Filtered Comparison for Oracle in ModelTransformation Testing

Olivier Finot, Jean-Marie Mottu, Gerson Sunyé, J. Christian Attiogbe

Filtered Comparison for Oracle in Model Transformation Testing

Olivier Finot (PhD Student),
Jean-Marie Mottu, Gerson Sunyé, and Christian Attiogbe (Advisors)

LINA CNRS UMR 6241
University of Nantes
2, rue de la Houssinière
F-44322 Nantes Cedex, France
olivier.finot@univ-nantes.fr

Abstract. Focusing on one part of a produced output helps in improving model transformation testing

1 Introduction

Models are becoming a key element in software engineering. With Model Driven Engineering, they are the heart of the development process. They are used to describe a system at some state of its development, and they evolve with transformations. Model transformations can be chained, up to the production of executable code.
However, any error in a transformation of such a chain will be spread to the resulting code. But while testing the produced code might detect the bug, finding its origin will be difficult. Therefore, it would be useful to check the chain's development by verifying the transformations.
Several contributions have already been published on the verification of model transformation. Our goal is to pursue research on this field and improve existing methods to test model transformations. More specifically, we focus on the problem of the definition of a test oracle.
In Section 2, we present our proposal for the definition of a partial oracle for the test of model transformations[1]. Then, in Section 3 we discuss current and future work. Finally we conclude in Section 4.

2 Partial Oracle for Model Transformation Testing

Model transformations are automated to be highly reused. If we want to reuse a piece of software, we have to trust it. We can not have any errors in something we will reuse numerous times. We use test to ensure the correctness of a model transformation w.r.t. its specification.
The tester provides a valid input model, then she executes the transformation under test over this input model. Finally the test oracle controls the output model produced by the SUT's execution.

[1] a paper on the subject is currently under review for ICST13

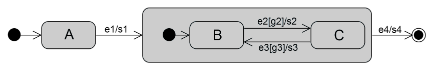
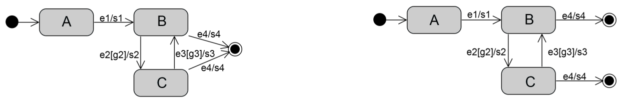


Fig. 1. Example M^{in}, of Hierarchical State Machine



(a) Variant M_1^{out} with only One Final State

(b) Variant M_2^{out} with Two Final States

Fig. 2. Possible Results for the Flattening of M^{in}

While several studies discuss the generation and selection of input models [1] [2] [3] [4], the oracle is seldom considered [5] [6].
Building test oracles, we face the transformation's output complexity. In some cases, the transformation's specification might allow several valid variants of a given output; the transformation here has polymorphic outputs. For example if we consider a program running an operation on a Finite State Machine (FSM) in order to flatten it, the input of this program is transformed into another FSM expressing the same behavior without using any composite state. We can transform the input model presented in Figure 1 into the output model depicted in Figure 2(a). With such state machines, the number of final states is not limited to only one. Thus, the FSM presented in Figure 2(b) is also a correct output for the flattening of the FSM presented in Figure 1.
While the implementation of such a transformation is deterministic, the specification allows several variants. The developer implements only one of these variants. However, when building an oracle, the tester must not consider the transformation's implementation, since she might be influenced by the errors made by the developer. Thus, she has to design an oracle that checks that the produced output of the Transformation Under Test is one of the possible variants.
The classical approach to build an oracle for model transformation testing, is to compare the produced output model with a reference one. The reference output model is the output model expected for the correct transformation of the input. Applying this approach to model transformations with polymorphic outputs would mean having the tester define one reference output model for each of the variants and then compare the produced output model to all of them. If the produced output model is identical to one of the reference output models, the test passes; otherwise it fails. Applying this classical approach to polymorphic outputs is time- and effort-consuming for the tester. She has to manually define all the reference outputs, and run the comparisons.
In [7], we propose a more efficient approach to build an oracle to test model transformations with polymorphic outputs. We notice that in the

