Verification of Model Transformation using Alloy

Xiaoliang Wang

Bergen University College, Norway

May 30, 2014
Outline

1 Introduction and Motivation

2 Graph-based Model Transformation System

3 Encoding of graph-based Model Transformation

4 Verification

5 Conclusion
Model-Driven Engineering (MDE)
In model-driven engineering, models are

- Primary artifacts
- Used to specify, generate and maintain code
Model Specification and Conformance

Program
Model Specification and Conformance

Program

written in

Program Language
Model Specification and Conformance

Program written in Program Language

Model

Verification of Model Transformation using Alloy
Model Specification and Conformance

Program written in Program Language

Model specified and conforms to Metamodel

Verification of Model Transformation using Alloy
Model Specification and Conformance

**Model Specfication and Conformance**

- **R&A** represent by **Design** represent by **Implementation** represent by **Maintain**
- **M_{R&A}** conforms to **M_D** conforms to **M_I** conforms to **M_M**
- **MM_{R&A}** **MM_D** **MM_I** **MM_M**

Xiaoliang Wang Bergen University College, Norway

Verification of Model Transformation using Alloy
In model-driven engineering, models are

- Primary artifacts
- Used to specify, generate and maintain code
- Manipulated by model transformations
Model transformations are executed by applying model transformation rules.

- Model transformation rules
  - are defined on metamodel level
  - tell how to execute a transformation, i.e., generate a target model from a source model

Xiaoliang Wang  Bergen University College, Norway

Verification of Model Transformation using Alloy
Model Transformation Conformance Problem

\[ M_{R&A} \text{ conforms to } MM_{R&A} \]

\[ \Rightarrow \]

\[ M_{R&A} \text{ conforms to } \]

\[ T_1 \rightarrow M_D \]

\[ \]

\[ M_{R&A} \text{ conforms to } \]

\[ MM_{R&A} \]

\[ \]

\[ MM_D \]
Direct Condition

\[ MS \xrightarrow{r} MT \quad \Rightarrow \quad MT \text{ conforms to } MM \]

\[ MS \text{ conforms to } MM \quad \Rightarrow \quad \forall = \Rightarrow M_1 \xrightarrow{r_1} M_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} M_n \text{ conforms to } MM \]
Sequential Condition

\[ M_1 \xrightarrow{r_1} M_2 \]

conforms to

\[ MM \]

\[ M_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} M_n \]

conforms to

\[ MM \]

\[ \exists \Rightarrow \]

\[ M_1 \xrightarrow{r_1} M_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} M_n \]

conforms to

\[ MM \]
Bounded Verification Approach

- Model Transformation System
  - Metamodel
  - Model Transformation Rules
- Conditions
  - Direct Condition
  - Sequential Condition

encoded to

Alloy Specification

checked by

Alloy Analyser
Outline

1. Introduction and Motivation
2. Graph-based Model Transformation System
3. Encoding of graph-based Model Transformation
4. Verification
5. Conclusion
Metamodel

<table>
<thead>
<tr>
<th>UML+OCL diagram</th>
<th>Structure diagram</th>
<th>Constraints text diagram</th>
</tr>
</thead>
</table>

Xiaoliang Wang, Bergen University College, Norway

Verification of Model Transformation using Alloy
A fully diagrammatic specification framework for MDE

Aims to be a diagrammatic formalism to specify and reason about models and model transformation
Dijkstra’s algorithm ensures that a critical resource is exclusively accessed by one process at a time.

When a process needs to access a resource, the process first sends a request. After the process has acquired the turn, it is allowed to access the resource only if no other process competes with it.
Models are specified by a diagrammatic specification with a graph structure
Constraints are added on part of the graph structure
Diagram Predicate Framework

\[ \begin{align*}
P & \xrightarrow{[0..1]} T \\
\text{inj} & \xrightarrow{[0..1]} \text{surj} \\
\text{inj} & \xrightarrow{[1]} \\
R & \xrightarrow{[-1]} \\
\Pi & \xrightarrow{\Sigma 1} \\
\alpha & \xrightarrow{\Sigma 1} \\
\text{Proposed Vis.} & \\
\text{Semantic Interpretation} & \\
\end{align*} \]

