

SyVOLT: Full Model Transformation Verification Using Contracts

Levi Lúcio, **Bentley James Oakes**

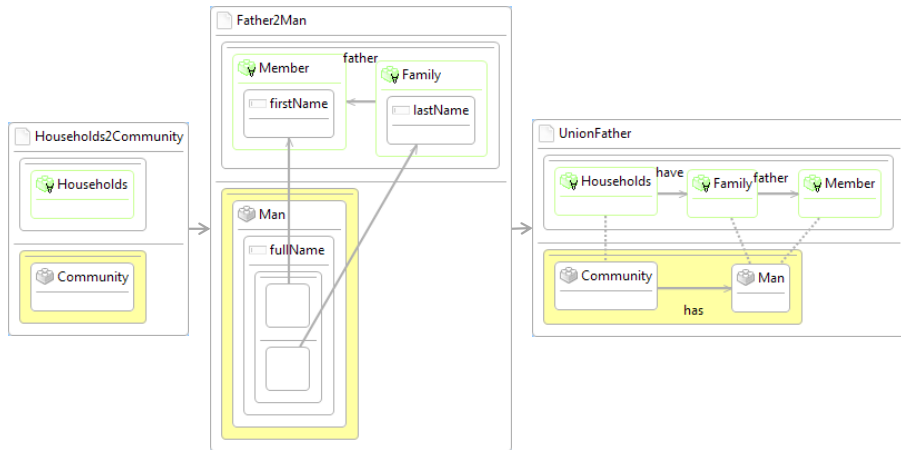
McGill University, Montreal
fortiss, Munich

July 28, 2016

- Model transformations are at the heart of model-driven engineering
- Want to verify correctness for transformation specifications
 - Verify visual pre- / post-condition contracts
 - Identify those combinations of rules where contracts hold or not
- Objective: Contract verification for all input models
 - Input independence

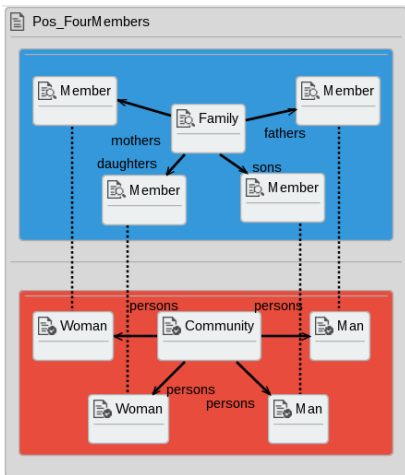
- Visual language for model transformations
- Graph-based, rule-based
- Rules are grouped in sequential layers
- Out-place so no rewriting performed
 - Suited for 'translation' transformations
- All its computations are terminating and confluent
 - Unbounded loops during execution are not allowed

Selim, Gehan et al. "Model transformations for migrating legacy deployment models in the automotive industry." *Software and Systems Modeling* 14, no. 1 (2013): 365-381.



- Rules arranged in layers
- Match graph on top of rules
- Apply graph on bottom
 - Produced when match graph is found

Pre- / Post- Visual Contracts

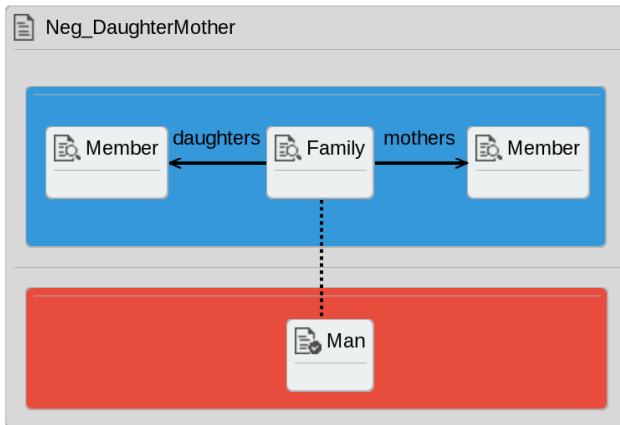


- If blue graph is found in input model, then red graph is found in output model
- Objective: Prove for all input models/transformation executions - input independence
- *A family with a father, mother, son, daughter should always produce two males and two females in the target community*

- All possible executions of the transformation are symbolically constructed
 - Built as sets of rules called path conditions
 - No rules execute, only rule 1 executes, rule 1 and rule 2 both execute
 - Rule dependencies/combinations resolved
- Final finite set of path conditions represents all possible transformation executions
- A contract holds for a transformation if it holds for all generated path conditions
- Otherwise, counter-example path conditions are produced
- Proving process completes within seconds

L. Lúcio, B. Oakes, and H. Vangheluwe. A technique for symbolically verifying properties of graph-based model transformations. Technical report, Technical Report SOCS-TR-2014.1, McGill U, 2014.

