Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

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**

Outline

1. **Inductive Invariant Checking for Graph Transformation Systems**
2. **Applications**

   Cyber-Physical Systems & Safety

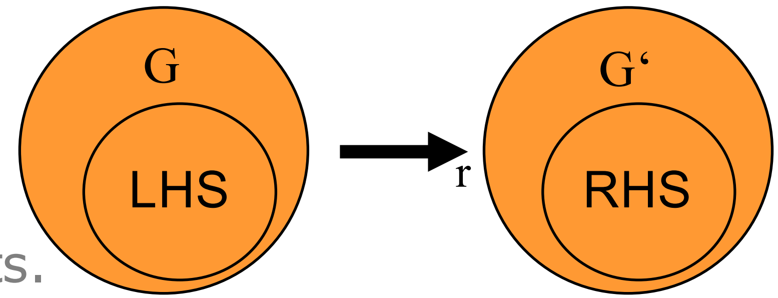
   Model Transformations & Correctness
3. **Summary & Open Challenges**

1. Graph Transformation Systems

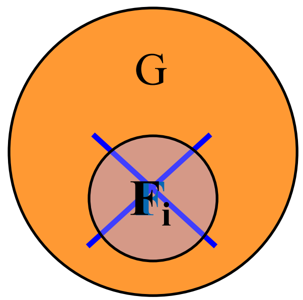
A **graph transformation system** (we omit here NACs) consists of

- a type graph describing all possible model configurations,
- a set of rules R with LHS and RHS, and
- a function $prio: R \rightarrow Int$ which assigns priorities to all rules.

We use graph constraints 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 $\mathbf{F_i}$ in F is **respected** by a graph G if it is not contained.

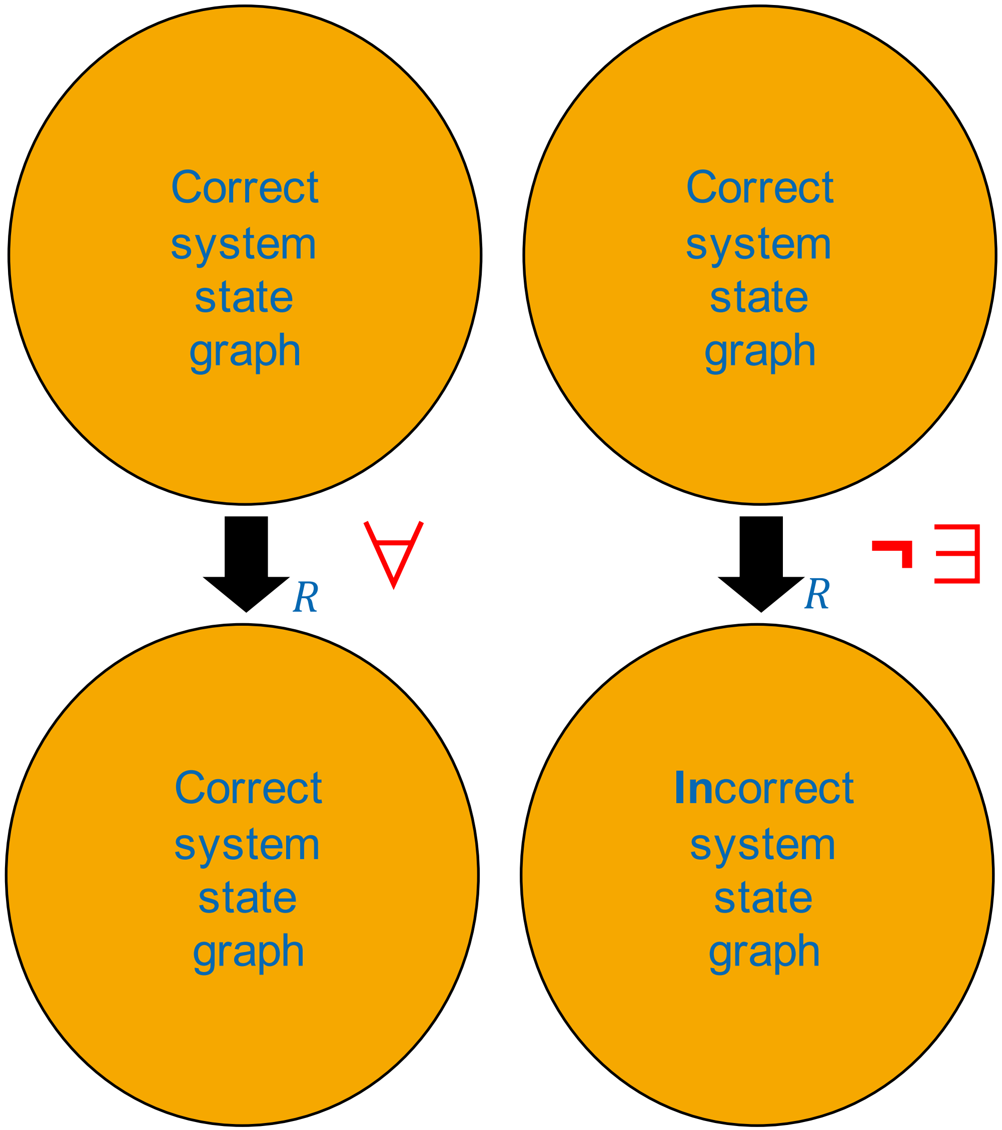


Analysis
- Inductive Invariants -

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 $G \rightarrow_R H$ holds that if G fulfills C then also H fulfills C.

$\forall (G \rightarrow_R H)$:

$$(G \vDash C) \Rightarrow (H \vDash C)$$



Analysis
- Invariant Checking -

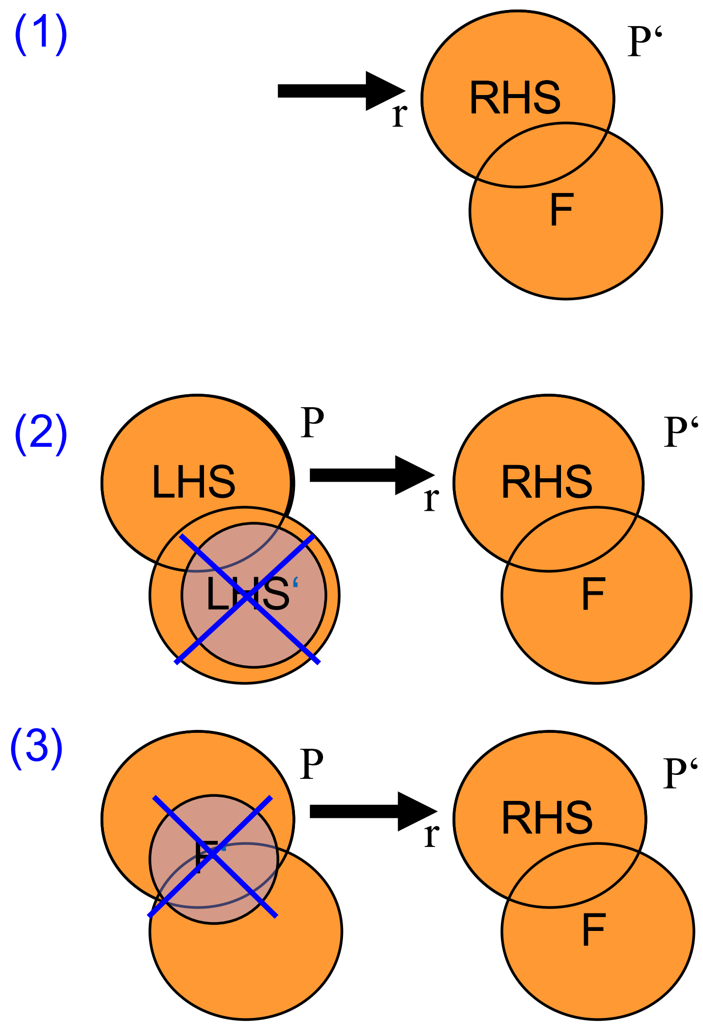
Observation: any possible counter-example must contain an intersection between the nodes of the RHS of the rule and the forbidden graph F. Therefore, if (P,r) is a **counterexample**, then:

(1) exists a P' which is the combination of a RHS of a rule r and a forbidden graph pattern F,

(2) $P \rightarrow_r P'$ (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**

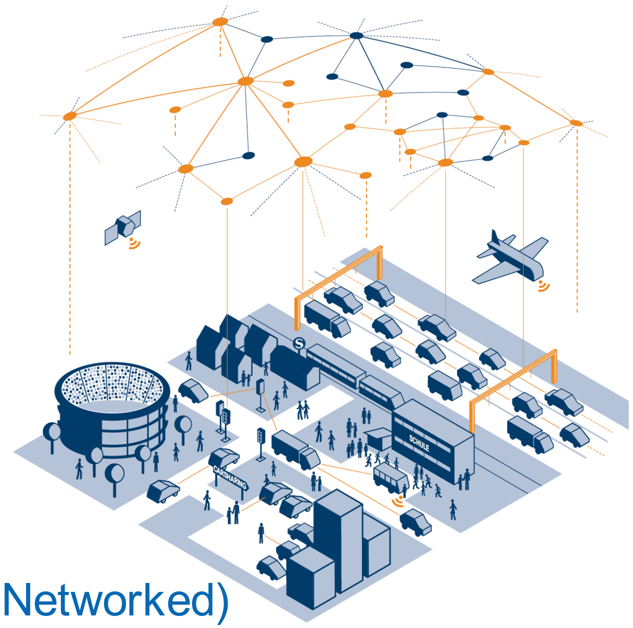
Outline

1. **Inductive Invariant Checking for Graph Transformation Systems**
2. **Applications**

   Cyber-Physical Systems & Safety

   Model Transformations & Correctness
3. **Summary & Open Challenges**

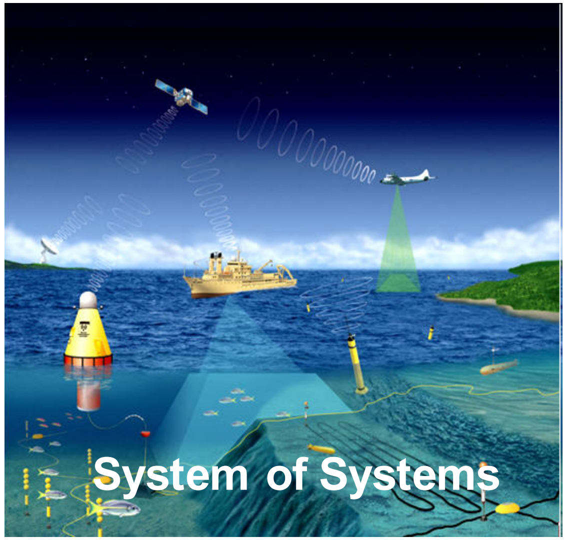
2. Applications: Cyber-Physical Systems & Safety



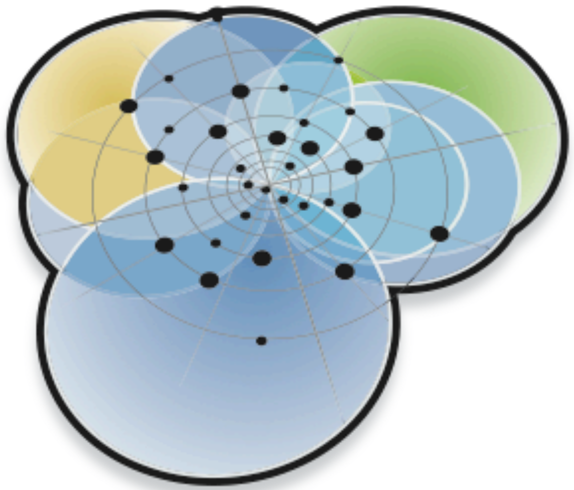
(Networked)
Cyber-Pyhsical Systems

Internet of Things

Smart City



http://oceanservice.noaa.gov/news/weeklynews/nov13/ioos-awards.html



Ultra-Large-Scale Systems

Smart Home

E-Health

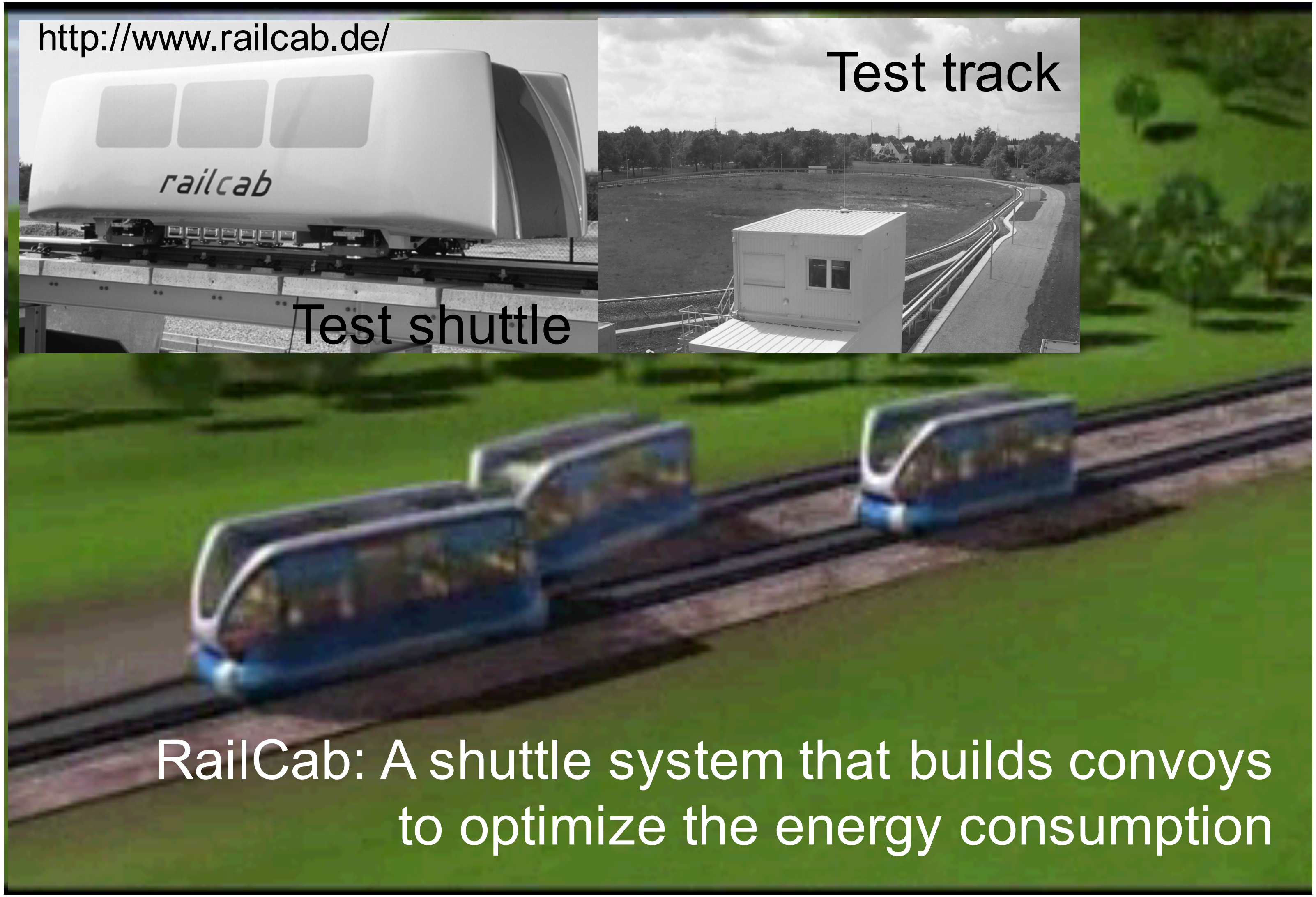
Ambient
Assisted Living

Smart Factory -
E.g. Industry 4.0

Smart Logistic

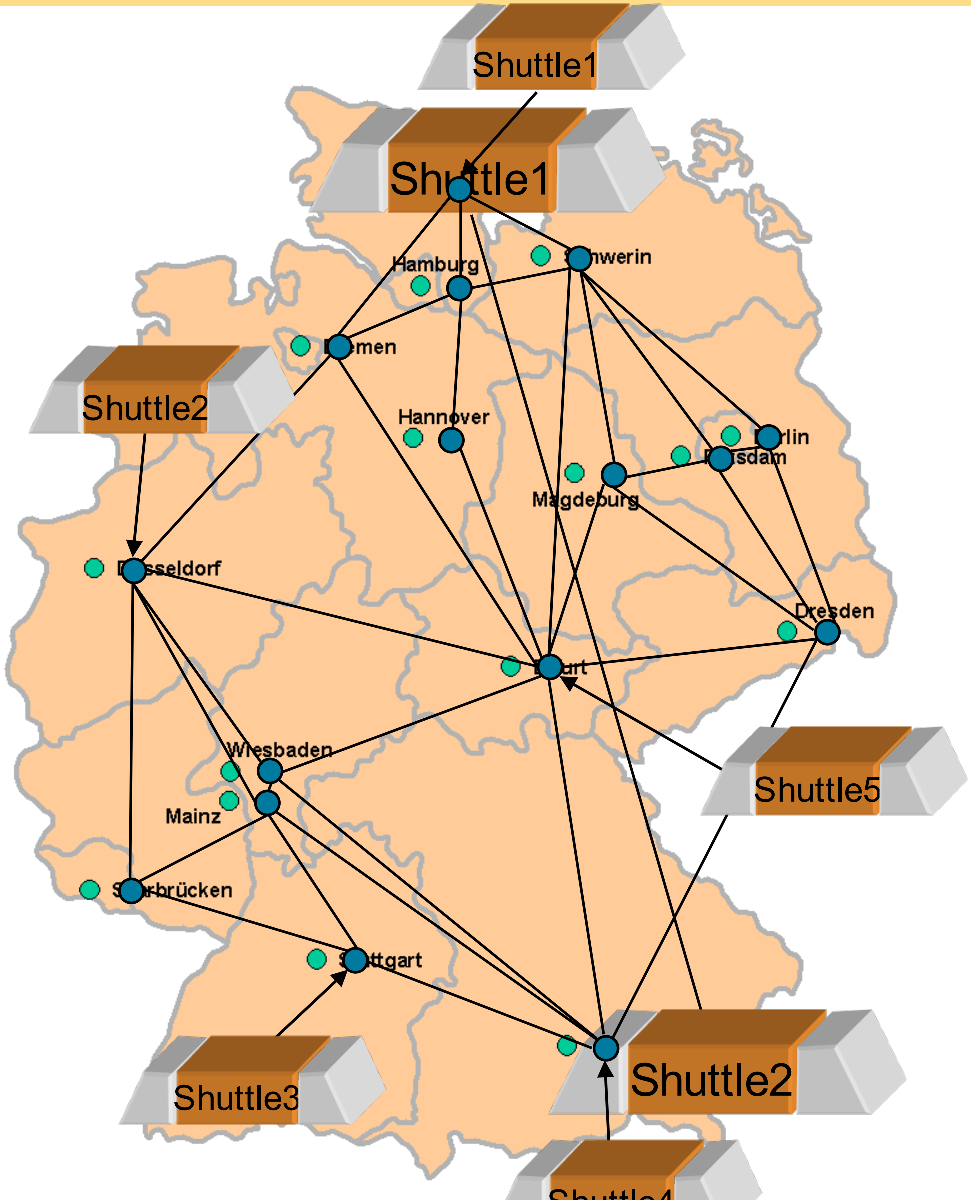
Micro Grids

Example: RailCab
- Challenges -



Example: RailCab
- Modeling Idea -

Hasso Plattner Institut



Modeling Problem:

- Shuttles move on a topology of tracks
- Arbitrary large topologies

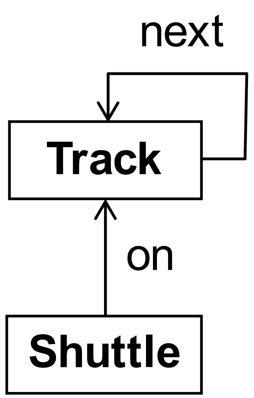
Solution:

