8

Completeness and Consistency in a Rule-Based System

Motoi Suwa, A. Carlisle Scott, and Edward H. Shortliffe

The builders of a knowledge-based expert system must ensure that the system will give its users accurate advice or correct solutions to their problems. The process of verifying that a system is accurate and reliable has two distinct components: checking that the knowledge base is correct, and verifying that the program can interpret and apply this information correctly. The first of these components has been the focus of the research described in this chapter; the second is discussed in Part Ten (Chapters 30 and 31).

Knowledge base debugging, the process of checking that a knowledge base is correct and complete, is one component of the larger problem of knowledge acquisition. This process involves testing and refining the system's knowledge in order to discover and correct a variety of errors that can arise during the process of transferring expertise from a human expert to a computer system. In this chapter, we discuss some common problems in knowledge acquisition and debugging and describe an automated assistant for checking the completeness and consistency of the knowledge base in the ONCOCIN system (discussed in Chapters 32 and 35).

As discussed in Chapters 7 and 9, an expert's knowledge must undergo a number of transformations before it can be used by a computer. First, the person acquires expertise in some domain through study, research, and experience. Next, the expert attempts to formalize this expertise and to express it in the internal representation of an expert system. Finally, the

This chapter is based on an article originally appearing in *The AI Magazine* 3: 16–21 (Autumn 1982).

knowledge, in a machine-readable form, is added to the computer system's knowledge base. Problems can arise at any stage in this process: the expert's knowledge may be incomplete, inconsistent, or even partly erroneous. Alternatively, while the expert's knowledge may be accurate and complete, it may not be adequately transferred to the computer-based representation. The latter problem typically occurs when an expert who does not understand computers works with a knowledge engineer who is unfamiliar with the problem domain; misunderstandings that arise are often unrecognized until performance errors occur. Finally, mistakes in spelling or syntax (made when the knowledge base is entered into the computer) are frequent sources of errors.

The knowledge base is generally constructed through collaboration between experts in the problem domain and knowledge engineers. This difficult and time-consuming task can be facilitated by a program that:

1. checks for inconsistencies and gaps in the knowledge base,
2. helps the experts and knowledge engineers communicate with each other, and
3. provides a clear and understandable display of the knowledge as the system will use it.

In the remainder of this chapter we discuss an experimental program with these capabilities.

8.1 Earlier Work

One goal of the TEIRESIAS program, described in the next chapter, was to provide aids for knowledge base debugging. TEIRESIAS allows an expert to judge whether or not MYCIN's diagnosis is correct, to track down the errors in the knowledge base that led to incorrect conclusions, and to alter, delete, or add rules in order to fix these errors. TEIRESIAS makes no formal assessment of rules at the time they are initially entered into the knowledge base.

In the EMYCIN system for building knowledge-based consultants (Chapter 15), the knowledge acquisition program fixes spelling errors, checks that rules are semantically and syntactically correct, and points out potentially erroneous interactions among rules. In addition, EMYCIN's knowledge base debugging facility includes the following options:

1. a trace of the system's reasoning process during a consultation, available to knowledge engineers familiar with the program's internal representation and control processes;

2. an interactive mechanism for reviewing and correcting the system's conclusions (a generalization of the TEIRESIAS program);
3. an interface to the system's explanation facility to produce automatically, at the end of a consultation, explanations of how the system reached its results; and
4. a verification mechanism, which compares the system's results at the end of a consultation with the stored "correct" results for the case that were saved from a previous interaction with the TEIRESIAS-like option. The comparison includes explanations of why the system made its incorrect conclusions and why it did not make the correct ones.

8.2 Systematic Checking of a Knowledge Base

The knowledge base debugging tools mentioned above allow a system builder to identify problems with the system's knowledge base by observing errors in its performance on test cases. While thorough testing is an essential part of verifying the consistency and completeness of a knowledge base, it is rarely possible to guarantee that a knowledge base is completely debugged, even after hundreds of test runs on sample test cases. TEIRESIAS was designed to aid in debugging an extensive rule set in a fully functional system. EMYCIN was designed to allow incremental building of a knowledge base and running consultations with only a skeletal knowledge base. However, EMYCIN assumes that the task of building a system is simply to encode and add the knowledge.

