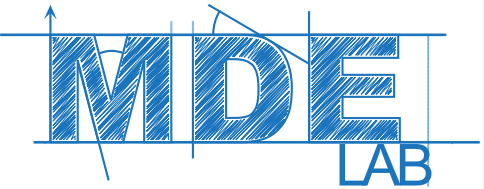
Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

The MDELab Tool Framework for the Development of Correct Model Transformations with Triple Graph Grammars

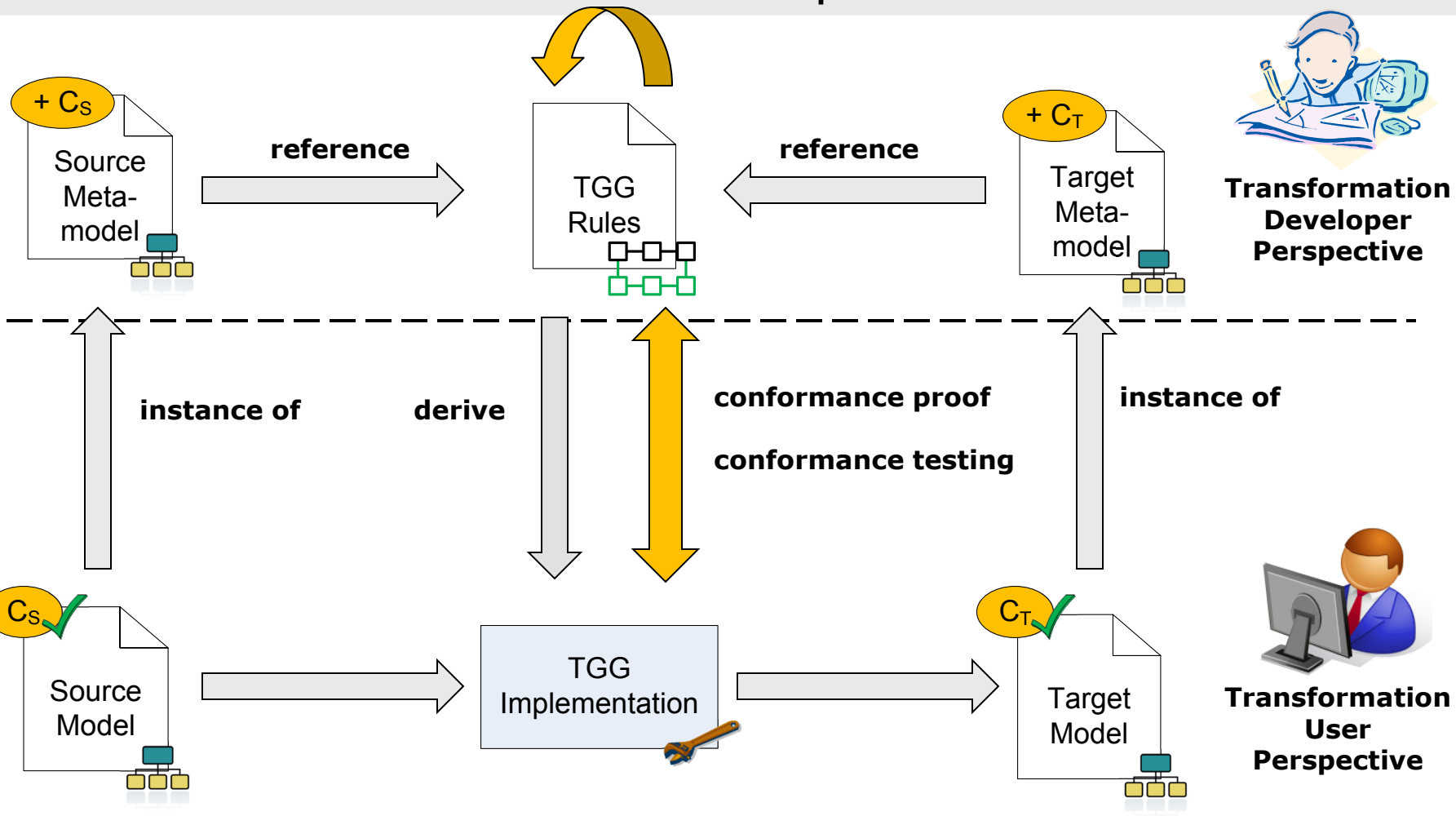
Stephan Hildebrandt, Leen Lambers, and Holger Giese



www.mdelab.de

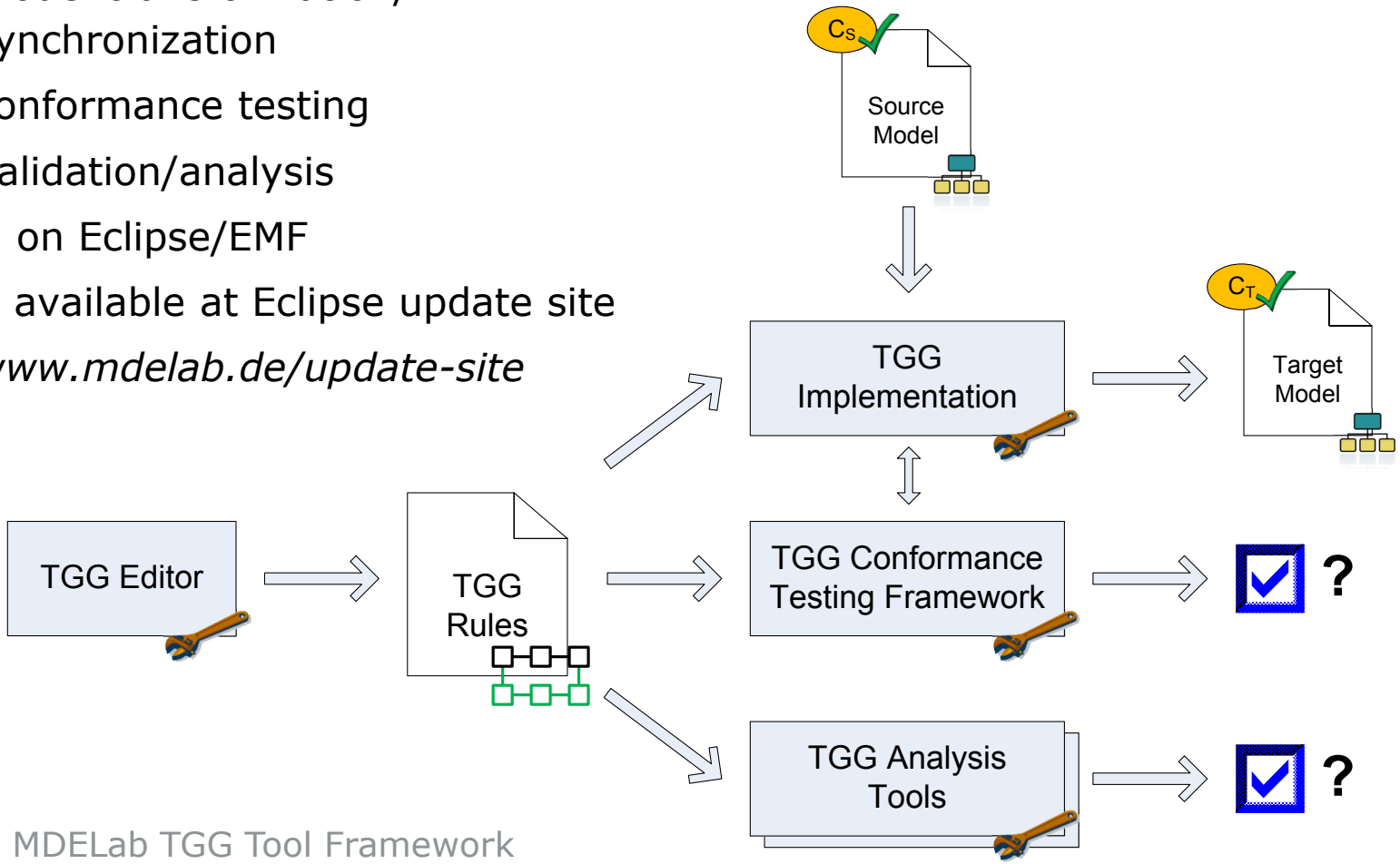
Overview

analyze
- **constraint validity**
- **behavior preservation**



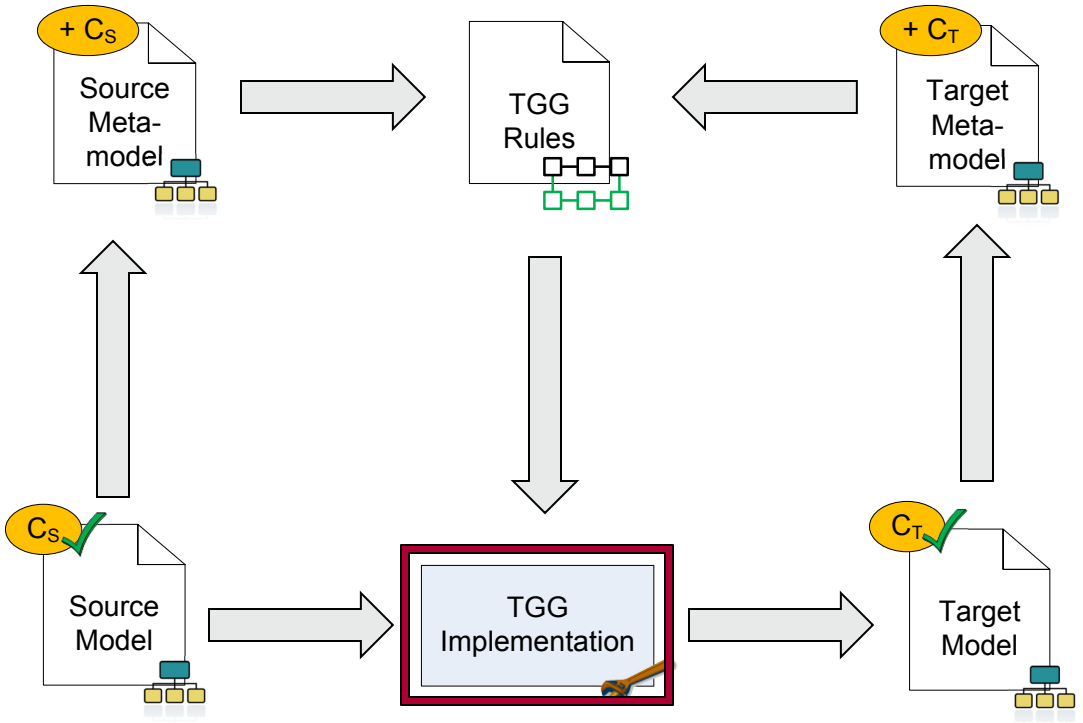
General Tool Architecture

- tool framework for TGG-based
  - model transformation/ synchronization
  - conformance testing
  - validation/analysis
- based on Eclipse/EMF
- partly available at Eclipse update site
  - *www.mdelab.de/update-site*

