Verification of Model Transformation using Alloy

Xiaoliang Wang

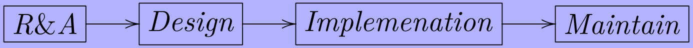
Bergen University College, Norway

May 30, 2014

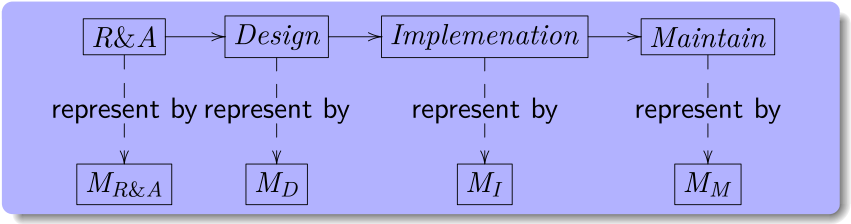
Outline

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

Model-Driven Engineering (MDE)



Model-Driven Engineering (MDE)

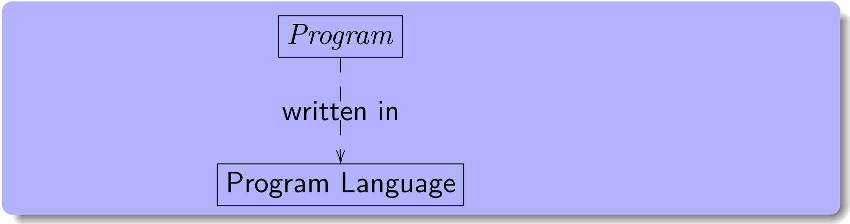


- In model-driven engineering, models are
  - Primary artifacts
  - Used to specify, generate and maintain code

