Advanced Measures for Empirical Testing

Joachim Baumeister
Institute of Computer Science, University of Würzburg, Germany
joba@uni-wuerzburg.de

Abstract

Empirical testing is still the most popular evaluation method for the development of intelligent systems. Besides classic forms of boolean testing of occurring solutions for a given set of findings more refined methods are required for a thorough evaluation of real world knowledge systems. We present extended precision and recall functions for interactive knowledge systems that are generalizations of the existing measures. Additionally, we propose a visualization method for inspecting the validation result for interactive systems. A case study with a second opinion system from the medical domain demonstrates the usefulness of the approach.

Introduction

In the context of quality management of (intelligent) systems we see that the empirical testing technique still denotes the most important and most frequently applied method. Empirical testing is simple and effective: previously solved test cases with correct results are given as input to the system and the derived results are compared with the expected results that are given in the test cases. The derivation quality is typically measured by precision/recall or the combining F-measure.

As the original versions of these measures are sufficient for many evaluation tasks we motivate that sometimes more advanced versions of the precision and recall are needed to meet the requirements of the evaluation process. We propose two extensions:

1. The *rated precision/recall* that is able to compare solution states rather than the usual boolean occurrence of states *derived/not derived*.
2. The *chained precision/recall* that not only takes into account the final solutions of a case but also the intermediate solutions during a problem-solving process and is able to weight intermediate solutions in comparison to the final solution.

The paper is organized as follows: We introduce the traditional evaluation measures precision and recall and show its extensions rated precision/recall and chained precision/recall. We show that every extension is a true generalization of the traditional measure. Thereafter, we introduce the visualization method DDTree that allows for an intuitive visualization of empirical test runs. The usefulness of these measures is demonstrated by a case study that was conducted during the evaluation of a medial second-opinion system for rescue missions. A discussion and outlook concludes the paper.

Empirical Testing with Sequential Test Cases

We first define basic notions that are used throughout the subsequent discussion of advanced testing measures.

Basic Notions

A (knowledge) system is typically defined by its possible input and output elements. In the context of intelligent systems a possible input is often called *finding* and a possible output is defined as a *solution*.

Definition 1 (Finding and Solution)**.** *Let I be the (universal) set of observable input values. A tuple $f = a : v$ is called a* finding*, where $a \in I$ is an input (attribute) and $v \in dom(a)$ is an assignable value. Let F be the universal set of findings, and let S be the universal set of output values, i.e.,* solutions *derivable by the knowledge system. In the simplest case a boolean value is assigned to a solution $s \in S$ in order to express the positive and negative derivation of the particular output. For the refined case a more expressive state range is assigned to s, for example to additionally represent a state of its possible derivation.*

Empirical testing usually runs a collection of test cases, where the expected results of each test case is known beforehand. Formally, a test case can be defined as follows.

Definition 2 (Test Case)**.** *A* test case tc *stores a list of findings and a collection of derived solutions:*

$$tc = \big((f_1, \ldots, f_p), (s_1, \ldots, s_q)\big)\,, \tag{1}$$

where $f_i \in F$ is a finding and $s_i \in S$ is a positively derived solution.

Since there is no order of the derived solutions in the test case every derived solution is equally important. Often it is beneficial to specify a more refined confirmation state of the particular solutions, for example, some solutions are only derived as possible outputs whereas other solutions are strongly derived as a suitable solution. For this reason we introduce the notion of a *rated test case*.

Definition 3 (Rated Test Case)**.** *A* rated test case rtc *stores a list of findings and a collection of rated solutions and is defined as follows:*

$$rtc = \big((f_1, \dots, f_p), (rs_1, \dots, rs_q)\big)\,, \tag{2}$$

where $f_i \in F$ is a finding and $rs_i = (s_i, r_i)$ is a rated solution, i.e., a rating r_i assigned to a solution $s_i \in S$. The specified ratings r_i are expected to be derived by a valid knowledge base when entering the findings f_i given in rtc.

Possible domains for ratings are real values in $[0, 1]$, for example to represent probabilities, but also symbolic values like $\{$undefined, excluded, suggested, established$\}$. It is easy to see that for a rating $r_i = \textit{established}$ $(\forall i = 1, \dots, q)$ a rated test case collapses to a standard test case.

