System Refinement in Practice – Using a Formal Method to Modify Real–Life Knowledge *

Rainer Knauf and **Ilka Philippow**
Ilmenau Technical University
Faculty of Computer Science and Automation
PO Box 10 05 65, 98684 Ilmenau, Germany
rainer.knauf@tu-ilmenau.de

Avelino J. Gonzalez
School of Electrical Engineering
and Computer Science
University of Central Florida
Orlando, FL 32816-2450, USA

Klaus P. Jantke
German Research Center
for Artificial Intelligence Ltd.
Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany

Dirk Salecker
Cocomore AG
Solmstr. 18, 60486 Frankfurt, Germany

Abstract

The pros and cons of formal methods are the subject of many discussions in Artificial Intelligence (AI). Here, the authors describe a formal method that aims at system refinement based on the results of a test case validation technology for rule-based systems. This technique provides sufficient information to estimate the validity of each single rule. Validity in this context is estimated by evaluating the test cases that used the considered rule. The objective is to overcome the particular invalidities that are revealed by the validation process. System refinement has to be set into the context of "learning by examples". Classical approaches are often not useful for system refinement in practice. They often lead to a knowledge base containing rules that are difficult to interpret by domain experts. The refinement process presented here is characterized by (1) using human expertise that also is a product of the validation technique and (2) keeping as much as possible of the (original) knowledge base. This is a way to avoid the drawbacks of other approaches and to enjoy the benefits of formal methods nevertheless. The validation process provides "better solutions" for test cases that have a solution which received a bad validity assessment by the validating experts. This knowledge is utilized by a formal reduction system. It reconstructs the rule set in a manner that provides the best rated solution for the entire test case set.

Introduction

The necessity of Validation and Verification of intelligent systems (V&V) is a result of the fact that today's information technology becomes more and more complex and less controllable. Its influence on our private, business, and public life gives a social dimension to the issue of its reliability. Since there are more and more safety–critical applications, invalid systems can be a real threat. The daily news are full of messages about the impact of non–valid systems: aircrafts that are out of legal control, medical devices that kill patients, alarm systems that don't realize an upcoming danger, ...

As a result of this insight, the authors focused the validation portion of V&V and developed a test case–based methodology for validation of rule based AI systems (see figure 1 for an overview and (Knauf 2000), for a detailed description). The developed technology covers five steps: (1) test case generation, (2) test case experimentation, (3) evaluation, (4) validity assessment, and (5) system refinement. These steps can be performed iteratively, where the process can be conducted again after the improvements have been made.

The validity assessment step leads to different validity degrees that express validity depending on it's purpose: validities associated with outputs, rules, and test data as well as a global system's validity.

Based on these validities, the last step leads to a new, restructured rule base that maps the test case set exactly to the solution that obtained the best rating from the expert panel in the validation session. Thus, the more the test case set is representative of the domain, the more the system refinement technology leads to a correct model of reality.

System refinement based on "better knowledge" provided in practice (or by experts, here) has to be considered in the context of "learning by examples". There are plenty of formal learning approaches that solve tasks like this. Usually, they aim at developing rules that map test data with "known" solutions (examples) to their correct solution.[1] They don't claim that all non–examples[2] are mapped correctly. Unfortunately, formal methods like these lead to rules that might reflect reality fairly well, but are not "readable" (or rather, "interpretable") by domain experts. Even worse, they might construct rules that reflect the examples correctly, but are wrong with respect to the causal connection they express. For example, if all examples that are used to construct medical diagnosis

[1] This property is called consistency by the machine learning community.

[2] Usually, the most possible cases that can occur in practice are not examples.

