Symbolic Verification of Translation Model Transformations

Levi Lucio, Bentley James Oakes, and Hans Vangheluwe

January 14, 2014

1. Overview
2. DSLTrans
3. Path Condition Construction
    - Building Path Constructions from Layers
    - Disambiguating Path Conditions
    - Generating All Path Conditions
4. Proving Properties
5. Implementation
6. Concluding Material

Outline

1. Overview
2. DSLTrans
3. Path Condition Construction
   - Building Path Constructions from Layers
   - Disambiguating Path Conditions
   - Generating All Path Conditions
4. Proving Properties
5. Implementation
6. Concluding Material

Overview

- Motivation is to formally prove properties on model transformations
- A method to construct set of path conditions for a DSLTrans transformation
- Set of path conditions represent all possible transformation executions
- Prove structural properties on path conditions implies properties proved on transformation
- Method is transformation-independent and graph-based

Contributions

- Algorithm to construct path conditions
- Algorithm to prove properties on these path conditions
- Validity and completeness proofs of these algorithms
- Performance and scalability results

Outline

1 Overview

2 DSLTrans

3 Path Condition Construction
- Building Path Constructions from Layers
- Disambiguating Path Conditions
- Generating All Path Conditions

4 Proving Properties

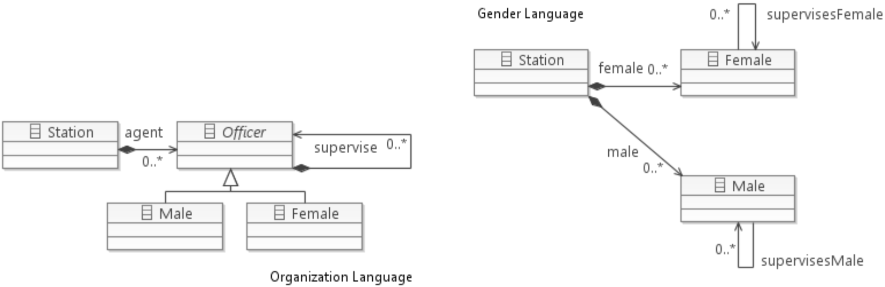
5 Implementation

6 Concluding Material

DSLTrans Overview

- Transformation language
- Turing-incomplete and outplace
  - Avoids unbounded recursion or non-determinism
  - Allows construction of provably-finite set of path conditions
- Transformation composed of rules in layers
  - Rules in first layer are repeatedly executed in deterministic random order until exhausted
  - Next layer of rules is then executed