Although the use of rated test cases improves the testing possibilities it is often not sufficient to test the derivation quality of the knowledge base *at the end* of each test case, but also to test the derivation state *during* the execution of a test case. In order to enable this type of testing we partition the test case into a sequence of (partial) test cases, where each partial test case stores its findings entered in this phase and the solutions (with ratings) derived so far. More formally we introduce the notion of a *sequential test case*.

Definition 4 (Sequential Test Case)**.** *A* sequential test case seq *is defined as a list of rated test cases rtc_i*

$$stc = (rtc_1, \dots, rtc_n)\,,$$

where a rated test case rtc_i depends on its predecessors rtc_j with $j < i$.

We see that a sequential test case partitions a standard test case into distinct rated test cases, where every case contains an ordered list of findings $f_{i,j}$ that are supposed to be entered by a user in the given order. Additionally, a rated test case stores a set of solutions $s_{i,j}$ with their corresponding ratings that are expected to be derived by a valid knowledge system after entering the findings $\cup_{k=1,\dots,j} f_{i,k}$, i.e., the findings $f_{i,j}$ and all preceding findings defined in the sequential test case.

It is worth noticing that the order of the finding sequences defined in a sequential test case is explicit and important. Thus, every sequence seq_i depends on its predecessor seq_{i-1}, especially with respect to the ratings of the particular solutions. The rating of solutions can also depend on findings that were entered in previous cases.

A sequential test case $stc = (rtc_1, \dots, rtc_n)$ is a generalization of a rated test case if the sequential test case contains only one finding sequence, i.e., for $n = 1$.

Simple Validation Measures

Traditionally, the quality of the derived solutions is computed using standard measures such as precision, recall, and the combined F-measure. In literature the measures simply compare the set of positively derived solutions der with the set of expected solutions exp.

Definition 5 (Precision)**.** *Let $tc = \big((f_1, \dots, f_p), exp\big)$ be a test case with the expected solutions $exp \subseteq S$ and let $der \subseteq S$ the set of actually derived solutions. Then, the precision of der and exp is defined as follows:*

$$precision(der, exp) = \begin{cases} \frac{|\, der \cap exp \,|}{|\, der \,|} & \textit{if } der \neq \{\}\,, \\ 1 & \textit{if } der = exp = \{\}\,, \\ 0 & \textit{otherwise.} \end{cases} \tag{3}$$

In summary, the precision measures how many of the derived solutions were actually expected to be derived by the case tc. Analogously, the recall of a test case is defined as follows.

Definition 6 (Recall)**.** *Let $tc = \big((f_1, \dots, f_p), exp\big)$ be a test case with the expected solutions $exp \subseteq S$ and let $der \subseteq S$ the set of actually derived solutions. Then, the recall of der and exp is defined as follows:*

$$recall(der, exp) = \begin{cases} |der \cap exp| \,\big/\, |exp| & \textit{if } exp \neq \{\}\,, \\ 1 & \textit{otherwise.} \end{cases} \tag{4}$$

The recall measures how many expected solutions were actually derived by the knowledge base. For multiple solutions it is usually interesting to provide a single metric that combines the precision and recall. Often, the *F-measure* is applied in such a context.

Definition 7 (F-Measure)**.** *Let $tc = \big((f_1, \dots, f_p), exp\big)$ be a test case with the expected solutions $exp \subseteq S$ and let $der \subseteq S$ the set of actually derived solutions. Then, the F-measure of der and exp is defined as follows:*

$$f_\beta(der, exp) = \\ = \frac{(\beta^2 + 1) \cdot precision(der, exp) \cdot recall(der, exp)}{\beta^2 \cdot precision(der, exp) + recall(der, exp)} \tag{5}$$

The F-measure computes the geometric mean of the precision and the recall of the derived solutions. Often, we use the f_1 measure, where precision and recall are equally weighted.

Extended Validation Measures

When using sequential test cases for empirical testing we need to take into account that we also have intermediate solutions defined in each finding sequence of a sequential test case. Furthermore, the traditional measures only perform a boolean check on the derived and expected solutions. Thus, only the (non-)derivation of the solutions is compared but not their actual rating. However, we often see a more precise rating of solutions, for example in Bayesian networks solutions are rated by probabilities $p \in [0, 1]$. In heuristic decision trees a solution can be derived either as *unclear*, *excluded*, *suggested* or *established*. In summary, a refined set of measures need to take the following into account:

1. Comparison of rated solutions instead of a boolean intersection.
2. Evaluate the quality of a chained case sequences instead of one single test case.

