On The Verification, Validation and Testing of Knowledge-Based Systems

Robert T. Plant

ABSTRACT—The aim of this paper is to consider the techniques available to knowledge engineers for the verification and validation of knowledge-based systems. Techniques considered include traditional approaches, formal approaches, and approaches developed within artificial intelligence. First, the paper demonstrates that it is impractical to employ exhaustive testing on knowledge-based systems, but that a case-based approach can be advantageous in that several sets of data are used, each set aimed at identifying a particular error type, and even though they do not show total system correctness individually, collectively, they do raise the level of correctness. The use of formal techniques in artificial intelligence is considered and it is shown how these methods are becoming more acceptable to define certain aspects of systems. These aspects include the static aspects of a system — the knowledge base and the control architecture can be specified. Questions are raised on formal techniques that have, so far, proven to be unsuitable (e.g., the rule interaction). However, the paper emphasizes that the specification of the knowledge base and knowledge representation are significant steps towards verifying and validating the system, because this can eradicate inconsistencies and incompleteness as well as epistemological errors. The use of functional programming and prototyping, the mechanisms through which a heightened level of correctness for the implementation of a knowledge-based system can be achieved, are also considered. Second, the paper suggests two alternative approaches that have assisted software engineers achieve higher degrees of correctness in conventional software systems, but that have not yet been applied to knowledge-based systems development. They are: critical testing (the process of selecting the most critical data to test a system or aspects of a system); and a statistical approach to obtain a reliability measure for the system. We suggest that each of the techniques have their own particular strengths and that, collectively, they are beneficial to the heightened correctness of a system. In order to integrate all of the proposed techniques, we outlined a rigorous development methodology that helps the knowledge engineer to integrate the different validation strategies.

Keywords: Expert systems, knowledge engineering, testing.

Introduction

The aim of this paper is to consider an aspect of artificial intelligence that is of vital importance, yet one that is too often neglected—the validation and verification of knowledge-based systems.

Validation and verification have been defined as follows:

"Validation: The process of evaluating software at the end of the software process to ensure compliance with software requirements [2].

"Verification: The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase [2]."

It follows, therefore, that one of the keys to effective evaluation of the software, and consequently to having valid and verified software, is to have effective testing techniques available. This paper focuses on the testing approaches applicable to assist in the verification and validation of knowledge-based systems.

To get a better perspective, first, consider the error types that can occur in knowledge-based systems and define terminology around them. This is followed by a discussion of the traditional approaches to software testing and the applicability of these techniques to knowledge-based systems testing. Having considered the traditional approaches to testing, several other methods of pursuing

Dr. Plant is a professor in the Department of Computer Information Systems University of Miami, Coral Gables, Florida, 33124.

examine the correctness of the output with respect to the function of the program over that input.

Output=Program(Input)

One approach to software testing is to test all possible inputs and validate their subsequent outputs. The fundamental problem for a practical system (even if finite in nature) is that the number of test paths necessary is extremely high, and, even with the aid of test data generators, the task for nontrivial systems is infeasible.

This problem is magnified for expert systems, as they tend to be nondeterministic in nature, and the systems are partial functions rather than total functions. In addition, expert systems often have to reason with incomplete input data, which raises the number of test cases. Further, the systems are often not static in nature; that is, they may be self-modifying systems that need constant retesting.

Therefore, the use of exhaustive testing is at present an implausible avenue to take. An alternative to exhaustive testing is to apply a test case strategy. For example: "We employed a system validation method described by Waterman [48] which called for a comparison of cases solved by our systems with solutions of human experts"[4].

Other case-based approaches to testing include:

- functional
- structural
- data
- random
- extracted
- extreme

In compiling test data for each, the knowledge engineer has to take into account different aspects—the data types, and the specification of the system—depending upon the aspect desired to examine.

The data used in functional cases is obtained by examining the functionality of the specification. Hayes describes a mechanism through which this can be achieved in a systematic fashion [16], while Ostrand and Salcer have developed a category-partition method for specifying and generating functional tests [30]. Work has also been performed upon the use of functional testing through context dependence [20,22,].
In structural testing the logical structure of the code is examined; one mechanism that can be utilized has been proposed by Rapps et al., who perform data flow analysis on the program to select the test paths [36].

The test set for data cases originates in examining the data elements of the program. Several models of test data adequacy have been proposed such as these by Weyker [49].

