Verification of Translation Model Transformations Validity and Completeness Results

Levi Lúcio[†], Bentley James Oakes[†], and Hans Vangheluwe[‡†]

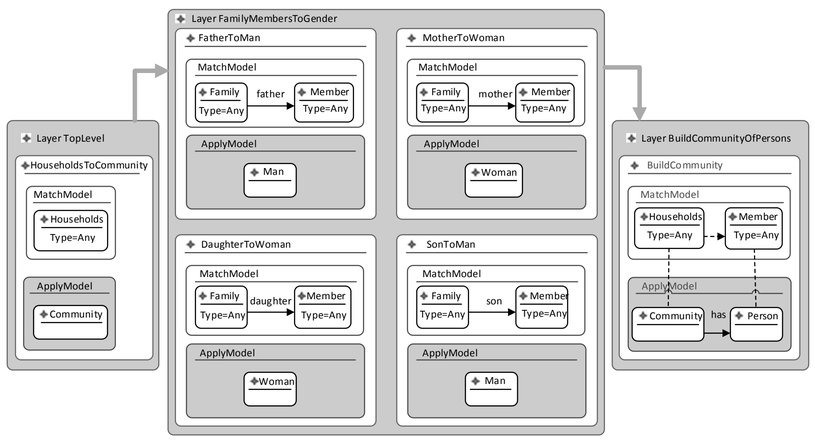
[†]McGill University, Montreal, Canada [‡]University of Antwerp, Belgium

Problem Statement

Our transformation verification approach employs a technique inspired from symbolic execution in order to automatically build a set of path conditions for a DSLTrans transformation [1, 2]. Each path condition abstractly represents a number of concrete transformation executions. Based on this formally defined abstraction relation, we show that **our path condition generation algorithm is both *valid* and *complete***. Using **path conditions**, we are then able to prove model syntax relation properties over all transformation executions in a model-independent way. This is done by examining if the properties hold on the set of path conditions. Through the **abstraction relation**, if the property does not hold on a path condition, the property will not hold on the transformation executions abstracted by the path condition. We show **this property proving step is both *valid* and *complete***.

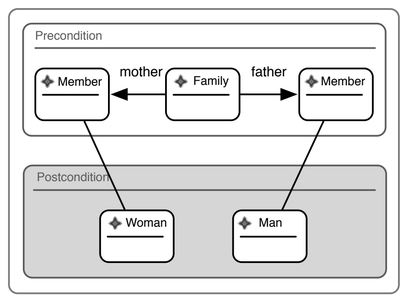
Main Artifacts in Model Transformation Verification

DSLTrans Transformation:



Six rules arranged in three layers that execute sequentially. Each rule is composed of a Match Model and an Apply Model, analogous to the LHS and RHS in other transformation formalisms.

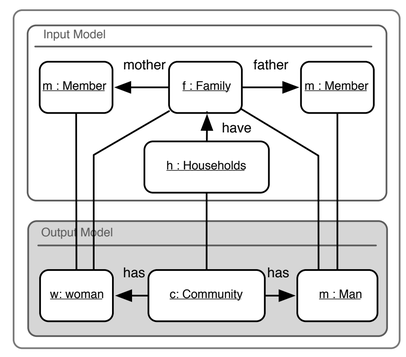
Property:



Model Syntax Relation *AtomicContract* with pre- and post- conditions

"the mother and father in a family will always be transformed into a woman and a man"

Transformation Execution:



Input and resulting Output

Traceability links record which output elements were created from which input elements

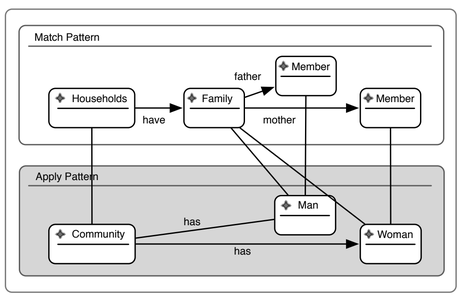
Proving Properties

The aim of our procedure is to prove *model syntax relation* properties on all transformation executions. However, as there are **infinitely many transformation executions**, the proof requires an **abstraction**.

Path conditions abstract transformation executions. A path condition represents a combination of rules of the transformation that can execute. In a path condition each rule is represented only once, granting the required abstraction and ensuring the generation of a finite number of path conditions.

