Refinement in Response to Validation

Susan Craw
School of Computer & Mathematical Sciences,
The Robert Gordon University, St Andrew Street,
Aberdeen AB1 1HG, Scotland, UK

smc@csd.abdn.ac.uk
Tel: +44 224 262715
FAX: +44 224 262727

D. Sleeman
Department of Computing Science,
University of Aberdeen, Elphinstone Road
Aberdeen AB9 2UE, Scotland, UK

sleeman@csd.abdn.ac.uk
Tel: +44 224 272288
FAX: +44 224 273422

Expert Systems with Applications, 8(3), 1995.

Also appears in J. Cardeñosa and P. Meseguer, editors, *Proceedings of the EUROVAV93 Workshop*, pages 85–99, Palma, SPAIN, 1993.

Abstract

Knowledge based systems (KBS) are being applied in ever increasing numbers. In parallel with the development of knowledge acquisition tools, is the demand for mechanisms to assure their quality, particularly in **safety critical applications**. Quality assurance is achieved by checking the contents of the KBS at various stages throughout its life cycle. But how does testing for quality assurance aggravate the already well-known knowledge acquisition bottleneck? The partial automation of checking and correcting the Knowledge Base (KB) is an obvious approach to reducing the bottleneck, but also a more routine treatment of checking will provide improved facilities for quality assurance. In addition to **identifying** the occurrence of faults, this paper suggests that **responding** to faults identified by validation is both useful and important. Therefore, **refinement** should be thought of as a companion to validation.

1 Introduction

This backdrop has seen the recent emergence of **validation** as a theme in main stream AI. However, the field has concentrated on the **identification** of faults. We wish to suggest that **responding** to faults identified by validation is both useful and important, and so **refinement** should be thought of as a companion to validation. On completion of a validation step, refinement should be executed with the benefit of the information assembled by validation.

This paper advocates the inclusion of refinement within a validation toolbox and describes the role of KRUST as an improvement tool within the ViVa project (Esprit P6125). For the rest of this paper we shall use the term **validation+** to refer to a process which combines refinement with validation; **validation** will not include the notion of refinement. The body of the paper is divided into 3 sections: Section 2 describes various refinement systems but focuses on our KRUST system, Section 3 looks at validation and Section 4 discusses the integration of validation and refinement with reference to an Esprit project dedicated to validation+. The final section summarises our ideas on the integration of validation and refinement.

2 Refinement

A KBS consists of a body of knowledge and a means of making deductions. In a real application, it is relatively easy to select an appropriate problem-solving strategy, but it is most unusual for the knowledge to be perfect. It is common for the knowledge to contain inconsistencies, be incomplete or perform badly on examples, and many of these faults must be removed before the KBS is of an acceptable quality. In this case, it is often assumed that the **procedural** component of the KBS is suitable, and refinement of the **KB** is the established technique to correct a KBS. As an exception, CONKRET (Lopez, 1991) does refine the control mechanism, because control is represented as **declarative** meta-**knowledge**.

A refinement system responds to the existence of evidence suggesting the need for change. Since many faults rely on the way that deductions are made from the knowledge, it is common for a refinement system to have some knowledge about the procedural mechanism so that it can explore causes of faults and possible solutions, but this clearly depends on the form of evidence given to the refinement system. So what types of evidence can be provided? Certainly, if the faulty knowledge is pinpointed exactly, (e.g. Rules R_i, R_j, R_k form a loop) then refinement can implement changes to this knowledge. However often the **effects** of the faults are identified and the major effort within refinement is precisely: "identifying exactly what should be changed, and how". Thus, the evidence may not refer directly to the KB; instead there may be an example which the KBS wrongly solves, or fails to solve. Such evidence may be quite rich; if it is accompanied by the expert's solution, a criticism of the KBS's solution, etc.

2.1 Traditional Approaches

The classic refinement system, TEIRESIAS (Davis, 1984), has many similarities with more recent projects, such as SEEK (Politakis, 1985), Dipmeter Adviser Refiner (Smith, Winston, Mitchell, & Buchanan, 1985), ODYSSEUS (Wilkins, 1988), namely:

- wrongly solved examples trigger refinement;
- the KB is rule-based; and
- knowledge of the problem-solving method identifies causes of faults.