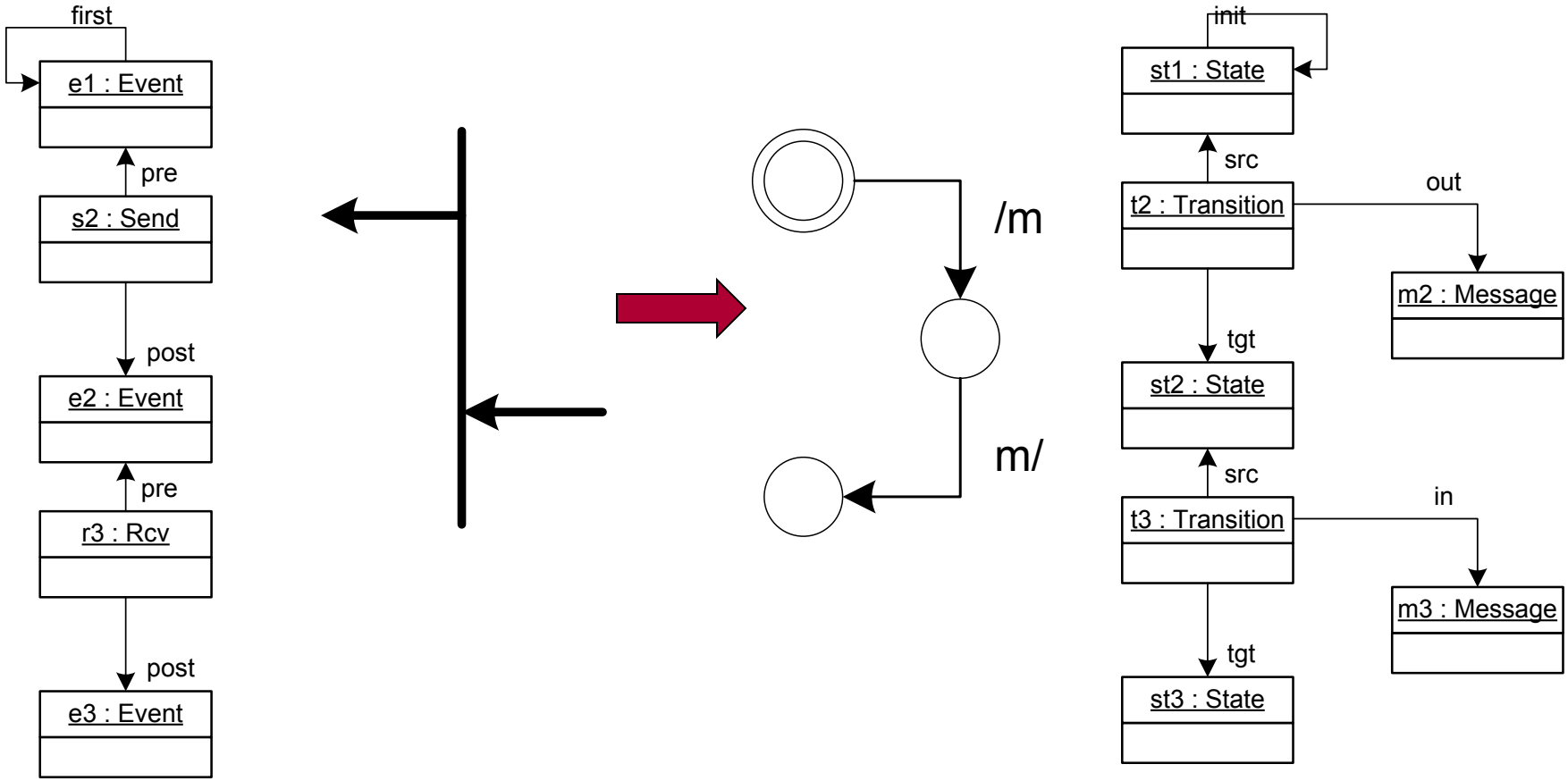


Outline

- Overview
- **TGG Implementation**
- TGG Conformance
- TGG Analysis
- Conclusion



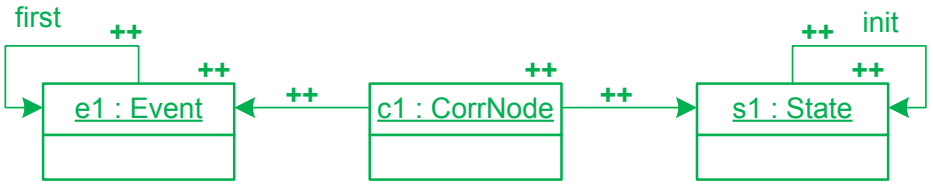
Example Transformation



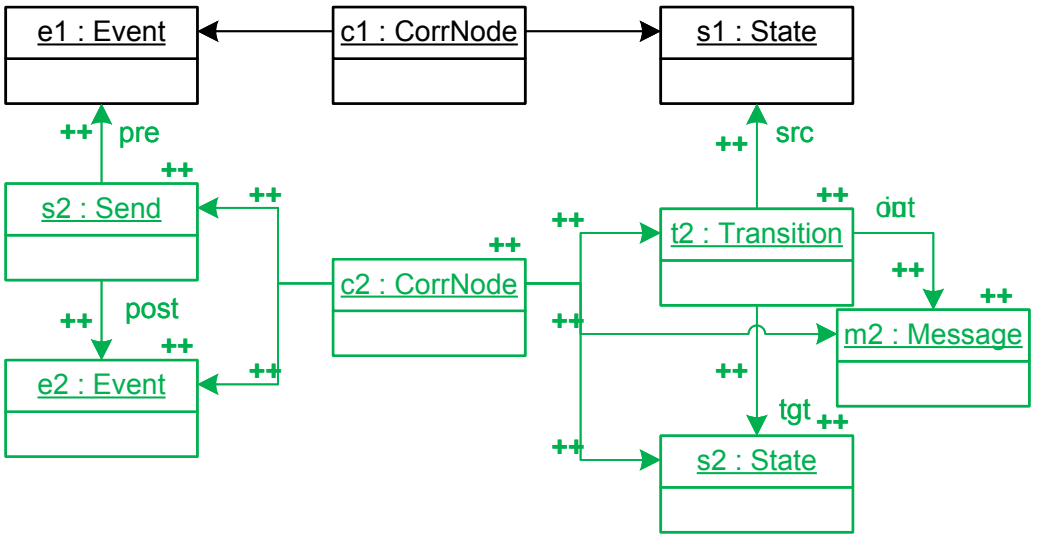
- transform sequence chart to automaton model

Example Transformation with TGGs

- bidirectional transformation specification
- combine three graph grammars
  - source model
  - target model
  - correspondence model



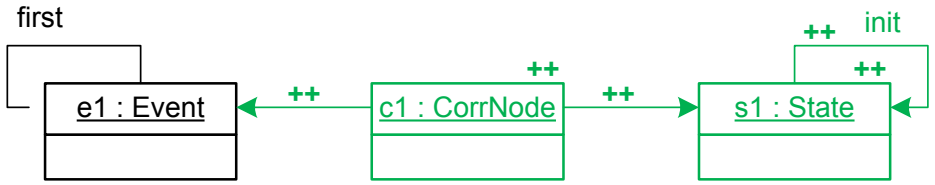
Axiom



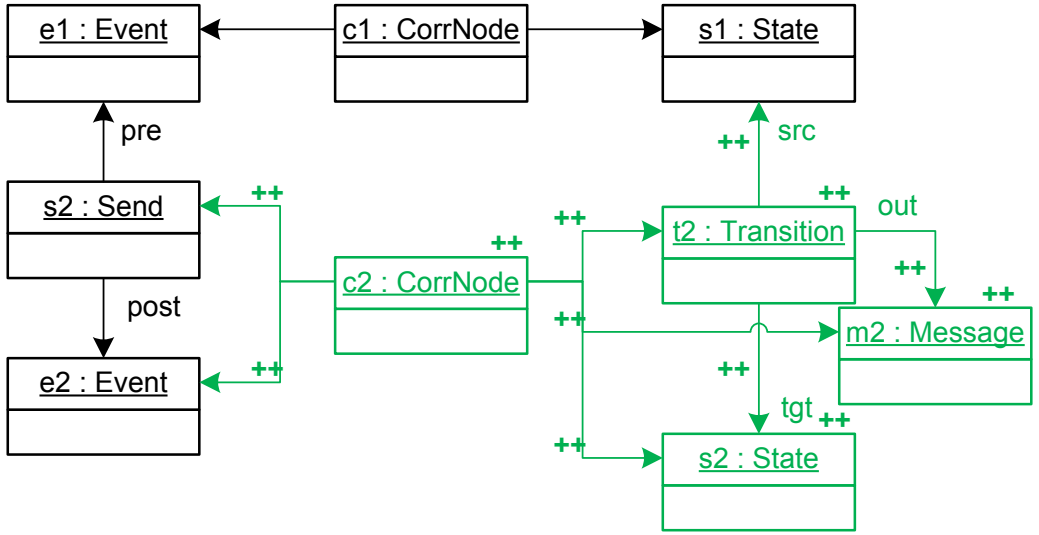
Rule 2 (Send)

Example Transformation with TGGs

- derive operational rules for transformation directions
  - forward
  - backward
  - mapping/ model integration
- add all source elements to application condition
- add elements for bookkeeping, optimizations, etc. (not shown here)

