ELSEVIER

Knowledge-Based Systems 12 (1999) 27–35

Knowledge-Based SYSTEMS

Applications of rule-base coverage measures to expert system evaluation

V. Barr*

Department of Computer Science, Hofstra University, Hempstead, NY 11550, USA

Received 24 September 1998; received in revised form 5 January 1999; accepted 5 January 1999

Abstract

Often a rule-based system is tested by checking its performance on a number of test cases with known solutions, modifying the system until it gives the correct results for all or a sufficiently high proportion of the test cases. This method cannot guarantee that the rule-base has been adequately or completely covered during the testing process. We introduce an approach to testing of rule-based systems, which uses coverage measures to guide and evaluate the testing process. In addition, the coverage measures can be used to assist rule-base pruning and identification of class dependencies, and serve as the foundation for a set of test data selection heuristics. We also introduce a complexity metric for rule-bases. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Rule-base system; Casual-associational network; Logical path graph

1. Introduction

Evaluation of a knowledge-based system is a multi-faceted problem, with numerous approaches and techniques. The results generated by the system must be evaluated, along with its features, the usability of the system, how easily it can be enhanced, and whether or not it has a positive impact on the people who are using the system in place of an approach which is not computer based. The system's performance must also be evaluated in light of its intended use [1]. If the expert system is meant to function as an intelligent assistant then it must satisfy the criterion of being a useful adjunct to the human problem solver. If the system is expected to emulate the reasoning of a human expert then a more rigorous evaluation of the system is needed.

During the last 20 years there has been considerable development and use of knowledge-based systems for medical decision support. In this period there has been heavy emphasis on functional analysis, addressing two primary questions:

- Does the system give the results we expect on test cases?
- Does the system improve the effectiveness of those who use it?

The emphasis on functional analysis can lead to seemingly strong statistical statements about the correctness of a system, demonstrating that it gives the correct result, or the same result as a human expert, in a high percentage of test cases. However, functional testing does not guarantee that all parts of the system are actually tested. If a section of the rule-base is not exercised during the functional test then there is no information about that section of the system and whether it is correct or contains errors. Further, many performance problems for rule-bases result from unforeseen rule interactions [2]. A test suite of known cases may never trigger these interactions, though they should be identified and corrected before a system is put into actual use.

The method we present enhances functional analysis of rule-based classification systems with a *rule-base coverage* assessment, overcoming limitations of common methods for rule-based expert systems evaluation. The underlying premise of this work is that an ideal testing method is one that guarantees that all possible reasoning paths through a rule-base have been exercised. As with procedural software, this is often an unreasonable and/or unattainable goal, possibly due to a lack of test data, to un-executable program paths, or to the size of the rule-base. Further, even if each possible path is exercised, we cannot realistically do so with each distinct set of test values that could cause its traversal. A reasonable goal is for the rule-base testing process to exercise every inference chain or provide information about the failure of the testing process to do so.

Usually verification and validation (V & V) of rule-based systems involves a static structural analysis (verification) method to detect internal inconsistencies, followed by a dynamic, functional, validation in which system behavior on a set of test cases is compared with expected results. The

E-mail address: vbarr@magic.hofstr.edu (V. Barr)

PII: S0950-7051(99)00005-2

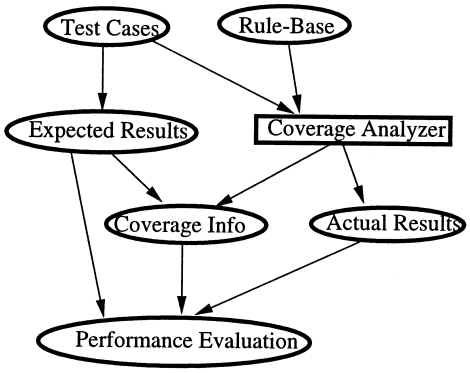

Fig. 1. Rule-base evaluation with coverage analysis.

weakness of a strictly functional approach to validation is that the test data available may not adequately cover the rule-base, and, at best, limited information about coverage will be obtained. System performance statistics are usually presented as if they apply to the entire rule-base, rather than just to the tested sections. This can lead to false estimates of system performance in actual use. The system performance indicated by the comparison of actual and expected results is relevant only for the tested sections, while performance in the untested sections cannot be predicted.

We must also consider completeness of the test set and age of the rule-base by the test data. *Completeness* of the test set refers to the degree to which the data represents all types of cases, which could be presented to the system under intended conditions of use. *Coverage* of the rule-base refers to how extensively possible combinations of inference relations are exercised during test data evaluation. In the trivial case, with a correct rule-base and a complete test suite, the test data would completely cover the rule-base, all actual results would agree with expected results, and we could predict completely correct performance of the rule-base in actual use. In the more usual situation we may have errors and incompleteness in the rule-base, as well as inadequacies in the test data. If we only judge the system based on a comparison of actual and expected results, the rule-base could perform well on the test data, but actually contain errors which are not identified due to incompleteness of the test data. This could lead to a false prediction of correct performance on all cases, when in fact we cannot make any accurate prediction about performance of the rule-base in those areas for which there is an absence of test data.