- State = Graph
- Reconfiguration rules = graph transformation rules
- Safety properties = forbidden graphs

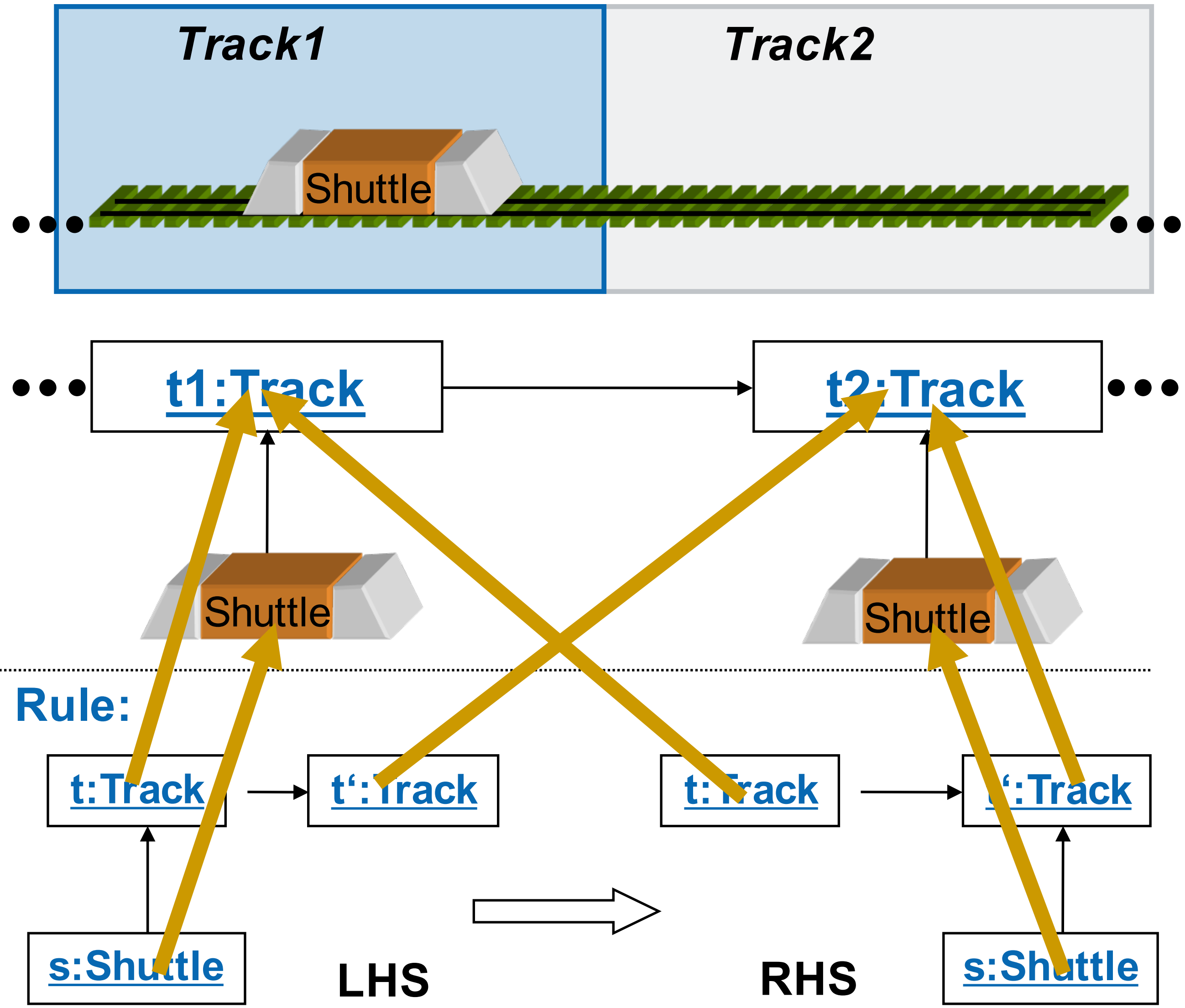
⇨ Formal Verification possible

Example
- A Naïve First Design -

- Map the tracks
- Map the shuttles

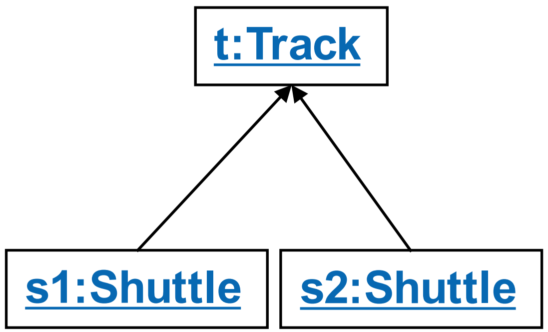


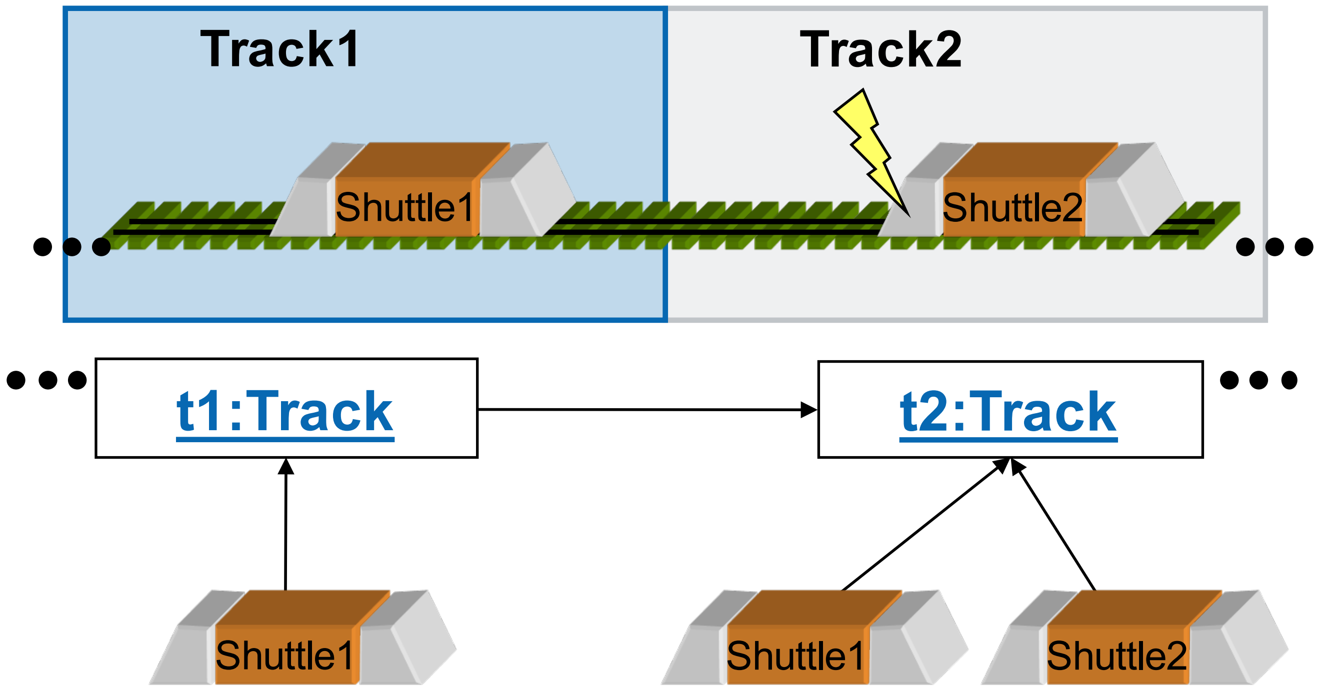
- Map the movement to rules (movement equals dynamic structural adaptation on the abstract level)



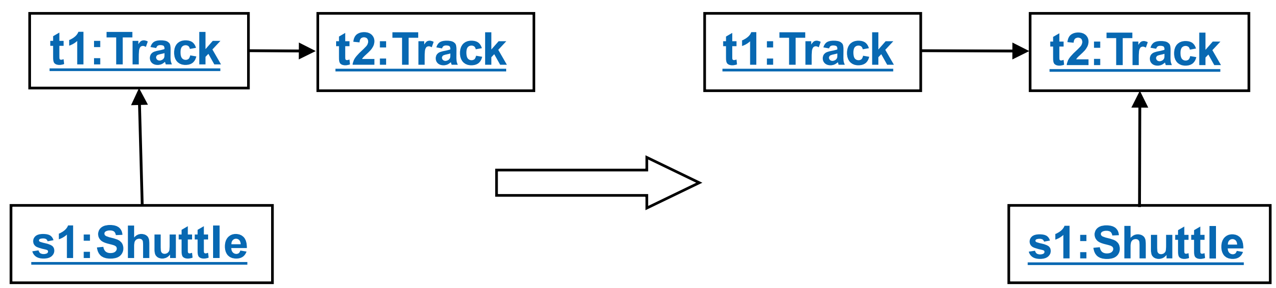
Example
- A Naïve First Design -

Forbidden Graph



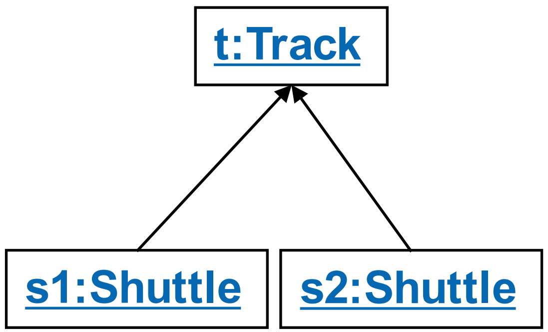


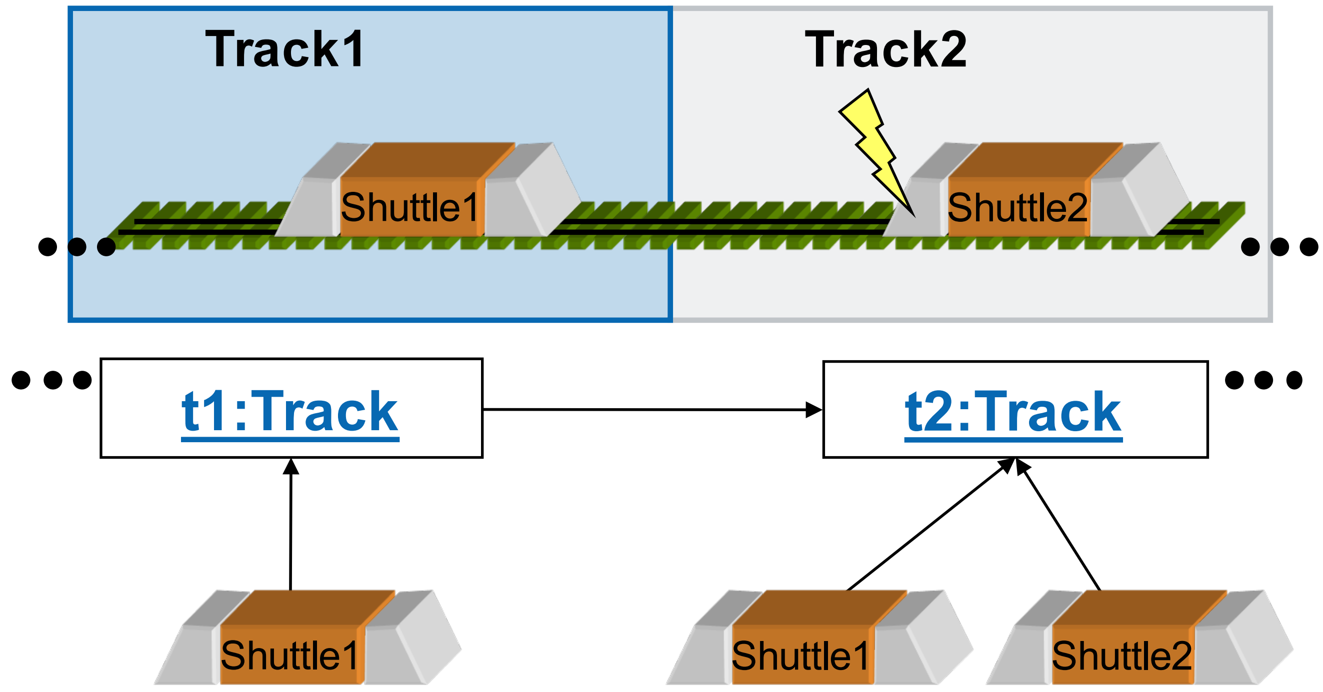
Rule:



Example - Analysis Challenge -

Forbidden Graph





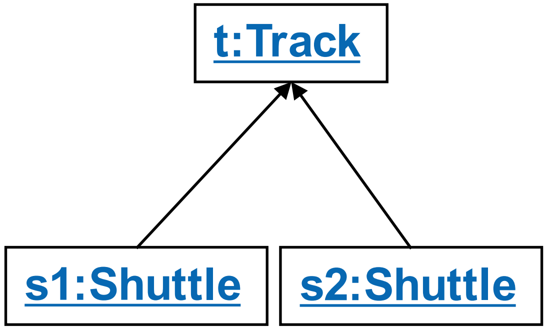
- **Correctness:** all reachable system graphs do not match the forbidden graph pattern

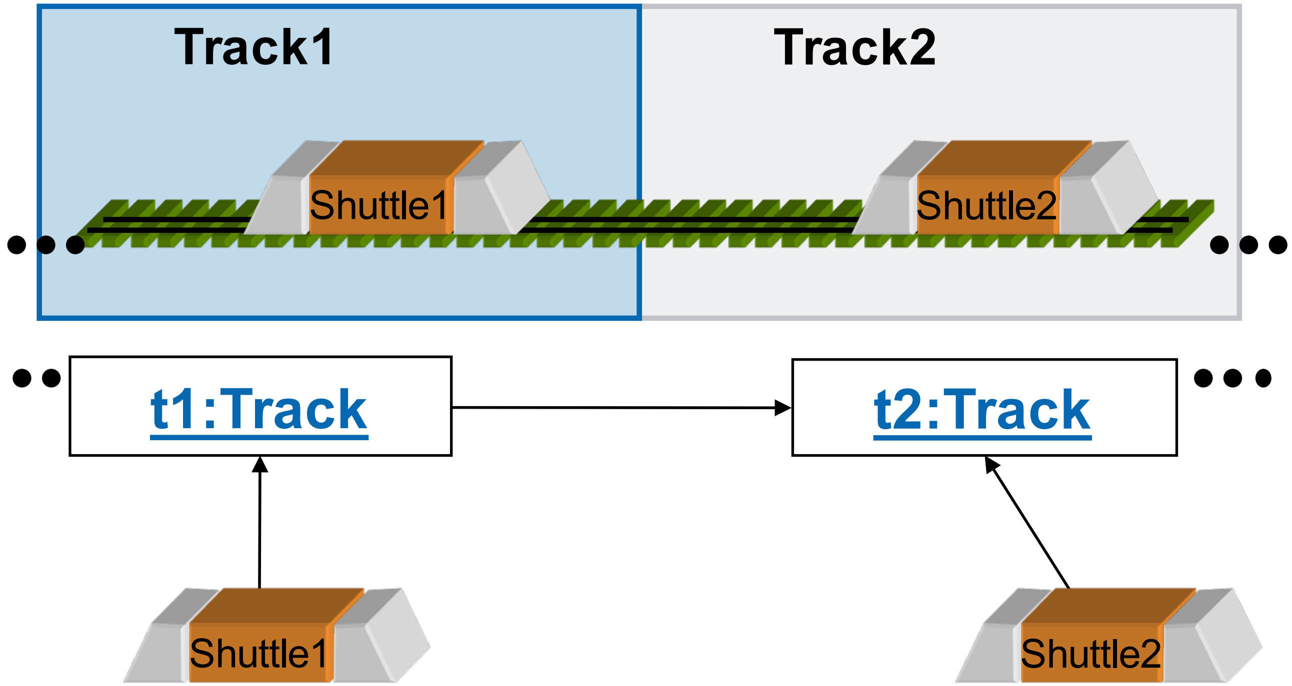
Problems:

- 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

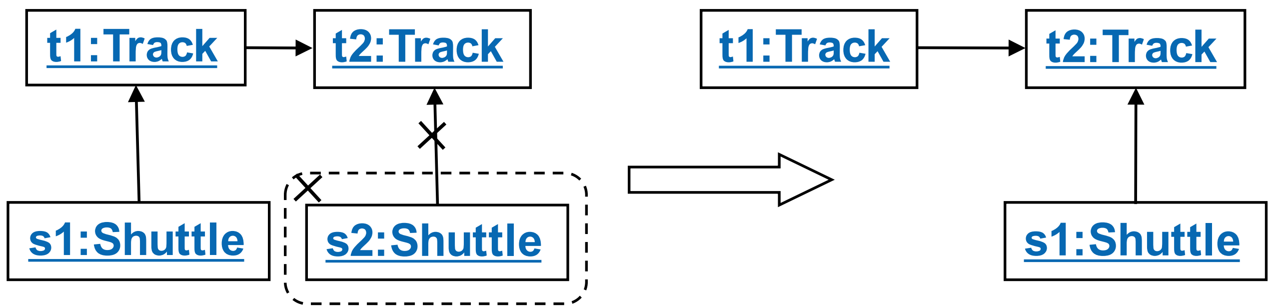




- **Correctness:** all reachable system graphs do not match the forbidden graph pattern

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

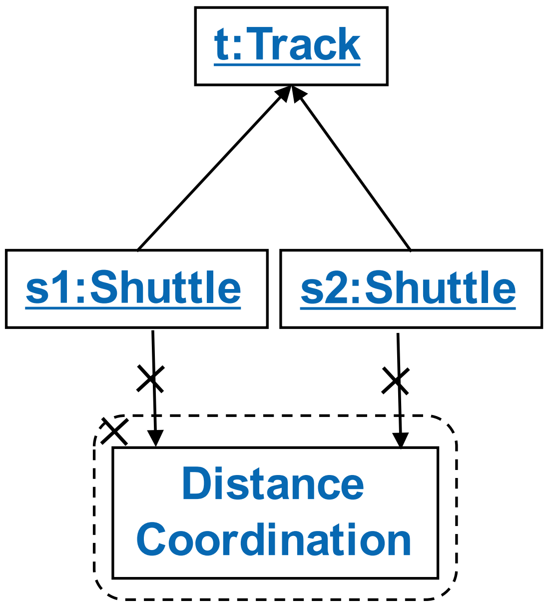
Rule:

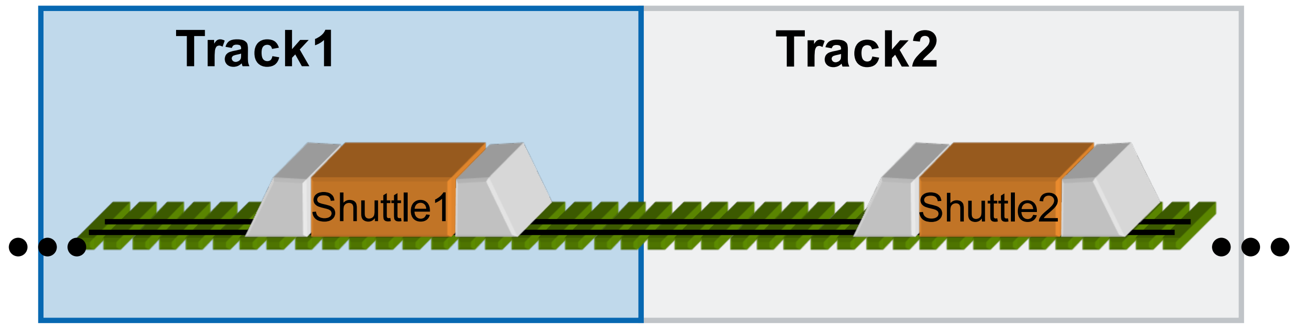


Example
- A More Realistic Design -

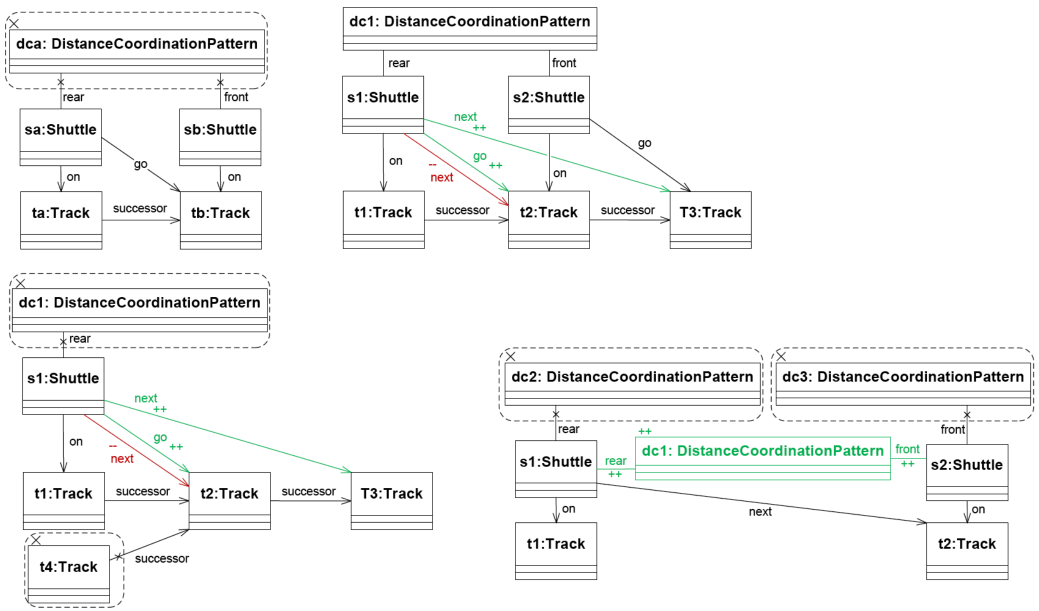
Hasso Plattner Institut

Forbidden Graph





Rules:



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

modeling: 6 graph transformation rules and 15 forbidden graphs

Analysis
- Model Checking -

Hasso Plattner Institut

[ICSE2006]

Model checking:

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

FUJABA Real-Time Tool Suite

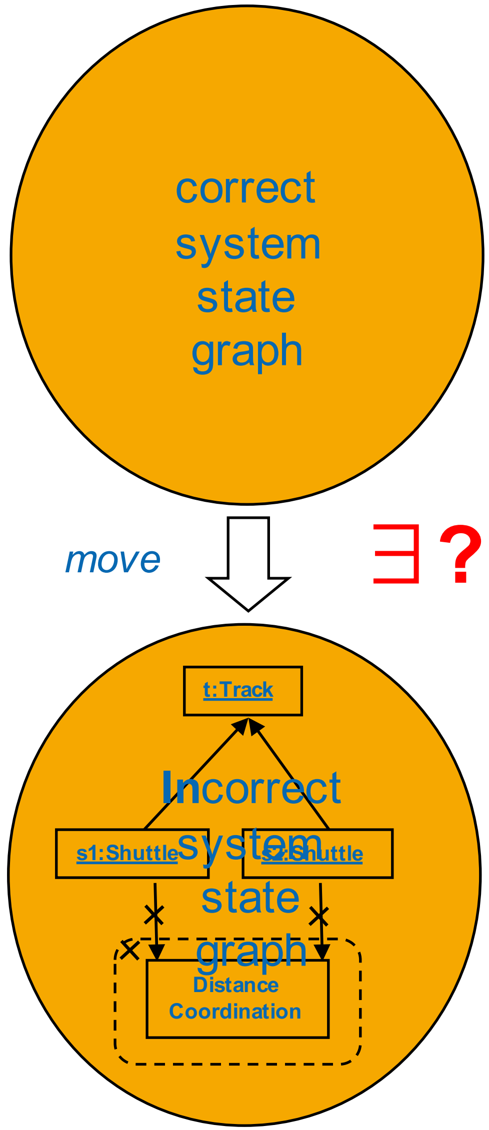
Verification times:

- 3 shuttles ⇨ 2 min
- 4 shuttles ⇨ 7 min
- 5 shuttles ⇨ 55 min
- 6 shuttles ???

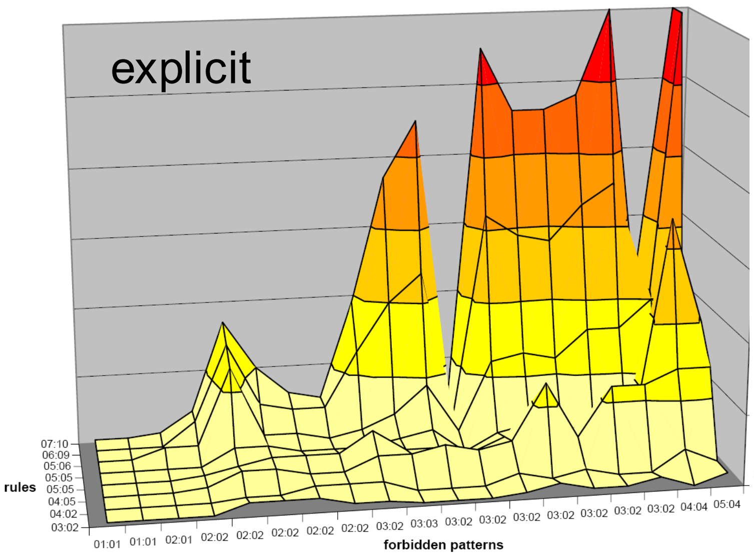
Analysis
- Invariant Checking -

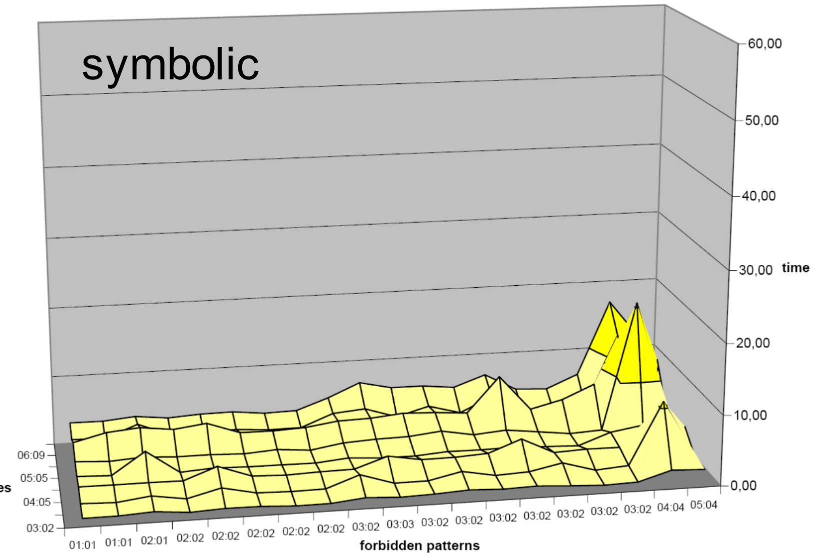
Verification:

- 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



Analysis
- Invariant Checking -





[ICSE2006]

The same scaling!

Rule / pattern (nodes:edges)	explicit	symbolic
goDC1(7:10) / invalidDCPattern(5:4)	744 s	11.2 s
goDC2(6:09) / invalidDCPattern(5:4)	170 s	6.5 s
goDC1(7:10) / noDC(4:4)	20 s	16.8 s
goDC2(6:09) / noDC(4:4)	7 s	13.1 s
goDC1(7:10) / unambigousOn(3:2)	60 s	6.1 s
goDC2(6:09) / unambigousOn(3:2)	36 s	2.9 s
goDC1(7:10) / unambigousNext(3:2)	48 s	4.1 s
goDC2(6:09) / unambigousNext(3:2)	33 s	2.1 s

Invariant checking:

- Infinite many initial configurations
- Infinite large models

Verification times:

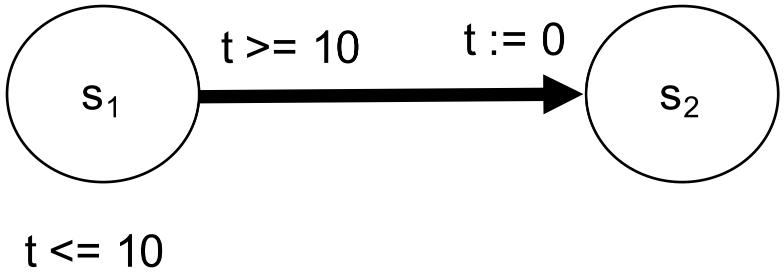
- explicit ⇨ 34 min
- symbolic ⇨ 5 min (Backend tool: CrocoPat)

Semantics & Time

[ISORC2008]

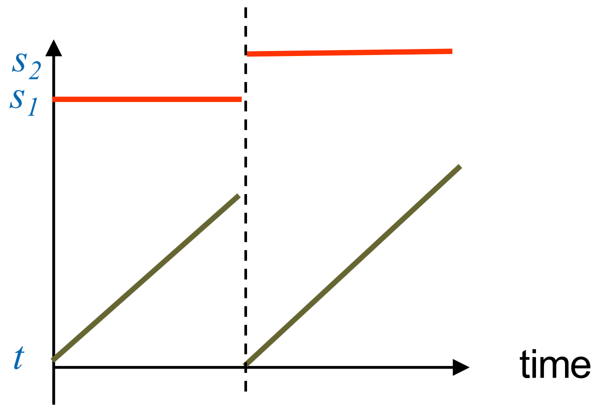
Timed automata

- 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)



Modeling & Time
- Timed GTS (TGTS) -

[ISORC2008]

A **timed graph transformation system** (we omit NACs) consists of

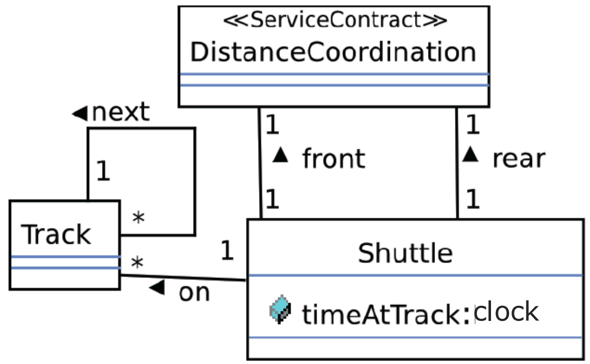
- a type graph describing all possible model configurations and the clocks of each node type,
- a set of rules R with LHS, RHS, clock condition, and clock updates,
- a subet of R of the urgent rules, and
- a function $prio: R \rightarrow Int$ which assigns priorities to all rules.

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

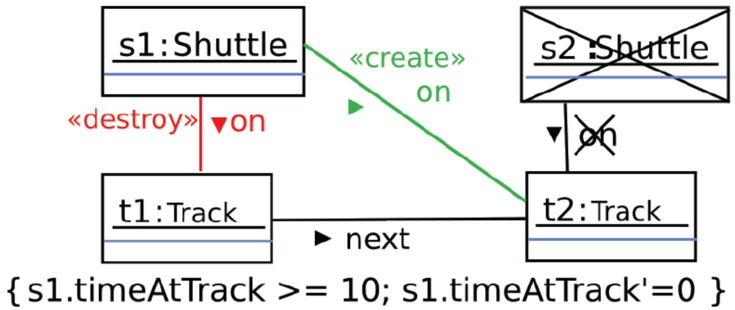
Modeling & Time
- Example -

[ISORC2008]

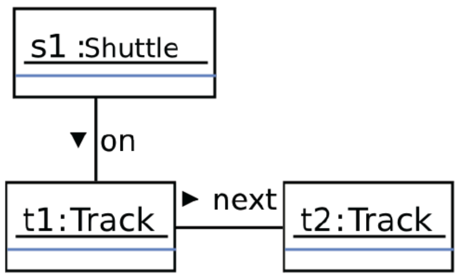
Type graph:

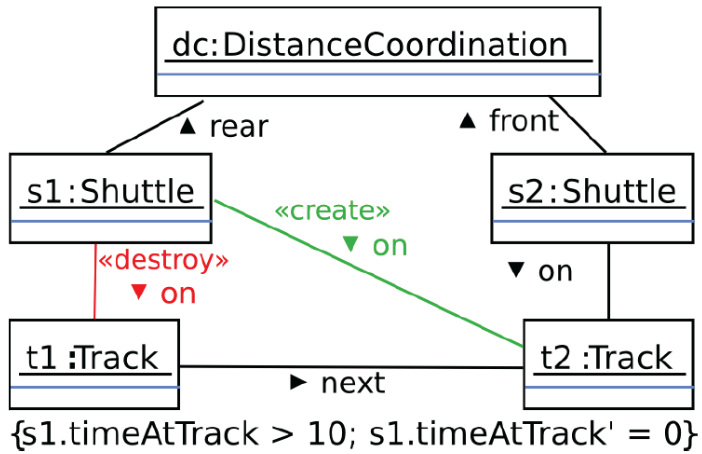


Rules (some only):



Simple instance graph:

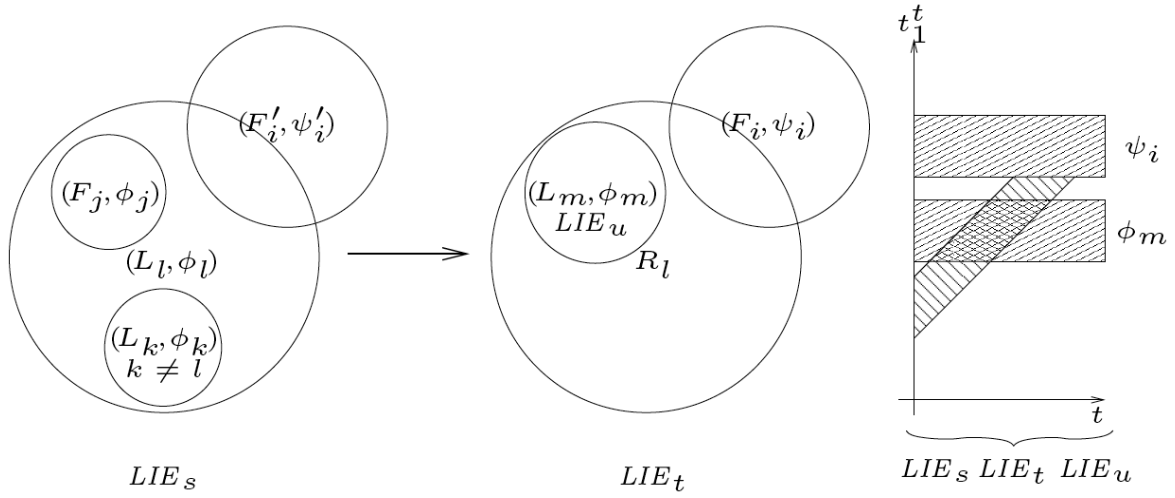




modeling: only 4 graph transformation rules and 2 forbidden pattern

Analysis
- Invariant Checking -

Hasso Plattner Institut



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

[ISORC2008]

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 >= 0 and
- no urgent rule is enabled for a t' with 0 <= t' < t.

Summary

- **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

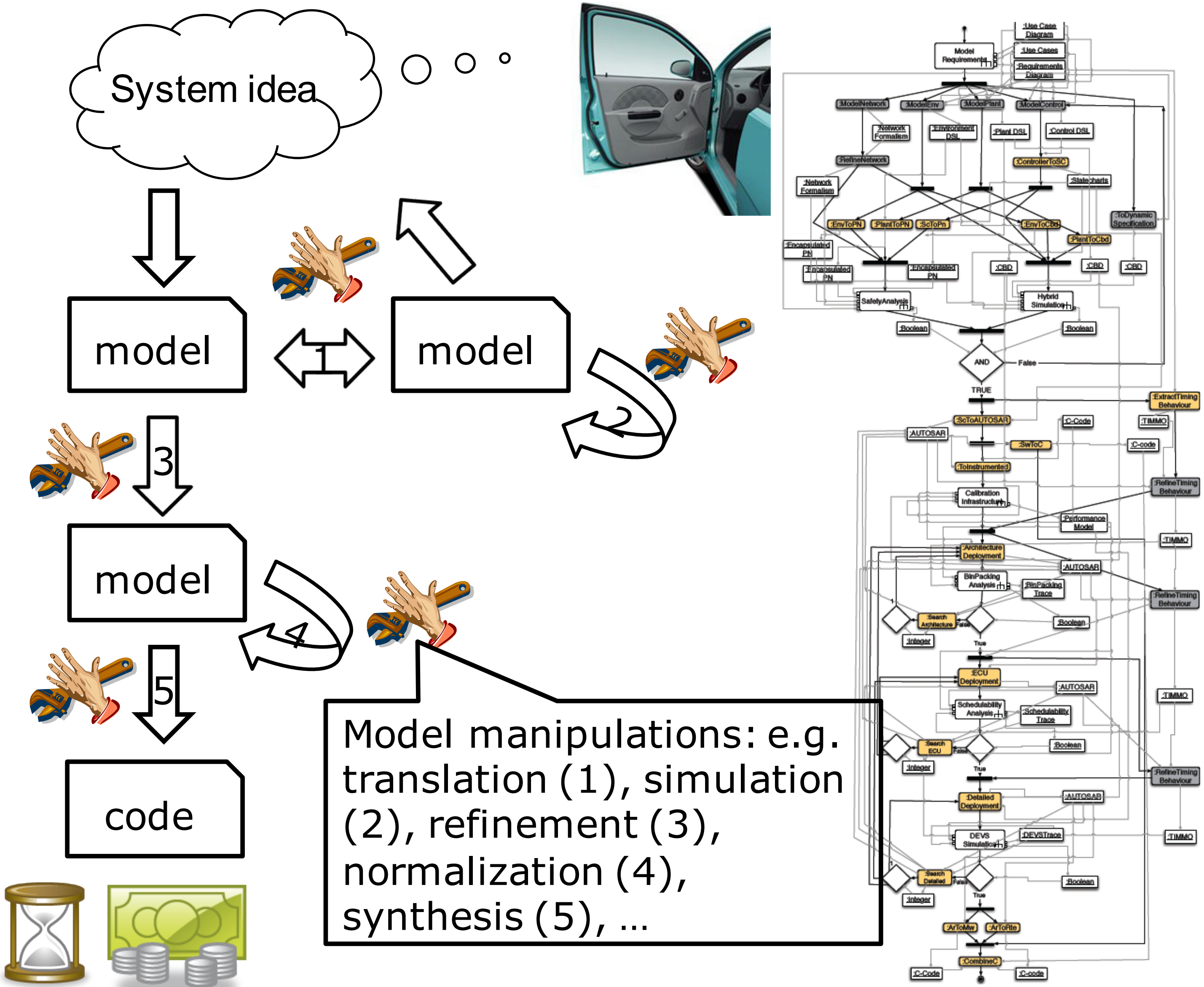
1. **Inductive Invariant Checking for Graph Transformation Systems**
2. **Applications**

   Cyber-Physical Systems & Safety

   Model Transformations & Correctness
3. **Summary & Open Challenges**

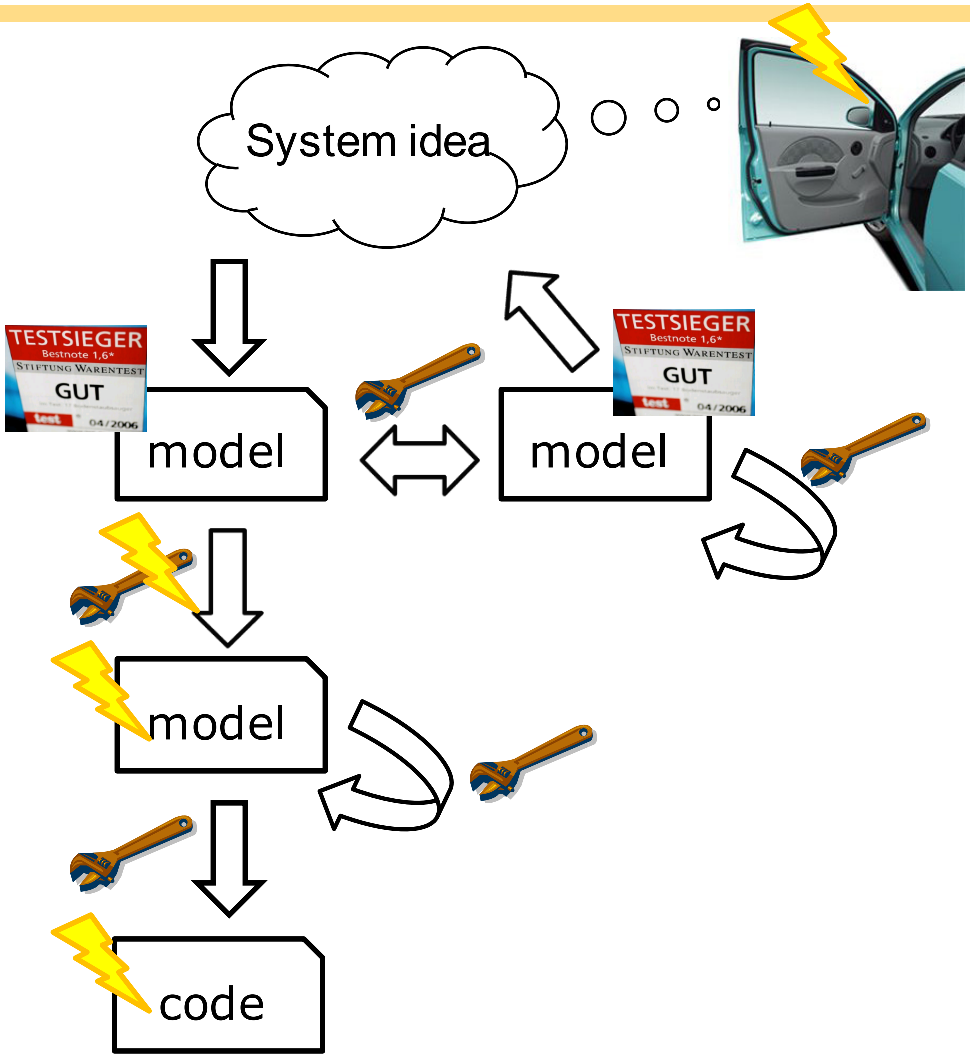
Why Model Transformations?