polymorphic outputs of a model transformation some elements do not change from one variant to the other. In the example of the Figure 2, this is the case for the initial and simple states as well as for the transitions between them, they belong to the common part of the variants; the other elements form the variable part.
Our idea is to build an oracle focussing on this common part. Since by definition, the common part is identical in all the variants, this oracle will only need one reference output model as oracle data. The produced output model is compared to the reference one. The result of this comparison is then filtered in order to eliminate any difference concerning the variable part of the output model. If the filtered result of the comparison is empty, the common parts of both the produced and reference output model are identical, the test passes. Otherwise, the produced output model contains errors, an then the test fails.
Figure 3 summarizes our approach. We provide as oracle data the reference output model as well as patterns. In order to eliminate the differences about the variable part, we need to know which elements belong to this variable part. We define these elements according to their types (e.g. their meta-classes). The patterns we provide are meta-model excerpts defining the variable part of the transformation's output, they are defined only once for all the test cases of a given transformation.

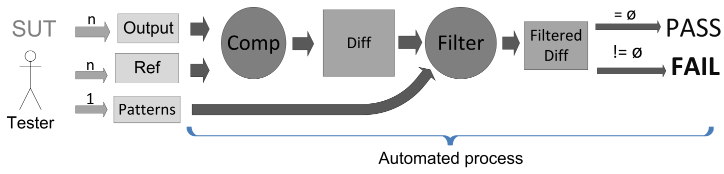


Fig. 3. Our Approach to Build a Partial Oracle

In our approach, we focus on controlling one part the produced output, producing a partial verdict. Nevertheless, a partial verdict is already a good piece of information. We are able to detect errors in the produced output model using only one reference output model, whereas the classical approach requires to define as many reference output models as correct variants of the output model.

3 Ongoing Work

To ensure the complete correctness of a model transformation with polymorphic outputs, the next step after the proposition of our approach is to be able to control the variable part of the produced output. Whereas for the common part we use a reference output model, the idea here is to work directly on the produced output model. The tester starts by checking that the expected elements for this variable part are present in the model (in Figure 2, transitions from the states B and C towards final states). Then she ensures that there is nothing else in the variable part (no other instance of the meta-model fragments used for the filter).
We discussed in Section 3 an approach aimed at partially controlling the polymorphic outputs of a model transformation. However, we believe

that the approach is not limited to these particular model transformations. Having a partial verdict could be interesting for other programs as well.
The larger the handled models become, the harder it is for the tester to define a reference output model. It is time and effort consuming for the tester to manually define a large and often complicated model. With our approach, she could use a partial reference output model to obtain a partial verdict about a part of the produced output model.
Our approach could be useful even outside the scope of model transformation testing. Models as well as graphs or databases can be seen as a particular kind of complex outputs. Complex outputs are not just big sets with many properties, these properties are organised and structured. Polymorphic outputs can also be seen as complex data, the complexity coming in this case from the numerous variants of a given output.
Our approach allows the tester to efficiently control part of a program's complex output and obtain a partial verdict. She only needs a partial reference output and a definition of the part she would not be interested in. The partial reference output could either be manually defined, or obtained through another version of the system under test in the case of regression testing.

4 Conclusion

The topic of this PhD is about test in a model driven engineering environment. Our first contribution is the proposition of an efficient approach to build a partial oracle controlling the common part of polymorphic outputs. We are currently working on controlling the remaining, variable part. Also while the proposed approach was defined to control the polymorphic outputs of a model transformation, it can be applied in other cases either inside our outside the scope of model driven engineering.

References

1. Sen, S., Baudry, B., Mottu, J.M.: On combining multi-formalism knowledge to select models for model transformation testing. In: ICST. (2008) 328–337
2. Fleurey, F., Baudry, B., Muller, P.A., Le Traon, Y.: Qualifying input test data for model transformations. SOSYM (2009)
3. Mottu, J.M., Sen, S., Tisi, M., Cabot, J.: Static analysis of model transformations for effective test generation. In: ISSRE. (2012)
4. González, C.A., Cabot, J.: Atltest: A white-box test generation approach for ATL transformations. In: MoDELS. (2012) 449–464
5. Mottu, J.M., Baudry, B., Le Traon, Y.: Model transformation testing : oracle issue. In: MoDeVVa. (2008)
6. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. ECEASST (2009)
7. Finot, O., Mottu, J.M., Sunye, G., Attiogbe, C.: Comparaison de modèles filtrée pour le test de transformations de modèles. In: Conférence en IngénieriE du Logiciel CIEL. (2012)