Constraint-based test data generation for database-driven applications

Marcozzi, Michaël

Published in:
Pre-Proceedings of the GTTSE-SLE 2011 Students' Workshop

Publication date:
2011

Document Version
Peer reviewed version

Constraint-based test data generation for database-driven applications

Michael Marcozzi*

GTTSE'11 Students' Workshop

Faculty of Computer Science
University of Namur, Belgium
mmr@info.fundp.ac.be
http://info.fundp.ac.be/~mmr

1 Problem Description and Motivation

From a pure mathematical object, created by logicians, software has now turned into a crucial part of the process of many human activities. As computers became more and more ubiquitous in the human society and programs more and more complex, the need for reliable software naturally emerged as a critical concern, which is the core motivation for many research works in software engineering.

Informally, a piece of software is said to be reliable if it behaves in the expected way for every possible condition of its execution. Software testing is a commonly applied approach for improving reliability of software, and it is considered as an indispensable technique to software engineering discipline [1]. In a nutshell, testing a software component means running it with respect to a set of its potential conditions of execution, and comparing its actual behavior with the expected one, in order to find out and correct errors.

The more a program is seen to behave correctly in respect to a large number of different conditions of execution, the more the confidence in the reliability of this program is increased [2]. Typically, in "real world" problems, it is not efficient (both for theoretical reasons and for practical reasons, like time and cost) and often even not possible to test a program in all the possible conditions of its execution. An important research question in testing is thus [1] the choice of an adequate set of conditions of execution, so that testing the program with this set of conditions gives efficiently a sufficient confidence in the reliability of the program. This question obviously raises two related research problems: first, how to define and (automatically) measure the adequacy of a set of test conditions, and then, how to select (automatically) an adequate set of conditions to test a program.

Adequacy of a test set is typically measured with respect to how well it exercises all the characteristics of the tested program itself, and/or of its specification [1]. *Program-based adequacy* measures how well a test set exercises the different potential behaviors implemented by the program. *Specification-based*

* F.R.S.-FNRS research fellow / Aspirant du F.R.S.-FNRS (www.fnrs.be)

adequacy measures how well a test set exercises the different features identified in the specification of the program. It is now widely acknowledged that an adequate test set should exercise both the characteristics of the specification and the program. Measuring how well a test set exercises these characteristics is typically achieved using either a *structural*, *fault-based* or *error-based approach*. The *structural approach* measures adequacy as the level of coverage during the tests of elements that constitute the formal structure of the specification/program. The *fault-based approach* measures how well a test set allows to distinguish between the actual program/specification and its potential faulty variants. The *error-based approach* establishes how well the test set exercises the known error-prone points in the program or the specification, given the knowledge about how programs typically differ from their specification.

Many formal and precise criteria have been proposed to evaluate the adequacy of a test set [1]. Such formal adequacy criteria are typically designed as the formal specification for an associated automatic test set generation technique. The idea of such techniques is to automatically select an adequate (according to the criterion chosen as specification) set of test conditions with respect to which a given program can be tested.

Many existing adequacy criterion and automatic test set generation techniques proposed in the literature are typically based on an underlying purely functional model of program (see e.g. [3, 4]). In many research works on testing, a program is indeed modeled as the implementation of a specified function. It receives some data as input and must compute the result of the specified function on this input, in order to return it as output [3]. In such a model, the conditions of execution of a program only consist in the input data it receives, while the purpose of testing the program will be to ensure it computes its specified function right. As a result, most automatic test set generation techniques are designed to generate a set of adequate input data that allow to test the program in an satisfying way. The syntax and semantics of these input data, i.e. their structure and meaning, have typically no special importance by themselves. A datum will indeed be selected as test datum only because it can exercise some particular characteristics of the computation enforced by the program or of the functional specification of this program, as specified by the chosen adequacy criterion [16]. Moreover, for the sake of simplicity, many research works typically focus on programs whose input data are defined on very simple input domains, like integers and reals.