In contrast, building a new expert system typically starts with the selection of knowledge representation formalisms and the design of a program to use the knowledge. Only when this has been done is it possible to encode the knowledge and write the program. The system may not be ready to run tests, even on simple cases, until much of the knowledge base is encoded. Regardless of how an expert system is developed, its developers can profit from a systematic check on the knowledge base without gathering extensive data for test runs, even before the full reasoning mechanism is functioning. This can be accomplished by a program that checks a knowledge base for completeness and consistency during the system's development.

8.2.1 Logical Checks for Consistency

When knowledge is represented in production rules, inconsistencies in the knowledge base appear as:

- *Conflict:* two rules succeed in the same situation but with conflicting results.
- *Redundancy:* two rules succeed in the same situation and have the same results.
- *Subsumption:* two rules have the same results, but one contains additional restrictions on the situations in which it will succeed. Whenever the more restrictive rule succeeds, the less restrictive rule also succeeds, resulting in redundancy.

Conflict, redundancy, and subsumption are defined above as logical conditions. These conditions can be detected if the syntax allows one to examine two rules and determine if situations exist in which both can succeed and whether the results of applying the two rules are identical, conflicting, or unrelated.

8.2.2 Logical Checks for Completeness

Incompleteness of the knowledge base is the result of:

- *Missing rules:* a situation exists in which a particular inference is required, but there is no rule that succeeds in that situation and produces the desired conclusion.

Missing rules can be detected logically if it is possible to enumerate all circumstances in which a given decision should be made or a given action should be taken.

8.2.3 Pragmatic Considerations

It is often pragmatic conditions, not purely logical ones, that determine whether or not there are inconsistencies in a knowledge base. The semantics of the domain may modify syntactic analysis. Of the three types of inconsistency described above, only conflict is guaranteed to be a true error.

In practice, logical redundancy may not cause problems. In a system where the first successful rule is the only one to succeed, a problem will arise only if one of two redundant rules is revised or deleted while the other is left unchanged. On the other hand, in a system using a scoring mechanism, such as the certainty factors in EMYCIN systems, redundant rules cause the same evidence to be counted twice, leading to erroneous increases in the weight of their conclusions.

In a set of rules that accumulate evidence for a particular hypothesis, one rule that subsumes another may cause an error by causing the same evidence to be counted twice. Alternatively, the expert might have pur-

posely written the rules so that the more restrictive one adds a little more weight to the conclusion made by the less restrictive one.

An exhaustive syntactic approach for identifying missing rules would assume that there should be a rule that applies in each situation defined by all possible combinations of domain variables. Some of these combinations, however, are not meaningful. For example, there are no males who are pregnant (by definition) and no infants who are alcoholics (by reason of circumstances). Like checking for consistency, checking for completeness generally requires some knowledge of the problem domain.

Because of these pragmatic considerations, an automated rule checker should display potential errors and allow an expert to indicate which ones represent real problems. It should prompt the expert for domain-specific information to explain why apparent errors are, in fact, acceptable. This information should be represented so that it can be used to make future checking more accurate.

8.3 Rule Checking in ONCOCIN

8.3.1 Brief Description of ONCOCIN

ONCOCIN (see Chapter 35) is a rule-based consultation system to advise physicians at the Stanford Medical Center cancer clinic on the management of patients who are on experimental treatment protocols. These protocols serve to ensure that data from patients on various treatment regimens can be compared in order to evaluate the success of therapy and to assess the relative effectiveness of alternative regimens. A protocol specifies when the patient should visit the clinic, what chemotherapy and/or radiation therapy the patient should receive on each visit, when laboratory tests should be performed, and under what circumstances and in what ways the recommended course of therapy should be modified.

