Advanced Measures for Empirical Testing

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

Abstract

Empirical testing is a very popular evaluation method for the development of intelligent systems. Here, previously solved problems with correct solutions are given as cases to the system. Validity is tested by comparing the expected results with the derived solutions. Besides classic forms of boolean testing of occurring solutions 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 denotes a very important and 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, for example see (Makhoul et al. 1999) for a detailed comparison.

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 are able to compare solution states rather than the usual boolean occurrence of solutions, i.e., the default states *derived/not derived*.
2. The *chained precision/recall* that not only take into account the final solutions of a case but also intermediate solutions during a problem-solving process. They are also able to weight intermediate solutions in comparison to the final solutions.

The paper is organized as follows: We introduce the classic 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 measures. 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 the 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) *Let I be the (universal) set of observable input values. An assignment $f : a = v$ is called a finding, where $a \in I$ is an input (attribute) and $v \in dom(a)$ is an assignable value. For a set of observable inputs I we call F_I the corresponding universal set of findings (in clear cases we omit index I and only use F for short).*

Definition 2 (Solution) *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 or negative derivation of the particular output. For the refined case a more expressive range of states 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 3 (Test Case) *A test case tc is a tuple storing a list of findings and a set of derived solutions:*

$$tc = \left([f_1, \ldots, f_p], \{s_1, \ldots, s_q\}\right), \quad (1)$$

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 solution and a *rated test case*, respectively.

Definition 4 (Rated Test Case) *Let R be the universal set of ratings that is used to (partially) order the set of solutions. The specified ratings $r_i \in R$ are expected to be derived by a valid knowledge base when entering the findings f_i given in the case.*

A rated test case rtc is a tuple consisting of a list of findings and a set of rated solutions:

$$rtc = \left([f_1, \ldots, f_p], \{rs_1, \ldots, rs_q\}\right), \tag{2}$$

where $f_i \in F$ is a finding and $rs_i \in S \times R$ is a rated solution, i.e., for $rs_i : (s_i = r_i)$ a rating $r_i \in R$ is assigned to a solution $s_i \in S$.

Possible domains for the universal set of ratings R are real values in $[0, 1]$, for example to represent probabilities derived by a Bayesian network, but also symbolic values like $R = \{$undefined, excluded, suggested, established$\}$. It is easy to see that for a rating $r_i = established$ $(\forall i = 1, \ldots, q)$ a rated test case collapses to a standard test case as given in Definition 3.

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

$$stc = [rtc_1, \ldots, rtc_n],$$

where a rated test case rtc_i depends on its predecessors rtc_j $(j < i)$, i.e., solutions in rtc_i are derived based on the observation of the findings in $rtc_1, \ldots, rtc_i$.

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 $\cup_{k=1,\ldots,j} f_{i,k}$, 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 rtc_i depends on its predecessor rtc_{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 $stc = [rtc_1, \ldots, rtc_n]$ is a generalization of a rated test case; for $n = 1$ a sequential test case contains only one sequence and collapses to a rated test case.

Traditional Validation Measures

Usually, the quality of the derived solutions is computed using standard measures such as precision, recall, and the combined F-measure (Makhoul et al. 1999). In literature the measures simply compare the set of positively derived solutions with the set of expected solutions.

Definition 6 (Precision) *Let $exp \subseteq S$ be the set of expected solutions and let $der \subseteq S$ be the set of derived solutions. Then, the precision of der and exp is defined as follows:*

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

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

Definition 7 (Recall) *Let $exp \subseteq S$ be the set of expected solutions and let $der \subseteq S$ be the set of derived solutions. Then, the recall of der and exp is defined as follows:*

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

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 8 (F-Measure) *Let $exp \subseteq S$ be the set of expected solutions and let $der \subseteq S$ be the set of 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 describes single measure to weight the outcomes of precision and recall of the derived solutions. Here, the constant β is used to weight the calculated precision in relation to the recall. Often, we use the f_1 measure, where precision and recall are defined to be equally important.

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 current 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 (Puppe 2000) 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 of the solution occurrences.
2. Evaluation of the quality of a chained case sequences instead of one single test case.

Concerning the first issue we introduce "rated" versions of the precision/recall measures that generalize the standard measures and are 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 We define the comparison of solution sets with ratings by introducing a special intersection function.

Definition 9 (Intersection of Rated Solutions) *Let $RS_1, RS_2 \subseteq S \times R$ be two sets of rated solutions. The rated intersection $\cap(RS_1, RS_2)$ is defined by*

$$\cap\,(RS_1, RS_2) = \\ = \left\{ s \in S \,\middle|\, (s = r_1) \in RS_1 \,\wedge\, (s = r_2) \in RS_2 \right\}. \quad (6)$$

Instead of simply intersecting the set of derived solutions with the set of expected solutions we use the function $\cap(RS_1, RS_2)$ to extract all solutions s contained in both rated solutions sets independent from their current rating.