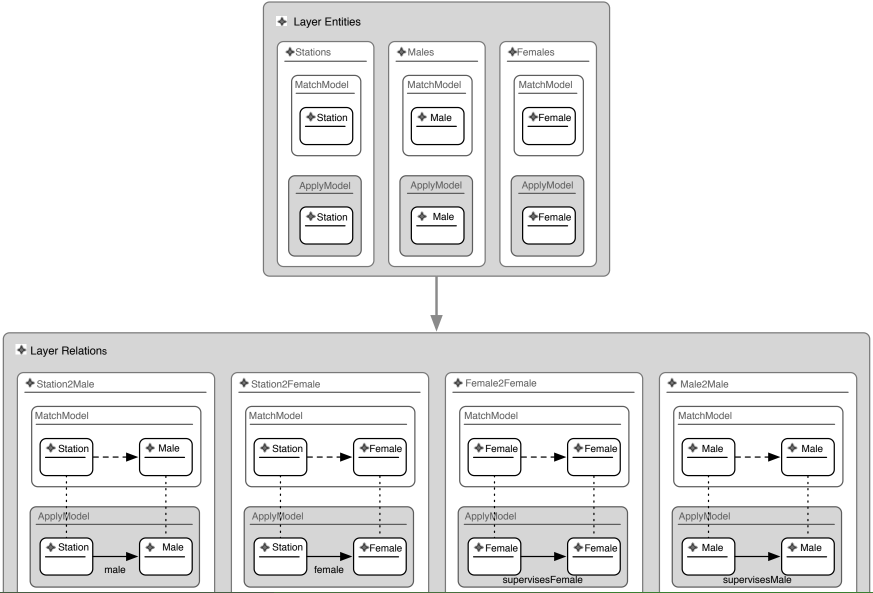
Metamodels for Running Example



(a) Chains of command

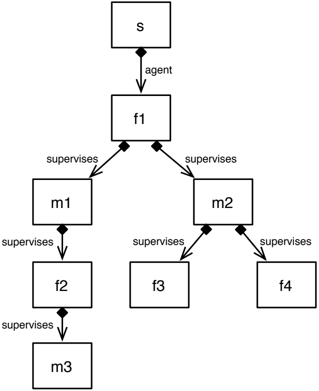
(b) Gender classification view

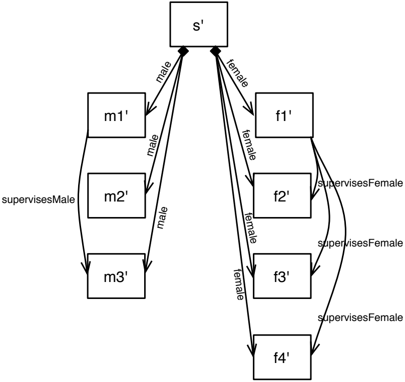
Example DSLTrans Transformation



Model Before/After Transformation

Objective is to flatten a chain of command given in the Organization language language into two independent sets of male and female officers represented in the Gender language





Outline

1. Overview
2. DSLTrans
3. Path Condition Construction
   - Building Path Constructions from Layers
   - Disambiguating Path Conditions
   - Generating All Path Conditions
4. Proving Properties
5. Implementation
6. Concluding Material

Path Conditions

- Similar to symbolic path in symbolic execution
- Formally represents concrete transformation executions
- Composed of a match graph and an apply graph
  - Path conditions look almost identical to DSLTrans rules

Rule Combinations

- Take powerset of rules in first layer
  - A, AB, ABC, BC
  - Abstract over number of times a rule executes

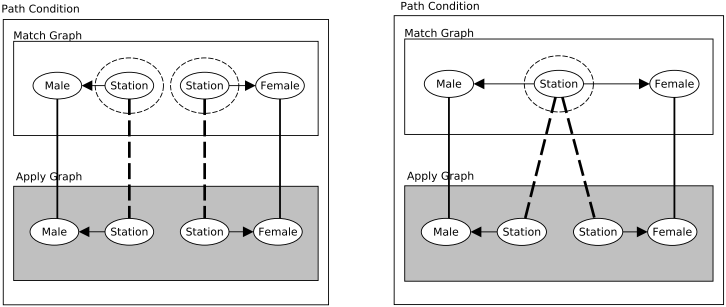
Also add traceability information to rules at this time

Path Conditions

- Take rule combination i.e. AC
- Union match graphs together
- Union apply graphs together

Disambiguating

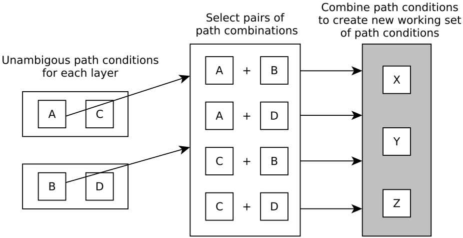
Match elements may refer to same element during transformation execution. Therefore, must disambiguate (recursively).



