Testing Model Transformations in Model Driven Engineering

Benoit Baudry
INRIA – IRISA
(currently visiting CSU)
Outline

- What about model transformation testing?
- Triskell’s contributions
  - Coverage criteria
  - Model synthesis
- Related work
- Challenges
Model Transformation Testing: Motivation

requirement 1.1: "Registering a book" the "book" becomes "registered" after the "librarian" did "register" the "book", the "book" is "available" after the "librarian" did "register" the "book".
Model Transformation Testing: Motivation

- A transformation is meant to be reused
  - But also has to be adapted from one project to another
- A transformation is meant to hide the complexity
  - we would like to trust the transformation as we trust a compiler
Dynamic testing process

Test data

Program

Execution

Result

Specification

Oracle

Test data evaluation

Localization / Debugging

Oracle

Verdict

Stopping criterion

Problems

Test data generation

true

false

non vérifié

INRIA
Dynamic transformation testing process

- Test data / test model
- Model transformation
- Execution
- Result / output model
- Specification
- Oracle
- Verdict (true/false)
- Stopping criterion
- Localization / Debugging
- Diagnosis

Problems

Test model generation
Dynamic transformation testing process

- Specific issues
- Complex data
  - Models are manipulated as sets of objects
- Complex constraints
- Lack of specific tools
Model Transformation Testing

- Currently in Triskell
  - Coverage criteria
  - Automatic synthesis of test models (in coll. With Mc Gill)
  - Specific fault models
Model transformation

Source metamodel
structure + constraints

Transformation
language

Target metamodel
structure + constraints

Input model
pre
condition

Transformation

Output model
post
condition
Test data generation: criteria

- Several model transformation languages
  - Different features
  - Different paradigms
  - Different domains
- We did not want to choose
- We define black-box criteria
  - Independent of the model transformation language
Test data generation: criteria

- Define test criteria based on the input metamodel
  - Intuition: a set of models is adequate for testing if every class of the input metamodel is instantiated at least once and if the properties have relevant values
- A model for testing is called a test model
Test data generation: Example

What we expect from test models:
• Every class to be instantiated
• Properties to take several relevant values
• Combine properties in a meaningful way

Possibly infinite set of models => Need stopping criteria
Relevant values for properties

Adapt category partition testing to define ranges of relevant values for properties of the metamodel
Relevant values for properties

- Define partitions for each property in the input metamodel
- A partition defines a set of ranges on a domain
  - choose one value in each range for the property
- Example
  - partition for AbstractState::label=\{[0],[1],[2..MaxInt]\}
  - A set of test models will need to have, at least three states with three different values for label
Relevant values for properties

Transition::\textbf{event} \{"", {'evt1'}, {'.+'}
Transition::\textbf{#source} \{1\}
Transition::\textbf{#target} \{1\}
AbstractState::\textbf{label} \{0\}, \{1\}, \{2..MaxInt\}
AbstractState::\textbf{#container} \{0\}, \{1\}
AbstractState::\textbf{#incomingTransition} \{0\}, \{1\}, \{2..MaxInt\}
AbstractState::\textbf{#outgoingTransition} \{0\}, \{1\}, \{2..MaxInt\}
State::\textbf{isInitial} \{true\}, \{false\}
State::\textbf{isFinal} \{true\}, \{false\}
Composite::\textbf{#ownedState} \{0\}, \{1\}, \{2..MaxInt\}
We would like to constrain the models to have a State with one outgoing transition and more than one incoming transitions.
Relevant object structures

- ObjectFragment
  - ModelFragment
  - PropertyConstraint
    - Range
      - StringRange
        - values: BooleanValue
      - IntegerRange
        - isComposite: Boolean
        - isDerived: Boolean
        - default: String
        - isAbstract: Boolean
    - ValuePartition
      - MultiplicityPartition
        - owningClass
        - isAbstract: Boolean
        - isComposite: Boolean
        - isDerived: Boolean
        - default: String
    - IntegerInterval
      - lower: Integer
      - upper: Integer
    - BooleanValue
      - false: 0
      - true: 1
    - BooleanRange
      - values: BooleanValue
      - regexp: String [1..1]
Relevant object structures

- Criteria define structures that must be covered by test models
- These criteria combine partitions
- One criterion = set of constraints
  - one criterion declares the set of ranges that should be covered by a set of test models
- Example
  - Range coverage: Each range of each partition for all properties of the meta-model must be used in at least one model.
Test criteria