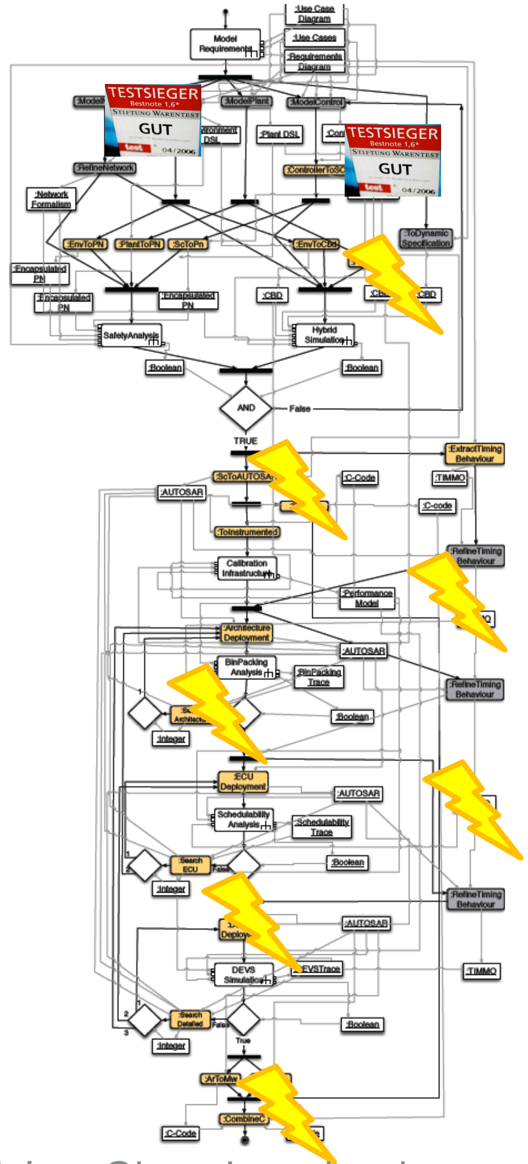
„The Heart and Soul of Model-Driven Engineering“ [SK03]



[Lucio+14]