Concerning the first issue we introduce a "rated" version of the precision/recall measures that generalize the standard measures but is applicable to arbitrary solution ratings. We further extend the measures by a sequentialized version of precision/recall in order to handle the second issue.

Rated Precision/Recall

Definition 8 (Rated Precision)**.** *Let $exp_{rs} \subseteq S$ be the expected solutions of a rated test case and let $der_{rs} \subseteq S$ be the collection of actually derived solutions. Then, the rated precision is defined as*

$$precision_{rs}(der_{rs}, exp_{rs}) = \\ = \begin{cases} prec_{rs}(der_{rs}, exp_{rs}) & \textit{if } der_{rs} \neq \{\}\,, \\ 1 & \textit{if } der_{rs} = \{\} \textit{ and } exp_{rs} = \{\}\,, \\ 0 & \textit{otherwise,} \end{cases} \tag{6}$$

where the $prec_{rs}$ is defined by

$$prec_{rs}(der_{rs}, exp_{rs}) = \\ = \frac{\sum_{s \in \cap(der_{rs}, exp_{rs})} rsim\big(r(s, der_{rs}), r(s, exp_{rs})\big)}{\mid der_{rs} \mid}\,. \tag{7}$$

Instead of simply intersecting the set of derived solutions with the set of expected solutions we use the function $\cap(der_{rs}, exp_{rs})$ to extract all solutions s contained in both rated solutions sets by

$$\cap\,(der_{rs}, exp_{rs}) = \\ = \Big\{\, s \in S \,\big|\, \exists (s', r_d) \in der_{rs} \wedge \exists (s', r_e) \in exp_{rs} \wedge s = s' \,\Big\}\,. \tag{8}$$

For all these solutions s we compute a similarity between the rating of s in der_{rs} and the rating of s contained in exp_{rs} by using the function $rsim(r(s, der_{rs}), r(s, exp_{rs}))$, where $r(s, rs)$ yields the rating of solution s in the rating set rs:

$$r(s, rs) = \begin{cases} r & \textit{for } (s, r) \in rs\,, \\ 0 & \textit{else.} \end{cases} \tag{9}$$

The rated similarity function needs to be defined appropriately for every possible domain of ratings. It is important to notice that $rsim$ always has to return a value $v \in [0, 1]$. If not defined then we can simply use the individual similarity function

$$rsim_i\big(r(s, der_{rs}), r(s, exp_{rs})\big) = \\ = \begin{cases} 1 & \text{if } r(s, der_{rs}) = r(s, exp_{rs})\,, \\ 0 & \text{else.} \end{cases} \tag{10}$$

When using the individual similarity function the rated similarity reduces to a boolean comparison as already known fromt the standard precision measure (see Equation 3).

Example. In the following we give an example for a possible rated similarity function that could be used for symbolic ratings with the following domain R = {unclear, excluded, suggested, established}.

$$rsim\big(r(s, der_{rs}), r(s, exp_{rs})\big) = \\ = \begin{cases} 1 & \text{if } r(s, der_{rs}) = r(s, exp_{rs})\,, \\ 0.8 & \text{if } r(s, der_{rs}) = \text{suggested} \wedge \\ & \quad r(s, exp_{rs}) = \text{established}\,, \\ 0.5 & \text{if } r(s, der_{rs}) = \text{established} \wedge \\ & \quad r(s, exp_{rs}) = \text{suggested}\,, \\ 0 & \text{else.} \end{cases} \tag{11}$$

We can see that the function gives a better similarity when the expected result is better that actually derived. In some applications the counter-intuition may be appropriate.

