Appears in *International Journal of Human-Computer Studies*, 44:245–256, 1996

Refinement Complements Verification and Validation

Susan Craw
School of Computer and Mathematical Sciences
The Robert Gordon University
Aberdeen, Scotland

Abstract

Knowledge based systems are being applied in ever increasing numbers. The development of knowledge acquisition tools has eased the "Knowledge Acquisition Bottleneck". More recently there has been a demand for mechanisms to assure the quality of knowledge based systems. Checking the contents of the knowledge base and the performance of the knowledge based system at various stages throughout its life cycle is an important component of quality assurance. Hence, the demand now is for verification and validation tools. However, traditionally, verification and validation have identified possible faults in the knowledge base. In contrast, this paper advocates the use of knowledge refinement to correct identified faults in parallel with the ongoing verification and validation, thus easing the progress towards correct knowledge based systems. An automated refinement tool is described which uses the output from verification and validation tools to assemble evidence from which the refinement process can propose repairs. It is hoped that automated refinement in parallel with validation and verification may ease the "Knowledge V&V Bottleneck".

1 Introduction

This demand for quality assurance has seen the emergence of **verification** and **validation** (V&V) as a theme in main stream AI. Instead of concentrating on the **identification** of faults, this paper suggests that **responding** to faults identified by V&V is both useful and important, and so **refinement** should be thought of as complementary to V&V. When a V&V phase has been completed, a refinement phase should be executed, with the benefit of the information assembled by V&V.

But how does V&V aggravate the already well-known knowledge acquisition bottleneck? Carbonell conducted a study of the distribution of knowledge engineering effort during Knowledge Based System (KBS) development (Carbonell 1991), and found that the reorganisation and correcting of knowledge is the most time consuming phase. Thus, the partial automation of checking and correcting is an obvious approach to reducing the conventional bottleneck, but also a more routine, and easier, treatment of checking will provide improved facilities for assuring quality.

Section 2 contains a description of the process of refinement and some research in this area. Section 3 focuses on the unusual features of our automated refinement tool, KRUST. Section 4 describes the evidence of faults found by V&V tools and Section 5 discusses how KRUST can utilise their output. The benefits of refining in parallel with V&V is summarised in Section 6.

2 Knowledge Refinement

We first consider refinement, since this is the focus of our interest, leaving a description of V&V until Section 4. A KBS consists of a knowledge base (KB) of static knowledge and a control mechanism to make deductions. The type of application often suggests a suitable problem solving strategy; e.g. heuristic classification for diagnosis problems. In contrast, it is common for the acquired knowledge to contain faults; e.g. the KB is inconsistent, the KB is incomplete, the KBS performs badly on examples. Many of the identified faults must be removed before the KBS is acceptable; i.e. the **KB** must be refined. Notice that although the control mechanism will not be altered, its effect must be considered; e.g. altering certainty factors in the KB can affect deductions.

Traditionally, refinement has been identified as a subfield of Knowledge Acquisition. Early in the KBS development, knowledge acquisition consists of new knowledge being assembled and integrated in the KB. However, as the KB evolves, it is often more appropriate to change existing knowledge rather than always acquire new knowledge. This process of altering knowledge, and possibly incorporating new knowledge, is known as knowledge refinement and is often a distinct step in the final phases of development; 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. As refinement generation becomes more automated it becomes an emerging field in Machine Learning, where it is often referred to as theory revision.

A refinement system responds to the existence of evidence suggesting the need for change. So what types of evidence can be provided? Traditionally the evidence takes the form of examples provided by the expert. Here, it is suggested that the faults identified by V&V tools are also suitable triggers for refinement. In many cases, evidence consists of the **effects** of the faults and the major effort within refinement is precisely: "identifying exactly what should be changed, and how". Here it is argued that knowledge refinement is a natural **extension** to V&V systems, and may even be considered as a **collaborating** system, which gains from the analysis undertaken during fault-finding. Thus refinement, also, becomes an ongoing process throughout the life cycle.