With the exception of ODYSSEUS, these refinement systems choose to refine the KB in **one** particular way and then make only these changes to the KB. In ODYSSEUS, problem-solving meta-knowledge identifies faults in the KB; any meta-rule which fails to fire indicates a possible fault in the domain knowledge. This approach may produce too many false alarms, so ODYSSEUS needs a way of separating true from false alarms and controlling the number of possible changes in reaction to an alarm, but this has not been described.

Refinement systems may use heuristics to focus attention on most likely causes of error and so restrict the search for possible changes to particular areas of the KB. However, an extensive supply of meta-knowledge, as required by Smith et al. (Smith et al., 1985), has the same, or worse, knowledge acquisition overhead as domain knowledge. In contrast, background knowledge, essential for refinement (and validation) systems, is more easily provided, being attribute types, value hierarchies, etc. It links existing faulty knowledge to appropriate replacement knowledge.

Traditionally, refinement has been identified as a subfield of Knowledge Acquisition; or Machine Learning if refinements are automatically proposed and only **sanctioned** by a human. Refinement is often a distinct step in the final phases of knowledge acquisition; most of the KB is acceptable, but small changes are made to the content of the knowledge (not the structure) so that unwanted behaviour does not re-occur. Here we argue that refinement is a natural **extension** to validation systems, and may even be considered as a **collaborating** system, which gains from the analysis undertaken during fault-finding. Since validation should be an ongoing process throughout the life cycle, refinement too should be used at all stages of the KBS development.

2.2 KRUST

Our interest in validation+ stems from our work in knowledge refinement; we have previously implemented a refinement system, KRUST (Craw & Sleeman, 1990). In common with other refinement systems, it refines rule-based KBSs and the process is triggered when the KBS fails to solve an example correctly. In contrast to other systems, however, KRUST considers **many** suggested refinements, but executes only a small subset of the refinements on the KB, thus proposing a small number of replacement KBs. It filters the refinements by setting increasingly selective tests which the refinements must pass. As the refinements reduce in number, the tests can be computationally more complex. In this way KRUST only rejects a proposed refinement once evidence against it has been found; i.e. it fails one of the tests.

KRUST's architecture is shown in Figure 1. A training example, which the KBS fails to solve correctly, is input and KRUST enters an interpretation phase. Possible causes of the failure are identified and the rules are classified into different categories of interest, from which suitable refinements are suggested. The refinement generation step gathers a set of possible refinements, each consisting of a set of rule changes; e.g. a named rule should be strengthened (made more difficult to satisfy), weakened (made easier to satisfy), given an increased chance of firing, etc. At this stage, the actual changes

Figure 1: **KRUST's Architecture**

are not specified; a refinement contains the **aims** of the changes. The refinements now meet the first filter; meta-knowledge and heuristics discard those proposed refinements which are believed to be poor. Only after this first selection process are the refinements executed on the KB, thus creating a set of refined KBs. At this stage, it is possible, and feasible, to run the refined KBs on sample tasks and compare the results with expert solutions. The second filter applies each refined KB to the training example and a set of tasks which **must** be solved correctly; we call these tasks "**chestnuts**". Any refined KB which fails a test is rejected. All refined KBs which pass these tests can be suggested to

the expert, or alternatively, as in the existing system, a fairly detailed judgement phase ranks the refined KBs and selects the most suitable one. A fuller description of KRUST's architecture and its application to a wine KB is presented in (Craw & Sleeman, 1990).

An effect of KRUST's approach of generating many refinements and testing their suitability, is that the tests available to KRUST can be varied, to allow KBs with different features to succeed. We have experimented by comparing the following two scenarios (Craw & Sleeman, 1991):

Black Box Testing: The refined KB must solve the training example correctly but one does not care how the KB has been changed. KRUST was applied to a KB where the tests were biased towards ones which the original KB would pass. This corresponds to a realistic use of KRUST; the KB is faulty, but not very!

Glass Box Testing: The original KB must be retrieved after an intentional corruption. KRUST was applied to the artificially corrupted KB, but now, the tests were not favourable to **this (the corrupted)** KB, but to the KB **before** the corruption, because we wanted KRUST to retrieve the original situation from the corruption, and so wanted KRUST to favour refined KBs which were most like the original KB, before the corruption.

