Motivation

Meta-model variants

- Meta-models are used to define modelling languages
- Different variants of a modelling language depending on scenario, project, goal...
- Having a meta-model for each variant is challenging to construct, analyse and maintain
Motivation
Example: variants of Petri nets

Different realizations

PetriNet

Place

Transition

```
posTokens inv: self.itokens >= 0
```

```
Place

in

Transition

out
```

```
PetriNet

places * trans *
```

```
PetriNet

places * in * trans *
```

```
Token

tokens *
```

```
Place

1

Transition

1
```

```
PetriNet

1

PetriNet

1
```

tokens as attributes

tokens as objects
Motivation

Example: variants of Petri nets

Different realizations

tokens as attributes

tokens as objects

Different features

hierarchical nets

state-machine nets
Motivation

Example: variants of Petri nets

Different realizations

- Tokens as attributes
- Tokens as objects

8 possible meta-models

Different features

- Hierarchical nets
- State-machine nets
Motivation

Meta-model product lines (MMPLs)

Meta-model product line:
compact representation of all meta-model variants
Motivation

Meta-model product lines (MMPLs)

Meta-model product line:
compact representation of all meta-model variants

Feature Model

Valid feature configurations:
- Simple xor Object
- StateMachine is optional
- Hierarchical is optional
Motivation
Meta-model product lines (MMPLs)

Meta-model product line: compact representation of all meta-model variants

150-Meta-Model

- Presence conditions
- Cardinality modifiers: min, max
- Inheritance modifiers: add, del

Presence conditions
Cardinality modifiers
Inheritance modifiers
Motivation

Meta-model product lines (MMPLs)

Meta-model product line:
compact representation of all meta-model variants

Feature Configuration

<Object, Hierarchical>

Meta-Model Derivation

\[
\text{isHierarchical } \text{inv: } \begin{cases} 
\text{self.places} \to \text{size}() > 0 \text{ or } \\
\text{self.trans} \to \text{size}() > 0 \text{) implies } \\
\text{self.in} \to \text{size}() + \\
\text{self.out} \to \text{size}() = 0
\end{cases}
\]
Motivation

Correctness of MMPLs

How to ensure a MMPL is correct?

1. ensure each meta-model is syntactically correct
   e.g., the target class of each meta-model reference belongs to the meta-model

2. ensure desirable properties in meta-model instances
   e.g., instantiability

There are well-known techniques to analyse this for a single meta-model.

However, generating and analysing each meta-model in the MMPL separately is time-consuming...
We lift meta-model analysis techniques to the product line level:
- syntactic analysis of meta-models
- satisfiability checking of meta-model properties

in order to improve performance

Based on a declarative notion of MMPL
- considers OCL well-formedness constraints
- amenable to automated analysis

Tool support

Evaluation of effectiveness of lifted analyses
Ensuring Well-formedness of Meta-Model Product Lines
Ensuring MMPL well-formedness
Lifted analysis of well-formed structure

- Every field is owned by one class
  How: PC of field $\implies$ PC of its owner-class

- Every reference points to a class
  How: PC of reference $\implies$ PC of its target-class

- Cardinality and inheritance are uniquely determined
  How: $\text{PC of min}_i \land \text{PC of min}_j$ is unsat (similar for max, inheritance)

- There are no inheritance cycles
  How (roughly): given a cycle in the 150MM, the conjunction of the PC of each inheritance relation is unsat
Ensuring MMPL well-formedness
Lifted syntactic analysis of invariants

- If an invariant is present, the accessed elements are also present
  How: \( \text{PC of invariant} \implies \text{PC of accessed fields + owner classes} \)

  Example:
  \( \text{PC of self.itokens} \geq 0 \implies \text{PC of itokens and PC of Place} \)
  \( \text{Simple} \implies \text{Simple } \land \text{true} \)

- Operators are applied on fields with appropriate cardinality
  How: if a collection operator is applied on a field, the PC of invariant
  \( \land \text{PC of any max}=1 \) is unsat
Analysing Properties of Meta-Model Instances
Is the set of models accepted by a meta-model the one intended?

\[ m_1 \models P \]

In the simplest case, if \( P \) is empty, this method permits assessing whether a meta-model has instances.
Is the set of models accepted by each meta-model the one intended?