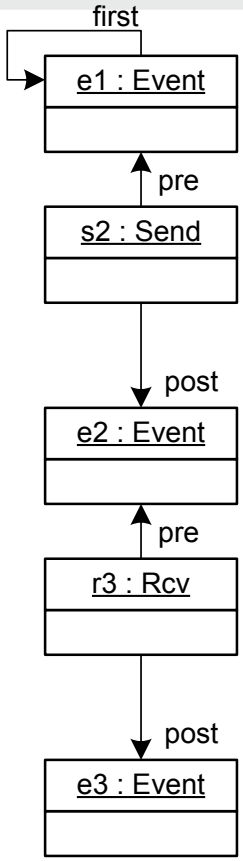


Operational Forward Axiom

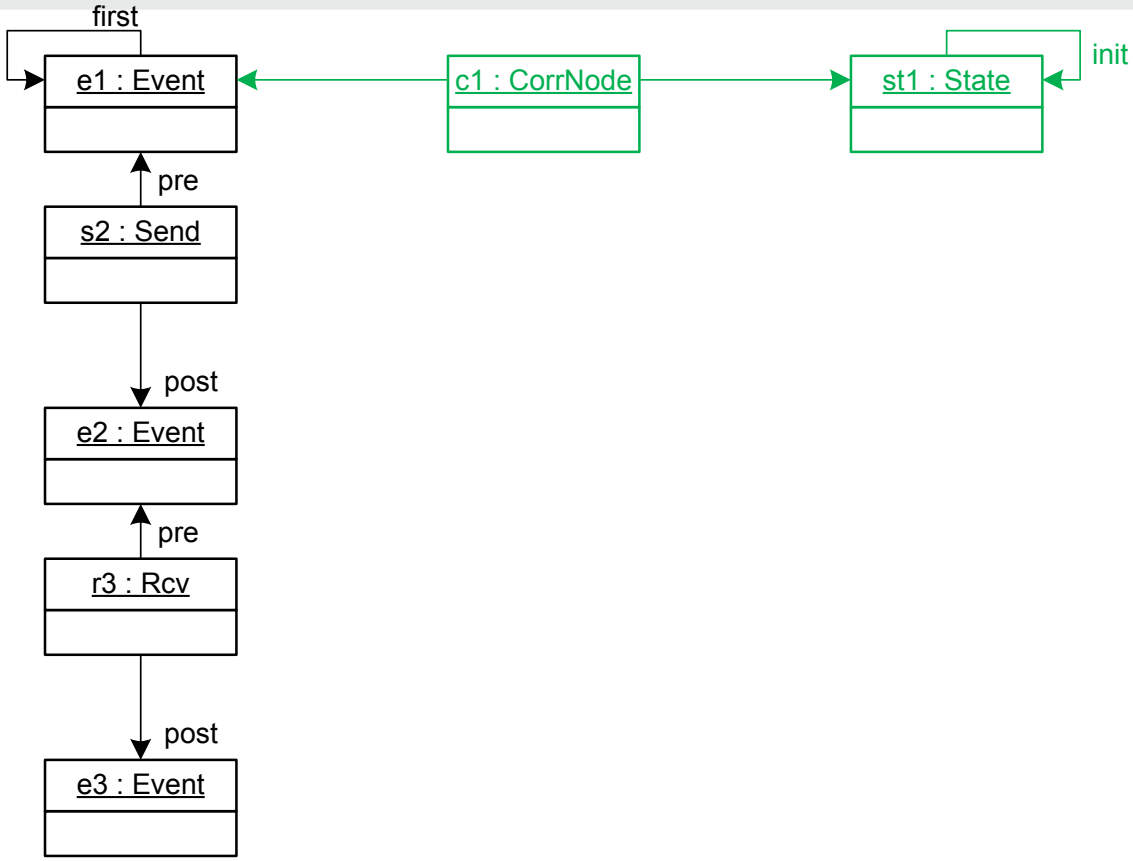


Operational Forward Rule 1

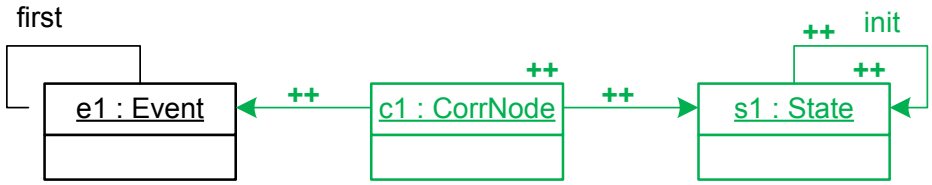
Example Transformation with TGGs



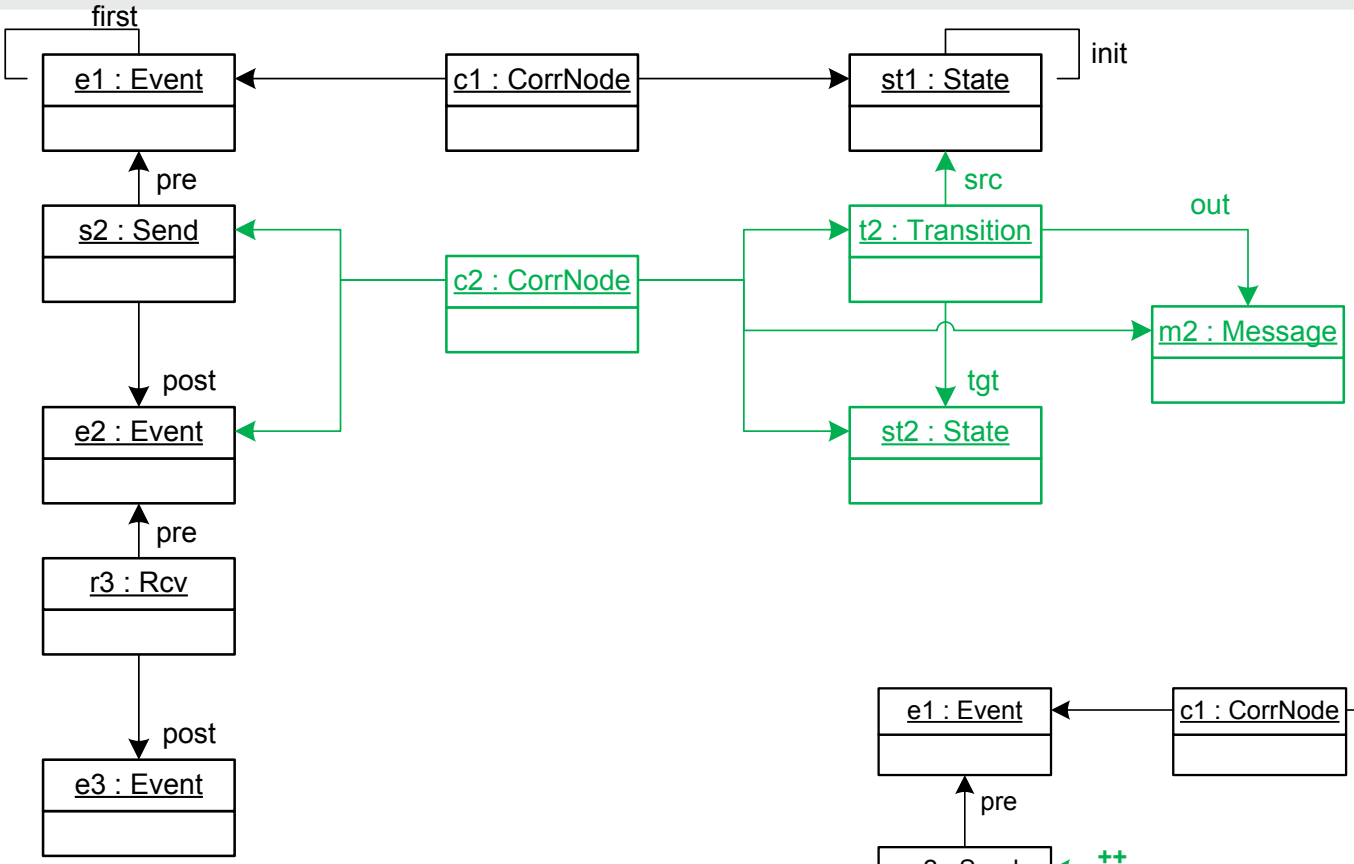
Example Transformation with TGGs

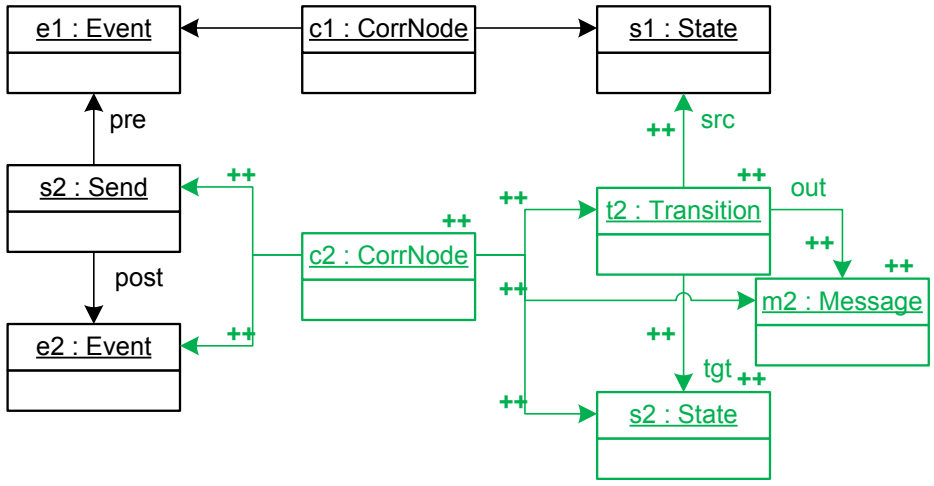


- apply axiom



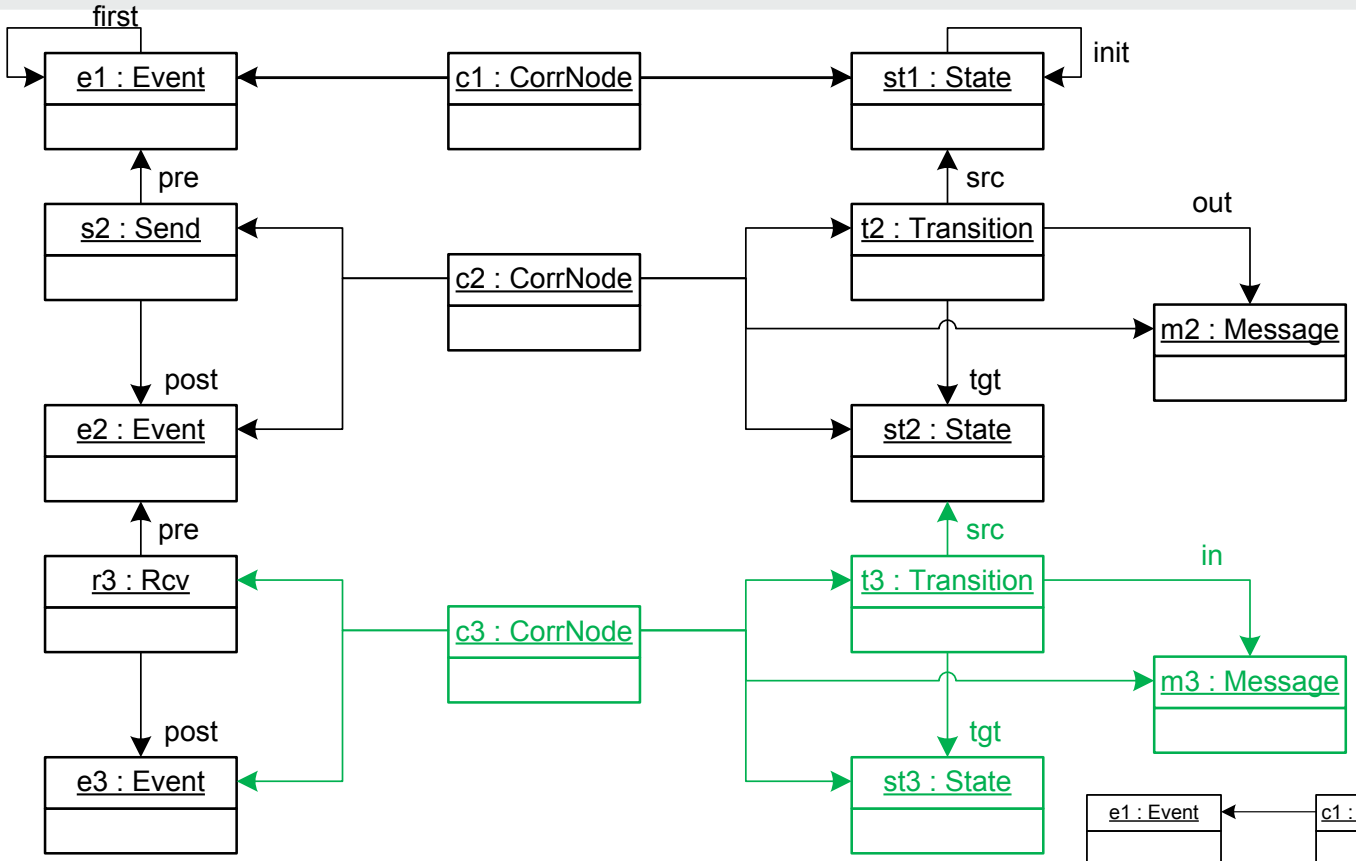
Example Transformation with TGGs

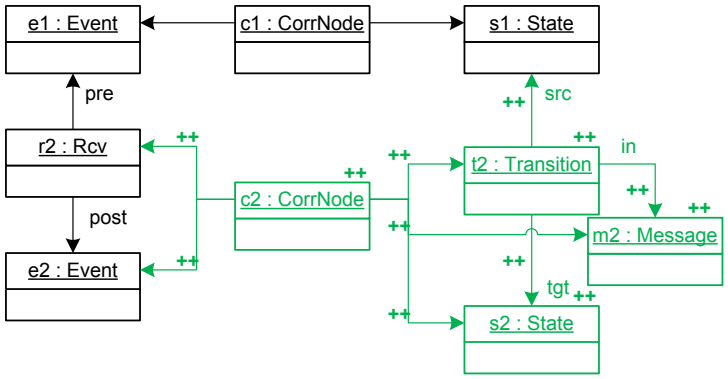




- apply rule 1

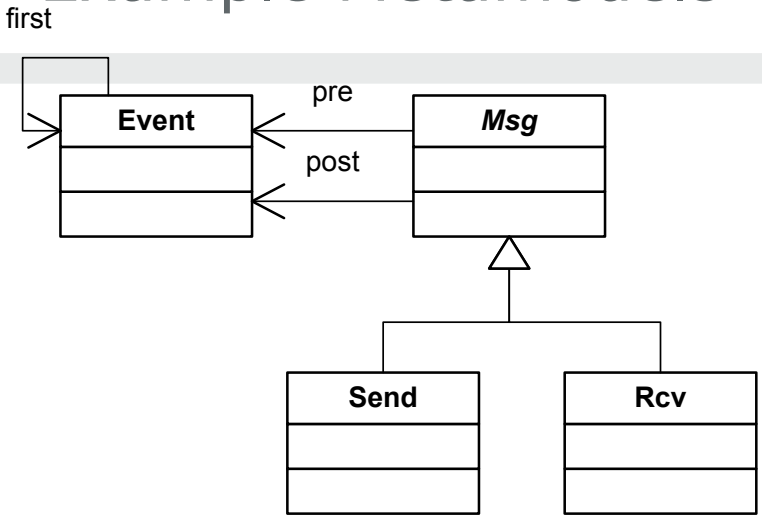
Example Transformation with TGGs



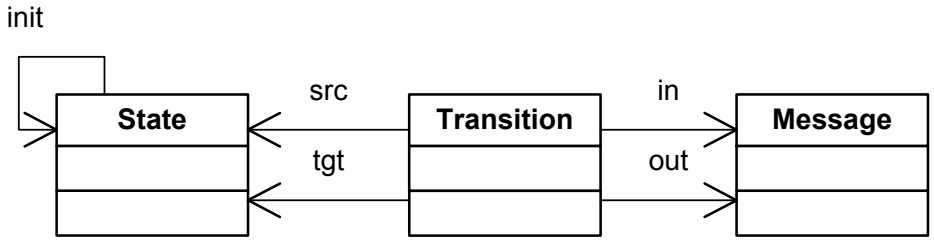


- apply rule 2

Example Metamodels



Sequence Chart Metamodel



Automaton Metamodel

context **Event** inv:
Msg->allInstances()
->select(m|m.pre = **self**)
->size()<=1

