RULE BASE INSPECTION USING ALGORITHMIC APPROACH FOR DATA-ACCESS ORIENTED KNOWLEDGE-BASED SYSTEMS

Fang-Yie Leu, Chien-Chiao Yang

Department of Electronic Engineering
National Taiwan Institute of Technology
Taiwan, R.O.C.

ABSTRACT
This paper describes an effective rule base inspection method that checks rule base consistency and completeness by using algorithmic approach. A knowledge base in this method is represented by one finite state diagram. Through this diagram some errors and anomalies of the rule base can be clearly illustrated and identified. The characteristics of those states representing such errors and anomalies in the state diagram are investigated, and the algorithms to effectively identify those defects are also introduced in this article.

I. Introduction
During the last two decades, artificial intelligence researchers have developed many famous intelligent systems, such as MYCIN, DENDRAL, and HEARSAY. Most of which are rule-based expert systems. So far rule-based technique has been widely adopted by system engineers to develop their intelligent systems. But, nowadays this research is proceeded to a knowledge-based system approach, which separates its domain knowledge from the rest of the system and collects rules for each rule set in it for a specific purpose. However, the process of checking that a knowledge base is correct and complete is one of the central problem of knowledge acquisition. The process involves testing and refining the system knowledge in order to discover and correct a variety of errors and anomalies that can arise during the process of transferring and transforming expertise, especially by knowledge engineers, from human experts or some knowledge sources to a computer system [1].

On the other hand, many books, papers, and reports claim that software inspection actually reduces software development cost, improving product quality, as well as the productivity and manageability of the software development process. Formal Technical Reviews in [2] is a formal inspection process. Fagan proposed another effective inspection process in 1976 that involves the interplay of five elements [3,4]. Nevertheless, these processes and their variations all require several experienced inspectors of similar worlds. Second, many times of inspection meetings are necessary to completely examine the whole rule base. Finally, the code inspection of a rule base is harder than that of an ordinary software program due to rule generalization which always deduces poor readability for the rules.

This paper introduces an effective rule base inspection method that inspects rule bases using algorithmic approach to automatically identify their possible errors and anomalies so that the software development cost, schedule, and quality and productivity can be dramatically saved, lowered, and improved, respectively. However, in order to completely examine a rule, a disjunctive-premises rule, IF A or B THEN C, should be torn into two conjunctive forms, IF A THEN C and IF B THEN C, in an attempt to avoid short circuit evaluation problem [5], i.e. if A is true, then B is not evaluated ( of course, not tested ). We also make an assumption that the assertions in the fact base and those given by the test cases are all error free to simplify the scope of this article. A rule base in this method is represented by a finite state diagram. We call it context transition diagram (CTD). Through the CTD, some defects in the rule base concerned can be clearly illustrated and identified.

II. Context Transition Diagram
A CTD graphically representing all possible inference steps and their results of a rule-based application is a quadruple, \( \{ (Q_i), (Q_f), (P_r), (Q_s) \} \), where \( (Q_i) \) and \( (Q_f) \) are the initial state and the final states of the application, respectively, \( (P_r) \) the rule base being inspected, and \( (Q_s) \) the states in the CTD but not in \( \{ (Q_i), (Q_f) \} \). In such a diagram, a state \( Q \) denoted by a circle is defined as the set \( \{ \text{ASSER} \} \) is a fact in fact base or in working memory at a given instance. An assertion is a dynamic result or static fact [6,7] believed by its users. An assertion base, the pool to retain assertions for a knowledge-based system, is defined as the union of the fact base and the working memory of a knowledge-based system. An arc in turn denoted by an arrow and labeled with a rule stands for that the transition from the state the arc leaves away from (incident from) to the one it points to (incident to) is caused by the rule. When the system is now in some state and some of its outgoing rules are fulfilled,
they become the candidates of the current inference step. Generally, the one selected by scheduler [6,7] will be executed immediately to transit the system from current state to the one the selected arc (rule) points to. In this method, the fulfilled rules are all fired in order to explore the whole possibility of the inference activities. Basically, a forward and a backward chaining systems have similar behavior, so only those about forward chaining is concerned in this article.