Contract Proving Example



- Statement: *A family with a mother and a daughter will always produce a community with a man*
- Fails on path condition:
'HFamComm_HMotherRule_HDaughterRule'

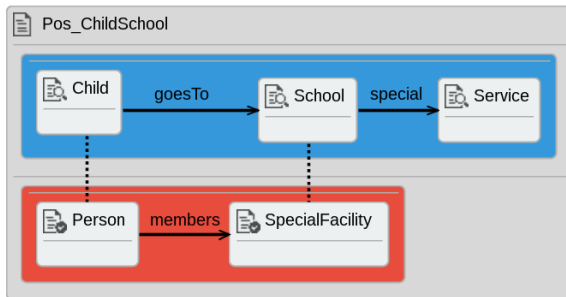
Performance

	ATL/ DSLTrans Rules	Path Conds. Generated	Time (s)	Contracts Proved	Time (s)	Memory (MB)
Families-to-Person	5 / 9	101	0.24	4	0.52	54
Extended Families-to-Person	10 / 19	366	3.89	10	7.35	59
GM-to-AUTOSAR (handbuilt)	5 / 9	13	0.18	9	0.15	58
GM-to-AUTOSAR (HOT)	5 / 9	10	0.26	9	0.15	60
UML-to-Kiltera	20 / 17	322	1.86	15	11.99	55

- Next few slides will discuss concepts expressible with our contract language.
 - Pattern contracts
 - Element attributes
 - Propositional logic and pivots
 - Syntactic invariants
 - Multiplicity invariants
- Prover reports rule reachability as well

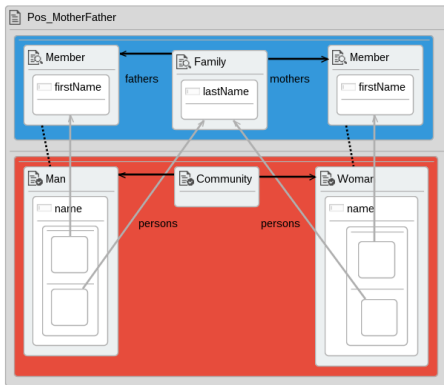
Selim, G.M.: Formal Verification of Graph-Based Model Transformations. Ph.D. thesis, Queen's University. 2015.

Pattern Contracts



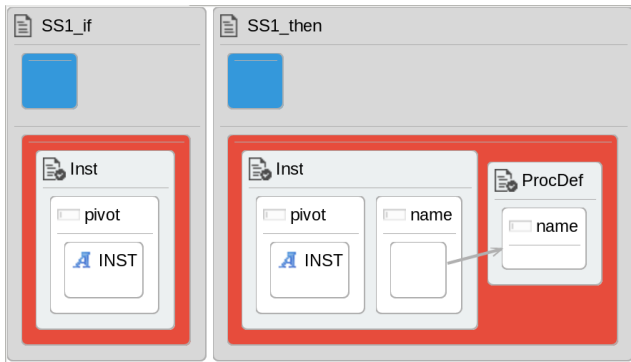
- Relates elements in input model to elements in output model
- *If a Child goesTo a School that has a special Service, then a SpecialFacility has the associated Person as a member*
- Intention is to allow verification of rule interaction
 - Three rules in example

Element Attributes



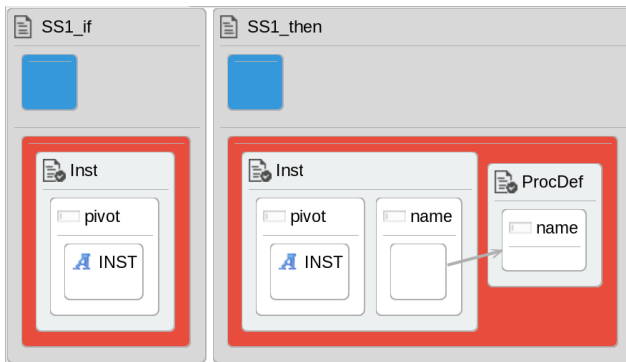
- Reasoning about (String) attributes of elements
- *Is the full name of the produced Person correctly created from the last name of the Family and the first name of the Member?*