Refinement techniques developed within the knowledge acquisition community have focused on updating the knowledge in a fairly mature system and many systems have been designed round particular KBSs and shells. The early classic refinement system, TEIRESIAS (Davis 1984), interacted extensively with the expert, assisting him to identify and correct the fault(s) in the Mycin KB. More recent systems reduced the interaction with the expert by incorporating learning techniques where the tool suggests repairs for the user to sanction. SEEK (Ginsberg 1988, Politakis 1985) chooses the refinement that repairs the most faults in a set of training examples. The refinement system for Dipmeter Adviser relies on an extensive supply of meta-knowledge to provide causal evidence for the failure of a case and thus to suggest updates. SALT and MOLE, both (Marcus 1988), and ODYSSEUS (Wilkins 1988) exploit an explicit representation of the problem solving strategy to identify missing or faulty domain knowledge.

The evolution of theory revision systems has been more directly related to inductive techniques and these systems often rely on the quality of their inductive learners to learn **appropriate** new knowledge. Thus EITHER (Ourston & Mooney 1994), FORTE (Richards & Mooney 1991) and AUDREY-II (Wogulis & Pazzani 1993) get good results despite restricting their changes to deleting a rule, inserting a rule, deleting a condition or inserting a condition. Multi-strategy revision systems apply several learning techniques to a range of knowledge sources, including the expert, in the process of repairing knowledge. WHY (Saitta, Botta & Neri 1993) revises a theory by learning from both a causal model and examples; MOBAL (Morik, Wrobel, Kietz & Emde 1993) contains many knowledge acquisition tools with access to a wide range of knowledge sources; and CLINT (De Raedt 1992) combines inductive learning with user interaction.

Many of these knowledge refinement and theory revision tools lack the flexibility to adapt to novel situations; e.g. react to the evidence provided by V&V tools, able to cope without many training examples.

Ripple down rules (RDR) (Compton, Edwards et al. 1992) is a knowledge representation which can be exploited to allow knowledge acquisition during maintenance (Kang, Gambetta & Compton 1994). In this case faults in the KB are fixed by always adding new knowledge to the RDR structure. One advantage of this approach is that incremental V&V is incorporated within the acquisition process. Disadvantages include the fact that the method relies on this particular knowledge representation and that refinements are achieved only by acquiring new knowledge to add to the KB. However, RDR allow knowledge acquisition and V&V to progress together in a maintenance role.

3 KRUST

We now focus on our KRUST refinement system and highlight the features that make it suitable for integration with V&V tools. In common with many other refinement systems, it refines rule-based KBSs on evidence provided by examples which the KBS fails to solve correctly. In contrast to other systems, however, KRUST considers **many** possible refinements to cure a single failure, 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.

3.1 KRUST's Approach

The components of KRUST are illustrated 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 classification rules are grouped into different categories of interest: error-causing rules contain the faulty solution as their conclusion and target rules contain the desired conclusion. The refinement generation step proposes suitable refinements by considering changes that prevent the error-causing rules from firing and allow a target rule to fire instead. A refinement may alter a condition of a target or error-causing rule; i.e. a **single** condition in error-causing rules should be strengthened (made more difficult to satisfy) or **all** unsatisfied conditions in target rules should be weakened (made easier to satisfy) for the training example. Alternatively, these changes to error-causing and target rules may be propagated to rules with these conditions as conclusion. For example, instead of strengthening the condition in an error-causing rule, **all** rules concluding this condition which are fireable are disabled by strengthening any one of their conditions. Similarly, any **one** rule concluding a failed condition for a target rule should be enabled by weakening each of its failed conditions. These changes are propagated throughout the rule structure. In addition to these changes to rule content, KRUST is also able to affect the way that error-causing and target rules are handled by the KBS control: error-causing rules are given a lower priority with respect to the conflict resolution strategy, and target rules have a raised priority. Finally, the conclusion of error-causing rules can be changed and new rules can be added. These refinements have the effect of crippling a faulty solution graph at any level and promoting a potential solution graph which fails to be used. At this stage, an actual change is not specified but contains the **aims** of the change; e.g. weaken the ith condition in rule j and give rule j a higher priority than rule k.

The refinements now meet a filter that discards those refinements that are believed to be poor; e.g. small changes are preferred so refinements that specify more than say 3 changes are rejected, or rule priority should not be changed so only strengthening and weakening refinements are allowed, etc.