Changes in Best Refined KB	Black	Black-1	Glass
Weaken 1 Rule only	1%	2%	0%
Strengthen at least 1 Rule only	51%	54%	20%
Weaken 1 Rule and Strengthen at least 1 Rule	3%	——	0%
Lower the Certainty of at least 1 Rule	0%	0%	0%
Raise the Certainty of 1 Rule	0%	0%	0%
Weaken 1 Rule and Lower the Certainty of at least 1 rule	0%	——	0%
Weaken 1 Rule and Raise its Certainty	3%	——	0%
Strengthen at least 1 Rule and Lower the Certainty of at least 1 rule	7%	8%	0%
Change the Conclusion of 1 Rule	9%	10%	20%
Add a New Rule	25%	27%	60%
Number of Best Refined KBs	67	63	5

Table 1: **Changes in Best Refined KB**

We compared the types of changes made during the refinement process in both trials and found that KRUST's behaviour was distinctive, despite the small number of runs in the glass box testing. Table 1 shows the type of change in the refined KB selected as best. The columns contain the following data:

Black: the results from black box testing;

Black-1: the results from black box testing where only refined KBs with single changes were counted[1]; and

Glass: the results from glass box testing.

By comparing the distribution of changes it would appear that strengthening a rule is much less favoured in glass box testing, whereas inserting a new rule is much more effective. We should like to investigate this testing further, to ensure that these results are echoed in more extensive studies with glass box testing.

KRUST proved to be quite sensitive to the tests with which it was provided. In the glass box testing above, KRUST was given tests which favoured the original KB and the refinement corresponding to the original KB survived all filtering processes. It was not necessarily chosen as the best KB, because some other refined KBs might out-perform it in the final ranking, for the particular test examples. (This is quite understandable, since the original KB might not be the optimal KB for these test examples.) In earlier glass box testing, KRUST had been given tests favouring the input KB; i.e. the corrupted KB, and although the refinement recreating the original KB had always been generated, it was always rejected during the test case filter.

We shall argue later that the flexibility provided by KRUST's speculative refinement generation and tailorable rejection of refinements makes it particularly suitable for integration with fault-finding systems.

3 Validation

The validation of KBSs does not conform to the standard definitions of Verification and Validation for conventional software, because the specifications for KBSs are fundamentally different from those for other software, primarily because of their great reliance on knowledge. The knowledge within a KBS must be validated as part of quality assurance, but it requires different sorts of specifications and special checking techniques. We are interested in the validation of the **declarative** parts of a KBS; i.e. the KB. This links with our interest in the refinement of the declarative knowledge. Although the validation is aimed at the declarative knowledge, we often need to explore the effects of the inference engine, because potentially faulty knowledge may not produce faults when used in conjunction with the inference engine. We note that the procedural components of a KBS **can** be specified and validated using techniques for conventional software, and is not addressed here.

3.1 Terminology

There is much debate about the terminology in knowledge validation since it does not fit the conventional model. The debate centres on the relationship between the type of specification being checked and the terminology given to the process. We shall adopt the position of Laurent (Laurent, 1992); namely:

Validation is the general process of checking that a KBS satisfies its specifications and is composed of:

[1]This selects those runs which were similar to that used for black box testing; i.e. where only a single corruption (and hence a single correction) was introduced.

- the **Verification** of formal specifications; and
- the **Evaluation** of pseudo-formal specifications.

We use the general term validation, without the need to say whether the tool performs verification or evaluation, because, from our perspective of refinement, we are interested in the **output** of validation tools, rather than the type of specifications they check. The only possible effect for refinement, of distinguishing between verification and evaluation, is that an evaluation anomaly may not be sufficiently well-founded to justify a refinement.

3.2 Consistency Checking

Consistency checking looks for contradictions in the knowledge and is based on a set of specifications defined within a consistency model. If the KBS satisfies all consistency specifications, then it is declared to be consistent. In practice, consistency checkers operate by looking for **in**consistencies! The consistency model contains formal specifications and so strictly these are verification tools. The following descriptions could be converted into formal specifications for a consistency model:

- the required arity of attributes - e.g. the arity of parents is 2;
- contradictory values for attributes which must not occur together - e.g. the apocryphal pregnant male
- no rule is redundant - a rule is redundant if its conditions imply the conditions of another with the same conclusion;
- no rules are ambivalent - two rules are ambivalent if the conditions of one imply the conditions of another and they contain contradictory conclusions; e.g. one rule deduces that the patient is male, the other deduces "he" is pregnant.
- no looping rules.

There are two approaches to consistency checking:

Static Analysis: considers only the knowledge explicitly represented in the KB and identifies the items of knowledge which contradict a specification; e.g. an attribute arity is wrong, contradictory values appear within a rule, redundant rules (the number of conditions determines the strength of conditions), ambivalent rules (subsets of conditions provide simultaneous firing), looping rules (but these may fit more neatly in the next category), etc. The output from static analysis contains explicit items of knowledge which contradict a particular specification.

