KBS Maintenance Validation Using Simulation

F. Coenen and T.J.M. Bench-Capon

Department of Computer Science, Liverpool University, Chadwick Building, P.O. Box 147, Liverpool L69 3BX, England.

ABSTRACT

Whenever a KBS maintenance session has taken place it is necessary to revalidate the KB. A number of validation/testing tools and techniques are available to assist in this process. However none of these fully address the real requirements of the maintenance engineer. In this paper these requirements are considered and a KBS simulation tool to facilitate the validation of KBS after maintenance has been carried out described. The tool utilises a graphical representation technique to depict the KB and allows the user to interactively "people" the KB either partially or fully. Inference and tracing mechanisms are then used to allow the maintenance engineer to validate the KBS by inspecting its behaviour. In this manner both those inferences that can be drawn, and those that cannot be drawn, as a result of a particular assertion or set of assertions, and what inferences lead to a particular conclusion, can be simply identified, so that a fully validated product can be returned to the end users.

INTRODUCTION.

When maintaining KBS it is necessary to revalidate the system after any maintenance action has taken place. By validation we mean the process of determining whether a KBS operates as the expert expects it to. A definition taken from Barry Boehm's definition of validation for traditional software systems (Boehm [1]); "are we building the right product?"

To this end there are a number of KBS verification and validation tools and techniques availables. An overview of these is given in Section 2. However, although these tools and techniques facilitate the identification of aberrant rules and clauses, they are essentially "black box" techniques, they do not directly address the validation requirements of the maintenance engineer. Broadly speaking the maintenance engineer needs to validate system behaviour and not just the output. Testing must therefore be "glass box" and step by step. In addition existing KBS validation/testing methods typically require complete specification of test cases for any results to be obtained; further there is no indication, till the test is complete, that the "right" test is being implemented.

We consider that KBS validation must be carried out in a transparent manner so that the behaviour of the KBS is obvious. More specifically we suggest that the requirements for validation of KBS maintenance engineers are as follows:-

i) The ability to interactively input data.

ii) The ability to observe individual inferences that result as individual or sets of input data are asserted thus avoiding the need to specify complete test cases.

iii) The ability to force inferences anywhere in the rule chain without recourse to determining what the necessary input data to establish a particular inference are and then needing to instantiating that data at the leaf nodes.

iv) Where incorrect results are obtained the ability to be able to easily identify the appropriate malevolent rule or rules.

v) Where expected inferences are not drawn the ability to identify why they were not drawn.

It is our belief that these requirements can only be addressed by KBS simulation supported by a graphical representation off the KB. Such a rule base simulation tool is described here. It has been specified (Coenen and Bench-Capon [2]) as part of the MAKE (Maintenance Assistance for Knowledge Engineers) project; a collaboration between Liverpool University, ICL and British Coal aimed at developing a suite of KBS maintenance tools supported by a KBS development methodology, called MADE (Make Authoring and Development Methodology), that will encourage the production of maintainable systems (Coenen et. al. [3]). The tools and methodology are designed to be used on KBSs represented using a predicate calculus notation translated into a conjunctive clausal form supported by a class hierarchy. The latter defines the vocabulary of the system and instances of the classes can be used to indicate the "state" of the system at any given time. The class hierarchy supports inheritance, but requires sub-classes to be strict specialisations of their super-class. The class hierarchy also specifies all the possible values that a particular attribute can have, these may be ranges, booleans (true or false), literals etc. For a more detailed description of the make up and construction of the class hierarchy interested readers are referred to Bench-Capon and Forder [4]).

The simulation tool allows the user to, step by step, create instances so as to "people" a KB and at any time observe what inferences can occur as a result, or alternatively to trace how a particular inference can be made. The KB is graphically represented using another tool developed as part of the MAKE project and currently in operation, the Rulemap (Coenen and Bench-Capon [5]), a brief description of which is given in Section 3. The process of peopling the rule base is then described in Section 4 and the inferencing and tracing mechanisms in Sections 5 and 6 respectively. Some conclusions are drawn in Section 7. The discussion is supported by a number of examples, these are based on the following trivial, but flawed, example rule base designed to establish the relation `cosister`.

```
RULE 1.
(personA cosister personB)
iff
(personA sister personB)
and
(personB sister personA)

RULE 2.
(personA sister personB)
iff
(personA sex female)
and
(personC == personD)
    when
        (personA hasParent personC)
        and
        (personB hasParent personD)
```

