A Search Based Test Data Generation Approach for Model Transformations

Atif Aftab Jilani[1], Muhammad Zohaib Iqbal[1,2], and Muhammad Uzair Khan[1]

[1]Software Quality Engineering and Testing Laboratory (QUEST),
National University of Computer & Emerging Sciences, Pakistan
[2]SnT Centre Luxembourg, Luxembourg
{atif.jilani,zohaib.iqbal,uzair.khan}@nu.edu.pk

Abstract. Model transformations are a fundamental part of Model Driven Engineering. Automated testing of model transformation is challenging due to the complexity of generating test models as test data. In the case of model transformations, the test model is an instance of a meta-model. Generating input models manually is a laborious and error prone task. Test cases are typically generated to satisfy a coverage criterion. Test data generation corresponding to various structural testing coverage criteria requires solving a number of predicates. For model transformation, these predicates typically consist of constraints on the source meta-model elements. In this paper, we propose an automated search-based test data generation approach for model transformations. The proposed approach is based on calculating approach level and branch distances to guide the search. For this purpose, we have developed specialized heuristics for calculating branch distances of model transformations. The approach allows test data generation corresponding to various coverage criteria, including statement coverage, branch coverage, and multiple condition/decision coverage. Our approach is generic and can be applied to various model transformation languages. Our developed tool, MOTTER, works with Atlas Transformation Language (ATL) as a proof of concept. We have successfully applied our approach on a well-known case study from ATL Zoo to generate test data.

Keywords: Software Testing, Model Transformation (MT), ATL, Search Based Testing (SBT), Structural Testing.

1 Introduction

Model transformations (MT) are a fundamental part of Model Driven Engineering (MDE). As for any other software, correctness of model transformation is of paramount importance. Automated testing of model transformations faces a number of specific challenges when compared to traditional software testing [1]. The foremost is the complexity of input/output models. The meta-models involved in the transformations typically comprise of a large set of elements. These elements have relationships, sometimes cyclic, that are restricted by the constraints define on the meta-model/model. Generating input models manually is laborious and error prone. On the

D. Di Ruscio and D. Varró (Eds.): ICMT 2014, LNCS 8568, pp. 17–24, 2014.

other hand automated generation of input models requires solving complex constraints on the meta-models.

In this paper, our objective is to enable automated structural testing of model transformations. The idea is to generate test cases that cover various execution paths of the software under test. We present an automated search-based test data generation approach for model transformations. To guide the search, we propose a fitness function specific for model transformation. The fitness function utilizes a so-called approach level and branch distance. The branch distance is calculated based on heuristics defined for various constructs of model transformations.

We selected Alternating Variable Method (AVM) as the search algorithm for this purpose because it has already been successfully applied for software testing [2]. We tailored AVM for our specific problem. To the best of our knowledge, this is the first work to report an automated model transformation structural test data generation approach based on search-based testing. To support the automation of the proposed approach, we also developed a tool called MOTTER (Model Transformation Testing Environment). We apply the test data generation approach on an open source model transformation from the ATL Zoo[1].

The rest of this paper is organized as follows: Section 2 provides the related work and presents the state of the art related to MT testing and its associated challenges. Section 3 discusses the proposed test model generation methodology. Section 4 discusses the tool support, whereas Section 5 discusses the application of the approach on a case study. Finally Section 6 concludes the paper.

2 Related Work

Fleurey *et al*.,[3] discuss category partitioning scheme and introduced the concept of effective meta-model. Wang *et al*, [4] explores verification and validation of source/target meta-models in term of coverage. Sen *et al*., [5] proposed various model generation strategies including random/unguided and input domain partition based strategies. Vallecillo *et al*., [6] propose the use of formal specification for test data generation. Gomez *et al.,* [7] use the concept of simulated annealing to generate test models. Cariou *et al.,* [8] proposed a method that use OCL contracts for the verification of model transformations. The work proposed by Guerra *et al*., [9], generates automated test models, from formal requirement specification and solved pre/post conditions and invariants using OCL. Wang and Kessentini [10] propose black box technique for the testing of meta-model structural information. The technique use search algorithms and provide structural coverage and meta-model coverage.

