu input, we also assume the order of the inputs is optional. Figure 8 illustrates this case.

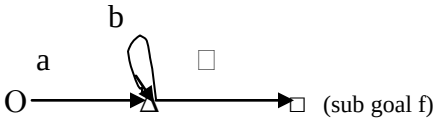

Figure 8. Illustration of the rule IF a,b THEN f.

Case 5: In case an input is given incorrectly, then another previously given input has to be inputted again. Let us consider Figure 9.

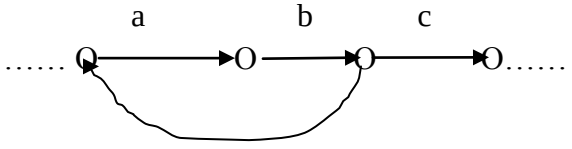

Figure 9. Illustration of Case 5.

In Figure 9, if the input b is wrongly entered, the input a has to be also reentered. But what should be done if there are more than one entered inputs needed to be reentered if a certain input

is incorrectly entered. To illustrate this case, we consider Figure 10. Figure 10 says if c fails, then a and b must be reentered again.

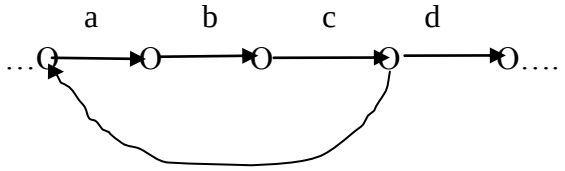

Figure 10. In this case if c is incorrectly entered, then the inputs a and b must be reentered again.

One main question, what should be done if a certain input only needed to be reentered in case of an input is entered incorrectly. To solve this problem a dashed link is used instead of the solid line. Figure 11 and Figure 12 depict this situation. In Figure 11, if c is entered incorrectly, then a must be reentered again before reentering c. The input b is neutral and need not to be reentered. In Figure 12, if d is entered incorrectly, then reenter the inputs a and b before reentering d. c input in this case need not to be reentered again. In Figure 13, if d is reentered incorrectly, then the inputs a and c is reentered before entering d again.

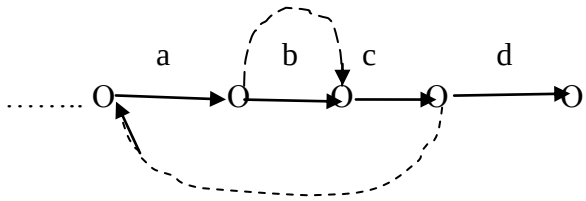

Figure 11. If the input c is entered then a input has to be reentered.

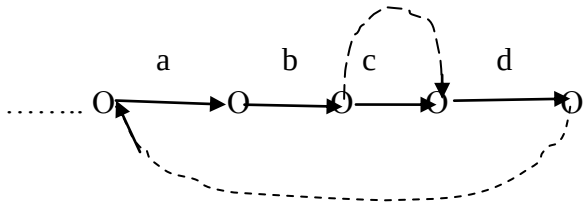

Figure 12. If the input d is entered incorrectly, then a and b inputs have to be reentered.

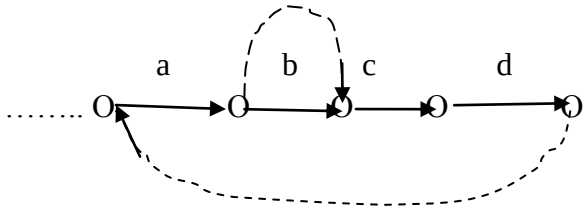

Figure 13. If the input d is entered incorrectly, then a and c inputs have to be reentered.

Case 6: If one of the inputs in the rule is a sub goal achieved from a previous goal, then the state appearance in RKRFA is □. Figure 14 shows this case where assumed g is a sub goal that is achieved by another rule. If we want to omit the order between a and g we use the ■. The ■ is similar as Δ, but the input in this case is a sub goal. This is illustrated in Figure 15.

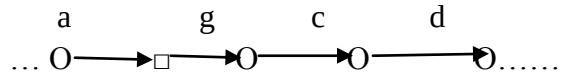

Figure 14. The g input is a sub goal with input sequence restriction.

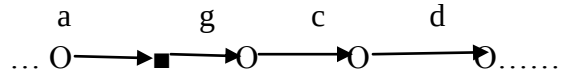

Figure 15. The g input is a sub goal with the optional input sequence between the sub goal g and the input a.

3. Conclusion

In this paper we presented a rule-based knowledge representation inspired from finite automata called RKRFA. The main benefit of RKRFA is that it can help the programmers to write correctly the rules in rule based systems that can be used in developing many systems such as expert systems. The proposed approach can guide the programmers to know how the inputs are sequenced and in which order they must be entered. Also RKRFA helps the programmer to distinguish between the direct user input or an input through menus .The proposed approach is easy to develop and simple to understand, and reduces the gap between the software designer and the programmer.

4. REFERENCES

[1] Bharadwaj K., Jain N., "Hirerichal Censored Production Rules (HCPRs) System", Data and Knowledge Engineering, Vol. 8, pp. 19-34, 1992.

[2] Cohen D., "Introduction to Computer Theory", John Wiley, 1998

[3] Hewahi N., " A General Rule Structure", Information and Software Technology, 44, pp. 451-457, 2002.

[4] Hewahi N., "Hidden Markov Model for Censored Production Rules",ICIT 2009, May 3-5, Amman, Jordan, 2009.

[5] Hewahi N., "An Intelligent Approach For Network Management Based on Hidden Markov Model",ACIT 2010, Dec. 14-16, Libya, 2010.

[6] Michalski R., Winston P., "Variable Precision Logic", Artificial Intelligence 29, pp. 121-145, 1986.