Investigating the Validity of a Test Case Selection Methodology for Expert System Validation

Jan-Eike Michels
Thomas Abel
Rainer Knauf
Technical University of Ilmenau
Ilmenau, Thuringia, Germany
Rainer.Knauf@TheoInf.TU-Ilmenau.de

Avelino J. Gonzalez
Electrical and Computer Engineering Dept.
University of Central Florida
Orlando, FL
USA
ajg@ece.engr.ucf.edu

Abstract

Providing assurances of performance is an important aspect of successful development and commercialization of expert systems. However, this can only be done if the quality of the system can be assured through a rigorous and effective validation process. However, a generally accepted validation technique that can, if implemented properly, lead to a determination of validity (a *validity statement*) has been an elusive goal. This has led to a generally haphazard way of validating expert systems. Validation has traditionally been mostly done through the use of test cases. A set of test cases, whose solution is previously known and benchmarked, is presented to the expert system. A comparison of the system's solutions to that of the test cases is then used to somehow generate a validity statement. It is an intuitive way of testing the performance of any system, but it does require some consideration as to how extensively to test the system in order to develop a reliable validity statement. One completely reliable statement of a system's validity could result from exhaustive testing of the system. However, that is commonly considered to be impractical for all but the most trivial of systems. A better means to select "good" test cases must be developed. The authors have developed a framework for such a system (Abel, Knauf and Gonzalez 1996). This paper describes an investigation undertaken to evaluate the effectiveness of this framework by validating a small but robust expert system to classify birds using this framework.

1. Introduction

Upon completion of a software system, the developer faces the difficult task to prove that the system is valid. This is true for conventional systems in general, but it is particularly difficult for expert systems due to lack of standards against which to validate them.

Whereas the task of system verification (i.e., proving the correct and complete transformation from the domain's specification to its implementation) can mostly be done without human intervention, validation requires human interaction. Validity asks the question: "Is reality accurately represented in the system?", and it is one of the fundamental requirements to assure a system's quality.

Validation has typically taken the form of generating and executing a set of test cases, and comparing the results to some known benchmark or standard. Test case-based validation can be considered as a five-stage process:

1. Test case generation – a set of test cases is created. Result: a set of test cases.
2. Test case experimentation - test cases are executed and the results are compared to benchmarks. Result: a protocol.
3. Test case evaluation – the results of analysis are evaluated. Result: a report.
4. Synthesis of a validity statement - a statement about the validity of the system is produced. Result: a validity statement.
5. System refinement – Any imperfections noted through the testing and evaluation process are corrected. Result: an improved system.

Before the validation process can begin, however, the system should be completely verified. For a more detailed discussion of the first four stages, see (Abel and Gonzalez 1997). The contributions contained in this paper refer to stages 1 and 2 above.

1.1 Test Case Generation

Clearly, the generation of test cases can greatly affect the rigor, and thus, the reliability of any resulting validity statement. The set of test cases must be composed of a sufficient number of "good" test cases to enable a reliable statement.

A naive approach for test case generation would be creating an *exhaustive set of test cases* (EST). The EST consists of all possible input/output-combinations in the system. However, other than for the most trivial of systems that have very few inputs, the EST would be composed of an immense number of test cases, thus making it a highly impractical procedure.

However, oftentimes, different values of one input may have the same abstract meaning, making it redundant to create different test cases with values for the same input that, while different, have the same effect on the solution. Thus, a *"quasi-exhaustive" set of test cases* (QuEST) which recognizes such redundancies, can be generated. Eliminating test cases that are functionally equivalent to

other test cases and subsumed by these accomplishes this. This reduction is domain- and representation-dependent and has to be carried out using only reliable, proven knowledge and known restrictions of the domain and its representation. The QuEST can be significantly smaller in cardinality that the EST, thus making it more practical for use. By definition, a set of test cases is quasi-exhaustive if the execution of this set leads to the same results as would the execution of the EST. See (Herrmann, Jantke and Knauf, 1997) and (Abel, Knauf and Gonzalez 1996] for more details about the QuEST. The advantages of the QuEST are that it is functionally equivalent to the EST, but significantly smaller. Nevertheless, for some systems, the QuEST can still be excessively large.

