Expert System Validation through Knowledge Base Refinement

Pedro Meseguer +
Institut d'Investigaci6 en Intel.ligencia Artificial, C.S.I.C.
Cami Sta Barbara s/n, 17300 Blanes (Girona)
SPAIN

Abstract

Knowledge base (KB) refinement is a suitable technique to support expert system (ES) validation. When used for validation, KB refinement should be guided not only by the number of errors to solve but also by the importance of those errors. Most serious errors should be solved first, even causing other errors of lower importance but assuring a neat validity gain. These are the bases for IMPROVER, a KB refinement tool designed to support ES validation. IMPROVER refines ES for medical diagnosis with this classification of error importance: false negative > false positive > ordering mismatch. IMPROVER is being used to validate a real ES and some empirical results are given.

1 Introduction

Software validation aims at assuring the compliance of a program with its requirements, which capture the needs of the final user. This concept remains fully applicable when the target software is an expert system (ES), although ES peculiarities demand specific validation methods. A main difficulty in E)S validation is the lack of detailed and testable requirements [Rushby 88]. Aldiough some work has been developed in tins sense, it becomes increasingly apparent that achieving a complete set of testable requirements for some ES tasks (i.e., medical diagnosis) is currently unfeasible [Krause et al, 91]. An important part of the validation process relies on manual testing using known cases and matching ES outputs against experts' opinions. When discrepancies exist, the ES is updated until a satisfactory performance is achieved.

Theory refinement considers the improvement of an approximate domain theory from a set of cases widi known solution. A refinement problem exists when some of these cases are treated incorrectly in the theory. Using machine learning techniques, the theory can be modified to achieve a correct treatment on all cases. It is assumed that the theory is close to a satisfactory state. The set of cases should be a representative sample of the problem domain.

+ Current address: Departament de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, SPAIN.

The manual testing-update cycle in ES validation can be automated using theory refinement techniques. The theory is the knowledge base (KB) and the process is known as KB refinement. This approach has been considered by several authors, which have developed automatic KB refinement tools. These works have been mainly focused on developing learning strategies to improve the KB, but the validity of the refined ES has not been considered in detail.

In this paper, we propose KB refinement as a suitable technique to support ES validation. This work shares concepts with previous refinement approaches, although the emphasis is put on the validity of the final ES instead on the learning capabilities of the refinement process. Since the main concern is to improve ES validity, performance errors are no longer considered of equal importance but they are weighted by their *relative impact* on the ES task. This impact depends on the type of error and on the elements involved in this error, aspects that are application-dependent. Based on these ideas we have developed IMPROVER, a KB refinement tool for ES validation, that is guided by error importance. When solving an error, IMPROVER may generate new errors of lower importance, but always assuring a neat validity gain. We have used IMPROVER to refine ES for medical diagnosis, where the solution for a case is a ranked list of elements. This has caused to consider a new type of refinement error, ordering mismatches.

This paper is organized as follows. Section 2 summarizes previous work. Section 3 analyzes KB refinement in ES validation. Section 4 details the *ES* model, and the refinement criteria and operators. Section 5 describes how IMPROVER works, while section 6 contains empirical results. Finally, section 7 encloses some conclusions.

2 Related Work

Verification is the most developed subfield in ES validation. Available verifiers give response to many structural problems (inconsistency, redundancy, circularity, unreachable goals, unfireable rules, etc.). However, the absence of structural problems is a necessary but not sufficient condition to get a valid system. Traditionally, validity has been assessed by testing (see MYCIN [Buchanan & Shortliffe 841 and RI [McDermont 81]). Manual testing against human experts is very expensive, resulting in a short number of test cases (10 in MYCIN, 50 in RI). Nevertheless, the problem is not the number of test cases but their coverage [O'Keefe et al, 87].

KB refinement systems by empirical learning over a library of known cases follow a common pattern composed of four phases: *identification, localization, generation* and *selection.* At the identification phase errors are detected by matching, for each case C_i, the set of hypothesis H_i, that the ES assigns to C_i against the correct set of hypothesis H_i^*. An hypothesis *h* is classified in: (i) *true positive,* $h \in H_i \cap H_i^*$,(ii) *true negative,* $h \notin H_i \cup H_i^*$,(iii) *false positive,* $h \in H_i$, $h \notin H_i^*$, or (iv) *false negative,* $h \notin H_i$, $h \in H_i^*$. False positives and false negatives are the errors to solve by KB refinement. A false positive indicates that the KB is too general and it should be specialized. Conversely, a false negative indicates that the KB is too specific and it should be generalized. The localization phase identifies the KB part responsible for these errors. The generation phase builds several refinements on this KB part. The selection phase determines those refinements to be implemented.

