Invariant Checking for Graph Transformation: Applications & Open Challenges

Dagstuhl Seminar 15451 on Verification of Evolving Graph Structures, November 2 - 6, 2015.

Holger Giese & Leen Lambers
Head of the System Analysis & Modeling Group
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Germany
holger.giese@hpi.de leen.lambers@hpi.de
Outline

1. Inductive Invariant Checking for Graph Transformation Systems

2. Applications
   Cyber-Physical Systems & Safety
   Model Transformations & Correctness

3. Summary & Open Challenges
Inductive Invariant Checking for Graph Transformation Systems

2. Applications
   Cyber-Physical Systems & Safety
   Model Transformations & Correctness

3. Summary & Open Challenges
A graph transformation system (we omit here NACs) consists of
- a type graph describing all possible model configurations,
- a set of rules $R$ with LHS and RHS, and
- a function $\text{prio} : R \rightarrow \text{Int}$ which assigns priorities to all rules.

We use graph constraints $\mathcal{C}$ (e.g., a set of forbidden graph patterns $F$) for defining unsafe situations.

- A rule $r$ of $R$ is enabled if an occurrence of its LHS in a graph $G$ exists.
- A rule $r$ of $R$ is applied on graph $G$ by replacing an occurrence of its LHS in $G$ by the RHS (DPO).
- A forbidden graph pattern $F_i$ in $F$ is respected by a graph $G$ if it is not contained.
**Def:** A graph constraint $C$ is an _inductive invariant for_ a set of graph rules $R$ if for all graphs $G$ and $H$ hand with $G \xrightarrow{R} H$ holds that if $G$ fulfills $C$ then also $H$ fulfills $C$.

$$\forall (G \xrightarrow{R} H): (G \models C) \Rightarrow (H \models C)$$
**Observation:** any possible counter-example must contain an intersection between the nodes of the RHS of the rule and the forbidden graph $F$. Therefore, if $(P,r)$ is a counterexample, then:

1. There exists a $P'$ which is the combination of a RHS of a rule $r$ and a forbidden graph pattern $F$,

2. $P \xrightarrow{r} P'$ (which implies that no rule $r'$ with higher priority can be applied), and

3. There exists no forbidden graph $F'$ which matches $P$ (as then the graph before was not correct already)

**Idea:** Algorithm constructs all possible counterexamples and checks whether any could be a real one.
Outline

1. Inductive Invariant Checking for Graph Transformation Systems

Applications

Cyber-Physical Systems & Safety
Model Transformations & Correctness

3. Summary & Open Challenges
Outline

1. Inductive Invariant Checking for Graph Transformation Systems
2. Applications
   - Cyber-Physical Systems & Safety
   - Model Transformations & Correctness
3. Summary & Open Challenges
2. Applications: Cyber-Physical Systems & Safety

(Networked) Cyber-Physical Systems

- Smart Factory - E.g. Industry 4.0
- Smart Logistic
- Micro Grids

Internet of Things

Smart City

Ultra-Large-Scale Systems

Smart Home

E-Health

Ambient Assisted Living

Example: RailCab
- Challenges -

http://www.railcab.de/

RailCab: A shuttle system that builds convoys to optimize the energy consumption

Test shuttle

Test track
Example: RailCab
- Modeling Idea -

**Modeling Problem:**
- Shuttles move on a topology of tracks
- Arbitrary large topologies

**Solution:**
- State = Graph
- Reconfiguration rules = graph transformation rules
- Safety properties = forbidden graphs
  $\Rightarrow$ Formal Verification possible
Example
- A Naïve First Design -

- Map the tracks
- Map the shuttles

Map the movement to rules (movement equals dynamic structural adaptation on the abstract level)

LHS

RHS

Rule:

s:Shuttle → t:Track

t:Track → t′:Track

s:Shuttle → t:Track

t:Track → t′:Track
Example
- A Naïve First Design -

Forbidden Graph

s1:Shuttle  s2:Shuttle

Rule:

s1:Shuttle  t1:Track  t2:Track  t1:Track  t2:Track  s1:Shuttle
Example - Analysis Challenge -

Forbidden Graph

- **Correctness:** all reachable system graphs do not match the forbidden graph pattern

Problems:
- fixed initial topology is **not known** (may change)
- there could be **infinite** many initial topologies
- there could be **infinite** many reachable system state graphs when the topology evolves (new tracks)
Example - A Naïve Second Design -

Forbidden Graph

- **Correctness:** all reachable system graphs do not match the forbidden graph pattern

```
Forbidden Graph

t:Track

s1:Shuttle  s2:Shuttle
```