One way to further reduce the set of test cases beyond the QuEST is to rigorously test the portions of the system that have a high priority, while less so those that have a lesser priority. The priority of a portion of a system can be based on certain user- or developer-defined criteria. Priority of a portion of the system can be defined to mean the cost of an error (financial, in human lives, etc.) in this particular group of rules. A set of test cases derived on the basis of such criteria is called a *"reasonable" set of test cases* (ReST).

In the test case selection stage for the ReST, different kinds of criteria (e.g., domain-related criteria, output-related criteria) are used to select the most relevant test cases to investigate a system's validity. See (Abel and Gonzalez 1997] for more details on this topic.

While the sets of test cases described above may sound conceptually valid as well as desirable, the question remains whether their use can lead to a reliable validity statement. Before answering that question, however, it remains to be seen whether they can be used to successfully detect errors that may exist in a rule base. Of particular interest is the determination of whether the QuEST is in fact functionally equivalent to the EST. The intent of the investigation described in this paper is to answer this last question.

1.2 Objectives of this Research

The objective of this research project is to test the procedures used in the first and second stages of the validation process briefly described in the introduction.

More specifically, this work puts into practice the concepts of the QuEST, and empirically determines their validity by applying them to a small but non-trivial expert system.

The basic idea behind this work was to determine whether errors artificially seeded into the selected test bed expert system would be detected by the test cases included in the QuEST. It was assumed that these errors would be detected by the EST due to its exhaustive nature, as it would have been highly impractical to actually generate, test and evaluate the very large EST for this system.

By detection, it is meant that a test case executed by the expert system provided a solution/answer which did not agree with that of a human expert exposed to the same set of inputs. No attempts were made to determine just how different the answers were. Rather, the existence of a difference in answer was noted. If a test case in the QuEST produced an answer different from that of an expert as a result of the seeded error, then this error was deemed to be detected by the QuEST.

2. Test Bed Expert System

Several expert systems of between 50 and 100 rules were evaluated for use as the test bed for this project. The one chosen was called *The Ornithologist*; written by Mr. Steve Bruce, a Master's student at the University of Central Florida. The Ornithologist was developed with Personal Consultant Plus (PC Plus), an expert system shell originally developed by Texas Instruments. Its purpose is to identify birds that can be observed in Florida. It was designed for bird watching and is, therefore, of a lighter character than, for example, a diagnostic system for power plants. Nevertheless, it has many similarities to more ambitious systems. The Ornithologist identifies 65 different birds, each of which belongs to one of the following species: *water birds*, *ducks*, *shore birds*, *prey birds*, *land birds*, *insect eaters* or *seed eaters*.

2.1 The Knowledge Base

The XPS uses a backward chaining reasoning technique to make a decision. Its knowledge base consists of 71 rules. Sixty-five of these produce final outputs, whereas the other six are intermediate rules that are used by some of the final hypotheses (conclusions). Each rule in the system that represents a hypothesis has a confidence factor of 100%. The Ornithologist uses up to fourteen different inputs, but it does not need all of them at all times. Nevertheless, at least two inputs are always necessary to identify a bird. The inputs are called parameters in the nomenclature of the system. A typical rule looks like this:

IF (SIZE > 17 AND SIZE < 24 AND WINTER-B AND
BILL-G AND AB-DUCK)
THEN (SPECIES = American Black Duck, Anas
rubripes)

This rule will fire if the value for the parameter 'SIZE' is both greater than 17 and less than 24 (inches), the intermediate hypothesis 'WINTER-B' is satisfied, and the values for both parameters BILL-G and AB-DUCK are 'true'. The final output is then a value for the parameter 'SPECIES' which is this time 'American Black Duck, Anas rubripes'.

2.2 Problems with the Ornithologist

It was initially assumed that the Ornithologist was correct and valid as taken from its author. However, during the examination of the Ornithologist, some problems could be found within it. These problems had to be solved, before the investigation could be started.

The Ornithologist was designed such that it is possible for a bird to have more than one different bill shape at the same time. This, of course, is not realistic. The author of the system stated that this was not the intended effect. Thus, the system was treated as if only one bill shape was possible as input.