Our testing approach, as outlined in Fig. 1, allows clear identification of incompleteness in the test data and potential errors in the rule-base through identification of sections of the rule-base that have not been exercised during functional test. This can indicate weaknesses in the test set and/or sections of the rule-base that may not be necessary. An incomplete test set can be supplemented with additional cases chosen from the available population, guided by a series of heuristics and the coverage analysis information. Alternatively, if there is no test data which covers certain parts of the system, it is possible that those sections should be pruned from the rule-base or modified.

Our approach carries out structural analysis of the rule-base using five rule-base coverage measures (RBCMs) which identify sections not exercised by the test data. This makes it possible to improve completeness of the test suite, thereby increasing the kinds of cases on which the rule-base will be tested and improving coverage of the rule-base.

In addition to the coverage analysis, we employ a rule-base representation which facilitates application of the coverage measures; a set of heuristics for re-sampling the population of available test cases, based on coverage information, as shown in Fig. 2; strategies for rule-base pruning and identification of class-dependencies; a rule-base complexity metric. In another study [3] the utility of the aforementioned is illustrated extensively using rule-bases which were prototypes for the AI/RHEUM system [4] and the TRUBAC (Testing with RUle-BAse Coverage), a tool which implements the coverage analysis method.

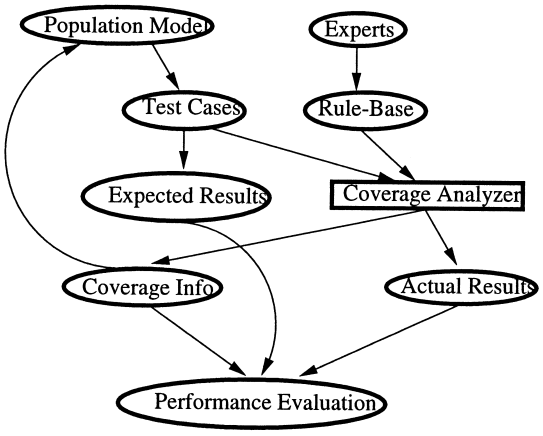

Fig. 2. Evaluation with coverage analysis and data re-sampling.

2. Related work

This work builds on both coverage-based testing methods for procedural software (see [5] for a review of methods and [6,7] for a data-flow approach to testing) and earlier work on rule-base analysis. Early approaches for rule-base analysis carried out only verification or validation. A number of systems, such as the ONCOCIN rule checker program (RCP) [8], CHECK [9,10], ESC (expert system checker) [11], and KB-Reducer [12,13] carry out only verification. Beyond their limitation to verification, these systems have additional weaknesses. RCP is limited to identification of static problems at the rule level, and cannot identify problems that result along longer reasoning chains. The CHECK system has better complexity than the RCP, but can be used only for systems developed using LES, the Lockheed expert systems development environment. ESC is very efficient if there are no conflicts or redundancies in

the rule-base, but can require exponential time if such problems exist. KB-Reducer is a verification tool, which operates on an implied network of rules. It has the advantage of checking a rule-base for inconsistency and redundancy over inference chains, and not just pairs of rules.

A number of dynamic analysis tools have also been developed, such as TEIRESIAS [14], and SEEK2 [15]. TEIRESIAS aids in debugging and knowledge acquisition by allowing the alternation, deletion or addition of rules in order to fix errors in the rule-base that led to incorrect conclusions. However, this process requires that the system tester is sufficiently expert in the problem domain to identify errors in the reasoning the system used to reach a conclusion. SEEK2 is an automated rule-base refinement tool which tries out various rule refinements based on the system's performance on known cases. However, the quality of the refinements produced will be determined by the breadth of the test cases used. SEEK2 does not judge how well the test cases cover the range and domain of the system being evaluated.

2.1. Causal-associational network

Graph-based methods for rule-base analysis involve the creation of a graph representation of the rules. An early example of such a representation (though not specifically used for V & V) is the causal-associational network (CASNET) for glaucoma diagnosis [16]. While our approach is similar to the approach used in CASNET, there are a number of differences, which stem from the ways the two methods use the graph structure. In CASNET the network serves as a direct representation of the knowledge, with interior nodes representing intermediate stages of disease progression. In our approach the graph is a direct representation of the knowledge base and intermediate nodes represent intermediate hypotheses in a logical sense, but may have no particular meaning relative to the problem domain unless the system designer built that into the rules.