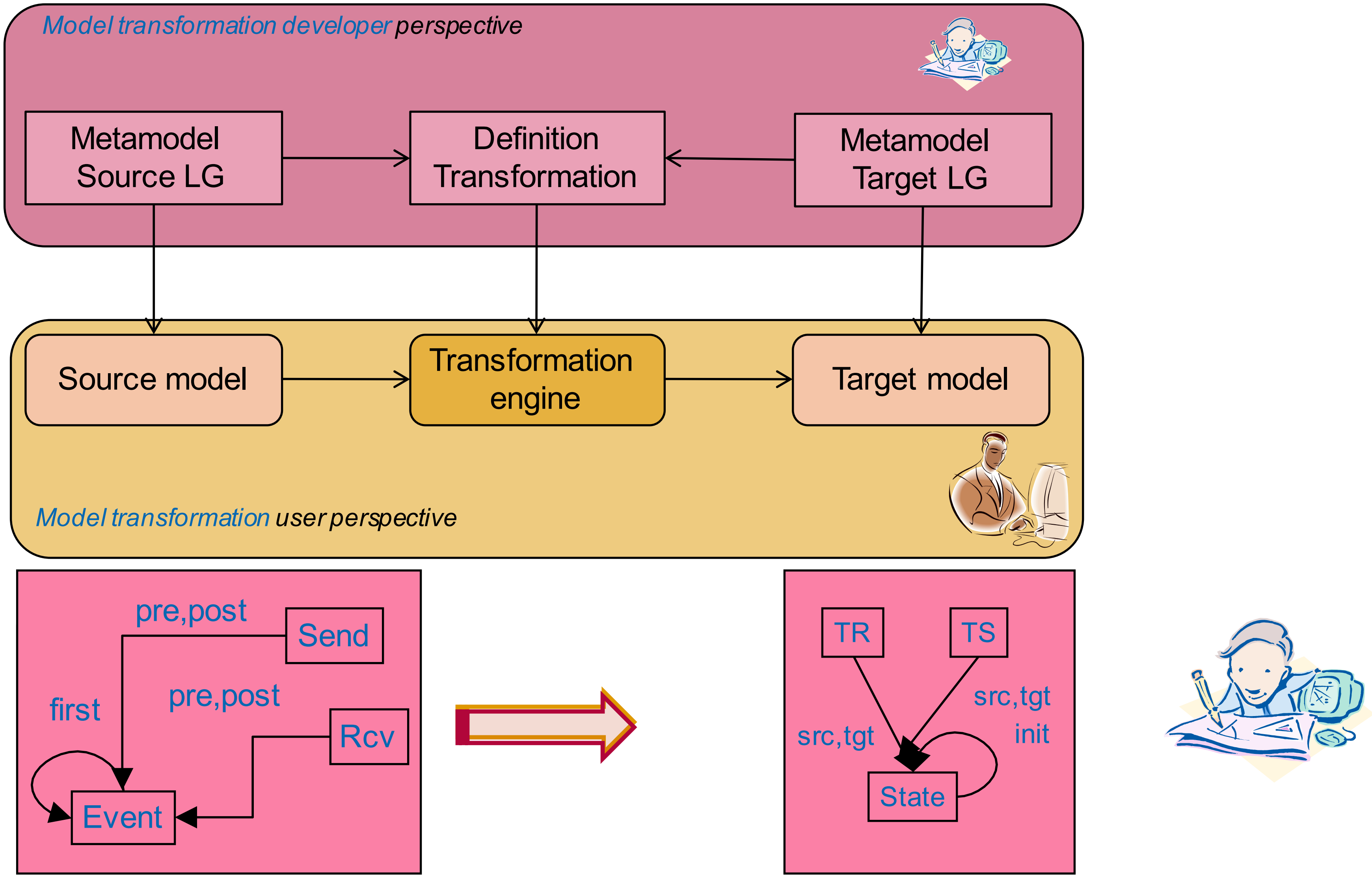
Correct Model Transformations



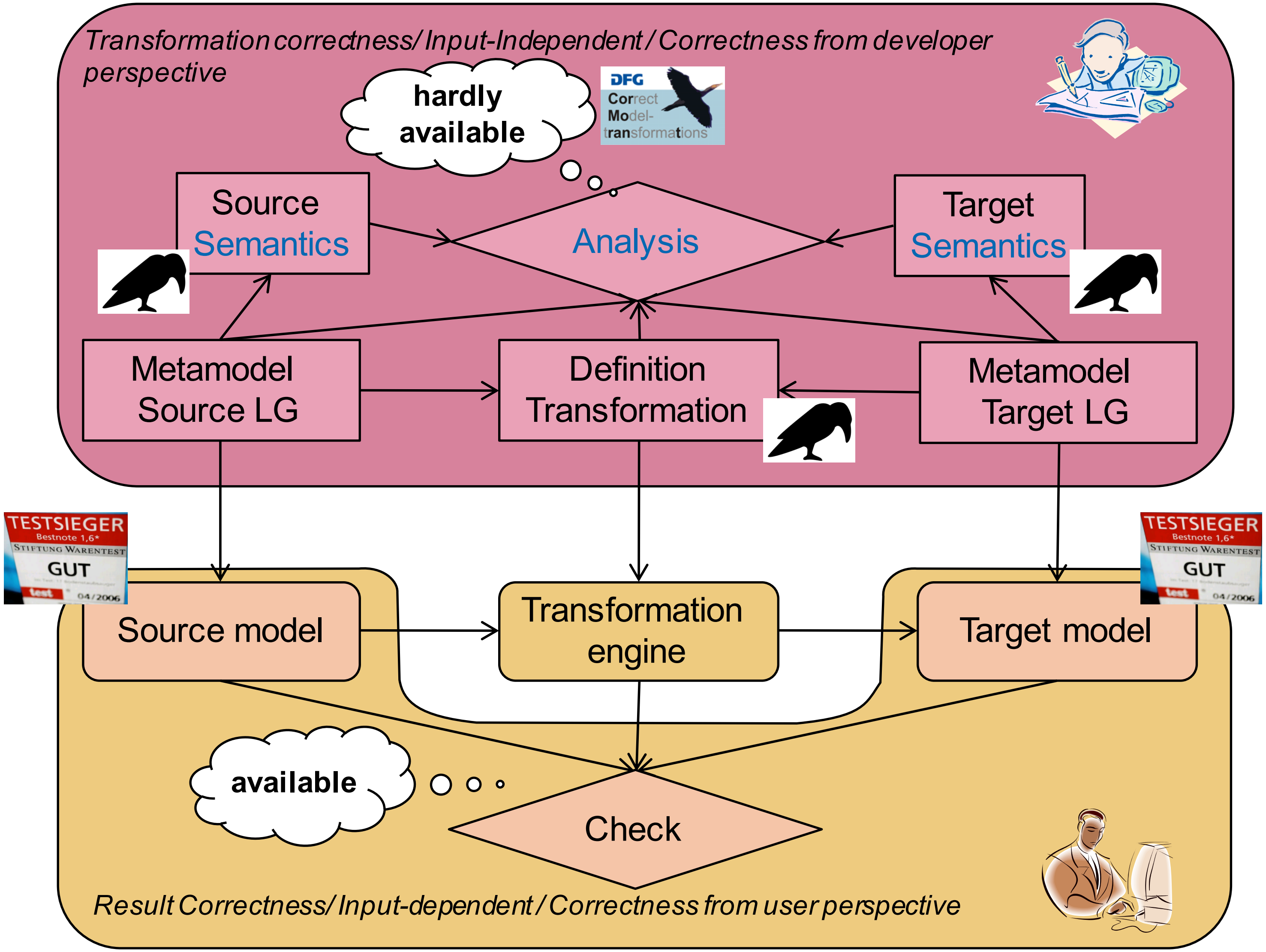


Model Transformations in a Nutshell

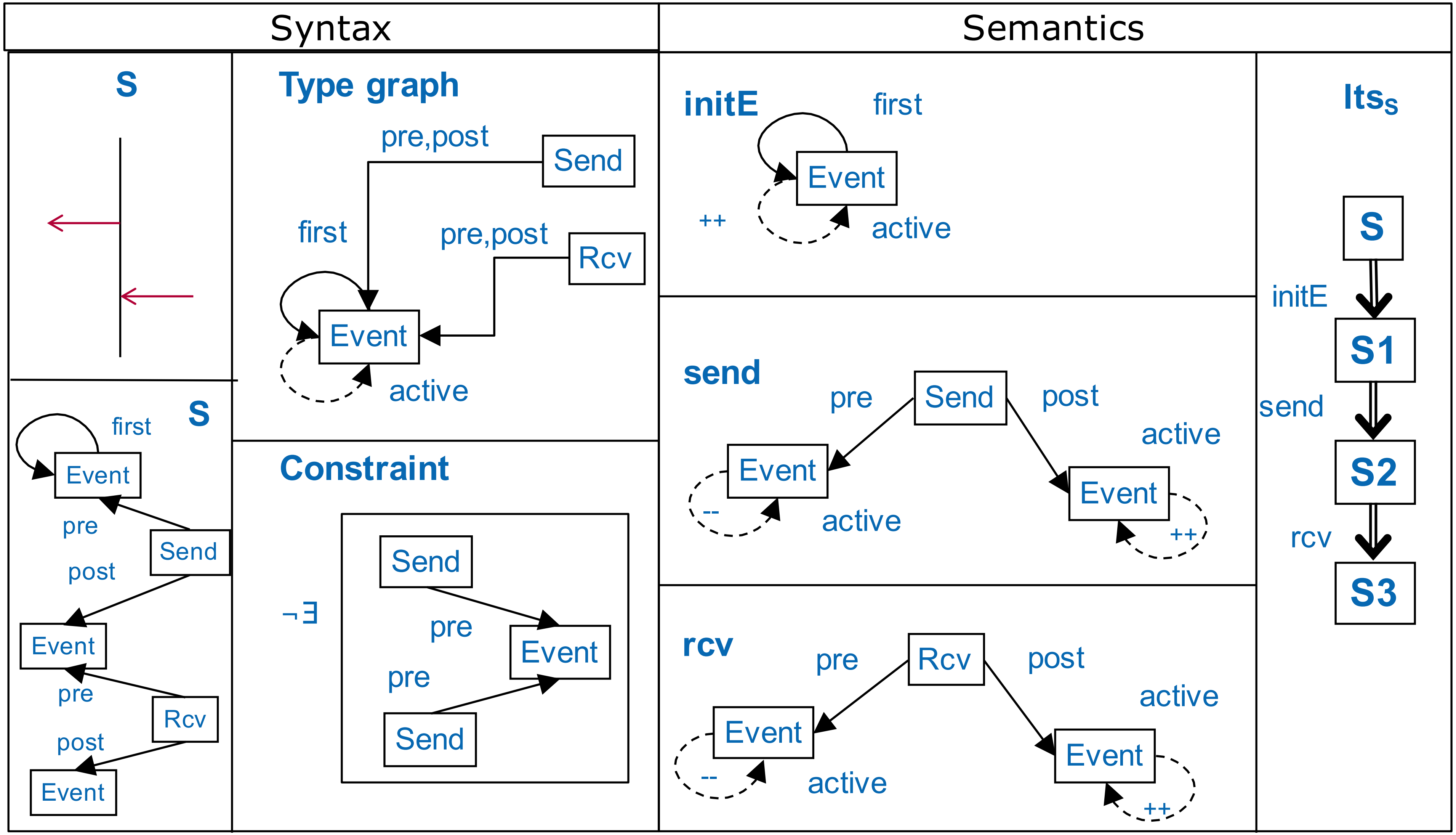
Hasso Plattner Institut



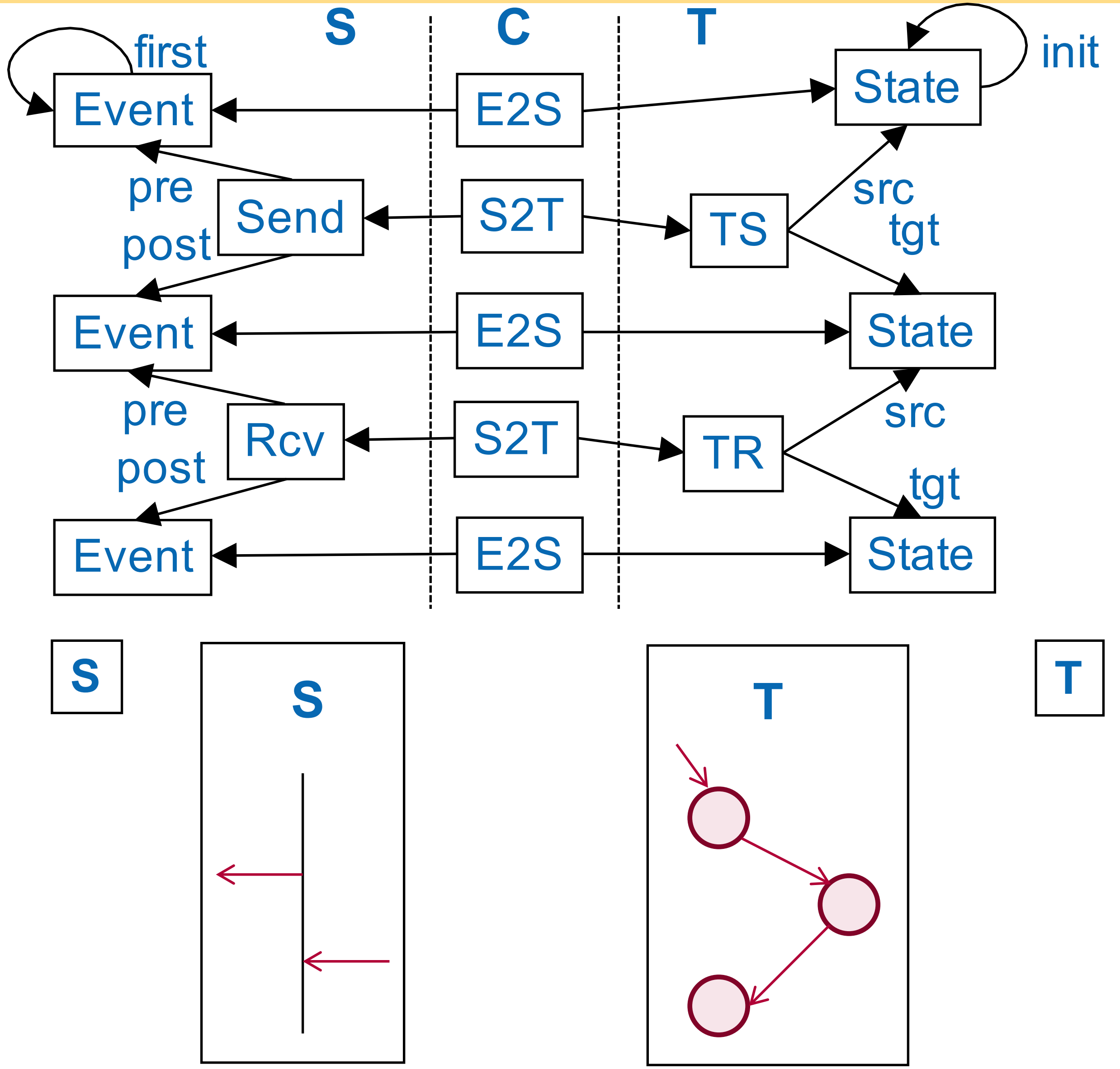
Correct Model Transformations



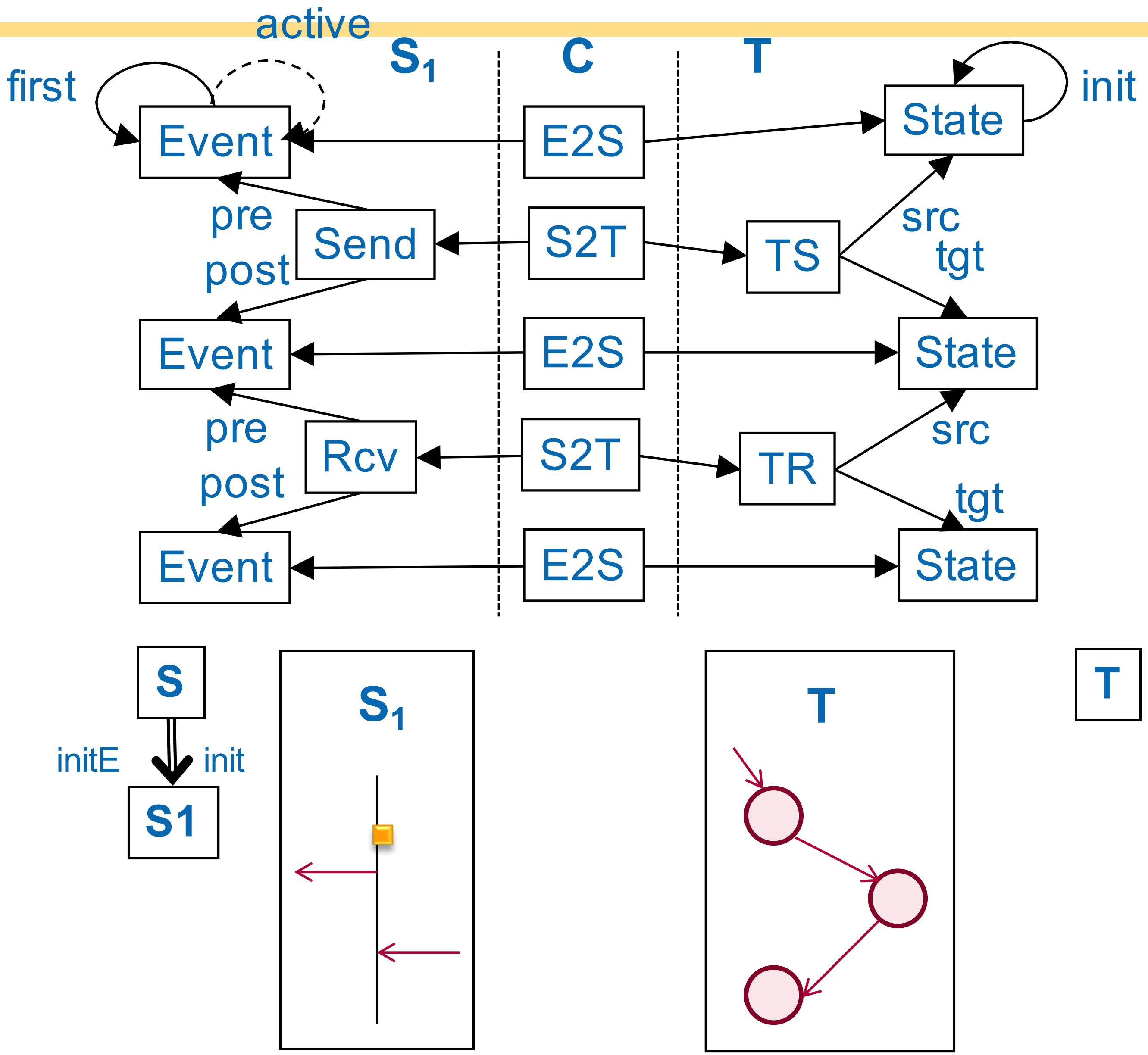
Behavior Definition using Graph Transformation



