SyVOLT: Full Model Transformation Verification Using Contracts

Levi Lúcio, Bentley James Oakes[†], Cláudio Gomes[‡], Gehan Selim[⋆], Juergen Dingel[⋆], James R. Cordy[⋆], and Hans Vangheluwe[†‡]

[†]McGill University, Montreal, Canada [‡]University of Antwerp, Belgium,
[⋆]Queen's University, Kingston, Canada

Problem Statement

We introduce SyVOLT, a plugin for the Eclipse development environment for the verification of structural pre- /post-condition contracts on model transformations. The plugin allows the user to build transformations in our transformation language DSLTrans using a visual editor. The pre-/post-condition contracts to be proved on the transformation can also be built in a similar interface. Our contract proving process is exhaustive, meaning that if a contract is said to hold, then the contract will hold for all input models of a transformation. If the contract does not hold, then the counter-examples (i.e., input models) where the contract fails will be presented.
Demo: `https://www.youtube.com/watch?v=8PrR5RhPptY`

SyVOLT Highlights

Input Independence and Exhaustiveness

SyVOLT proves that pre-/post-condition contracts hold for a model transformation. Such contracts establish relations between patterns occurring in input and output models of a model transformation. If a contract holds, a formal guarantee exists that whenever a transformation?s input model contains the pattern specified in the pre-condition of the contract, the transformation?s output model will contain the pattern specified in the contract?s post-condition. Contracts can op- tionally include traceability relations between input and output patterns. Our technique is exhaustive and input-independent, in the sense that whenever a contract holds, it will hold for all possible input models for that transformation. This is possible because SyVOLT operates on specifications of out-place model transformations, where unbounded loops and model element deletions are not allowed. A discussion on the soundness and completeness of our approach is provided in [16].

Push-Button Proofs

The proving process for a SyVOLT contract is fully au- tomatic and all of the approach?s formal details are com- pletely hidden from the user. Once the transformation and the contracts of interest are created, one command will start the property proving process. This process will automatically create all required artifacts (as detailed in Section III), run the process, and provide the results to the user within the Eclipse environment. This allows the user to continually stay within the Eclipse environment, where he or she develops the contracts and the model transformations.

Based on Symbolic Execution

Our technique shares its principles with symbolic execu- tion, a classic method to verify code. The underlying idea entails building a finite representation of the (infinite) set of computations that can be expressed by a model transformation specification. In this context, each symbolic execution ? which in the context of our work we call a path condition ? is an overlapping combination of a subset of the transformation?s rules. Because a path condition contains a number of rules, it represents the execution of the model transformation over any input model those rules match on. Contracts of interest are proved on the set of path conditions built for a model transformation, and are extrapolated to the infinite set of the model transformation?s computations through an abstraction relation [16].

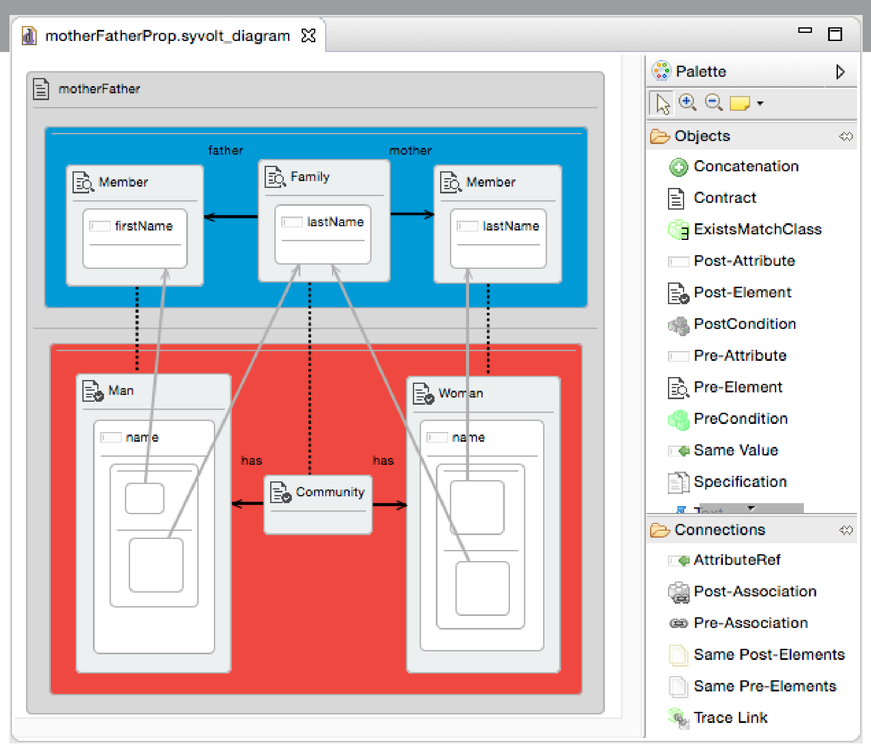
Proving Contracts about ATL Model Transformations