Kuster *et al*., [11] reported the challenges associated with White box MT testing. Buttner *et al.,* [12] proposed the use of first order semantics for a declarative subset of ATL. Gonzalez and Cabot [13] discuss dependency graph, examine dependency graph by applying traditional coverage criteria and generate test case as OCL expression for ATL. McQuillan *et al.,* [14] proposed various white box testing criteria for

[1] `http://www.eclipse.org/m2m/atl`

ATL transformations such as, rule coverage, instruction coverage and decision coverage. Mottu *et al.,*[15] proposed a constraint satisfaction problem in Alloy.

The work presented here is significantly different from the above approaches as we adopt a search-based test data generation approach for automated white-box testing of model transformations. We build on the previous work of OCL solver [2, 16, 17] to generate valid meta-model instances and provide search heuristics for various model transformation language constructs.

3 Automated Test Data Generation for MT

This section discusses the automated test data generation approach for structural testing of model transformations.

3.1 Test Case Representation

A test case in our context is a set of input models (i.e., a set of instances of input meta-model) that provide maximum coverage of the model transformation under test. A number of coverage criteria have been developed for structural testing of software programs [18]. To achieve a specified coverage level, the test data needs to solve various predicates in the transformation language. In the case of model transformations, these predicates are typically constraints on the elements of meta-models.

3.2 Problem Representation

In the context of test data generation for model transformations, a problem is equivalent to a transformation language predicate. A language predicate *P* (problem) is composed of a set of Boolean clauses $\{b_1, b_2 \ldots b_n\}$ joined by various Boolean operations, such as, *and*, *or* and *not*. Each clause b_i, itself comprises of various variables $\{b_{i1}, b_{i2} \ldots b_{iZ}\}$ used in the clause. To solve a problem (*P*), the search algorithm first needs to solve all clauses which are (n) in number, and for each clause (b_i), need to generate correct values for the entire variables till (z). To generate test data for transformation predicates, search algorithm needs guidance. We provide heuristics for various clauses of model transformation languages. The heuristics are defined as a branch distance function *d()*, which returns a numerical value representing how close the value was to solving the branch condition. A value zero represents that branch condition is satisfied; otherwise a positive numerical non-zero value is returned that provides an estimate of distance to satisfy the constraint.

3.3 Test Data Generation

The algorithm for search-based test data generation for model transformations is shown in Fig. 1. Following sections discuss the various steps of the strategy.

Generating Instance Models. The first step is to generate a random instance of meta-model. The generated model should be a valid instance of the source meta-model.

Generating a valid instance requires solving the various constraints on the meta-model. The generated instance should also contain links corresponding to the mandatory associations of the meta-model (i.e., having a multiplicity of 1 or above). A number of techniques have been proposed in the literature for generating meta-model instances [19]. A major problem is satisfying the various constraints on the meta-models, typically written in Object Constrain Language (OCL). For generating instances that satisfy the OCL constraints, we extended the approach presented in [2].

Algorithm ***generateTestData*(*mm, CFG, max*)**
Input *mm*: source meta-model, *CFG*: Control flow graph, *max*; No of maximum iterations
Declare *C*: Set of conditions={}, n: # of iteration performed T_m: A random test data (instance of a meta-model), b_i: A Condition from C,

1. **begin**
2. Generate a random instance T_m of *mm* as test data
3. Traverse T_m on *CFG* and add all branching conditions into *C*.
4. **for each** $C_i \in C$
5. Calculate fitness $f(O) = mini{=}0 \rightarrow C.size(ACi(O) + nor\ (BCi(O)))$
6. **if** $f(O) != 0$ AND $n < max$
7. **then** modify T_m by adding/modifying instances of meta-elements according to search algorithm.
8. Increment *n*
9. **end if.**
10. **end for**
11. **end**

Fig. 1. Algorithm for the proposed test data generation strategy

To generate an instance of source meta-model we first traverse the model transformation under test to identify the set of meta-model elements used in the transformation. This set is referred to as an effective meta-model [3]. The identified set of elements is typically related to other elements not used in the transformations. We keep all elements as part of the effective meta-model that have mandatory relationships. We initially generate instances of all meta-elements used in the transformation predicate and then add links between the instances based on the meta-model.