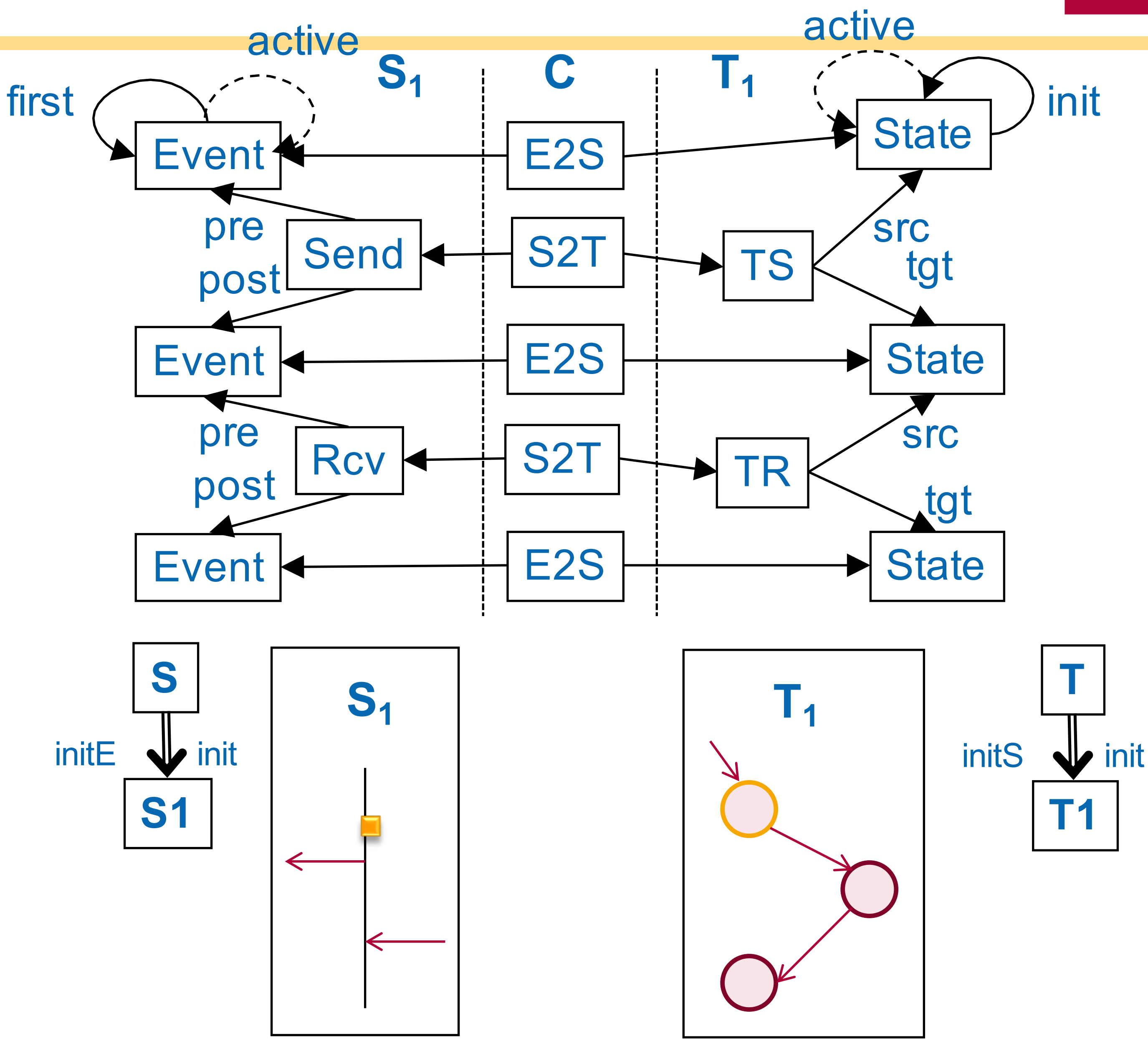
Behavior Preservation via Bisimulation



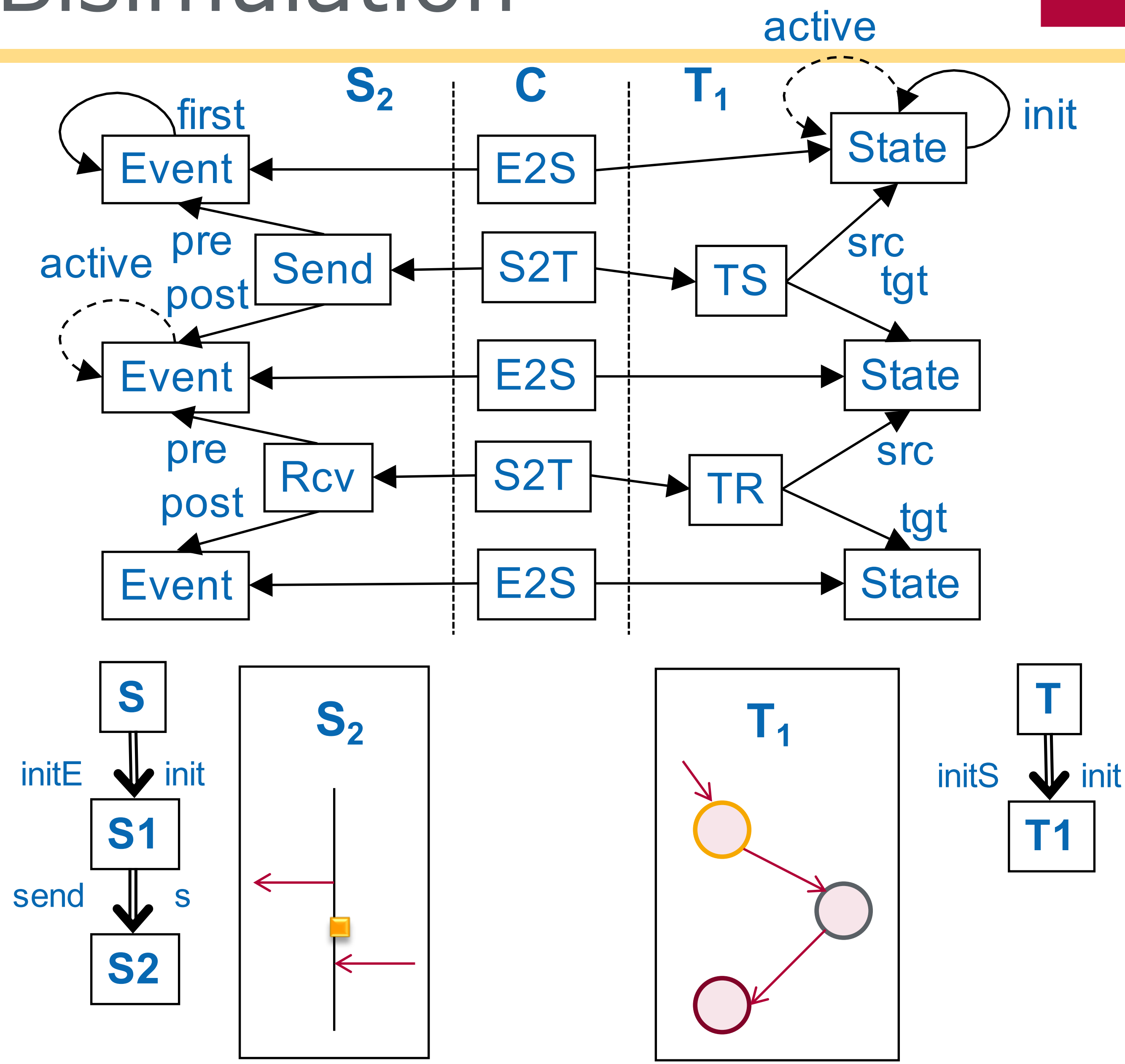
Behavior Preservation via Bisimulation



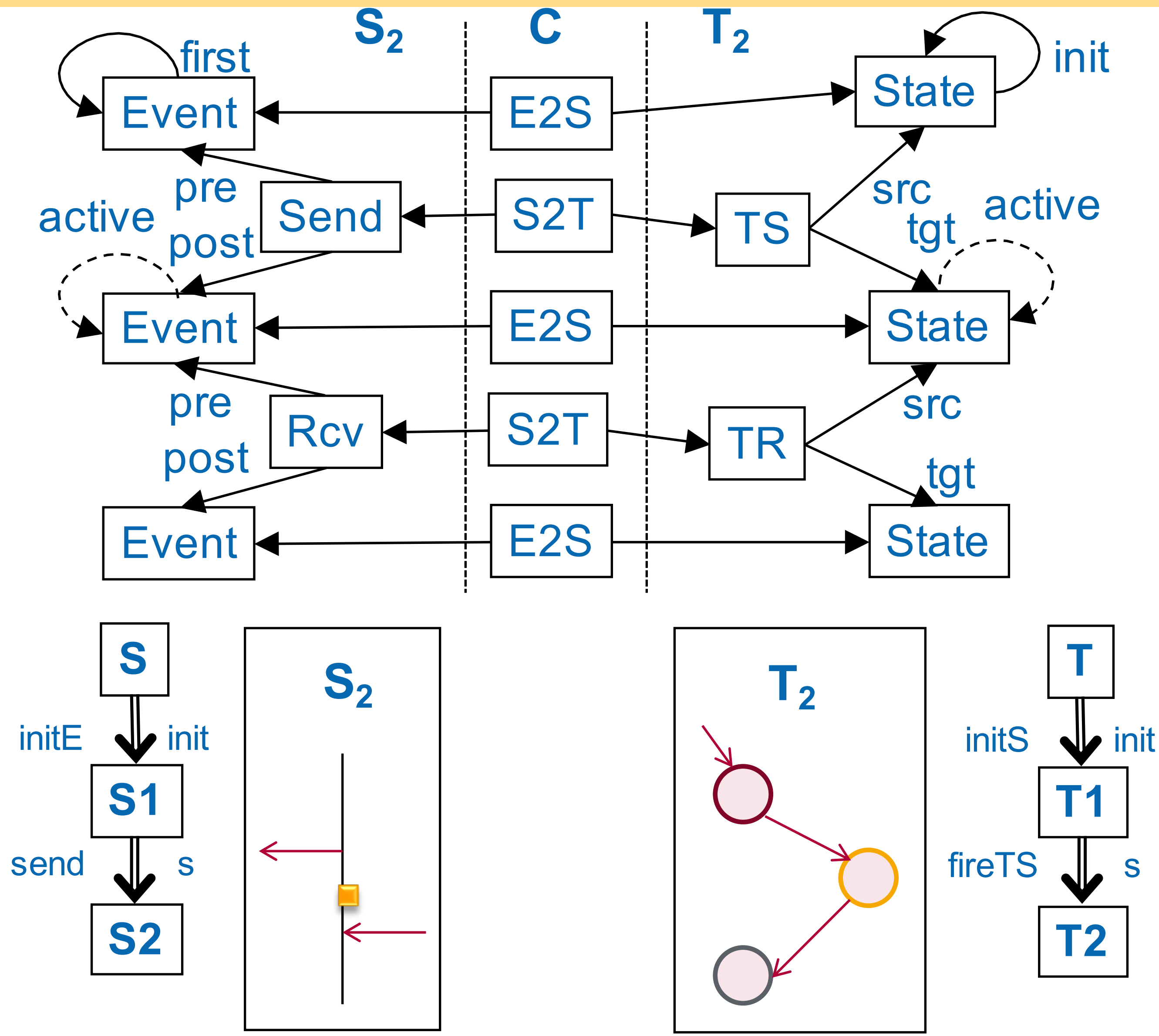
Behavior Preservation via Bisimulation



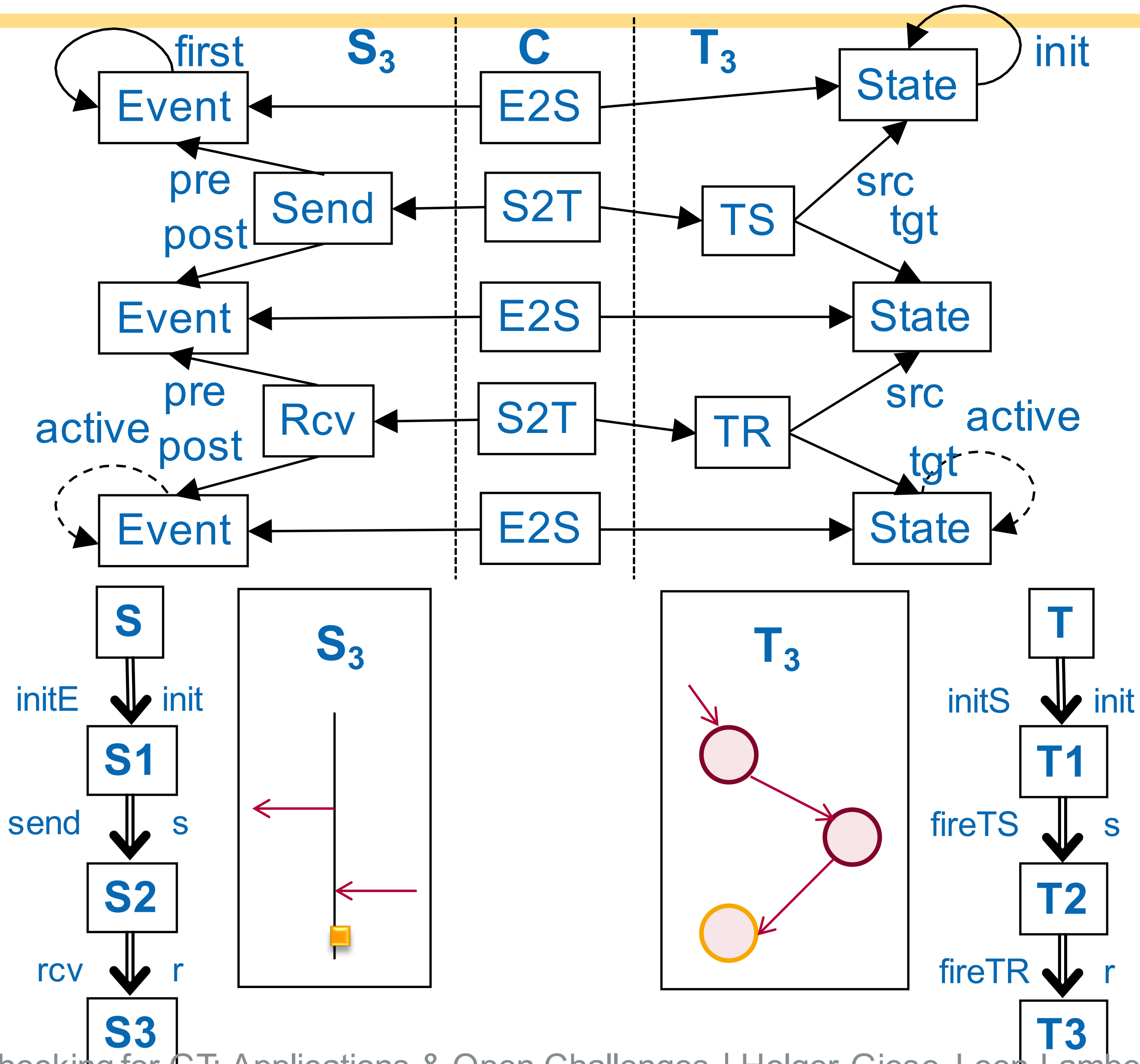
Behavior Preservation via Bisimulation



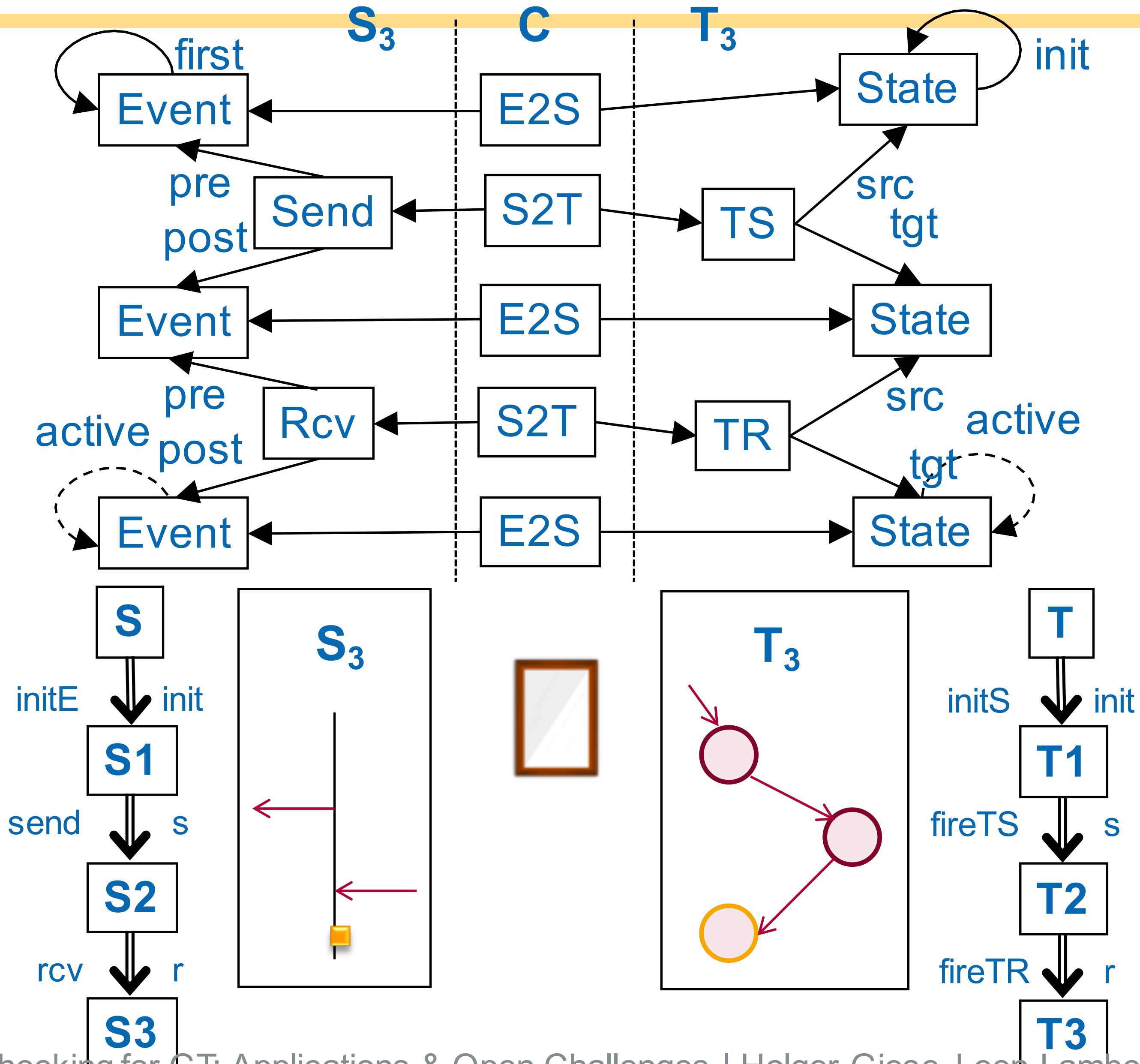
Behavior Preservation via Bisimulation



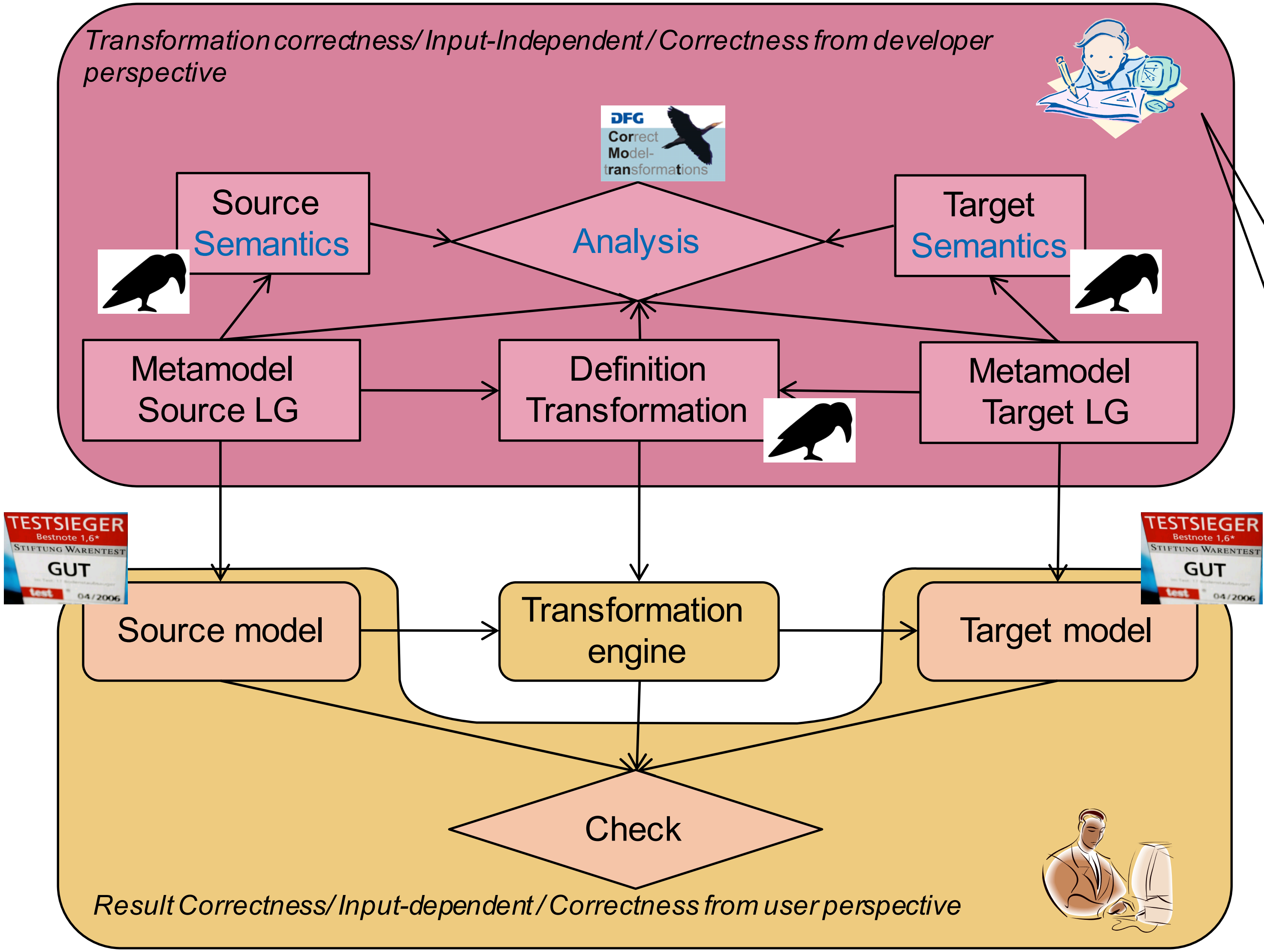
Behavior Preservation via Bisimulation



Behavior Preservation via Bisimulation



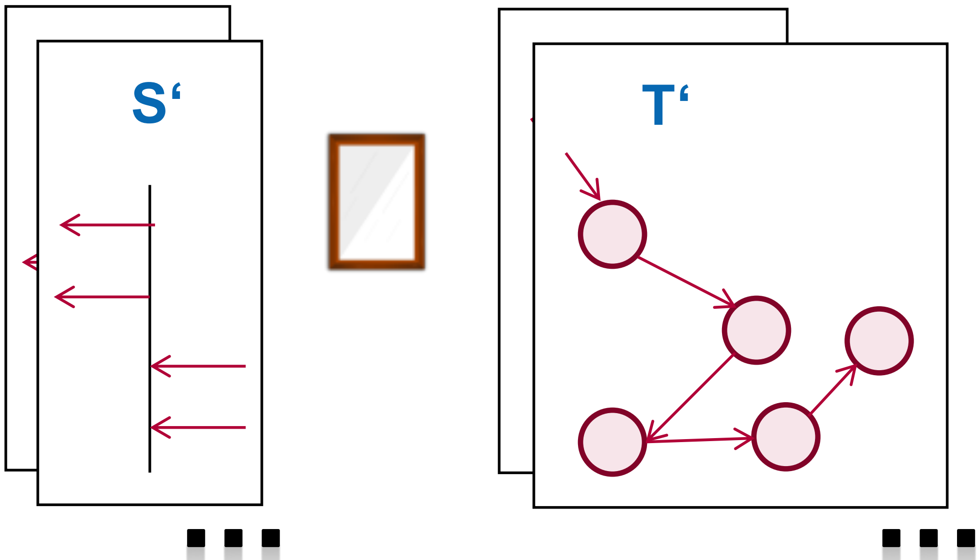
Behavior Preservation via Bisimulation



LTSs of **each** source and target model of the MT bisimilar?

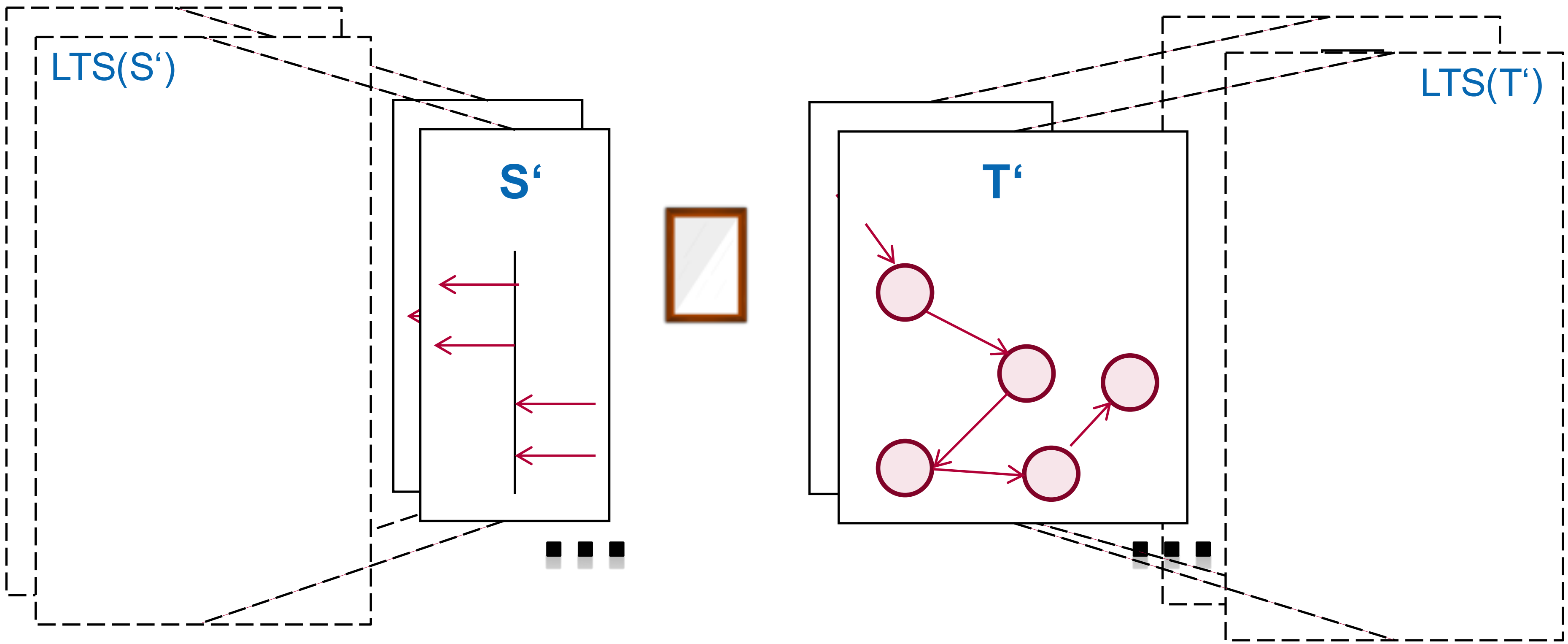
Behavior Preservation
Two-Step Algorithm

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



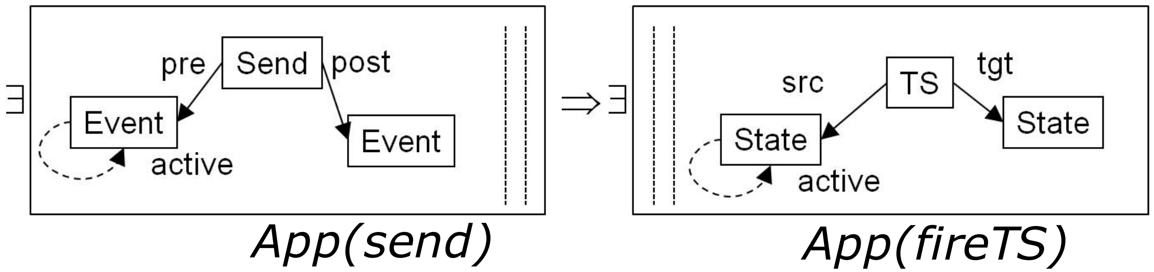
Behavior Preservation Two-Step Algorithm

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**

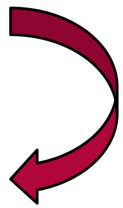


Behavior Preservation Two-Step Algorithm

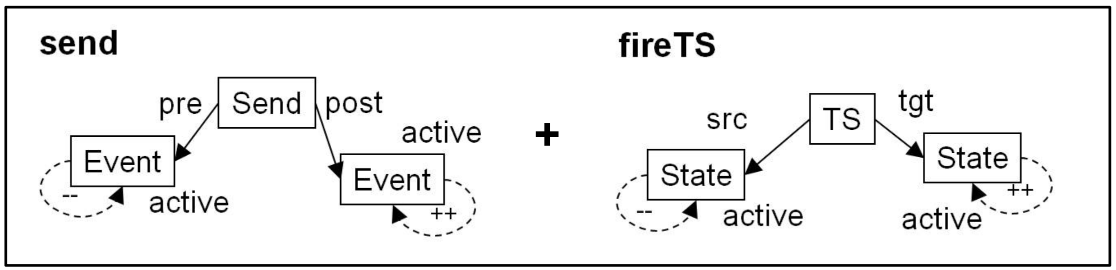
1. MT definition only allows source and target models that are in the bisimulation relation
2. **Behavior definitions of source and target modeling languages preserve bisimulation relation**



Bisimulation constraint

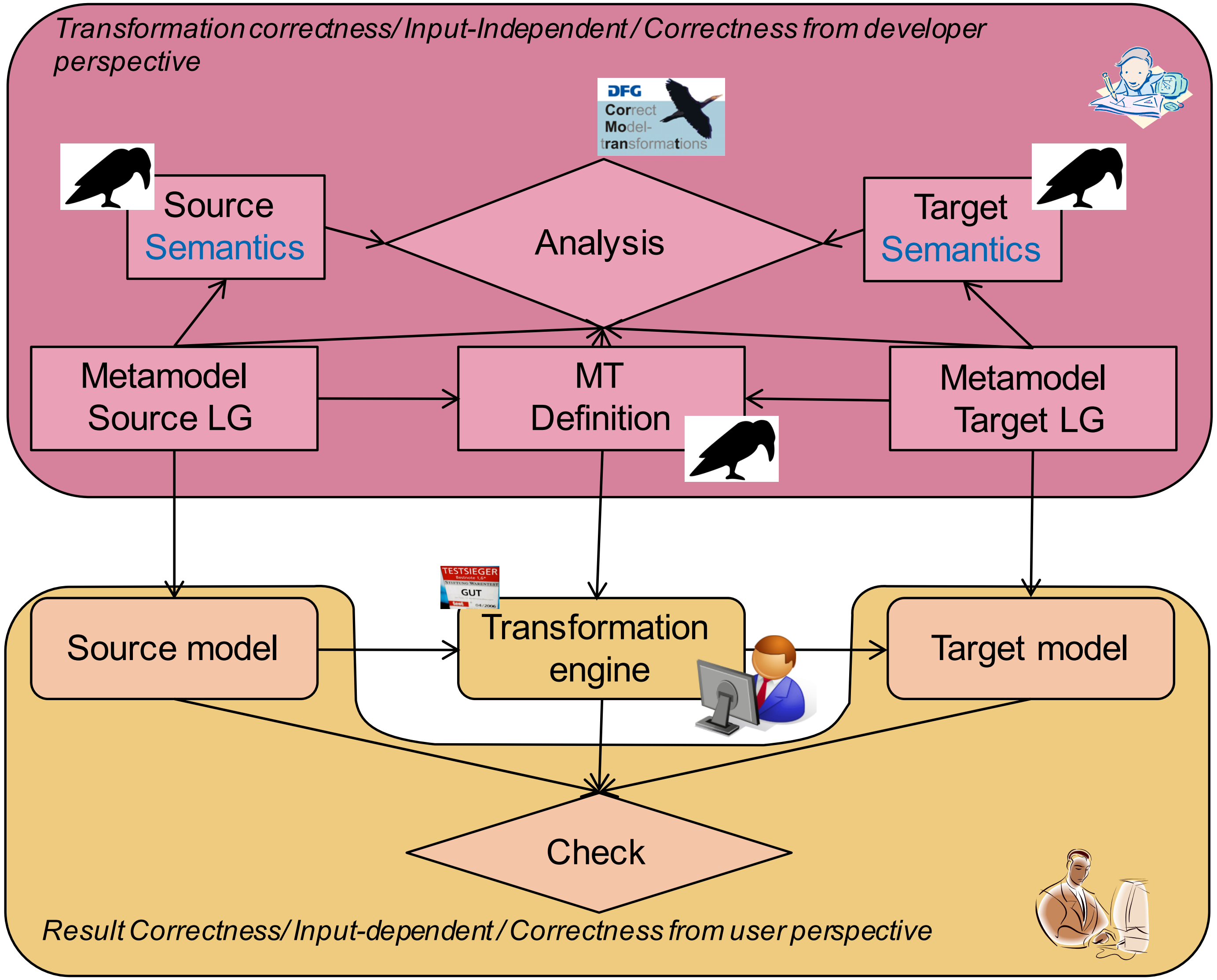


invariant



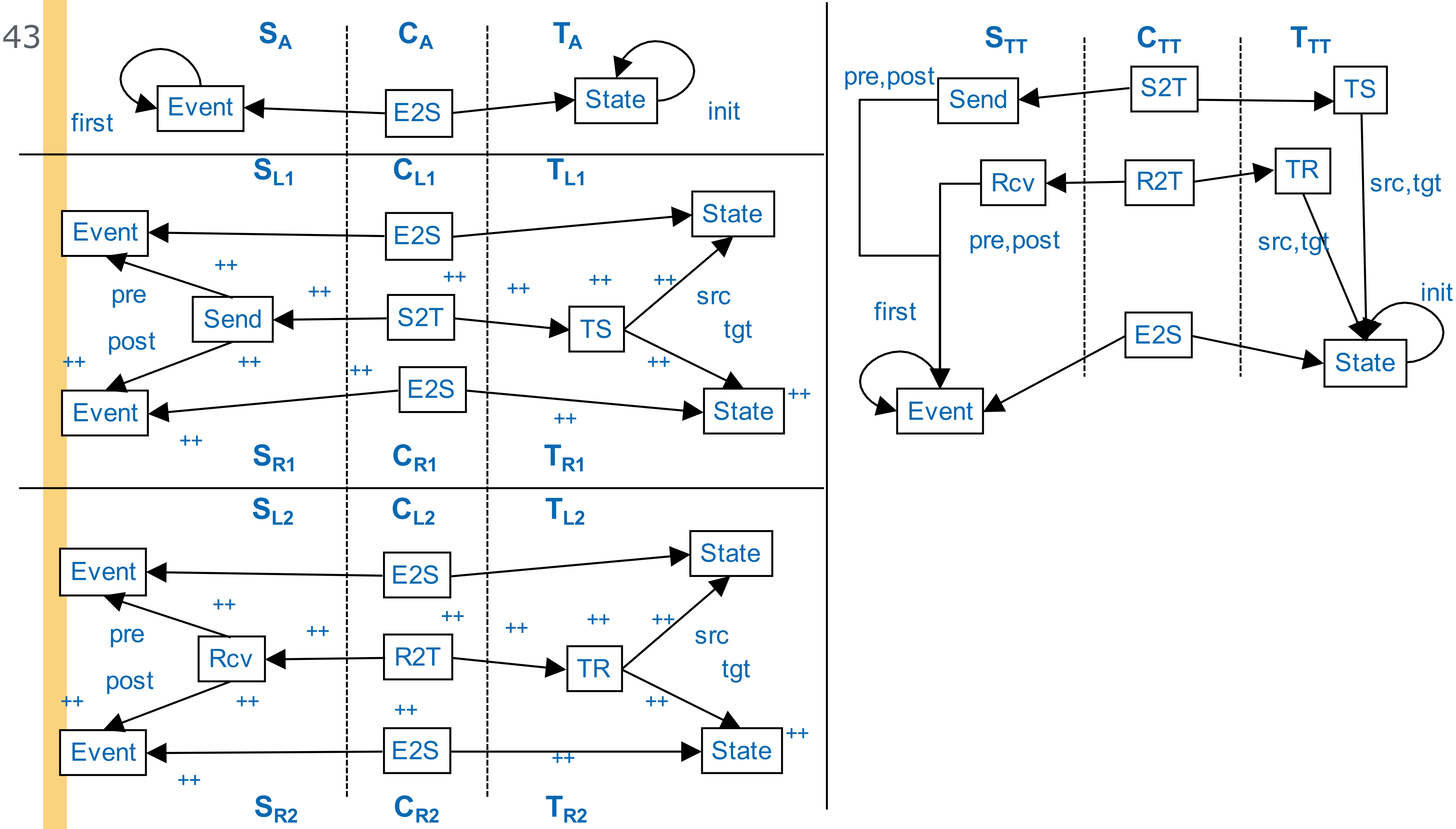
Parallel application source + target rules