2.2. KB-Reducer

As mentioned earlier, KB-Reducer [12,13] uses the implied network of rules to do the rule-base analysis. The process of knowledge base reduction involves calculation of all possible logically independent and minimal sets of inputs under which the knowledge base will conclude each assertion (each class). In order for the reduction process to work, the rules of the knowledge base must form an acyclic network under the *depends-on* relation, defined in Ref. [12]. If the network is acyclic, then KB-Reducer proceeds to label each hypothesis H with the set of environments that lead to the assertion of H, where an environment is itself a set of findings. KB-Reducer carries out the labeling process on the rules in an order such that no rule is processed before any rules on which it depends. As each rule is processed, KB-Reducer updates the partial label for the hypotheses the rule asserts, and checks for redundancy and contradiction.

2.3. Completeness Verifier

COVER (COmpleteness VERifier) [17–19] is another approach which combines both a functional and structural analysis of the rule-base. COVER carries out seven verification checks: redundancy, conflict, subsumption, unsatisfiable conditions, dead-end rules, circularity and missing rules. The rules must either be written in or converted to a language based on first-order logic, and COVER must be given the set of final hypotheses (classes), as well as information about any semantic constraints.

A primary difference between COVER and the work described here is in the granularity of the graph constructed. In COVER the nodes of the graph represent rules, and the edges represent dependencies or relations between rules, while in our representation each rule is itself represented by a small sub-graph, which allows us to carry out a more detailed analysis of which inference relations have and have not been exercised by the test data.

2.4. Pr/T nets

The graph representation we are using is closer in some respects to the Pr/T net representation of a rule-base than it is to any of the other graph-based methods. In the majority of the graph-based methods a rule from the rule-base is equivalent to a node in the graph, while in our graph nodes correspond to individual findings or hypotheses, not to entire rules. Similarly, in the Pr/T net representation [20] the level of details is the findings and hypotheses, not the rules. However, only static analysis of the rule base is carried out using the Pr/T net representation.

2.5. Path Hunter and Path Tracer

The VV & T approach based on the execution path model, incorporated in Path Hunter and Path Tracer [21,22], shares the fundamental premise upon which our work is based: functional validation may show that the system performs well on the test cases, but there may still be problems in portions of the rule base that were never exercised during testing. The goal of Path Hunter/Path Tracer is the selection of a set of test cases that exercise the structural components of the rule-base as exhaustively as possible. This involves firing all rules, and also firing every "causal sequence" of rules. The model used to identify all possible dynamic causal rule firing sequences is the *rule execution path* (equivalent to a sub-DAG in our representation).

Path Hunter uses structural path analysis to detect potential interactions between rules in a rule-based and to identify problems within the rule-base, essentially using a path enumeration step. The complexity is controlled by precomputing the logical completion for each subproblem and by

the use of equivalence classes of rules, formed by collecting redundant rules together into one class, which reduces the number of paths that Path Hunter must generate. (In our approach the step of explicit identification of redundant and ambiguous rules is unnecessary, as they will be identified through the graph construction process. Therefore, while there may be redundant rules in the rule-base, there will be no duplication of those rules within our graph representation.)

Path Tracer is a tool for structural rule-base testing, using the paths generated by Path Hunter, in conjunction with traces of dynamic rule firings, to determine how extensively the possible execution paths are covered by the test data.

While there are a number of similarities in the premises which underlay our approach and Path Hunter/Path Tracer, there are also significant differences between the two approaches. First, the graph we construct models the rule-base at the level of findings and hypotheses, rather than at the rule level as is carried out in the Path Hunter representation. While this may make the graph somewhat larger, it allows us to carry out both V & V with one representation. This is in contrast to COVER, which requires two representations, the first-order logic translation of the rules and the dependency graph, in order to carry out verification alone.

Second, the methods by which rule-base coverage are determined or measured are quite different. Path Tracer does its assessment of path coverage based on the number of causal dependencies observed in the trace file after the test data is run, using a number of strategies to map concrete paths observed at run time to the abstract paths generated by Path Hunter. In our approach, as the effect of concrete firings is indicated directly in the graph representation of the rule-base, we can determine rule-base coverage directly from the graph after the test data is run. We determine the extent of rule-base coverage by considering whether the state of the graph after the test data is run satisfies the four rule-base coverage measures.

2.6. Logical path graph model

The logical path graph (LPG) model [23,24] is based on program control flow analysis. It attempts to apply cyclomatic complexity and basis path testing to the rule-base environment to find a graphical representation of rule-bases which could then be used to determine rule-base complexity and determine a set of paths through the rule-base that, when executed, would adequately test the rules and their interactions.

The logical path graph is a directed graph in which the nodes represent individual rules and the edges are determined by logical paths through the rule base. The goal of Kiper's work [23,24] is to use the LPG to determine a set of paths through the rule-base such that traversal of those paths during system testing represents an adequate test of the rules and their interactions. However, there are certain problems that arise in the use of logical paths. If there are multiple edges entering a node in the graph, they can be interpreted either as an AND or an OR relation. In order to avoid the possibility of OR edges the node must be replicated, in effect creating in the LPG the kind of redundancy which we usually try to remove from rule-bases. This results in the possibility of a single rule being represented by multiple nodes. If both AND and OR edges are to be allowed then the person evaluating the LPG must know what type each edge is. In addition, because of the possibility of multiple nodes representing a single rule, each node has to be labeled not just with the rule number but also with the *condition set*, the set of all conditions asserted by nodes on the path leading to the node.

