Verification of Graph-based Model Transformations Using Alloy

Xiaoliang Wang[1] and Yngve Lamo[1]

Bergen University College, Norway
{xwa,yla}@hib.no

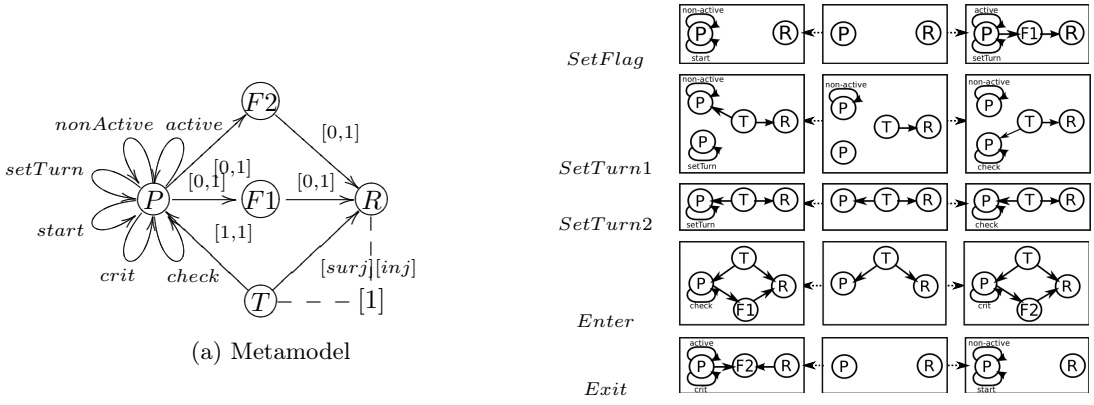
1 Introduction

In model driven engineering (MDE), models are considered the basis for software development. They are used to specify the domain under study, to generate program code and for documentation purposes etc. Ideally, a model in the next development phase can be automatically generated from a model used in the previous phase by model transformations. Such automation makes MDE appealing by offering more consistent software and higher productivity. However, validation of model transformations should be ensured. Without validation, errors in some transformations is transferred to the following phase, which may result in erroneous software. Usually, a model transformation is executed by applying model transformation rules on a model. A model transformation system consists of a set of such rules. Our work aims to verify if a model transformation system is correct w.r.t. conformance, i.e. for each valid source model is the target model obtained after the transformation still valid. We focus on graph-based model transformations [3] and present a bounded verification approach based on first order logic (FOL). The idea is to translate a model transformation system into a relational logic specification. Then we use the Alloy model analyzer[1], to check if any invalid target model are created by the transformation. To illustrate our approach, we run an example in Diagram Predicate Framework (DPF) [4], a framework which provides diagrammatic modelling and model transformations based on graph transformations. The example will be expressed in relational logic before the Alloy Analyzer [1] will be used to verify the system. Note that the approach can be proceeded automatically in DPF.

2 Verification Example

A model transformation system [3] consists of a metamodel $\mathcal{MM}$[1] and a set of model transformation rules $\{r : L \xleftarrow{\varphi} K \xrightarrow{\psi} R\}$. L, K, R are the left-hand side, the gluing graph and the right-hand side. The two morphisms φ and ψ are injective. L and R are typed by $\mathcal{MM}$, but is not necessary a valid instance of $\mathcal{MM}$. Figure 1 shows a variant of Dijkstra's algorithm for mutual exclusion [2] presented as a model transformation system in DPF. The algorithm ensures that a critical resource is accessed exclusively by one process each time. In DPF, the structural syntax is specified by a directed graph. R is the resource which processes P access. T tells which process is currently accessing R. The flags $\{F1, F2\}$ and the states $\{nonActive, active, start, crit, check, setTurn\}$ are used to control how to access R. $T \rightarrow P$ means that the P is eligible to access R. A reflexive arrow on P labeled with one of the six states means that the process is at such a state. An arrow from P to one of the two flags means

[1] For exogenous transformations, an integrated metamodel which interrelates the source and target metamodel can be construct. Then the integrated metamodel can be used as the metamodel of the model transformation system. Examples can be found in [3].