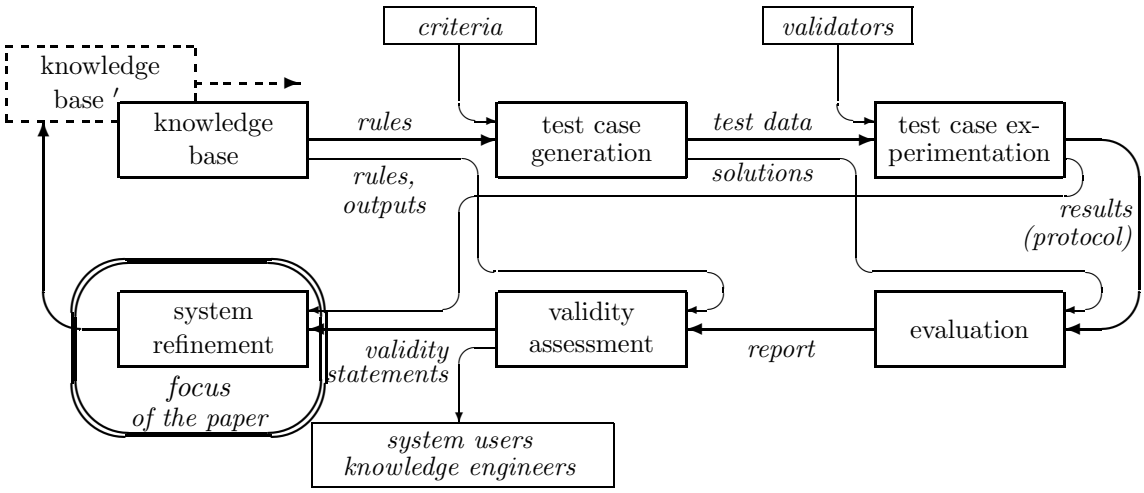

Figure 1: Steps in the Proposed Validation Process

rules, consider patients that have a properties like (A) *blue eyes* and (B) an *increased blood pressure* suffer from (C) *peptic ulcer*, the rule construction technique might produce a rule that implies from (A) and (B) to (C): $A \wedge B \rightarrow C$.[3]

To avoid this drawback, system refinement techniques should be based upon two fundamental assumptions (cf. (Ackermann, Fowler, and Ebenau 1984), (Adrion, Branstadt, and Cherniavsky 1982)):

1. It is assumed that the initial rule base was written with the intention of being correct, and if it is not correct, then a close variant of it is.
2. It is assumed that each component of the rule base appears there for some reason. Therefore, if a rule is invalid, it cannot just be removed. Rather, we have either
   (a) to find out its reason for being in the rule base, and find one (or several) alternative rule(s) that will satisfy that reason after the incorrect rule has been discarded or
   (b) to develop ideas how to modify the rule with the objective of improvement.

Thus, the present technique tries to change the rule base as little as possible and to provide a substitute for each piece of the knowledge base that will be removed.

The main idea of the refinement technique developed here is to find rules that are "guilty" in the system's invalidity and to replace them by "better rules". A rule is "better", if it leads to a solution that received "better marks" from the experts (the validation panel) than the system's solution.

Here, we introduce the refinement strategy and discuss the developed technique in the context of the assumptions above. Finally, we generalize these insights.

[3]As far as the authors know, at least the eye color is probably not a reason for peptic ulcer.

The Developed Refinement Strategy

Context Conditions

The developed refinement strategy is part of the validation technique described in (Knauf 2000). This is a complete 5–step methodology that can be performed iteratively. Each cycle begins with a knowledge base (KB) and ends up with a (hopefully) "better" KB, i.e. a more valid one.[4]

System refinement is the last of these five steps and closes the loop of each cycle. It is the preceding step (validity assessment) that provides different validity statements according to their purpose: validities associated with outputs, rules, and test cases, as well as a final, total validity of the entire system.

The main idea of the technique presented here is that we know something about (test) cases where the system performed these cases not adequately. And we know much more: The experts also solved the test cases and they even rated the solutions provided by humans within the TURING test experimentation. Therefore, we might have "better solutions" for test cases for which the system under examination received "bad marks". This knowledge is used for system refinement.