Early refinement systems like SEEK [Politakis & Weiss 84] and SEEK2 [Ginsberg et al, 85] refine propositional rule bases with a restricted form of uncertainty. Rule usage statistics are used in the localization phase. Refinements arc generated using heuristics. The Antidote Algorithm [Wilkins & Buchanan 86], solves misdiagnoses on rule bases under (probability based) uncertainty. Rules are ranked by responsibility on the identified errors. The only refinement operator is rule deletion. The KRUST system [Craw & Sleeman, 90] refines rule bases considering rule priority. All possible refinements are generated and filtered by heuristics. Refined KB are ranked using SEEK-like statistics. The RTLS system [Ginsberg 88] refines an operationalized propositional rule base by modifying label hypotheses. Refinements consist on label generalization/specialization. EITHER [Ourston & Mooney, 90] and DUCTOR [Cain 91] refine propositional theories. The localization phase is based on explanation analysis and the generation phase uses inductive learning techniques. Selection is made by testing theory performance. These refinement systems consider a simple knowledge representation, namely rules in propositional logic with some kind of uncertainty and, only KRUST, analyzes the role of control. The ES output consists on a single element, which can cause a false positive or a false negative. False positives and false negatives are considered equally important.

3 Validation and Refinement

In machine learning the goal of KB refinement is to improve ES performance by empirical learning. ES performance is measured as the number of errors detected executing the ES on the case library. When KB refinement is used for validation purposes, the goal is to improve ES validity, that is to say, the ES should be more valid after implementing each single refinement in the KB. This goal slightly differs from former one, since *more valid* is not equivalent to *performing a lower number of errors.* Different errors can have a different impact on the overall ES validity, so decreasing the number of errors does not always mean producing a better ES. To increase ES validity, KB refinement should be guided by error importance with respect to the ES task. Most serious errors should be solved first, even causing some errors of lower importance, but always assuring a neat validity gain. A classification of error importance with respect to the ES task is needed. This classification is application-dependent and may be based on the error type and on the elements involved in an error.

For instance, in the medical diagnosis domain a false negative (a diagnosis that does not appear in the ES output but it should) is a more serious error than a false positive (a diagnosis that appears in the ES output but it should not). A false negative may cause the actual origin of an illness to be ignored. A false positive introduces an extra diagnosis (what can be seen as noise in the ES output) but does not cause missing the right one. Difference in error importance comes from the impact of each error on the ES task.

The usage of KB refinement for validation also differs from KB refinement in machine learning. Every suggested modification by automatic refinement should be evaluated by the human expert responsible for the ES prior any update. Detailed justification of the proposed modifications should be provided, detailing the solved errors as well as the potential new errors introduced. After expert's evaluation, the modification can be accepted, rejected or modified. Refinement failures should also be reported to the expert, since they leave unsolved errors that will be treated by hand.

A KB refinement system and its two main functions, the automatic testing facility and the learning capability, can be of great help in the validation process during the ES life-cycle. At the development stage, the automatic testing facility can evaluate the achieved performance as the subset of cases correctly solved in the case library, while the learning capability can improve the current prototype. At the maintenance stage, automatic testing can evaluate the impact of KB updates on the ES performance, activating the learning capability if needed. When new cases are added to the library, these processes can be easily repeated. In this way, an accurate measure of the ES performance is always available. Obviously, a KB refinement system does not release human developers from their responsibilities. They should control all the steps in the refinement process and they should solve the remaining unsolved issues. A KB refinement system automatizes a set of activities that would require lots of effort if they were performed manually, providing modifications that objectively improve the ES validity according with the classification of error importance considered for the specific ES task.

