CHECKING AN EXPERT SYSTEMS KNOWLEDGE BASE FOR CONSISTENCY AND COMPLETENESS

T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Peoora

Lookheed Research and Development
0/92-10 B/254E
3251 Hanover Street, Palo Alto, CA. 94304

ABSTRACT

In this paper we desoribe a program that verifies the oonsistenoy and completeness of expert system knowledge bases whioh utilize the Lookheed Expert System (LES) framework. The algorithms desoribed here are not specific to LES and oan be applied to most rule-based systems. The program, oalled CHECK, combines logioal principles as well as speoifio information about the knowledge representation formalism of LES. The program oheoks for redundant rules, conflictlng rules, subsumed rules, missing rules, ciroular rules, unreachable olauses, and deadend olauses. It also generates a dependency ohart whioh shows the dependencies among the rules and between the rules and the goals. CHECK oan help the knowledge engineer to deteot many programming errors even before the knowledge base testing phase. It also helps deteot gaps in the knowledge base whioh the knowledge engineer and the expert might have overlooked. A wide variety of knowledge bases have been analyzed using CHECK.

1.0 INTRODUCTION

The Lookheed Expert System (LES) is a generio rule-based expert system tool [1] (similar to EMYCIN [2]) that has been used as a framework to construct expert systems in many areas suoh as electronic equipment diagnosis, design oheoking, photo Interpretation, and hazard analysis. LES employs a combination of goal-driven and data-driven rules with the latter being attaohed to the faotual database (demons). One objective in the design of LES was to make it easy to use. Thus, many debugging tools and aids were added to the LES program. One of these aids is the knowledge base completeness and oonsistenoy verification program oalled CHECK. Its purpose is to help a knowledge engineer oheok the knowledge base whioh he oonstruoted for logioally redundant rules, conflicting rules, subsumed rules, missing rules, unreachable olauses, and deadend olauses. CHECK does not perform any syntax oheoking on the rules, sinoe this is done automatically when the rule files are loaded into the knowledge base. CHECK statioally analyzes the knowledge base (i.e., after the rules and facts are loaded into the knowledge base), unlike TEIRESIAS, whioh performs an assessment of rules in the setting of a problem-solving session [31.

Suwa, Soott, and Shortliffe [4] have written a program for verifying that a set of rules comprehensively spans the knowledge base. This program was devised and tested within the oontext of the ONCOCIN system (an EMYCIN-like system). Our work differs from theirs in that CHECK inoludes unreachable olauses and deadend olauses as two additional rule oheoking oriteria. Furthermore, CHECK produoes a dependency ohart and deteots any oiroular rule ohains. Also, CHECK was devised and tested on a generio expert system with oase-grammar rules and a "frame" database. It has been used to analyze a wide variety of knowledge bases. CHECK does not take into aooount oertainty factcrs when oheoking the rule base.

2.0 CHECKING FOR CONSISTENCY AND COMPLETENESS

A statio analysis of the rules oan deteot many potential problems and gaps that exist in a rule base. We now identify and give definitions for seven oriteria that are used by CHECK to perform statio analysis of any rule base oonstruoted for use with LES. The first four oriteria are concerned with potential problems, whereas the last three oriteria are oonoerned with gaps in the knowledge base.

2.1 POTENTIAL PROBLEMS IN A KNOWLEDGE BASE

By statioally analyzing the logioal semantios of the rules represented in LES's case grammar format, CHECK oan deteot redundant rules, oonflioting rules, rules that are subsumed by other rules, and circular-rule ohains. The following definitions for these four potential problems are used in CHECK:

- Redundant rules: two rules succeed in the same situation and have the same results. In LES, this means that the IF parts of the two rules are

equivalent, and one or more THEN olauses are also equivalent. Because LES allows variables in rules, equivalent means that the same specific objeot names can match their corresponding variables. For example the rule "p(x) —> q(x)" is equivalent to the rule "p(y) —> q(y)", where x and y are variables.

- o Conflicting rules: two rules succeed in the same situation but with conflicting results. In LES, this means that the IF parts of the two rules are equivalent, but one or more THEN olauses are oontradiotory, or one pair of IF olauses is oontradiotory while they have equivalent THEN olauses. For example, the rule "p(x) —> not(q(x))" is oontradiotory to the rule "p(x) —> q(x)".

- o Subsumed rules: two rules have the same results, but one oontains additional constraints on the situations in whioh it will suooeed. In LES, this means one or more THEN olauses are equivalent, but the IF part of one rule oontains fewer oonstraints and/or clauses than the IF part of the other rule. For example, the rule "(p(x) and q(y)) —> r(z)" is subsumed by the rule "p(x) —> r(z)".