An especially important test case strategy is that of testing extreme cases. As it is at the boundary conditions that knowledge-based systems are at their most valuable, they are also their most vulnerable if there is not enough support knowledge to make correct deductions. The testing of extreme cases is difficult, for the location of the boundaries is not always known. This, to a large extent, depends on the interaction of knowledge in the knowledge base. A solution to the selection of extreme data may lie in the examination of the system specification; work is ongoing in this area [34].

Thus, each of these testing strategies examines a different aspect of the system, and collectively, they are valuable in raising the level of system correctness. However, it should be noted that even when all the strategies are used together this does not guarantee total correctness.

Testing of general software under differing case-based criteria has been documented [37, 7, 8].

The Formal Approach

An alternative approach to software development is to implement and then test the system, rather than design it to specify the software system in such a way that it is correct at the design level. This approach, sometimes referred to as "mathematical validation" [31], uses a formal specification style, based upon a formal language, in order to produce a specification that can be reasoned about. This specification can then, through a series of refinement steps, be transformed into an implementation fulfilling the program's "correctness argument" [24]. Currently, in software engineering, a number of proven methods are being applied to real-world applications—these include VDM [24] and Z [42]. As these languages have become more refined, it has become easier to develop specifications in them.

The area of artificial intelligence that has led the way is human-computer interaction [14, 23]. The language Z has been applied to the problem of specification within artificial intelligence. The language was inherently suitable for the area of game playing, as these domains are finite, with well defined structures. Teruel has shown how Z could be applied to this problem domain by specifying games such as Ludo, Othello, and Best of Three [47, 38]. The use of a formal specification language such as Z can also be demonstrated by

"because there are large areas of knowledge that, although amenable in theory to the frequency analysis of statistical probability, defy rigorous analysis because of insufficient data and, in a practical sense, because experts resist expressing their reasoning processes in coherent probabilistic terms" [41].

Following this, several other approaches to the statistical evaluation of system knowledge have been considered, including the theory of fuzzy sets, proposed by Zadah [52]. However, experts also find this an unnatural mechanism in which to relate their knowledge, and thus, it has subsequently found limited pragmatic use. AI workers have also attempted to utilize the theory of choices [46] and confirmation theory [44], but these have not met with total success. This leads to the adoption of the Dempster-Shafer theory of evidence, a model that has many of the advantageous features from the certainty factors approach, but a stronger mathematical basis [39].

The area of certainty factors within knowledge-based systems is one that is still evolving and there are still several problem areas to overcome. First, there is the area of certainty factor interaction; several methodologies have been proposed to deal with this (e.g., Bonczek-Eagin, Max Method, Min Method, Probability Sum Method, Average Method, Bonczek-Eagin Method 2) [19]. However, as the certainties for many different factors are combined together over many system cycles, it has not been shown that these algebras will always give the results intended by the domain expert or even an approximation of them. Also, what is the correlation between one expert's certainty factor for an item and another expert's certainty for the same item or a different item of knowledge?

This approach is beneficial in the appraisal of individual results from the system under test; however, the technique does not enable us to judge the validity of the system as a whole, and again, knowledge engineers will only know that a system is correct for a subset of the total input domain.

Alternative Strategies

In the previous sections of the paper, we have examined the conventional approaches to testing and we have shown that even though each of the techniques has associated with it certain strengths, each is severely limited in its ability to move towards a statement of total correctness. In this section we will consider two alternative strategies to those of conventional testing. First, we will discuss critical testing, where research focuses upon adaptive techniques for optimizing the data sets used in the case approach to testing; this will be followed by consideration of the research into reliability theory, where developers can use and create models that predict the failure of their systems.

Critical Testing

A program is correct when the implementation matches the specification:

$$P(D)=f(D)$$

where

D = Input data (the domain)
P = Program
f = Formal Specification

To do this, it is necessary to perform exhaustive testing,

$$P(d_0)..P(d_n)$$

where n can be very large, if not infinite.

We can, therefore, only test a limited number of cases. Once tested, however, we can then state that the program will perform correctly with respect to this input set:

$$P^*(D)=f(D)$$

where P* = the program tested over the test data set.

The selection of the test data is, therefore, a critical operation, and consequently, the criteria by which the critical test data for an application are selected have been the focus of much research. This area has been influenced by Gerhart, Goodenough, and others who considered the theoretical aspects of procedures to select reliable and valid test data for conventional programs [5, 13]. Their method is based, in part, on the construction of a "condition table" that displays the logically possible combinations of conditions within the program. However, the large number of conditions found in knowledge-based systems may prohibit use of this technique.