The author of the system also indicated that he had intended to show the user a scanned picture of the bird as the last question. However, due to time constraints, this was not possible for him to do. That is the reason why the 'last question' describes the bird nearly as if you see it on a picture. Here, it is again theoretically possible to answer two different last questions at the same time with 'yes'. But this doesn't make any sense from a practical standpoint, since a bird could not have two totally different appearances. Therefore, whenever a last question is answered with 'yes', all other last questions are automatically answered with 'no'.

Furthermore, the Ornithologist contained one intermediate rule that was not used by any other rule. Therefore, it could be omitted from the system. Later, it was re-introduced as a modification of the original system.

Another problem existed in that in rule 46, the input sensor BELOW is compared to the value 'white'. Since 'white' is not a legal value for this sensor, this rule never fired. The correct value in this rule should be 'whitish'. This error was fixed (i.e., 'white' was replaced by 'whitish' in the rule).

Another error lies within the tool PC Plus. Although rule 23 was in the knowledge base, it never fired, even when its premise was satisfied. After the same knowledge base was saved without a modification and the expert system was started again, the rule fired. Nevertheless, rule 23 is considered as part of the knowledge base.

3. Empirical Evaluation Procedure

One problem immediately faced was the lack of access to a human expert to assist with the validation process of the QuEST. This was needed in order to determine whether the test case answers were correct or not. To overcome this problem, the original system (as corrected above) was treated as the expert whose answers were always correct. Additionally, errors were seeded into copies of the original system to see whether the test cases that composed the QuEST and the ReST were able to detect them. The test cases were presented both to the faulty system as well as to the original. Differences in their responses were indicative of an error in the faulty version. The work was composed of the following steps:

- Generate the QuEST for the original Ornithologist.
- Derive faulty versions of the Ornithologist by seeding errors into copies of the original system.
- Generate the QuEST's for these faulty systems.
- Use these QuESTs as test cases for the original system as well as for the faulty derived systems.
- Observe if any conclusion could be drawn from the different outputs of the original and the faulty systems.
- Generate the ReST for all systems.
- Observe which errors could still be detected. The observations made are the basis for this paper.

The last 2 steps, those dealing with the ReST are not discussed in this paper.

3.1 The Exhaustive Set of Test Cases (EST)

An exhaustive set of test cases for The Ornithologist consists of all possible combinations of system inputs and the related final outputs. To the outputs belong not only the 65 birds the system can identify, but also an extra output, when there is no rule that can fire for a given input combination: 'The XPS was unable to make any conclusions'.

As mentioned previously, the Ornithologist uses up to 14 different input dimensions. Although not all dimensions are used at all times, one rule exists that uses seven different input dimensions to identify a bird (i.e., rule 56). On the other hand, another rule (rule 42) only needs two inputs in order to identify the bird correctly. In general, most of the rules need five or six different inputs. Although a specific input dimension is not needed for a certain rule in order to identify a bird, it is possible that the system requires a value for that dimension. Because of this, it makes sense that a test case has defined values (i.e., values that are accepted by the system as an input) for all fourteen input dimensions, before it is presented to the system, even though, these inputs might not all be needed.

The next issue is how many possible values each input dimension can legally accept. An in-depth discussion of this is found in (Michels 1998). However, in summary, the total amount of all possible combinations of all fourteen input dimensions is 541 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 12 * 13 * 5 * 5 * 65 = 35,108,736,000.

From the above number, it is obviously not practically feasible to have an exhaustive set of test cases (more than 35 billion test cases). There has to be a way to reduce this very large number to one that can be adequately handled. The QuEST is an attempt to do just this.

3.2 Errors Seeded

A total of 12 errors were seeded in rules that affect intermediate hypotheses, and an additional 24 were seeded in rules that affect final conclusions. Space limitations

preclude a detailed discussion of these errors. However, these errors generally consisted of altering the premise of a rule by adding, deleting or modifying a premise pattern, adding a new rule or leaving out an existing rule. Only one error was seeded at a time. All these represented differences from the original system, and thus were considered faults. The errors were labeled as *Mod1* through *Mod35*, where, of course, *Mod* stands for *modification.* Each mod represents one error seeded into an otherwise correct version of the Ornithologist system. Additionally, some of the modifications contained a slight variation, and these were labeled with an "a" next to the number. These modifications are completely described in (Michels 1998).

