Correctness of Constraint-Aware Model Transformations

Xiaoliang Wang, Yngve Lamo

Department of Computer Engineering, Bergen University College, Norway

27 October 2011
Nordic Workshop on Programming Theory, Västerås, Sweden

Outline

Introduction

Diagram Predicate Framework

Correctness of Model Transformation

Model-Driven Engineering (MDE)

- In model-driven engineering, models are
  - Primary artefacts
  - Used to specify, generate and maintain code
  - Manipulated by model transformations
- Advantage
  - Productivity is greatly improved
  - Consistence between models is assured

Model Transformation

- Model transformation is automatic
    - Platform Independent Model (PIM) → Platform Specific Model (PSM)
    - Model → executable code
    - Model refactoring
- Improves the software development productivity and quality

Model Transformation

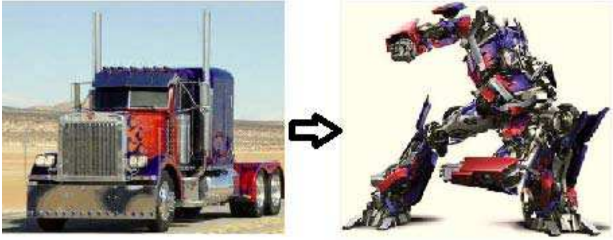
- Model transformations:
    - Source models → Target models
- Model transformation rules:
    - Source metamodel ↔ Target metamodel
- Given a source model and a set of model transformation rules, we use the following transformation process:
    - Find a suitable rule
    - Change on the source model according to the rule
    - Generate a new model which satisfies the target metamodel
    - Repeat the process until there is no suitable rule found

Correctness of Model Transformation

- Software programs need validation before deployment

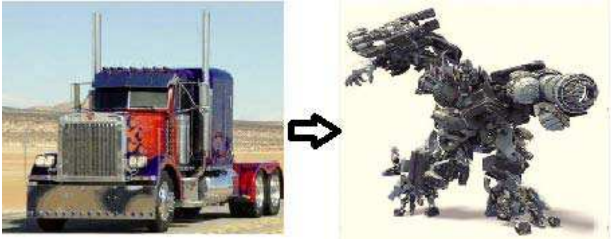
Correctness of Model Transformation

- Model transformations must also be reliable



Correctness of Model Transformation

- Model transformations must also be reliable



Correctness of Model Transformation

- Model transformation rules are designed manually
- In order to ensure reliability, it is necessary to check the correctness of the model transformation

Outline

Introduction

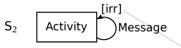
Diagram Predicate Framework

Correctness of Model Transformation

Diagram Predicate Framework (DPF)

- A fully diagrammatic specification framework for MDE
- Aims to be a diagrammatic formalism to define and reason about models and model transformations

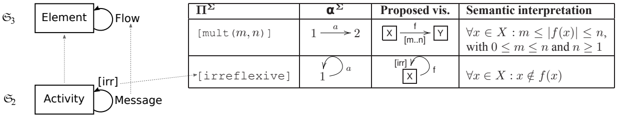
Diagram Predicate Framework (DPF)



Π^{Σ}	α^{Σ}	Proposed vis.	Semantic interpretation
`[mult(`m,n`)]`	$1 \xrightarrow{a} 2$	$\boxed{X} \xrightarrow[\text{[m..n]}]{f} \boxed{Y}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
`[irreflexive]`	$\begin{tikzcd} 1 \arrow[loop right, "a"] \end{tikzcd}$	$\begin{tikzcd} \boxed{X} \arrow[loop right, "f", "\text{[irr]}"'] \end{tikzcd}$	$\forall x \in X : x \notin f(x)$

- Models are formalized as diagrammatic specifications which consist of an underlying graph structure together with a set of atomic constraints

Diagram Predicate Framework (DPF)



Π^{Σ}	α^{Σ}	Proposed vis.	Semantic interpretation
`[mult(`m, n`)]`	$1 \xrightarrow{a} 2$	$\boxed{\text{X}} \xrightarrow[\text{[m..n]}]{\text{f}} \boxed{\text{Y}}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
`[irreflexive]`	$\begin{tikzcd} 1 \arrow[loop right, "a"] \end{tikzcd}$	$\begin{tikzcd} \boxed{\text{X}} \arrow[loop above, "\text{f}", "\text{[irr]}"'] \end{tikzcd}$	$\forall x \in X : x \notin f(x)$

A modelling language is formalized as a modelling formalism $(\Sigma_2 \triangleright S_2, S_2, \Sigma_3)$