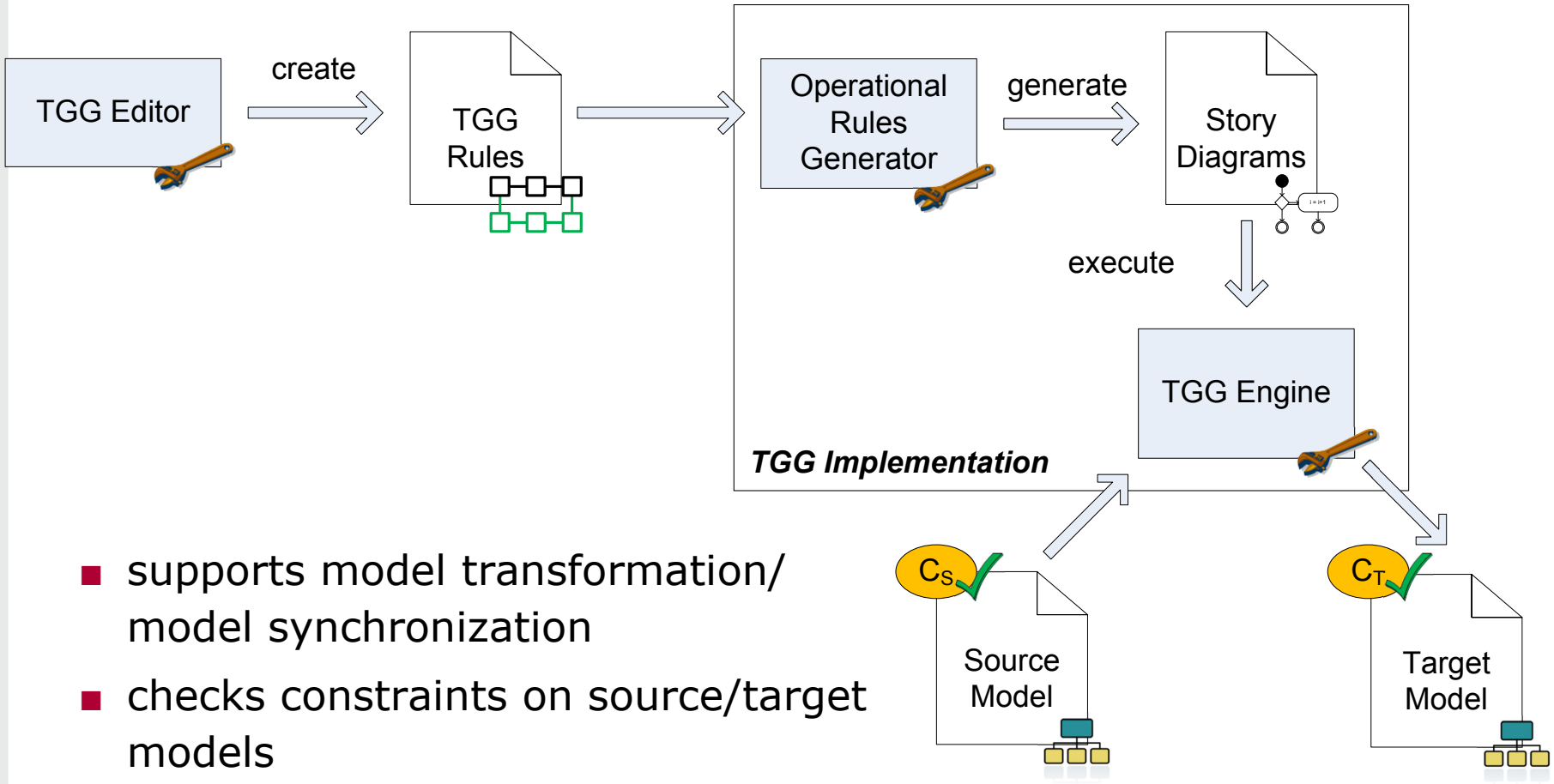
*Only one **Msg** may be sent with each **Event**.*

context **Transition** inv:
not (**self**.in <> null and **self**.out <> null)

*A **Transition** may not have an **in** and an **out** **Message** at the same time.*

- metamodel contain constraints
- ➢ consider metamodel constraints in tooling

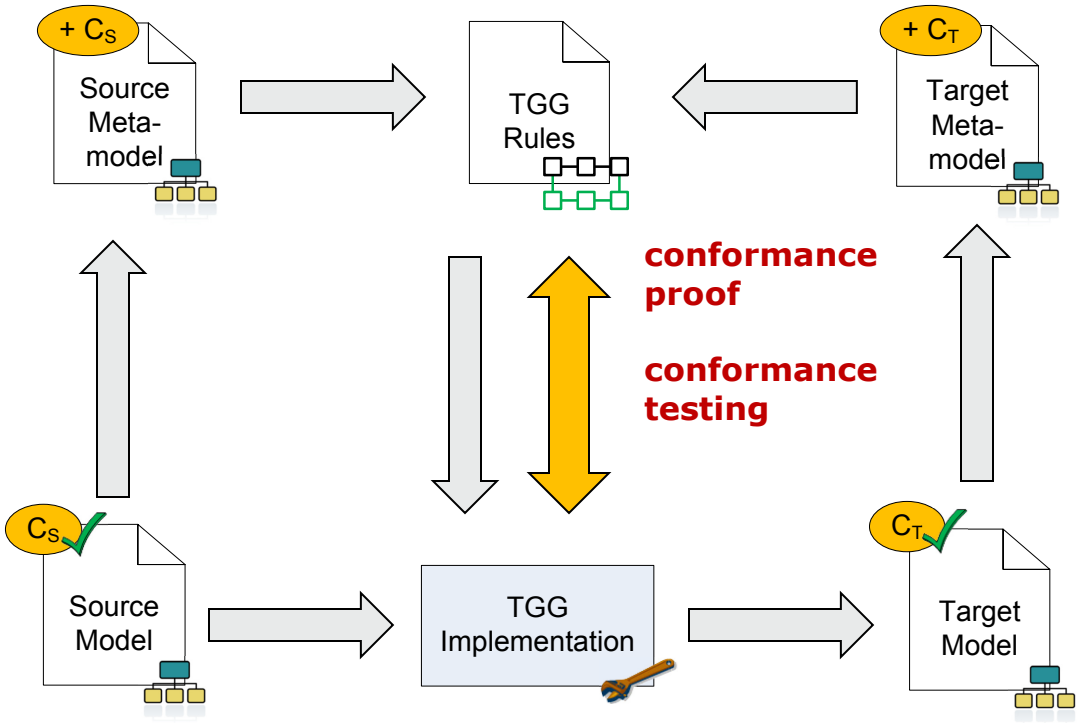
TGG Implementation



- supports model transformation/ model synchronization
- checks constraints on source/target models

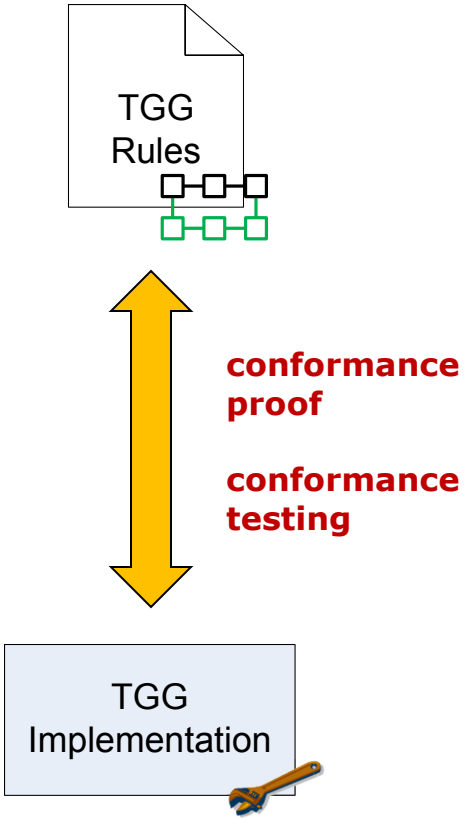
Outline

- Overview
- TGG Implementation
- **TGG Conformance**
- TGG Analysis
- Conclusion
- Future Work



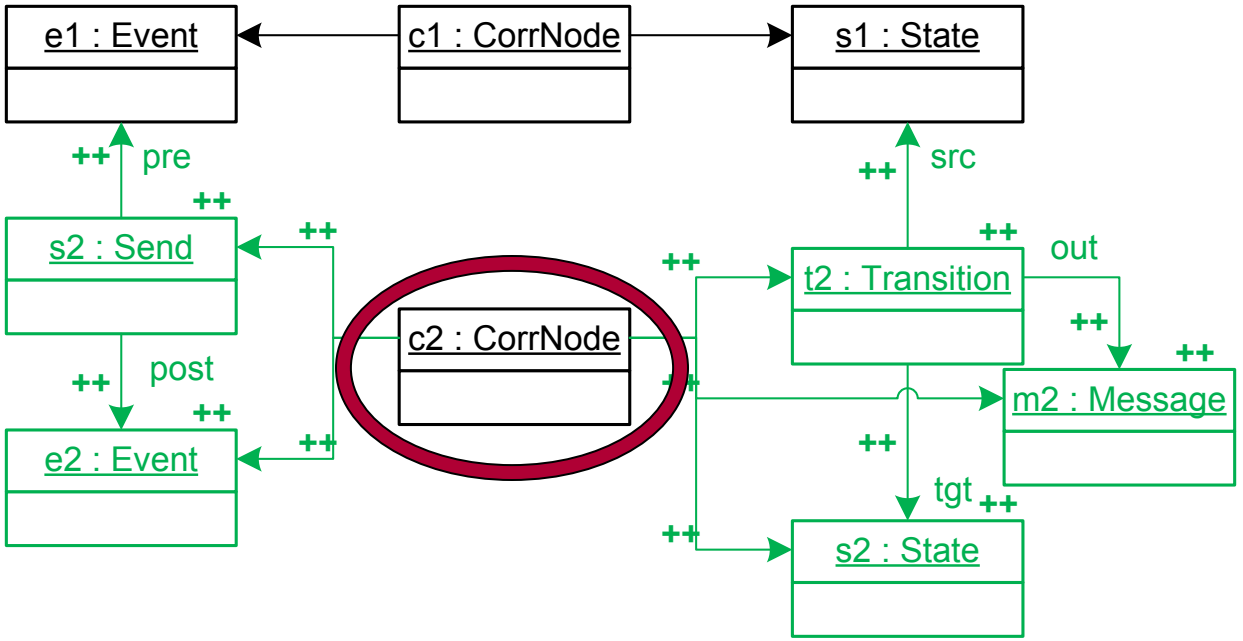
TGG Conformance Proof

- proof conformance of algorithm of TGG implementation to TGG specification
- TGG has to satisfy several criteria
  - terminating
  - conflict-free
  - ...
- ensure that model transformation is deterministic and efficiently executable



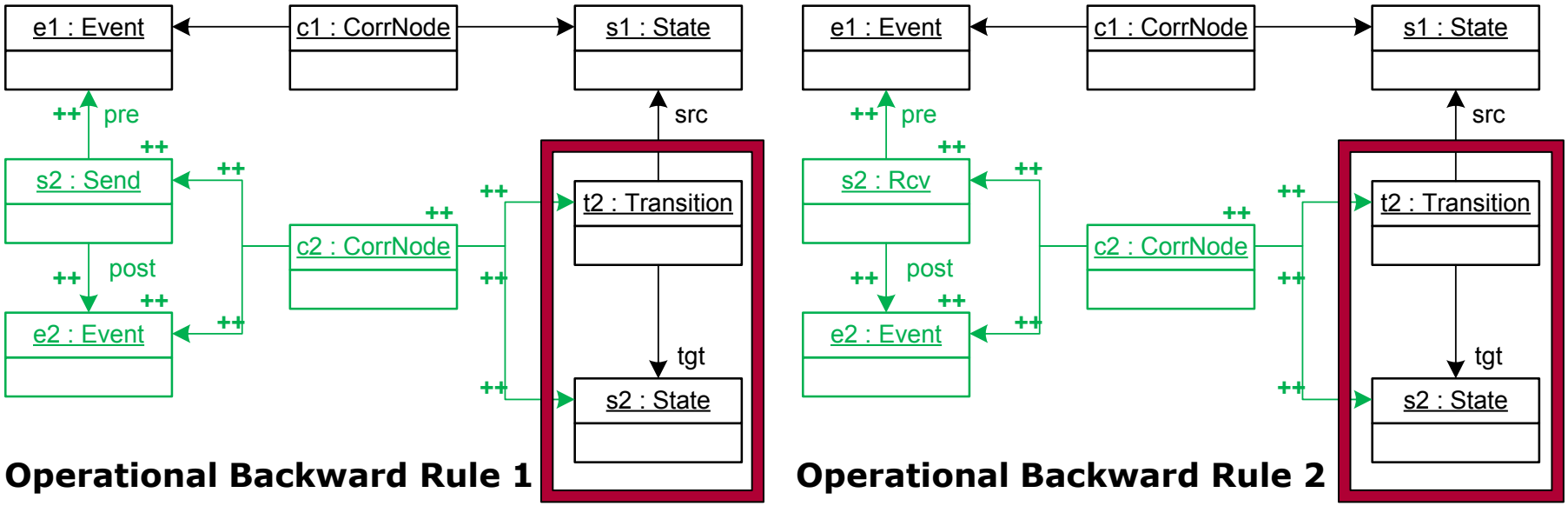
- for more information
  - Giese, Holger, Hildebrandt, Stephan, Lambers, Leen *Bridging the gap between formal semantics and implementation of triple graph grammars*. Software & Systems Modeling, 2012