Behavior Preservation

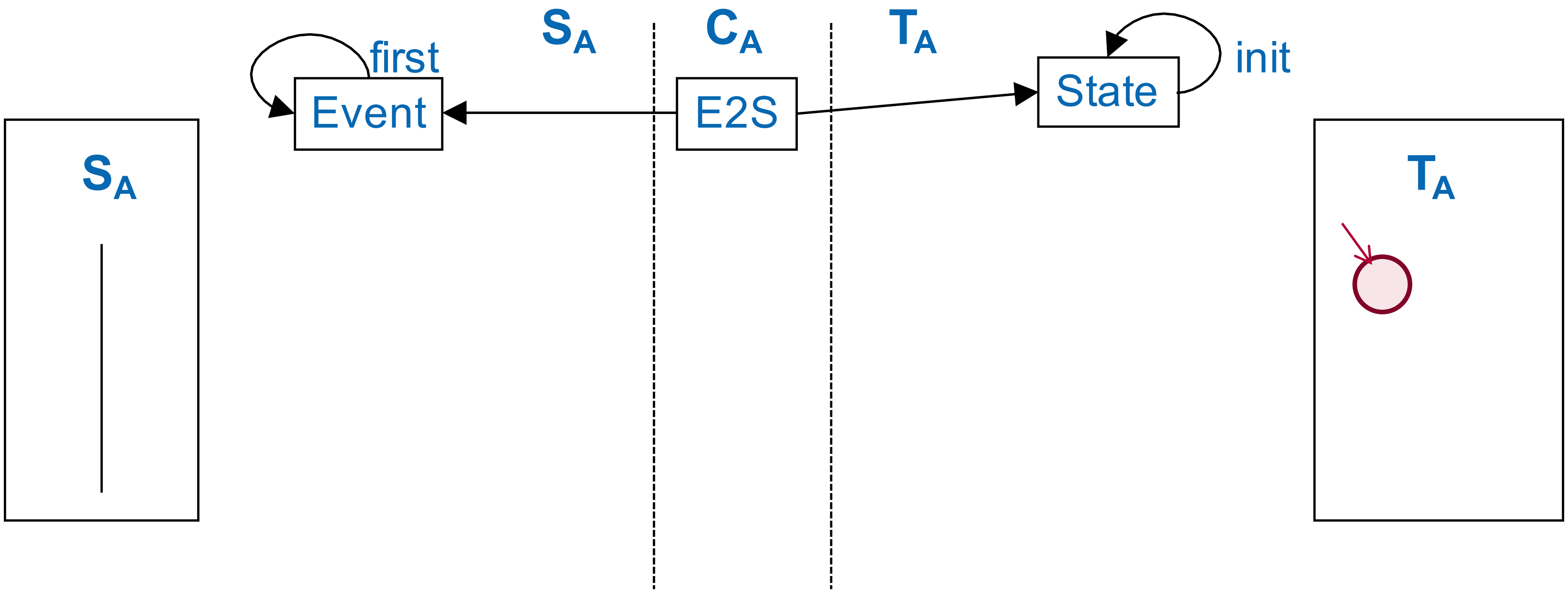


MT languages can be *operational* (e.g. QVT Operational), *relational* (e.g. TGGs, QVT Relational) or *hybrid* (e.g. ATL)

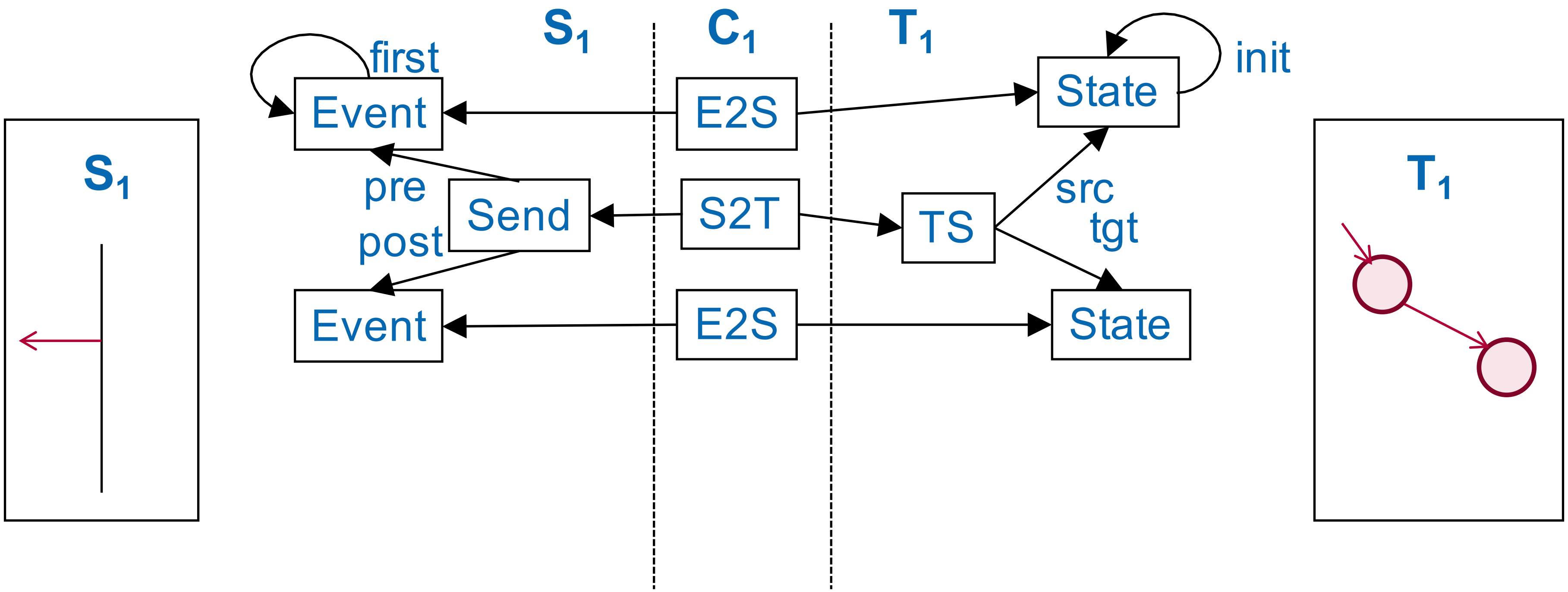
Triple Graph Grammar = Relational Specification



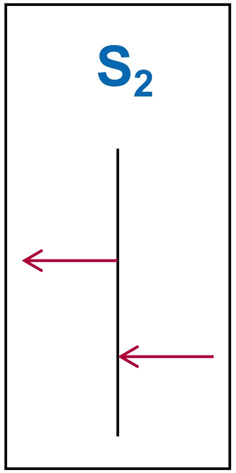
Triple Graph Grammar = Relational Specification

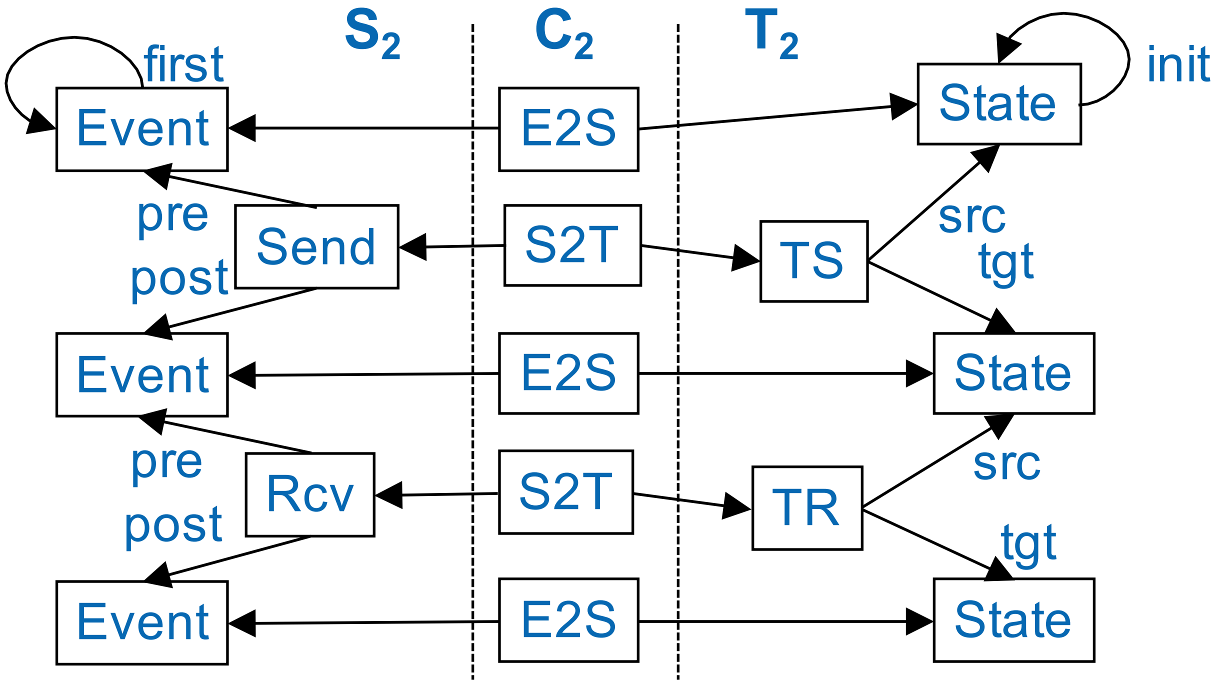


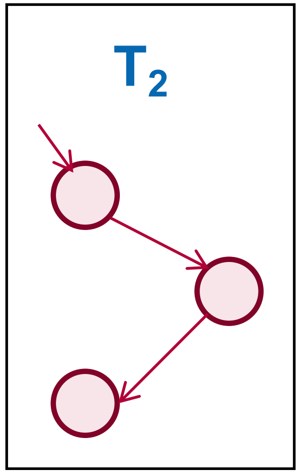
Triple Graph Grammar = Relational Specification



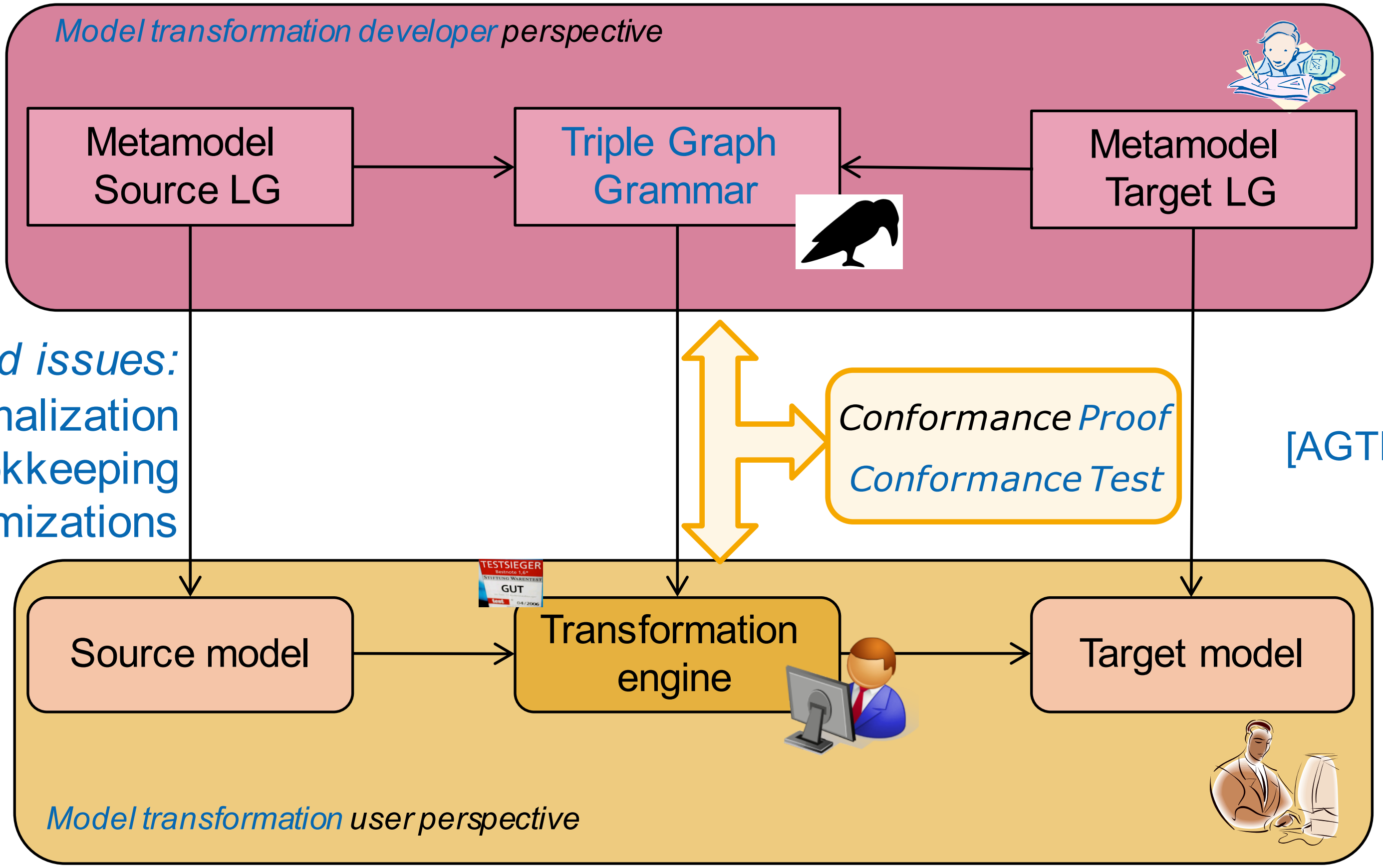
Triple Graph Grammar = Relational Specification







Conformance Checking Relational Specification ~ Implementation



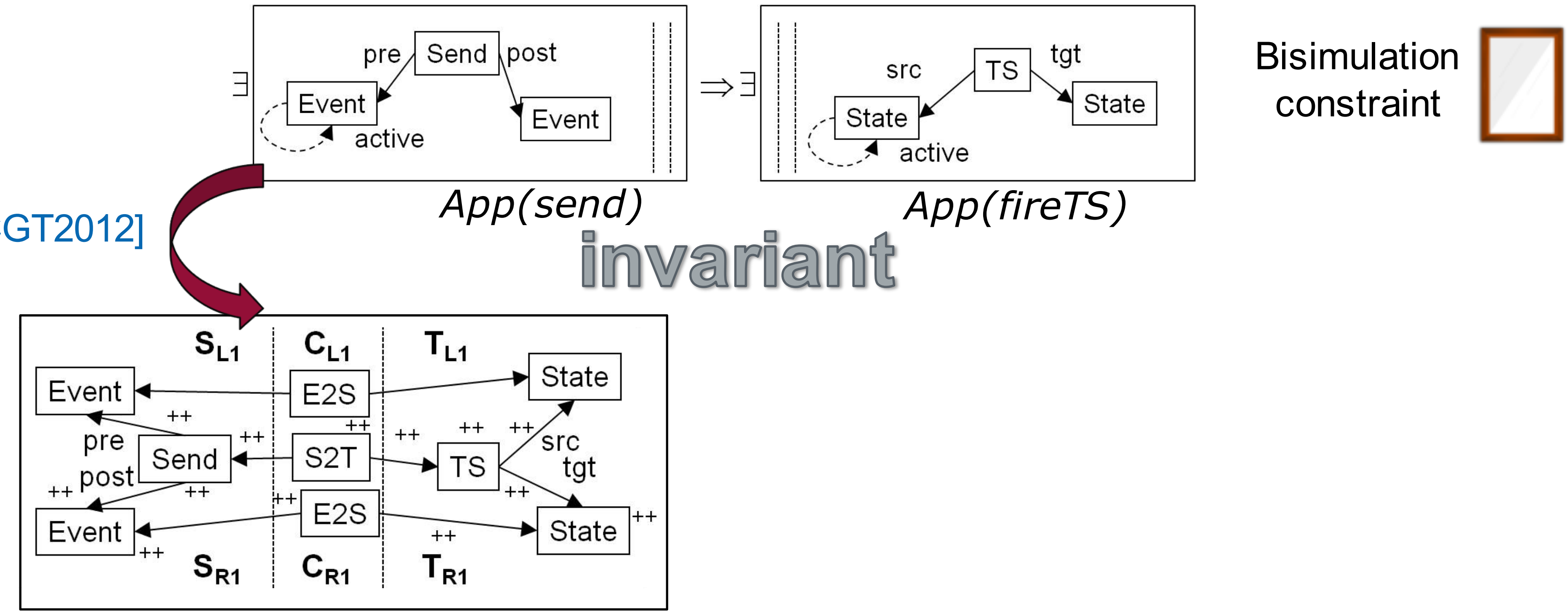
Advanced issues:
- TGG formalization
- Bookkeeping
- Optimizations

[AGTIVE11, SoSyM12]

