Towards the Automation of Metamorphic Testing in Model Transformations

Javier Troya, Sergio Segura, and Antonio Ruiz-Cortés

Department of Computer Languages and Systems
Universidad de Sevilla, Spain
{jtroya, sergiosegura, aruiz}@us.es

Abstract. Model transformations are the cornerstone of Model-Driven Engineering, and provide the essential mechanisms for manipulating and transforming models. Checking whether the output of a model transformation is correct is a manual and error-prone task, this is referred to as the oracle problem in the software testing literature. The correctness of the model transformation program is crucial for the proper generation of its output, so it should be tested. Metamorphic testing is a testing technique to alleviate the oracle problem consisting on exploiting the relations between different inputs and outputs of the program under test, so-called metamorphic relations. In this paper we give an insight into our approach to generically define metamorphic relations for model transformations, which can be automatically instantiated given any specific model transformation.

Keywords: Metamorphic Testing, Model Transformation, Automation, Generic

1 Introduction

Model Transformations (MTs) are the cornerstone of Model-Driven Engineering (MDE). They provide the essential mechanisms for manipulating and transforming models. Checking whether the output of a model transformation is correct is a manual and error-prone task, this is refereed to as the oracle problem in the software testing literature. Indeed, the quality of the generated software artifacts is highly affected by the correctness of the developed model transformations. For this reason, several approaches have been proposed that verify the correct behavior of the transformations using formal methods [6,1] or certify their behavior for a selected set of test models mainly to identity bugs in a cost-effective way [2,7].

Metamorphic Testing (MT') [5] is a methodology designed to alleviate the oracle problem. Different from conventional testing strategies, MT' consists on exploiting the relations between different inputs and outputs of the program under test, so-called Metamorphic Relations (MRs). In practice, MRs define possible modifications to a test input and how those changes are propagated to the program output. A basic example of MR can be defined for the program that computes the sine function. Let us suppose we want to know the exact value of $\sin(5)$. Is an observed output of 0.091 correct? A mathematical property of the sine function states that $\sin(x) = \sin(\pi - x)$, and we can use this to test whether $\sin(5) = \sin(\pi - 5)$ without knowing the concrete values of either sine calculation. Currently, the biggest limitation of MT' has to do with the

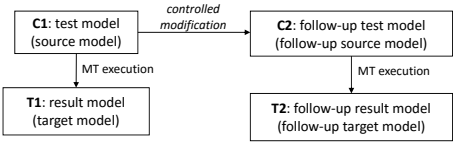


Fig. 1: Metamorphic Testing in Model Transformations.

definition of the MRs. In fact, the automatic generation of MRs has been acknowledged as one of the big challenges in MT' [5].

Figure 1 displays the scenario of MT' in MTs. We have a test model (C1), which is typically the source model. If we apply the MT, we obtain the result model (T1), i.e., the target model. By applying a controlled modification in the test model, we obtain the follow-up test model (C2). If we now apply the same MT to C2, we get the follow-up result model (T2). In this context, a MR considers the modification done in C2 with respect to C1, and the consequences that this has in T2 with respect to T1. We show an example in next section.

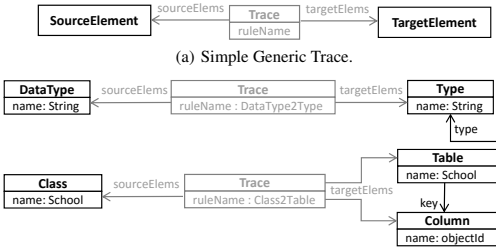
As far as we are concerned, there is only one approach that applies MT' in MTs [3]. The authors demonstrate the effectiveness and feasibility of its application, although they apply it manually in a specific scenario, for which they define the MRs. In our approach, we propose to automatically generate MRs for any model transformation, as we explain in the next section. Then, in Section 3 we describe our next steps.

2 Approach

The goal of our approach is to automate the process of metamorphic testing (MT') in model transformations (MTs). Thereby, we propose the automatic generation of metamorphic relations (MRs) for any model transformation. We work with transformations written in the ATL Transformation Language due to its importance both in academia and industry.

In order to automatically extract information out of a transformation, we make use of explicit trace models. A trace model can be automatically obtained from a transformation execution, e.g., by using Jouault's *TraceAdder* [4], and is composed of a set of traces, one for each rule execution. A trace captures the name of the applied rule and the elements of the source model (*sourceElems* relationship) that are used to create new elements in the target model (*targetElems* relationship). Therefore, by navigating the trace model, we know which target element(s) have been created from which source element(s) and by which rule. A simple example of a generic trace is shown in Figure 2(a). Please note that more than one element may appear as *sourceElems* and *targetElems*. We consider this trace as generic because each of the three elements appearing in it (*SourceElement*, *Trace* and *TargetElement*) can be instantiated in a particular scenario.