4 Refinement on an ES Model

In order to carry out these ideas in practice, we selected for refinement the medical application PNEUMON-IA [Verdaguer 89], developed on the shell MILORD [Sierra 89]. This choice fixed the kind of ES model to be refined, that is to say, the specific task and the knowledge representation. This has influenced clearly the criteria and operators used in the refinement process, although the basic ideas remain fully applicable for any ES. The ES task is pneumoniae diagnosis, so this work may be applicable to other ES for medical diagnosis. The knowledge representation used is based on rules, including metarules and uncertainty management. In the following, we detail this ES model and the refinement criteria and operators we apply on it.

4.1 The ES Model

The ES model is based on rules, underlying propositional logic, with uncertainty management, including implicit and explicit control, and monotonia

KB structure. The KB is denoted by a 4-tuple $<F, R, M, MR>$ where *F* is a set *of facts, R* is a set of *rules, M* is a set of *modules* and *MR* is a set of *metarules.* A fact $f \in F$ represents an attribute in the problem domain and has both a value and a certainty value (cv) associated. Special facts called *goals* drive the deduction process. Two kind of rules exist, *concluding rules* and *up-down rules,* forming the disjoint sets R_{cl} and R_{ud}, such that $R = R_{cl} \cup R_{ud}$. concluding rule $r \in R_{cl}$ is formed by a conjunction of conditions on facts in its left-hand side *(lhs),* an assertion about the value of one fact in its right-hand side *(rhs)* and a cv. When r is fired, the concluding fact is asserted with a *cv* computed from the cvs of $lhs(r)$ and r. An up-down rule *rERud* is formed by a conjunction of conditions on facts in *lhs(r)* and an action to increase or decrease the cv of a fact/ in *rhs(r).* Up-down rules on / are always fired after concluding rules on/. Rules are fired backwards. A module contains a collection of rules and one or several goals. Each rule belongs to one module. A metarule $mr \in MR$ is formed by a conjunction of conditions on facts in *lhs(mr),* and an action in $rhs(mr)$. There are two types of actions: on modules and on the ES. Metarules acting on modules have a *cv* associated, as a measure of their strength. Metarules are fired forward as soon as their conditions are fulfilled.

Concerning uncertainty management, a cv is assigned to each fact representing positive evidence. Uncertainty is propagated through rule firing, using the functions *cv-conjunction* and *cv-modus-ponens.* The function *cv-disjunction* computes the cv of a fact independently deduced by different rules. The *certainty threshold x* cuts all deductions with a cv less than x.

Control is divided in implicit and explicit. Implicit control is coded as the conflict-set resolution criteria. Three criteria of decreasing importance have been considered: (i) select the most specific rule, (ii) select the rule with highest cv, and (iii) select the first rule. These criteria establish a total order in *R.* Explicit control is coded in metarules acting on modules or on the whole ES. Metarules on modules can perform two actions, *add m* or *remove m,* meaning that module m will be activated or inhibited for deduction. A module can be activated several times, but once it has been inhibited it cannot be activated again. Active modules are kept in the active module list *(ACL).* Modules in the *ACL* are ordered by the cv of their adding metarule. On the whole ES only the *stop* action can be performed.

ES function. The ES model works as follows: when it starts, a metarule builds up an initial *ACL.* Then the following cycle starts. The first module in the *ACL* is selected as the current module. Its goals are pursued using the rules contained in it. As soon as new facts are deduced, metarules are tested for firing, and the *ACL* is eventually updated. When every goal in the current module has been tried, a new current module is selected and the cycle restarts. The ES stops when no more modules are available in the *ACL* or a metarule stopping the ES is fired. After termination, the ES output is the set of deduced goals ordered by their cv. To test performance, the ES output in a case C,, the ordered set H_i, is matched against the correct ordered set H_i^*, obtaining the following errors: (i) *false negative,* $h \notin H_i, h \in H_i^*$, (ii) *false positive,* $h \in H_i, h \notin H_i^*$, and (iii) *ordering mismatch,* $h \in H_i, h \in H_i^*$, *position* $(h, H_i) \neq$ *position* (h, H_i^*). Ordering mismatch is a new type of error caused by the structure of ES output.