Syntactic Checks



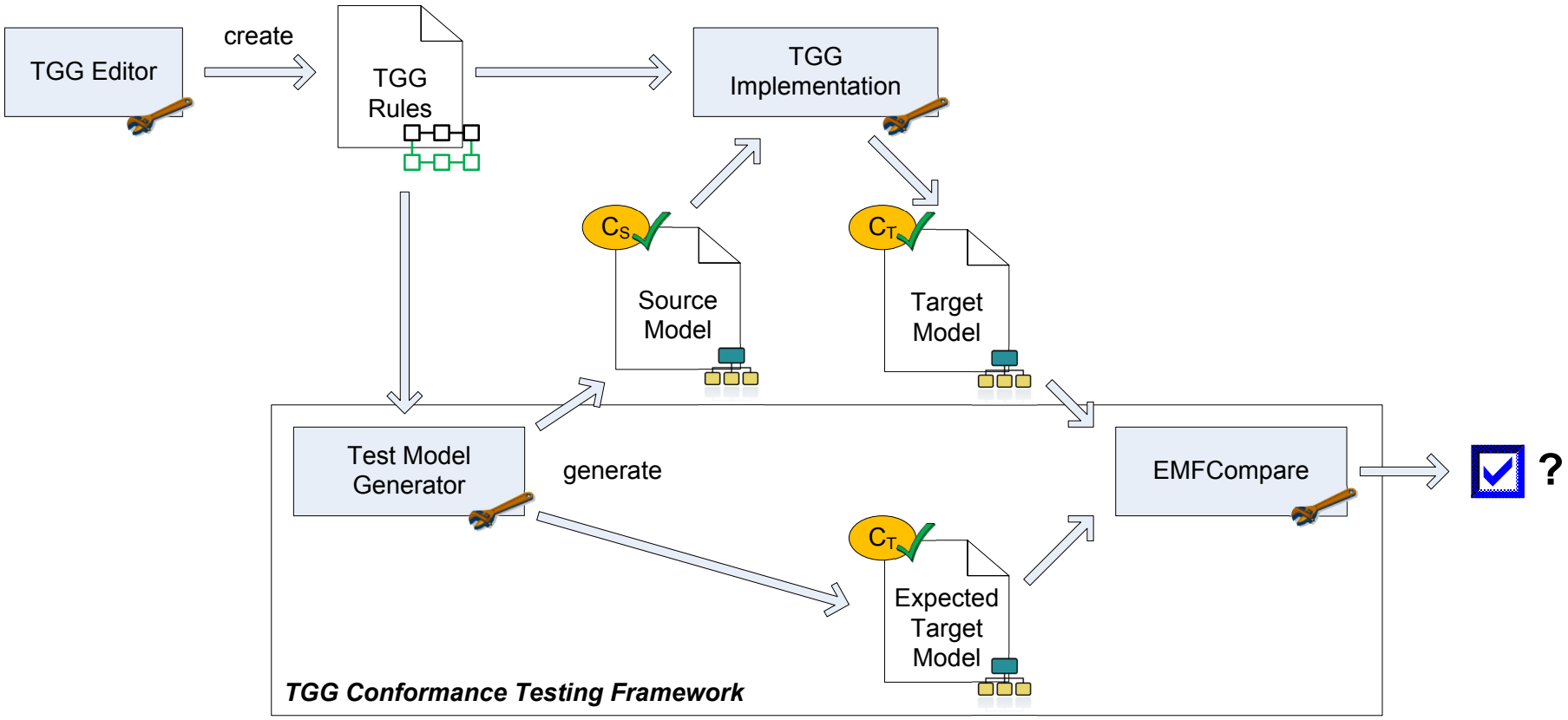
- check that TGG rules satisfy structural constraints, e.g.,
  - create exactly one correspondence node
- as specified in formal conformance proof
- implemented using Xtend Check
- available via EMF Validation Framework

Conflict Check



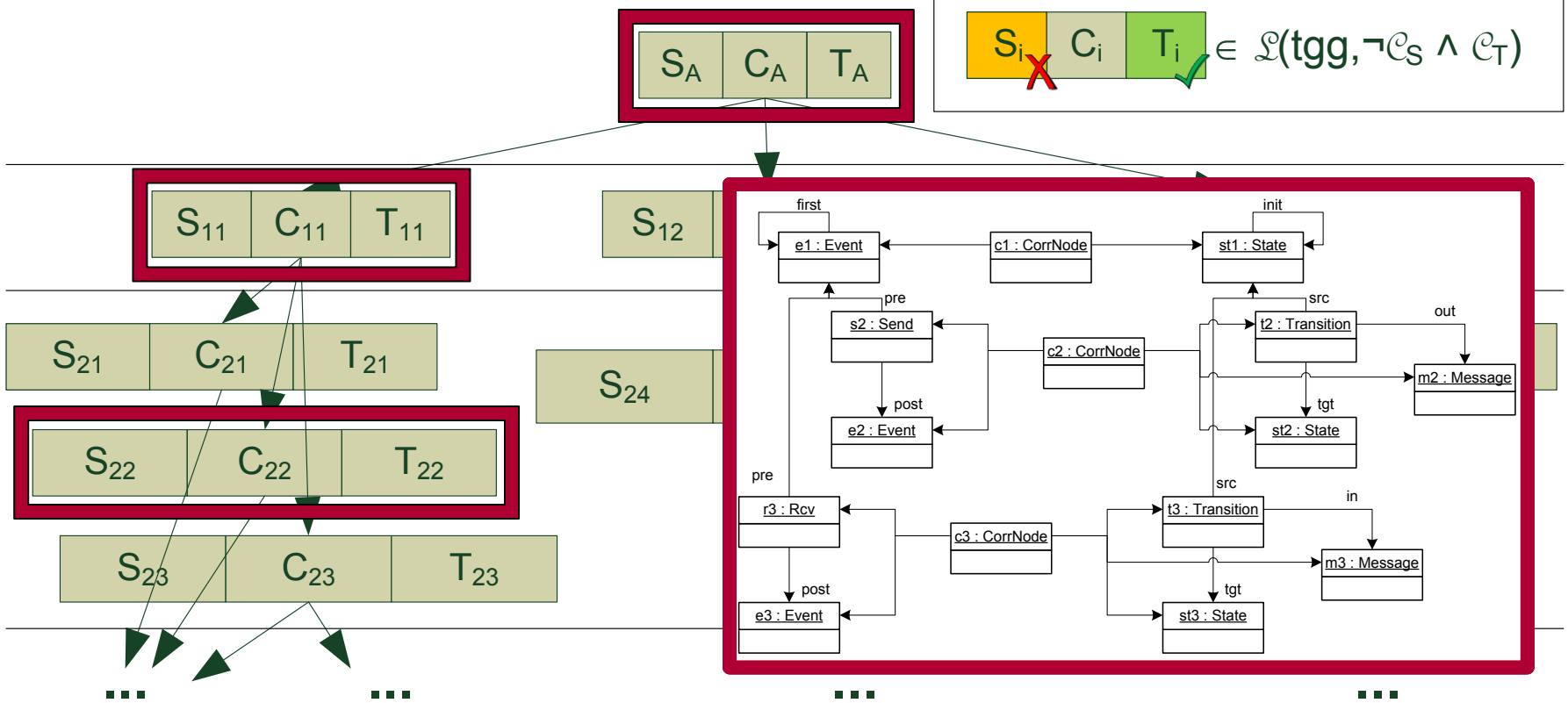
- conformance proof requires deterministic TGGs, i.e., no conflicts between operational forward/backward TGG rules
- use critical-pair analysis of *AGG*

TGG Conformance Testing Framework



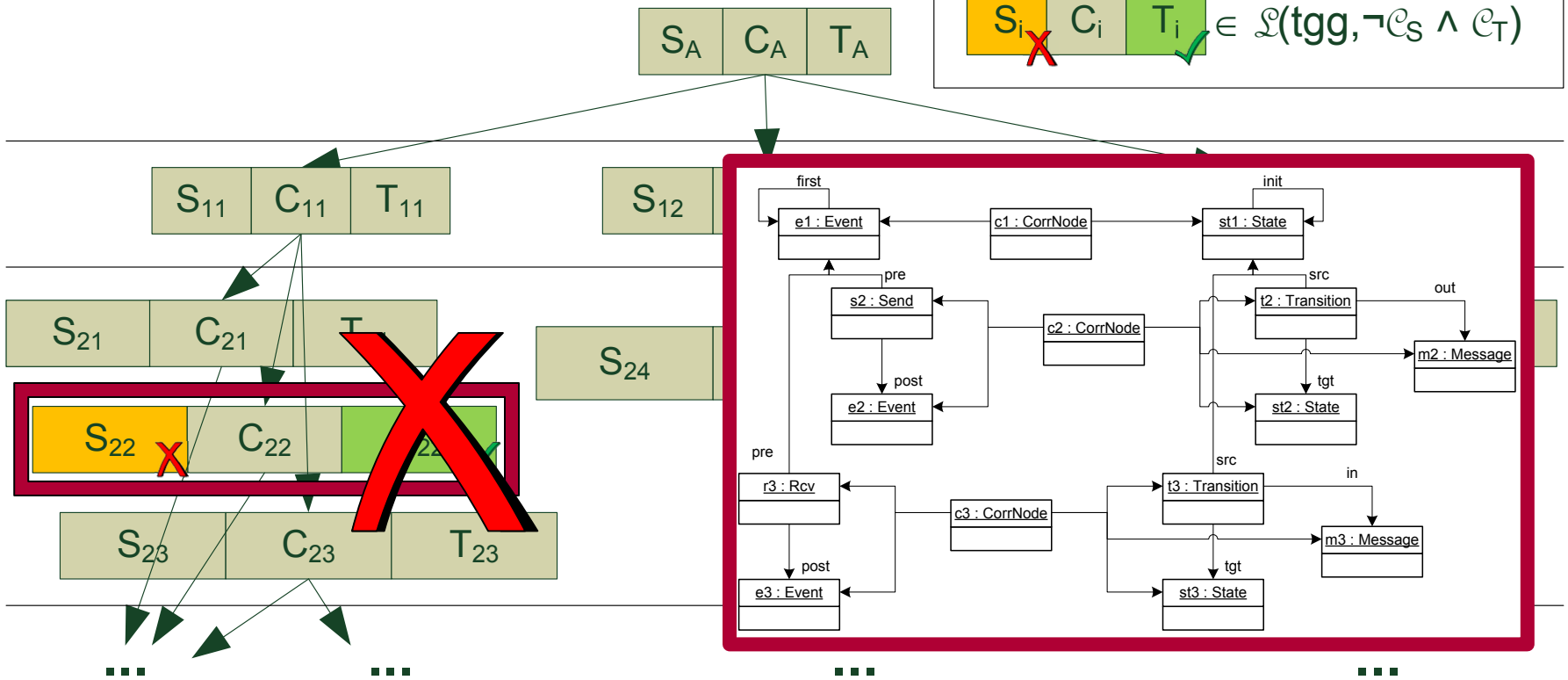
- test conformance of TGG implementation with specification
- generate source and expected target model simultaneously

