Structure-Based Validation of Rule-Based Systems

Alun D. Preece[‡], Clifford Grossner[†], P. Gokul Chander[†], and T. Radhakrishnan[†]

[‡] Department of Computing Science
University of Aberdeen
King's College, Aberdeen
Scotland, United Kingdom AB24 3UE
Phone +44 1224 272296; FAX +44 1224 273422
Email apreece@csd.abdn.ac.uk

[†]Computer Science Department, Concordia University
1455 De Maisonneuve Blvd. Ouest
Montréal, Québec, Canada H3G 1M8

Abstract

A central problem in validating rule-based systems is ensuring that the structural components of the system have been tested sufficiently thoroughly. In this paper, we present a method that can be used for structure-based testing of rule-based systems. Our method includes a model for determining the structural components in a rule base, a metric for quantifying coverage (the degree to which the structural components in a rule base have been tested), and techniques for analyzing the coverage obtained when a rule base is exercised on a given test set. We present the results produced by applying our method for structure-based testing to a complex rule-based system that had been previously subjected to functional testing. The results we obtained indicate that our method for structure-based testing reveals several types of faults not shown by previous testing, and quantifies the extent to which the rule base has been tested.

Keywords Knowledge-based systems, rule-based systems, software engineering, structural analysis, testing, validation tools.

1 Introduction and Motivation

In software engineering, *validation* is the process of determining whether or not a program meets the requirements of its users [2, 40]. Here, *users* includes prospective hands-on users of the program, experts in its application domain, and the user-organisation as a whole. *Rule-based systems* are best-suited to solving problems in application domains characterised by having a great many special cases, and therefore not amenable to solution by general algorithms [39]. A rule-based system comprises a *rule base*, which is a collection of if-then rules, encoding the actions the system should take in specific cases; and an *inference engine*, which selects and applies rules appropriate to the case-at-hand [11].

1.1 Validation of Rule-Based Systems

In validating a rule-based system, users want to be assured that the system will function adequately well in all cases; that is, that the rules the system applies in each case are appropriate.[1] Ideally, we would like to check the program against a precise specification of what it would mean for the system to "function adequately well in all cases". Unfortunately, it has been shown that this is not possible in general [42]; for rule-based systems, it is not even practical, because the rule base itself is the best specification of what the system should do in each case [23, 38]. Pragmatically, the best we can do is to check the structure and behaviour of the rule-based system with reference to a number of criteria that have been shown to be important in the software validation literature [4, 17, 32]. The main approaches are summarised below.

1.1.1 Static Verification

Static verification techniques check the rule base of the rule-based system for logical anomalies such as inconsistency and incompleteness. The motivation is that a rule base containing

[1] In additional to functional adequacy, there are other facets of validity, including *usability*, which we do not consider here. Broader perspectives on validation are provided by [1, 26].

such anomalies is unlikely to be valid. A number of automatic checking tools have been developed to perform static verification of rule-based systems, for example, [3, 12, 31, 35]). These tools check the rule base in isolation; they do not run the rule-based system as a whole. Typically, the rule base is modeled using propositional or predicate calculus, although decision tables [8, 41] and Petri Nets [25] have also been used. A survey of a representative sample of these approaches is presented in [30].

Use of automatic checking tools makes static verification relatively inexpensive in human cost, although the computational costs may be large (conceivably, intractable) for complex rule bases [24, 44]. The major limitation of static verification is that it does not check the *correctness* of the rule base. All of the tools listed above are capable of showing that a rule base will never produce inconsistent results, but none of them can guarantee that the results the system does produce for a given case are the right ones. Similarly, several of the tools can show that a rule base is complete to the extent that there is no combination of valid input for which the system will fail to produce results, but again there is no way for the tool to check that the results produced are valid. Correctness can be demonstrated only by comparing the results produced by the system with those expected by users; that is, *testing* the system [2, 10].

1.1.2 Function-Based Testing

Function-based testing is a well-known software engineering technique that bases the selection of test cases upon the functional requirements of the system, without regard for how the system is implemented [2, 10, 36]. The pioneering expert system projects MYCIN and XCON were validated in this manner [6, 9]. The success of function-based testing is dependent upon the existence of a "representative" set of test cases for the problem that is solved by the system: the test set should include sample instances for each distinct operation performed by the system, and for each distinct level of difficulty [26]. The distinct operations performed by the system, and their distinct levels of difficulty, need to be determined by consultation with domain experts; preferably, they should be specified in the system requirements documents [5].

The creation of test cases is expensive because test cases must be defined by domain experts, and it is error-prone because the experts may exhibit bias [1, 26]. Function-based testing reveals only whether acceptable results are achieved for the given test cases; a case not in the test set may trigger inappropriate rules at run-time, leading to invalid results. Function-based testing is therefore best-suited to checking that the rules of the system can cope adequately well with instances of "typical" and "unusual" tasks; it is not well-suited to complete testing of situations in which the rules will be triggered.

1.1.3 Structure-Based Testing

Structure-based testing is a technique which complements function-based testing and static verification. In structure-based testing, test cases are selected on the basis of which structural components of the system they are expected to exercise [2, 27, 42]; the objective is to show that the system produces acceptable results for a set of test cases that exercise all structural components of the system. In contrast to function-based testing, which views the system as a "black box", structure-based testing offers a "glass box" view of the workings of the system; thus, it tells us not only whether the results produced by the system are acceptable, but also reveals how those results were arrived at. Inefficient or inappropriate interactions between components of the system will be exposed.

In summary, if a rule-based system has successfully undergone static verification, function-based testing, and structure-based testing, users will be assured that:

- the rule base is free from logical flaws: it will never produce inconsistent results, and it contains rules which cover all valid input combinations;
- the rule-based system produces acceptable results on sample cases representing instances of each "typical" and "unusual" task listed in the requirements documents;
- all structural components of the system have been exercised in testing.