In: Proc. of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005), AAAI Press, 2005 pp. 814-815

Semi-Automatic Generation of Test Cases by Case Morphing

Joachim Baumeister[1], **Rainer Knauf**[2], **Frank Puppe**[1]

[1] Department of Computer Science, University of Wuerzburg, Germany
email: {baumeister, puppe}@informatik.uni-wuerzburg.de

[2] Department of Computer Science and Automation, Technical University of Ilmenau, Germany
email: rainer.knauf@tu-ilmenau.de

Introduction

The success of knowledge systems for diagnostic tasks has been proved in the last decades. Here, the evaluation of the developed knowledge system is an important and critical issue to deal with. Besides analysis techniques, such as manual inspection and static verification the use of *empirical testing* (running previously solved test cases) is the most common approach for the validation of knowledge systems. In the past, some approaches for the automated generation of test cases have been presented, cf. (Gupta & Biegel 1990; Gonzalez & Dankel 1993; Knauf, Gonzalez, & Abel 2002; Knauf *et al.* 2004). Such *conservative* methods are useful for generating suitable test cases in arbitrary domains, but require the availability of explicit knowledge, either represented as already formalized knowledge (e.g. rules) or described as generation knowledge (e.g. constraints or causal dependency models).

In this paper, we briefly introduce a novel method for test case generation which is appropriate in the context of a test-first approach. Test-first approaches postulate the development of suitable test cases *before* the corresponding functionality is added to the system, e.g. knowledge is formalized and added to the knowledge base. Test-first approaches, e.g. (Baumeister, Seipel, & Puppe 2004) have some advantages when compared to conservative approaches: 1) The developer needs to think about the desired functionality of the system before coding knowledge. 2) Test knowledge is not acquired depended on particular knowledge slices, e.g. rules. 3) Unexpected errors (compared to conservative approaches) in knowledge can be detected due to the creative nature of the approach, e.g. cases that were defined to derive a particular solution but actually do not.

Basic Definitions For the description of the method we distinguish *input values* given to a knowledge system and *output values* that are derived by the knowledge system for a given set of inputs. More formally, we define Ω_{obs} to be the (universal) set of observable *inputs* $a : v$, where $a \in \Omega_a$ is an attribute and $v \in dom(a)$ is an assignable value. An observable input $a : v$ is often called a *finding*. Let Ω_{sol} be the universe set of (boolean) *output values*, i.e. solutions derivable by the knowledge system. A test-case c is defined as a tuple $c = (\, OBS_c, SOL_c \,)$, where $OBS_c \subseteq \Omega_{obs}$ is the *problem description* of the case, i.e. the observed inputs of the case c; $OBS_c = \{f_{1,c}, \ldots, f_{n,c}\}$. The set $SOL_c \subseteq \Omega_{sol}$ contains the (correct) solutions of case c. A *test suite* is a collection of test cases that is used for empirical testing.

Method Overview

The presented morphing method is proposed to be a semi-automatic process for generating test cases according to the test-first approach. Since no available knowledge can be used to support the construction of test cases the manual development of the test suite is required. Thus, it is reasonable to support the developer during the manual construction of the test suite. For each solution the developer should think about: a) Which inputs depend on this solution? b) Which other solutions are depending on this solution? When creating new test cases the developer should include the results of this consideration into the development process, i.e. by creating test cases that include dependent input objects and dependent solutions. For each new solution and its derivation knowledge, respectively, we propose a 5-fold process with detailed descriptions in the following sections:

1. Consider dependent findings and (already existing) solutions. Manually create a collection of test cases that tries to take the considerations into account.
2. Formalize new derivation knowledge, e.g. rules, that infer the considered solution.
3. Check whether the new knowledge successfully passes the test cases. Start for each manually defined test case the case morphing process, and remove all generated cases that derive an incorrect solution. Select the k most diverse cases from the remaining morphing set (Smyth & McClave 2001).
4. Coverage check: Investigate the coverage of the manually defined cases and the remaining generated cases w.r.t. the formalized knowledge. If the coverage of these tests is found to be unsatisfactory, then propose a collection of input values that should also be included in additional test cases. These new input values serve as an input for a further (parameterized) case morphing phase.
5. The remaining test cases are presented to the developer; here the knowledge base can be refined if presented cases

are in contradiction to the developers knowledge or expectations. Successfully reviewed cases are accepted and included in the (already existing) test suite.

We now describe the morphing process sketched in Step 3 step in more detail.

Morphing Cases

The idea of morphing an existing case $c = (OBS_c, SOL_c)$ is quite simple: new cases are generated by gradually increasing or decreasing the values of findings in OBS_c. We restrict the number of morphed cases by a threshold value t, which limits the maximum number of generated morphs for each finding.