Dynamic Analysis: explores deductions within the KB which contradict a specification; e.g. a deduction is false, contradictory values appear in a deduction, redundant and ambivalent rules (allowing deduction may increase the number of rule interactions), looping rules (these are more appropriate here where chains of rules are automatically explored), etc. Dynamic analysis is computationally expensive because it involves the exhaustive search of all possible deductions. In circumstances where this is not feasible, heuristics may guide the search for likely inconsistencies. Since dynamic analysis is deduction based, the anomalies are expressed in terms of examples or traces.

Static consistency checking can often be achieved exhaustively as in ONCOCIN's checker (Suwa, Scott, & Shortliffe, 1984). Although COVADIS (Rousset, 1988) and COVER (Preece & Shinghal, 1992) are exhaustive dynamic checkers, SACCO (Ayel & Laurent, 1991) relies on heuristic guidance to avoid the computational complexity of dynamic checking.

Clearly exhaustive checking produces a complete set of anomalies with respect to the specifications, whereas heuristic checking is unlikely to find **all** anomalies. We also note that the anomalies found are only **possible** faults; they logically follow from the knowledge but they may not occur in practice, if the KBS has a restrictive inference engine. The anomalies must be checked under the inference engine, to determine whether they are actual faults. Alternatively, the inference engine can be taken into account when defining the specifications.

3.3 Completeness Checking

Completeness checking is closely related to knowledge acquisition, but the term implies that the process occurs towards the end of a particular acquisition phase; one believes that the knowledge is complete, but fears that small items of knowledge are missing. The repertory grids of AQUINAS (Boose, 1988) display existing knowledge in a structured, but domain independent way, and allow the user to inspect the knowledge and thus volunteer additional knowledge. OPAL (Musen, Fagan, Combs, & Shortliffe, 1987) is designed specifically for the cancer therapy domain and displays its knowledge in a standard cancer therapy format, familiar to its users. These systems cannot be considered to **automatically** discover missing knowledge. In contrast, ONCOCIN's checker (Suwa et al., 1984) is an automated static completeness (as well as consistency) checker, and MOLE and SALT, described in (Marcus, 1988), use knowledge about problem solving to identify knowledge which is believed to be relevant.

Completeness checkers inspect the existing knowledge to identify gaps by predicting a case which cannot be solved or identifying a set of related rules which exclude some circumstance.

3.4 Testing

Testing Tools judge the quality of the KBS by running it on examples. Testing tools, such as VORTEX (Cross & Grisoni, 1990), may record the effects of testing by documenting traces or providing a statistical summary of rule usage. Another function of testing tools is to record any examples which the KBS cannot solve; this provides evidence of incompleteness.

Of course testing depends heavily on the relevance, importance or even criticality of the examples. SYCOJET (Ayel & Laurent, 1991) approaches testing from a different angle; it generates **pertinent** examples by exploring possible deductive chains within the KB.

3.5 Validation Knowledge

The validation tools described above are knowledge-based, and in particular, the specifications may require knowledge about the attributes; e.g. value types, value hierarchies,

arity, etc. Also, non-exhaustive validation requires heuristics to guide the search for anomalies; e.g. borderline cases may provide a source of likely faults.

4 Validation+

Validation+ integrates Refinement and Validation. Instead of requiring the expert to supply task-solution pairs, the refinements are based on the evidence provided by the validation tools. Towards the end of the development of a KBS, the emphasis may be on demonstrating that the KBS is **correct** (faults are not expected), however validation should be an **ongoing** process throughout development, and assistance to correct faults, as they appear, is a natural extension to finding the faults in the first place. Although the VALID project (Esprit P2148) produced a validation+ toolbox offering a range of tools, the tools did not appear to be closely integrated. VORTEX (Cross & Grisoni, 1990) contains a set of integrated tools but is dedicated to a **single KB**. Here we describe an integrated toolbox whose aim is to validate+ a range of KBSs.

4.1 Integrating KRUST

We are interested in developing KRUST so that it benefits from the additional evidence which can be provided by validation, to re-use searches required during validation and to judge the quality of refined KBs using validation. We first consider the types of additional evidence.

In Figure 2, we have superimposed some validation tools on KRUST's architecture. The arrows show the flow of evidence to KRUST, their labels indicating the type of evidence. We have separated test sample generation from general testing tools since these have a special use within KRUST: providing task-solution pairs for choosing the best refined KB, and possibly chestnut cases for filtering.