3.3 The Quasi-Exhaustive Set of Test Cases

The approach of how to generate such a QuEST is described in detail in (Abel, Knauf and Gonzalez 1996). Below is a summarization of this procedure:

1. Break down the range of an input into sub-ranges where its values are considered to be equivalent in terms of its effects on the final conclusions,
2. Compute an initial set of potential test cases, P, based upon combinations of values within these sub ranges,
3. Sort these cases into several sets P_i having very distinctive features, and,
4. Filter all P_i by eliminating those that are subsumed by others.

The QuEST is the union of all sets of positive test cases that imply certain hypotheses. The set of negative test cases is called P0. A 'negative test case' is one whose inputs are such that the hypothesis that it is designed to conclude is not positively concluded. For example, if a rule is intended to detect a faulty component, a negative test case for that conclusion is one whose input values do not conclude that the component is faulty.

A tool called TC-Gen was used to generate the QuEST automatically using the technique described in (Abel, Knauf and Gonzalez 1996]. TC-Gen uses the structure of the rule base and the legal values of the inputs to determine the meaningful test cases. If one conclusion or intermediate hypothesis does not make use of one particular input, it simply uses that input's declared *default* value. This is an artificial means of allowing all the test cases to have equally sized input vectors. The default values are determined by TC-Gen, and not by the expert system or its developer.

The QuEST for the Ornithologist (original) consisted of 317 test cases and the set P0 of 1063 test cases. Compared to the 35,108,736,000 test cases in the EST, the 1480 test cases of both the QuEST and P0 represent a significant a huge reduction.

One of the goals of this investigation was to determine whether the QuEST generated by the TC-Gen is really quasi-exhaustive. That is, does the QuEST contain test cases that will detect an error in an XPS if there is one? The definition of the QuEST is that it is functionally equivalent to the EST, so that theoretically, it must detect ALL errors in the knowledge base.

Whenever the original Ornithologist and the altered version differed in an output for the same test case input, the output of the original was treated as correct while the output of the other was considered to be incorrect. Whenever such a discrepancy occurred, the test case was inspected to determine whether the discrepancy came about as a result of the seeded error.

As a result of this investigation, it was discovered that the set P0 assumes an important role in the validation process. This will be explained below.

3.4 Errors in Intermediate and Final Hypotheses

An error in an intermediate hypothesis had more consequences on the QuEST than did an error in a final hypothesis. This is because an intermediate hypothesis was used by one or more final hypotheses. However, if one intermediate hypothesis was only used by one other final hypothesis, then this intermediate hypothesis could also be the source of the error (as it happened with the alterations Mod 4a, Mod 5a and Mod 6a).

3.4.1 Detected Errors. The errors in alterations Mod 1 to Mod 8, which introduced errors in rules affecting the intermediate hypotheses, were all detected. This means that there existed test cases in the QuEST that indicated an anomaly. Nevertheless, an anomaly in the process was discovered. There were two final hypotheses for which no test case existed that identified errors. This was interesting, since these final hypotheses used rules that made use of the erroneous intermediate hypothesis. The expectation would have been that since the intermediate hypothesis was incorrect, the final conclusion would also be incorrect. The answer was that these rules used a negation of the intermediate hypothesis in their premises. Since the error seeded caused the affected intermediate hypothesis to be not true for different values than originally planned, this was not readily apparent from the cases in the QuEST. One reason it was discovered is that there were several other final conclusions that used this hypothesis, and not as a negation. What would have happened, however, if there was only one final conclusion that would employ an hypothesis, and that would be as a negation? In order to further investigate this question, the alterations to modifications Mod4, Mod5 and Mod6 were made, resulting in Mod4a, Mod5a, and Mod6a. These seeded errors in intermediate rules whose hypotheses are only used by one other final conclusion as a negation. Theoretically, it would have been found if an EST were employed. Inclusion of the test cases in P0, however, would remedy this discrepancy.

In the errors affecting the final conclusions, alterations Mod 11 to Mod 13, Mod 17 to Mod 19, Mod 21 to Mod 24, Mod 26 to Mod 32, Mod 34 and Mod 35 all were

detected appropriately. That leaves several errors that went undetected. These are discussed below:

3.4.2 Undetected Errors. There were several errors that went undetected. This section will attempt to explain why this happened.

