Towards a Two Layered Verification Approach for Compiled Graph Transformation

Ákos Horváth

Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
1117 Budapest, Magyar Tudósok krt. 2.
ahorvath@mit.bme.hu

1 Introduction

As model driven software development (MDSD) is being applied more and more in the safety critical (SC) and dependable system development processes there is an increasing need for verified model transformations to guarantee certain semantic properties to hold after their execution. For instance, when transforming UML models into Petri nets, the results of a formal analysis can be invalidated by erroneous model transformations when the system developer cannot easily distinguish whether an error is in the design or in the transformation.

In this paper we introduce our vision for verifying property preservation of graph transformation systems with a two layered approach.

2 Overview of the Approach

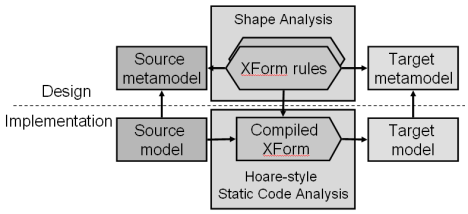

Fig. 1. Overview of the approach

Fig 1 gives an overview of our common model transformation process. The model transformation (*XForm rules*) is specified by a number of graph transformation rules. The GT rules are specified with respect to the metamodels of the source and the target metamodel. From these rule specifications, a compiled transformation (*compiled XForm*) is generated (see in [1]). The automatically derived compiled *XForm* transformation transforms a source model into a target model.

Our goal is to verify property preservation for the compiled transformation, meaning that, if a certain property holds in the source model after executing the transformation it will also hold in the target model. To achieve this we separated the verification process into two steps.

– First, we plan to apply shape analysis [2] on the *XForm* rules to summarize the behavior of a statement on an infinite set of possible rundown states of the GT rules. Shape analysis concerns the problem of determining *shape invariants* for

programs that perform destructive updates on dynamically allocated storage. This way correctness of transformation rules applied to *any* model of the specified type can be verified (the concrete instances of the metamodels are irrelevant for the proof).

– Then, as the result of the shape analysis is based on the assumption that the GT rule specification are ”executed” semantically correct, in the second step we focus on the correctness check of the compiled GT rules. As the correctness of the generated compiled code depends on the correctness of the generator itself, that is usually a complex software components which cannot be verified easily. We use an alternative assurance approach, in which the generator is extended with formal program specification to enable Hoare-style [3] safety analysis for each individually generated GT rule. The crucial step in this approach is to extend the generator to produce all required annotations without compromising the assurance provided by the subsequent verification phase.

2.1 Analysis of Model Transformation Specification

Shape analysis: In our approach we plan to use the TVLA [4] (Three-Valued-Logic Analyzer), a system for automatically generating a static (shape) analysis implementation from the operational semantics of *XForm* rules. The small-step structural operational semantics is written in a meta-language based on first-order predicate logic with transitive closure. The main idea is that program states are represented as logical structures and the program transition system is defined using first order logical formulas. TVLA automatically generates the abstract semantics, and, for each program point, produces an abstract representation of the program states at that point. TVLA relies on a fundamental abstraction operation for converting a potentially unbounded structure into a bounded 3-valued structure (logic). 3-valued logic extends boolean logic by introducing a third value 1/2 denoting values that may be 0 or 1. A 3-valued logical structure can be used as an abstraction of a larger 2-valued logical structure. This is achieved by allowing an abstract state (i.e., a 3-valued logical structure) to include summary nodes, i.e., individuals that correspond to one or more individuals in a concrete state represented by that abstract state.

Our initial examples with the TVLA system shows that the mapping of the metamodel to the TVLA is a key problem for efficient shape analysis generation.

2.2 Analysis of Model Transformation Implementation

Hoare-style platform specific code analyzers: Hoare logic is a formal system to provide a set of logical rules in order to reason about the correctness of computer programs with the rigour of mathematical logic. The central feature of Hoare logic is the *Hoare triple*. A triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple is of the form $\{P\}\,C\,\{Q\}$ where P, Q and C are *precondition*, *postcondition* and *command*, respectively. Based on the concept of pre-/postcondition introduced in the Hoare triple, *design by contract* [5] (DBC or programming by contract) prescribes that software designers should define precise verifiable interface specifications (pre/postconditions) for software components based upon the theory of abstract

data types and the concept of a business contract. This means that *contracts* provides semantics to formally describe the behavior of a program module, removing potential ambiguity with regard to the module implementation.

Tools built upon the DBC methodology include the logic of predicate calculus and Dijkstra's weakest precondition calculations. We focused our studies on two of the most widely used frameworks: the (i) *Spec#* [6] programming system having developed at Microsoft Research to extend C# with formally verifiably method contracts in the form of pre-/postconditions as well as object invariants, and the (ii) *KeY* [7] formal software development system built upon a semi-automated prover over the Java Dynamic Logic (JavaDL) calculus (with support to Java Modeling Language (JML)) which covers the complete Java Card language, and additionally supports some Java SE features such as multi-dimensional arrays and dynamic object creation.

Both approaches look promising but our initial experiments show that none of them provide efficient support for: (i) dynamic casting of complex data structures (e.g., arrays), (ii) effective handling of nested loop invariants, (iii) contracts for library functions and finally (iv) user-friendly feedback from proof obligations.

3 Conclusion and Future Work

We have presented an ongoing work how graph transformations can be verified with a combination of shape analysis (with TVLA) and static code analyzer (e.g., Spec#, KeY). In the current state of our research, we have studied the boundaries of Hoare-style static code analyzers with respect to complex object navigation (as being the core of transformation implementation). It resulted in state space explosion in case of common implementations of GT rules and have to be further studied to achieve analyzable implementation.

As for the future, we plan to finish formalizing GT rules in 3-valued logic to achieve feasible shape analyze results and adapt the model logic described in [8] to capture properties of graph models.

References

1. Balogh, A., Varró, G., Varró, D., Pataricza, A.: Compiling model transformations to EJB3-specific transformer plugins. (April 2006) 1288–1295
2. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3–valued logic. In: Symposium on Principles of Programming Languages, ACM Press (1999) 105–118
3. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10) (1969) 576–580
4. Lev-Ami, T., Manevich, R., Sagiv, S.: Tvla: A system for generating abstract interpreters. In Jacquart, R., ed.: IFIP Congress Topical Sessions, Kluwer (August 2004) 367–376
5. Meyer, B.: Applying "design by contract". Object-Oriented Systems and Applications **25**(10) (October 1994) 40–51
6. Spec#: The Spec# programming system `http://research.microsoft.com/specsharp/`.
7. The KeY Project: Integrated deductive software design `http://www.key-project.org/`.
8. Boneva I.B. and Rensink A. and Kurban M.E. and Bauer, J.: Graph abstraction and abstract graph transformation. Technical Report TR-CTIT-07-50, University of Twente (2007)