- Specification S_2 represents the metamodel of the language
- Signature Σ_3 contains predicates which are used to add constraints to the metamodel S_2
- Typed signature $\Sigma_2 \triangleright S_2$ contains predicates which are used to add constraints to the specification S_1 that are specified by the modelling formalism

Diagram Predicate Framework (DPF)

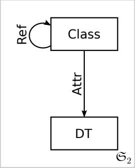
Constraint-Aware Model Transformation

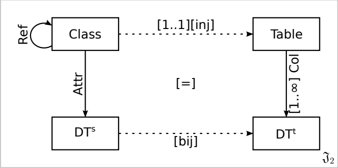
Joined Modelling Formalism

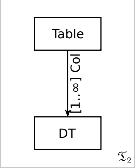
Σ_3

$\Gamma_3 := \Sigma_3 \uplus \Xi_3 \uplus \Theta_3$

Θ_3







$\Sigma_2 \rhd S_2$

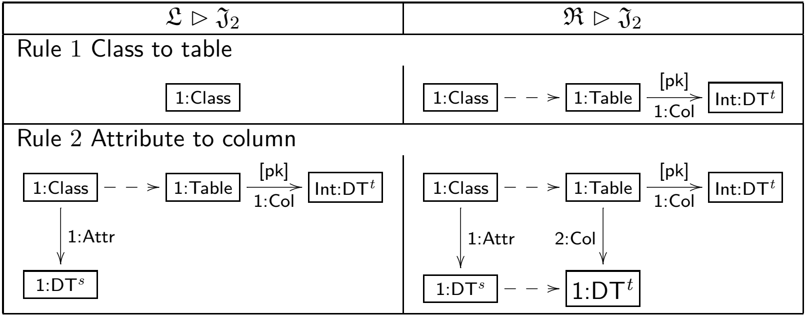
$\Gamma_2 := \Sigma_2 \rhd S_2 \uplus \Theta_2 \rhd T_2$

$\Theta_2 \rhd T_2$

Diagram Predicate Framework (DPF)

Constraint-Aware Model Transformation

Model transformation rules



Outline

Introduction

Diagram Predicate Framework

Correctness of Model Transformation

Correctness of Model Transformation

A match of a rule:

- It exists a graph homomorphism from the left hand side of the rule to the model

If a match of a rule is found in a model, we say that the rule is applicable to the model.

A model transformation is correct if:

- For any valid source model, a sequence of applicable rules which constructs a valid target model can be found

Correctness of Model Transformation

Rule application strategy

- When several rules are applicable at the same time
- When several matches of a rule are found in the model

Which method to use

For correctness of program:

- Testing: Never completely identify all the defects
- Theorem provers: Need a mathematical formalization of the program and involves human activities
- Model checkers: State explosion problem

Which method to use

Model transformations are automatic

- Run automatic tests of model transformations
- A sequence of applicable rules to constructs a desired target model
- Feedbacks assisting the designers to correct the rules

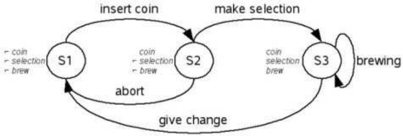
Which method to use

- For any determinstic program, each input only have one execution path
- For a model transformation, several different sequences of applicable rules may exist
- Model checker can check all the possible sequences

Model Checking

Model checking is an automatic way to verify that a model satisfies a given specification

- Model is represented as a Kripke structure



- Specification is formalized in temporal logic, CTL or LTL
- $E[(\neg selection)\,U\,(brew)]$

Verification Process

Given

- Joint modelling formalism (JMF), including the source metamodel (SMM) and the target metamodel (TMM)
- Transformation rules (MTRs)
- Source model (SM)

A kripke structure can be constructed through this procedure

Verification Process

- We define a initial state i representing SM
- For each state $s \in S$ and for every MTR $r : \mathfrak{L} \rhd S_2 \hookrightarrow \mathfrak{R} \rhd S_2$ we check $IsMatch(Model, \mathfrak{L} \rhd S_2)$. If it is true, the rule is applicable
- For each state $s \in S$ and for every applicable MTR $r : \mathfrak{L} \rhd S_2 \hookrightarrow \mathfrak{R} \rhd S_2$, we define a new state $r(s) \in S$ and a transition $t : s \to r(s)$

Verification Process

Correctness property:
In the future there is a state where no more rule is applicable and from this state a valid target model can be derived. In CTL, it is formalized as

$$EF!AnyRuleApplicable(Model, MTRs)\&\&IsInstanceof(getTargetModel(Model), TMM)$$

Future Work

- Find a suitable way to make rule application terminate
- Find way to implement the approach
- Find way to evaluate the approach
  - Efficency of checking
  - Number of states handled by the model checker

Thank you!

Questions?

For more information visit: `http://dpf.hib.no/`