There are significant differences between the LPG and our approach. The graph structure we propose is based on findings and hypotheses and directly models the logical relations within rule antecedents, whereas the structure used for the LPG is built at the rule level. This difference in the graph construction allows us to avoid putting identifying information on edges or replicate rule representations, as is the case in the LPG.

3. Testing with rule-base coverage measures

The first step in rule-base testing with coverage measures is to build a graph representation of the rule-base. Our method uses a directed acyclic graph (DAG) representation. We assume a generic propositional rule-base language [3] into which other rule-base languages can be translated. During construction of the DAG, pairwise redundant rules, pairwise simple contradictory rules and potential contradictions (ambiguities) are identified. After DAG construction is complete, static analysis (verification) of the rule-base reports dangling conditions (an antecedent component that is not defined as a finding and is not found as the consequent of another rule), useless conclusions, and cycles in the rule-base. At this point the rule-based could be modified to eliminate or correct any static problems.

The static analysis phase is followed by dynamic analysis of the rule-base using test cases. As test cases are processed, one or more of several rule-base coverage measures (RBCMs) can be reviewed in order to determine the quality of the test data supplied thus far. Additional information about the rule-base and its testing can also be used by the system tester to guide the selection of future test data. The tester would start by providing sufficient test data to satisfy the simplest functional measure (conclude each class of the system) and proceed to the more difficult structural measures. Finally, if the user is not able to provide sufficient data to attain the desired degree of rule-base coverage (according to the selected criterion), the user can use the DAG representation to synthesize data, which can then be reviewed by an expert to determine if the data represents a valid case in the problem domain.

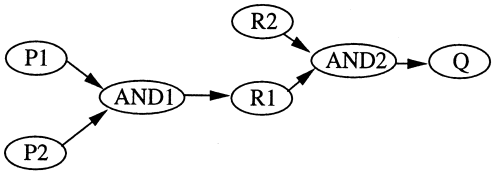

Fig. 3. DAG representation of two related rules.

This testing approach, described more fully later, has been implemented in the TRUBAC tool (Testing with RUle-BAse Coverage) [3,25].

3.1. Rule-base representation

In order to evaluate the rule-base coverage, the rule-base must be in a form which allows identification of sections which have and have not been covered by the test data. Our representation is based on the AND/OR graph implicit in the rule base [26]. The DAG has a source node, corresponding to working memory, and a sink node, corresponding to success in reaching one of the classes (diagnoses or goals) of the system. Interior nodes are sub-class nodes (SUBs), representing intermediate hypotheses, and operator nodes, representing the allowable operators AND, OR, and NOFM[1] nodes. These operator nodes represent the fact that the conjunction and/or disjunction of multiple components of an antecedent must be true in order for the conclusion of a rule to be entered into working memory. There are edges from the source to each finding and from each class to the sink. The antecedent of each rule is represented by a subgraph, which connects findings and sub-class nodes to operators as indicated by the antecedent. Each antecedent-consequent connection represented by a rule is also represented by an edge from the subgraph for the antecedent to the node for the consequent:

For example, the representation of the two rules

If *P*1 and *P*2 then *R*1

If *R*1 and *R*2 then *Q*

is shown in Fig. 3. P1, P2 and R2 could be findings or sub-class nodes, while R1 and Q are either class or sub-class nodes. The complete graph of a rule-base is constructed by linking together the individual structures for successive rules. Using this framework we can easily represent rule-bases, including those in which the certainty factors are hard coded within the consequent definitions (rather than computed during the inference process) such as the rules in Fig. 4.

3.2. Rule-based coverage measures

Using the DAG structure described earlier, we define *execution paths* in a rule-based system. Each reasoning chain through the rule-base corresponds to a sub-DAG, which includes all nodes and edges in the DAG corresponding to the rules fired during a particular chain of inferences. This include: source node; sink node; all nodes corresponding to the findings involved; the node for the concluded class; all nodes corresponding to antecedent components, operators, and rule consequents; and all edges involved in antecedent–consequent links formed by the rule connections used in the reasoning chain. An individual rule firing involves all edges and nodes in the graph that corresponds to that rule. An *execution path* is, therefore, the sub-DAG that corresponds to all the rules fired along a particular reasoning chain executed due to a specific set of findings.

Ideally, in testing a rule-base, we would like to provide sufficient test data to cause every possible execution path of the DAG to be traversed. This corresponds to firing all rules in every combination possible. As this is usually not reasonable in the testing environment, we propose the following hierarchy of rule-base coverage measures (RBCMs) in order to guide the selection of test data and give an objective measure of how well a test suite has covered the rule-base.