ES task. The ES task is medical diagnosis, specifically diagnosis of extrahospitalary pneumoniae in adults in the first days of infection [Verdaguer 89]. The ES goal is to obtain, from the patient's data, the subset of microorganisms that are more likely to cause the infection. In very few cases this subset is formed by a single element, because usual symptoms do not discriminate enough to isolate a single cause. The classification of error importance is as follows: *false negative > false positive > ordering mismatch.* Clearly a false negative is a more important error than a false positive in medical diagnosis (at least for pneumoniae diagnosis), and both are more important than ordering mismatch, which simply indicates discrepancy in diagnostic ranking.

4.2 Refinement Criteria and Operators

First of all, KB refinement in ES validation should be guided by error importance with respect to the ES task, solving most important errors first. According to the classification of error importance for pneumoniae diagnosis, refinement should try to solve false negatives first, followed by false positives and finally ordering mismatches. The classification of error importance is application-dependent, so this one is only valid for the considered ES.

Second, every type of knowledge in the KB is subject to potential refinement. This applies to domain and control knowledge, since both types of knowledge may be responsible for false negatives and false positives, while domain knowledge is the only responsible for ordering mismatches. Refinement of certainty degrees of rules and metarules is obviously included.

Third, the number of generated refinements must be controlled. This is needed to prevent combinatorial explosion, since the set of modifications that can potentially solve an error is very large. If every potential refinement was tried, the process would be computationally intractable. For this reason, all the implemented systems include some heuristics to control refinement generation. They are based on the following assumption: the KB state is close to a correct state. Based on this assumption, we made two choices: (i) minimal changes are preferred, and (ii) refinements cannot delete KB objects. This second choice supposes that every KB object has some prior justification, what is quite reasonable for ES built with the support of knowledge engineers and following some development methodology. In other words, we assume that single erroneous rules as totally wrong associations of conditions and conclusion do not exist[1]. Errors are caused by KB

[1] If they exists, they are easily detected by the human expert using pure KB inspection. Difficult problems always involve several rules (and/or metarules) forming a deductive chain.

incompleteness or by small rule defects, that can affect both control and domain knowledge. The KB is assumed consistent, since consistency and other structural properties can be achieved using verifiers (see [Meseguer 91] and [Meseguer 92] for an incremental verifier on this ES model). Based on these assumptions, the legal refinements operators are the following:

OP1. Generalize/specialize conditions in the left-hand side of rules/metarules.
OP2. Modify the certainty degree of rules/metarules.
OP3. Modify the certainty degree in conclusions of up-down rules.
OP4. Modify the right-hand side of a metarule.
OPS. Add conditions to the left-hand side of rules/metarules.
OP6. Add new rules/metarules to the KB.

Proposed modifications may cause new errors. The following criteria establish when a modification is acceptable to guarantee a neat validity gain:

AC1. A modification solving *n* false negatives but causing *p* false positives is acceptable when $n \geq p$.
AC2. A modification solving *p* false positives but causing *n* false negatives is acceptable when $p \geq n * \alpha$. We have determined empirically $\alpha = 2$ for PNEUMON-1 A.
AC3. A modification solving *o* ordering mismatches is acceptable when (i) it does not generates any new false negative nor positive, and (ii) when it causes *o'* new ordering mismatches, $o > o'$.

5 IMPROVER: A Tool for KB Refinement

IMPROVER is a KB refinement tool to enhance ES validity. It is composed of three stages: *solving-false-negatives. solving-false-posiiives* and *solving-ordering-mismatches.* Each stage tries to solve an specific type of error. Stages are invoked in this order, because solving an error may generate other errors of lower importance. Stage ordering can be altered, to adapt IMPROVER to other classifications of error importance. IMPROVER has limited the generated refinements to one elementary change on a single KB object. This choice has been quite effective, preventing an exaggerated use of computational resources. In the following, we explain how IMPROVER works in every stage and in every refinement phase. The error identification phase is common to all the stages and is explained separately.

5.1 Error Identification

The first issue in error identification is the definition of the right solution for a case, the *gold standard.* [Gasching et al, 83) provides two definitions of gold standard for a case: (i) the objective correct answer, or (ii) what a human expert considers as the correct answer, using the same information that is available to the ES. They take the second definition because the first one is often inapplicable. This is also our approach, since in medical diagnosis the exact illness cause is often unknown, and in occasions can only be obtained by aggressive tests or by autopsy.