(a) Locating elements

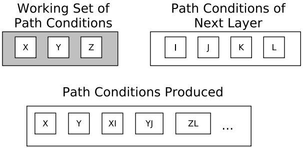
(b) Merging elements

Combination Overview



Unambigous path conditions for each layer
A
C
B
D
Select pairs of path combinations
A + B
A + D
C + B
C + D
Combine path conditions to create new working set of path conditions
X
Y
Z

(a) **Combining path conditions from two layers to produce working set of path conditions**



(b) Combining the working set with the next layer's path conditions

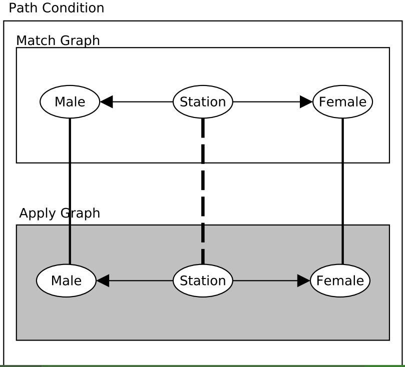
Combining Two Path Conditions

- Take path conditions A from layer 1, and B from layer 2
- Four different cases for combination
    1. No interaction between A and B
    2. A prevents B from holding
    3. Both A and B *may* hold
    4. Both A and B *will* hold

Dependencies must be resolved by algorithm

Path Condition Dependencies

- Dependencies represented by backward link
- Backward links are a construct in the DSLTrans rule language
    - Enforce that a particular element in a apply graph was created from the connected element in the match graph



Case 1

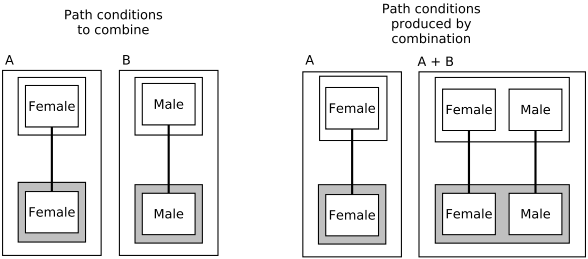

Path conditions to combine
A
B
Female
Female
Male
Male
Path conditions produced by combination
A
A + B
Female
Female
Female
Male
Female
Male

Figure : No dependencies between A and B

Case 2

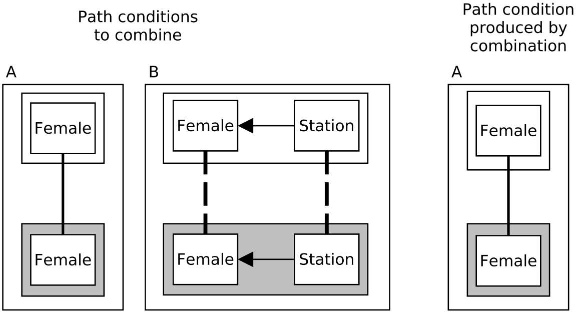


Figure : B's dependencies are not satisfied by A

Case 3

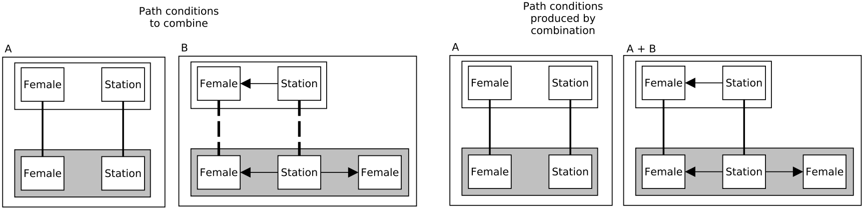


Figure : B's dependencies are partially satisfied by A

Case 4

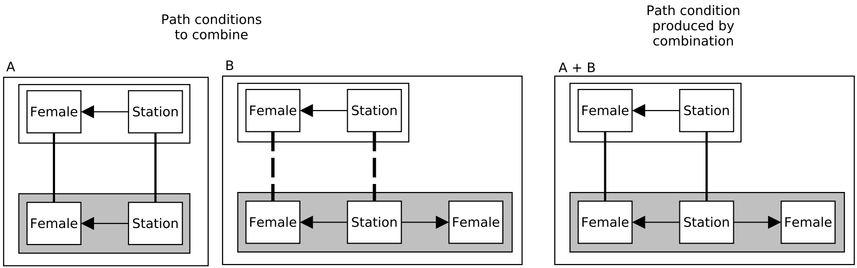
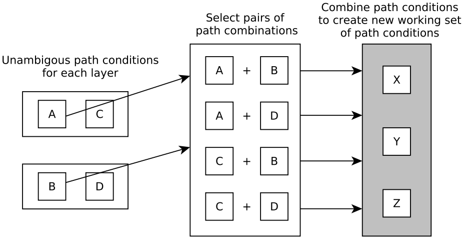
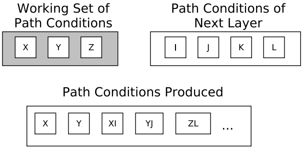