Each-class :Satisfied if the test data causes traversal of one execution path to each class of the system. This is equivalent to providing, for each class, one test case that concludes that class. This is a very minimal coverage measure, which should be satisfied by all expert systems developers, regardless of their overall testing strategy.

Each-hypoth: Satisfied if the test data causes traversal of execution paths such that each sub-class is reached, as well as each class. This shows that each sub-class is actually reachable from the source and *appears* to be a relevant part of the system. The set of test data which satisfies this coverage measure is a superset of that which satisfies *Each-class*.

Each-class-every-sub: There may be many execution paths that connect each sub-class to each class (and no execution paths for some sub-class to class combinations). This coverage measure is satisfied if, for each sub-class to class combination connected by some execution path, at least one execution path, which includes the combination is executed. This coverage measure is stronger than *Each-hypoth*.

Each-class-every-finding: There may be many execution paths that connect each finding to each class (and no execution paths for some finding-class combinations). This coverage measure is satisfied if, for each finding-class combination connected by some execution path, at least one execution path which includes the combination

```
if lgas then flod(0.80)
if fcws and between(flod, 0.20, 1) then wait(0.90)
```

Fig. 4. Rules with hard coded certainty factors.

[1] NOFM nodes represent the construction "if *N* of the following *M* things are true, then…", which is a feature of EXPERT [27] rule-bases.

is executed. This coverage measure is stronger than *Each-class* but is incomparable to *Each-hypoth* as *Each-hypoth* can be satisfied without complete traversal of any execution paths for some finding-class combinations.

All-edges: This RBCM is satisfied if the data causes traversal of a set of execution paths such that every inference relationship (every edge in the graph) is utilized along some inference path. While this will not guarantee that all rules will be used in every combination possible, it will guarantee that every rule is used in all possible ways along some execution path. In a rule-base with no NOFM nodes, the data that satisfies this RBCM will be a superset of the data necessary to satisfy *Each-class-every-finding* and *Each-class-every-sub*.

In typical usage the tester will run a number of test cases and then query TRUBAC to determine whether each RBCM has been satisfied. If a coverage measure is not satisfied then TRUBAC will show what relationships remain to be covered in order to satisfy the RBCM.

We assume the existence of an oracle that determines if the result given by the rule-base for a test case is correct[2] or not. In this work we do not consider the issue of specifications and how we determine if an answer is correct or not. If an answer is wrong, then the execution path that led to its conclusion is suspect and must be studied for errors. If the answer is right then the path is only of interest to the extent that it helps satisfy the coverage measure(s) chosen by the tester.

In a very complex rule-base, or one for which there is very little test data, the RBCMs help the tester determine ways in which the data is deficient in testing portions of the rule-base. The coverage measures provide the user with information that can lead to the acquisition or development of additional test cases, or will indicate errors in the rule-base. This approach can also address a difficulty often faced by those who test expert systems, which is a paucity of test data. An additional feature of this approach to rule-base testing can help the user gain insight into the correctness of the system even when little test data is available. In addition to evaluating the five coverage measures, the DAG framework can be used to generate test data, which would lead to traversal of any execution paths not exercised by the test data. This synthesized data can then be shown to one or more experts in order to determine if each synthesized test case, which is a reflection of the logic embodied within a section of rule-base (corresponding to an execution path), makes sense in the context of the problem domain which the rule-base was designed to handle. If the expert does not agree that the data represents a plausible case in the problem domain then the section of the rule-base represented by that section of the graph must be reviewed for errors.

[2] For a given test case the experts may not agree on one answer, in which case the system will be considered correct if its answer is in the set of answers agreed on by the experts as possible for that case. There are a number of issues that are raised by the desire to have a "gold standard" [28] against which to measure the expert system's answers. For example, if the system's answer agrees with that of the expert, should we consider the system to be correct, even if we subsequently learn that both were wrong.

4. Applications of coverage analysis

In addition to providing information about the testing process itself, the coverage analysis can be used to enhance testing and facilitate other kinds of rule-base analysis, as described below.

4.1. Heuristics for test data selection

There may be a large redundant pool of available test data, from which a subset of cases must be selected for the test suite. Running all available cases can be infeasible due to the length of time it may require. A random selection of test cases may give statistical confirmation that the system works properly for the tested situations, but a poor selection of cases will lead to an incomplete test set which then leads to incomplete coverage of the rule-base. If a test set which we know is incomplete leads to complete coverage of the rule-base then the rule-base is incomplete and is not capable of handling precisely those types of cases that are absent from the test suite. If an incomplete test set leads to incomplete coverage, we can use the coverage information as the foundation for a set of heuristics for test data selection from the available population, in order to construct a test set that will maximize rule-base coverage.