A CTD in this method is set up by using breadth-first strategy. The initial state picks up the assertions that are initially gathered in the assertion base including those obtained from test cases, as well as the facts in the fact base of the application concerned. A final state is the state that its deriving rule (or rules) directly accesses the data base. In other words, a data-base-access rule leads to a final state. (Readers may give different reasonable constraints for the final states in order to apply this method to some other application systems). Fig 1 illustrates a part of a CTD.

After the diagram is completely generated, knowledge engineers can invoke the inspection algorithms developed for inspecting and identifying the possible defects so that the debugging task becomes simpler and easier.

III. State and Premise Closures

In this method, closure concepts [8] are used for investigating the transitive connections of the states and assertions in a given CTD so that the inspection algorithms can formally perform their examinations. The forward closure (backward closure) of a state Q, denoted by F(Q) + (B(Q) + ), is defined to be the set of states Q' such that Q -- > Q' (Q' -- > Q ) is followed from the given diagram, where Q -- > Q' (Q' -- > Q ) means there exists a directed path from Q to Q' (Q' to Q ). The premise closure of the fact X in state Q, denoted by Q.X + , is defined as the set of assertions that can be derived from X for the given CTD. Where X must be the premise of an outgoing rule of Q. Of course, corresponding to premise closure, there is a conclusion closure used in backward-chaining. But we don't deal with it here.

IV. The Characteristics of Possible Errors and Anomalies

Consistency and completeness are the two major problems that should be solved [1] for the rule base in a knowledge-based system so that the rules in it can be free from errors and anomalies. Rules may be incorrect, incomplete, inconsistent, or entirely missing. In this article, the first two cases are concluded to the semantics errors of a rule, and the third means some rules are not consistent with another ones. Both of them are regarded as inconsistency problems of a rule-based system, while the fourth is treated as a rule base incompleteness problem.

1. Rule Base Completeness

According to the discussion above, we can conclude that an incomplete rule-based system is a system lacking some rules which should be involved to make the system work properly. Without those rules, a forward reasoning application system starting at the initial state of the given CTD may be unable to achieve its final states -- dead-end rule problem, or can't arrive at some states of the given diagram -- unreachable rule problem.

A. Dead-end Rules

Two sufficient conditions are needed for a dead-end rule: one is that the rule suspends the progress of the inference process after its action part is accomplished; the other is that the result state obtained by applying the rule is not a final state. In a CTD, a non-final state Q with out-deg(Q) = 0 is a dead-end rule because there is no way for the inference engine to move the application away from it. Any rule that makes a system enter a dead-end state is a dead-end rule. Q1, in Fig 2 (a), is an example. Rule1 though rule 3 are such rules. In (b), Q2 is a final state, so the inference engine need not continue its execution. We can then conclude that neither is Q2 a dead-end state, nor are rule 4 through rule 6 dead-end rules. A state Q with out-deg(Q) > 0 and in-deg(Q) > 0 individually is not a dead-end state, but it also has the chance to be. That will be discussed in the paragraph "The Variations of Dead-end Rules".

B. Unreachable Rules

There are also two sufficient conditions to issue a rule that is unreachable: one is that no assertions in the states of the given CTD can match the premise part of the rule; and the other is that the node the arrow, corresponding to the rule, incident from is not the initial state. An unreachable state appearing in a CTD is a non-initial state with in-degree = 0. Since it is not the initial and there is also no way to conduct the system from other states into the one, it is clear that the application can not reach the state forever and its outgoing rules, of course, can never be matched. They are indeed unreachable. Rule 7 through rule 9 in Fig 3 (a) are such examples. But rule 10 through rule 12 in Fig 3 (b) are not, for the inference engine starts its operation from the initial state and these rules each has the opportunity to be applied. They are reachable rules. However, a state Q with out-deg(Q) > 0 and in-deg(Q) > 0 may be unreachable as that of dead-end state. We describe it in the paragraph "The Variations of Unreachable Rules".