**Remark:** still too naïve as shuttles require time to break and convoys are not considered

```
Rule:

- t1:Track → t2:Track
- t1:Track → t2:Track
```

Example 1

- Track1
  - Shuttle1
- Track2
  - Shuttle2

Example 2

- Track1
  - Shuttle1
  - t1:Track
- Track2
  - Shuttle2
  - t2:Track

Example 3

- Track1
  - Shuttle1
- Track2
  - t1:Track → t2:Track
- Track2
  - s1:Shuttle

Example 4

- Track1
  - Shuttle1
- Track2
  - s1:Shuttle

```
Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
```
Example
- A More Realistic Design -

Forbidden Graph

```
t:Track

s1:Shuttle
s2:Shuttle

Distance Coordination
```

**Now**, the analysis results can guarantee the absence of collisions!

**Rules:**

modeling: 6 graph transformation rules and 15 forbidden graphs

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Analysis - Model Checking -

Model checking:
- Backend tool: the GTS model checker GROOVE
- Topology with only 15 tracks

Verification times:
- 3 shuttles \(\Rightarrow\) 2 min
- 4 shuttles \(\Rightarrow\) 7 min
- 5 shuttles \(\Rightarrow\) 55 min
- 6 shuttles ???
Analysis
- Invariant Checking -

Verification:
- Analyze whether structural changes can lead from safe to unsafe situations (inductive invariants)
  - Supports infinite many start configurations specified only by their structural properties
  - Supports infinite state models
  - Produced counterexamples allow to incrementally develop the right inductive invariants
Invariant checking:
- Infinite many initial configurations
- Infinite large models

Verification times:
- explicit ⇒ 34 min
- symbolic ⇒ 5 min (Backend tool: CrocoPat)

<table>
<thead>
<tr>
<th>Rule / pattern (nodes:edges)</th>
<th>explicit</th>
<th>symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td>goDC1(7:10) / invalidDCPattern(5:4)</td>
<td>744 s</td>
<td>11.2 s</td>
</tr>
<tr>
<td>goDC2(6:09) / invalidDCPattern(5:4)</td>
<td>170 s</td>
<td>6.5 s</td>
</tr>
<tr>
<td>goDC1(7:10) / noDC(4:4)</td>
<td>20 s</td>
<td>16.8 s</td>
</tr>
<tr>
<td>goDC2(6:09) / noDC(4:4)</td>
<td>7 s</td>
<td>13.1 s</td>
</tr>
<tr>
<td>goDC1(7:10) / unambiguousOn(3:2)</td>
<td>60 s</td>
<td>6.1 s</td>
</tr>
<tr>
<td>goDC2(6:09) / unambiguousOn(3:2)</td>
<td>36 s</td>
<td>2.9 s</td>
</tr>
<tr>
<td>goDC1(7:10) / unambiguousNext(3:2)</td>
<td>48 s</td>
<td>4.1 s</td>
</tr>
<tr>
<td>goDC2(6:09) / unambiguousNext(3:2)</td>
<td>33 s</td>
<td>2.1 s</td>
</tr>
</tbody>
</table>
Semantics & Time

Timed automata
- clock variables with
  - invariants for states
  - conditions and assignments/resets for transitions

Semantics:
- **instantaneous transitions** where no time passes by
- **time transitions** where time can pass by and the automata stays within a state (clock values increase)

![Diagram showing states and transitions with time constraints](image.png)
A **timed graph transformation system** (we omit NACs) consists of

- a type graph describing all possible model configurations and the clocks of each node type,
- a set of rules $R$ with LHS, RHS, clock condition, and clock updates,
- a subset of $R$ of the urgent rules, and
- a function $prio: R \rightarrow \mathbb{Int}$ which assigns priorities to all rules.

- A state $(G,a)$ is a graph $G$ plus an evaluation $a$ for all clocks.
- An **instantaneous step** (rule application) $(G,a) \xrightarrow{r} (G',a')$, where $G'$ results from $G$ applying the GTS rule and $a'$ is derived from $a$ by applying the clock update of $r$, is instantaneous and no time passes.
- A **time step** $(G,a) \xrightarrow{\delta} (G,a')$ where $a'$ is derived from $a$ by adding the delay $\delta$ to each evaluation to denote the passing of time, which can only happen when in between no urgent rule is enabled.
Modeling & Time
- Example -

**Type graph:**

**Simple instance graph:**

---

**Rules (some only):**

\{ s1.timeAtTrack >= 10; s1.timeAtTrack' = 0 \}

---

modeling: only 4 graph transformation rules and 2 forbidden pattern

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Analysis
- Invariant Checking -

Checker: solve 1) as for GTS and encode 2) using linear inequalities (CPLEX)

1) Determine all source target and applicable rule $r$ such that
- the invariants holds for the source pattern,
- the resulting target pattern potentially breaks the invariant

2) Determine for a counterexample from 1) consisting of a source target and applicable rule $r$ whether
- the resulting target pattern breaks the invariant for $t \geq 0$ and
- no urgent rule is enabled for a $t'$ with $0 \leq t' < t$. 