The idea of our approach is to define generic MRs for generic traces. These MRs can then be instantiated together with the generic traces. For instance, considering the generic trace of Figure 2(a), we know that if we have a test model (C1, Figure 1) and we add an element of type *SourceElement* in the follow-up test model (C2), then an



(a) Simple Generic Trace.

(b) Traces for the transformation excerpt of Listing 1.2.

Fig. 2: Generic and instantiated traces

element of type *TargetElement* is created for it in the follow-up result model (T2). This means that T2 has one more element of this type than T1. Having this information into account, we can define the first generic MR shown in Listing 1.1, written in the OCL language. Besides, the number of elements of any other type should remain the same in T1 and T2, so further MRs can be defined, such as the second one in the same listing.

Listing 1.1: Generic MRs for the addition of a *SourceElement*

```
1 T1_TargetElement.allInstances()->size()=T2_TargetElement.allInstances()->size()-1
2 T1_AnyOtherType.allInstances()->size()=T2_AnyOtherType.allInstances()->size()
```

In order to show an example of instantiation of the generic MRs shown before, we choose the well-known *Class2Relational* case study. We will focus on the excerpt of the transformation, which has been slightly modified for simplicity in the explanation, shown in Listing 1.2. If we have a source model with a *DataType* and a *Class* and we apply the transformation, the resulting trace model is depicted in Figure 2(b). We can see a trace that reflects the creation of a *Type* from a *DataType* and another one that stores the creation of a *Table* and a *Column* from a *Class*.

Listing 1.2: Excerpt of Class2Relational transformation

```
1  rule DataType2Type {
2    from
3      dt : Class!DataType
4    to
5      out : Relational!Type (
6        name <- dt.name
7      )
8  }
9
10
```

```
11 rule Class2Table {
12   from c : Class!Class
13   to
14     out : Relational!Table (
15       name <- c.name,
16       key <- key),
17     key : Relational!Column (
18       name <- 'objectId',
19       type <- thisModule.objectIdType)
20 }
```

Since these two traces are instantiations of the generic one shown in Figure 2(a), we can also instantiate the MRs shown in Listing 1.1. In particular, we have two scenarios. The first one consists of adding an element of type *DataType* in C2, what yields the

MRs shown in Listing 1.3. In the second scenario we add an element of type *Class* in C2, obtaining the MRs shown in Listing 1.4.

Listing 1.3: MRs for the addition of a *DataType* in C2

```
T1_Type.allInstances()->size()=T2_Type.allInstances()->size()-1
T1_Column.allInstances()->size()=T2_Column.allInstances()->size()
T1_Table.allInstances()->size()=T2_Table.allInstances()->size()
```

Listing 1.4: MRs for the addition of a *Class* in C2

```
T1_Column.allInstances()->size()=T2_Column.allInstances()->size()-1
T1_Table.allInstances()->size()=T2_Table.allInstances()->size()-1
T1_Type.allInstances()->size()=T2_Type.allInstances()->size()
```

3 Next Steps and Observations

In this paper we have given an insight into our approach to automate the generation of MRs for MTs. We identify generic patterns in the traces, from which we define generic MRs organized as well in patterns. For instance, one pattern is the generic trace and MRs we have shown in this paper. Despite its simplicity, we are performing ongoing works defining more patterns where elements, attributes and relationships are taken into account, so that we end up with a large set of MRs. One of the purposes of the generated MRs is to automate regression tests, since they can be checked as to whether they hold in different versions of the model transformation program and for any test model.

As mentioned, our approach takes as input one or more executions of a MT, i.e., the resulting trace models. The number of executions of the MT received and their size influence the completeness of the MRs generated. For instance, if a rule is never applied in any of the executions received as input, no MRs will consider its behavior.

Acknowledgments. This work has been partially funded by the European Commission (FEDER) and Spanish Gov. under CICYT project BELI (TIN2015-70560-R), and by the Andalusian Gov. projects THEOS (TIC-5906) and COPAS (P12- TIC-1867).

References

1. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A tridimensional approach for studying the formal verification of model transformations. In: Proc. of ICST. pp. 921–928. IEEE (2012)
2. Gogolla, M., Vallecillo, A.: *Tract*able Model Transformation Testing. In: ECMFA. LNCS, vol. 6698, pp. 221–235. Springer (2011)
3. Jiang, M., Chen, T.Y., Kuo, F., Zhou, Z., Ding, Z.: Testing Model Transformation Programs using Metamorphic Testing. In: SEKE'14. pp. 94–99 (2014)
4. Jouault, F.: Loosely Coupled Traceability for ATL. In: Workshop Proc. of ECMDA (2005)
5. Segura, S., Fraser, G., Sanchez, A., Ruiz-Cortes, A.: A survey on metamorphic testing. IEEE Transactions on Software Engineering (99), 1–1 (2016)
6. Troya, J., Vallecillo, A.: A Rewriting Logic Semantics for ATL. Journal of Object Technology 10, 5:1–29 (2011)
7. Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L.: Formal specification and testing of model transformations. In: Proc. of SFM. LNCS, vol. 7320, pp. 399–437. Springer (2012)