To date we have restricted our work to classification systems in which each goal of the system is a class and we consider each intermediate hypothesis to be a sub-class. Assuming that we have a degree of meta-knowledge about the make-up of individual test cases (e.g. what facts or intermediate hypotheses are involved in each case), a set of heuristics for data selection is:

1. For each class, select a test case which concludes *only* that class.
2. For each class not yet tested, select a test case which will conclude it (and additional classes). At this point *Each-class* will be satisfied.
3. Select test cases which will conclude unused sub-classes. This satisfies *Each-hypoth*.
4. Select test cases which will cover sub-class to class relations, and direct finding to class relations for findings which do not lead to an intermediate sub-class. This satisfies *Each-class-every-sub*.
5. Select test cases which will cover finding to class relations for findings which represent alternative ways to conclude sub-classes. That is, while a sub-class to class relation may have been covered, there may be multiple ways to conclude the sub-class. This satisfies *Each-class-every-finding*.

In [3,29] we show that the use of coverage information,

Table 1
Overlap of sub-classes for class pairs

	PM	PSS	SLE	MCTD	RA
PM		0	0	22.22	22.22
PSS	0		16.67	16.67	16.67
SLE	0	9.09		0	0
MCTD	18.18	9.09	0		**36.36**
RA	28.57	14.29	0	**57.14**	

along with meta-knowledge about the pool of available cases, can successfully be used to select the test cases in a way that contributes to a useful test of the rule-base. This results in greater assurance that the system has been tested and works correctly for the situations that we expect to encounter most often, as well as clearly identifying those parts of the system that still require more examination, testing, or refinement.

4.2. Class dependence

Another aspect of rule-base analysis or evaluation is the identification of class dependencies, in which the rules that lead to the conclusion of one class are also highly involved in the conclusion of another class. If two classes, C1 and C2, are dependent, and we have a large number of test cases that will be classified as C1, it may be that some of those cases will also be classified as C2, although with differing certainty factors. Because of this, testing for one class may help achieve coverage for the other. Further, if C1 and C2 are dependent classes and we change rules for C1 then we should rerun test cases, which conclude C1, and rerun test cases, which conclude C2. This will verify that changing rules for C1 did not inadvertently affect the system's ability to properly classify C2 as well.

To determine if two classes have rules in common we look at sharing or overlap among the sets of sub-classes that can lead to a class. A high degree of overlap among sub-classes implies overlap among the rules and a degree of dependency between those classes.

This information can be obtained immediately from data collected while the DAG representation is built. For each class a list of all the sub-classes which can help to conclude the class is formed, and then these lists are compared for pairs of classes. Table 1 shows the overlap figures for a small prototype of the AI/RHEUM system for rheumatology diagnosis [4]. These figures indicate that 57% of the sub-classes, which lead to RA (rheumatoid arthritis) lead to MCTD (mixed connective tissue disease), while 36% of the sub-classes, which can lead to MCTD also lead to RA. There is asymmetry in these figure's results because the absolute number of sub-classes which can lead to the classes is different. These figures indicate that modifications made to the rules for RA would possibly also affect the performance of the system on cases that should be classified as MCTD, while there would be a lesser affect on the system's classification of cases as RA if the rules for MCTD were modified.

These results are consistent with those found in [30], which uses Monte-Carlo simulation-based techniques to carry out rule-base evaluation. However, in our approach we obtain this information as a by-product of DAG construction, without the overhead of rule-base execution.

4.3. Rule-base pruning

Once a rule-base has been constructed, it is possible that not all the rules are necessary for the rule-base to perform correctly. For example, during incremental development some early rules may be supplanted by rules added later. If the system can be pruned by removing rules or components within rules, and the performance on test cases is not affected, then it is possible that there were unnecessary rules or that the test cases are not adequate to evaluate the entire rule-base [30]. Further, we expect that a smaller rule-base will run more efficiently.

Coverage information can focus the pruning steps on sections of the rule-base which have not been executed by the test data. If a section of the rule-base is never executed during test suite execution then either there are unnecessary rules, which can be pruned, or the test suite is not sufficiently rich. In the latter case, additional test cases are necessary to cover the un-executed section of the rule-base. In general the developer and/or the expert will decide whether the proper approach is to prune or to add test cases.

If the test suite is truly representative of cases that will be found in the application environment, and the rule-base performs correctly on the test set, then it is likely that the uncovered portion of the rule-base is in fact unnecessary and can be pruned. If the test set is complete and the rule-base performs incorrectly then the uncovered sections of the rule-base are candidates for rule refinement. The coverage measures provide information about why the rules were never used, based on findings and sub-classes which appear in rule antecedents but are not present in any test cases. Removal of these antecedent components from the rules may generalize them sufficiently that they will correctly handle some of the test cases and will be covered by an existing portion of the test suite.