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Modeling: 4 rules and 2 forbidden pattern with time vs. 6 rules and 15 forbidden pattern for the untimed case

Verification: average of 329 ms for the timed case vs. 5 min for the untimed case

Remark:
- Also a Hybrid GTS and a possible mapping of the 2nd step of the invariant checking problem to a model checker for hybrid automata (PHAVer) has been developed.
Outline

1. Inductive Invariant Checking for Graph Transformation Systems
2. Applications
   - Cyber-Physical Systems & Safety
   - Model Transformations & Correctness
3. Summary & Open Challenges
Why Model Transformations?

“The Heart and Soul of Model-Driven Engineering” [SK03]

System idea

model

model

model

code

| Translation (1), simulation (2), refinement (3), normalization (4), synthesis (5), ...

Model manipulations: e.g.
Correct Model Transformations

System idea

- model
- model
- model
- code
Model Transformations in a Nutshell

Model transformation developer perspective

Metamodel Source LG → Definition Transformation → Metamodel Target LG

Model transformation user perspective

Source model → Transformation engine → Target model

Event: first, pre,post, Send, Rcv

State: TR, TS, src,tgt, init
Correct Model Transformations

Transformation correctness/Input-Independent/Correctness from developer perspective

Source Semantics

Analysis

Definition Transformation

Target Semantics

Metamodel Source LG

Metamodel Target LG

Source model

Transformation engine

Target model

Check

Result Correctness/Input-dependent/Correctness from user perspective

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
## Behavior Definition using Graph Transformation

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
<th>Invariant Checking for GT: Applications &amp; Open Challenges</th>
<th>Holger Giese, Leen Lambers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S</strong></td>
<td><strong>initE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>first</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>++</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td><strong>active</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Send</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rcv</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>first</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre,post</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>¬E</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Send</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>post</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rcv</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>post</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Send</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>post</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rcv</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>post</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>¬</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>active</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Type graph**

**send**

**rcv**

**Its**

**initE**

**S**

**S1**

**S2**

**S3**
Behavior Preservation via Bisimulation
Behavior Preservation via Bisimulation

In the diagram, the states and events are connected to illustrate behavior preservation through bisimulation. The diagram includes states labeled as $S_1$, $S$, and $T$, events like Send and Rcv, and transitions indicating pre and post conditions.

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Invarian Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Behavior Preservation via Bisimulation

Transformation correctness/Input-Independent/Correctness from developer perspective

Source Semantics

Metamodel Source LG

Definition Transformation

Target Semantics

Metamodel Target LG

Analysis

Source model

Transformation engine

Target model

Check

Result Correctness/Input-dependent/Correctness from user perspective

LTSs of each source and target model of the MT bisimilar?
1. **MT definition only allows source and target models that are in the bisimulation relation**

2. Behavior definitions of source and target modeling languages preserve bisimulation relation
Behavior Preservation
Two-Step Algorithm

1. MT definition only allows source and target models that are in the bisimulation relation

2. Behavior definitions of source and target modeling languages preserve bisimulation relation
Behavior Preservation
Two-Step Algorithm

1. MT definition only allows source and target models that are in the bisimulation relation

2. Behavior definitions of source and target modeling languages preserve bisimulation relation
Behavior Preservation

MT languages can be operational (e.g., QVT Operational), relational (e.g., TGGs, QVT Relational) or hybrid (e.g., ATL).
Triple Graph Grammar = Relational Specification

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Triple Graph Grammar = Relational Specification
Triple Graph Grammar = Relational Specification
Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Conformance Checking
Relational Specification ~ Implementation

Advanced issues:
- TGG formalization
- Bookkeeping
- Optimizations

[AGTIVE11, SoSyM12]
Behavior Preservation
Two-Step Algorithm

1. MT definition only allows source and target models that are in the bisimulation relation
   - Relational definition (TGGs) vs. Operational definition (SDs)

   ![Diagram of TGG rules and relations]

ICA12012

Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
Invariant Checking for GT: Applications & Open Challenges | Holger Giese, Leen Lambers
1. **MT definition only allows source and target models that are in the bisimulation relation**