Where `personA`, `personB`, `personC` and `personD` are all members of the class `person`. The rule base then translates into the following conjunctive clauses:-

```
CLAUSE 1.1
(personA cosister personB)
if
(personA sister personB)
and
(personB sister personA)

CLAUSE 1.2
(not personA cosister personB)
if
(not personA sister personB)
```

```
CLAUSE 1.3
(not personA cosister personB)
if
(personB sister personA)

CLAUSE 2.1
(person A sister personB)
if
(personA sex female)
and
(personC == personD)
    when
        (personA hasParent personC)
        and
        (personB hasParent personD)

CLAUSE 2.2
(not personA sister personB)
if
(not personA sex female)

CLAUSE 2.3
(not personA sister personB)
if
(not personC == personD)
when
    (personA hasParent personC)
    and
    (personB hasParent personD)
```

OVERVIEW OF SOME EXISTING TOOLS AND TECHNIQUES.

There are a number of verification and validation techniques and tools available that may be used to revalidate a KB after maintenance has been carried out. These include Suwa's ONCOCIN rule checking program (Suwa et. al. [6]), CHECK (Nguyen et. al. [7]) and EVA (Chang et. al. [8]) to mention a few of the more notable. To a large extent these examine the rule base statically detecting anomalies such as subsumption, redundancy, inconsistencies etc. However the proof of the KB lies in the consequences it licences. Thus the KBS must also be examined dynamically. Typically this is carried out by running test cases and comparing the result with that suggested by domain experts. Where the results do not compare favourably further maintenance must be carried out. This requires the identification of the rule or rules that cause the KBS to behave incorrectly which may not necessarily be those that the knowledge engineer has been working on. There are a number of tools available to do this. The simplest are the debugging tools and tracers supplied with many shells and implementations of expert system languages such as Prolog and Lisp. However in a realistically sized application these become impractical to use. Further they require a lot of user interaction, have very low level commands and provide no indication of "how" and "why". More sophisticated methods include execution trail tracers and graphers. These indicate the inferences that have been made and how they have been made by allowing the user to step through the inference process. A criticism of these methods is that a considerable amount of deduction is still required on behalf of the user. Use can also be made of the explanation facilities provided with many applications, but these tend to be aimed at the end user and not the maintenance engineer. Some authors (Neches et. al. [9]) consider that explanation capabilities should be specifically developed to facilitate maintenance and indeed there may be a case for this.

Mention should also be made here of the various rule refinement systems that have been developed such as TEIRESIAS (Davis [10]), KRUST (Craw and Sleeman [11]) and SEEK (Politakis [12]) and SEEK2 (Ginsberg and Weiss [13]). However it should be emphasised that these are rule refinement systems and are not directed at maintenance consisting of (say) the addition of a number rules coupled with the removal and modification of others.

DISPLAYING THE RULE BASE USING RULEMAP.

The Rulemap, also sometimes referred to as the Datamap, graphically displays a rule base at both the rule-attribute and proposition-clause level. It allows the user to navigate up and down the individual levels and to move between the two. The Rulemap also has a number of facilities attached to it to allow the user to interact with it, for example to inspect the possible values that an attribute may have or to view an entire rule in its MADE representation.