<table>
<thead>
<tr>
<th>( \Pi^{\Sigma 1} )</th>
<th>( \alpha^{\Sigma 1} )</th>
<th>Proposed Vis.</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[m..n]</td>
<td>1 ( \xrightarrow{a} ) 2</td>
<td>( X \xrightarrow{f} Y ) ( [m..n] )</td>
<td>( \forall x \in X : m \leq</td>
</tr>
<tr>
<td>[inj]</td>
<td>1 ( \xrightarrow{a} ) 2</td>
<td>( X \xrightarrow{f} Y ) ( [\text{inj}] )</td>
<td>( \forall x_1, x_2 \in X : f(x_1) = f(x_2) \implies x_1 = x_2 )</td>
</tr>
</tbody>
</table>

Verification of Model Transformation using Alloy

Xiaoliang Wang
Bergen University College, Norway
Diagram Predicate Framework

**Diagram: Predicate Framework**

\[ \langle F_0 \rangle \langle F_1 \rangle \langle F_2 \rangle \langle ST \rangle \xrightarrow{[0..1]} T \]

\[ inj\] \[ surj\]

\[ R \xrightarrow{[0..1]} inj \]

\[ 1 \]

**Table: Proposed Vis. and Semantic Interpretation**

<table>
<thead>
<tr>
<th>( \Pi^{2} )</th>
<th>( \alpha^{2} )</th>
<th>Proposed Vis.</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[F0]</td>
<td>1</td>
<td>( X ) ( &lt;F_0&gt; )</td>
<td></td>
</tr>
<tr>
<td>[F1]</td>
<td>1</td>
<td>( X ) ( &lt;F_1&gt; )</td>
<td></td>
</tr>
<tr>
<td>[F2]</td>
<td>1</td>
<td>( X ) ( &lt;F_2&gt; )</td>
<td></td>
</tr>
<tr>
<td>[ST]</td>
<td>1</td>
<td>( X ) ( &lt;ST&gt; )</td>
<td></td>
</tr>
</tbody>
</table>

Xiaoliang Wang
Bergen University College, Norway

Verification of Model Transformation using Alloy
There is exactly one resource in a model, which is ensured by the multiplicity [1] on $R$. A similar constraint also applies to $T$.

There is exactly one process owning the turn, which is ensured by the multiplicity [0..1], [inj] and [surj] on $P \rightarrow T$. 
3 Each process may access the resource, represented by the arrow $P \rightarrow R$. A multiplicity $[0..1]$ is used to restrict only one such arrow for each process.

4 In the system, at most one process is accessing the resource, ensured by $[0..1]$ and $[\text{inj}]$ on the arrow $P \rightarrow R$. 

Xiaoliang Wang
Bergen University College, Norway

Verification of Model Transformation using Alloy
Each process should have exactly one of the flags (or states) \( F0, F1, F2 \) or \( ST \) (explained below). These flags are specified as annotations \([F0], [F1], [F2], [ST]\) marking the processes (instances of P).
Conformance Example

**Valid**

1. $P_1$\textcolor{blue}{<F2>} $T$
   
2. $R$\textcolor{blue}{<F1>} $2:P$

**Type Violation**

1. $P_1$\textcolor{blue}{<F2>} $T$
   
2. $R$\textcolor{blue}{<F1>} $2:P$

**Constraint Violation**

1. $P_1$\textcolor{blue}{<F2>} $T$
   
2. $R$\textcolor{blue}{<F1>} $2:P$

Xiaoliang Wang, Bergen University College, Norway

Verification of Model Transformation using Alloy
Model Transformation is executed according to model transformation rules.

Model transformation rules specify how to transform a model.

Graph-based model transformation can be present as graph production.