Propositional Logic and Pivots



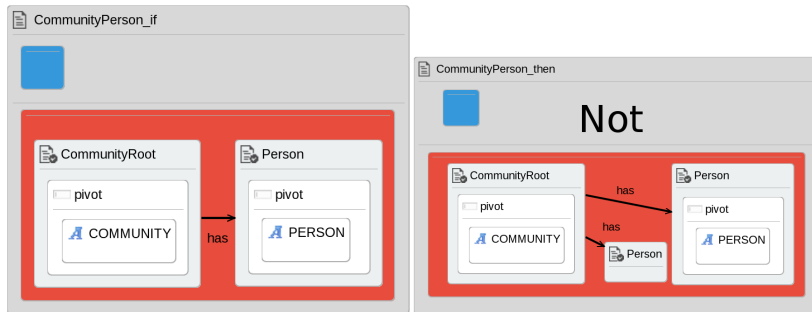
- Contracts can be combined with AND, OR, NOT, IF-THEN
- Pivots ensure that same element is bound in both contracts
- *If there is an Inst element, then that Inst element has the same name as a ProcDef element*

Syntactic Invariants



- Check if path condition has well-formed input or output syntax
- *If there is an Inst element, then that Inst element has the same name as a ProcDef element*

Multiplicity Invariants



- *Only one Person is in the Community in the output model*
- Abstraction of our approach loses multiplicity information
 - Multiple applications of a rule are not represented
- Contract only fails if *two Persons are always created in output*

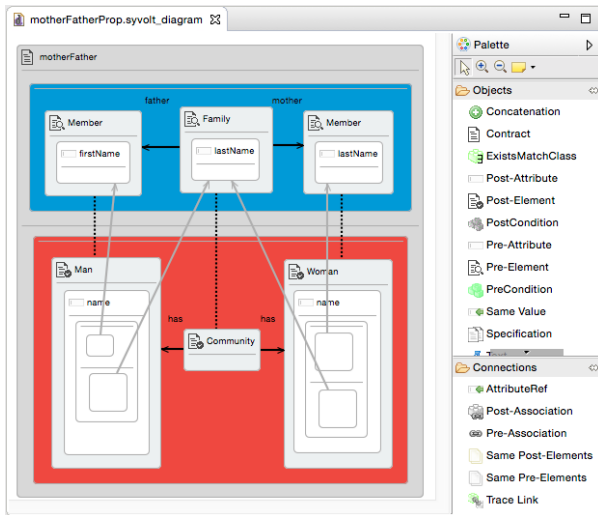
- DSLTrans transformation language only manipulates Strings
 - Could pack data and operations into Strings
- Limitations from symbolic execution technique
 - Loses multiplicity information
 - Cannot count elements created
 - Difficult to express *for each A element, there exists unique B element*

Contract Limitations

- No query language implemented
 - Cannot create sets of elements
 - Cannot match pattern with inheritance and report subtype names
 - No indirect links/navigation
- Difficult to reason about negative contracts
- Cannot validate instance data
 - 'Do all names start with G'
 - 'Is gender == male or age < 18'

- Fully Verifying Transformation Contracts for Declarative ATL
Bentley James Oakes, Javier Troya, Levi Lucio, and Manuel Wimmer
Proceedings of MODELS 2015
 - Extended to journal article: Full Contract Verification for ATL using Symbolic Execution, SoSyM (to appear)
- Finding and Fixing Bugs in Model Transformations with Formal Verification: An Experience Report
Gehan M. K. Selim, James R. Cordy, Juergen Dingel, Levi Lucio and Bentley James Oakes
Proceedings of AMT 2015

Eclipse Plugin



- Eclipse GMF plugin for building DSLTrans transformations and contracts.

- Verification of visual contracts on DSLTrans transformations
- Approach is complete for all transformation executions
- Can extend contract language expressiveness
- Eclipse plugin to build transformation and contracts
- Current work: Verification of mbeddr transformation
- Thank you for your time!

SyVOLT: Full Model Transformation Verification Using Contracts