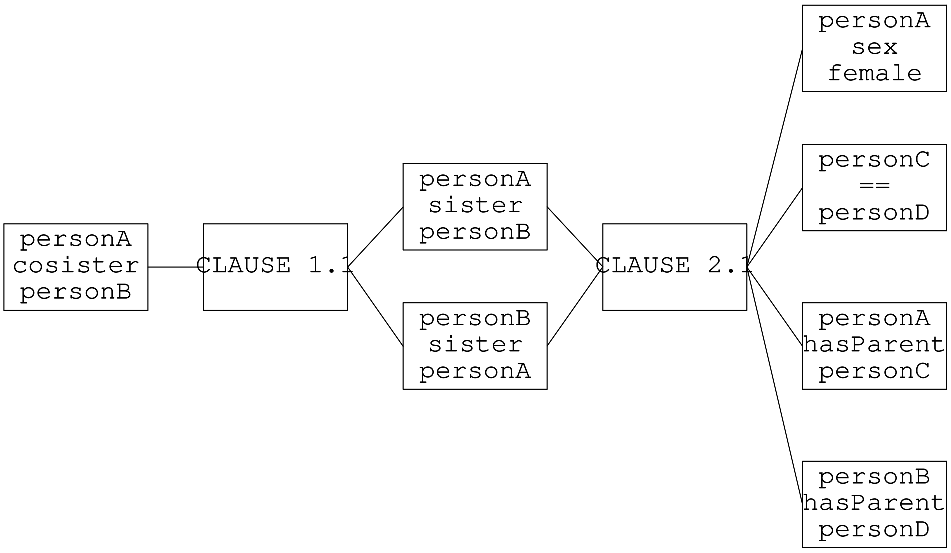


Fig 1. Proposition-Clause Rulemap

Figure 1 shows part of the proposition-clause Rulemap for the example rule base (the clauses with negated heads are not shown). Due to the conjunctive clausal form used all propositional furcations in the graph represent conjunctions. Thus any two or more propositions branching out from a clause are cojoined, while any two or more clauses branching out from a proposition represent alternatives. In other words we can identify "and" and "or" connectors from the graph.

It is the proposition-clause level representation which is used in conjunction with the rule base simulation tool.

PEOPLING THE RULE BASE.

To animate a rule base we must "people" it by creating instances of entities and by asserting propositions to establish attribute values for those entities. This is implemented interactively by clicking on propositions for which the user wishes to supply an instance. To this end a number of types of proposition have been identified. For our example we need consider only two:-

1) PREDICATIONS.

   Single entity propositions, for example `personA sex female` where the single entity is `personA`.

2) RELATIONS.

   Dual entity propositions, for example `personA cosister personB` where the entity pairs are `personA` and `personB`.

When peopling the rule base the following requirements are relevant.

(i) Provision for the user to be able to, if he/she so desires, to select a particular instance from a list of existing instances (if any) applicable to the class of object under consideration.

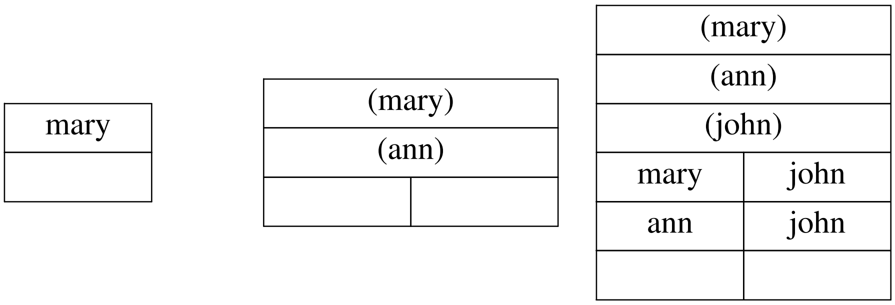
(ii) The ability to prevent the user from asserting a particular instance where this will contradict previous assertions. For example if the proposition `mary sex female` has been asserted then the user should be prevented from asserting a contradictory proposition such as `not mary sex female (assuming that in both cases mary` refers to the same instance).

(iii) An indication of instances that have already been associated with a proposition.

When starting with an unpeopled Rule Base there will be no existing instances of entities available from which the user may make suitable selections. The user must therefore create the required instances. This is achieved by clicking on a proposition we wish to assert. The result will be empty menus of the form illustrated in Figure 2 (a) and (b) depending on the type of proposition selected. The user can now enter an instance for the proposition in question by typing into the box. In the case of a relational proposition both boxes must be completed.

(a) Predication (b) Relation

Fig 2. Proposition Menus (1)



(a) personA sex female (b) personA sex female (c) personA parent personC

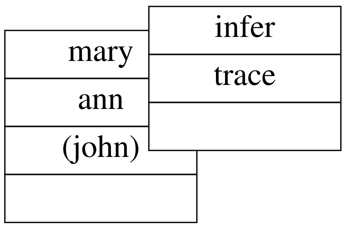
Fig 3. Proposition Menus (2)

For example if we were to click on the proposition `personA sex female` we could (say) enter the instance `mary` into the empty menu displayed. The effect of this would then be to assert the proposition `mary sex female.` We could create another instance for the same proposition, by clicking on it again. This time a menu of the form given in Figure 3(a) would appear, note that `mary` now appears in this menu indicating that this instance already exists and has been associate with the attribute-value `sex female.` Another instance can now be created, say `ann,` by typing into the empty block again. The effect of this is to assert the proposition `ann sex female.` We can also assert propositions using existing instances. For example if we click on the proposition `personA hasParent personC` a menu of the form depicted in Figure 3(b) would result. The bracketted instances indicate that these instances exist and that they have been bound to other propositions in the rule base but not the proposition presently under consideration, i.e. they are possible candidates for the subject(s) of the selected proposition. We can either select one or more of these existing instances or create further new instances by typing into the empty blocks. The first will be instantiated to `personA` and the second to `personC.` For example we may select `mary` from the list of existing instances for the entity `person` and instantiate it to `personA` and create a new instance, `john,` and instantiate it to `personC.` The effect of this will be to assert the proposition `mary hasParent john.` If we clicked on the `personA hasParent personC` proposition again the proposition menu will now be of the form given in Figure 3(c). Similarly we can assert the proposition `ann hasParent john.`