- o Circular rules: a set of rules is a circular-rule set if the ohaining of those rules in the set forms a cycle. For example, if we had a set of rules as follows: (1) "p(x) —> q(x)" (2) "q(x) —> r(x)" (3) "r(x) —> p(x)" and the goal is r(A), where A is a constant, then the system will enter an infinite loop at run time, unless the system has a speoial way of handling circular rules.

2.2 POTENTIAL GAPS IN A KNOWLEDGE BASE

The development of a knowledge-based system is an iterative prooess in whioh knowledge is enooded, tested, added, ohanged, and refined. This iterative prooess often leaves gaps in the knowledge base whioh both the knowledge engineer and the expert may have overlooked during the knowledge aoquisition prooess. In LES, we have found three situations indicative of gaps in the knowledge base. These three situations, called CD missing rules, (2) unreachable olauses, and (3) deadend olauses are desoribed below:

- o Missing rules: a situation in whioh some values in the set of possible values (called legal values) of an objeot's attribute are not oovered by any rule's IF olauses (i.e., the legal values in the set are oovered only partially or not at all). A partially oovered attribute oan prohibit the system from attaining a oonolusion or oause it to make a wrong oonolusion when an unoovered attribute value is enoountered during run time.

- o Unreachable clauses: in a goal-driven produotion system, a THEN olause of a rule should either matoh a goal olause or matoh an IF olause of another rule (in the same rule set). Otherwise, the THEN olause is unreachable.

- o Deadend Glauses: to aohieve a goal (or subgoal) in LES, it is required that either: (1) the attributes of the goal olause are askable (user provides needed information) or (2) that the goal olause is matched by a THEN olause of one of the rules in the rule sets applying to that goal. If neither of these oonditions is satisfied then the goal olause oan not be achieved, i.e., it is a "deadend olause". Similarly, the IF olauses of a rule also must meet one of these two oonditions, or they are "deadend olauses".

2.3 DEPENDENCY CHART AND CIRCULAR-RULE CHAINS DETECTION

As a by-produot of the rule oheoking, CHECK generates a dependency ohart whioh shows the interactions among the rules and between the rules and the goal olauses. An example of a dependency ohart for a small problem is shown in Figure 1. A "*" indioates that one or more olauses in the IF part of a rule or a goal olause (G.C.) matches one or more olauses in the THEN part of a rule. The dependenoy ohart is very useful when the knowledge engineer deletes, modifies, or adds rules to the rule base.

Note that in Figure 1, the "*" 's indicate the dependencies for the original rule set. By adding a olause to Rule 2, the "*2" dependencles appeared. Note, Rule 2 now references itself—a self-oiroular rule. By the addition of one olause to Rule 1, the "*1" dependenoles appeared. This also oauses the rule set to be circular, sinoe an IF olause of Rule 1 is matohed by THEN olauses of Rule 7 and Rule 8 whioh in turn matoh an IF olause of Rule 1. Circular rules should be avoided since they oan lead to an infinite loop at run time. Some expert systems, suoh as EKYCIN, handle oiroular rules in a speoial way. Nevertheless, the knowledge engineer will want to know whioh rules are oiroular. So, CHECK uses the dependenoy ohart to generate graphs representing the interaotions between rules, and uses a cyclio graph detection algorithm to deteot oiroular rule ohains.

