Towards user-friendly and efficient analysis with Alloy

Xiaoliang Wang, Adrian Rutle, and Yngve Lamo

Bergen University College,
P.O. Box 7030, N-5020, Bergen, Norway
Email:xwa,aru,yla@hib.no

In Model-driven engineering (MDE), structural models (also called static models in [6]) represent software at the early phases of software development. They identify the artifacts and their relationships in the problem domain. These models can be specified as graph-based structures and constraints in different formalisms, e.g., UML class diagram [8] and the Object Constraint Language (OCL) invariants [7]; in the structures, nodes represent the artifacts while edges represent the relationships; the constraints express requirements of the problem domain. An instance of a structural model is a graph which is well-typed by the underlying graph of the model, and, in addition, satisfies all the constraints of the model.

Usually, structural models are specified by a modelling language within a modelling tool; this may cause errors. Thus, these models should be verified to ensure correctness. In addition, in MDE, models are gradually refined in subsequent phases which then finally result in software. Therefore, the verification of models can avoid propagating of errors into the software. Moreover, it is obvious that finding design mistakes as early as in the modelling phase helps to build better software at a lower cost.

Different properties of structural models are studied in MDE [3]. For instance, *consistency* requests that a model has at least one instance; *lack of redundant constraints* requests that, given a model, there exists no constraint C_1 that can be derived from another constraint C_2, i.e., there exists at least one instance of the model which satisfies C_1 but not C_2. These properties can be categorised into *validity*, i.e., whether all the instances of a model satisfy a property, and *satisfiability*, i.e., whether there exists an instance which satisfies a property.

Several approaches have been presented to verify such properties on structural models [6]. Generally, they translate a structural model and a property into a specification in some formalism, e.g., Relational Logic [2, 4], etc. Then the specification is analysed by theorem provers or constraint solvers to answer whether the model satisfies the property. But these approaches are not integrated into the modelling tools; to use these approaches, the model designers have to switch from the modelling tool to a verification tool and need background in the verification methods. Moreover, most approaches present instances when the properties are satisfied, but give no feedback when the properties are violated. This is not convenient for model designing.

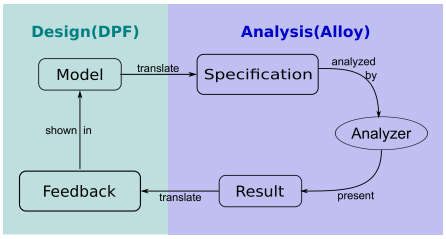

Figure 1: Workflow for analysing structural models using Alloy

In this work, we present a bounded verification approach of structural models using Alloy [1] and integrate it into a modelling tool DPF Model Editor [10]. The procedure of the approach is

illustrated in Fig. 1. It translates a structural model specified in Diagram Predicate Framework (DPF) [9] and a property into an Alloy specification. Then, the specification is examined by the Alloy Analyzer to check if the model satisfies the property or not. If the property is satisfied (violated), an instance (counterexample) of the model is generated. Otherwise, it means that there are some problems with the model. Then, the problematic part of the model will be highlighted and displayed in the DPF Model Editor to assist the model designer to identify the problem. For example, a civil status model which modifies the traditional civil status model in [5] originally specified in UML and OCL is present in Figure 2. It is inconsistent and the constraints which contradict each other are highlighted and presented in Figure 3. Thus, the model designer can verify the model under design and receive user-friendly feedback which he can understand, without knowing the underlying verification technique[1].

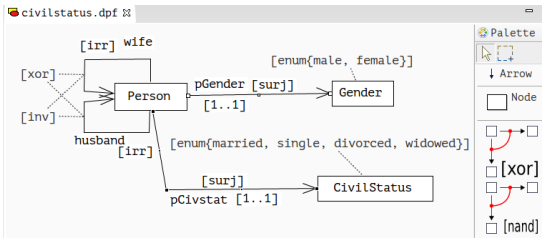

Figure 2: Civil Status Model in DPF

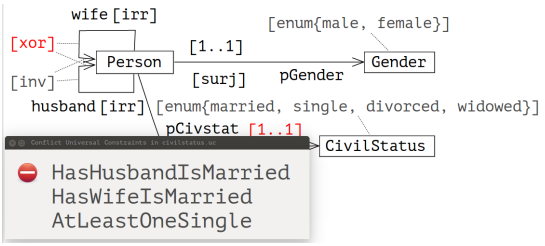

Figure 3: Hightlight the problem

The approach is bounded; the approach finds instances or counterexamples which satisfies or violated properties within a bounded search space. The space is determined by a *scope*; i.e., a user-defined number which restricts the number of instances of each model element. However, there is no systematic way to decide which scope is needed, and, although different scopes could be sufficient for different model elements, the same size is usually used for all model elements. In addition, this approach has scalability problems since the search space grows exponentially along with the scope. It means that the verification of large models with a large search space may take long time or become intractable.