As in MYCIN, a rule in ONCOCIN has an *action* part that concludes a *value* for some *parameter* on the basis of values of other parameters in the rule's *condition* part. Currently, however, all parameter values can be determined with certainty; there is no need to use weighted belief measures. When a rule succeeds, its action parameter becomes *known* so no other rules with the same action parameter will be tried.

In contrast to MYCIN, rules in ONCOCIN specify the *context* in which they apply. Examples of ONCOCIN contexts are drugs, chemotherapies (i.e., drug combinations), and protocols. A rule that determines the dose of a drug may be specific to the drug alone or to both the drug and the chemotherapy. In the latter case, the context of the rule would be the list of pairs of drug and chemotherapy for which the rule is valid. At any time

during a consultation, the *current context* represents the particular drug, chemotherapy, and protocol currently under consideration.

In order to determine the value of a parameter, the system tries rules that conclude about that parameter and that apply in the current context. For example, Rule 75 shown below is invoked to determine the value of the parameter current attenuated dose. The condition will be checked only when the current context is a drug in the chemotherapy MOPP or a drug in the chemotherapy PAVE. Clause 1 of the condition gives a reason to attenuate (lessen) the doses of drugs, and clause 2 mentions a reason not to attenuate more than 75%.

RULE 75

[*action parameter*]	(a)	To determine the current attenuated dose	
[*context*]	(b)	for all drugs in MOPP, or for all drugs in PAVE:	
[*condition*]	IF:	1)	This is the start of the first cycle after a cycle was aborted, and
		2)	The blood counts do not warrant dose attenuation
[*action*]	THEN:		Conclude that the current attenuated dose is 75 percent of the previous dose

Certain rules for determining the value of a parameter serve special functions. Some give a "definitional" value in the specified context. These are called *initial rules* and are tried first. Other rules provide a (possibly context-dependent) "default" or "usual" value in the event that no other rule succeeds. These are called *default rules* and are applied last. Rules that do not serve either of these special functions are called *normal rules.* Concluding a parameter's value consists of trying, in order, three groups of rules: initial, normal, then default. A rule's *classification* tells which of these three groups it belongs to.[1]

[1]Internally in LISP, the context, condition, action, and classification are properties of an atom naming the rule. The internal form of Rule 75 is

RULE075

```
CONTEXT:          ((MOPP DRUG)(PAVE DRUG))
CONDITION:        (AND ($IS POST.ABORT 1)
                       ($IS NORMALCOUNTS YES))
ACTION:           (CONCLUDEVALUE ATTENDOSE (PERCENTOF 75 PREVIOUSDOSE))
CLASSIFICATION:   NORMAL
```

As in MYCIN, the LISP functions that are used in conditions or actions in ONCOCIN have *templates* indicating what role their arguments play. For example, both $IS and CONCLUDEVALUE take a parameter as their first argument and a value of that parameter as their second argument. Each function also has a *descriptor* representing its meaning. For example, the descriptor of $IS shows that the function will succeed when the parameter value of its first argument is equal to its second argument.

8.3.2 Overview of the Rule-Checking Program

A rule's context and condition together describe the situations in which it applies. The templates and descriptors of rule functions make it possible to determine the combination of values of condition parameters that will cause a rule to succeed. The rule's context property shows the context(s) in which the rule applies. The contexts and conditions of two rules can therefore be examined to determine if there are situations in which both can succeed. If so, and if the rules conclude different values for the same parameter, they are in conflict. If they conclude the same thing, except that one contains extra condition clauses, then one subsumes the other.