Fitness Functions for MT Language. Search algorithms are guided by fitness functions that evaluate the quality of a candidate solution. The fitness function, for example, in the case of structural coverage can evaluate how far a particular test case is from solving a predicate. The fitness functions are problem-specific and need to be defined and tuned according to the problem being targeted.

Model transformation languages are similar to programming languages in a way that they are imperative and have control flow and side effects. The model transformation languages are also similar to Object Constraint Language (OCL), because they are written on modeling elements (and their syntax is inspired from OCL). Therefore the fitness function that we developed for testing of model transformations is adapted from the fitness functions of programming languages and OCL [2]. The goal of the

search is to minimize the fitness function f, which evaluates how far a particular test case is from solving a predicate. If the predicate is solved, then f(t) = 0.

Since our approach is based on heuristics, the generated solutions of our approach are not necessarily optimal. The heuristics do not guarantee that the optimal solution will be found in a reasonable time. However, various software engineering problems faced by the industry have been successfully solved using search based algorithms.[20]. Our fitness function is a combination of approach level and branch distance and can be represented by the following equation:

$$f(O) = \min_{i=0 \rightarrow TP.size}(A_{TPi}(O) + nor\ (B_{TPi}(O)))$$

where O is an instance of input meta-model generated as a candidate test data, TP is a set of target predicates to be solved. ATPi(O) represents the approach level achieved by test data O. The approach level calculates the minimum number of predicates required to be solved to reach the target predicate TPi.

$B_{TPi}(O)$ represents the branch distance of a target predicate *TP*. The branch distance heuristically evaluates how far the input data are, from solving a predicate. The branch distance guides the search to find instances of meta-model that solve the target predicates. For example, to solve a predicate on a class Account: $account \rightarrow size\ () > 10$, the search needs to create eleven Account instances.

We consider a normalized value (nor) for branch distance between the values [0, 1], since branch distance is considered less important than approach level. We apply a widely used normalizing function for this purpose [2]: nor(x) = x/x+1.

To calculate both the approach level and branch distance, we instrumented the transformation language code. Based on the coverage criterion, in some cases, the generated test data not only needs to satisfy the predicates to true, but also needs to satisfy the negation of the predicates (for example, to achieve branch coverage). In all such cases, we simply negate the predicate and for the negated predicate, generate the data that evaluated the negated predicate to true. To calculate the approach level, an important step is to construct a control flow graph (CFG) of the model transformation code. The CFG provides the guidance to the algorithm to achieve the desire coverage.

Branch Distances for MT Constructs. The transformation languages have a number of predefined data types, called primitive types. Typical primitive types include Boolean, Integer, Real, and String. The predicates are defined on attributes of primitive types, collection types or meta-model classes and combine the attributes with various operators resulting in a Boolean output. Branch distance calculations for various important operations of model transformations are adopted from [2].

Applying the Search Algorithm. We selected Alternating Variable Method (AVM) [2] as the search algorithm. For a set of variables $\{v_1, v_2,....v_n\}$, AVM works to maximize the fitness of v_1, by keeping the values of other variable constant, which are generated randomly. It stops, if the solution is found. Otherwise if solution is not found or fitness is lesser than v_1, AVM switch to the second variable v_2. Now all other variables will be kept constant. The search continues until a solution is found or all the variables are explored. If a randomly generated initial model is not able to satisfy the target predicate, a fitness value is generated for the test model. We generate a new

model by modifying the previous model. If the fitness of new model is greater than that of previous model, the new model is used for next search iteration.