Figure 1: **KRUST's Modules**

The refinements are 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. Another filter applies each refined KB to the training example and a set of tasks which **must** be solved correctly. Any refined KB which fails any of these tests is rejected. We note that although each refined KB has been designed in reaction to the training case, rule interaction may mean that a refined KB still does not perform correctly on the training case. All the remaining refined KBs can be suggested to the expert, or alternatively, as in the current system, a fairly detailed evaluation phase ranks the refined KBs and selects the most suitable one. A fuller description of KRUST's architecture and its application to a wine advising KB is presented in (Craw & Sleeman 1990).

3.1.1 KRUST's Refined KBs

An effect of KRUST's generate-and-test approach is that the tests available to KRUST can be varied, to allow refined KBs with different features to succeed. Experiments using a wine-advising KBS are described in (Craw & Sleeman 1991) and have compared KRUST's behaviour in the following two scenarios.

Improvement: This scenario is the normal use of KRUST for maintenance. The current KB should be repaired because a fault has been identified. However, the KB is mature so the repair should be small so that it does not affect already correct cases.The refined KB must solve the training example correctly and the filtering tests are biased towards ones which the current KB being refined would pass. This is the normal use of KRUST – the KB is faulty, but not very!

Recovery: This scenario is an artificial, contrived demonstration of KRUST's abilities. The original KB must be retrieved after an intentional corruption. This scenario is not how KRUST would be used in reality but it demonstrates KRUST's ability to identify and execute a particular change and thus indicates its flexibility. KRUST is applied to the artificially corrupted KB, but now, the tests are not favourable to **this (the corrupted)** KB, but to the KB **before** the corruption. KRUST has to retrieve the original KB from the corruption, and so KRUST should favour refined KBs which are most like the uncorrupted KB.

KRUST was sensitive to the tests with which it was provided. In the recovery scenario testing above, the refinement corresponding to the original KB survived all filtering processes but 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 reasonable, since the original KB might not be the optimal KB for these test examples. In fact if all the examples have the original KB's solution as their solution then KRUST does recommend the original KB as the refined KB.

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

Improvement: the results from 60 test runs in the improvement scenario. These were organised as 4 batches of 15 test runs where the batches were of increasing difficulty; the training cases required more drastic changes.

Recovery: the results from 5 test runs in the recovery scenario. We randomly generated a corruption of each of 5 types: weaken a condition in a classification rule, strengthen a condition in a classification rule, alter the recommended wine in a classification rule, interchange 2 classification rules and strengthen a condition in a lower level rule.

It is interesting to compare the the distribution of changes. In the improvement scenario preventing error-causing rules from firing was a popular refinement. In these cases the correct knowledge already appeared in the KB but the error-causing rule(s) were too general. Inserting a new rule to cover an exceptional example was fairly common. The popularity of these changes suits the maturity of the KB.

Changes in Best Refined KB	KBs with these Changes (%)	
	Improvement	Recovery
Strengthen at least 1 Rule only	51	20
Add a New Rule	25	60
Change the Conclusion of 1 Rule	9	20
Strengthen at least 1 Rule and Lower the Priority of at least 1 Rule	7	——
Weaken 1 Rule and Strengthen at least 1 Rule	3	——
Weaken 1 Rule and Raise its Priority	3	——
Weaken 1 Rule only	1	0
Lower the Priority of at least 1 Rule only	0	0
Raise the Priority of at least 1 Rule only	0	0
Weaken 1 Rule and Lower the Priority of at least 1 Rule	0	——
Number of Best Refined KBs	67	5

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

In the recovery scenario, a completely different pattern of refinements were successful. Inserting a new rule to overcome the corruption was most popular and this reflects the fact that the KB contained one isolated corruption. Thus, the inserted rule was sufficiently specialised to correct the corruption and not affect other examples. We note however that the corruptions were actually evenly spread over the possible corruptions.

KRUST's speculative refinement generation has been flexible enough to consider a range of possible repairs. The rejection of proposed refinements can be tailored to suit the situation; in an improvement scenario refinements which fail on general examples are rejected but in a corruption scenario examples baised towards the uncorrupted KB reject refinements which have added further corruptions. The ability to consider many refinements initially makes KRUST particularly suitable to cope with the output from fault-finding systems.