In experiments run on the RHEUM rule-base, the first phase of pruning was based on TRUBAC's static analysis results. We were able to prune 5 rules out of 76, as well as eliminate 85 findings and 3 classes. This significantly reduced the overall size of the graph over which coverage was evaluated from 333 nodes to 241 nodes [3]. Two additional pruning iterations, based on the coverage data and the assumption that the test set was complete, eliminated 20 rules, 6 findings, and 15 components of rule antecedents. This resulted in an overall 34% reduction in the number of rules and a 40% reduction in the graph size.

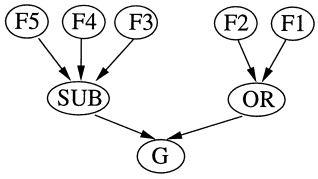

Fig. 5. Graph of rule-base with OR and SUB.

4.4. A metric for rule-based systems

The coverage measures can be very useful for the testing process. However, it would also be useful to have a way to predict or measure the complexity of the system and of the testing process before testing is begun. We would also like to be able to compare the complexity of different rule-bases, particularly if there are multiple rule-bases that handle problems within a common domain. This raises the issue of whether there is a reasonable analog to the control flow graph and, more importantly, a complexity metric for rule-bases which would reflect the relationship between rule-base structure and system complexity.

The graph representation proposed here serves as a suitable foundation for such a complexity metric for rule-based systems. The graph imposes no particular execution order on the rules, and it represents all logical relations that are inherent within the rule-base. However, graph-based metrics such as McCabe's cyclomatic complexity metric cannot adequately determine the number of execution paths in a rule-base. The actual number of execution paths is based on the logical relationships in the rule-base, using the following mechanism:

- For each finding, we assume there is only one path to it.
- For each OR or SUB node, consider the parent nodes. The number of paths to the OR or SUB node is the sum of the paths to the parent nodes.
- For each AND node, compute the product of the number of paths that lead to each parent of the AND.
- Class nodes are treated like OR or SUB nodes. The total number of paths through the rule-base is computed by adding up the number of paths to each class node. In Fig. 5 there are a total of five paths to *G* based on three paths to the SUB node and two paths to the OR node.

This metric can be computed fairly easily by visiting the nodes in topological order and saving in each node the number of paths to that node.

This execution path metric can serve a number of purposes in rule-base development and analysis. The total number of execution paths represents the maximum number of test cases needed for complete coverage of the rule-base according to the strongest rule-base coverage measure (*All-edges*).

However, usually the actual number of data sets needed will be less than the number of execution paths, as often, particularly in diagnosis systems, one test set may cover a number of execution paths to different diagnoses.

5. Conclusions and future work

This work shows that there are numerous uses of rule-base coverage data in the testing process. Rule-base performance evaluation can be misleading unless care is taken to identify problems with both the test data and the rule-base. Both the test data and the rule-base can be improved by using information about the extent to which the test data has covered the rule-base under test.

This work can be extended in a number of directions. Quantitative performance prediction can be computed based on performance of the system on test cases, a measure of how well the test data covers the rule-base, and a measure of the degree to which the test set is representative of the population for which the system is intended. A second area of extension is for systems which have dynamic computation of certainty factors, which requires modification of the rule-base coverage measures [3] as well as changes to the implementation and the data selection heuristics. It would also be useful to extend the approach to systems that are not acyclic and prepositional. This would greatly increase the practicality of this method for testing rule-bases in a variety of application areas.

Another area which should be studied in the future is that of the relationship between rule-base complexity and the difficulty of carrying out the testing process. While intuitively it may seem that a more complex rule-base should undergo more complex and stringent testing, it may in fact be the impact of an incorrect result that should determine the quality of the testing. For example, in a medical diagnosis system if an incorrect result could lead to not providing treatment to an ill patient then all steps possible should be taken to ensure that the system works correctly, no matter how complex or simple the rule-base is.

Finally, it may be possible to extend this approach to analyze the Bayesian belief networks. The probabilistic relationships between nodes of the network, with information flow which is bi-directional along the arcs, and nodes which may be dependent in some contexts and independent in others, significantly complicates this task.

References

[1] P. Jackson, Introduction to Expert Systems, 2, Addison-Wesley, Reading, MA, 1990.
[2] R. O'Keefe, D.E. O'Leary, Expert system verification and validation: a survey and tutorial, Artificial Intelligence Review 7 (1993) 3–42.
[3] V. Barr, Applications of rule-base coverage measures to expert system evaluation, PhD thesis, Rutgers University, 1996.
[4] L.C. Kingsland, The evaluation of medical expert systems: experiences with the AI/RHEUM knowledge-based consultant system in rheumatology, Proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care, Washington DC, 1985 pp. 292–295.
[5] W.R. Adrion, M.A. Branstad, J.C. Cherniavsky, Validation, verification, and testing of computer software, ACM Computing Surveys 14 (2) (1982) 159–192.