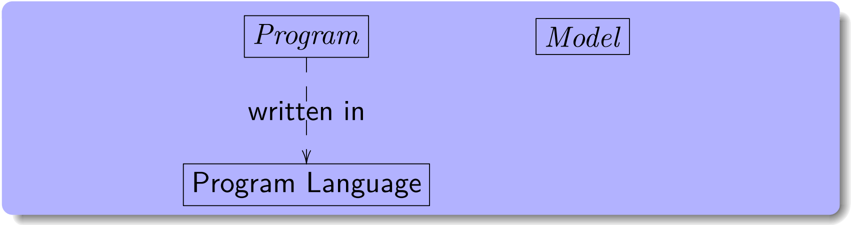
Model Specification and Conformance

Program

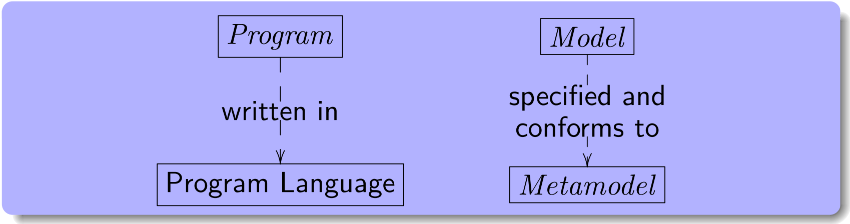
Model Specification and Conformance



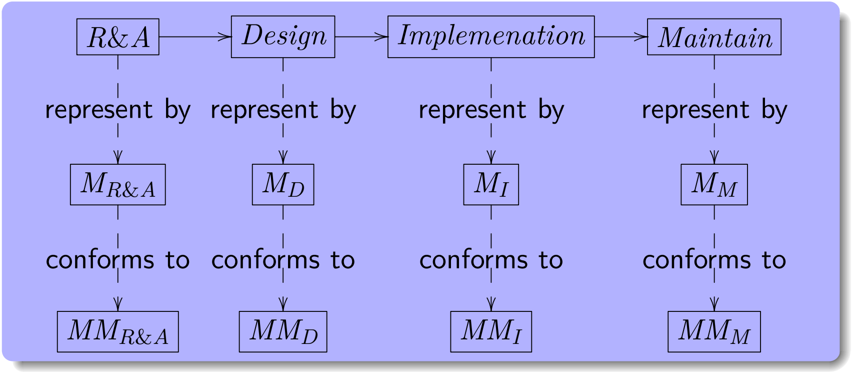
Model Specification and Conformance



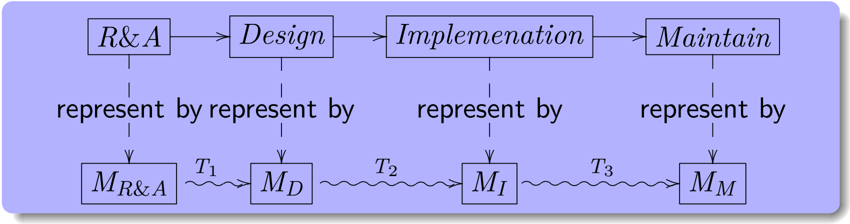
Model Specification and Conformance



Model Specification and Conformance

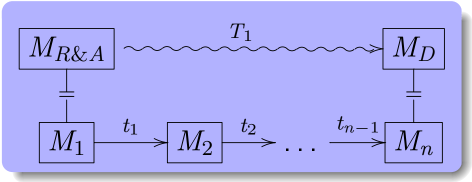


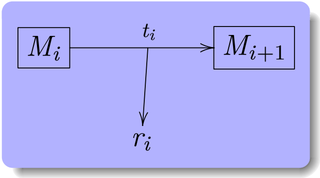
Model Transformation



- In model-driven engineering, models are
  - Primary artifacts
  - Used to specify, generate and maintain code
  - Manipulated by model transformations

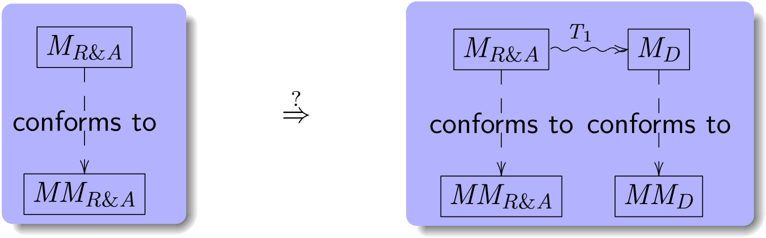
Model Transformation



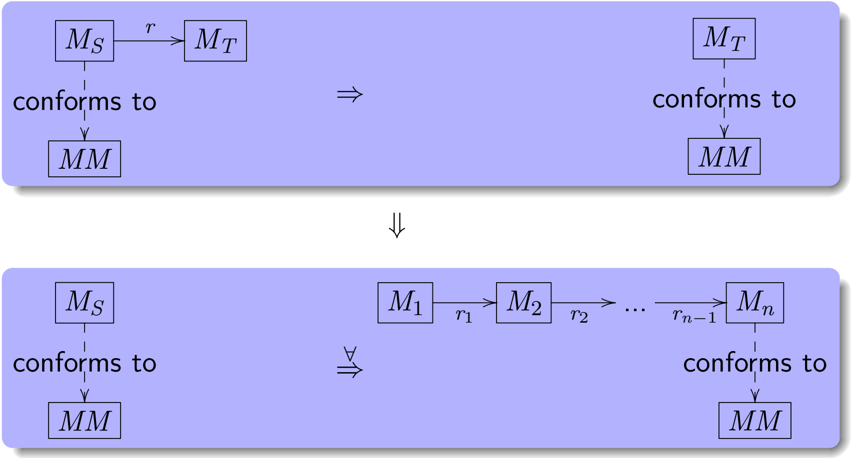


- Model transformations are executed by applying model transformation rules.
- Model transformation rules
  - are defined on metamodel level
  - tell how to execute a transformation, i.e., generate a target model from a source model