(a) Metamodel

(b) Transformation rules

Figure 1: Dijkstra Mutual Exclusive Model Transformation System

that P is marked with such a flag. Here, constraints are specified by diagrammatic predicates on part of the graph, where predicates are denoted by $[PredicateName]$. For example, each flag may have at most one arrow to R, which is ensured by the multiplicity constraint $[0, 1]$ on the respective arrows.

Based on the diagrammatic modelling framework, DPF also provides a framework to specify constraint-aware model transformations[5], which means that the transformation rules may contain constraints. However in this paper we only consider metamodel constraints. In the example, the metamodel is shown in Figure 1a while Figure 1b shows the model transformation rules. Rule $SetFlag$ requests access to R. Rule $SetTurn1$ and $SetTurn2$ assign T to one process depending on the context. Rule $Enter$ lets the eligible process access R, while rule $Exit$ finishes accessing R.

3 Verification Approach

We can use existing tools to specify model transformation systems, but most tools have no verification mechanism for model transformations. In this section we will propose a bounded verification approach, the idea is to translate a model transformation system to a relational logic specification. Each component; metamodel (including structure and constraints) and model transformation rules, can be encoded in the logic. In DPF the structure of a metamodel is a graph, hence we can use functions and constraints to express the graph. Those functions and constraints represent all the possible model instances typed by the graph. For example, nodes and edges in metamodels are encoded as Alloy signatures:

```
sig <NodeType>{}                    sig <EdgeType>{src, trg: one <NodeType>}
```

Here, an explicit constraint that each edge should have one source node and one target node, is translated as one in this signature. Metamodel constraints further restricts the instances of the metamodel. It should be noticed that, since our approach is based on FOL, constraints are restricted to those which can be expressed in FOL. For example, the constraint $[surj]$ on Arrow $T \rightarrow F1$ can be expressed as a $Fact$ in Alloy as follows:

```
fact{all n : SV_F1 | one e : SE_PF1 | e.trg=n}
```

In this work, we focus on graph-based model transformations using a classical double-pushout (DPO) approach [3]. For each transformation, a source model is transformed into a target model according to a model transformation rule. In the source model, the elements matched by the rule are deleted or transformed into target elements while others are unchanged. In the target model, the elements are added or transformed from source elements while others are unchanged. In this way, a model transformation can be viewed as a relation between the source and the target, which can also be expressed in FOL.

After we have encoded the model transformation system into a FOL specification, we can verify the system. In practice, we check if an invalid target model is created from a valid source model. If such an invalid target model is found, we can assert that the system is not correct w.r.t. conformance, otherwise, the system is assured correct. Note that the counterexample can help the designer to redesign the system.

Since we check if any invalid target model is transformed from some valid source model, constraints are handled differently depending on if they belong to the source or the target. Source constraints are translated as $fact$ and target constraints are translated into *check*. Then we use the Alloy Analyzer to verify the specification. We do not check all the rules simultaneously. A transformation may involve several rules. But the Parallelism Theorem [3] states that a transformation is equal to a sequential application of the rules. This enables us to check rules one by one. Besides, constraints are also checked one by one for simplicity. Besides, if we put all the target constraints together, it is not easy to analyse the counterexample when a system is not correct. The Alloy Analyzer performs check over a user-defined finite scopes. It means the approach can only verify a system in some finite scopes. The verification shows that the example is not correct. But adding some constraints on the source model and a Negative Application Condition (NAC) on rule $setFlag$ makes the system correct.

4 Conclusion and Future work

We have presented a bounded verification approach of graph-based model transformations. An example in DPF is given to illustrate the approach. The approach verifies a transformation system's correctness w.r.t. conformance by checking that each transformation from a valid source model can create a valid target model. However, for systems which have transformations creating invalid target models, correctness cannot be verified here. We will consider this in the future. Furthermore, since the Alloy Analyzer performs check over a user-defined scope, the approach is incomplete in that it cannot check the instances out of the scope. Moreover the limit of the scope which can be handled with the approach is not clear, this should be explored.

References

[1] Alloy. *Project Web Site*. `http://alloy.mit.edu/community/`.

[2] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*. 2001.

[3] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science)*. Springer. 2006.

[4] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A diagrammatic formalisation of mof-based modelling languages. *Objects, Components, Models and Patterns*, 2009.

[5] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A formal approach to the specification and transformation of constraints in mde. *Journal of Logic and Algebraic Programming*, 81(4), 2012.