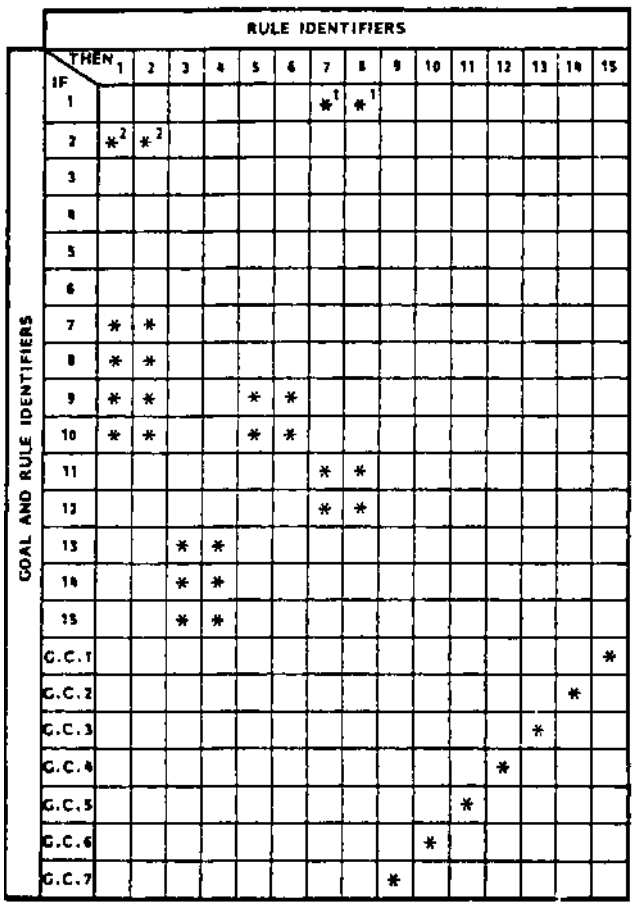

Figure 1

3.0 IMPLEMENTATION OF RULE CHECKER

In solving a problem, the knowledge engineer may write several sets of goal-driven rules with eaoh set having a unique subjeot category. (In LES it is oonvenient to put rules in different subjeot categories so that the system oan solve different goals using only those rule sets which apply to that goal.; To solve a partioular goal, often he will seleot several goal-driven rule sets and WHEN rules (demons). Sinoe these rule sets are generated over a period of time, it is quite possible that their interaction will oause some problems. Thus, for eaoh goal it is neoessary to compare the rules (in the rule sets speoified by that goal) against eaoh other and against the olauses of that goal. We now show an algorithm (in an Algol-like notation) whioh CHECK uses. The algorithm does the checking for a set of subjeot oategories with N rules and a goal with G olauses.

Beoause an IF part or a THEN part oan have more than one olause, the comparison between one part and another is handled by comparing a olause of one part to every olause in the other part.

These algorithms also work for forward-ohalning rules, whioh are oalled WHEN rules in LES. However, the criterion unreachable clauses is not applicable to forward-chainlng rules.

```
procedure Analyze_KB(Rules,Goal,N,G);
begin
   for i = 1 to N begin
      /* compare rules against each other */
      for j = 1 to N begin
         for k = first_clause(i) to last_clause(i) begin
            for n = first_clause(j) to last_clause(j) begin
               match_result(k) = Compare_clauses(k,n);
            end; /* l */
         end;  /* k */
      end;  /* j */
      /* compare goal clauses against rule clauses */
      for g = 1 to G begin
         for k = first_clause(i) to last_clause(i) begin
            match_result(g) = Compare_clauses(g,k);
         end;
      end;
      /* collect information on attributes coverage */
      for p = first_lf_clause(l) to last_lf_clause(l) begin
         determine attribute referred by clause p;
         store the attribute's value covered by clause p;
      end;
   end;  /* l */
   /* check for possible problems in rules */
   for l = 1 to N begin
      for k = first_clause(l) to last_clause(i) begin
         matched_rule = Transform(l,k,clause_relations);
         while (matched_rule <> 0) do begin
            Check_problems(l,matched_rule,clause_relations);
            matched_rule = Transform(l,k,clause_relations);
         end;
      end;
    end;
    /* check for possible gaps in rules and goals */
    Check_gaps(goal_clauses,rules);
    /* generate the dependency chart */
    Generate_dependency_chart(goal_clauses,rules);
    /* check for possible missing rules */
    for m = first_category to last_category begin
       for n = first_attribute(m) to last_attribute(m) begin
          compare covered values with legal values;
          if not_completely_covered then
             inform user that some rule is missing;
          if illegal_attribute_value then
             inform user that attribute value is illegal;
       end;  /* for n */
    end;   /* for m */
end;  /* Analyze_KB */
```

```
procedure Check_gaps(goal_clauses,rules);
begin
   for l = first_then_clause to last_then_clause begin
      if (l did not match any IF clause or GOAL clause) then
         then_result(l) = UNREACHABLE;
   end;
   for l = first_goal_clause to last_goal_clause begin
      if (l not_match any THEN clause & not askable(l)) then
          goal_result(l) = DEADEND;
   end;
   for l = first_lf_clause to last_lf_clause begin
      if (l not_match any THEN clause & not askable(l)) then
         lf_result(l) = DEADEND;
   end;
end;  /* Check_gaps */
```

```
procedure Check_problems(i,m,clause_relations);
begin
   conflict_count = 0;
   subset = FALSE;
   superset = TRUE;
   for c = first_if_clause(i) to last_if_clause(i) begin
      if clause_relations(c) = SUBSET then
         subset = TRUE;
      else if clause_relations(c) = SUPERSET then
         superset = TRUE;
      else if clause_relations(c) = CONFLICT then
         conflict_count = conflict_count + 1;
      else
         if_if = SAME;   /* composite result may be SAME */
   end;

   if (conflict_count > 1) then
      if_if = DIFFERENT;
   else if (conflict_count=1 & not (subset or superset)) then
      if_if = CONFLICT;
   else begin /* conflict_count = 0 => no conflict */
      if (subset & superset) then
         if_if = DIFFERENT;
      else if subset then
         if_if = SUBSET;
      else if superset then
         if_if = SUPERSET;
      else;
   end;

   if ( (if_if = SAME) &
      number_of_if_clauses(i) = number_of_if_clauses(m)) then
      for t=first_then_clause(i) to last_then_clause(i) begin
         if (clause_relations(t) = SAME) then
            result(i,m,t) = REDUNDANT;
         else if (clause_relation(t) = CONFLICT) then
            result(i,m,t) = CONFLICT;
         else;
      end;
   else if (if_if = CONFLICT) then
      for t=first_then_clause(i) to last_then_clause(i) begin
         if (clause_relations(t) = SAME) then
            result(i,m,t) = CONFLICT;
      end;
   else if ( (if_if = SAME) &
      number_of_if_clauses(i) < number_of_if_clauses(m)) then
      for t=first_then_clause(i) to last_then_clause(i) begin
         if (clause_relations(t) = SAME) then
            result(i,m,t) = SUBSET;
      end;
   else if ( (if_if = SUBSET) &
      number_of_if_clauses(i) >= number_of_if_clauses(m)) then
      for t=first_then_clause(i) to last_then_clause(i) begin
         if ( (clause_relations(t) = SAME) or
              (clause_relations(t) = SUBSET) ) then
            result(i,m,t) = SUBSET;
      end;

   else if ( (if_if = SUPERSET) &
      number_of_if_clauses(i) = number_of_if_clauses(m)) then
      for t=first_then_clause(i) to last_then_clause(i) begin
         if ( (clause_relations(t) = SAME) or
              (clause_relations(t) = SUPERSET) ) then
            result(i,m,t) = SUPERSET;
      end;
   else;
end;  /* Check_problems */
```