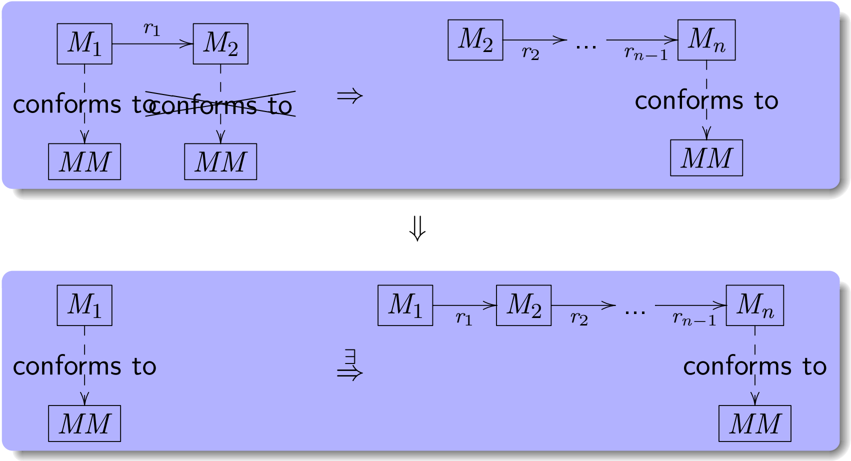
Model Transformation Conformance Problem



Direct Condition



Sequential Condition



Bounded Verification Approach

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

encoded to ⟹ **Alloy Specification**

⇓ *checked by*

Alloy Analyser

Outline

Metamodel

	Structure	Constraints
UML+OCL	diagram	text
DPF	diagram	diagram

Diagram Predicate Framework

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

Diagram Predicate Framework

- 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.

Diagram Predicate Framework

Models are specified by a diagrammatic specification with a graph structure

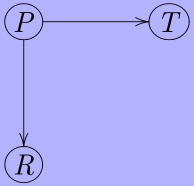


Diagram Predicate Framework

Constraints are added on part of the graph structure

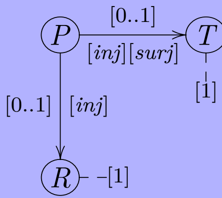
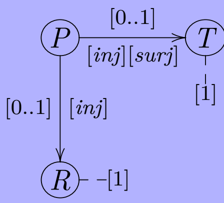
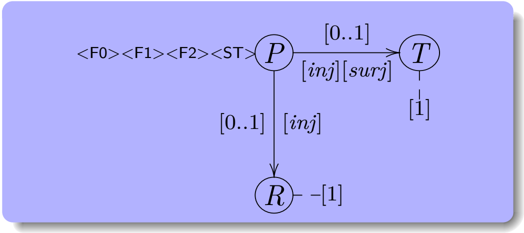


Diagram Predicate Framework



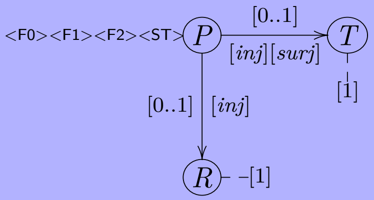
Π^{Σ_1}	α^{Σ_1}	Proposed Vis.	Semantic Interpretation
[m..n]	$1 \xrightarrow{a} 2$	$X \xrightarrow[f]{[m..n]} Y$	$\forall\ x \in X : m \leq \lvert f(x)\rvert \leq n$, *with* $0 \leq m \leq n$ *and* $1 \leq n$
[inj]	$1 \xrightarrow{a} 2$	$X \xrightarrow[f]{[inj]} Y$	$\forall x_1, x_2 \in X : f(x_1) = f(x_2) \implies x_1 = x_2$

Diagram Predicate Framework



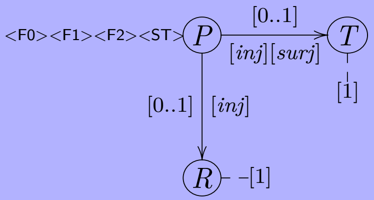
Π^{Σ_2}	α^{Σ_2}	Proposed Vis.	Semantic Interpretation
[F0]	1	X <F0>	
[F1]	1	X <F1>	
[F2]	1	X <F2>	
[ST]	1	X <ST>	