These definitions of inconsistencies simplify the task of checking the knowledge base. The rules can be partitioned into disjoint sets, each of which concludes about the same parameter in the same context. The resulting rule sets can be checked independently. To check a set of rules, the program:

1. finds all parameters used in the conditions of these rules;
2. makes a table, displaying all possible combinations of condition parameter values and the corresponding values that will be concluded for the action parameters (see Figure 8-1);[2] and
3. checks the tables for conflict, redundancy, subsumption, and missing rules; then displays the table with a summary of any potential errors that were found. The rule checker assumes that there should be a rule for each possible combination of values of condition parameters; it hypothesizes missing rules on this assumption (see Figure 8-2).[3]

ONCOCIN's rule checker *dynamically* examines a rule set to determine which condition parameters are currently used to conclude a given action parameter. These parameters determine what columns should appear in the table for the rule set. The program does *not* expect that each of the parameters should be used in every rule in the set (as illustrated by Rule 76 in the example of the next subsection). In contrast, TEIRESIAS (see next chapter) examined the "nearly complete" MYCIN knowledge base and built *static* rule models showing (among other things) which condition parameters were used (in the existing knowledge base) to conclude a given action parameter. When a new rule was added to MYCIN, it was compared

[2]Because a parameter's value is always known with certainty and the possible values are mutually exclusive, the different combinations of condition parameter values are disjoint. If a rule corresponding to one combination succeeds, rules corresponding to other combinations in the same table will fail. This would not be true in an EMYCIN consultation system in which the values of some parameters can be concluded with less than complete certainty. In such cases, the combinations in a given table would not necessarily be disjoint.

[3]We plan to add a mechanism to acquire information about the meanings of parameters and the relationships among them and to use this information to omit semantically impossible combinations from subsequent tables.

Rule set: 667 600 82 80 69 67 76

Context: the drug cytoxan in the chemotherapy CVP

Action parameter: the current attenuated dose

Condition parameters:
NORMALCOUNTS—the blood counts do not warrant dose attenuation
CYCLE—the current chemotherapy cycle number
SIGXRT—the number of cycles since significant radiation

Abbreviations in the *Value* column:
V_1—the previous dose advanced by 50 mg/m^2
V_2—250 mg/m^2 attenuated by the minimum count attenuation
V_3—the minimum of 250 mg/m^2 and the previous dose
V_4—the minimum of 250 mg/m^2 and the previous dose attenuated by the minimum count attenuation.

Evaluation	*Rule*	*Value*	*NORMALCOUNTS*	*CYCLE*	*SIGXRT*	*Combination*
	80	250mg/m^2	YES	1	1	C_1
	76 (D)	V_1	YES	(1)	(1)	C_1
R	667	V_2	NO	1	1	C_2
R	67	V_2	NO	1	1	C_2
	76 (D)	V_1	YES	(1)	(OTHER)	C_3
M	—		NO	1	OTHER	C_4
	82	V_3	YES	OTHER	1	C_5
	76 (D)	V_1	YES	(OTHER)	(1)	C_5
C	600	V_3	NO	OTHER	1	C_6
C	69	V_4	NO	OTHER	1	C_6
	76 (D)	V_1	YES	(OTHER)	(OTHER)	C_7
M	—		NO	OTHER	OTHER	C_8

Summary of Comparison

Conflict exists in combination(s): C_6 (RULE600 RULE069)
Redundancy exists in combination(s): C_2 (RULE667 RULE067)
Missing rules are in combination(s): C_4, C_8

Notes

Evaluation: M—missing; C—conflict; R—redundant

Rules: Default rules are indicated by (D).

Values of condition parameters: A value in parentheses indicates that the parameter is not explicitly used in the rule, but the rule will succeed when the parameter has the indicated value.

FIGURE 8-1 An example of output from ONCOCIN's rule-checking program.

Missing rule corresponding to combination C_4:

To determine the current attenuated dose for Cytoxan in CVP
IF: 1) The blood counts do warrant dose attenuation,
2) The current chemotherapy cycle number is 1, and
3) This is not the start of the first cycle after *significant radiation*
THEN: Conclude that the current attenuated dose is . . .

FIGURE 8-2 Proposed missing rule (English translation). Note that no value is given for the action parameter; this could be filled in by the system builder if the rule looked appropriate for addition to the knowledge base.

with the rule model for its action parameter. TEIRESIAS proposed *missing clauses* if some condition parameters in the model did not appear in the new rule.