4.0 gnyMAttv

In this paper we desoribed a program called CHECK whose function is to deteot four potential problems (redundant rules, confllоting rules, subsumed rules, and oiroular rules) and three potential gaps (missing rules, unreaohable clauses, and deadend olauses) in a knowledge base utilizing the LES framework. We applied the oonsistenoy and completeness verifioation method of Suwa, Soott, and Shortliffe [4] to the generio expert system LES with good results. Furthermore, we have extended the checkIng to inolude oiroular rules, unreaohable olauses, and deadend olauses. We also showed a general algorithm whioh performs the oheoking funotion efficiently. Finally, as a by-produot of the rule oheoking processing, CHECK generates a dependenoy ohart whioh shows how the rules oouple and interaot with eaoh other and with the goals; this ohart should help the knowledge engineer to identify immediately the effeots of deleting, adding, or modifying rules.

From our experiences with oonetruoting different knowledge bases, we find that many ohanges and additions to the rule sets ooour during the development of a knowledge base. Thus, a tool suoh as CHECK that oan deteot many potential problems and gaps in the knowledge base should be very useful to the knowledge engineer in helping him to develop a knowledge base rapidly and accurately.

The major area of improvement for CHECK is the handling of oertainty faotors in the rules sinoe LES allows the rules to have oertainty faotors associated with them; this may require the definitions for the seven conditions oovered in this paper to be revised.

REFERENCES

[1] W. A. Perkins, and T. J. Laffey. "LES: A General Expert System and Its Applications", Proc SPIE's Technical Symposium East, Applications of Artificial Intelligence, Arlington, VA., May 3-4, 1984, pp. 46-57.

[2] w. J. van Melle, System Aids in Constructing Consultation Programs. UMI Researoh Press, Ann Arbor, MI (1981).

[3] R. Davis, "Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases", Doctcral dissertation, Computer Soienoe Department, Stanford University, 1976.

[4] M. Suwa, A. C. Soott, and E. H. Shortliffe, "An Approaoh to Verifying Completeness and Consistenoy in a Rule-Based Expert System", The AI Magazine. Fall 1982, pp. 16-21.