Constraints



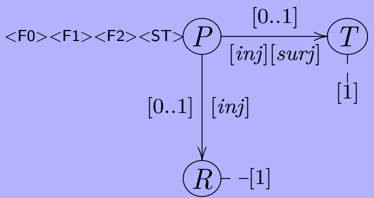
1. There is exactly one resource in a model, which is ensured by the multiplicity $[1]$ on R. A similar constraint also applies to T
2. There is exactly one process owning the turn, which is ensured by the multiplicity $[0..1]$, $[inj]$ and $[surj]$ on $P \rightarrow T$

Constraints



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 $[inj]$ on the arrow $P \rightarrow R$

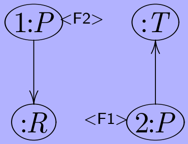
Constraints



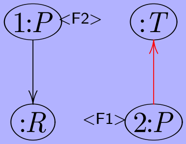
5. Each process should have exactly one of the flags (or states) $F0$, $F1$,$F2$ or ST (explained below). These flags are specified as annotations $[F0]$, $[F1]$, $[F2]$, $[ST]$ marking the processes (instances of P)

Conformance Example

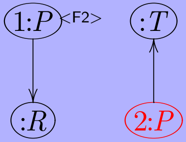
Valid



Type Violation



Constraint Violation



Model Transformation

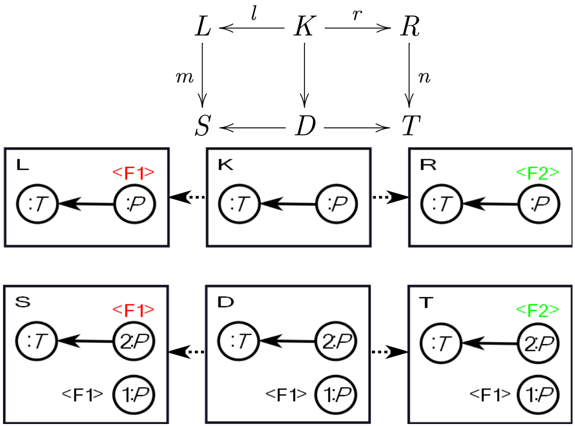
- Model Transformation is executed according to model transformation rules.
- Model transformation rules specify how to transform a model.
- Graph-based model transformation can be present as graph production.
- Each graph production $L \xleftarrow{l} K \xrightarrow{r} R$ consists the left-hand side, the gluing graph and the right-hand side, where graph morphisms l and r are injective.

Model Transformation Rules

Rule	L	K	R
Request	:P <F0>	:P	:P <F1>
SetFlag2	:T ← 1:P <F1>	:T ← 1:P	:T ← 1:P <F2>
CheckTurn	1:P <F1> :T ← 2:P <F0>	1:P :T ← 2:P <F0>	1:P <ST> :T ← 2:P <F0>
GetTurn	1:P <ST> :T ← 2:P	1:P 2:P	:T ← 1:P <F2> 2:P
Compete	R ← 1:P <F2> 2:P <F2>	1:P 2:P <F2>	1:P <F1> 2:P <F2>
Access	:R 1:P <F2> 2:P <F2>	:R 1:P <F2>	:R ← 1:P <F2>
Exit	:R ← :P <F2>	:R :P	:R :P <F0>

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

Double-Pushout Approach



Outline

Bounded Verification Approach

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

encoded to ⟹

Alloy Specification

⇓ *checked by*

Alloy Analyser

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

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.

Alloy Language

- 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.