Test Model Generator



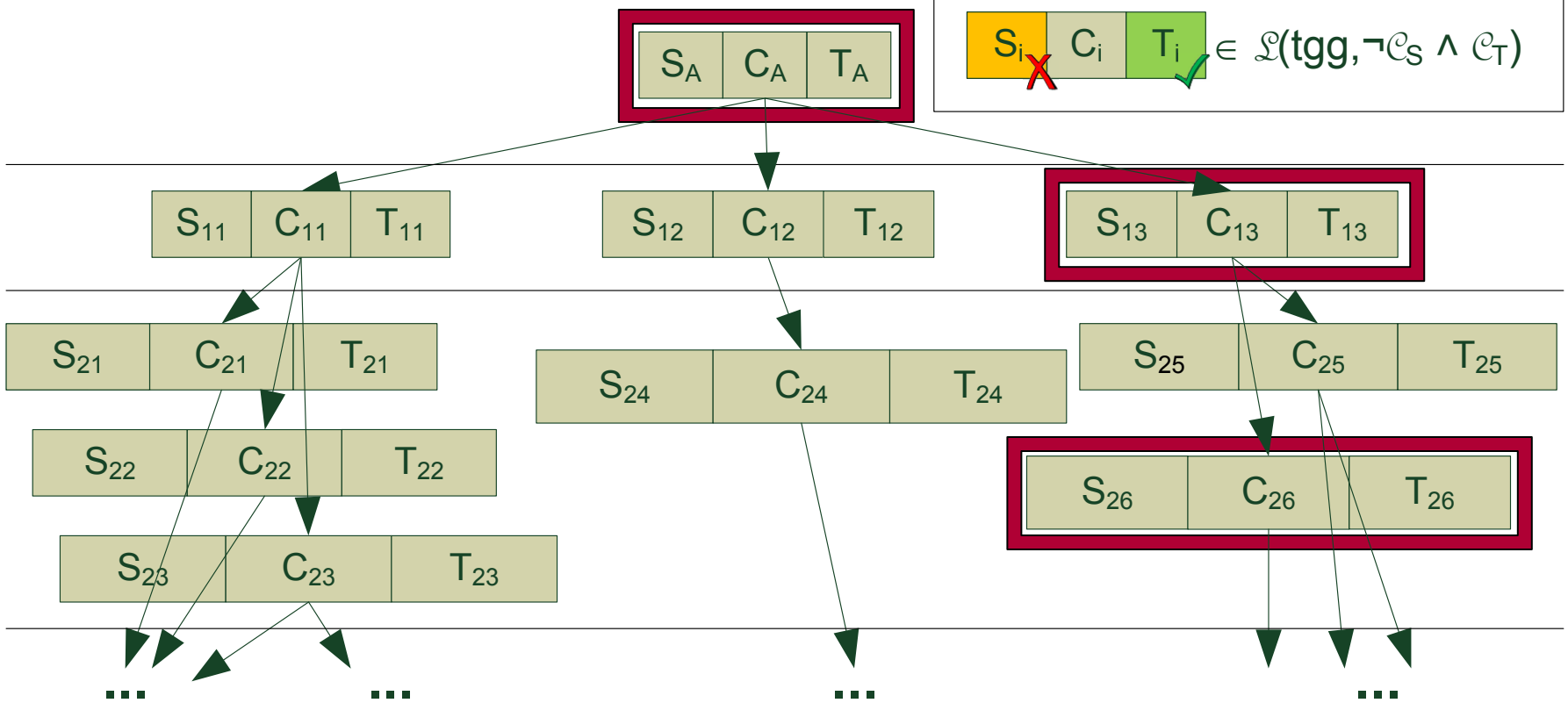
- apply TGG rules randomly to generate models

Test Model Generator



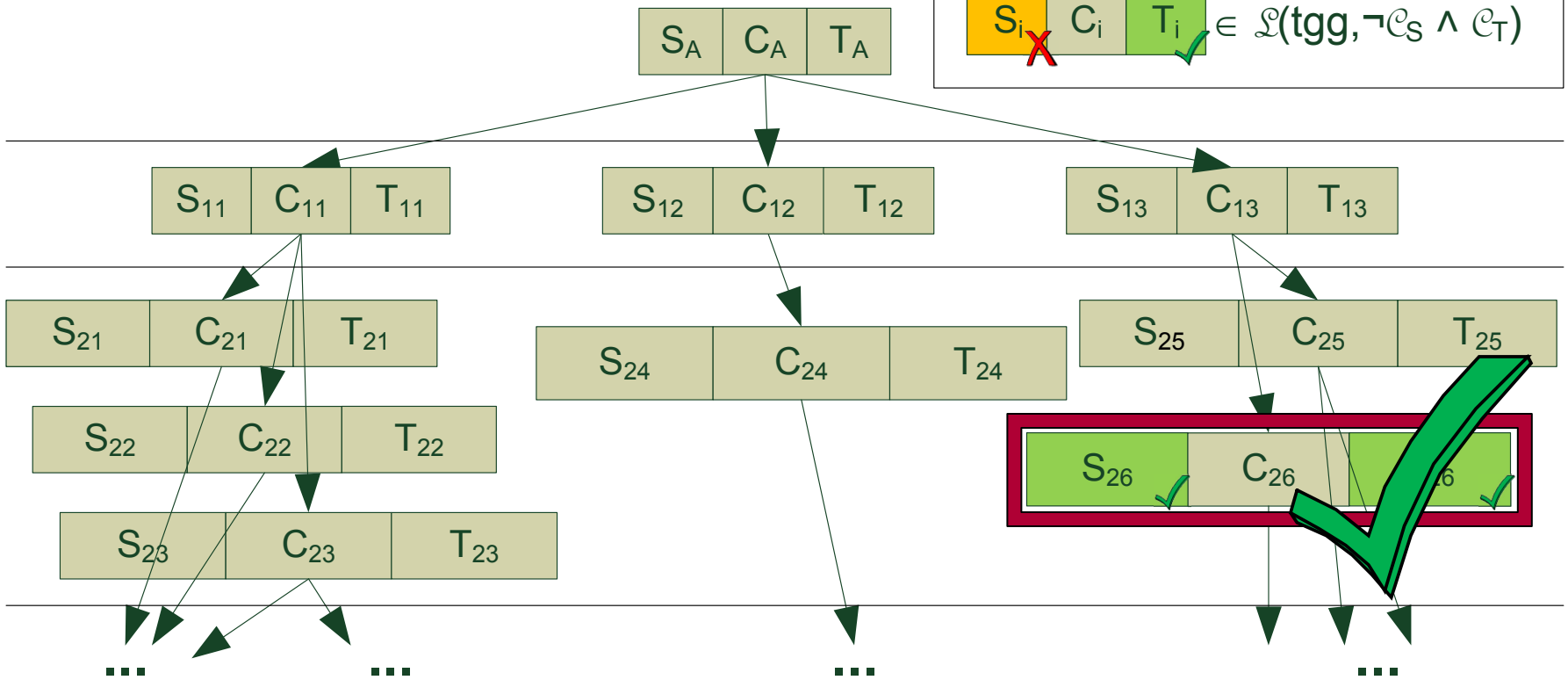
- check constraints on generated models
- if models are invalid, generate new models

Test Model Generator



- check constraints on generated models
- if models are invalid, generate new models

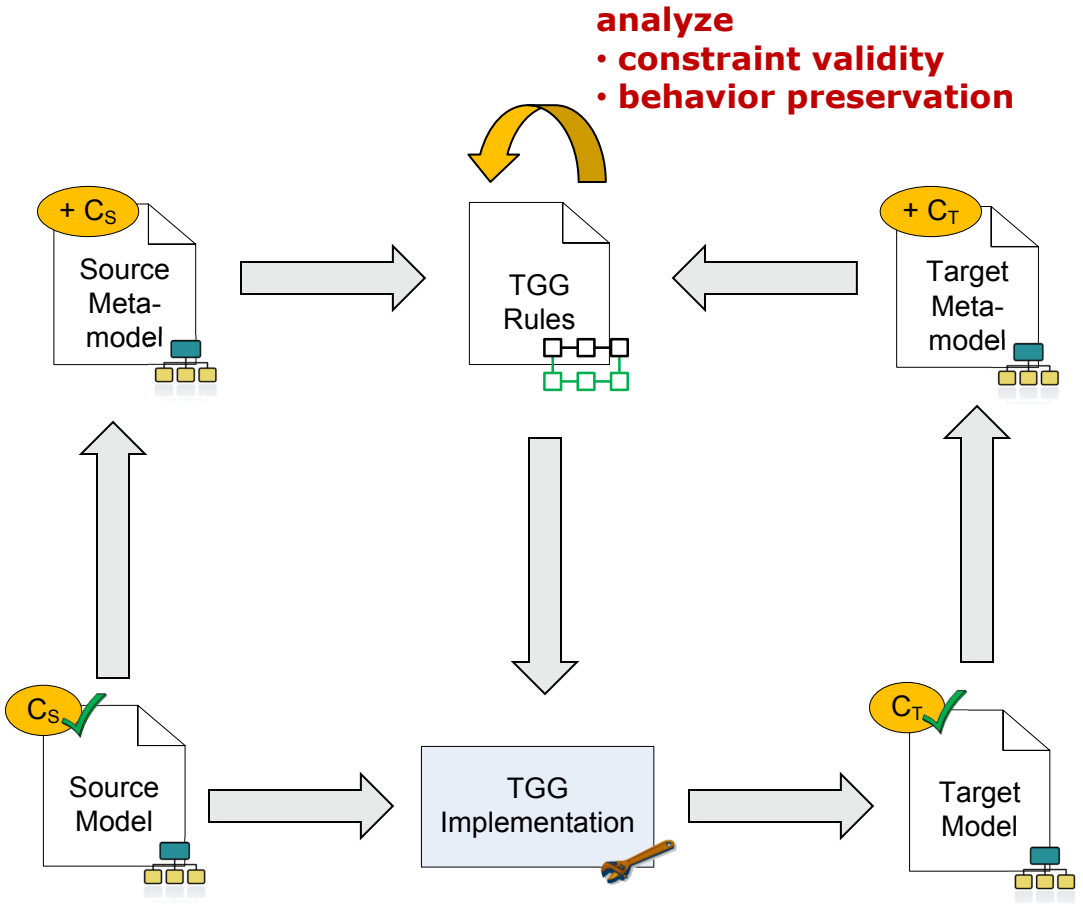
Test Model Generator



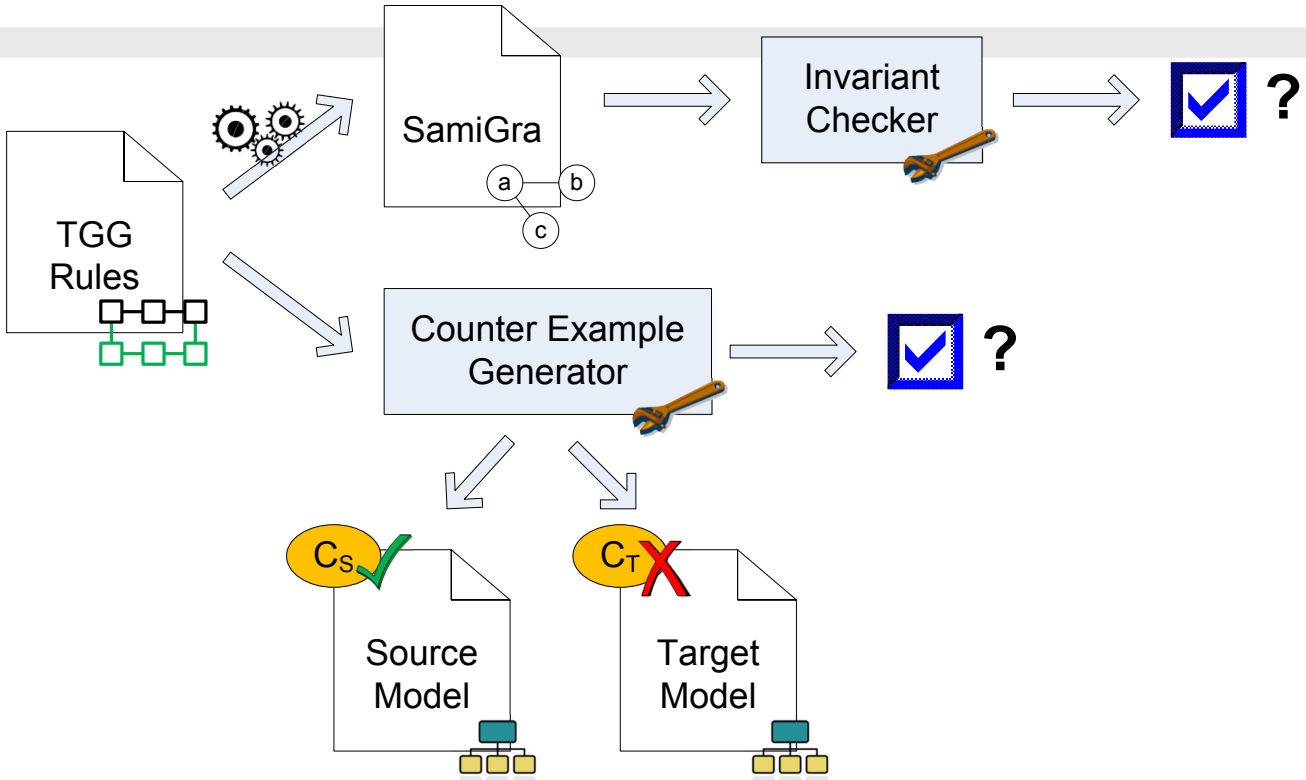
- output source model and corresponding target model