8.3.3 An Example

ONCOCIN's rule-checking program can check the entire rule base, or can interface with the system's knowledge acquisition program and check only those rules affected by recent changes to the knowledge base. This latter mode is illustrated by the example in Figure 8-1. Here the system builder is trying to determine if the recent addition of one rule and deletion of another have introduced errors.

The rules checked in the example conclude the current attenuated dose for the drug cytoxan in the chemotherapy named CVP. There are three condition parameters commonly used in those rules. Of these, NORMALCOUNTS takes YES or NO as its value. CYCLE and SIGXRT take integer values. The only value of CYCLE or SIGXRT that was mentioned explicitly in any rule is 1; therefore, the table has rows for values 1 and OTHER (i.e., other than 1).

The table shows that Rule 80 concludes that the attenuated dose should have a value of 250 milligrams per square meter when the blood counts do not warrant dose attenuation (NORMALCOUNTS = YES), the chemotherapy cycle number is 1 (CYCLE = 1), and this is the first cycle after significant radiation (SIGXRT = 1). This combination of values of the condition parameters is labeled C_1.

Rule 76, shown next in Figure 8-1, can succeed in the same situation (C_1) as Rule 80, but it concludes a different dose. These rules do not conflict, however, because Rule 76 is a default rule, which will be invoked only if all normal rules (including Rule 80) fail. Note that NORMALCOUNTS is the only condition parameter that appears explicitly in Rule 76, as indicated by the parentheses around the values of the other two

Rule set: 33 24

Context: the drug DTIC in the chemotherapy ABVD

Action parameter: the dose attenuation due to low WBC

Default value: 100

Evaluation	*Rule*	*Value (percentage)*	*WBC (in thousands)*	*Combination*
			0 1.5 2 3 5	
	33	25	.... *** 0	C_1
	24	50	..... ***0 ...	C_2

Summary of Comparison

No problems were found.

Notes

Asterisks appear beneath values included by the rule.
Zeros appear beneath upper and lower bounds that are not included.
(e.g., Rule 33 applies when $1.5 \leq \text{WBC} < 2.0$)

FIGURE 8-3 A table of rules with ranges of numerical values.

parameters. Rule 76 will succeed in all combinations that include NORMALCOUNTS = YES (namely C_1, C_3, C_5, and C_7).

Rules 667 and 67 are redundant (marked R) because both use combination C_2 to conclude the value labeled V_2 (250 mg/m^2 attenuated by the minimum count attenuation).

Rule 600 is in conflict with Rule 69 (both marked C) because both use combination C_6 but conclude different values (and both are categorized as normal rules).

No rules exist for combinations C_4 and C_8, so the program hypothesizes that rules are missing.

The system builder can enter ONCOCIN's knowledge acquisition program to correct any of the errors found by the rule checker. A missing rule can be displayed in either LISP or English (Figure 8-2) and then added to the system's knowledge base after the expert has provided a value for its action parameter.

If a summary table is too big to display, it is divided into a number of subtables by assigning constant values to some of the condition parameters. If the conditions involve ranges of numeric values, the table will display these ranges graphically as illustrated in Figure 8-3.

8.4 Effects of the Rule-Checking Program

The rule-checking program described in this chapter was developed at the same time that ONCOCIN's knowledge base was being built. During this time, periodic runs of the rule checker suggested missing rules that had been overlooked by the oncology expert. They also detected conflicting and redundant rules, generally because a rule had the incorrect context and therefore appeared in the wrong table.

A number of inconsistencies in the use of domain concepts were revealed by the rule checker. For example, on one occasion the program proposed a missing rule for a meaningless combination of condition parameter values. In discussing the domain knowledge that expressed the interrelationship among the values, it became clear that a number of individual yes/no valued parameters could be represented more logically as different values for the same parameter.

The knowledge engineers and oncology experts alike have found the rule checker's tabular display of rule sets much easier to interpret than a rule-by-rule display. Having tabular summaries of related rules has facilitated the task of modifying the knowledge base. Although the program described assists a knowledge engineer in ensuring the consistency and completeness of the rule set in the ONCOCIN system, its design is general, so it can be adapted to other rule-based systems.