However, many pieces of software designed nowadays are data-centered, i.e. the focus is much less on the computation enforced by the program, and much more on an efficient and reliable processing of the data that the program manipulates. This is typically the case with database-driven applications, i.e. large-scale programs designed to manipulate data that reside in a database and as such are persistent between different runs of the program. While testing such programs, the state of the complex and highly structured manipulated database should certainly be part, as well as the traditional inputs of the program, of the different execution conditions exercised by the tests. Moreover, a major concern

for the testing process of database-driven applications will be to verify that the program enforces and maintains the reliability of the data in the manipulated database. The subject of my doctoral research is the study of automatic generation of test conditions sets for such database-driven applications. The two main research questions that will be investigated are the following:

- How should the traditional criteria of test adequacy evolve in the frame of database-driven applications? In particular, how should the need to test the enforcement and maintaining of the reliability of the data by the program be integrated with the state-of-the-art adequacy criteria, existing in the literature for computation-centered programs?
- Can a coherent and complete formal framework, based on an unified model of a database-driven application, notably integrating the data model of the database with the functional model of the program, be proposed to generate automatically adequate test sets for database-driven applications?

2 Brief overview of Related Work

Test input generation for database applications has only occasionally been considered before.

In [6], authors present a set of tools to test database applications. Database states are treated independently of the program inputs, and generated on the basis of the database schema and of heuristics aiding at the generation of states likely to expose application faults. In [7], authors propose to use information from the database schema to generalize a classical fault and program-based adequacy criterion to programs containing SQL statements. In [8], authors introduce a family of test adequacy criteria for database-driven applications inspired by existing structural program-based criteria. In [9], authors propose an algorithm based on a simultaneously concrete and symbolic (concolic) execution of the tested program, to generate both input data for the program as well as suitable database states to systematically explore all paths of the program. In [10], authors adopt a similar concolic approach, but where the program is executed on a parameterized mock database.

Whereas they offer interesting contributions to the question of test data generation for database-driven applications, none of these works seems to present an unified approach, based on a firmly established definition of test adequacy for database-driven applications, and in which the data model of the database is fully integrated with the functional model of the program.

3 Proposed Solution

At this early stage of research, we intend to generate automatically test data for database-driven applications by generalizing the constraint-based test case generation framework proposed in [11]. This framework, developed in the frame of the traditional functional model of program, would be extended to deal with

a new model of program and a new definition of test adequacy that we would propose for database-driven applications.

Constraint-based test data generation, originally introduced in [3], basically transforms the problem of generating test data into a constraint programming problem over finite domains. This approach has been advocated many times, e.g. [12–14] and it is used in different testing tools, e.g. Godzilla [15] and InKA [12]. Key advantages of constraint-based test data generation are the conceptually clean modeling of the problem as a single constraint problem, its promised efficiency due to the existence of fast constraint solvers, e.g. based on SMT [17], and the fact that the approach is to some extent independent of (and can thus by parametrized by) an adequacy criterion since the latter basically represents the particular chosen set of constraints and the search procedure used in the solver [11].

The independence of this approach with regard to the chosen adequacy criterion should allow us to adapt it to a new proposed adequacy criterion, developed to ensure an efficient testing of the reliability of database-driven applications. A starting point to define this new criterion could be the criteria proposed in [8].

Moreover, the initial database state could be considered as an additional input datum of the program, allowing us to use a unique constraint system, unifying the internal program data constraints, the abstract data model and the developer's assertions. We could deal with the highly complex structure of these new input data to generate (that should typically verify the database schema and any additional constraint), by using abstract models of different views of this structure, to ease the expression of the data model and to guide the subsequent test data generation process, as advocated in [16].

4 Research Method

At this early stage of research, we are considering the following research method:

- Propose a clear and formal model defining what a database-driven application is in the frame of this work.
- Propose a revised notion of adequacy for testing programs instantiating this model.
- Based on this work, evaluate how well the proposed constraint-based and data-extended approach can help to generate adequate test sets for database-driven applications. Identify potential problems.
- Propose solutions to the eventual identified problems or search for another approach.
- Develop a formal framework implementing the adopted approach.
- Establish how the proposed work could be helpful to the related research domains (software and database (re-)engineering) in general, and the other way round.

This method should not be considered as a purely linear one, as interleaving and iterations of the different steps will typically appear. The work will of course be based on a constant deep review of the existing related scientific literature.

Note: This paper is an introductory paper to the research presented in [18–22].