The Refinement Technique

The proposed technique consists of the following steps:

1. First, those rules that are guilty in the system's invalid behavior are discovered. Since our validation assessment (step 4 of the entire technique) provides validity statements that are associated with (final) system's outputs, we look for the "guilty rules" by considering rules that have final outputs as their

[4]Of course, validity is a property that can only be estimated with respect to the used methodology in general and with respect to the used test case generation method and the involved validating expert panel in particular.

then–parts. If such a final output revealed as invalid for the associated system's input, we call this rule "guilty".

The main idea is to analyze, which test cases used which rules and which validity degree has been associated to the system's solution to the test data of these cases. Furthermore, this step includes the generation of an "optimal solution" for each of these test data, which can either be the system's solution or a solution presented by a human expert.

2. Next, the simple situation is considered that all test cases using a guilty rule have the same optimal solution. Here, "repair" consists in simply substituting this invalid *then*–part by this optimal solution.
3. In the third step, guilty rules with various optimal solutions for the test cases using the rule are considered. Here, a reduction system is developed that systematically constructs one or some new rule(s) as a substitute for the guilty rule.
4. The resulting new rules are (still) "one–shot rules", i.e. they infer directly from system's inputs to system's outputs. To utilize the pre–compiled knowledge in the knowledge base, which occurs as rules with an intermediate hypothesis as their *then*–parts, the new rule(s) are re–compiled in a fourth step.

   Furthermore, in this step the resulting knowledge base will be inspected for rules, which can never be used, because their *then*–part is an intermediate hypothesis, which is not needed after the rule base refinement. These rules will be removed.

The last step does not change anything in the input–output behavior of the resulting rule base. It just makes the rule base more compact by reducing the number of rules. It also makes it easier to understand and to interpret the rules by humans.[5]

Finding "Guilty Rules" All rules having a conclusion part that is a final solution sol_k, are the subject of the considerations. Based on the results of the previous steps of the validation technique (Knauf 2000), we are able to compute the following:

1. For each rule r_l, there is a validity associated with this rule $v(r_l) = \frac{1}{|T_l|} \sum_{[t_j, sol_k] \in T_l} v_{sys}(t_j)$. Here, T_l is the subset of test cases that used the rule r_l, $v_{sys}(t_j)$ is a validity degree of the system for a test case t_j as computed in a previous step.

   v_{sys} is a number that ranges between 0 (totally invalid) and 1 (totally valid) in the eyes of the validation panel.
2. There is a set T_l^* containing all test cases with test data parts occurring in T_l and all solution parts, which came up in the experimentation, regardless of whether the solution is given by an expert or the system: $T_l^* = T_l \cup \{[t_j, sol(e_i, t_j)] : \exists [t_j, sol_k] \in T_l\}$
3. Next, T_l^* is split according to the different solution parts $sol_1, \ldots, sol_p, \ldots, sol_m$ of the test cases in T_l^*. This leads to m disjoint subsets $T_{li}^* \subseteq T_l^*$ $T_{l1}^*, \ldots, T_{lp}^*, \ldots, T_{lm}^*$. One of the subsets contains the test cases with the system's solution sol_k.
4. Analogously to $v_{sys}(sol_k)$, a validity $v(r_l, sol_p)$ $(1 \leq p \leq m)$ of each solution sol_p can be computed: $v(r_l, sol_p) = \frac{1}{|T_{lp}^*|} \sum_{[t_j, sol_p] \in T_{lp}^*} \frac{1}{\sum_{i=1}^{n} (cpt(e_i, t_j) \cdot c_{ijq})} \cdot \sum_{i=1}^{n} (cpt(e_i, t_j) \cdot c_{ijq} \cdot r_{ijq})$. Here, the c_i and r_i are the certainties and ratings provided by the experts during the experimentation session and $cpt(e_i, t_j)$ is the estimated competence of the expert e_i for a test case t_j as proposed in (Knauf 2000).
5. The "optimal validity" $v_{opt}(r_l)$ of a rule r_l is the maximum of all $v(r_l, sol_p)$ among the solutions sol_p occurring in T_l^*. The associated solution is the optimal solution sol_{opt} of r_l: $v_{opt}(r_l, sol_{opt}) = max(\{v(r_l, sol_1) : 1 \leq i \leq m\})$ $v_{opt}(r_l)$ is an upper limit of the rule-associated validity of r_l.