C. The Variations of Dead-end Rules

Some variations of dead-end rule are illustrated in Fig 4. In (a), Q5 is a non-final state and it is also an articulation node that disconnects the subdiagrams that were previously connected when it is removed. The subdiagram SD consists of the subdiagrams SD1, SD2 and the arrows laid between them. If the arrows attaching to Q5 but not connecting to SD1 are all incident to Q5 and the final states in SD are entirely in SD2 and the arcs laid between SD1 and SD2 are wholly from SD2 to SD1, then there is no way for the
system to leave Q5 and SD1 and achieve any final states once the system enters to Q5 or any states in SD1. In fact, we can view the union of Q5, SD1 and those arcs laid between them in either directions as a "virtual dead-end state". That is, Q5 and the states Q in SD1 are dead-end states even though out-deg(Q) is larger than zero, and the rules corresponding to the arcs attaching to Q5 no matter those incident to or incident from it, and the rules contained in SD1 and those from SD2 to SD1 are all dead-end rules.

Besides, other variations are illustrated in Fig 4 (b) through (d) as well. In (b), SD degenerates to the subdiagram as the SD1 in (a). That is, the SD2 in (b) is an empty diagram. In (c), the SD degenerates to a linear state chain with chain length = n - 2, which originates and terminates both at Q7. In (d), the SD further shrinks to an arrow, chain length = 1. Furthermore, if the SD is totally an empty diagram and its outlet arcs, of course, doesn't exist any more, the state is indeed a dead-end state as the one in Fig 2 (a).

D. The Variations of Unreachable Rules

Unreachable rules have the similar variations which are shown in Fig 5 (a) through (d). In (a), Q9 is a non-initial articulation node around which the arrows, besides those also attaching to the subdiagram SD, are outlet arcs, and SD = SD1 U SD2 U (the arcs laid between SD1 and SD2). If the states enclosed in SD1 are totally non-initial and the arcs connecting SD1 and SD2 are all incident from the former to the latter, then no matter the initial state is in SD2 or not there is no way for the system to arrive at SD1 and Q9. Here, again, we consider the union of Q9, SD1 and the arcs laid between them in either directions as a "virtual unreachable state". That is, Q9 and the states Q in SD1 are unreachable even if in-deg(Q) is greater than zero. The rules corresponding to the arcs incident to or from Q9, the rules involved in SD1, and those laid between SD1 and SD2 are then unreachable rules. Fig 5 (b) through (d) illustrate the degenerated subdiagrams as those in Fig 4 (b) through (d).

2. Rule Base Consistency

Consistency is the other serious problem that may exist in the rule base of a knowledge-based system. The possible situations are that (1) two rules are identical -- redundant rules; (2) two rules mutually contradict with each other -- conflict rules; (3) a set of rules constitutes a directed cyclic loop that may lead the inference engine to go around it -- circular rules. The details are described below.

A. Redundant Rules

[1] defines redundant rules as two rules that succeed in the same situation and have the same results, see Fig 6. But, we have further discussion in the following. There are four combinations that may occur. Case 1: the two premises and the two actions are respectively identical. They are duplicate rules. Case 2: the premises are identical, while the actions are not. The actions may either identical but with different syntactic forms or something different but producing the same results by some chance. The former is clearly duplicate. The latter denotes that two such rules form a choice set. Case 3: the premises are different, whereas the actions are identical. In fact, they are derived from a disjunction rule, IF (p1 or p2) THEN a1, by splitting its IF part for reducing the rule complexity. Case 4: neither the premises nor the actions are respectively identical. However, if one of the rules in the latter situation of case 2 or in case 3 or 4 does not appear elsewhere in the CTD, that one, of course only one of the two if both are such, can be struck out since the transition caused by the eliminated one can be systematically obtained by applying the remainder.

B. Conflict Rules

[1] defines conflict rules as two rules that succeed in the same situation but with conflicting results. In Fig 7 (a), rule 1 and rule 2 both have the same premises, but the results of the actions, action 1 and ~action 1, contradict with each other. Some mistakes must appear between them.

Besides comparing rules, conflict situations may also implicitly occur among two sets of related rules. If two contradictory facts, say Y in state Q' and ~Y in Q", are derived from the same assertion, say X in Q, then some semantics errors must actually stand along the paths from Q to Q' and from Q to Q" with respect to the rules concerning Q.X*. see Fig 7 (b).