Figure : B's dependencies are fully satisfied by A

Repeated Combinations


Unambigous path conditions for each layer
A
C
B
D
Select pairs of path combinations
A + B
A + D
C + B
C + D
Combine path conditions to create new working set of path conditions
X
Y
Z

(a) Combining path conditions from two layers to produce working set of path conditions



(b) **Combining the working set with the next layer's path conditions**

Outline

1 Overview

2 DSLTrans

3 Path Condition Construction
   - Building Path Constructions from Layers
   - Disambiguating Path Conditions
   - Generating All Path Conditions

4 Proving Properties

5 Implementation

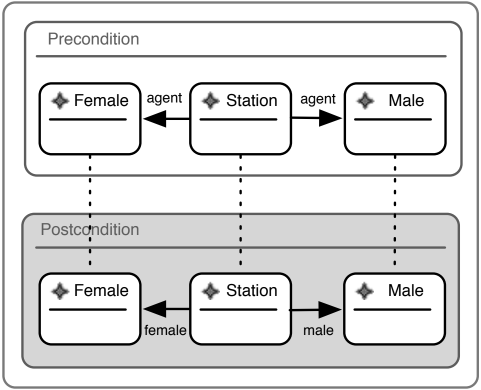
6 Concluding Material

Property Structure

- Very similar to DSLTrans rules and path conditions
- Composed of pre-condition and post-condition graphs

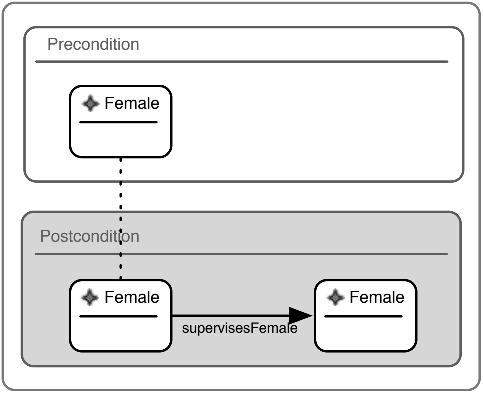
Property Examples

Property 1 – Expected to hold: A model which includes a police station that has both a male and female chief officers will be transformed into a model where the male chief officer will exist in the male set and the female chief officer will exist in the female set

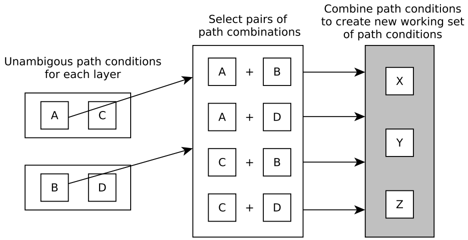


Property Examples

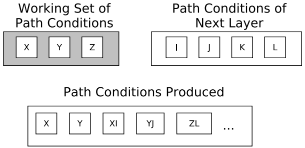
Property 2 – Not expected to hold: Any model which includes a female officer will be transformed into a model where that female officer will always supervise another female officer



Repeated Combinations



(a) Combining path conditions from two layers to produce working set of path conditions



(b) **Combining the working set with the next layer's path conditions**

Outline

1 Overview

2 DSLTrans

3 Path Condition Construction
- Building Path Constructions from Layers
- Disambiguating Path Conditions
- Generating All Path Conditions