Definition 9 (Rated Recall)**.** *Let $exp_{rs} \subseteq S$ be the expected solutions of a rated test case and let $der_{rs} \subseteq S$ be the collection of actually derived solutions. Then, the rated recall is defined as*

$$recall_{rs}(der_{rs}, exp_{rs}) = \\ = \begin{cases} rec_{rs}(der_{rs}, exp_{rs}) & \textit{if } exp \neq \{\}\,, \\ 1 & \textit{otherwise,} \end{cases} \tag{12}$$

where the rec_{rs} is defined by

$$prec_{rs}(der_{rs}, exp_{rs}) = \\ = \frac{\sum_{s \in \cap(der_{rs}, exp_{rs})} rsim\big(r(s, der_{rs}), r(s, exp_{rs})\big)}{\mid exp_{rs} \mid}\,.$$

As already introduced in the context of Definition 8 we reuse the function $\cap(der_{rs}, exp_{rs})$ defined in Equation 8 and the rated similarity function $rsim(\dots)$ discussed before. For the individual similarity function given in Equation 10 the rated recall $recall_{rs}$ is equivalent to the standard recall measure $recall$.

Chained Precision/Recall Based on the extensions of precision/recall made above we further generalize the measure to evaluate the quality of a sequential test case.

Definition 10 (Chained and Rated Precision)**.** *Let $stc = (rtc_1, \dots, rtc_n)$ be a sequential test case. Every rated test case rtc_i stores its expected solutions $exp_{i,rs}$ at the phase i of the test case stc. Accordingly, we define $der_{i,rs}$ to be the actually derived solutions in phase i. Then, we define the* chained and rated precision *for $DER_{rs} = (der_{1,rs}, \dots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \dots, exp_{n,rs})$ as follows:*

$$precision_{rs,c}(DER_{rs}, EXP_{rs}) = \\ = \frac{\sum_{i=1\dots n} wp(i) \cdot precision_{rs}(der_{i,rs}, exp_{i,rs})}{\sum_{i=1\dots n} wp(i)}\,, \tag{13}$$

where $wp(i) \in [0, 1]$ for all $i \in 1, \dots, n$ defines the weight of the intermediate solutions in phase i.

It is easy to see that for $wp(i) = 1$ and $n = 1$ the chained and rated precision $precision_{rs,c}$ yields the rated precision $precision_{rs}$ introduced in Definition 8.

The appropriate specification of the weights depends on the particular application domain. We see two typical possibilities to define the weights for the chained and rated precision:

- **Equi-Important:** The quality of the derived solutions is equally important for every phase, i.e., $wp(i) = 1$ for all $i = 1, \dots, n$.
- **Inverse-Annealing:** The quality of the derived solutions becomes more important in later phases. Then, we define $wp(i) = i/n$ for all $i = 1, \dots, n$.

The definition of the chained and rated recall is analogous to the definition of the chained and rated precision.

Definition 11 (Chained and Rated Recall)**.** *Let $stc = (rtc_1, \dots, rtc_n)$ be a sequential test case. Every rated test case rtc_i stores its expected solutions $exp_{i,rs}$ at the phase i of the test case stc. Accordingly, we define $der_{i,rs}$ to be the actually derived solutions in phase i. Then, we define the* chained and rated recall *for $DER_{rs} = (der_{1,rs}, \dots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \dots, exp_{n,rs})$ as follows:*

$$recall_{rs,c}(DER_{rs}, EXP_{rs}) = \\ = \frac{\sum_{i=1\dots n} wr(i) \cdot recall_{rs}(der_{i,rs}, exp_{i,rs})}{\sum_{i=1\dots n} wr(i)}\,, \quad (14)$$

where $wr(i) \in [0, 1]$ for all $i \in 1, \dots, n$ defines the weight of the intermediate solutions in phase i.

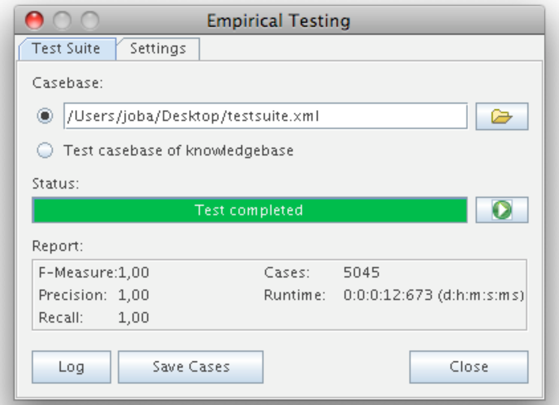
In the context of the chained and rated recall we are able to specify a distinct weighting function wr in order to define a different weighting scheme compared to the weighting of the computed precisions. However, often the same weighting function is used for wp and wr.