- Six test criteria (different combinations of ranges)
  - AllRanges
  - AllPartitions
  - + 4 class criteria
    - object fragments constrain each property of the object

- Do not consider constraints on the metamodel
  - Might generate insatisfiable fragments
Evaluating a set of models

- A prototype tool: MMCC
  - Framework for partitions and fragments definitions
- Computes a set of model fragments according to
  - Input metamodel
  - Test criterion
- Checks the coverage of a set of test models
  - With respect to the set of model fragments
Automatic synthesis of test models

- Automatic synthesis useful to
  - Limit the effort for test generation
  - Evaluate the test criteria

- Challenges:
  - Combine different sources of knowledge
  - Expressed in different formalisms
  - Complex constraints
Automatic synthesis of test models

Meta-model

Model Transformation Pre-condition

Test Model Knowledge
1. Test Model Objectives
2. Model Fragments

specifies

specifies

specifies

Test Models
The Solution(1):
Combining Knowledge to Common Constraint Language

- ECORE Model
- OCL Constraints
- OCL Constraint
- Requirements/Natural Language
- Objects and Property ranges

Expressed as

Pre-condition

Test Model Objectives

Transformed to

Common Constraint Language:
Alloy
Model synthesis

The *run* command:

run test_requirement1 for 1 ClassModel, 5 int, exactly 5 Class, exactly 20 Attribute, exactly 4 PrimitiveDataType, exactly 5 Association

1. Specify a scope
2. Specify an exact number of objects

Output: Alloy model instance that satisfies meta-model + pre-condition + test_requirement1 and has the specified size
CARTIER: OVERALL FRAMEWORK

Pre-condition
OCL

Input Meta-model MM_{in}
ECore Model
OCL

Test Model Knowledge
Model Fragments
Test Model Objectives

transformed to

transformed to

transformed to

transformed to

First-order Relational Logic Statements (in Alloy)
specifies a set of test models

Boolean CNF Formula
solved by
SAT Solver

a solution is a selected Test Model

Alloy XML transformed to XMI

Ecore XMI
input to
Model Transformation MT(IO)

Output Meta-model MM_{out}
ECore | OCL

Post-condition
OCL

Graphs

Graphs

cartier: overall framework

specifies a set of output models
Perspectives on model synthesis

- Strengthen the tool
  - Automate what can be

- Experiments

- Design experiments to test model transformations

- We want to numerically estimate via mutation analysis the efficiency of test models
Mutation Analysis
Mutation Analysis

- Evaluate the set of models
  - Producing a Mutation Score

<table>
<thead>
<tr>
<th>Mutant</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Killed</td>
<td>5/8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test set → P (insert the operators) → mutation operators → results of P → oracle

- Mutants killed
- Mutants alive

Yes: sufficient
No: suppress the equivalent mutants
improve the test set
Mutation Analysis

- Analysis based on fault models
- Faults are based on syntax of programming languages
  - Most common errors
  - For procedural languages, OO languages…
Mutation analysis for model transformation

- What errors occur in a model transformation?
- Implementation language independency
  - Too many different languages
- Lack data on common errors
Abstract transformation operations

- Navigation, filtering, creation, modification
  - Example of one transformation

A

B "persistent"
name : string
ID : int

(a)

B
name : string
ID : int

(b)

B "persistent"
name : string
ID : int

(c)

A

name : string
ID : int

(d)

"persistent"

(e)

ID

(f)

navigation
creation

INRIA
Mutation operators

- **Navigation**
  - Relation to the same class
  - Relation to another class
  - Relation sequence modification with deletion
  - Relation sequence modification with addition

- **Filtering**
  - Perturbation in the condition
  - Delete a predicate
  - Add a predicate

- **Creation**
  - Replace an object by a compatible one
  - Miss association creation
  - Add association creation
One specific operator example

**Navigation**
- Relation to the Same Class Change - RSCC
Mutation Analysis

- The proposed operator have been adapted to the Kermeta language
- Experiments:
  - To compare mutation operators
  - To evaluate the coverage criteria
  - To evaluate different knowledge for test generation
Perspectives in Triskell

- **Experiment!**
  - We have spent a lot of time defining ideas and building the tools

- **White-box techniques for specific languages**
  - Specific adequacy criteria
  - Fault localization

- **Oracle function definition**