4 Verification and Validation

It is well known that the V&V of KBSs does not conform to the standard definitions of Verification and Validation for conventional software. The specification for a KBS is fundamentally different from those for other software, because the tasks the KBS must achieve cannot be specified exactly, in some domains it is allowable for the KBS to fail sometimes, and plausibly correct answers are often sufficient. Since the problem solving method and any procedural aspects of the knowledge can be checked by conventional V&V, this paper concentrates on the V&V, and refinement, of the **declarative** parts of a KBS; i.e. the KB. The types of V&V processes will be categorised according to the type of fault they identify.

4.1 Consistency Checking

Consistency checking looks for contradictions which occur in the knowledge, or are deducible from the knowledge. Possible inconsistencies include:

- contradictory values for attributes can be deduced for the same object - e.g. MYCIN's apocryphal pregnant male;
- a rule is redundant - i.e. its conditions imply the conditions of another with the same conclusion;
- a rule cannot be reached - i.e. its conditions are never satisfied;
- two rules are ambivalent - i.e. the conditions of one imply the conditions of the other and they contain contradictory conclusions - e.g. one rule deduces that the patient is male, the other deduces "he" is pregnant.
- a cycle exists - i.e. a set of rules $R_1 \ldots R_n$ exist so that the conclusion of R_i appears in the antecedent of R_{i+1} $\forall i = 1, 2, \ldots n-1$, and the conclusion of R_n appears in the antecedent of R_1
- an attribute has the wrong arity - e.g. parents(abel, cain, adam, eve);

Static consistency checking is concerned with only the knowledge explicitly represented in the KB and can often be achieved exhaustively (e.g. ONCOCIN's checker (Suwa, Scott & Shortliffe 1984)). Although **dynamic** checkers can be exhaustive (e.g. COVADIS (Rousset 1988), COVER (Preece & Shinghal 1992)), they often rely on heuristic guidance to focus on likely cases of inconsistency (e.g. SACCO (Ayel & Laurent 1991)). Clearly exhaustive checking produces a complete set of possible anomalies with respect to the specifications, whereas heuristic checking is unlikely to find **all** anomalies.

Any anomalies found are, however, 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 KBS's inference engine, to determine whether they are **actual** faults. Alternatively, the inference engine can be taken into account when defining consistency.

4.2 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 small items of knowledge may be missing. Completeness checkers have two distinct functionalities:

- User-driven checkers display existing knowledge in a structured, but domain independent way, and allow the user to inspect the knowledge and thus volunteer additional knowledge (e.g. AQUINAS (Boose 1988), OPAL (Musen, Fagan, Combs & Shortliffe 1987)).
- Automatic checkers discover missing knowledge and ask the user to supply knowledge which will fill this gap. They often use knowledge about problem solving to identify knowledge which is believed to be relevant. (e.g. MOLE, SALT (Marcus 1988)).

4.3 Testing

A testing tool measures the quality of the KBS by running it on examples. It can record the effects of testing by documenting traces, providing a statistical summary of rule usage or recording any examples which the KBS cannot solve (e.g. VORTEX (Cross & Grisoni 1990)). Testing depends heavily on the relevance, importance, or even criticality of the examples. SYCOJET (Ayel & Laurent 1991) is a testing tool with a different slant; it generates **pertinent** examples.

5 Integrating V&V and Refinement

We now consider how the tasks undertaken by V&V tools can provide different types of evidence on which refinement tools can base knowledge updating. In addition, the use of V&V tools can enhance the filtering process which is so central to KRUST's success. This discussion is based on KRUST.

In Figure 2, some V&V tools have been superimposed on KRUST's architecture. The arrows show the flow of evidence to KRUST, their labels indicating the type of evidence. We have included a test sample generation tool with the dual possible roles for KRUST of providing evidence of faults and supplying pertinent examples for KRUST's evaluation phase.

5.1 Providing Evidence

We consider each class of tool and describe how the evidence each provides can be used by KRUST.