References

1. H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29:366–427, December 1997.
2. C. Kaner, J. L. Falk, and H. Q. Nguyen. *Testing Computer Software, Second Edition.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1999.
3. R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation, 1997.
4. W. E. Howden. Reliability of the path analysis testing strategy. *IEEE Trans. Softw. Eng.*, 2:208–215, May 1976.
5. A. Demakov, S. Zelenov, and S. Zelenova. Using abstract models for the generation of test data with a complex structure. *Programming and Computer Software*, 34:341–350, 2008. 10.1134/S0361768808060054.
6. D. Chays, Y. Deng, P. G. Frankl, S. Dan, F. I. Vokolos, and E. J. Weyuker. An agenda for testing relational database applications: Research articles. *Softw. Test. Verif. Reliab.*, 14:17–44, March 2004.
7. W. K. Chan, S. C. Cheung, and T. H. Tse. Fault-based testing of database application programs with conceptual data model. In *Proceedings of the Fifth International Conference on Quality Software*, QSIC '05, pages 187–196, Washington, DC, USA, 2005. IEEE Computer Society.
8. G. M. Kapfhammer and M. L. Soffa. A family of test adequacy criteria for database-driven applications. *SIGSOFT Softw. Eng. Notes*, 28:98–107, September 2003.
9. M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, pages 151–162, New York, NY, USA, 2007. ACM.
10. K. Taneja, Y. Zhang, and T. Xie. Moda: automated test generation for database applications via mock objects. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 289–292, New York, NY, USA, 2010. ACM.
11. F. Degrave, T. Schrijvers, and W. Vanhoof. Towards a framework for constraint-based test case generation. In D. De Schreye, editor, *Logic-Based Program Synthesis and Transformation*, volume 6037 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin / Heidelberg, 2010. $10.1007/978-3-642-12592-8_10$.
12. A. Gotlieb, B. Botella, and M. Rueher. Automatic test data generation using constraint solving techniques. In *Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '98, pages 53–62, New York, NY, USA, 1998. ACM.
13. A. Gotlieb, T. Denmat, and B. Botella. Goal-oriented test data generation for pointer programs. *Information and Software Technology*, 49(9-10):1030 – 1044, 2007.
14. E. Albert, M. Gómez-Zamalloa, and G. Puebla. Logic-based program synthesis and transformation. chapter Test Data Generation of Bytecode by CLP Partial Evaluation, pages 4–23. Springer-Verlag, Berlin, Heidelberg, 2009.
15. A. J. Offutt, Z. Jin, and J. Pan. The dynamic domain reduction procedure for test data generation. *Softw. Pract. Exper.*, 29:167–193, February 1999.
16. A. Demakov, S. Zelenov, and S. Zelenova. Using abstract models for the generation of test data with a complex structure. *Programming and Computer Software*, 34:341–350, 2008. 10.1134/S0361768808060054.
17. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(¡i¿t¡/i¿). *J. ACM*, 53:937–977, November 2006.

18. M. Marcozzi, W. Vanhoof, and J.-L. Hainaut. Test input generation for database programs using relational constraints. In *Proceedings of the Fifth International Workshop on Testing Database Systems*, DBTest '12, pages 6:1–6:6, New York, NY, USA, 2012. ACM.
19. M. Marcozzi, W. Vanhoof, and J.-L. Hainaut. A relational symbolic execution algorithm for constraint-based testing of database programs. In *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on*, pages 179–188, 2013.
20. M. Marcozzi, W. Vanhoof, and J.-L. Hainaut. Testing database programs using relational symbolic execution. Technical report, University of Namur, 2014.
21. M. Marcozzi, W. Vanhoof, and J.-L. Hainaut. Towards Testing of Full-Scale SQL Applications using Relational Symbolic Execution. In *Proceedings of the 6th Workshop on Constraints in Software Testing, Verification, and Analysis*, CSTVA '14, Pages 12–17 ACM New York, NY, USA, 2014
22. M. Marcozzi, W. Vanhoof, and J.-L. Hainaut. A Direct Symbolic Execution of SQL Code for Testing of Data-Oriented Applications. Technical report, CoRR: a computing research repository, 2015.