[6] P. Frankl, E. Weyuker, A data flow testing tool, Proceedings of IEEE Softfair II, San Francisco, December 1985.
[7] S. Rapps, E. Weyuker, Selecting software test data using data flow information, IEEE Transactions on Software Engineering 11 (4) (1985) 367–375.
[8] M. Suwa, S.C. Scott, E.H. Shortliffe, An approach to verifying completeness and consistency in rule-based expert system, AI Magazine 3 (4) (1982) 16–21.
[9] T.A. Nguyen, W.A. Perkins, T.J. Laffey, D. Pecora, Checking an expert systems knowledge base for consistency and completeness, Proceedings of the Ninth IJCAI, Menlo Park, CA, 1985, pp. 374–378.
[10] T.A. Nguyen, W.A. Perkins, T.J. Laffey, D. Pecora, Knowledge base verification, AI Magazine 8 (2) (1987) 69–75.
[11] B.J. Cragun, H.J. Steudel, A decision-table-based processor for checking completeness and consistency in rule-based expert systems, International Journal of Man-Machine Studies 26 (1987) 633–648.
[12] A Ginsberg, A new approach to checking knowledge bases for inconsistency and redundancy, Proceedings of the Third Annual Expert Systems in Government Conference, Washington DC, 1987, pp. 102–111.
[13] A. Ginsburg, Automatic Refinement of Expert System Knowledge Bases, Pitman, London, 1988.
[14] R. Davis, Interactive transfer of expertise, in: B.G. Buchanan, E.H. Shortliffe (Eds.), Rule-Based Expert Systems, Addison-Wesley, Reading, MA, 1984, pp. 171.
[15] A. Ginsberg, S. Weiss, P. Politakis, SEEK2: a generalized approach to automatic knowledge base refinement, Proceedings of IJCAI-85, 1985.
[16] S.M. Weiss, C.A. Kulikowski, S. Amarel, A. Safir, A model-based method for computer-aided medical decision-making, Artificial Intelligence 11 (1978) 145–172.
[17] A.D. Preece, Verification of rule-based expert systems in wide domains, Research and Development in Expert Systems VI, Proceedings of Expert Systems '89, British Computer Society Specialist Group on Expert Systems, London, 1989, pp. 66–77.
[18] A.D. Preece, R. Shinghal, Foundation and application of knowledge base verification, International Journal of Intelligent Systems 9 (1994) 683–701.
[19] P.D. Grogono, A.D. Preece, R. Shinghal, C.Y. Suen, A review of expert systems evaluation techniques, Workshop on Validation and Verification of Knowledge-Based Systems, 11th National Conference on Artificial Intelligence, Washington DC, 1993 pp. 120–125.
[20] D. Zhang, D. Nguyen, A technique for knowledge base verification, IEEE International Workshop on Tools for Artificial Intelligence, 1989, pp. 399–406.
[21] C. Grossner, A.D. Preece, P.G. Chander, T, Radhakrishnan, C.Y. Suen, Exploring the structure of rule based systems, Proceedings of the 11th National Conference on Artificial Intelligence, Washington DC, 1993, pp. 704–709.
[22] A.D. Preece, C. Grossner, P.G. Chander, T. Radhakrishnan, Structural validation of expert systems using a formal model, Workshop on Validation and Verification of Knowledge-Based Systems, 11th National Conference on Artificial Intelligence, Washington DC, 1993, pp. 19–26,.
[23] U. Gupta, J. Kiper, B. Ly, A. Preece, Developing criteria for comparing specific V and V tools (from First Winter Workshop on Verification and Validation of Knowledge-Based Systems), AAAI-92 Workshop on Verification and Validation of Knowledge-Based Systems, San Jose, CA, 1992.
[24] J. Kiper, Structural testing of rule-based expert systems, ACM Transaction on Software Engineering and Methodology 1 (2) (1992) 168–187.
[25] V. Barr, TRUBAC: a tool for testing expert systems with rule-base coverage measures, Proceedings of the 13th Annual Pacific Northwest Software Quality Conference, Portland, OR, 1995.
[26] P. Meseguer, Structural and performance metrics for rule-based expert systems, Proceedings of the European Workshop on the Verification and Validation of Knowledge Based Systems, pp. 165–178, Cambridge, England, 1991.
[27] S.M. Weiss, K.B. Kern, C.A. Kulikowski, M. Uschold, A guide to the EXPERT consultation system. Technical Report CBM-TR-94, Department of Computer Science, Laboratory for Computer Science Research, Rutgers University, 1987.
[28] B.G. Buchanan, E.H. Shortliffe, Rule-Based Expert Systems The problem of evaluation, Addison-Wesley, Reading, MA, 1985.
[29] V. Barr, Rule-base coverage analysis applied to test case selection, Annals of Software Engineering, 1997.
[30] N. Indurkhya, Monte-carlo simulation-based evaluation and refinement of rule-based systems, Technical Report DCS-TR-277, Department of Computer Science, Rutgers University, 1991.