Each graph production $L \xleftarrow{l} K \xrightarrow{r} R$ consists of the left-hand side, the gluing graph and the right-hand side, where graph morphisms $l$ and $r$ are injective.
<table>
<thead>
<tr>
<th>Rule</th>
<th>L</th>
<th>K</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>:P&lt;\text{F0}&gt;</td>
<td>:P</td>
<td>:P&lt;\text{F1}&gt;</td>
</tr>
<tr>
<td>SetFlag2</td>
<td>:T&lt;1:P&lt;\text{F1}&gt;</td>
<td>:T&lt;1:P</td>
<td>:T&lt;1:P&lt;\text{F2}&gt;</td>
</tr>
<tr>
<td>CheckTurn</td>
<td>1:P&lt;\text{F1}&gt;</td>
<td>1:P</td>
<td>1:P&lt;\text{ST}&gt;</td>
</tr>
<tr>
<td>GetTurn</td>
<td>:T&lt;2:P&lt;\text{F0}&gt;</td>
<td>2:P</td>
<td>:T&lt;1:P&lt;\text{F2}&gt;</td>
</tr>
<tr>
<td>Compete</td>
<td>\text{R}&lt;1:P&lt;\text{F2}&gt;</td>
<td>1:P</td>
<td>1:P&lt;\text{F1}&gt;</td>
</tr>
<tr>
<td>Access</td>
<td>\text{R}&lt;1:P&lt;\text{F2}&gt;</td>
<td>\text{R}&lt;1:P&lt;\text{F2}&gt;</td>
<td>\text{R}&lt;1:P&lt;\text{F2}&gt;</td>
</tr>
<tr>
<td>Exit</td>
<td>\text{R}&lt;\text{P}&lt;\text{F2}&gt;</td>
<td>\text{R}&lt;\text{P}</td>
<td>\text{R}&lt;\text{P}&lt;\text{F0}&gt;</td>
</tr>
</tbody>
</table>
Double-Pushout Approach

L \xleftarrow{l} K \xrightarrow{r} R
\quad m \downarrow \quad \downarrow \quad n
S \xleftarrow{m} D \xrightarrow{n} T

\begin{align*}
\text{L} & \quad \text{T} & \quad \text{P} \\
\text{K} & \quad \text{T} & \quad \text{P} \\
\text{R} & \quad \text{T} & \quad \text{P} \\
\text{S} & \quad \text{T} & \quad \text{2P} \\
\text{D} & \quad \text{T} & \quad \text{2P} \\
\text{T} & \quad \text{T} & \quad \text{2P}
\end{align*}
Outline

1. Introduction and Motivation
2. Graph-based Model Transformation System
3. Encoding of graph-based Model Transformation
4. Verification
5. Conclusion
Bounded Verification Approach

- Model Transformation System
  - Metamodel
  - Model Transformation Rules
- Conditions
  - Direct Condition
  - Sequential Condition

encoded to

Alloy Specification

checked by

Alloy Analyser

Xiaoliang Wang Bergen University College, Norway

Verification of Model Transformation using Alloy
Alloy consists of

- the Alloy specification language
  - is a declarative language based on relational logic
  - suited to describe complex model structures and constraints
- the Alloy analyser analyses specifications.
A model structure is defined as *signatures*

Each *signature* defines a typed element in the structure, representing a set of instances of this type

Relations among the typed elements are defined by the fields of the signatures

Constraints on the structure can be defined as *facts* while predicates as *pred*
Alloy Analyzer

- find valid instances well-typed by the structure and satisfying its constraints by executing the `run` command
- verify some properties by calling the `check` command to find counterexamples
- NB! Alloy performs a bounded check, i.e., for each signature, a user-defined scope bounds the number of its instances.
## Model Transformations Encoding

<table>
<thead>
<tr>
<th>Element</th>
<th>Encoded As</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>sig Graph</td>
</tr>
<tr>
<td>General</td>
<td></td>
</tr>
<tr>
<td>Direct Model Transformation</td>
<td>sig Trans</td>
</tr>
<tr>
<td>Model Transformations</td>
<td>sig Path</td>
</tr>
<tr>
<td>Specific</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>sig Node</td>
</tr>
<tr>
<td>Edge</td>
<td>sig Edge</td>
</tr>
<tr>
<td>rule application</td>
<td>pred rule</td>
</tr>
<tr>
<td>constraints</td>
<td>fact</td>
</tr>
</tbody>
</table>
General Part

Model encoding

Models are encoded as the signature \textit{Graph}

\begin{verbatim}
1 sig Graph{
  2  nodes: set Node_i+...+Node_m,
  3  edges: set Edge_j+...+Edge_n}