In the simplest case, if $P$ is empty, this method permits assessing whether some meta-model in the MMPL has instances.
Analysing instance properties
Classification of property types

- Property specification
  - structural
  - mixed

Solutions
Result
- model
- config
- one
- all

Search strategy

- Property satisfiability
- Search scope
  - exists
  - forall
  - notexists

Artefact Decision
- condition
  - exists
  - forall
  - notexists

Type
Analysing instance properties
Classification of property types

Property specification
- structural
- mixed

Search strategy

Property satisfiability
- \(\exists_m\)
- \(\forall_m\)
- \(\neg\exists_m\)

Search scope
- Config scope
- Feature exercising

- partial
- total
Analysing instance properties

Classification of property types

- Property specification
  - structural
  - mixed

- Search strategy
  - Property satisfiability
  - Config scope
  - Feature exercising

- Result
  - Artefact
  - Decision condition

- Type
  - exists_{MM}
  - forAll_{MM}
  - notExists_{MM}

- Solutions
  - one
  - all
  - model
  - config
Analysing instance properties

Property types

Some analyses of interest (8 more in the paper):

- MMPL instantiability: configuration that yields an instantiable MM
  Configuration: \(< one, config, total, exists_m >\)

- Global invariant: is a property satisfied by every model of every MM?
  Example: all Petri nets have at least one place
  Configuration: \(< forAll_{MM}, forAll_m, ... >\)
  Property: Place.all() → notNotEmpty()

- Safety property: is a property satisfied by no model?
  Example: no model has isolated transitions
  Configuration: \(< forAll_{MM}, notExists_m, ... >\)
  Property: Transition.all() → exists(in→isEmpty() and out→isEmpty())
We also consider mixed properties
Example: Are transitions with one input only possible on state machines?
Configuration: \(< \textit{mixed}, \ldots >\)
Property: Transition.all() → forall(in → size() = 1) implies StateMachine
Lifted analysis of meta-model instances

(1) Encoding of MMPL as a regular meta-model

Feature-explicit meta-model

- 150MM
- Feature model is class FMC, where features are booleans
- PCs and modifiers are invariants
- Property to check is invariant
Lifted analysis of meta-model instances

(2) Analysing feature-explicit meta-model

Look for an instance of the FEMM using a model finder

- MMPL instantiability, weak properties (MMs where some model satisfies P): finding a solution implies satisfaction

Safety properties (MMs where no model satisfies P): find all configs where some model satisfies P, and then return the rest

Global invariants (MMs where all models satisfy P): find all configs where some model satisfies \textbf{not} P, and then return the rest
Tooling
- Eclipse plugin: [http://miso.es/tools/merlin](http://miso.es/tools/merlin)
- Feature model specified with FeatureIDE
- 150MM specified as an Ecore meta-model with annotations
- Static analysis of OCL uses Eclipse OCL project and Sat4J
- USE Validator model finder

Merlin
Evaluation
## Efficiency of syntactic analysis

### Enumerative approach vs Lifted analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>#Feats</th>
<th>#MMs</th>
<th>#classes/#invs/#PCs/#modifs</th>
<th>Lifted time</th>
<th>Enum time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running example</td>
<td>6</td>
<td>8</td>
<td>4/2/5/2</td>
<td>0.039s</td>
<td>0.19s</td>
</tr>
<tr>
<td>Relational DDBB</td>
<td>10</td>
<td>24</td>
<td>7/0/17/0</td>
<td>0.094s</td>
<td>0.45s</td>
</tr>
<tr>
<td>Graphs [29]</td>
<td>16</td>
<td>256</td>
<td>5/6/14/3</td>
<td>0.103s</td>
<td>22.36s</td>
</tr>
<tr>
<td>Automata</td>
<td>20</td>
<td>2.016</td>
<td>6/5/18/0</td>
<td>0.135s</td>
<td>102.9s</td>
</tr>
<tr>
<td>Role modelling [27]</td>
<td>48</td>
<td>&gt;2.395.000</td>
<td>40/0/32/9</td>
<td>0.735s</td>
<td>&gt;1h</td>
</tr>
</tbody>
</table>

⇒ Lifted analysis was much faster
Evaluation

Efficiency of instance property analysis

- Enumerative approach vs Lifted analysis

- MMPL instantiability (finding an instantiable meta-model)
  - Enumerative approach:
    1. generate product meta-model
    2. check meta-model instantiability by model finding
    3. if the meta-model has instances, conclude
    4. else, go to 1

- 22 variants of the Automata MMPL, each with a different percentage of instantiable meta-models
Evaluation
Efficiency of instance property analysis

Lifted analysis is (up to 1,000x) faster if <85% instantiable MMs

In the rest of cases, lifted analysis is slightly slower (120 vs 100 ms)

Rationale: 1 more complex search (lift.) vs many simpler searches (enum.)

⇒ The fewer MMs satisfy a property, the faster lifted analysis is
Conclusions

and

Future Work
Summary

- MMPLS: Declarative specification of meta-model variants
- Lifting of existing meta-model analysis techniques to the PL level
  - syntactic correctness of meta-models
  - checking properties on meta-model instances
- Initial implementation and evaluation
Current and next steps

- Transformation product lines, coupled to MMPLs


- Extend definition of MMPL with type modifiers for references

- Expand analyses, e.g., to discover subsumption of MM variants

- Extend the evaluation for other kinds of properties
Analysing Meta-Model Product Lines

Esther Guerra, Juan de Lara
Universidad Autónoma de Madrid (Spain)

Marsha Chechik, Rick Salay
University of Toronto (Canada)

esther.guerra@uam.es
http://miso.es/tools/merlin

Questions?