4 Tool Support

In this section, we present our implementation of model transformation testing environment (MOTTER). Fig. 2 shows the architectural diagram of the MOTTER tool. We have developed MOTTER in java language that enables it to interact with the already existing components of OCL Solver [2]. Currently, MOTTER support ATL only, but it is designed in a way to support several model transformation languages. MOTTER is performing various tasks, it supports ATL compilation, shows compile time error and at same time able to execute a program in way that CFG could be extracted. For a given program in ATL, MOTTER constructs the CFG for the given transformation, its component ATLExecutor executes the transformation for a given source model and calculates the fitness and the branches covered so far.

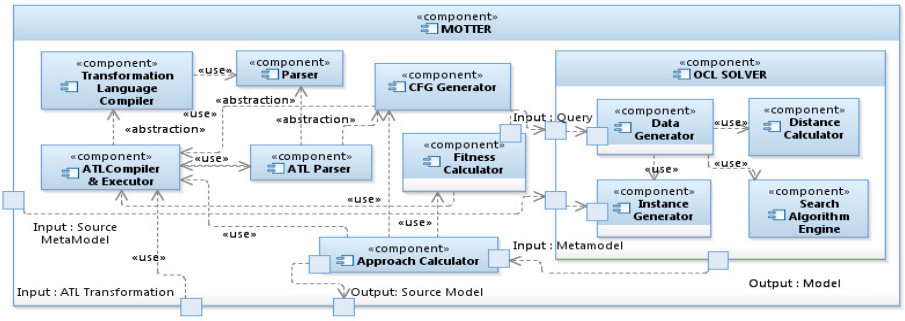

Fig. 2. Architectural diagram of the MOTTER tool

Coverage Analyzer and fitness calculator ensure that coverage criterion, such as branch coverage are achieved. Fitness calculator guides the coverage analyzer regarding the fitness of the instance model and calculates the approach level. The Solver in MOTTER is a refined version of OCL Solver [2]; the original OCL solver is OCL specific and generates data values for OCL queries. For MOTTER, the data values are not simple. The values are classes, instance of meta-model and include the relationships between the classes. Object Generator component generates the object model that serves as a test model to ensure coverage of transformation. Distance Calculator module calculates distances of transformation construct. Search Algorithm Engine component uses AVM to solve the heuristics and test data is generated by the data generator component. The Data Generator component guides the search by generating values that solve the heuristics.

5 Case Study

In this section, we demonstrate how test models are generated by applying our approach on a famous *SimpleClass2SimpleRDBMS* ATL transformation [21]. The case

study has six helper methods and one matched rule. The matched rule *PersistentClass2Table* is considered as the main rule. It comprises of nine (9) predicate statements, such as *tuple.type→oclIsKindOf(SimpleClass!Class)*. To exercise various coverage criteria, these predicates have upmost importance. We first generate test data for all branch coverage. All Branch coverage requires exercising of each statement and conditions, and to do so all predicates need to be solved.

We slightly modified some statements in the transformation as some of the code segments of the original transformation could not be executed (part of the dead code). Since, we have nine different branching conditions and for each such condition, our tool has generated data that satisfies the conditions and their negations. Consider a condition, taken from *persistentClass2Table*, *acc→size()=0*. The condition has an approach level value one, because to exercise this we first need to solve the condition, *tuple.type→oclIsKindOf(SimpleClass!Class)*. MOTTER has successfully solved all nine conditions and generates various object models (test models) to satisfy all branch coverage, decision coverage and statement coverage criterion. The case study demonstrates the applicability of the approach on real transformations. The performance and evaluation of the approach is not discussed due to space limitation.

6 Conclusion

We discussed an automated, structural search-based test data generation approach for model transformations testing. Our approach generates test data to satisfy various structural coverage criteria, such as branch coverage. To guide the search, we developed a fitness function that comprises of approach level and branch distance. To calculate branch distance for model transformation constructs, we adopted the existing heuristics for programming languages and Object Constraint Language. We not only generate meta-elements instances of effective meta-model but also handle the mandatory relationships that exist between different meta-elements. Therefore, our instance generation approach is able to generate valid meta-model instances. The output of the approach is a set of instance models of the source meta-model that can be used as test models to attain transformation coverage. The use of search based heuristics for the automated test data (model) generation particularly in the case for model transformation is a major contribution of the work. We applied Alternating Variable Method (AVM) as a search algorithm for test data generation. The applicability of the approach is demonstrated by applying on a widely referred case study from the ATL transformation zoo, the *SimpleClass2SimpleRDBMS* transformation. The case study covers a number of important ATL constructs. The proposed approach successfully generated test models to achieve the desired coverage. We also developed a prototype tool MOTTER to automate the proposed methodology. The tool currently supports transformation written in ATL, but it is extensible to handle other transformation languages.