1 fact{all g:Graph|all n:g.nodes|all e:Edge_j+...+Edge_n|(e.src=n
  or e.trg=n) implies e in g.edges}
\end{verbatim}
Direct model transformations are encoded as the signature \textit{Trans}

\begin{verbatim}
1 sig Trans{
2   rule: one Rule,
3   source: one Graph,
4   target: one Graph,
5   dnodes, anodes: set Node_{i+...+Node_{m}},
6   dedges, aedges: set Edge_{j+...+Edge_{n}}}
\end{verbatim}

Xiaoliang Wang Bergen University College, Norway

Verification of Model Transformation using Alloy
General Part
Valid Transformations

pred valid_trans [t:Trans] {
all edge : Edge | (edge.src in t.dnodes or edge.trg in t.dnodes) implies edge in t.dedges
all edge : Edge | (edge.src in t.anodes or edge.trg in t.anodes) implies edge in t.aedges

t.dnodes in t.source.nodes and t.dedges in t.source.edges

t.anodes in t.target.nodes and t.aedges in t.target.edges

t.source.nodes-t.dnodes=t.target.nodes-t.anodes

t.source.edges-t.dedges=t.target.edges-t.aedges

rule1[t] or ... or rule_n[t]

fact {all t:Trans | valid_trans [t]}
A sequence of direct model transformations is encoded as path signature of length n.

```alloy
1 sig Path3{trans1, trans2, trans3: one Trans}
3 fact{Path3.trans1+Path3.trans2+Path3.trans3=Trans}
```
Assuming a metamodel consisting of $m$ nodes and $n$ edges. Nodes and edges are discriminable from name. Each node(edge) named $i(j)$ is encoded as a $Node_i(Edge_j)$ signature.

```
1 sig Node_i{1≤i≤m}
2 sig Edge_j{1≤j≤n}
3 src:one Node_s,1≤s≤m
4 trg:one Node_t,1≤t≤m
```
Specific Part
Metamodel constraints encoding

Semantics of DPF predicates are specified in Java or OCL. Those can be encoded to FOL, expressed in Alloy.

```
pred source_valid[t:Trans] {
    // multiplicity on T min:1; max:1
    let count=#NT&t.source.nodes | count >= 1 and count <= 1
    // multiplicity on R min:1; max:1
    let count=#NR&t.source.nodes | count >= 1 and count <= 1
    // injective on PT: P -> T
    all n:(NP&t.source.nodes) | no e1, e2:(APT&t.source.arrows) | (e1 != e2 and e1.src = n and e2.src = n and e1.trg = e2.trg)
    // injective on PR: P -> R
    all n:(NP&t.source.nodes) | no e1, e2:(APR&t.source.arrows) | (e1 != e2 and e1.src = n and e2.src = n and e1.trg = e2.trg)
    ...
}
```
According to DPO, in every direct model transformation \( t : S \xrightarrow{p} T \) applying rule \( p = L \xleftarrow{l} K \xrightarrow{r} R \),

- the source(target) model has one match of the left(right) part of the rule \( p \). The matched parts are deleted or added according to the rule.

```
1 one m:L→S|(all n:m(l(K_N)))|n in t.source.nodes-t.dnodes) and
2 (all e:m(l(K_E)))|e in t.source.edges-t.dedges) and
3 (all n:m(L_N)
m(l(K_N)))|n in t.dnodes) and
4 (all e:m(L_E)
m(l(K_E)))|e in t.dedges)
5 one n:R→T|(all n:n(r(K_N)))|n in t.target.nodes-t.anodes) and
6 (all e:n(r(K_E)))|e in t.target.edges-t.aedges) and
7 (all n:n(R_N)
\ n(r(K_N)))|n in t.anodes) and
8 (all e:n(R_E)
\ n(r(K_E)))|e in t.aedges)
```
Direct Model Transformation Encoding

Exactly one rule is applied

No elements other than the matched part are deleted or added in the transformation.