Testing Visualization with DDTrees

The analysis of a testing session can be improved by visualizing its outcomes. One possible way is the adaptation of the Unit-Testing metaphor that uses a colored bar indicating the overall outcome. While running the suite of test cases the color of the bar remains green until an error in a test case occurs. In consequence, a red bar shows that at least one failure has been reported in a test case. In Figure 1 the empirical testing tool of the knowledge development environment d3web.KnowME (Baumeister and others 2008) is shown. Here a medical knowledge base is tested against a test suite with 5045 test cases.

Although the metaphor allows for a quick and intuitive analysis of the overall result it lacks when errors occur and a deeper analysis becomes important. Often a debugging session, e.g., (Zacharias and Abecker 2007), of the erroneous test case is initiated, but the context of the test case with respect to similar (non erroneous test cases) is difficult to perceive.

In the past it has been proposed to visualize the test suite as a tree (Baumeister, Menge, and Puppe 2008). In this paper we revive the approach since it allows to interactively analyze and evaluate the validation results. For a knowledge system with an interview logic it also allows for the intuitive


Empirical Testing

Test Suite | Settings

Casebase:

● /Users/joba/Desktop/testsuite.xml

○ Test casebase of knowledgebase

Status:

Test completed

Report:

```
F-Measure:1,00          Cases:    5045
Precision:  1,00        Runtime:  0:0:0:12:673 (d:h:m:s:ms)
Recall:     1,00
```

Figure 1: Empirical testing of a knowledge base using the unit testing metaphor with green/red colored bars.

analysis of the dialog behavior, thus verifying the interview knowledge.

Introduction to DDTrees

In summary, a DDTree arranges the test cases of a test suite in a (poly-)tree. Every test case is represented by a path from the root to a leaf of the tree. A node of such a path contains the following information:

1. The previously asked question (for simpler reference)
2. The currently derived solutions ranked according their status
3. The currently asked question; indeed the node represents this currently active input

Every arc starting from a node and its currently active input i is labeled with a possible/allowed answer v of the input i. Thus, a node i and an outgoing arc with label v defines a possible finding $i : v$ contained in the test case.

Therefore, a test case with its intermediate results and its interview behavior is retrieved by navigating from the root of the tree to one leaf. The leaf usually contains only the previously asked input and the final solutions of the particular case.

An example DDTree is shown in Figure 2. For instance, input *Question 1* is initially asked; for finding *Question 1 : yes* the system derives the solutions *Solution 2* and *Solution 3* with 10 points, thereafter *Question 4* is asked. If this input is answered with *yes* then *Solution 2* is rated with 1009 points, whereas *Solution 3* remains at 10 points.

For larger DDTrees the entire tree is typically cut in equi-with subtrees in order to allow for a simple analysis.

The technique considers two important issues of the validation task: the indication/inspection of erroneous cases and the validation of new cases that emerged from the modification/extension of the knowledge base. In summary, a tree is generated from the suite of test cases. Correct cases and their arcs, respectively, are greyed-out, whereas the arcs of erroneous cases are highlighted in red color. Yet un-inspected