Abstraction Relation

Example Path Condition:



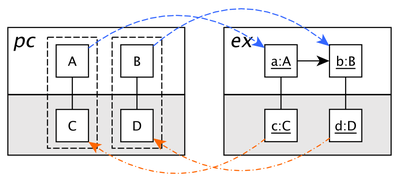
Path conditions are created by combining transformation rules [3]

A **Path Condition** *abstracts* a transformation execution when:

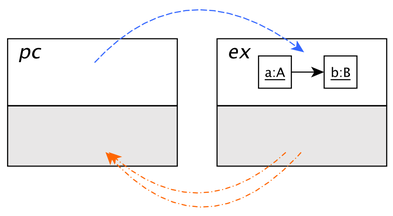
- An ***Injective Typed Graph Homomorphism*** exists between the match part of each rule (component) in the path condition and the input of the transformation execution, **and**
- A ***Surjective Typed Graph Homomorphism*** exists between the output of the transformation execution (including traceability links) and the apply graph of the path condition.

Abstraction Holds

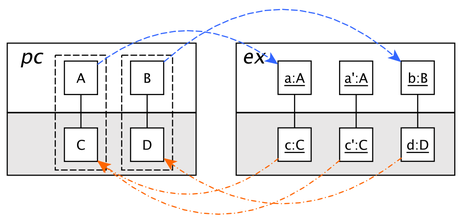
Injective matches into different Transformation Execution elements



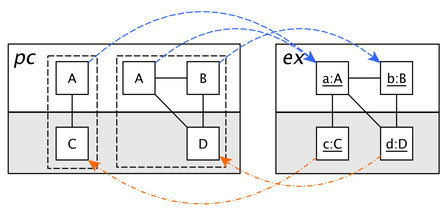
Empty Path Condition



Injective matches into different Transformation Execution elements

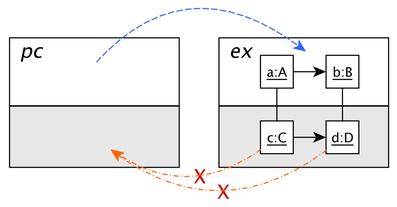


Injective Matches into same execution elements

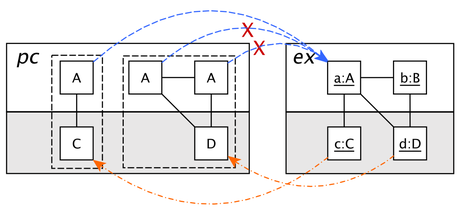


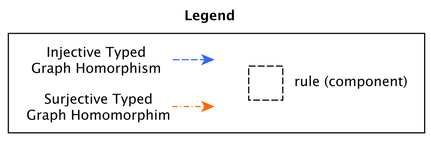
Abstraction does not Hold

Empty Path Condition



Non-Injective Match





Path Condition Generation Validity/Completeness Theorems [3]

Validity: Every path condition abstracts at least one transformation execution.

Completeness: Every transformation execution is abstracted by one path condition.

Uniqueness: Every transformation execution is abstracted by ***exactly*** one path condition.

Property Proving Validity/Completeness Theorems [3]

Validity: Checking a property on a path condition and on a transformation execution that path condition abstracts produces the same result.

Completeness: A property can be proved to hold or not hold on all transformation executions.

Future Work

- Integrate in the theory the treatment of rules whose match graphs are subgraphs of other rules (already in the tool).
- Integrate contract negation in the theory (already in the tool).
- Integrate negative application conditions (NACs) into our symbolic verification approach.
- Expand expressiveness of the property language by allowing properties that involve multiple applications of the same rule.

Bibliography

[1] Levi Lúcio, Bruno Barroca, Vasco Amaral, *A Technique for Automatic Validation of Model Transformations*, Proceedings of the MoDELS 2010 Conference, Springer, pp. 136-150.

[2] Bruno Barroca, Levi Lúcio, Vasco Amaral, Roberto Félix and Vasco Sousa, *DSLTrans: A Turing Incomplete Transformation Language*, Proceedings of the SLE 2010 conference, Springer 2010, pp. 296-305.

[3] Levi Lúcio, Bentley Oakes and Hans Vangheluwe, *A Technique for Symbolically Verifying Properties of Graph-Based Model Transformations*, Technical Report SOCS-TR-2014.1, McGill University, 2013.