An alternative technique to that of Goodenough is the concept of "adequate" test data, which has been defined

"A test data set T is adequate if P behaves correctly on T but all incorrect programs behave incorrectly" [6].

However, it has been shown from a theoretical standpoint that it is not possible to construct a general-purpose test selection procedure for valid test data, as the function is

performed while maximizing the return on that testing. While this development process is occurring, the knowledge engineer can compile data to produce reliability figures. This approach can be combined with some pragmatic techniques, such as ensuring that in critical situations the system is fail-safe or that multiple systems, developed independently, check upon each other's results.

References

1. Abdel-Ghaly, A.A., Chan, P.Y., and Littlewood, B. (1986). Evaluations of competing software reliability predictions. *IEEE Transactions on Software Engineering. SE-12, No 9.* (950-967).

2. Boehm, B.W. (1981). An experiment in small scale application software engineering. *IEEE Transactions on Software Engineering SE-7, No. 5.* (482-493).

3. Budde, R. (1984). *Approaches to prototyping.* New York: Springer-Verlag.

4. Davis, S.J., and Usera, C.A. (1989). Expert system configures intrusion detection systems. *Interfaces 19:(2),* 61-69.

5. DeMillo, R.A., Lipton, R.J., and Sayward, F.G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer, 11,* 34-41.

6. DeMillo, R.A., McCracken, W.M., Martin, R.J., and Passafiume, J.F. (1987). *Software testing and evaluation.* Menlo Park, CA: Benjamin-Cummings.

7. Downs, T. (1985). An approach to modeling software testing with some applications. *IEEE Transactions on Software Engineering. SE-11(4).*

8. Duran, J.W., and Ntafos, S.C. (1984). An evaluation of random testing. *IEEE Transactions on Software Engineering. SE-10(4),* 438-443.

9. Flon, L. and Haberman, A.N. (1976). Towards the construction of verifiable software systems. *Proc. of Conf. on Data Abstraction, Definition and Structure. SIGPLAN Notices* 2, No. (2). 141-148.

10. Floyd, C. (1984). A systematic look at prototyping. In Budde, R. (Ed.), *Approaches to Prototyping.* New York: Springer-Verlag.

11. Gold, D. (1985). *Toward the specification of expert systems.* M.Sc. dissertation. Programming Research Group, University of Oxford.

12. Gold. D., and Plant, R.T. (1990). *The formal specification of a production system.* Working paper CIS/90/2. Department of Computer Information Systems, University of Miami.

13. Goodenough, J.B., and Gerhart, S.L. (1975). Towards a theory of test data selection. *IEEE Transactions on Software Engineering,* SE-1 No.(2), 156-173.

14. Harrison, M., Thimbleby, H. (Eds.) 1990. *Formal methods in human-computer interaction.* New York: Cambridge University Press.

15. Harper, R. (1989). *Investigation of the applicability of a functional programming model to fault tolerant parallel processing for knowledge-based systems.* NASA contractor report 181938.

16. Hayes, I.J. (1986). Specification directed module testing. *IEEE Transactions on Software Engineering. SE-12(1),* 124-133.

17. Henderson, P., Jones, G.A., and Jones, S.B.(1983). The LISPKIT manual. *Programming Research Group Monograph, PRG-31.* Oxford University, Computer Laboratory.

18. Hetzel, W.C. (1984). *The complete guide to software testing.* QED Information Sciences.

19. Holstapple, C.W., and Whinston, A. B. (1987). *Business expert systems.* Homewood IL: Irwin Press.

20. Howden, W.E. (1980). Functional program testing. *IEEE Transactions on Software Engineering, SE-6(2):* March 1980, 162-169.

21. Howden, W.E. (1982). Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering. SE-8, (4),* 371-379.

22. Howden, W.E. (1986). A functional approach to program testing and analysis. *IEEE Transactions on Software Engineering.* SE-12(10); 997-1005.

23. Jacob, R.J.K. (1983). Using formal specifications in the design of a human-computer interface. *Communications of the ACM, (26).*

50. Winston, P.H. and Horn, B.K.P. (1989). *LISP.* Reading, M.A: Addison-Wesley.

51. Woodward, M.R., Hennell, M.A., and Hedley, D. (1980). Experiences with path testing and analysis and testing of programs. *IEEE Transactions on Software Engineering. SE-6 No. (3),* 278-286.

52. Zadeh, L.A. (1965). Fuzzy sets. *Information and Control 8.*