Correctness of Constraint-Aware Model Transformations

Xiaoliang Wang, Yngve Lamo
Bergen University College
{xwa,yla}@hib.no

Model transformations are important in Model Driven Engineering (MDE). They automate software development steps and greatly improve productivity and reduce software errors. However, the design of model transformation rules requires lots of manual work. To fully take advantage of MDE, correctness of model transformation rules should be ensured. In this paper, we present an ongoing work to use model checking techniques to validate model transformation rules. The work also studies how rule application strategies affect correctness and efficiency of model transformations.

1 Introduction

Model driven Engineering (MDE) turns out to be a promising software development methodology. MDE promotes the use of models as the primary artefacts in software development. Models are used to specify, simulate, generate code and maintain the resulting applications. They are manipulated by model transformations throughout the software development life cycle. In that way, consistence between models is assured and productivity is greatly improved.

In MDE, models are described in modelling languages. The most prevailing modelling language is the Unified Modelling Language (UML) [4] proposed by OMG [4]. The syntax of a modelling language is usually specified by a metamodel. For example, the metamodel of a UML model is also specified by UML. In that way, a UML model is an instance of and conforms to UML. Besides, constraints are also required to define the properties that a valid instance of the model should possess. UML uses a text-based language, the Object Constraint Language (OCL) [4] to define software constraints. Any valid UML model must satisfy the given OCL constraints.

1.1 Model Transformations

A model transformation is the automatic generation of target models from source models. The relation between source model elements and target model elements is usually specified by model transformation rules. Given a source model, a target model is expected to be constructed by applying a sequence of rules. To define the rules, a language that coordinates both the source metamodel and the target metamodel is needed. OMG has invented the Query/View/Transformation (QVT) language [4] that has become the de facto industry standard model transformation language.

Model transformations play an important role in MDE. Models are modified by model transformations for different development purposes. For example, when deploying an application, a Platform Independent Model (PIM) should be transformed into a suitable Platform Specific Model (PSM) in accordance to the hardware requirements. These models should be used for automatic code generation.

However, right now, model transformation rules are designed manually. In order to ensure reliability, it is necessary to check the correctness of the model transformation. A match of a rule in a model means a graph homomorphism can be found from the left hand side of the rule to the model. If a match of a rule is found in a model, we say that the rule is applicable to the model. A correct model transformation means that for any valid source model, a sequence of applicable rules which constructs a valid target model can be found.

Besides, the choice of which rule to apply also influences what kind of target model is constructed out of the source model. The rule application strategy controls which rule should be applied when several rules are applicable at the same time. It also decides the resulting target model when several matches

of a rule are found in the model. So the rule application strategies are also important when checking correctness.

There are existing projects like Henshin [7] that use model checking techniques to check correctness of model transformations. But so far we are not aware of any projects that uses model checking for constraint-aware models transformations. This work presents a model checking approach to validate constraint-aware model transformations. Moreover, application strategies are also studied to analyse their impact to the correctness of model transformations. The proposed approach is based on the Diagram Predicate Framework (DPF) [5] which provides a formalization of (meta)modelling and model transformation based on graph theory [2] and category theory [1].

2 Diagram Predicate Framework

DPF aims to build a fully *diagrammatic specification framework* for MDE. That is to develop and use a *diagrammatic formalism* to define and reason about models and *model transformations*. In DPF, models are formalized as diagrammatic specifications which consist of an underlying graph structure together with a set of atomic constraints. A modelling language is formalized as a modelling formalism $(\Sigma_2 \rhd S_2, S_2, \Sigma_3)$. The specification S_2 represents the metamodel of the language; the signature Σ_3 contains predicates which are used to add constraints to the metamodel S_2; while the typed signature $\Sigma_2 \rhd S_2$ contains predicates which are used to add constraints to the specification S_1 that are specified by the modelling formalism. DPF also takes constraints in model transformations [6] into account. Based on this, given a modelling formalism $(\Sigma_2 \rhd S_2, S_2, \Sigma_3)$, constraint-aware rules are formalized as a typed specification morphism $r : \mathfrak{L} \rhd S_2 \hookrightarrow \mathfrak{R} \rhd S_2$. Usually, the modelling formalism used in model transformation is a joint one which relates source and target modelling languages together.