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

Model Transformations Encoding

	Element	Encoded As
General	Model	*sig* Graph
	Direct Model Transformation	*sig* Trans
	Model Transformations	*sig* Path
Specific	Node	*sig* Node
	Edge	*sig* Edge
	rule application	*pred* rule
	constraints	*fact*

General Part

Model encoding

Models are encoded as the signature *Graph*

```
sig Graph{
nodes:set Node_i+...+Node_m,
edges:set Edge_j+...+Edge_n}
```

```
sig Graph{
nodes:set NP+NR+NT+Int,
edges:set APT+APR+APA}
```

```
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}
```

General Part

Transformation Encoding

Direct model transformations are encoded as the signature *Trans*

```
sig Trans{
rule: one Rule,
source:one Graph,
target:one Graph,
dnodes, anodes:set Node_i+...
    +Node_m,
dedges, aedges:set Edge_j+...
    +Edge_n}
```

```
sig Trans{
rule:one Rule,
source:one Graph,
target:one Graph,
dnodes,anodes:set NP+NR+NT+
    Int,
dedges,aedges:set APT+APR+
    APA}
```

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
rule_1[t] or ... or rule_n[t]}
fact{all t:Trans|valid_trans[t]}
```

A sequence of direct model transformations is is encoded as path signature of length n.

```
sig Path3{trans1, trans2, trans3: one Trans}
fact{all p:Path3|p.trans1.target=p.trans2.source and p.
    trans2.target= p.trans3.source}
fact{Path3.trans1+Path3.trans2+Path3.trans3=Trans}
```

Specific Part

Nodes and Edges

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.

```
sig Node_i{}//1≤i≤m
sig Edge_j{//1≤j≤n
    src:one Node_s,//1≤s≤m
    trg:one Node_t}//1≤t≤m
```

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

```
pred source_valid[t:Trans]{
    //mulitplicity on T min:1;max:1
    let count=#NT&t.source.nodes|count>=1 and count<=1
    //mulitplicity 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)
    ...
}
```

Direct Model Transformation Encoding

Matching

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.

```
one  m:L→S|(all  n:m(l(K_N))|n  in  t.source.nodes-t.dnodes)  and
     (all  e:m(l(K_E))|e  in  t.source.edges-t.dedges)  and
     (all  n:m(L_N)\m(l(K_N))|n  in  t.dnodes)  and
     (all  e:m(L_E)\m(l(K_E))|e  in  t.dedges)
one  n:R→T|(all  n:n(r(K_N))|n  in  t.target.nodes-t.anodes)  and
     (all  e:n(r(K_E))|e  in  t.target.edges-t.aedges)  and
     (all  n:n(R_N)\ n(r(K_N))|n  in  t.anodes)  and
     (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)

```
all e:t|e in t.dnodes implies (or_{i=1}^{i=m} (
    one m:replace(L,e,e_i)→S | (all n:m(l(K_N))| n in t.
        source.nodes-t.dnodes) and
    (all e:m(l(K_E))| e in t.source.edges-t.dedges) and
    (all n:m(replace(L_N,e,e_i))\m(l(K_N))|n in t.dnodes) and
    (all e:m(replace(L_E,e,e_i))\m(l(K_E))|e in t.dedges)))
```

Direct Model Transformation Encoding

Exactly one rule is applied

- 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)

```
#{trans.dnodes&t}=1              no trans.dnodes&t
#{trans.dedges&t}=1              no trans.dedges&t
#{trans.anodes&t}=1              no trans.anodes&t
#{trans.aedges&t}=1              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.