Definition 10 (Rated Precision) *Let $exp_{rs} \subset S \times R$ be the set of expected solutions of a rated test case, and let $der_{rs} \subset S \times R$ be the set of derived solutions. Then, the rated precision is defined as*

$$precision_{rs}(der_{rs}, exp_{rs}) = \\ = \begin{cases} prec_{rs}(der_{rs}, exp_{rs}) & \text{if } der_{rs} \neq \emptyset\,, \\ 1 & \text{if } der_{rs} = exp_{rs} = \emptyset\,, \\ 0 & \text{else}, \end{cases} \quad (7)$$

where the $prec_{rs}$ is defined by

$$prec_{rs}(der_{rs}, exp_{rs}) = \\ = \frac{\sum_{s \in \cap(der_{rs}, exp_{rs})} rsim\big(r(s, der_{rs}), r(s, exp_{rs})\big)}{|\, der_{rs} \,|}. \quad (8)$$

The rated precision is computed by using the similarity function $rsim : R \times R \to [0,1]$ for different ratings of a solution s contained in der_{rs} as well as in exp_{rs}. Function $r(s, RS)$ returns the current rating of solution s in the rated solution set RS, i.e.,

$$r(s, RS) = \begin{cases} r & \text{for } (s = r) \in RS\,, \\ 0 & \text{else.} \end{cases} \quad (9)$$

Before applying the definition of the rated precision in an application domain, we need to formulate the function for the rated similarity appropriately. It is important to notice that the similarity function $rsim$ is defined for a value range between 0 and 1, i.e., $rsim : R \times R \to [0,1]$. If it is not defined appropriately for a specific application, then we can simply use the individual similarity function as the default:

$$rsim_i\big(r(s, RS_1), r(s, RS_2)\big) = \\ = \begin{cases} 1 & \text{if } r(s, RS_1) = r(s, RS_2)\,, \\ 0 & \text{else.} \end{cases} \quad (10)$$

When using the individual similarity function the rated similarity reduces to a boolean comparison as already known from 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}.

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

We can see that the function uses the intermediate evaluations 0.8 and 0.5, so that it returns a better similarity when the expected result is better than currently derived. In some applications the counter-intuition may be appropriate.

Definition 11 (Rated Recall) *Let $exp_{rs} \subset S \times R$ be the expected solutions of a rated test case and let $der_{rs} \subset S \times R$ be the collection of derived solutions. Then, the rated recall is defined as*

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

where the rec_{rs} is defined by

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

As already introduced in the context of Definition 9 we use the function $\cap(RS_1, RS_2)$ defined in Equation 6 and the rated similarity function $rsim(\dots)$ as 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 12 (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 $exp_{i,rs} \subset S \times R$ in sequence i of the test case stc. Accordingly, we define $der_{i,rs} \subset S \times R$ to be the derived solutions in sequence i. Then, we define the chained and rated precision for $DER_{rs} = (der_{1,rs}, \ldots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \ldots, exp_{n,rs})$ as follows:

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

where $wp : \mathbb{N} \to [0, 1]$ defines the weight of the intermediate solutions in sequence i. The measure $precision_{rs}$ was already introduced in Definition 10.

It is easy to see that for $n = 1$ and $wp(n) = 1$ the chained and rated precision $precision_{rs,c}$ yields the rated precision $precision_{rs}$ introduced in Definition 10.

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 sequence, i.e., $wp(i) = 1$ for all $i = 1, \ldots, n$.
- **Inverse-annealing:** The quality of the derived solutions becomes more important in later sequences. Then, we define $wp(i) = i/n$ for $i = 1, \ldots, n$.

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

Definition 13 (Chained and Rated Recall) *Let $stc = (rtc_1, \ldots, rtc_n)$ be a sequential test case. Every rated test case rtc_i stores its expected solutions $exp_{i,rs} \subset S \times R$ at the sequence i of the test case stc. Accordingly, we define $der_{i,rs} \subset S \times R$ to be the derived solutions in sequence i. Then, we compute the chained and rated recall for $DER_{rs} = (der_{1,rs}, \ldots, der_{n,rs})$ and $EXP_{rs} = (exp_{1,rs}, \ldots, exp_{n,rs})$ as follows:*

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

where $wr(i) \in [0, 1]$ for all $i \in 1, \ldots, n$ defines the weight of the intermediate solutions in sequence i. The measure $recall_{rs}$ was already introduced in Definition 11.

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 previous sections introduced measures for an in-depth analysis of a test case. In practice, a suite of cases is used for the validation of a knowledge base. With the growing size of the suite the application of visualization methods become important in order to simplify the analysis of the results. 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, see for example (Baumeister, Seipel, and Puppe 2009) for a detailed discussion on unit testing of knowledge bases. 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. 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 this approach since it allows to interactively analyze and evaluate the validation results. For a knowledge system with an interview logic it also supports the intuitive analysis of the dialog behavior, thus verifying the interview knowledge.