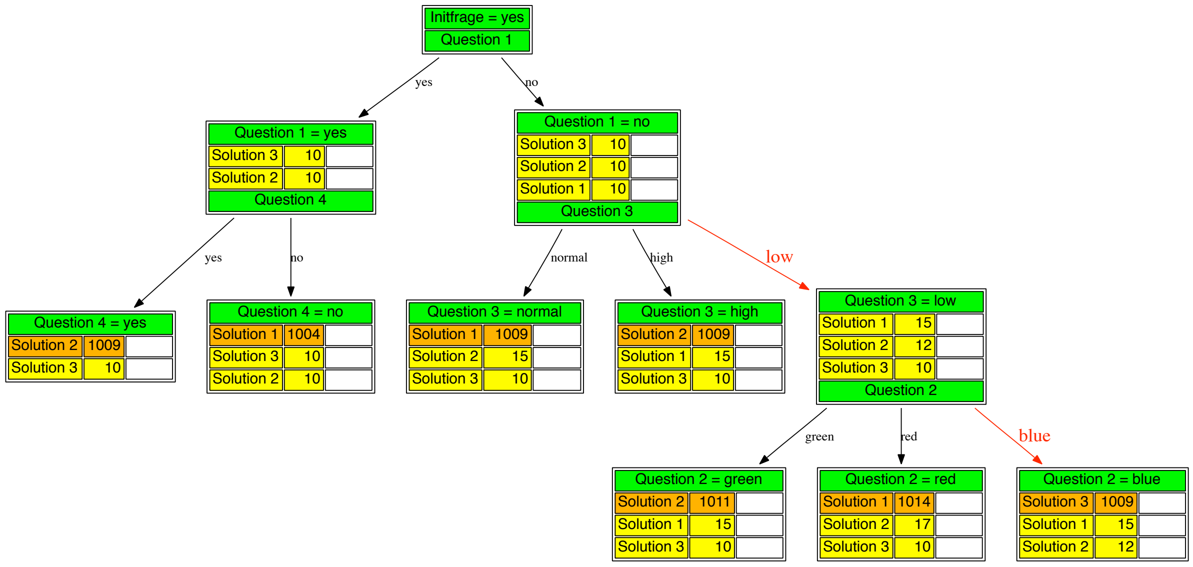


Figure 2: An example DDTree: Each test case is represented by a path from the root to a leaf of the tree. Erroneous cases are labeled with red arcs.

cases are not highlighted at all and printed normally. In this way, the developer can grasp the following tasks in a simple and intuitive manner:

1. **Acknowledge correct cases:** the developer easily identifies the correct cases since their arcs are greyed-out. Like the green bar shown in Figure 1 the DDNet approach gives an immediate feedback of the system's validity. The more grey the tree is drawn the more valid it appears to the developer.
2. **Verify new cases:** the developer inspects the new cases (not highlighted) and marks them, if correct. Otherwise, the knowledge base needs to be refined in an appropriate way.
3. **Analyze incorrect cases:** the developer needs to inspect incorrect cases that are highlighted in red color from the beginning of the incorrect behavior. The preceding and correct beginning of the case is not marked in red color. Thus, the sub-sequence of the erroneous case is simple to grasp. Since the adjacent and similar cases are also depicted in the tree, the context of the erroneous case is easy to understand.

Validation Process

The process of using DDTrees for evaluation is as follows:

1. **Initialization:** Create an initially empty collection of previously reviewed cases $PRC = \{\}$.
2. **Case Generation (optional):** If a sufficient test suite is not available, we propose to generate the total cover of test cases for a given knowledge base. Note that this step can imply the combinatorial enumeration of all possible finding values, and is therefore not applicable in general domains. However, it is quite appropriate in smaller domains and knowledge bases using a decision tree representation that restricts the meaningful combinations of findings. Alternatively, general methods for test case generation can be applied, e.g., (Gupta and Biegel 1990; Gonzalez and Dankel 1993; Knauf, Gonzalez, and Abel 2002).
   - All possible cases are recorded using an automated interview bot, i.e., we fill the set of recorded cases RC. The bot simulates an interactive dialog with the knowledge system by iteratively answering the possible values of the currently presented question. After providing an answer to the current question, the bot recursively retrieves the next follow-up question to be answered. A new case is stored if no follow-up question is asked by the system any more. During the simulation of an interview the bot also stores the intermediate solutions, that are derived during the problem-solving session.
3. **Visualization:** The test case suite is rendered using a rooted tree graph drawing algorithm, for example see (Sugiyama 2002). The arcs of new cases are not highlighted in the tree, if they were not recorded beforehand. These cases need to be manually inspected by the developer. The sequences of the correct and previously reviewed cases $c \in PRC \cap RC$ are greyed-out. The remaining cases were recorded previously but now show a different derivation at some point in the case. Starting from this point the cases are highlighted in red color in order to call attention to the incorrect behavior of the system in the context of this cases. The color of the arcs is computed by the rated/chained precision and recall measures that were defined in the previous section. For example the right branch of the DDTree shown in Figure 2 depicted in

red color.