To compute a gold standard the opinion of a group of independent experts is required for each case. Experts usually disagree, so a consensus function is needed. Experts' opinions are ranked lists of diagnoses, and the consensus function generates another ranked list in the following form. Each diagnosis is assigned to the position obtained by computing the position mean value of the diagnosis in the experts opinions. Diagnoses with very close positions are grouped into a class, assigned to the mean value of their corresponding positions. The last class in the consensus is eliminated, because it corresponds to diagnoses only mentioned by one expert in a low position.

Errors are identified by matching the ES output against the gold standard for every case. IMPROVER obtains the ES output simulating ES execution. Thus, data coming from ES execution simulation and from KB refinement are treated homogeneously. IMPROVER performs independent analyses of domain and control knowledge, detecting defects that would be occluded by odier defects in actual ES execution. Specifically, domain knowledge is represented by an *and/or tree,* where *and* nodes represent rules and *or* nodes represent facts in rule conclusions. Rule priority is also recorded. Control knowledge is represented in a separated *and/or tree,* where *and* nodes represent metarules and *or* nodes represent modules. Domain and control knowledge can be refined independently. Deduction details for all die deduced facts in all cases are recorded in a table indexed by cases/diagnoses. It allows refinement to focus on specific KB parts, always considering the whole case library.

5.2 Solving False Negatives

Localization When a false negative diagnosis *d* is detected, both control and domain knowledge are analyzed. Control knowledge is responsible for the false negative when the module *m* containing *d* has not been visited when it should. This can happen by one of the following causes:

FN 1. No metarule adding *m* has been fired.
FN2. A metarule removing *m* has been fired.
EN3. A metarule adding m has been fired but execution has terminated before m has been visited.

Domain knowledge is responsible for the false negative when, assuming that *m* has been visited, *d* has not been deduced. This can happen by one of the following causes:

FN4. A rule required to deduce *d* has not been satisfied.

FN5. Threshold T has cut the deduction for *d.*

FN6. An up-down rule has decreased *d* certainty below T.

Causes related to control or domain knowledge can coexist. For causes requiring generalization, FN1 and FN4, the partial proof trees for the unsatisfied rules/metarules are computed and passed to the next phases. For the rest of cause types, the responsible rules/metarules are located and passed to the next phases.

Generation/Selection. Each cause type is treated as follows. Selected modifications satisfy AC1.

FN1. Unsatisfied conditions in the partial proof trees are tentatively generalized (OP1). One or several metarules adding *m* are inductively learned (OP6), taking as positive examples the set of false negatives that is being solved and as negative examples the set of false positives caused by the previous unsuccessful generalization.
FN2. Condidons in metarule *mr* removing *m* are specialized (OP1). Conditions are added to *mr* (OPS), taking as positive examples the true negatives in which *mr* has been fired, and

as negative examples the false negatives that are being solved.

FN3. The certainty degree of a metarule adding m is increased (OP2). Module m it is located at the beginning of the *add* part of the fired metarule (OP4).

FN4. Analogous to FN1 substituting metarules by rules.

FN5. The certainty value of the fact / responsible for threshold action is increased. This can be made by cither increasing the *cv* of rules concluding /(OP2) or increasing the *cv* of facts supporting / (OP2 or OP3). Fact / is identified as the deducible fact with lowest cv in *lhs(r),* being *r* the rule where the deduction has been cut.

FN6. The up-down rule r is smoothed making smaller the certainty subtraction (OP3). The *cv* of *d* before r is applied is increased, following the FN5 procedure. Rule *r* is specialized following the FN2 procedure.

5.3 Solving False Positives

Localization. When a false positive diagnosis *d* is detected, both control and domain knowledge are analyzed. Contrary to the false negative case, there arc not definite but tentative causes since there is no evidence of what rule/metarule should not been fired. Control knowledge is responsible for the false positive when the module *m* containing *d* has been visited but it should not. This can happen by the following causes:

FP1. A metarule adding *m* has been fired but it should not.

FP2. A metarule removing *m* has not been fired but it should.

FP3. Execution has terminated after visiting *m* but it should terminate before.

Domain knowledge is responsible for the false positive when, assuming that m hits been correctly visited, *d* should not been deduced. This can happen by the following causes:

FP4. Rules required to deduce *d* have been fired.

FP5. Certainty of rules used to deduce *d* is too high to be cut by x.

FP6. An up-down rule has increased *d* certainty.