C. Circular Rules

Circular rules are the set of rules that circulate the inference engine in some sequence and along which sequence no final state is involved. Fig 8 illustrates such an example. The cyclic diagram, composed of rule 1 through rule n and their related states, may guide the inference engine to travel around it. There might be syntax or semantic errors among circular rules, knowledge engineers or experts have to inspect them cautiously.

V. The Algorithms

Algorithm 1, for establishing a CTD, first involves the initial state and tries to match all the rules in the rule base being inspected, and inserts all the fulfilled rules R into R_Queue, a FIFO(first-in-first-out) queue for holding the applicable rules under the current state S. After finishing the matching task, it removes these fulfilled rules R1 from R_Queue and applied them one by one in order to generate states S1 and then connects S to S1 by an arrow labelled with R1. If S1 is a new state, algorithm 1 insert it into S_Queue, a FIFO queue for retaining the unvisited states. Next, final-state collection follows. If R1 accesses the application data base, S1 should be a final state. When R_Queue is exhausted, a state S2 is removed from S_Queue and renamed to S, the current state, and the whole process repeats, and so forth until both R_Queue and S_Queue are empty. After that, the backword
generating process starts. The backward generating process tries to obtain new states by matching rules in the rule base concerned with the assertions of each state in the diagram previously generated to collect the candidate rules and the states obtained by applying these rules, and storing them also in R-Queue and S-Queue, respectively. We don't enclose the initial state, because no state locates in front of it.

Algorithm 1 : Generating a CTD
Input:
\{pr\} = \{Rule | Rule is in the rule base being inspected \}
\{Asi\} = \{Fact | Fact is initially in assertion base \}
Output: The CTD = \{\{Qs\}, \{Qf\}, \{Pr\}, \{Qi\}\}
Begin
a). Gather the initial assertions for Qs;
b). S = Qs; /* s: current state */
c). Try to find all the fulfilled rules R in \{pr\} and insert R into R_Queue;
d). While (R_Queue is not empty )
   { Remove a rule R1 from R_Queue;
      Fire R1 under S;
      If (the state S1 generated by applying R1 is a new state)
      { Print(S1);
        Insert S1 to the rear of S_Queue;
        If ( R1 accesses the data base of the application)
        { Qf} = \{Qf\} U S1;
        Connect S to S1 with an arrow from S to S1;
        Labeled the arrow with R1; }
      e). If ( S_Queue is not empty )
      { Remove the front element, state S2, from S_Queue;
        S = S2;
        Goto step c;
      }
   f). Apply the steps that are similar to Step c to e backwards
      in order to explore all the possible states;
g). \{Qi\} = \{Q | Q is a state in CTD, but Q is not in
      ((\{Qf\} U \{Qs\}))
   End.

Algorithm 2 : Dead-end rule detection
Input: The output of Algorithm 1, i.e. a CTD
Output: Dead-end states
Dead-end rules
Begin
a). Gather all the states Q that can be derived backwards
   from \{Qf\}:
   \{Non-dead-end-state\} = \{Non-dead-end-state\} U Q;
b). Find out all the paths PATH from Q to \{Qf\}:
   \{Non-dead-end-rule\} = \{Non-dead-end-rule\} U
   \{R | R is a rule along PATH\};
c). \{Dead-end-state\} = Universal-of-state -
   \{Non-dead-end-state\};
   \{Dead-end-rule\} = Universal-of-rule -
   \{Non-dead-end-rule\};
End.

Algorithm 3 : Unreachable rule detection
Input: The output of Algorithm 1
Output: Unreachable rules
Unreachable rules
Begin
a). Collect all the states \(Q\) that the application can arrive at
   from \(Qs\);
   \{Reachable-state\} = \{Reachable-state\} U Q;
b). Find out all the paths PATH from \(Qs\) to \(Q\):
   \{Reachable-rule\} = \{Reachable-rule\} U
   \{R | R is a rule along PATH\};
c). \{Unreachable-state\} = Universal-of-state -
   \{Reachable-state\};
   \{Unreachable-rule\} = Universal-of-rule -
   \{Reachable-rule\};
End.