Outline

- Overview
- TGG Implementation
- TGG Conformance
- **TGG Analysis**
- Conclusion
- Future Work

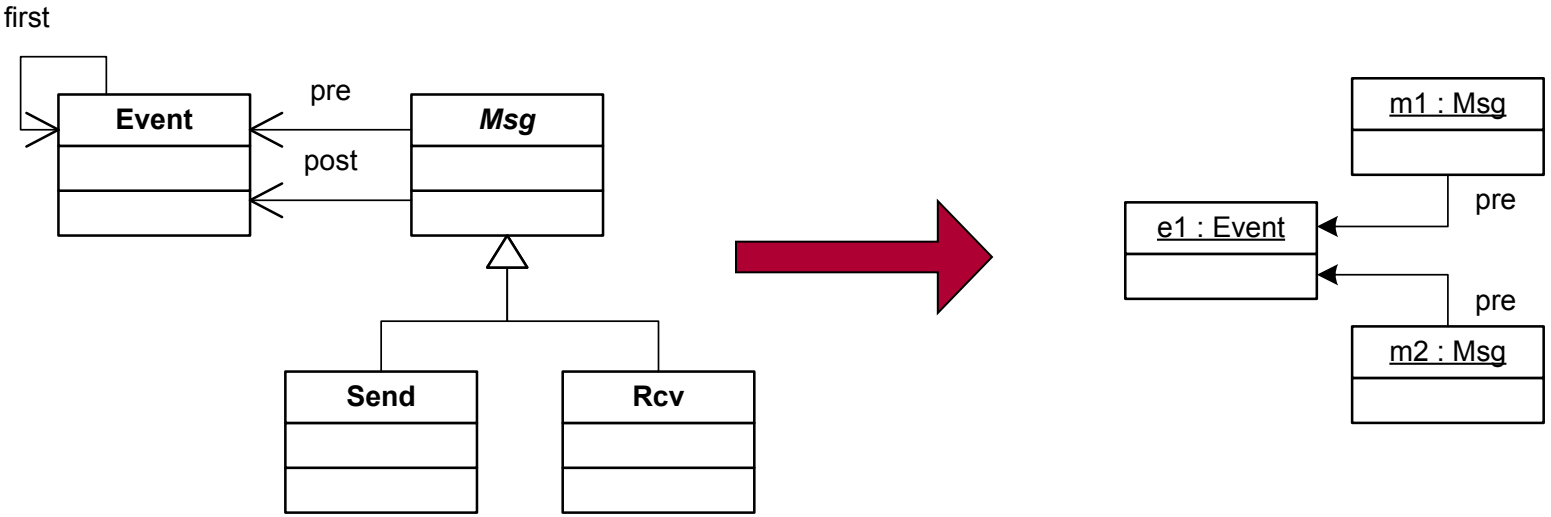


TGG Validation



- check *forward/backward validity* of TGG
  - *forward valid* TGG: valid source model is always transformed to valid target model
- search for counter examples
  - valid source model transformed to invalid target model

Invariant Checker

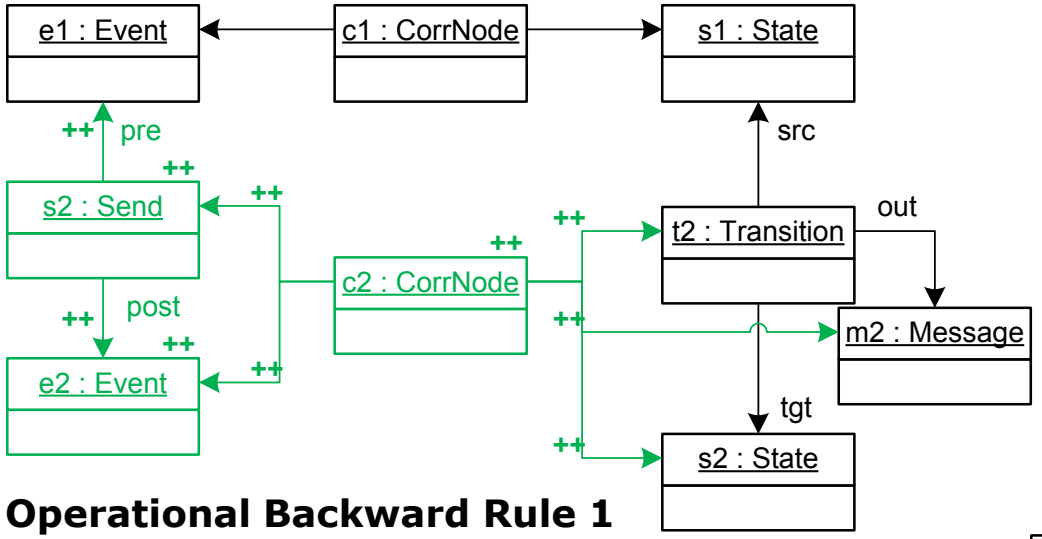


context **Event** inv:
Msg->allInstances()
->select(m|m.pre = **self**)
->size()<=1

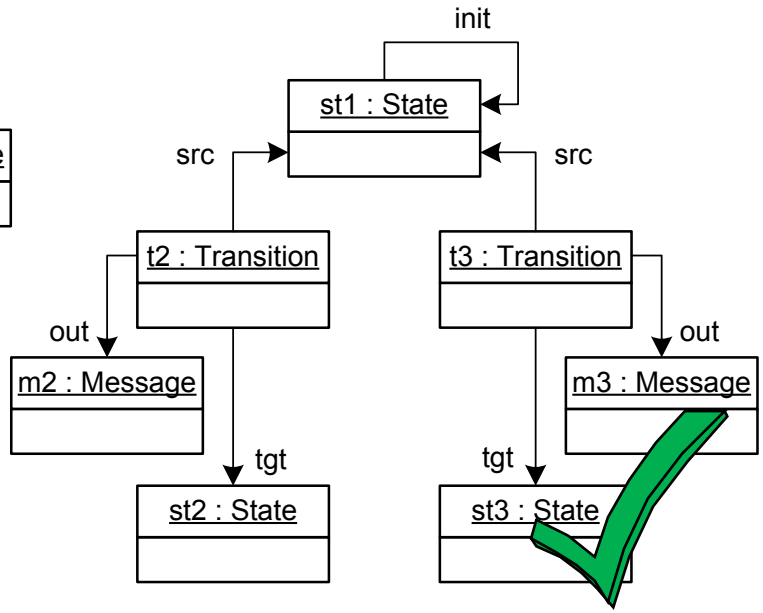
*Only one **Msg** may be sent with each **Event**.*

- translate constraints to forbidden graph patterns

Invariant Checker

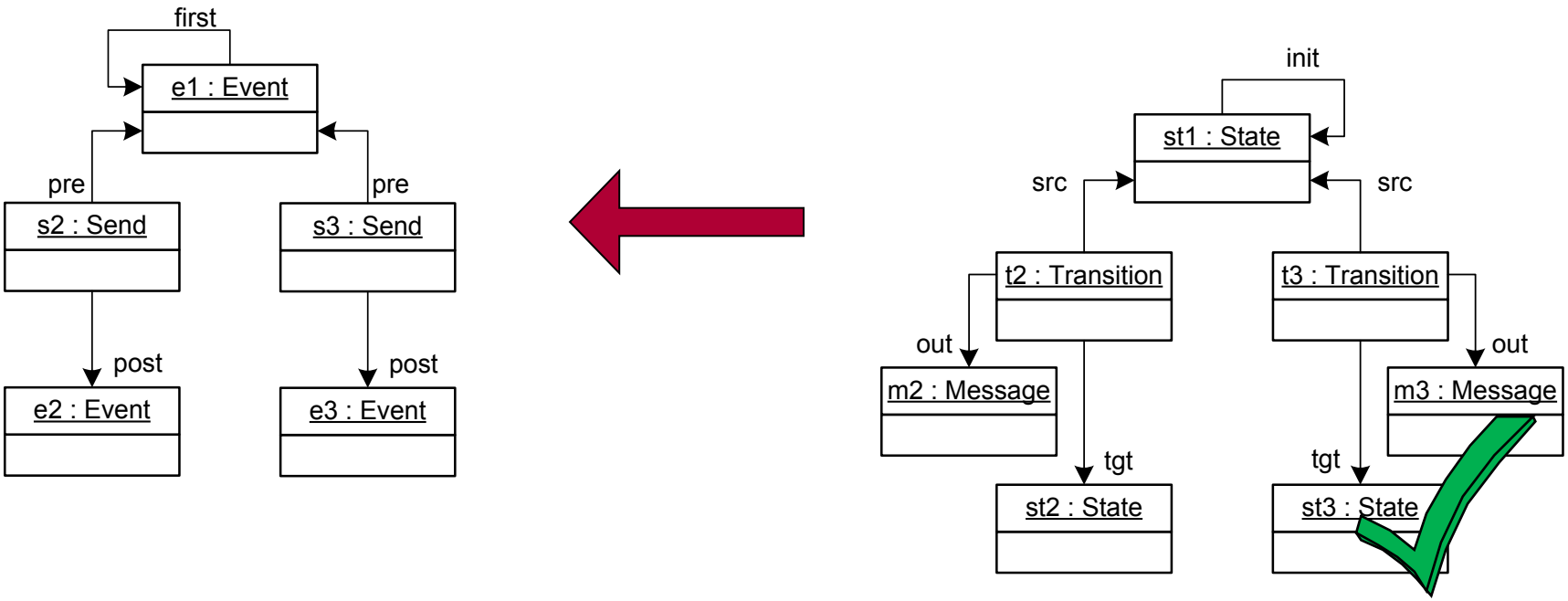


Operational Backward Rule 1



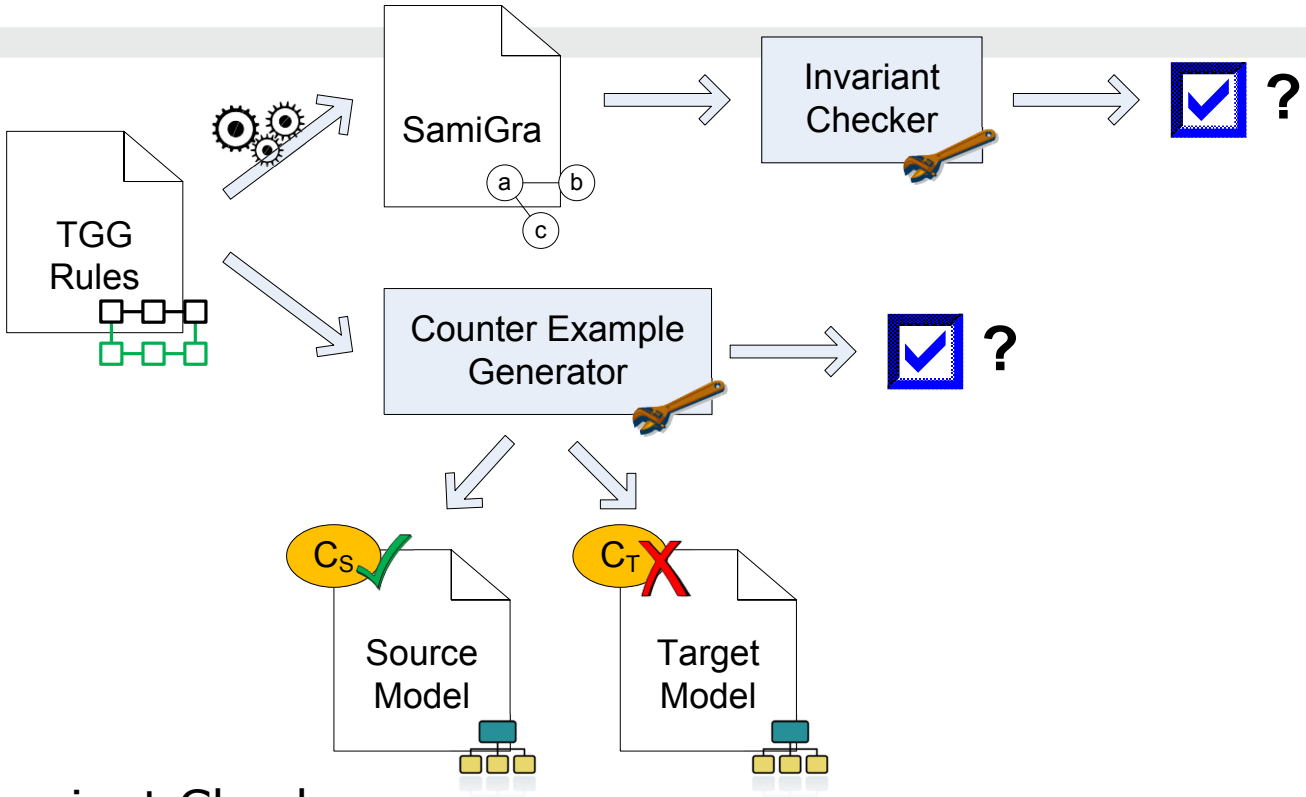
- search for symbolic counter examples where *operational* rule application leads to forbidden pattern