Ciiven that no definite evidence of the actual cause exists (except for FP6), all the possible causes for a false positive are considered. For the cause IP2 requiring generalization, the partial proof trees requiring a minimum number of assumptions are computed and passed to next phases. For the rest of causes, responsible rules are located and passed to the next phases.

Generation/Selection. Each cause type is treated as follows. Selected modifications satisfy AC2.

FP1. Conditions in metarules *mr* adding *m* are specialized (OP1). Each metarule *mr* is specialized (OP4), taking as positive examples the true positives in which *mr* has been fired, and as negative examples the false positives that are being solved.

FP2. Unsatisfied conditions in the partial proof trees are tentatively generalized (OPI). One or several metarules removing m are inductively learned (OP6), taking as positive examples the set of false positives that is being solved and as negative examples the set of true positives caused by the previous unsuccessful generalization.

FP3. The certainty degree of metarules adding *m* is decreased (OP2). Module m is located at the end of the *add* part of the fired metarule (OP4).

FP4. Analogous to FP1 substituting metarules by rules.

FP5. Certainty degree of rules involved in the deduction is decreased (OP2).

FP6. When an up-down rule r has increased the cv of a false positive it is specialized following the FP1 procedure.

5.4 Solving Ordering Mismatches

Localization. When an ordering mismatch is detected only domain knowledge is analyzed because control knowledge has no effect in cvs of diagnoses. It can happen by one of the following causes:

OMI. Certainty degree of rules used to deduce *d* is too high/too low.

OM2. An up-down rule has incorrectly increased/decreased *d* ceruiinty.

Generation/Selection. Each cause type is treated as follows. Selected modifications satisfy AC3.

OMI. Certainty degree of rules involved in the deduction of *d* is decreased/increased (OP2).

OM2. The up-down rule *r* is smoothed making smaller the certainty addition/subtraction (OP3). The cv of *d* before *r* is applied is decreased/increased, following the OMI procedure

6 Empirical Results

We have used IMPROVER to validate PNEUMON-IA, an ES composed of 500 facts, 600 rules, 100 metarules and 24 modules. The case library is composed of 66 cases. We have the recommendations of *five* independent experts for every case in the library. Using the consensus function (section 5.1), we have obtained the gold standard for every case, that has been used by IMPROVER to refine PNEUMON-IA.

To evaluate the performance level of PNEUMON-IA before and after the refinement process, we compare it against human expert competence. Thus, we have matched the recommendations of the five independent experts against the gold standard for every case, obtaining the number of false negatives, false positives and ordering mismatches that each expert has performed with respect to the gold standard. We remind that the gold standard was computed as a consensus among the opinions of these experts. We also matched the recommendations of PNEUMON-IA, before and alter the use of IMPROVER, against the gold standard. The results of these comparisons, summarized for all the cases, are recorded in table 1.

Regarding false negatives, PNEUMON-IA (before) surpasses in performance to a human expert only (expert 4), while the other four experts exhibit a better performance. However, PNEUMON-IA (after) surpasses in performance *to all the human experts in the most important error type.* This result shows clearly the usefulness and power of IMPROVER, as well as the quality of the knowledge contained in PNEUMON-IA. Regarding false positives, all the experts surpass in performance to both PNEUMON-IA (before) and PNEUMON-IA (after), which decreases 11 false positives. Experts perform a low number of false positives *at the expenses of a* high number of false negatives (for example, see expert 4). On the contrary, PNEUMON-IA

	E1	E2	E3	E4	E5	PN (before)	PN (after)
#FN	69	60	57	118	67	85	42
#FP	38	32	45	18	31	87	76
#OM	25	31	23	11	20	13	18
#DIAG	187	190	206	118	182	222	252

Table 1. Comparison among five human experts (E1-E5) and PNEUMON-IA (PN) before and after refinement. #FN, #FP,#OM and #DIAG stand for the number of false negatives, false positives, ordering mismatches and total number of diagnoses, considering the whole case library.

follows the classification of error importance. Regarding ordering mismatches, both PNEUMON-IA (before) and PNEUMON-IA (after) surpass in performance to all experts except for expert 4. The number of ordering mismatches increases in 5 during the refinement process as a consequence of solving more important errors, specifically false negatives. This is a good example of how the number of errors of a given type may increase at the expenses of solving other errors, resulting in a better ES. In summary, PNEUMON-IA after refinement is clearly more valid than before, what evidences that KB refinement is a very valuable technique for ES validation.