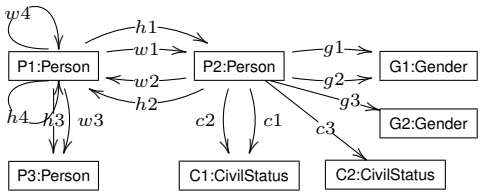

Figure 4: The scope graph for the civil status model

To solve these issues, we propose two contributions. The first one is to initialise a systematic way to decide the search space based on constraints (properties) definitions; here we focus on constraints which can be expressed in FOL. Given such constraints, we assume that there exists a *scope graph* such that, for each instance (counterexample) of the model, there exists smaller or equal instances (counterexamples) that are contained by the scope graph. However, since FOL is undecidable, such a scope graph do not exist for arbitrary constraints. In this work, we construct an approximation of the scope graph based on the syntax of the constraints. The

[1]This part which verifies structural models and presents the result of verification user-friendly, along with the splitting technique which is present in the sequel, is submitted to and accepted by Modevva2015.

approximation of the scope graph can be used to derive a scope to verify a structural model. For example, the approximation of the scope graph for the civil status model is shown in Figure 4. It contains 3 **Person**, 4 **husband**, 4 **wife**, 3 **pGender**, 3 **pCiviStat**, 2 **Gender**, 2 **CivilStatus**. This can be used as the scope for the consistence check of the model.

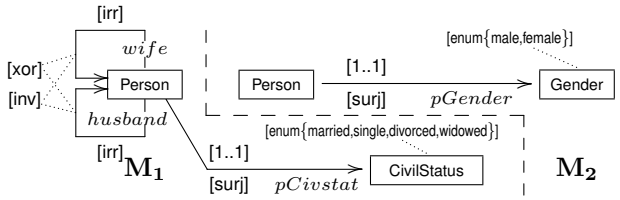

Figure 5: Submodels

The second contribution is a splitting technique for verification. A model can be split into submodels based on the *factors* of the constraints, i.e., the model elements which are affected by the constraints. We will look for submodels which are *left-total*, i.e. submodels of which the instances can be extended to instances of the model. We outline an approach to find these left-total submodels based on *forbidden patterns* of the constraints. That is, these submodels do not contain any match of patterns which violate (or are forbidden by) any constraints of the model. Then the validation of a model can be reduced to the validation of its left-total submodels. The civil status model can be split into two submodels which are shown in Figure 5. The submodel M_1 is left-total and the consistence check of the model can be reduced to the consistence check of M_1 rather than the whole model. An experimental result shows that it alleviates the scalability problem.

References

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

[2] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from UML to Alloy. *Software and Systems Modeling*, 2009.

[3] Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93:1–23, 2014.

[4] Martin Gogolla, Jörn Bohling, and Mark Richters. Validating UML and OCL models in USE by automatic snapshot generation. *Software and System Modeling*, 4(4):386–398, 2005.

[5] Martin Gogolla, Fabian Büttner, and Jordi Cabot. Initiating a benchmark for UML and OCL analysis tools. In *7th TAP*, pages 115–132, 2013.

[6] Carlos A. González and Jordi Cabot. Formal verification of static software models in MDE: A systematic review. *Information & Software Technology*, 56(8):821–838, 2014.

[7] Object Management Group. *Object Constraint Language Specification*, February 2014. `http://www.omg.org/spec/OCL/2.4/`.

[8] Object Management Group. *Unified Modeling Language Specification*, May 2015. `http://www.omg.org/spec/UML/2.5/`.

[9] Adrian Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, University of Bergen, 2010.

[10] Lamo Yngve, Wang Xiaoliang, Mantz Florian, Bech Øyvind, Sandven Anders, and Rutle Adrian. DPF Workbench: a multi-level language workbench for MDE. In *Proceedings of the Estonian Academy of Sciences*, pages 3–15, 2013.