In this manner the rule base can be peopled interactively with a complete or partial set of test data. In the above illustration this has been carried out from the leaf propositions. It is suggested that this is the approach that will most closely match the normal operation of the KB. However instances can equally well be created by clicking on other propositions in the KB, if other aspects of the behaviour of the KB need to be investigated.

INFERENCING.

Inferencing can take place at any time, although to be effective some peopling must have taken place. The inferencing process uses the inference engine that forms part of MADE thus no additional control is required. The user proceeds as follows. First a subject for the inference must be selected by clicking on a proposition, selecting an instance from the resulting proposition menu which is to be the subject of the inference and then dragging right. A sub-menu will now appear from which the inference option can be selected. The effect of selecting this option is that inferences are attempted with respect to the selected subject for the inference based on the information asserted to date. The inference trail or trails are indicated on the Rulemap, either leading up to a root proposition or until the inference can not continue any further due to the absence of necessary information. In the latter case further propositions can be asserted as necessary in the same manner as described above.



personA sex female

Fig 4. Proposition Sub-Menu

For example if we wish to select `mary` as the subject of our inference we may click on the proposition `personA sex female,` select the instance `mary` from the resulting proposition menu and then drag right and

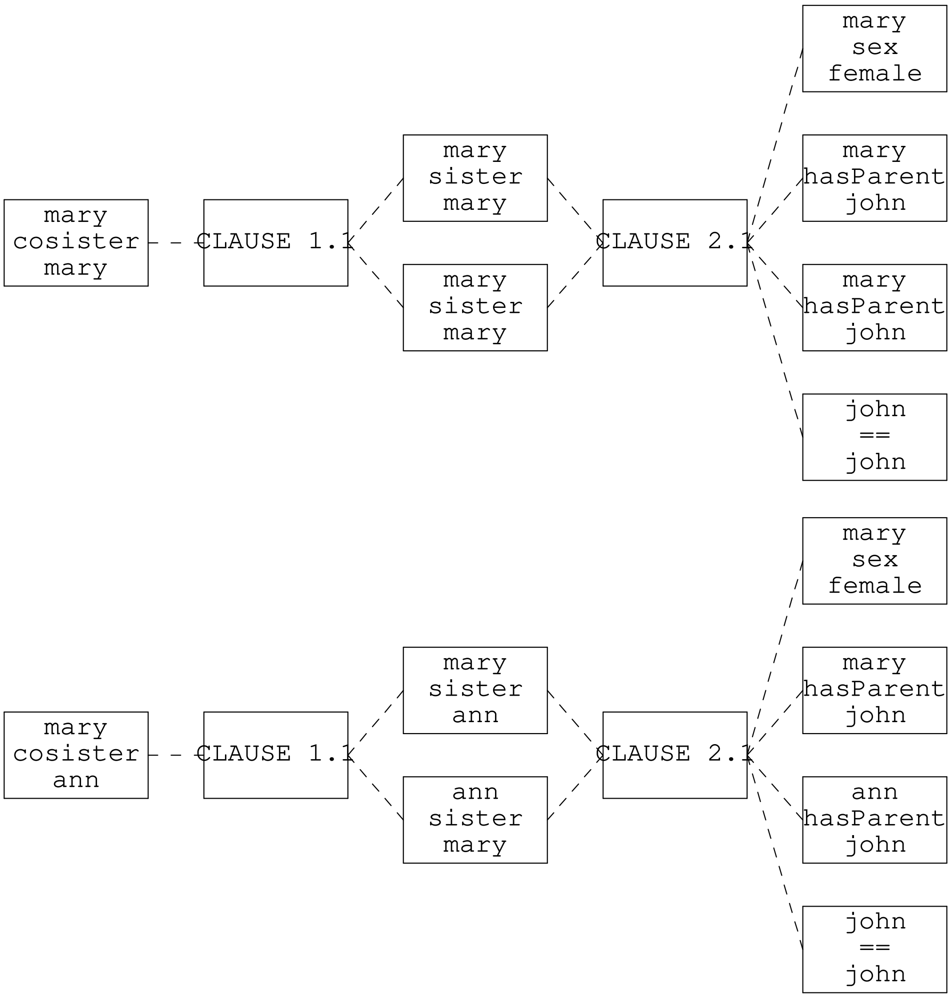


Figure 5: Proposition-Clause Rulemap

select the infer option (Figure 4). A Rulemap of the form shown in Figure 5 will result (again clauses with negated heads are not shown). Note that where applicable instance names have been instantiated and that the inference trail is highlighted. The Rulemap immediately shows a flaw in the KB in that a person can be a cosister of themselves, this is clearly not the intention. At first glance the problem might appear to be located in Rule 1, but if we look a little further along the graph we can see that the problem is actually located in Rule 2, since `mary sister mary` is itself undesired. We can now inspect Rule 2, using the Rulemap facilities. From this it then becomes clear that Rule 2 requires an additional proposition:-

```
RULE 2.
(personA sister personB)
iff
(personA sex female)
and
(personC == personD)
    when
         (personA hasParent personC)
         and
         (personB hasParent personD)
and
(personA == personB)
```

TRACING.

Where as inferencing is a bottom up approach to inspecting the KB, tracing is the reverse top down approach i.e. root to leaf propositions. Before tracing can be implemented a start proposition, not necessarily a head proposition, must be selected as the point at which the trace is to commence. This is carried out in a similar manner as described above by selecting a proposition, identifying an instance and then dragging right and selecting the trace option from the resulting sub-menu (Figure 4).