```
one d1,d2,...,dm,a1,a2,...,an|d1.st=d2.st=...=dm.st=a1.st=a2.st,
    ...=an.st
```

Outline

Bounded Verification Approach

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

encoded to ⟹ **Alloy Specification**

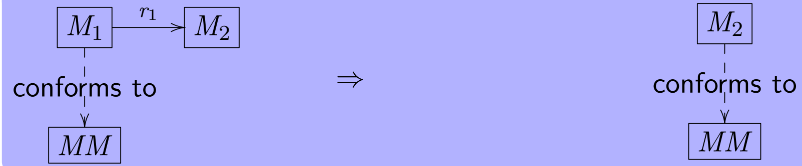
checked by ⇓

Alloy Analyser

Verification

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

Direct Condition



$$\forall \mathcal{S}, \mathcal{T} : \; \mathcal{S} \rhd \mathcal{SM} \wedge (\mathcal{S} \rightarrow \mathcal{T}) \Rightarrow \mathcal{T} \rhd \mathcal{TM} \qquad (1)$$

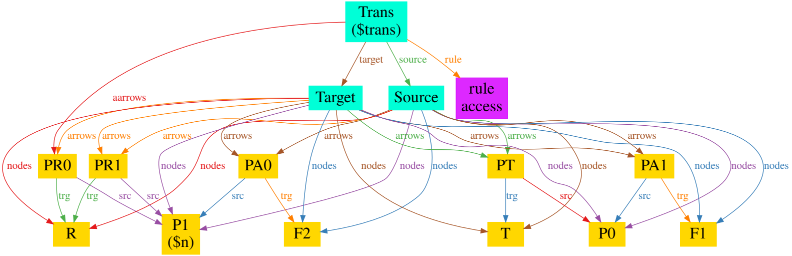
Verification Result

Commands

```
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
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
check{all trans:Trans|all n:(NF2&tran.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:



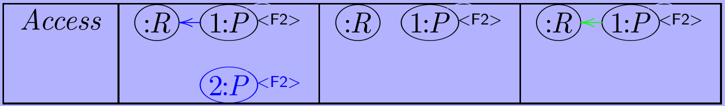
Fix the problem

- Add more constraints on the metamodel

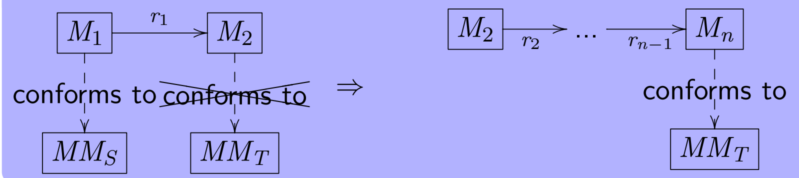
```
If a process is accessing the resource, it is at state <F2>
If a process is at state <ST>, it doesn't own turn
```

- Add NACS on certain rules

Revised Rule Access



Sequential Condition



$$\forall \mathbf{S}, \mathbf{T_0} \quad : \quad (\mathbf{S} \rhd \mathbf{M} \wedge \mathbf{S} \rightarrow \mathbf{T_0} \wedge \neg(\mathbf{T_0} \rhd \mathbf{M})) \Rightarrow \exists \mathbf{T_1}, ..., \mathbf{T_n} : \bigwedge_{i=1}^{n-1} \neg(\mathbf{T_i} \rhd \mathbf{M}) \wedge \mathbf{T_n} \rhd \mathbf{M} \wedge \bigwedge_{i=0}^{n-1} \mathbf{T_i} \rightarrow \mathbf{T_{i+1}} \tag{2}$$

Verification

- Verify systems by finding counterexamples
- Verify constraint which violates the direct condition
- Verify stepwise, 2, 3, . . . , n

Verification Result

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

```
sig Path{trans0,trans1:Trans}{trans0.target=trans1.source}
fact{all table:Table&trans0.source.nodes|some pk:PK&trans0.
    source.edges|pk.src=table}
fact{some table:Table&trans0.target.nodes|(no pk:PK&trans0.
    target.edges|pk.src=table) and (some pk1:PK&trans1.
    aedges|pk1.src=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($live$).

Liveness properties Verification

- 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

Liveness Properties Result

Loop sequence is found on a path of length 3

```
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])}
for 20 but exactly 1 Path3 exactly 3 Trans, exactly 3 Graph, exactly 1 Rule
pred pre[graph:Graph, n:P]{some e:(APA&graph.arrows)|e.src=n and e.trg=F1}
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)}
```

Analysis and Future work

- 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-define 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/`