The related rules, RULESET'. After that, only unreachable states and rules remain.

In algorithm 4, an object consists of two adjacent states and one of their connecting arcs. This algorithm, for redundant rules detection, checks each object, \(Q_x\) and \(Q_y\) and the arc or one of the arcs from \(Q_x\) to \(Q_y\), in the given CTD by temporarily removing the arc of the object from the CTD and then deriving \(F(Q_x)^+\) to see if \(Q_y\) is in \(F(Q_x)^-\). If it is true, the redundant path or paths are searched by depth-first strategy.

Algorithm 5, for conflict rules inspection, derives each premise closure \(Q\cdot X^-\) for the outgoing rules of each state \(Q\) in the given CTD to see if the new-obtained closure element \(Y\) in state \(Q'\) is already in \(Q\cdot X^-\) or not. If \(Q'\cdot Y\), for any state \(Q\) in the CTD, is not in \(Q\cdot X^-\), \(Q'\cdot Y\) must be the first occurrence of \(Y\) derived from current \(Q\cdot X^-\). We join it to \(Q\cdot X^-\) as a new element and continue the derivations until a final state is achieved or a contradiction is discovered. If \(Q'\cdot \neg Y\), for some state \(Q'\) in the CTD, is in \(Q\cdot X^-\), it backtraces the deriving paths, which are marked with special symbols during the inference process, from \(Q'\) to \(Q\) and from \(Q'\) to \(Q\) telling users the contradiction.
Algorithm 4: Redundant rules detection
Input: The output of Algorithm 1
Output: Redundant rules
Begin
a). For (Every object, say Qx and Qy and an arc ARC from Qx to Qy, in the given CTD)
   {Temporarily delete ARC from CTD;
    Derive forward closure of Qx, F(Qx)*;
    If ( Qy in F(Qx)*)
    While ( there exists at least a directed path PATH, from Qx to Qy yet to be visited)
    Redundant-rules = Redundant-rules U
     {PATH, ARC, (Qx,Qy)};
    Restore ARC back to the CTD; }
End.

Algorithm 5: Conflict rules inspection
Input: The output of Algorithm 1
Output: Conflict rules
Begin
a). For (Each state, Q in the given CTD)
   For ( Every premise X of an out-going rule of Q )
      {Q.X+ = {Q.X};
       An assertion Y in state Q’ is derived from Q.X+;
       Mark the arc that derive Y with an **;
       If (Q'.Y, for some state Q’ in the given CTD, is already in Q.X+ )
       {Backtrace the paths PATH1 and PATH2, from Q’ to Q
        and from Q' to Q, respectively;
        Conflict-Rules = Conflict-Rules U
         {((Q.X, Q’.Y, PATH1), (Q.X, Q’.Y, PATH2))}; }
      Else
       If (Q'.Y is not in Q.X+, where Q’ is any state except Q’)
       Q.X+ = Q.X+ U {Q’.Y}; }
End.

Algorithm 6: Circular rules detection
Input: The output of Algorithm 1
Output: Circular rules
Begin
a). Remove all final states and their connection arcs;
b). Remove all states that have only in-going or out-going arcs and their connecting arcs;
c). For (Every object, Qx and Qy and an arc ARC from Qx to Qy, in the given CTD)
   {Temporarily delete ARC from the CTD;
    Derive forward closure of Qy, F(Qy)*;
    If ( Qx is in F(Qy)*)
    While (There exists another directed path, PATH, from Qy to Qx yet to be visited)
    Circular-rules = Circular-rules U
     {((PATH, ARC), (Qx, Qy)};
    Restore back ARC to the CTD; }
End.

VI. Conclusion
In this paper, we have described: how to construct a CTD; the semantics of a CTD; the concepts of state closure and premise closure; the characteristics of the errors and the potential anomalies existing in a CTD; and the algorithms for inspecting them.

References
Fig 1: A part of a context transition diagram

Fig 2: A dead-end state and a non-dead-end state

Fig 3: An unreachable state and a reachable state

Fig 4: The variations of dead-end state

Fig 5: The variations of unreachable state

Fig 6: Redundant rules

Fig 7: Conflict rules

Fig 8: Circular rules