   If $v_{opt}(r_l, sol_{opt}) > v(r_l)$, there is a solution within T_l^* which got better marks from the experts than the system's solution. In this case, r_l is a guilty rule.

Simple Refinement by Conclusion Replacement If all test cases within T_l of a guilty rule r_l have the same optimal solution sol_k, which was different from the system's solution, the conclusion-part of this rule has to be substituted by sol_k.

Replacing the if–part of the Remaining Guilty Rules is performed by the following technique:

1. T_l of the rule r_l is split into subsets T_l^s $(1 \leq s \leq n)$ according to the solution sol_s for each t_j that obtained the highest validity $v(r_l, sol_s)$. The new if–part(s) of the new rule(s) instead of r_l are expressions $e_i \in E$ of a set of p new alternative rules $\{r_l^1, r_l^2, \ldots, r_l^p\}$ for each T_l^s and will be noted as a set of sets $P_l^s = \{\{e_1^1, \ldots, e_{p_1}^1\}, \ldots, \{e_1^p, \ldots, e_{p_p}^p\}\}$. The corresponding rule set of P_l^s is
   $r_l^1 : \bigwedge_{i=1}^{p_1} e_i^1 \rightarrow sol_s \quad \ldots \quad r_l^p : \bigwedge_{i=1}^{p_p} e_i^p \rightarrow sol_s$
2. Pos is the set of Positions (dimensions of the input space), at which the input data $t_j \in \pi_{inp}(T_l^s)$ of the test cases $t_j \in T_l^s$ are **not** identical. The generation of the if-parts P_l^s is managed by a *Reduction System*, which is applied to Triples $[T_l^s, Pos, P_l^s]$ until Pos becomes the empty set $\emptyset$.
3. The starting point of the reduction is $[T_l^s, Pos, P_l^s]$ with $P_l^s = \{\{(s_1 = s_1^{ident}), \ldots, (s_q = s_q^{ident})\}\}$. $s_1, \ldots, s_q$ are those positions where all test data $t_j \in \pi_{inp}(T_l^s)$ have the same (identical) value s_i^{ident} and again, Pos is the set of the remaining positions.

[5] This is an important issue to make the method reliable and acceptable for subject matter experts.

The reduction rules that are applied to these Triples are shown in table 1. Some detailed explanation can be found in (Knauf 2000), e.g.

Recompiling the new rules and removing the unused rules is performed by considering two cases:

First, if the if-part of a new rule contains a subset of expressions that is the complete if-part of another rule having an intermediate solution as its *then*-part, this subset is replaced by the corresponding intermediate solution:

$$\begin{array}{l} \exists r_i : (if\text{-part}_1 \rightarrow int_1) \\ \exists r_j : (if\text{-part}_1 \wedge if\text{-part}_2 \rightarrow \textit{int-or-sol}) \quad \Rightarrow \\ r_j : (if\text{-part}_1 \wedge if\text{-part}_2 \rightarrow \textit{int-or-sol}) \hookrightarrow \\ \qquad\qquad (int_1 \wedge if\text{-part}_2 \rightarrow \textit{int-or-sol}) \end{array}$$

Second, we remove rules that having an intermediate hypothesis as its *then*-part, which is not used in any if-part of any rule:

$$\begin{array}{l} \exists r_i : (if\text{-part}_1 \rightarrow int_1) \\ \neg\exists r_j : (int_1 \wedge if\text{-part}_2 \rightarrow \textit{int-or-sol}) \quad \Rightarrow \\ r_i : (if\text{-part}_1 \rightarrow int_1) \;\hookrightarrow\; \emptyset \end{array}$$

The Technique in the Context of the Assumptions

The technique introduced here tries to follow the ideas of (Ackermann, Fowler, and Ebenau 1984) and (Adrion, Branstadt, and Cherniavsky 1982) that are originally developed for use in classical software validation and mentioned here as an introduction.