Many dynamic validation tools explore deduction chains within the KB, a type of search conducted by KRUST. The validation tools can pass the results of these efforts[2] directly to KRUST, which then removes those deductions not produced by the inference engine, as another filtering process. Thus, the search effort within the integrated tools is not duplicated. Figure 2 also shows the links to KRUST from the Validation Knowledge, described in Section 3.5. It is a source of heuristics for refinement filtering and relevant replacement knowledge for knowledge updates.

KRUST adopts the generate and test model and thus benefits from the availability of rejection criteria. In Figure 2, the Completeness, Consistency and Testing tools have been enclosed in a box whose shading matches the second filter in KRUST, for which these validation tools can provide tests; e.g. consistency checking rejects KBs containing contradictory values. In particular, incremental verification (Meseguer, 1992) would explore faults introduced by changes. Testing tools provide an empirical checking as is used currently in KRUST. Another aspect of validation is the development of KB metrics, available to finally rank KRUST's remaining refined KBs, and select the best to present to the user.

[2]The search within SACCO (Ayel & Laurent, 1991) and Meseguer's incremental verifier (Meseguer, 1992) provides extended labels for the conclusions within the KB.

Figure 2: **KRUST's Integration within Validation+**

4.2 The ViVa Project

The ViVa project (Esprit P6125)[3] addresses the issue of Validation+. Its aim is exemplified by its title, "**Verification, Improvement & VAlidation of Knowledge Based Systems**" with the following (shortened) descriptions from the Technical Annexe:

Verification denotes the process of determining whether the outcome of a given phase of KBS development meets all the requirements established during the previous phase. This corresponds to the question of whether the KBS is built the right way.

Improvement denotes the process which interprets the discrepancies and anomalies identified by the verification and validation processes and which supports the consequent refinement of the KBS.

[3]The partners in this project are Computer Resources International A/S (DK), CISI Ingenierie (F), European Space Agency (Int), Lloyd's Register of Shipping (UK), Logica Cambridge Ltd. (UK), University of Aberdeen (UK) and Université de Savoie (F).

Validation denotes the process of evaluating a KBS to ensure compliance with the requirements. This corresponds to the question of whether the KBS is the right system.

In addition to providing an integrated toolbox of Validation+ tools, ViVa will define a methodology for using the tools. An important criterion for validation+ tools, and knowledge acquisition tools in general, is their suitability for real-life KBSs. Access to real-life KBSs is often difficult for researchers, but the ViVa project is applications-driven, and will provide both KBSs and descriptions of the validation+ needs of KBS developers.

4.3 ViVa Tools

Improvement, as defined above, implies the two stages: interpretation of faults and implementation of changes. This coincides with our notion of refinement, whose major task is identifying the cause of "failure"; making the changes is normally a relatively easy exercise. The two stages of interpretation and implementation were indicated in Figure 1 by shading. The other ViVa tools cover the types of validation described in Section 3 and satisfy the Verification and Validation objectives of the project. Figure 2 thus represents the functionality of a KRUST-based improvement tool within ViVa.

KRUST is currently a prototype refinement system, which acts on a restricted set of KBSs, in response to the evidence provided by wrongly solved cases. Within the ViVa project, KRUST will be developed to refine a wider range of rule-based systems and other knowledge representations, and to respond to a greater variety of fault-indicating evidence, produced by validation tools.

4.4 Knowledge Representation

Validation+ systems must have access to the KBS, but this may restrict target KBSs to:

- one particular KBS, e.g. TEIRESIAS (Davis, 1984) refines MYCIN;
- one particular shell or model, e.g. SEEK (Politakis, 1985) refines KBSs built in EXPERT; or
- one standard representation into which the KBS must be translated, e.g. COVER (Preece & Shinghal, 1992) validates a Prolog representation of the KBS.

Clearly the last option is the most general but it does demand that translation occurs at some stage; either before the validation+ tool is used, or as part of its operation.

It is common for an integrated toolbox to adopt the last option and have a Common Knowledge Representation, tailored to the tools, so that all the necessary hooks required by the tools are easily available. ViVa has chosen this approach. Applications must be converted to this formalism, but translators from standard shells should be straightforward.