The Atlas Transformation Language (ATL) [1] is commonly-used in both industrial and academic applications. In order to enable contract proving on ATL transformations, we have developed a higher-order transformation that is able to automatically transform declarative ATL transformations into DSLTrans transformations [19]. In the future we will integrate this higher-order transformation into SyVOLT?s user interface.

Scalability and Speed

We have some evidence that SyVOLT scales to transfor- mations of practical interest. In particular we have verified contracts on DSLTrans transformations with up to over 60 rules, and ATL transformations with up to 13 rules [19]. From our own experience, the size of a DSLTrans transformation ranges from 10 up to 50 rules, while the average size of an ATL transformation is around 20 rules [15]. Even though our technique is exhaustive, our experiments show that verifica- tion can be performed within seconds. Gehan Selim?s PhD thesis [22] provides further evidence of SyVOLT?s speed, by verifying a relatively large model transformation for giving semantics to the UML-RT language in terms of the Kiltera process language [20]. SyVOLT?s symbolic execution engine is fully homegrown [17] and does not depend on third-party solvers. Although this has implied a large effort to build the codebase, it has allowed us to have the required control over the code to iteratively optimize the engine for space and time economy. [23] demonstrates that our prover is substantially faster than similar approaches based on SAT solvers.

Integration with Eclipse / Graphical Modelling



Eclipse Frontend

Eclipse is a popular development environment, as many model transformation tools such as ATL, DSLTrans [10] and EGL [3] are integrated with the Eclipse Modeling Framework (EMF) [2]. To take advantage of this ecosystem, SyVOLT integrates with EMF to represent models in a multitude of syntaxes, from graphical to textual. Modellers may then operate in their preferred syntax, although the authors suggest the visual representation of a contract in the SyVOLT editor allows for intuitive understanding of the contract?s meaning.

Counter-Examples

When a given contract does not hold on a given model transformation, SyVOLT can produce additional information for the user to pinpoint where the contract?s violation occurs. This information is in the form of the set of model transforma- tion rules used to build a particular path condition for which the contract fails. A counter-example is any input model where this set of rules would execute. For example, the sample output in Figure 1 alerts the user that the contract motherFather will fail when only the mother and father rules execute in the transformation.

```
 〉 Proving contracts:
〉 Contract ‘‘daughterMother’’ holds for all input models!
〉 Contract ‘‘motherFather’’ does NOT hold for all input models!  The
contract fails on the following Path Conditions:
['EmptyPathCondition_RootRule_FatherRule_MotherRule', ...]
〉 The smallest Path Conditions where the contract fails are:
['EmptyPathCondition_FatherRule_MotherRule']
〉 Time to verify 2 contracts:  11.6834638966 seconds.
```

Architecture

The following model-driven development tools have been used in SyVOLT?s