Whenever a rule is indicated as "guilty" in some invalid system behavior, it will be changed as slightly as it can be to map the examples consistently. The basic idea behind the approach is to keep this rule and change it in a manner that the counter–examples, i.e. the test cases that have been solved incorrectly by the considered rule, will be excluded from using this rule.

To handle these counter–examples, some extra rules will be computed, which map these test cases to their correct solution.[6] Since our only hint about the present kind of invalidity is some better final solution to the examined test cases, the rule reconstructing technique focuses on rules that have final solutions as their conclusion parts.

In a first setting, the upcoming new rules infer directly from the system's inputs to the system's outputs. On one hand, these rules secure a correct input–output behavior of the system. On the other hand, such rules tend to be non–readable and non–interpretable in the context of the rest of the knowledge base. Usually, they tend to have many expressions in their condition parts, i.e. they are very long.

To avoid this drawback and to minimize the number of rules by utilizing existing rules as pre–compiled knowledge, the computed rules are adapted to fit in the context of the rest of the rule base by using their conclusion–parts within the condition part of the upcoming rules.

In particular, if there is at least one test case that is performed correct by this rule, a **very** close variant of the original rule remains in the system. This variant is characterized by having the same conclusion part and a slightly changed (usually longer) condition part. These changes are due to the fact that the counter–examples, i.e. the examples with a known better solution, have to be excluded.

To summarize, the presented technique keeps as much as it can from each rule. This is performed by

1. reconstructing the rules in a manner such that they handle the counter–examples (and only them) differently from the original rule base[7] and
2. using as much as it can from the original rule base by "compiling" the new rules together with the original ones in the knowledge base.

The latter issue utilizes correlations between the former (human–made) and the upcoming (artificially computed) pieces of knowledge.

The authors believe that this is the right thing to do, because it sets the new knowledge in the context with the original one. This way, the modified rule base should be more easily to be interpreted by human experts (i.e. people, who expressed the original knowledge) and additionally, the knowledge base will be optimal with respect to its size, i.e. both the number of rules and their length.

Conclusion

The authors feel that the presented technique is a general way to refine AI systems in particular, and technical systems that contain knowledge in general.

Engineers who develop and refine any technical system usually say "*Never change a working system.*". Here, we extend this point of view by postulating the following three general issues:

1. Don't change a system that works well.
2. Do change a system that works almost well as slightly as you can. Keep as much as you can to avoid the risk making things worse and just change those parts that handle the particular invalidities.
3. Do not try to refine a system that is invalid for many cases or for extraordinary important cases (safety critical ones, for example). Here, some thought to rebuild either the system or even the general approach should be invested.

[6] In the context of validation, we can define "correct" just by "*There is no different man–made solution that obtained better marks by the validation panel.*" More can't be done.

[7] Without effecting the basic message of the present paper, here we should "admit", that this is a simplified description of the truth. Especially in case of non–discrete input data (numbers ranging quasi continuously between some lower and upper possible value, for example) the situation occurs slightly more complicated. For details, see (Knauf 2000).

Table 1: Reduction rules to construct better rules systematically

Reduction rules

R1 • $pos \in Pos$, s_{pos} has a value set with no well-defined $\leq$ relation
• $\{s^1_{pos}, \ldots, s^m_{pos}\}$ are the values of s_{pos} occurring in T^s_l $\Rightarrow$

$$[T^s_l, Pos, \{p_1, \ldots, p_n\}] \hookrightarrow$$

1. $[T^{s,1}_l \setminus \{[t_j, sol_s] \in T^s_l : s_{pos} \neq s^1_{pos}\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s^1_{pos})\}]$

2. $[T^{s,2}_l \setminus \{[t_j, sol_s] \in T^s_l : s_{pos} \neq s^2_{pos}\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s^2_{pos})\}]$

• • •

m. $[T^{s,m}_l \setminus \{[t_j, sol_s] \in T^s_l : s_{pos} \neq s^m_{pos}\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s^m_{pos})\}]$

Continue with each $T^{s,i}_l$ $(1 \leq i \leq m)$ separately.