5 Summary

This paper has described how refinement can enhance the functionality of validation systems. Instead of simply identifying possible faults, these are explored further, so that

anomalies can be rectified by actually refining the KB in response to the evidence that faults have been found. We believe that KRUST is a suitable refinement system for integration with various validation systems because its behaviour can be moulded to suit the available evidence. It generates many refinements to remove a particular anomaly, but it can be influenced when choosing which of these refinements to reject. Thus it is sufficiently flexible to adapt to the various types of change which may be required.

The aim of a general validation+ approach is to ease the process of validation. By incorporating a wide-ranging validation+ toolbox, within a structured knowledge acquisition methodology, it is hoped that ViVa can provide validation+ at all stages of the life cycle. Of course, we are interested in **improvement** throughout the life cycle and would anticipate that easing the validation burden by additionally suggesting **how** to fix anomalies, may make validation, in the form of validation+, a more attractive undertaking. Currently, KBS developers fear using validation because of the anomalies it may uncover; indicating how repairs may be made should lessen this fear.

6 Acknowledgements

We thank our ViVa project partners for useful discussions on validation and its application.

7 References

Ayel, M., & Laurent, J.-P. (1991). SACCO-SYCOJET: Two different ways of verifying knowledge-based systems. In Ayel, M., & Laurent, J.-P. (Eds.), *Validation, Verification and Test of Knowledge-Based Systems*, pp. 63–76. Wiley.

Boose, J. H. (1988). Uses of repertory grid-centred knowledge acquisition tools for knowledge-based systems. *International Journal of Man-Machine Studies*, *29*, 287–310.

Craw, S., & Sleeman, D. (1990). Automating the refinement of knowledge-based systems. In Aiello, L. C. (Ed.), *Proceedings of the ECAI90 Conference*, pp. 167–172 Stockholm, SWEDEN. Pitman.

Craw, S., & Sleeman, D. (1991). The flexibility of speculative refinement. In Birnbaum, L., & Collins, G. (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop on Machine Learning*, pp. 28–32 Evanston, IL. Morgan Kaufmann.

Cross, S., & Grisoni, M. (1990). A methodology for producing validated real-time expert systems. In *Proceedings of the Advisory Group for Aerospace Research and Development: Knowledge Based System Applications for Guidance and Control, AGARD-CP-474*, pp. 27–30.

Davis, R. (1984). Interactive transfer of expertise. In Buchanan, B., & Shortliffe, E. H. (Eds.), *Rule-Based Expert Systems*, pp. 171–205. Addison-Wesley, Reading, MA.

Laurent, J.-P. (1992). Proposals for a valid terminology in KBS validation. In Neumann, B. (Ed.), *Proceedings of the ECAI92 Conference*, pp. 829–834 Vienna, AUSTRIA. Wiley.

Lopez, B. (1991). CONKRET: A control knowledge refinement tool. In Ayel, M., & Laurent, J.-P. (Eds.), *Validation, Verification and Test of Knowledge-Based Systems*, pp. 191–206. Wiley.

Marcus, S. (Ed.). (1988). *Automating Knowledge Acquisition for Expert Systems*. Kluwer, Boston.

Meseguer, P. (1992). Incremental verification of rule-based expert systems. In Neumann, B. (Ed.), *Proceedings of the ECAI92 Conference*, pp. 840–844 Vienna, AUSTRIA. Wiley.

Musen, M. A., Fagan, L. M., Combs, D. M., & Shortliffe, E. H. (1987). Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies*, *26*, 105–121.

Politakis, P. G. (1985). *Empirical Analysis for Expert Systems*. Research Notes in Artificial Intelligence. Pitman, London.

Preece, A. D., & Shinghal, R. (1992). Validating knowledge bases by anomaly detection: An experience report. In Neumann, B. (Ed.), *Proceedings of the ECAI92 Conference*, pp. 835–839 Vienna, AUSTRIA. Wiley.

Rousset, M. C. (1988). On the consistency of knowledge bases:the COVADIS system. In *Proceedings of the ECAI88 Conference*, pp. 79–84 München, GERMANY.

Smith, R. G., Winston, H. A., Mitchell, T. M., & Buchanan, B. G. (1985). Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the Ninth IJCAI Conference*, pp. 367–374 Los Angeles, CA.

Suwa, M., Scott, A. C., & Shortliffe, E. H. (1984). Completeness and consistency in a rule-based system. In Buchanan, B. G., & Shortliffe, E. H. (Eds.), *Rule-Based Expert Systems*, pp. 159–170. Addison-Wesley, Reading, MA.

Wilkins, D. C. (1988). Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 646–651 Minneapolis, MN.