Figure 2: **KRUST's Evidence from V&V**

From Consistency Checking Tools

If the identified inconsistency does cause an actual fault then it must be removed. KRUST's reaction to each of the inconsistencies discussed in Section 4.1 follows:

Contradictory Values: The expert decides which of these conclusions is correct and KRUST is presented with a training case consisting of the inputs to the KBS, together with the correct and wrong conclusions. For the special cases of **ambivalent rules** and **borderline examples**, similar refinement steps are appropriate.

Redundant Rule: KRUST prevents the more general rule from firing by strengthening its conditions in all possible ways to remove the redundancy with the more specialised rule.

Unreachable Rule: KRUST weakens the rule's conditions so that it can be reached.

Cycling Rules: KRUST breaks the cycle in all possible places by strengthening each of these rules one at a time.

Attribute with Wrong Arity: KRUST removes the extra arguments, or adds suitable arguments, in all possible ways. New arguments are obtained from the knowledge taxonomies used to implement changes.

From Completeness Checking Tools

KRUST organises the integration of the user-supplied, new knowledge within the existing KB. The user supplies the differentiating facts for the unsolved case and KRUST assembles them into a new rule which it inserts with a higher priority than any error-causing rule. We wish to incorporate an inductive learner within KRUST so that more general new rules can be learned.

From Testing Tools

Testing tools use a set of examples to monitor the testing process. These examples are a source of training examples for KRUST. The user can investigate the testing output (e.g. traces) to select those examples for which he wishes the KB repaired. Testing tools that incorporate test sample generation can themselves provide pertinent examples from which the expert may choose training examples.

5.2 Supplying Evaluation Metrics

In addition to V&V evidence, KRUST can utilise the test examples used by V&V and the metrics available to testing tools in its KBS filtering and evaluation phases. KRUST adopts a generate-and-test model and thus benefits from the pruning achieved by having access to accurate rejection criteria. The second filter requires **important** examples, ones which the KB must answer correctly. Test examples from the requirements specification are suitable benchmarks on which to reject refined KBs which fail them. Test sample generation tools can supply pertinent examples from which the expert can select **important** ones. After this coarse filtering the remaining refined KBs are ranked according to "performance". Again test generation tools can provide relevant test examples and, in addition, constraints on the test generation can concentrate attention on particular parts of the KB.

In addition to the use of testing tools to document filtering, KRUST can also employ V&V tools as coarse filters; e.g. consistency checking rejects KBs containing contradictory values, incremental verification (Meseguer 1992) would explore faults introduced by the updates. Testing tools also offer a range of metrics, on which KRUST can base its final evaluation of refined KBs, thus extending KRUST's current statistical method. In Figure 2, the Completeness, Consistency and Testing tools have been linked to KRUST's second filter and evaluation module to indicate that these V&V tools can be called to perform part of the pruning and evaluation.

5.3 Additional Knowledge

The V&V tools described above are knowledge-based and often heuristic. Not only can KRUST utilise this additional knowledge, but also many dynamic V&V tools explore deduction chains within the KB, a type of search conducted by KRUST. The V&V tools can pass the results of these efforts directly to KRUST. Thus, the search effort within the integrated tools is not duplicated. Figure 2 also shows the links to KRUST from this so called Validation Knowledge.

5.4 The ViVa Project

The ViVa project (Esprit P6125)[1] addressed the issue of integrating V&V and refinement tools as exemplified by its full title: "**V***erification,* **I***mprovement &* **VA***lidation of Knowledge Based Systems*". In addition to providing an integrated toolset of V&V and refinement tools, ViVa aimed to define a methodology for using the tools. ViVa also had a common knowledge representation, which contained all the necessary hooks required by the tools, easily available.

An important criterion for KBS development tools is their suitability for real-life KBSs. Access to real-life KBSs for academic research is often difficult, but the ViVa project was applications-driven, and aimed to provide both KBSs and descriptions of the V&V and refinement needs of KBS developers.

Improvement, as defined above, implies the two stages: interpretation of faults and implementation of changes. This coincides with KRUST's 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. The other ViVa tools cover the types of V&V described in Section 4 and satisfy the Verification and Validation objectives of the project. Figure 2 thus represents the functionality of a KRUST-based improvement tool within the ViVa toolset.