1. If more than one elements \((e_1, e_2, \ldots, e_m)\) of the same type \(t\) are deleted(added)

\[
\begin{aligned}
1 & \text{ all } e : t \mid e \text{ in } t.\text{dnodes implies } (\text{or}_{i=1}^{i=m}( \text{one } m : \text{replace}(L, e, e_i) \rightarrow S \mid \text{all } n : m(1(K_N)) \mid n \text{ in } t.\text{source.noset}-t.\text{dnodes}) \text{ and }) \\
2 & \text{ (all } e : m(1(K_E)) \mid e \text{ in } t.\text{source.edges}-t.\text{dedges}) \text{ and }) \\
3 & \text{ (all } n : m(\text{replace}(L_N, e, e_i)) \setminus m(1(K_N)) \mid n \text{ in } t.\text{dnodes}) \text{ and }) \\
4 & \text{ (all } e : m(\text{replace}(L_E, e, e_i)) \setminus m(1(K_E)) \mid e \text{ in } t.\text{dedges}))
\end{aligned}
\]
No elements other than the matched part are deleted or added in the transformation.

2 If one or no element of the type \( t \) is deleted (added)

1 \#\{\text{trans.dnodes}&t\} = 1  
   \text{no trans.dnodes}&t

2 \#\{\text{trans.dedges}&t\} = 1  
   \text{no trans.dedges}&t

3 \#\{\text{trans.anodes}&t\} = 1  
   \text{no trans.anodes}&t

4 \#\{\text{trans.aedges}&t\} = 1  
   \text{no trans.aedges}&t
Direct Model Transformation Encoding

- No elements other than the matched part are deleted or added in the transformation.

3. For each kept node $n$, if there are both deleted edges $(d_1, d_2, \ldots, d_m)$ and added edges $(a_1, a_2, \ldots, a_n)$ related to it, its context should be changed as the rule specified. $st$ is $src$ or $trg$ field of the node depending on how those edges are related to the node.

\[
\text{one } d_1, d_2, \ldots, d_m, a_1, a_2, \ldots, a_n \mid d_1.st=d_2.st=\ldots=d_m.st=a_1.st=a_2.st, \ldots=a_n.st
\]
Outline

1. Introduction and Motivation
2. Graph-based Model Transformation System
3. Encoding of graph-based Model Transformation
4. Verification
5. Conclusion

Xiaoliang Wang Bergen University College, Norway

Verification of Model Transformation using Alloy
Bounded Verification Approach

- Model Transformation System
  - Metamodel
  - Model Transformation Rules
- Conditions
  - Direct Condition
  - Sequential Condition

encoded to

Alloy Specification

checked by

Alloy Analyser

Xiaoliang Wang
Bergen University College, Norway

Verification of Model Transformation using Alloy
Verification

- Verify systems by finding counterexamples
- Verify constraint one by one
Direct Condition

\[ M_1 \xrightarrow{r_1} M_2 \]

conforms to

\[ \forall S, T : S \triangleright SM \land (S \rightarrow T) \Rightarrow T \triangleright TM \] (1)
Verification Result

Commands

1. \texttt{check\{all trans:Trans|all n:(NP&trans.target.nodes)| no e1, e2:(APR&trans.target.arrows)| (e1!=e2 and e1.src=n and e2.src=n and e1.trg=e2.trg)}\} for 25 but exactly 1 Trans, exactly 2 Graph, exactly 1 Rule

2. \texttt{check\{all trans:Trans|all n:(NP&trans.target.nodes)| lone e:(APT&trans.target.arrows)|e.src=n\} for 25 but exactly 1 Trans, exactly 2 Graph, exactly 1 Rule

3. \texttt{check\{all trans:Trans|all n:(NF2&trans.target.nodes)|one e:AF2R&trans.target.arrows|e.src=n\} for 4 but exactly 1 NR, exactly 1 NT, exactly 1 ATR, exactly 1 Trans, exactly 2 Graph, exactly 1 Rule
One Counterexample

It shows that 3 constraints are violated. One example is given as following:

Xiaoliang Wang Bergen University College, Norway
Verification of Model Transformation using Alloy
Fix the problem

- Add more constraints on the metamodel
  1. If a process is accessing the resource, it is at state $<F2>$
  2. If a process is at state $<ST>$, it doesn’t own turn