The tracing facility clearly indicates the inference path that would have to be followed to establish the start proposition. Unlike the infer option no account is taken of whether the user has asserted any propositions along the route. Further, during the operation the Rulemap is pruned so that irrelevant branches are not shown.

The pruning process is implemented by removing "nullified" propositions and clauses from the Rulemap. In its simplest form a nullified proposition is a proposition which cannot be true. If we consider the following two propositions:-

```
(Entity Attribute true)
(Entity Attribute false)
```

where the entity and attribute referred to are both the same, and if we know that the first proposition is "true" then the second must be "false". If we required our trace to start at the first proposition the second would be irrelevant and therefore considered to be nullified. A nullified clause in turn is one for which at least one body proposition is nullified (clauses consists of conjuncted propositions). Nullified propositions and clauses do not appear in the Rulemap when implementing the trace option. This greatly enhances the clarity of the trace as the inference trail is demonstrated free of the clutter of unrelated rules and clauses. To determine whether a proposition or clause is nullified or not the following "rules" must be applied starting with the nullification of the start proposition for the trace.

RULE 1:
If a nullified proposition appears in the body of a clause and the clause is disjuncted with a sibling clause nullify the clause and any other propositions in the body but not the head proposition.

RULE 2:
If a nullified proposition appears in the body of a clause and the clause is not disjuncted with a sibling clause nullify the clause and any other propositions in the body or the head proposition.

RULE 3:
If a nullified proposition appears in the head of a clause nullify the clause and the body propositions unless a body proposition is also located in the body of another clause which has not be nullified.

Rules 1 and 2 are first used to prune clauses and propositions located above the nullified inverse of the start proposition and Rule 3 the section of the Rulemap below the nullified inverse of the start proposition. These rules are applied repeatedly to a clause list, clause by clause, until a "clear run" is attained. The nullification process proceeds both up and down the Rulemap towards both the root and leaf propositions.

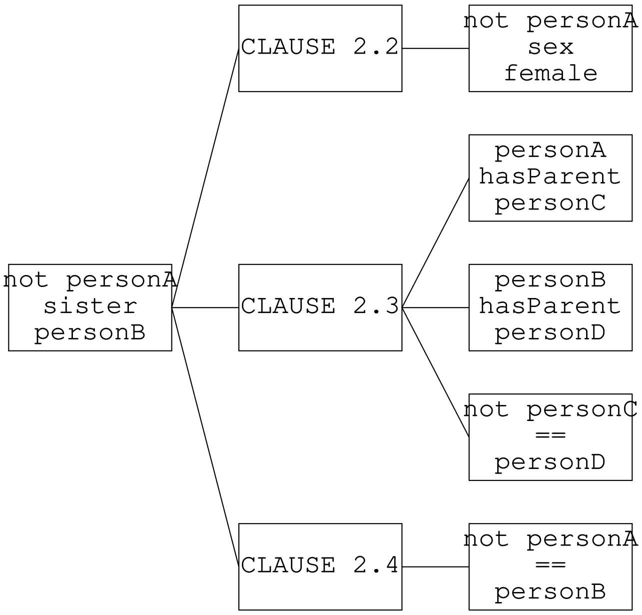


Figure 6: Inference Trace

If, for example, we wish to start the trace from the proposition `not personA sister personB` we click on it, select an instance, and then drag right ad select the trace option from the associated submenu (Figure 4). The resulting Rulemap is depicted in Figure 6. Note firstly that the execution trail leads all the way down to the leaf propositions required to establish the root proposition, unlike the infer option this will occur regardless of whether the intervening propositions have been asserted or not. Secondly note that the Rulemap has been pruned by ignoring irrelevant branches.