This project was rescoped after 21 months to concentrate on building a toolbox of validation and verification tools.

[1]The ViVa partners were 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).

6 Summary

In this paper we have described how an automated refinement tool can make use of the output of V&V tools and thus give added functionality to the V&V tools. In addition to simply identifying possible faults, the KB is explored so that the causes of the faults can be identified and the anomalies can be rectified by actually refining the KB in response to the evidence that faults have been found. We have discussed the suitability of our KRUST refinement tool to act in parallel with various V&V systems. Its behaviour can be moulded to suit the available evidence and so can make use of various forms of output from V&V tools as training information. In particular, KRUST is unusual in generating many refinements initially to remove a particular anomaly. However its pruning and evaluation modules can be biased to suit the available evidence. Thus KRUST can be regarded as a flexible refinement tool which can be adapted to favour the various types of change which may be required.

The importance of quality assurance means that a user should be encouraged to undertake V&V at all stages of the development of a KBS. The burden of V&V is eased by incorporating refinement within the fault-finding phase, because such a system additionally suggests **how** anomalies may be fixed. We feel that KBS developers may be encouraged to undertake more rigorous V&V if a complementary refinement process can suggest repairs for the anomalies which are highlighted. Thus the combined use of V&V and refinement may ease the "Knowledge V&V Bottleneck".

7 Acknowledgements

This work has developed from my PhD studies with Derek Sleeman at the University of Aberdeen and I thank him for this and my inclusion as part of the ViVa team in Aberdeen. I am also grateful to Marc Ayel, Laurence Vignollet and Stephane Talbot from LIA, Université de Savoie for discussions about the SACCO and SYCOJET tools. Finally I wish to thank the anonymous reviewer who directed my attention to Ripple Down Rules for his helpful discussion comparing their role to assist knowledge maintenance and more standard knowledge refinement.

References

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

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.

Carbonell, J. G. (1991), Scaling up knowledge-based systems via machine learning, Invited Talk, Fifth European Working Session on Learning (EWSL-91), Oporto, PORTUGAL.

Compton, P., Edwards, G. et al. (1992), 'Ripple down rules: Turning knowledge acquisition into knowledge maintenance', *Artificial Intelligence in Medicine* **4**, 47–59.

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

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

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* B. Buchanan & E. H. Shortliffe, eds, 'Rule-Based Expert Systems', Addison-Wesley, Reading, MA, pp. 171–205.

De Raedt, L. (1992), *Interactive Theory Revision*, Academic Press, London.

Ginsberg, A. (1988), *Automatic Refinement of Expert System Knowledge Bases*, Research Notes in Artificial Intelligence, Pitman, London.

Kang, B. H., Gambetta, W. & Compton, P. (1994), Validation and verification with ripple down rules, *in* 'AAAI Workshop on Validation and Verification', Seattle, WA, pp. 64–69.

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

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

Morik, K., Wrobel, S., Kietz, J.-U. & Emde, W. (1993), *Knowledge Acquisition and Machine Learning*, Academic Press, London.

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.

Ourston, D. & Mooney, R. (1994), 'Theory refinement combining analytical and empirical methods', *Artificial Intelligence* **66**, 273–309.

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* B. Neumann, ed., 'Proceedings of the ECAI92 Conference', Wiley, Vienna, AUSTRIA, pp. 835–839.

Richards, B. L. & Mooney, R. J. (1991), First-order theory revision, *in* L. Birnbaum & G. Collins, eds, 'Machine Learning: Proceedings of the Eighth International Workshop', Morgan Kaufmann, Evanston, IL, pp. 447–451.

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

Saitta, L., Botta, M. & Neri, F. (1993), 'Multistrategy learning and theory revision', *Machine Learning* **11**(2/3), 153–172.

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

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

Wogulis, J. & Pazzani, M. (1993), A methodology for evaluating theory revision systems: Results with AUDREY II, *in* R. Bajcsy, ed., 'Proceedings of the Thirteenth IJCAI Conference', Chambery, FRANCE, pp. 1128–1134.