Behavior Preservation Two-Step Algorithm

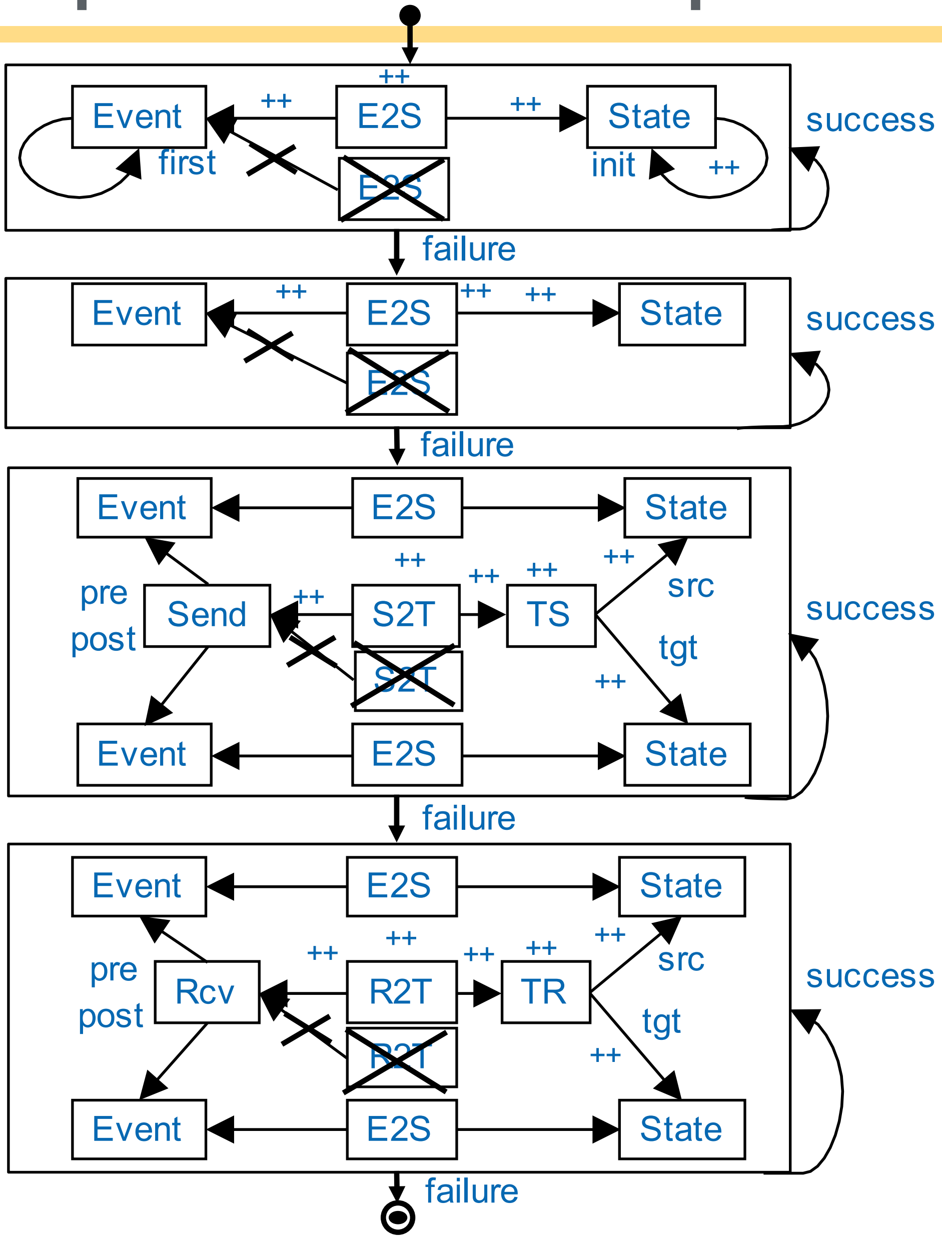
1. **MT definition only allows source and target models that are in the bisimulation relation**
   - **Relational definition (TGGs)** vs. Operational definition (SDs)

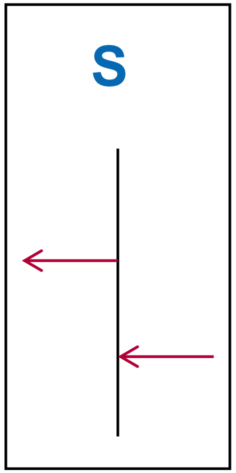
[ICGT2012]

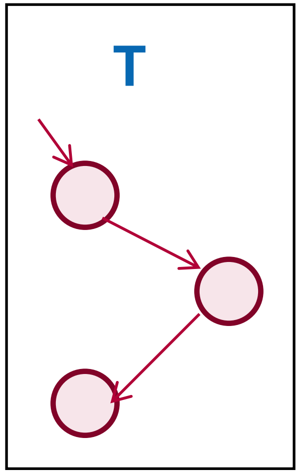


TGG rules

Story Diagrams Operational Specification



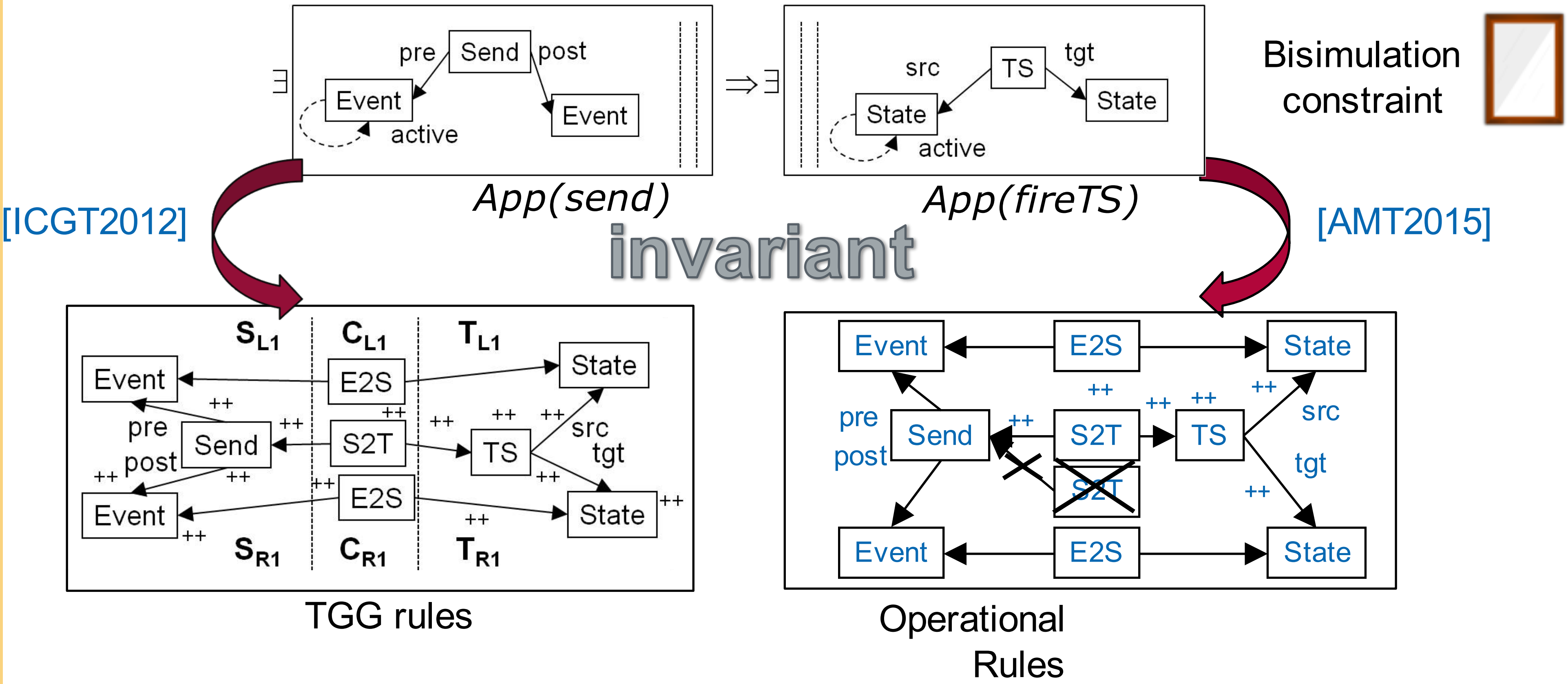




Behavior Preservation
Two-Step Algorithm

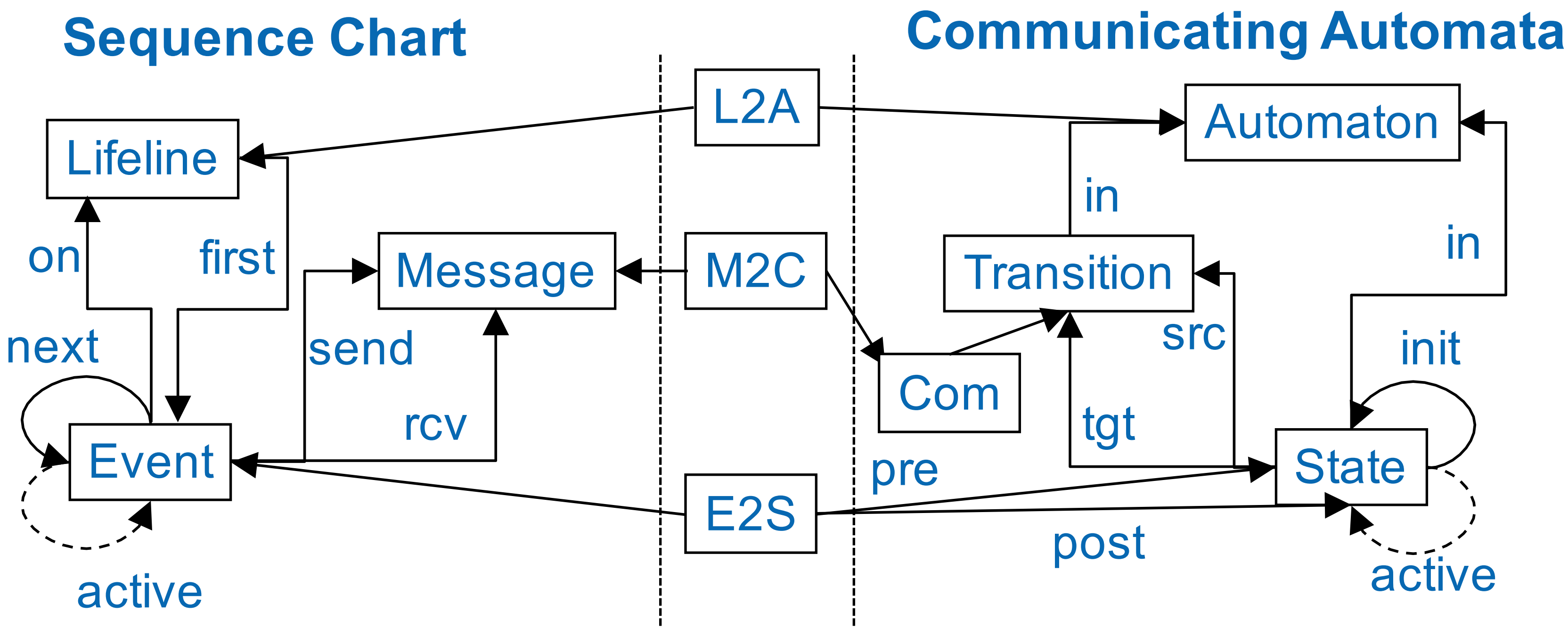
1. MT definition only allows source and target models that are in the bisimulation relation

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



Behavior Preservation More Advanced

- Addressing non-deterministic behavioral rule application
  - Case Study Sequence Charts 2 Communicating Automata
- Addressing not only behavioral equivalence, but also e.g. behavioral refinement



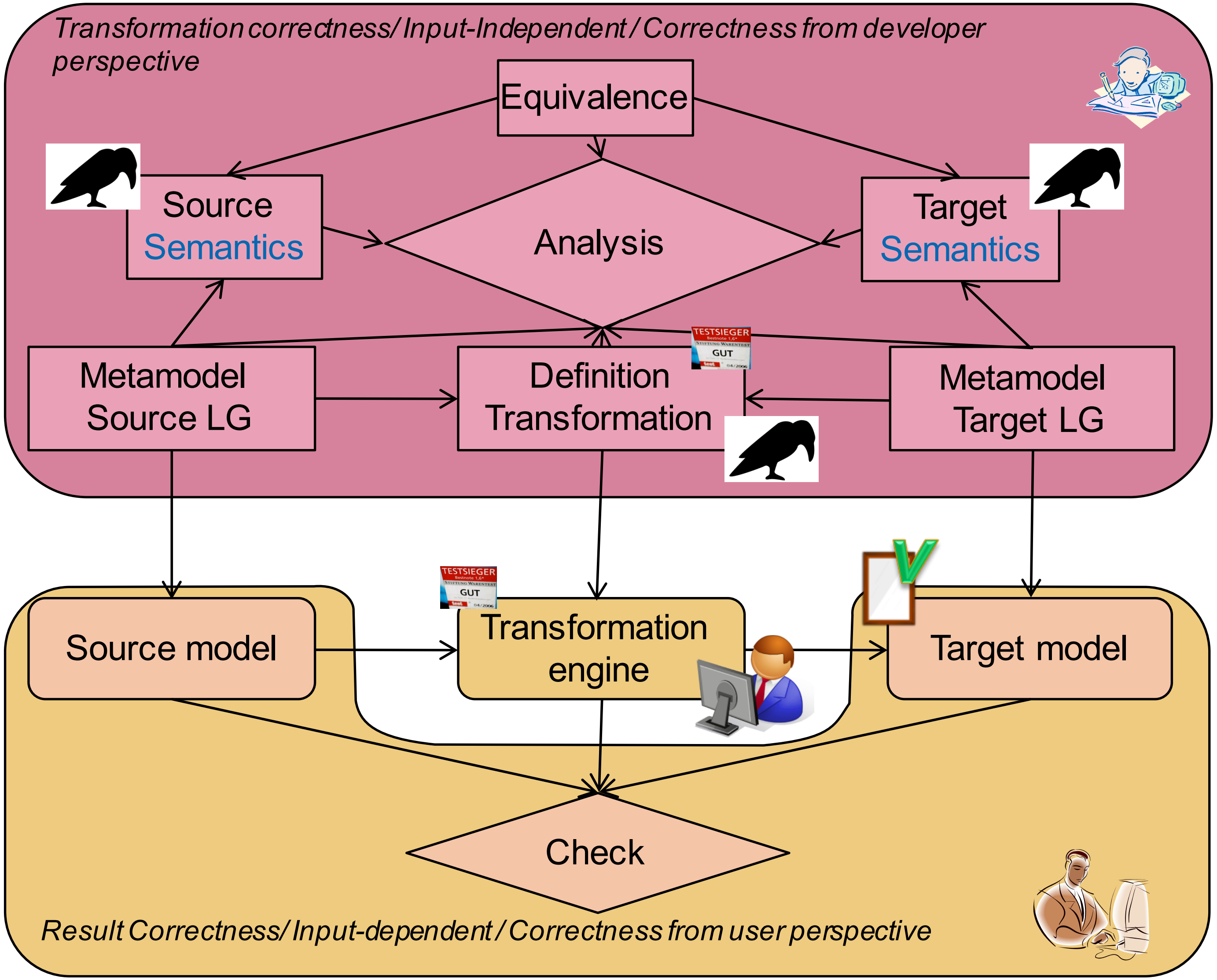
Tool Support

Hasso Plattner Institut

[ICGT2015]

					without advanced implication check				with advanced implication check			
	Characteristics		SeekSat/ProCon		Invcheck-total		Invcheck-partial		Invcheck-total/imp		nvcheck-partial/imp	
Example	Check	Complexity	time (s)	result	time (s)	result	time (s)	result	time (s)	result	time (s)	result
MT - Simple - Semantics	subproblem	4	20	true	<1	true	<1	true	<1	true	<1	true
MT - Simple - Semantics	subproblem	4	20	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - TGG	subproblem	4	<1	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - TGG	subproblem	4	<1	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - Semantic	subproblem	5	10	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - Semantic	subproblem	5	9	true	<1	true	<1	true	<1	true	<1	true
MT - Simple - Semantics	subproblem	11	40	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - TGG	subproblem	11	<1	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - Semantic	subproblem	12		out of memory		timeout	<1	false negatives		timeout	<1	true
MT - Complex - Semantic	subproblem	17	17	true	<1	false negatives	<1	false negatives	<1	true	<1	true
MT - Complex - TGG	subproblem	20		timeout	<1	true	<1	true	<1	true	<1	true
MT - Simple - Semantics	subproblem	30	20	true	<1	true	<1	true	<1	true	<1	true
MT - Simple - Semantics	subproblem	70	40	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - Semantic	subproblem	72		timeout	<1	false negatives	1	false negatives	1,5	true	1,5	true
MT - Simple - Semantics	subproblem	78	6,5	true	<1	true	<1	true	<1	true	<1	true
MT - Complex - Semantic	subproblem	188		out of memory	1,5	false negatives	2,5	false negatives	<1	true	<1	true
Car Platooning	subproblem	258	<1	true	<1	true	<1	true	<1	true	<1	true
Car Platooning	subproblem	610	<1	true	<1	true	<1	true	<1	true	<1	true
MT - Simple - Semantics	subproblem	807		timeout	<1	true	<1	true	<1	true	<1	true
Car Platooning	complete	947	<1	false	<1	false negatives	<1	false negatives	3	false	3	false
MT - Simple - TGG	subproblem	2778	220	true	1,5	false negatives	1	false negatives	1,5	true	1	true
MT - Simple - TGG	subproblem	2778	226	true	1,25	false negatives	1	false negatives	1,25	true	1	true
MT - Simple - Semantics	complete	3870		timeout	1,5	true	1	true	1,5	true	1	true
MT - Simple - TGG	complete	5556	562	true	2	false negatives	2	false negatives	2,25	true	1,75	true
MT - Complex - Semantic	subproblem	607312		out of memory		timeout	90	false negatives		timeout	<1	true
MT - Complex - Semantic	complete	607500		out of memory		timeout	95	false negatives		timeout	<1	true
MT - Complex - TGG	complete	1817622		timeout		timeout	~100min	true		timeout	~50min	true

Summary
Correct Model Transformations



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 hazarduous 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.

Open Challenges

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

[SK03] Sendall, S. & Kozaczynski, W. : Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software, 2003 , 20 , 42-45.

[ICSE2006] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein and Daniela Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In Proc. of the 28th International Conference on Software Engineering (ICSE), Shanghai, China, ACM Press, 2006.

[ISORC2008] Basil Becker and Holger Giese. On Safe Service-Oriented Real-Time Coordination for Autonomous Vehicles. In In Proc. of 11th International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC), Pages 203--210, IEEE Computer Society Press, 5-7 May 2008.

[TR2012] Basil Becker and Holger Giese. Cyber-Physical Systems with Dynamic Structure: Towards Modeling and Verification of Inductive Invariants. Technical report, 64, Hasso Plattner Institute at the University of Potsdam, Germany, 2012.

[ICGT2012b] Christian Krause and Holger Giese. Probabilistic Graph Transformation Systems. In Proceedings of Intern. Conf. on Graph Transformation (ICGT' 12), Vol. 7562:311-325 of Lecture Notes in Computer Science, Springer-Verlag, 2012.

[Lucio+14] Levi Lúcio, Moussa Amrani, Juergen Dingel, Leen Lambers, Rick Salay, Gehan M.K.Selim, Eugene Syriani, Manuel Wimmer Model transformation intents and their properties Software and Systems Modeling, Online First, Springer, 2014

[SoSyM12] Holger Giese, Stephan Hildebrandt, and Leen Lambers. Bridging the gap between formal semantics and implementation of triple graph grammars - ensuring conformance of relational model transformation specifications and implementations. Software and Systems Modeling, pages 1–27, 2012.

[ICGT12] Holger Giese and Leen Lambers. Towards automatic verification of behavior preservation for model transformation via invariant checking. In Proceedings of International Conference on Graph Transformation (ICGT'12), volume 7562 of LNCS, pages 249–263. Springer, 2012.

[ICGT2015] Johannes Dyck and Holger Giese. Inductive Invariant Checking with Partial Negative Application Conditions. In Francesco Parisi-Presicce and Bernhard Westfechtel editors, Graph Transformation, Vol. 9151:237-253 of Lecture Notes in Computer Science, Springer International Publishing, 2015.

[AMT15] Johannes Dyck, Holger Giese, Leen Lambers, Sebastian Schlesinger, Sabine Glesner, Towards the Automatic Verification of Behavior Preservation at the Transformation Level for Operational Model Transformations in Fourth Workshop on the Analysis of Model Transformations, CEUR Workshop Proceedings, 2015