Morphing a Finding A given finding $f = a : v$ is morphed by using a morphing function $morph : \Omega_a \times \Omega_V \rightarrow \times_{i=1}^{n} \Omega_V$, which returns a list of the t most adjacent values $v' \in dom(a) \setminus \{v\}$ for a value $v \in dom(a)$; for a numerical finding f we first divide the value range into p equal partitions, and we select the partition p_i which contains the value v. Then, the function $morph$ can deal with the partitions in the same way as for symbolic values. Thus, for a given finding $a : v_i$ the function yields $morph(a, v_i) = (v_{i_1}, \ldots, v_{i_{t'}})$, where the distance $d(v_i, v_{i_j}) \leq d(v, v')$ for all $v' \in dom(a) \setminus \{v_{i_1}, \ldots, v_{i_{t'}}\}$. Note that $t' \leq t$, i.e. the actual number of selected values t' can be smaller than t, if $|dom(a)| - 1 < t$.

Generate Cases with Morphed Findings For a given test case $c = (OBS_c, SOL_c)$ we enumerate the possible combinations of morphed findings in a $|OBS_c|$-dimensional matrix M. Each entry of the matrix $M_{i,j}$ describes a problem description of a morphed case. We see that the number of entries in the matrix M, i.e. the number of generated problem descriptions, is at most $|OBS_c|^t$. For larger $|OBS_c|$ and t the number of the generated cases may be too large. Therefore, we discuss additional domain knowledge that further restricts the size of the generated matrix.

Using Domain Knowledge Ontological knowledge can be defined for the values of a finding which provides information about the normality of the values, i.e. their pathological importance in the application domain. In general, an abnormality function $abn : \Omega_{obs} \rightarrow \Omega_{abn}$ is defined by the domain specialist. With a given abnormality function we are able to refine the morphing function $morph$, so that normal values are omitted during the generation of the generation of the matrix, i.e. reducing the number of considered findings $f \in OBS_c$. Then, all findings with normal values are not morphed but are included with original values in the generated problem descriptions.

Filter Generated Cases Each generated problem description $OBS_{c'}$ in the matrix M is passed to the knowledge system in order to derive a solution $SOL_{c'}$ for $OBS_{c'}$. The case is removed if an incorrect solution set was derived, i.e. the derived solutions differ from the solutions included in the manually defined case. The remaining cases are filtered according to their diversity: We select the k most diverse cases from the remaining cases. The diversity of cases was investigated e.g. in (Smyth & McClave 2001).

Computing the Coverage of the Test Suite

Test cases are generated before the acquisition of the corresponding knowledge. In consequence, we cannot guarantee a suitable cover of the knowledge to be tested. For this reason, a check is performed in order to identify areas of the acquired knowledge, that are not covered by the test suite. Such, missing combinations of findings are used in a subsequent iteration of the morphing process, and additional cases are generated. For rule-based knowledge representations appropriate methods has been investigated thoroughly in the past, cf. (Barr 1999; Knauf, Gonzalez, & Abel 2002).

Discussion

We classified the presented method as a *creative* approach in contrast to traditional *conservative* ones. When comparing the approaches we see some differences: Creative approaches are independent of the knowledge to be tested and therefore can be applied for arbitrary representations of knowledge. Furthermore, they are appropriate for development process models postulating a *test-first paradigm*, e.g. an agile methodology (Baumeister, Seipel, & Puppe 2004). However, since conservative approaches use the available knowledge, they can guarantee that generated test cases completely cover all aspects of the acquired knowledge. By nature, creative approaches cannot consider the coverage of the knowledge during the generation of the test cases, but we cope with this problem by iterating the morphing step based on a coverage analysis.

References

[Barr 1999] Barr, V. 1999. Applications of Rule-Base Coverage Measures to Expert System Evaluation. *Knowledge-Based Systems* 12:27–35.

[Baumeister, Seipel, & Puppe 2004] Baumeister, J.; Seipel, D.; and Puppe, F. 2004. Using Automated Tests and Restructuring Methods for an Agile Development of Diagnostic Knowledge Systems. In *Proc. 17th Intl. FLAIRS Conference*, 319–324. AAAI Press.

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

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

[Knauf *et al*. 2004] Knauf, R.; Spreeuwenberg, S.; Gerrits, R.; and Jendreck, M. 2004. A Step out of the Ivory Tower: Experiences with Adapting a Test Case Generation Idea to Business Rules. In *Proc. 17th Intl. FLAIRS Conference*, 343–348. AAAI Press.

[Knauf, Gonzalez, & Abel 2002] 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.

[Smyth & McClave 2001] Smyth, B., and McClave, P. 2001. Similarity vs. Diversity. In *Proc. 4th Intl. Conference on Case-Based Reasoning, ICCBR 2001*, 347–361. Springer, LNAI 2080.