Considering computational complexity, it has been proved [Valtorta 91] that rule base refinement is exponential in the worst case. Empirical results indicate that, under some assumptions, this problem is tractable for medium-size KB. IMPROVER has required 4, 16 and 7 CPU hours on a SIIN-4/260, for solving false negatives, false positives, and ordering mismatches respectively.

7 Conclusions

From this work we extract the following conclusions. First, KB refinement techniques can be effectively used to support ES validation. Second, when used for validation purposes, KB refinement should be guided by error importance with respect to the ES task, in order to assure a neat validity gain. In addition to satisfy acceptance criteria, final acceptance of refinements depends on the expert responsible for ES development. Third, when solving an error, KB refinement must consider every kind of knowledge that could be responsible for it. And four, using some heuristic assumptions to limit the number of modifications, refinement is computationally feasible for medium-size KB.

Acknowledgements

I thank Ram6n L6pez de Mantaras and Enric Plaza for reading a previous version of this paper and providing many useful comments. I specially thank Dr. Albert Verdaguer for his support on evaluating refinements, and Romero Donlo for her collaboration on writing this paper.

References

Buchanan B.G., Shortliffe E.H. (1984). Rule-Based Expert Systems. The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley.

Cain T. (1991). The DUCTOR: A Theory Revision System for Propositional Domains. Proceedings of the 8th International Workshop on Machine Learning, 485-489.

Craw S, Sleeman D. (1990). Automating the Refinement of Knowledge-Based Systems. Proceedings of the 9th European Conference on Artificial Intelligence, 167-172.

Gasching J., Klahr P., Pople H., Shortliffe E., Terry A. (1983). Evaluation of Expert Systems: Issues and Case Studies. Building Expert Systems, Hayes-Roth, Waterman and Lenat eds., Addison-Wesley.

Ginsberg A., Weiss S., Politakis P. (1985). SEEK2: a Generalized Approach to Automatic Knowledge Base Refinement. Proceedings of the 9th International Joint Conference on Artificial Intelligence, 367-374.

Ginsberg A. (1988). Theory Revision via Prior Operationalization. Proceedings of the 7th National Conference on Artificial Intelligence, 590-595.

Krause P., O'Neil M., Glowinski A. (1991). Can we Formally Specify a Medical Decision Support System?. Proceedings of the European Workshop on Verification, Validation and Testing ofKBS, 247-258.

McDermott J. (1981). RI: The Formative Years, AI Magazine, 2:21-29.

Meseguer P. (1991) Verification of Multi-Level Rule-Based Expert Systems. Proceedings of the 9th National Conference on Artificial Intelligence, 323-328.

Meseguer P. (1992). Incremental Verification of Rule-Based Expert Systems. Proceedings of the 10th European Conference on Artificial Intelligence, 840-844.

O'Keefe R.M., Balci O., Smith E. (1987). Validating Expert System Performance, IEEE Expert, vol 2, num 4, 81-89.

Ourston D., Mooney R.J. (1990). Changing the Rules: A Comprehensive Approach to Theory Refinement. Proceedings of the 8th National Conference on Artificial Intelligence, 815-820.

Politakis P., Weiss S.M. (1984). Using Empirical Analysis lo Refine Expert System Knowledge Bases. Artificial Intelligence, vol 22, 23-48.

Rushby J. (1988). Quality Measures and Assurance for AI Software. SRI-CSL-88-7R, SRI International.

Sierra C. (1989). MILORD: Arquitectura multinivell per a sistemes experts en classificacio. PhD Thesis, Universitat Politecnica de Catalunya.

Valtorta M. (1991). Some results on the computational complexity of refining confidence factors. International Journal of Approximate Reasoning, 5(2).

Verdaguer A. (1989). Pneumon-ia: Desenvolupament i validacio d'un si sterna expert d'ajuda al diagnostic medic. PhD Thesis, Universitat Autonoma de Barcelona.

Wilkins D.C., Buchanan B.C. (1986). On Debugging Rule Sets when Reasoning Under Uncertainly. Proceedings of the 5th National Conference on Artificial Intelligence, 448-454.