References

1. Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y., Mottu, J.M.: Barriers to systematic model transformation testing. Communications of the ACM 53(6), 139–143 (2010)
2. Ali, S., Iqbal, M., Arcuri, A., Briand, L.: Generating Test Data from OCL Constraints with Search Techniques. IEEE Transactions on Software Engineering 39(10), 26 (2013)

3. Fleurey, F., Baudry, B., Muller, P.A., Traon, Y.L.: Qualifying input test data for model transformations. Software and Systems Modeling 8(2), 185–203 (2009)
4. Wang, J., Kim, S.-K., Carrington, D.: Automatic generation of test models for model transformations. In: 19th Australian Conference on Software Engineering, ASWEC 2008. IEEE (2008)
5. Sen, S., Baudry, B., Mottu, J.-M.: Automatic model generation strategies for model transformation testing. Theory and Practice of Model Transformations, 148–164 (2009)
6. Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L.: Formal specification and testing of model transformations. Formal Methods for Model-Driven Engineering, 399–437 (2012)
7. Gómez, J.J.C., Baudry, B., Sahraoui, H.: Searching the boundaries of a modeling space to test metamodels. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST). IEEE (2012)
8. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. In: Proceedings of the Workshop the Pragmatics of OCL and Other Textual Specification Languages at MoDELS (2009)
9. Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Automated verification of model transformations based on visual contracts. Automated Software Engineering, 1–42 (2012)
10. Wang, W., Kessentini, M., Jiang, W.: Test Cases Generation for Model Transformations from Structural Information. In: 17th European Conference on Software Maintenance and Reengineering, Genova, Italy (2013)
11. Küster, J.M., Abd-El-Razik, M.: Validation of model transformations–first experiences using a white box approach. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 193–204. Springer, Heidelberg (2007)
12. Büttner, F., Egea, M., Cabot, J.: On verifying ATL transformations using 'off-the-shelf' SMT solvers. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 432–448. Springer, Heidelberg (2012)
13. González, C.A., Cabot, J.: ATLTest: A White-Box Test Generation Approach for ATL Transformations. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 449–464. Springer, Heidelberg (2012)
14. McQuillan, J.A., Power, J.F.: White-box coverage criteria for model transformations. In: Model Transformation with ATL, p. 63 (2009)
15. Mottu, J.-M., Sen, S., Tisi, M., Cabot, J.: Static Analysis of Model Transformations for Effective Test Generation. In: ISSRE-23rd IEEE International Symposium on Software Reliability Engineering (2012)
16. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: A Search-based OCL Constraint Solver for Model-based Test Data Generation. In: 2011 IEEE 11th International Conference on Quality Software, pp. 41–50 (2011)
17. Ali, S., Iqbal, M.Z., Arcuri, A.: Improved Heuristics for Solving OCL Constraints using Search Algorithms. In: Proceeding of the Sixteen Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion (GECCO). ACM, Vancouver (2014)
18. Myers, G., Badgett, T., Thomas, T., Sandler, C.: The art of software testing. Wiley (2004)
19. Wu, H., Monahan, R., Power, J.F.: Metamodel Instance Generation: A systematic literature review. arXiv preprint arXiv:1211.6322 (2012)
20. McMinn, P.: Search - based software test data generation: A survey. Software Testing, Verification and Reliability 14(2), 105–156 (2004)
21. Bézivin, J., Schürr, A., Tratt, L.: Model transformations in practice workshop. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 120–127. Springer, Heidelberg (2006)