Mod14, 15, 16 and 20 had a similar situation as in Mod4a, 5a, and 6a. This is because while all positive test cases in the original are also positive in the modified system, such is not the case for the negative test cases. Once again, this would be resolved by making use of the cases in P0.

Nevertheless, there were three errors not detected by either the QuEST or P0. These represent potentially more serious situations.

Mod9 represents an intermediate hypothesis that was added to the rule base. This hypothesis is not used by any other rule, so the TC-Gen tool treated it as a final hypothesis, and it generated test cases as if it were in fact able to identify birds. Since it is not, it used all the default values for the rules, and neither the original system, nor the modified one knew of any bird that is described by all the default values. Thus, the original system and the faulty one agreed in that these inputs identify no bird, i.e., "The system was not able to make any conclusions". The question that remains is whether this type of error is realistic, and whether it should have been detected as an incomplete element of the knowledge base during the verification process.

Mod33 removes a rule from the rule base. Since the methodology for developing the QuEST uses the systems own rule base structure to generate the test cases, it cannot detect an error by omission. Thus, such errors are not detectable by using the QuEST or P0.

Mod25, the last remaining unexplained undetected error results from the deletion of a pattern in a rule's premise that references an input dimension. Since TC-Gen puts the default value in the test case input vector if the input is not being used by the test case, if the default value happens to agree with the value of the input had the input been used, the error will not be detected. This is exactly what happened in Mod25. Of course, this could be solved by changing the way in which TC-Gen generates the test cases.

4. Summary and Conclusion

This investigation provided an opportunity to implement the concept of a Quasi-exhaustive set of test cases, and to evaluate whether it is in reality equivalent to the exhaustive set of test cases. An expert system consisting of 71 rules called the Ornithologist was used as the test bed. It was modified several times, each version introducing one and only one error. There were several types of errors introduced. Then the QuEST for the modified version was generated. These test cases were presented to the original system as well as to the modified version to look for differences between their solutions. If there were any differences, and the difference could be traced to the seeded error, then it was determined that the QuEST was successful in detecting that error. If an error produced no differences in output for all the test cases in the QuEST, then that error was said to be undetected. Errors in rules that lead to intermediate hypotheses as well as to final conclusions were included. In all, 36 errors were introduced.

Of the 36 errors introduced, 10 went undetected. However, a closer examination of the cause of the undetected errors provided a valuable insight into the workings of expert systems. Furthermore, 9 of the 10 undetected errors could be easily detected by making some rather simple changes to the QuEST generation procedure. The last one represented an error of omission, and therefore not detectable. However, it is doubtful that the EST would also be able to detect such errors of omission. The effect of that is that the number of test cases in the QuEST is increase somewhat as a result of including the set P0 as part of the QuEST. This leaves it up to the ReST to further reduce the cardinality of the test case set to a point that it is practical.

Thirty-six cases do not represent a large number of errors n order to declare success or failure. However, these cases represent the different types of errors that could exist in an expert system. Therefore, the authors consider the QuEST to be in fact functionally equivalent to the EST if the recommendations included herein are implemented.

Further research in this topic, of course, centers on the ReST, and how to reduce that number.

5. References

Abel, T., Gonzalez, A. J., 1997, "Utilizing Criteria to Reduce a Set of Test Cases for Expert System Validation", Proceedings of the Tenth Florida Artificial Intelligence Research Symposium (FLAIRS-97), Daytona Beach, FL., USA, pp. 402-406.

Abel, T., Knauf, R., Gonzalez, A. J., 1996, "Generation of a Minimal Set of Test Cases that is Functionally Equivalent to an Exhaustive Set, for Use in Knowledge-Based System Validation", Proceedings of the Ninth Florida Artificial Intelligence Research Symposium (FLAIRS'96), Key West, FL., USA, pp. 280 - 284.

Herrmann, J., Jantke, K. P., Knauf, R., 1997, "Towards Using Structural Knowledge for System Validation", Technical Report Meme-IMP-2/1997, Hokkaido University Sapporo, Meme Media Laboratory, February/March 1997.

Michels, J. E., 1998, "Validating a Validation Tool – Experiences with a Test Case Providing Strategy", Undergraduate Practicum, Technical University of Ilmenau and University of Central Florida, January 1998.