4. **Manual review of the DDTree (optional):** Here, only previously unreviewed cases $c \notin RC$ need to be reviewed. Every unreviewed case, i.e., every path from the root to a leaf, is manually inspected by a domain specialist (not necessarily the developer of the knowledge base). For this step we recommend to print out the entire graph on a poster in order to obtain a better overview of the interview workflow. The classic review on a printed poster offers a couple of benefits especially for domain specialists not familiar with a concrete validation software. For example, already traversed and reviewed paths can be easily highlighted with a text marker without knowing a specific software.
5. **Storing the test suite:** If all reviewed cases are inspected successfully and are marked as *correct* by the domain specialist, then these cases are also stored in the test suite of "previously reviewed cases" PRC.
6. **Knowledge modification:** After changing the knowledge base, the previous steps are iterated starting with step 2. All previously reviewed cases – that have not changed in this iteration – are highlighted in the tree. Thus, the domain specialist intuitively identifies the new or changed paths in the tree that have to be reviewed in this iteration.

Erroneous cases are highlighted in the visualization in an orange color from the part of their erroneous behavior. The visualization of such a case directly corresponds to the extended precision and recall measures defined above.

As an advantage of this visualization the domain specialist can easily "see" the context of the current case he/she is inspecting, e.g., what will happen if the question is answered differently, and which solutions are still possible at this stage, etc. Furthermore, no computer skills are required; the specialist can concentrate on the domain knowledge and does not have to struggle with the keyboard/mouse.

Case Study

The presented work was successfully applied in the context of the development and evolution of the medical knowledge system *Digitalys CareMate* that is sold as a second-opinion system in medical rescue service. Currently the knowledge base of the system comprises about 200 findings indicating the derivation of 120 solutions. About 1500 rules were developed to implement the interview strategy as well as the rated derivation of the solutions. An improved version of the knowledge formalization pattern *heuristic decision tree* (Puppe 2000) was used to implement the system.

After a first review phase in March 2008, a final review meeting of the release candidate of the system was realized in July 2008 (lasting three days). The metaphor of the DDTree was perceived to be very intuitive for the domain specialist.

Further, the use of printed posters for inspecting the (large) sub-trees helped significantly during the evaluation phase. Since no computer was required the (almost unexperienced) domain specialist could start immediately to work with text marker and pen. The intuitive "user interface" was also beneficial for erroneous areas of the tree. For example, when identifying errors the specialist could simply write/draw some text/corrections on the paper, e.g., linking a question to another sub-tree by drawing the edge manually on the poster, making comments etc.

Discussion

For the development of intelligent systems *empirical testing* denotes one of the most popular evaluation methods today. In its classic form, the empirical test evaluates a collection of test cases using the measures precision and recall. However, these measures only cover the overall outcome of the case. We have motivated that the simple boolean evaluation function is not always appropriate for real world application, especially when erroneous cases should be inspected and refined in a interactive manner.

We introduced extended measures for precision and recall, and we described a visualization technique that is capable for an interactive analysis and validation. The presented work was successfully applied in a case study implementing the evaluation of the real-world medical system *Digitalys CareMate*. During the project we learned that the verication/validation step can be signicantly simplied by "analog methods" that help the experts to step away from the original system and to review the knowledge from a distinct perspective.

References

Baumeister, J., et al. 2008. The knowledge modeling environment d3web.KnowME. open-source at: http://d3web.sourceforge.net.

Baumeister, J.; Menge, M.; and Puppe, F. 2008. Visualization Techniques for the Evaluation of Knowledge Systems. In *FLAIRS'08: Proceedings of the 21th International Florida Artificial Intelligence Research Society Conference*, 329–334. AAAI Press.

Gonzalez, A. J., and Dankel, D. D. 1993. *The Engineering of Knowledge–Based Systems – Theory and Practice*. Prentice Hall.

Gupta, U. G., and Biegel, J. 1990. A Rule–Based Intelligent Test Case Generator. In *Proceedings of the AAAI–90 Workshop on Knowledge–Based System Verification, Validation and Testing*. AAAI Press.

Knauf, R.; Gonzalez, A. J.; and Abel, T. 2002. A Framework for Validation of Rule-Based Systems. *IEEE Transactions of Systems, Man and Cybernetics - Part B: Cybernetics* 32(3):281–295.

Puppe, F. 2000. Knowledge Formalization Patterns. In *Proceedings of PKAW 2000*.

Sugiyama, K. 2002. *Graph Drawing and Applications for Software and Knowledge Engineers*. World Scientific.

Zacharias, V., and Abecker, A. 2007. On Modern Debugging For Rule-Based Systems. In *Proc. of the 19th International Conference on Software Engineering & Knowledge Engineering (SEKE'2007)*, 349–353.