2.1 Correctness of Model Transformations

Software programs need validation before deployment. Testing is enough for less critical applications. For more critical applications, reliability could be ensured by use of theorem provers or model checkers. Testing can never completely identify all the defects, but it can help the programmer to find bugs and refine the program. The use of theorem provers and model checkers can guarantee programs without bugs. But the application of theorem provers needs a mathematical formalization of the program and involves human activities whereas model checkers have the state explosion problem.

In a similar way validation of model transformations is also required. Model transformations are executed automatically, hence it could be possible to run automatic tests of model transformations. Hopefully, if the test result is true, a sequence of applicable rules will be given that constructs a desired target model. Otherwise, it will give feedbacks assisting the designers to correct the rules. However, there is a difference of testing programs and transformations. For any determinstic program, each input only have one execution path. For a model transformation, there maybe several different sequences of applicable rules. To validate correctness of a model transformation, all the possible sequences should be checked by a model checker. By considering models as states and rules as transitions, a model transformation can be translated into a finite state machine. In order to test the transformation rules by use of model checking, the source model and transformation rules should be translated into a Kripke structure. Before continuing, a short introduction about model checking is given.

2.2 Model checking

Model checking is an automatic way to verify that a model satisfies a given specification. Usually, the model is represented by a Kripke structure, while the specification is formalized in temporal logic, e.g. CTL or LTL [3]. Formally a Kripke structure is defined as a 4-tuple, $M = (S, I, R, L)$, where S is a finite set of states s. I is a set of initial states $I \subseteq S$. R is a transition relation $R \subseteq S \times S$ such that R is

left-total, i.e., $\forall s \in S, \exists s' \in S$, such that $(s, s') \in R$. And L is a labelling function $L : S \to 2^{AP}$, where AP is a set of *atomic propositions* [3].

Recall that the joint modelling formalism (JMF) includes the source metamodel (SMM) and the target metamodel (TMM). We now give the translation procedure from DPF model transformations to Kripke structures that are used in model checking. From the joint modelling formalism (JMF), the model transformation rules (MTRs) and the source model(SM), we construct a Kripke structure in the following way:

- We define a initial state i representing SM
- For each state $s \in S$ and for every MTR $r : \mathfrak{L} \rhd S_2 \hookrightarrow \mathfrak{R} \rhd S_2$ we check $IsMatch(Model, \mathfrak{L} \rhd S_2)$. If it is true, the rule is applicable
- For each state $s \in S$ and for every applicable MTR $r : \mathfrak{L} \rhd S_2 \hookrightarrow \mathfrak{R} \rhd S_2$, we define a new state $r(s) \in S$ and a transition $t : s \to r(s)$

Note that $IsMatch$ checks if a match of the left input pattern is found in $Model$. If so, a new state $r(s)$ and a new transition $t : s \to r(s)$ are created. An important factor here is the rule application strategy. Two situations are under consideration. Firstly, several matches may be found in a model for one rule. Secondly, several rules may be applicable to the current model. Different strategies to control these scenarioes results in different target models. Existing strategies are e.g. negative application conditions (NAC) and layering of rules. Analysis of affection of those strategies can be carried out during the model checking. Moreover, transformations are constraint-aware in DPF. That means when constructing the state space and the transition relationship the related constraints should be considered.

Following this procedure a state space can be derived. It contains all the reachable states, that are models derived by application of rules and valid instances of the JMF. The property to be checked is that in the future there is a state where no more rule is applicable and from this state a valid target model can be derived. In CTL, this property can be formalized as:

$$EF!AnyRuleApplicable(Model, MTRs)\&\&IsInstanceof(getTargetModel(Model), TMM)$$

Existing model checking technologies can be used to check if the property is satisfied. Different instances of the source metamodel can be created to test the MTRs. A valid sequence of applicable rules are obtained if the result is true. Otherwise feedback will be given to help the designer to modify the model transformation. As said before, testing can never guarantee transformation correctness, in the future we will study how theorem provers can be used to ensure correctness of model transformations.

References

[1] M. Barr and C. Wells. *Category Theory for Computing Science (2^{nd} Edition)*. Prentice Hall, 1995.

[2] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, March 2006.

[3] E. C. Jr., O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[4] Object Management Group. *Web site*. http://www.omg.org.

[5] A. Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.

[6] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Formalisation of Constraint-Aware Model Transformations. In D. Rosenblum and G. Taentzer, editors, *FASE 2010*, volume 6013 of *LNCS*, pages 13–28. Springer, 2010.

[7] The EMF Henshin Transformation Tool. *Project Web Site*. http://www.eclipse.org/modeling/emft/henshin/.