   - Relational definition (TGGs) vs. **Operational definition (SDs)**

---

**Behavior Preservation**

**Two-Step Algorithm**

---

**1. MT definition only allows source and target models that are in the bisimulation relation**

- Relational definition (TGGs) vs. **Operational definition (SDs)**
- Addressing non-deterministic behavioral rule application
- Case Study Sequence Charts 2 Communicating Automata
- Addressing not only behavioral equivalence, but also e.g. behavioral refinement
## Tool Support

### Invariant Checking for GT: Applications & Open Challenges

**Holger Giese, Leen Lambers**

**ICGT2015**

<table>
<thead>
<tr>
<th>Example</th>
<th>Characteristics</th>
<th>SeekSat/ProCon</th>
<th>without advanced implication check</th>
<th>with advanced implication check</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Invcheck-total</td>
<td>Invcheck-partial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>time (s)</td>
<td>result</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>4</td>
<td>20</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>4</td>
<td>20</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - TGG</td>
<td>subproblem</td>
<td>4</td>
<td>&lt;1</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - TGG</td>
<td>subproblem</td>
<td>5</td>
<td>10</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>5</td>
<td>9</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>11</td>
<td>40</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>12</td>
<td>out of memory</td>
<td>timeout</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>17</td>
<td>17</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - TGG</td>
<td>subproblem</td>
<td>20</td>
<td>timeout</td>
<td>&lt;1</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>30</td>
<td>20</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>70</td>
<td>40</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>72</td>
<td>timeout</td>
<td>&lt;1</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>78</td>
<td>6.5</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>188</td>
<td>out of memory</td>
<td>1.5</td>
</tr>
<tr>
<td>Car Platooning</td>
<td>subproblem</td>
<td>258</td>
<td>&lt;1</td>
<td>true</td>
</tr>
<tr>
<td>Car Platooning</td>
<td>subproblem</td>
<td>610</td>
<td>&lt;1</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>subproblem</td>
<td>807</td>
<td>timeout</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Car Platooning</td>
<td>complete</td>
<td>947</td>
<td>&lt;1</td>
<td>false</td>
</tr>
<tr>
<td>MT - Simple - TGG</td>
<td>subproblem</td>
<td>2778</td>
<td>220</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - TGG</td>
<td>subproblem</td>
<td>2778</td>
<td>226</td>
<td>true</td>
</tr>
<tr>
<td>MT - Simple - Semantics</td>
<td>complete</td>
<td>3870</td>
<td>timeout</td>
<td>1.5</td>
</tr>
<tr>
<td>MT - Complex - TGG</td>
<td>complete</td>
<td>5556</td>
<td>562</td>
<td>true</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>subproblem</td>
<td>607312</td>
<td>out of memory</td>
<td>timeout</td>
</tr>
<tr>
<td>MT - Complex - Semantics</td>
<td>complete</td>
<td>607500</td>
<td>out of memory</td>
<td>timeout</td>
</tr>
<tr>
<td>MT - Complex - TGG</td>
<td>complete</td>
<td>1817622</td>
<td>timeout</td>
<td>timeout</td>
</tr>
</tbody>
</table>
Summary
Correct Model Transformations

Transformation correctness/Input-Independent/Correctness from developer perspective

Equivalence

Source Semantics

Metamodel Source LG

Definition Transformation

Target Semantics

Metamodel Target LG

Source model

Transformation engine

Target model

Check

Result Correctness/Input-dependent/Correctness from user perspective
Outline

1. Inductive Invariant Checking for Graph Transformation Systems

2. Applications
   Cyber-Physical Systems & Safety
   Model Transformations & Correctness

Summary & Open Challenges
3. Summary

- **Invariant checking** can be used to
  - establish state properties required for safety (e.g., forbidden hazardous situations) for systems where the state space can be captured by evolving graph structures or
  - verify complex properties of model transformations such as behavior preservation from the developer perspective.

- **More expressive variants** can lead to more compact models and also less scalability issues (e.g., more natural encoding of time).

- The feature of our invariant checking that somehow minimal **counterexamples** are generated helps to incrementally develop the right inductive invariants.
1) Oftentimes very strong inductive invariants are required that are not very intuitive.
   ➡ Support debugging of semantics and invariants
   ➡ Support automated strengthening of the inductive invariants

2) Expressiveness of the GTS variants (e.g., data, OO concepts, time, etc.) and the related invariant checker support is sometimes a limited factor.
   ➡ Support more GTS variants and invariants
   ➡ Native vs. non-native GT invariant checking

3) Scalability of the invariant checker
   ➡ Special support for specific cases (behavioral rules, TGGs, …)
   ➡ Native vs. non-native GT invariant checking
Thank you!
References