R2 • $pos \in Pos$, s_{pos} has a value set with a well-defined $\leq$-relation
• s^{min}_{pos} is the smallest value of s_{pos} within T^s_l
• s^{max}_{pos} is the largest value of s_{pos} within T^s_l $\Rightarrow$

$$[T^s_l, Pos, \{p_1, \ldots, p_n\}] \quad \hookrightarrow \quad [T^s_l, Pos \setminus \{pos\}, \textstyle\bigcup_{i=1}^{n} p_i \cup \{(s_{pos} \geq s^{min}_{pos}), (s_{pos} \leq s^{max}_{pos})\} \cup S_{excl}]$$

S_{excl} is the set of excluded values for s_{pos}, which have to mapped to a solution different from sol_s because of belonging to some other T^v_u with $v \neq s$:

$$S_{excl} = \{(s_{pos} \neq s^j_{pos}) : \ \exists [t_j, sol_s] \in T^s_l \ \exists [t_m, sol_v] \in T^v_u (v \neq s)$$
$$\text{with } \forall p \neq pos \ ((s^j_p = s^m_p) \ \text{ and } \ (s^{min}_{pos} < s^m_{pos} < s^{max}_{pos})) \ \}$$

The refinement strategy presented here falls into the second class, i.e. it provides a formal method to "repair" a working system with the aim of handling singular invalid (test) cases correct in future and to keep the former input/output behavior for all other cases.

Since the test cases can be considered as examples, i.e. as input/output pairs with a known correct output[8], the strategy should also be considered in the context of "learning by examples".

Frequently used technologies to perform tasks like these (ID3 (Quinlan 1983), for example) aim at producing a rule set that classifies the examples correctly. On the one hand, they enjoy consistency with the examples, but on the other hand, they suffer from being somehow "artificially constructed" and are not interpretable by domain experts.

Moreover, the risk that such rules reflect the reality wrong is much higher than by using techniques such as the one presented here. This is because the "non–examples" (i.e. all test data that can occur in practice but are not a member of the example set respectively the test case set, in our approach) are used to optimize the upcoming rule set. The inputs that are not mentioned in any example (that are not a test case, in our setting) are mapped to *any* output that is "useful" with respect to some optimization issue (the number or the length of the constructed rules, for example).

[8]Again, correctness here means validity and is nothing more and nothing less than a behavior that obtained "good marks" by some (human) validation panel.

Our technique, on the other hand, is based on the assumption, that all "non–examples" are handled correctly by the former knowledge base. It doesn't change the behavior for cases that have not been examined as test cases within the validation technology. Since the historic knowledge base is a product of human thought, the probability that these "non–examples" are handled correctly is much higher.

To summarize, we feel that a competent knowledge engineer assumption should be the basis to handle cases that are not shown to be performed invalid by the AI system.

References

Ackermann; Fowler and Ebenau 1984. *Software Validation.* Amsterdam: Elsevier, 1984.

Adrion, W. & Branstadt, M. & Cherniavsky, J. 1982. Validation, verification and testing of computer software. *ACM Computing Surveys*, vol. 14, number 2, pp. 159–182, 1982.

Knauf, R. 2000. *Validating Rule–Based Systems – A Complete Methodology.* Habilitation Thesis, Ilmenau Technical University, Faculty of Computer Science and Automation, Aachen: Shaker, 2000.

Quinlan, J. (1983). Learning efficient classification procedures and their application to chess end games. Michalsky et.al. (eds.): *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, CA: Tioga Publishing Corp., 1983.