The pruning process will start with the nullification of the inverse of the start proposition `personA sister personB.` Clauses 2.1 will then be nullified according to Rule 3. Where applicable, by repeated application of Rule 3, the entire branch leading down from a nullified start proposition will be nullified and hence not depicted in the rule map.

CONCLUSIONS.

A rulebase simulation tool to animate and illustrate the behaviour of a KBS has been described. Broadly three circumstances are envisaged where this tool may be of use:-

1. Where a test case has produced some wrong answer either because some unexpected conclusion has been drawn, or because some desired conclusion has not been drawn.
2. To ensure that, given some data, some particular conclusions are drawn and/or some other particular conclusions are not drawn.
3 To explore the KB, perhaps to determine the effect of a particular Rule, i.e. what consequences flow from a rule on an assertion or set of assertions, or what assertion or set of assertions would licence a particular conclusion.

More specifically, given one or more assertions, the user will be able to determine one or more of the following:-

1. What Inferences can be drawn.
2. What Inferences cannot be drawn.
3. What Inferences lead to a particular conclusion.
4. What further assertions will be of interest in exercising the KB.

These can all be done using the infer and trace facilities described above.

In some cases it may still be desirable to load part or all of a test case before commencing interaction, in which case, rather than asserting each proposition individually using the peopling method described above. Provision will be made to load the assertions in the form of a case. Cases can be constructed using an editor and then loaded into the simulation tool where appropriate inferencing and tracing can then take place. The case can then be extended or modified in the light of the result.

REFERENCES.

1. Boehm, B.W. Software Engineering Economics. Prentice-Hall, 1981.
2. Coenen, F., Bench-Capon, T.J.M. & Smith. M.J. KBS Development Using X Windows: The Made Development methodology Presented at UKUUG 91, Liverpool, p64-72, 1991.
3. Coenen, F. and Bench-Capon, T.J.M. A Specification for a Dynamic Rule Base Testing Tool. MAKE Report 5/91, Deliverable D5b, Dep.

Computer Science, Liverpool University, 1991.

4. Bench-Capon, T.J.M. and Forder, J.M. Knowledge Representation for Legal Applications. In Bench-Capon, T.J.M. (Ed). Knowledge Based Systems and legal Applications. Academic Press, p245-264, 1991.

5. Coenen, F. and Bench-Capon, T.J.M. A Graphical Interactive Tool for KBS Maintenance. In Karagiannis, D. (ed). Database and Expert Systems Applications. Proceedings DEXA 91, Berlin, Springer-Verlag, p166-171, 1991.

6. Suwa, M., Scott, A.C. and Shortliffe, E.H. Completeness and Consistency in a Rule-Based System. In Buchanan, B.G. and Shortliffe, E.H. (eds). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, p159-170, 1984.

7. Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora D. Knowledge Base verification. AI Magazine, summer, p67-75, 1987.

8. Chang, C.L., Combs, J.B. and Stachowitz, R.A. A Report on the Expert Systems Validation Associate (EVA). Experts Systems with Applications, Vol 1, No 3, p219-230, 1991.

9. Neches, R., Swartout, W.R. and Moore, J.D. Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development. IEEE Transactions on Software Engineering, Vol SE-11, No 11, p1337-1351, 1985.

10. Davis, R. Interactive Transfer of Expertise. In Buchanan, B.G. and Shortliffe, E.H. (eds). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, p171-205, 1984.

11. Craw, S. and Sleeman, D. Automating the Refinement of Knowledge-Based Systems. Proceedings ECAI 90, Stockholm, August 6-10, p167-172, 1990.

12. Politakis, P. Empirical Analysis of Expert Systems. In Research Notes in Artificial Intelligence 6, 1985. Pitman.

13. Ginsberg, A. and Weiss, S. SEEK 2: A Generalised Approach to Automatic Knowledge Base Refinement. Proceedings IJCAI, p367-374, 1985.