4 Proving Properties

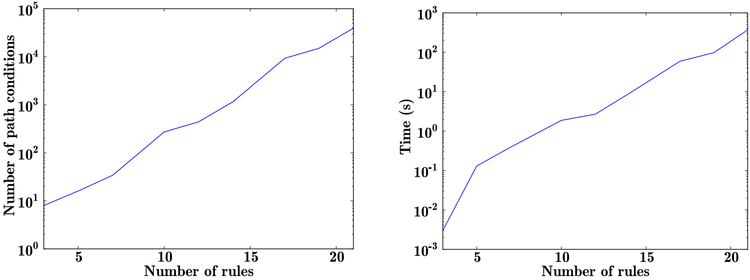
5 Implementation

6 Concluding Material

Implementation Overview

- Prototype built in Python and T-Core
  - T-Core is library for efficiently matching/rewriting graphs
- Naive complexity is calculated to be larger than $O(2^{2*r})$ where r is the number of rules in the transformation
  - Complexity made worse when disambiguation is required
- Optimizations such as memoisation/caching/lazy disambiguation performed

Metrics for PC Creation

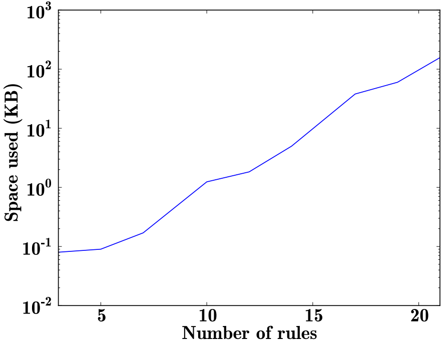


(a) Number of rules vs. path conds. created

(b) Number of rules vs. time taken

Rules	3	5	7	10	12	14	17	19	21
Path conds. created	8	16	34	272	442	1156	9248	15028	39304
Time taken(s)	0.01	0.13	0.39	1.87	2.68	9.00	59.08	97.52	369.19

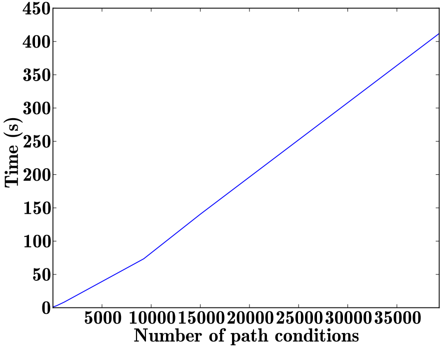
Metrics for PC Creation



(a) Number of rules vs. memory used

Rules	3	5	7	10	12	14	17	19	21
Memory used (KB)	0.08	0.096	0.17	1.24	1.83	4.98	38.01	60.10	156.79

Property-Proving Time



(a) Time required to prove the property that holds on all path conditions

Rules	3	5	7	10	12	14	17	19	21
Proof time	-	0.19	1.26	2.40	3.40	8.38	73.51	140.77	412.02

Property-Proving Time

For property that was not expected to hold for all path conditions, proof time is constant

- Algorithm detects counter-example quickly and does not need to process further path conditions

Outline

1 Overview

2 DSLTrans

3 Path Condition Construction
- Building Path Constructions from Layers
- Disambiguating Path Conditions
- Generating All Path Conditions

4 Proving Properties

5 Implementation

6 Concluding Material

Contributions

Contributions of this work include:

- Algorithm to construct a set of path conditions for a DSLTrans transformation
- Property-checking algorithm to prove model syntax relation properties over these path conditions, and therefore over all concrete transformation executions
- Validity and completeness proofs for the above algorithms (not shown in this presentation)
- Discussion of optimisation and scalability concerns
    - Successfully used in industrial case-study

Future Work

- Extend property language with attributes
- Negative DSLTrans constructs such as NACs
- Application to further case studies
- Develop tools to automatically create artifacts needed in algorithms
  - Requires development of HOTs
  - Bentley is currently working on this, involves model evolution concerns

Thank You

This work has been developed in the context of the NECSIS project, funded by Automotive Partnership Canada.