Introduction to DDTrees

DDTree stands for *derivation/dialog tree* since it uses a tree structure to visualize the derivation as well as the dialog behaviour of the system. 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 the currently active input.

Every arc starting from a node and its currently active input $i \in I$ is labeled with a possible/allowed answer $v \in dom(i)$ 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 1. 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. In this example the points directly correspond to the rating of the solutions.

Larger DDTrees can be partitioned into smaller trees by extracting the subtrees under the root into single trees that are in turn validated.

DDTree as a Suite of Sequential Test Cases. It is important to notice that each path of the DDTree staring from the root to a leaf corresponds to a *sequential test case* introduced in Definition 5, where each node of such a path represents a

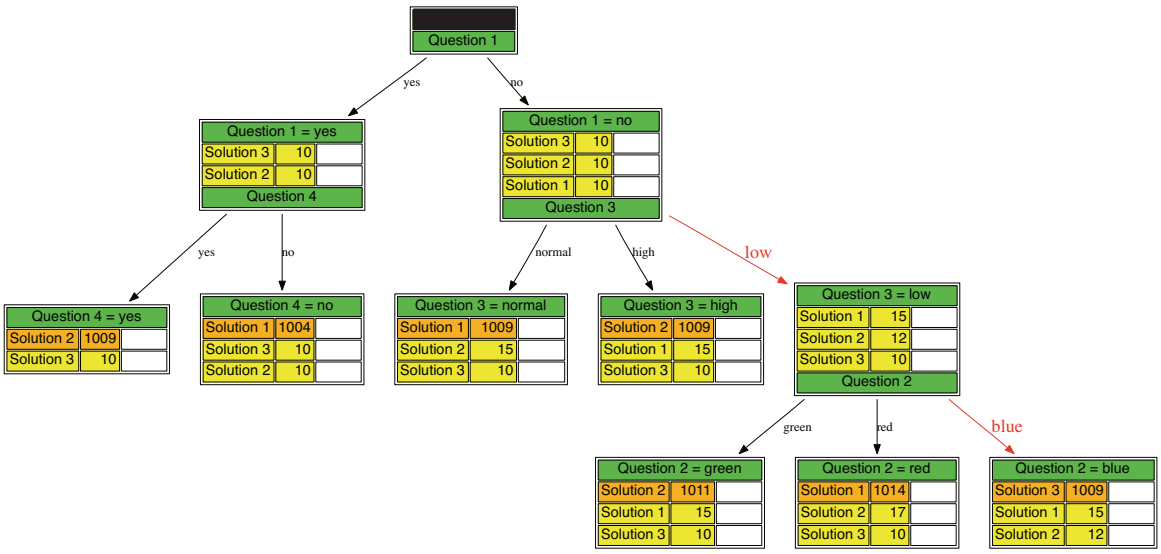

Figure 1: An example DDTree: Each test case is represented by a path from the root to a leaf of the tree. Erroneous cases are labeled with red arcs.

rated test case $rtc_i = \left([f_1], \{rs_1, \ldots, rs_q\}\right)$ with the previously answered question as the finding f_1 and the currently derived solutions as $\{rs_1, \ldots, rs_q\}$. Since the DDTree can be interpreted as a test suite of sequential test cases it becomes possible to apply the refined testing measures defined above.

Application of the DDTree. The technique considers two important issues of the validation task: the indication/inspection of erroneous cases (i.e., with precision/recall less than 1) 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 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. 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 appropriately.
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.

For the identification of the erroneous parts of the tree (i.e., faulty sequences of a test case) we use the chained precision and recall measures as defined in Definitions 12 and 13.

Validation Process

The process of using DDTrees for evaluation is as follows:

1. **Initialization:** Create an initially empty collection of previously reviewed cases $PRC = \emptyset$.
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. 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).
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 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 arc is computed according to 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 1 is rendered in red color.

4. **Manual review of the DDTree:** 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, comments can be written on the poster, etc.
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 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 extended 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 by 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 applications, especially when erroneous cases should be inspected and refined in an 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*. In the future we are planning to discuss the presented work in the context of other sophisticated tree-like knowledge representations such as XTT (Nalepa and Ligeza 2005).

References

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.

Baumeister, J.; Seipel, D.; and Puppe, F. 2009. Agile development of rule systems (in press). In Giurca; Gasevic; and Taveter., eds., *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. IGI Publishing.

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.

Makhoul, J.; Kubala, F.; Schwartz, R.; and Weischedel, R. 1999. Performance measures for information extraction. In *Proceedings of DARPA Broadcast News Workshop*, 249–252.

Nalepa, G. J., and Ligeza, A. 2005. A graphical tabular model for rule-based logic programming and verification. *Systems Science* 31(2):89–95.

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.