- Add NACS on certain rules

Revised Rule Access

<table>
<thead>
<tr>
<th>Access</th>
<th>1: $P^{&lt;F2&gt;}$</th>
<th>1: $P^{&lt;F2&gt;}$</th>
<th>1: $P^{&lt;F2&gt;}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$R^{&lt;F2&gt;}$</td>
<td>$R^{&lt;F2&gt;}$</td>
<td>$R^{&lt;F2&gt;}$</td>
</tr>
</tbody>
</table>
Sequential Condition

\[
\begin{align*}
M_1 \xrightarrow{r_1} M_2 & \quad \text{conforms to} \quad MM_S \\
M_2 \quad & \quad \Rightarrow \quad \text{conforms to} \\
M_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} M_n & \quad \text{conforms to} \\
\end{align*}
\]

\[
\forall S, T_0 : (S \triangleright M \land S \rightarrow T_0 \land \neg (T_0 \triangleright M)) \Rightarrow
\]

\[
\exists T_1, \ldots, T_n : \bigwedge_{i=1}^{n-1} \neg(T_i \triangleright M) \land T_n \triangleright M \land \bigwedge_{i=0}^{n-1} T_i \rightarrow T_{i+1}
\]
- Verify systems by finding counterexamples
- Verify constraint which violates the direct condition
- Verify stepwise, 2, 3, \ldots, n
Object Oriented systems into entity-relationship models

- each class should be transformed into a table
- each table should have a primary key

Path of length 2

1 \text{sig} \hspace{0.5em} \text{Path}\{\text{trans}_0, \text{trans}_1 : \text{Trans}\}\{\text{trans}_0.\text{target} = \text{trans}_1.\text{source}\}

2 \text{fact}\{\text{all} \hspace{0.5em} \text{table}: \text{Table} & \text{trans}_0.\text{source}.\text{nodes} | \text{some} \hspace{0.5em} \text{pk}: \text{PK} & \text{trans}_0.\text{source}.\text{edges} | \text{pk}.\text{src} = \text{table}\}

3 \text{fact}\{\text{some} \hspace{0.5em} \text{table}: \text{Table} & \text{trans}_0.\text{target}.\text{nodes} | (\text{no} \hspace{0.5em} \text{pk}: \text{PK} & \text{trans}_0.\text{target}.\text{edges} | \text{pk}.\text{src} = \text{table}) \text{ and } (\text{some} \hspace{0.5em} \text{pk}_1: \text{PK} & \text{trans}_1.\text{aedges} | \text{pk}_1.\text{src} = \text{table})\}
Liveness properties

- Liveness property specify something good will eventually happen

**Example**

If a process sends request to access the resource(pre), it will eventually accesses the resource(live).
Given a liveness property $p$, if for each possible sequence of transformations $M_0 \rightarrow M_1 \cdots \rightarrow M_k$, some model $M_i (i \in \{0..k\})$ satisfies $p$, the system satisfies the liveness property $p$.

A loop sequence is a sequence of transformations $M_0 \rightarrow M_1 \cdots \rightarrow M_k$ where $M_k$ equals to one of the preceding models $M_i (i \in \{0..k\})$ and no model on the sequence satisfies the property.
Loop sequence is found on a path of length 3

1. run\{some path:Path3|some n:P|pre[path.t1.source, n] and (not live[path.t1.target, n]) and (not live[pat4h.t2.target, n]) and (not live[path.t3.target, n])\}

2. for 20 but exactly 1 Path3 exactly 3 Trans, exactly 3 Graph, exactly 1 Rule

3. pred pre[graph:Graph, n:P]{some e:(APA&graph.arrows)|e.src=n and e.trg=F1}

4. pred live[graph:Graph, n:P]{(some e:(APA&graph.arrows)|e.src=n and e.trg=F2) and (some e:(APR&graph.arrows)|e.src=n)}
The approach can be applied to model transformation systems which transforms between different metamodels.

The approach is bounded in that it verifies in a user-defined scope.

The approach is bounded and restricted by scalability problem. We use annotation modelling approach to solve the problem. The result shows the scope is leveled up from 3 to 25. We should find other mechanism to fix the problem, maybe other solver like SMT.
Thank you!
Questions?

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