Invariant Checker



- operational backward rule 1 may be applied arbitrarily
- create *Event* with multiple *Msg*
- ➢ TGG is not *backward valid*

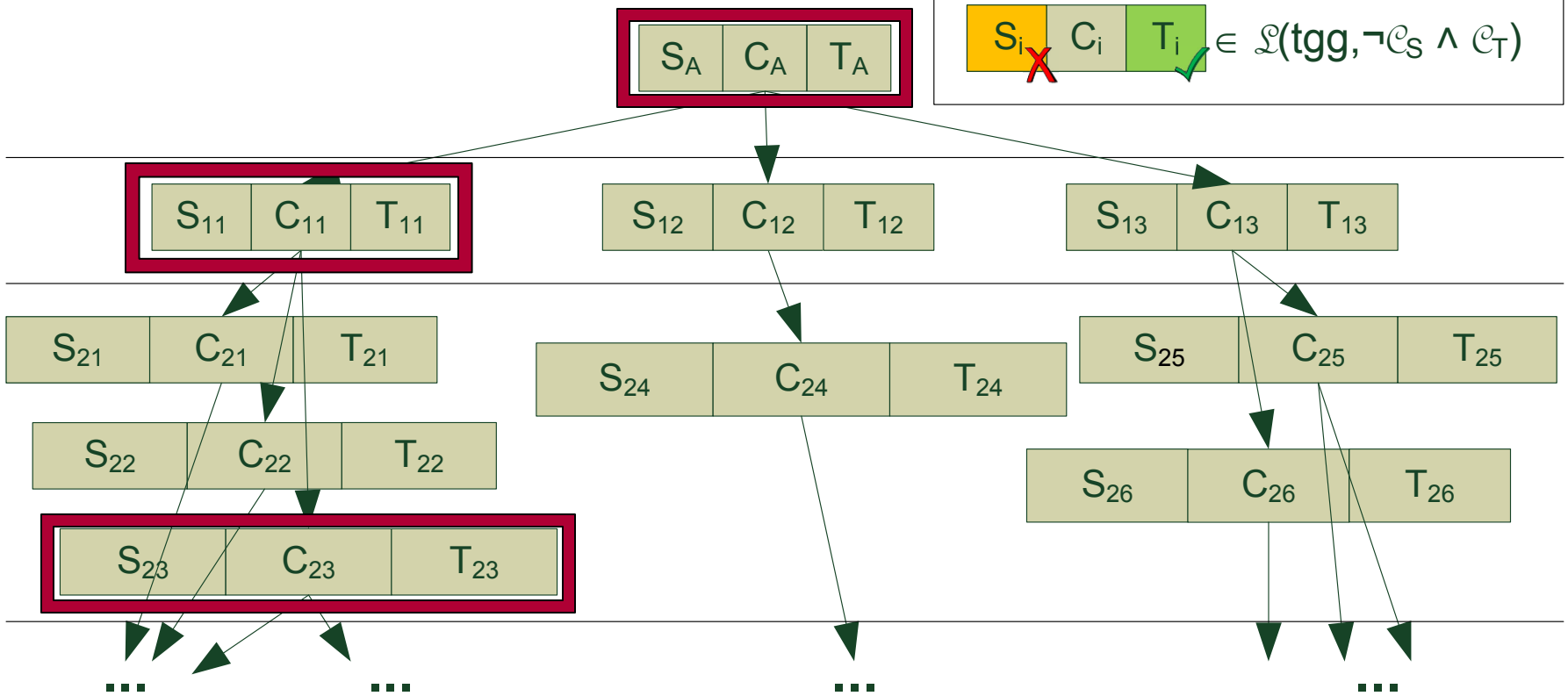
TGG Validation



- Invariant Checker
  - complete static analysis
  - translate constrains to graph patterns

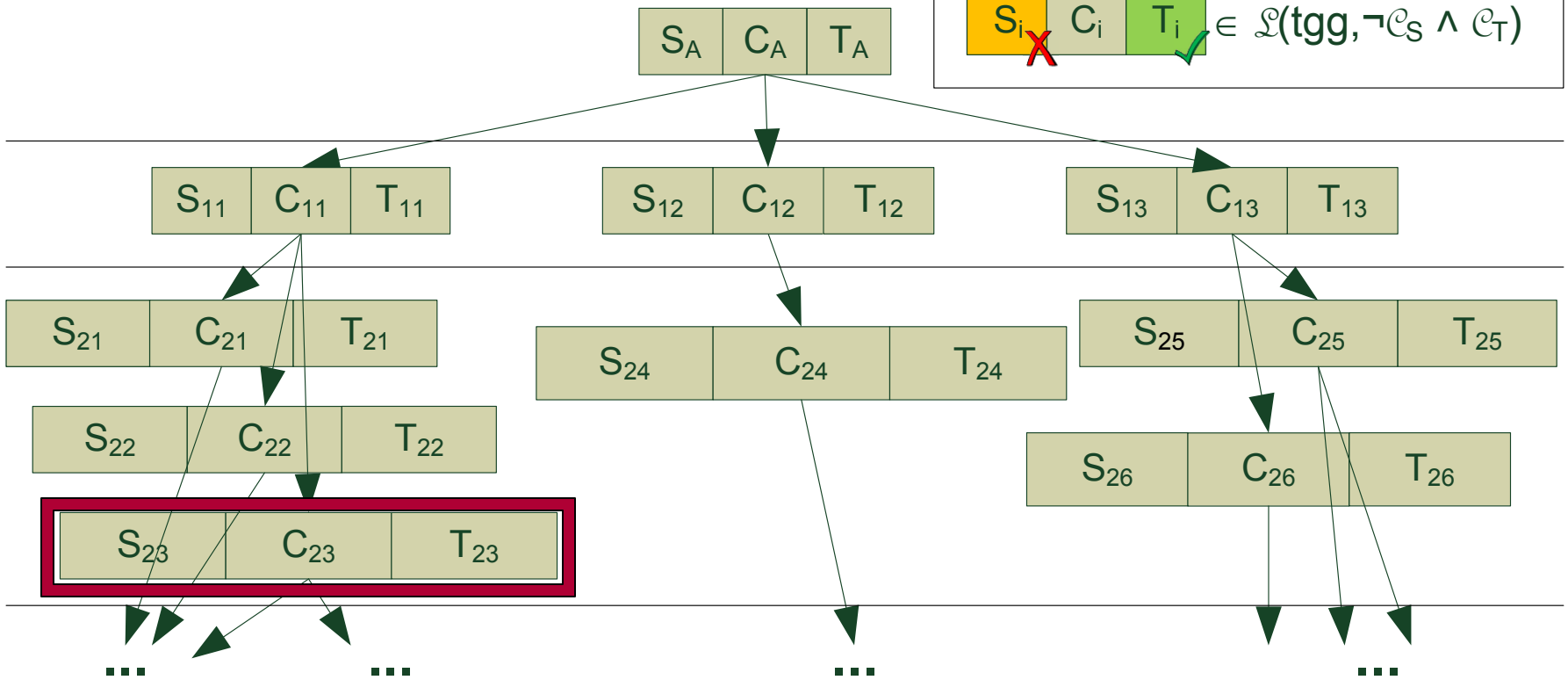
➢ different approach: Counter Example Generator

Counter Example Generator



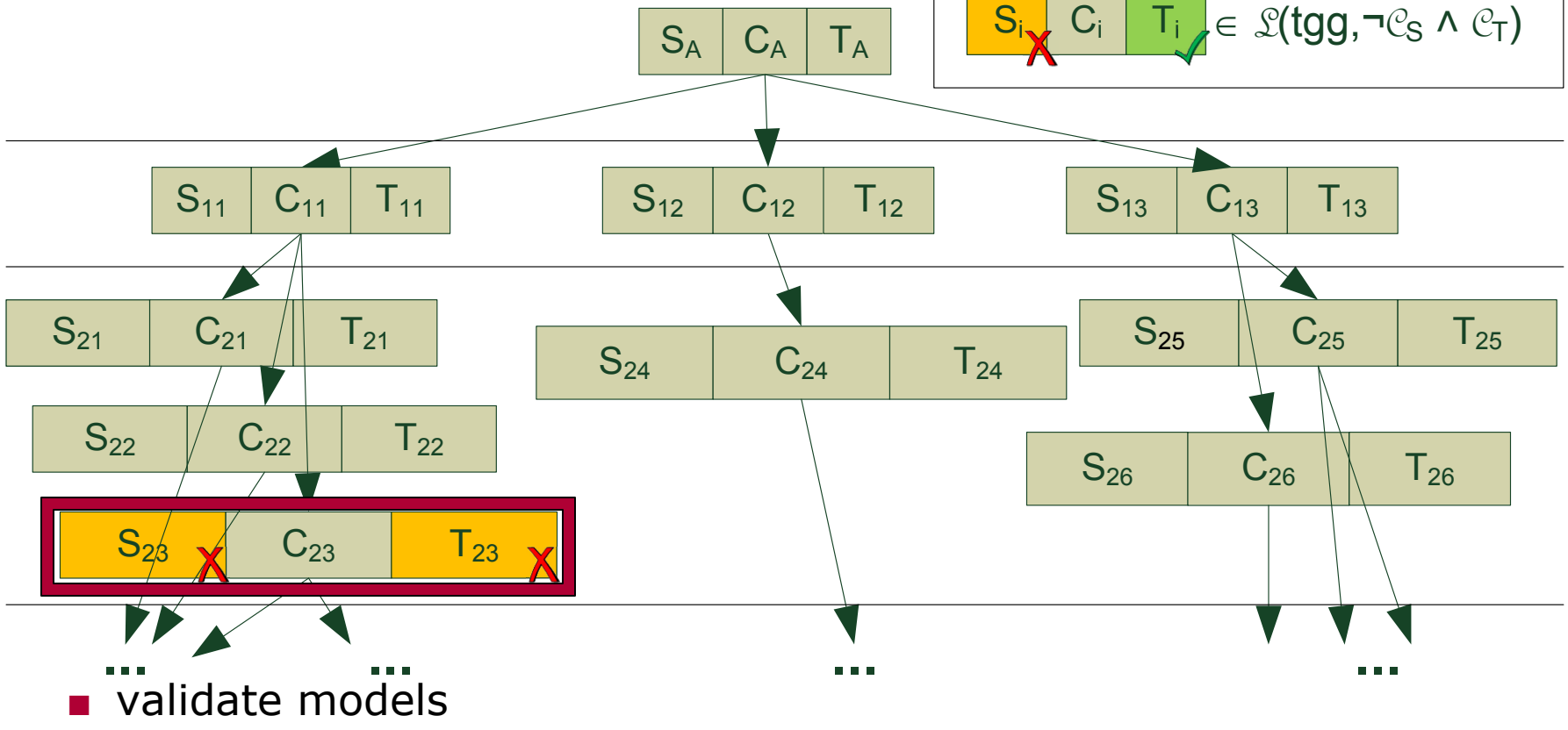
- generate random pair of models

Counter Example Generator



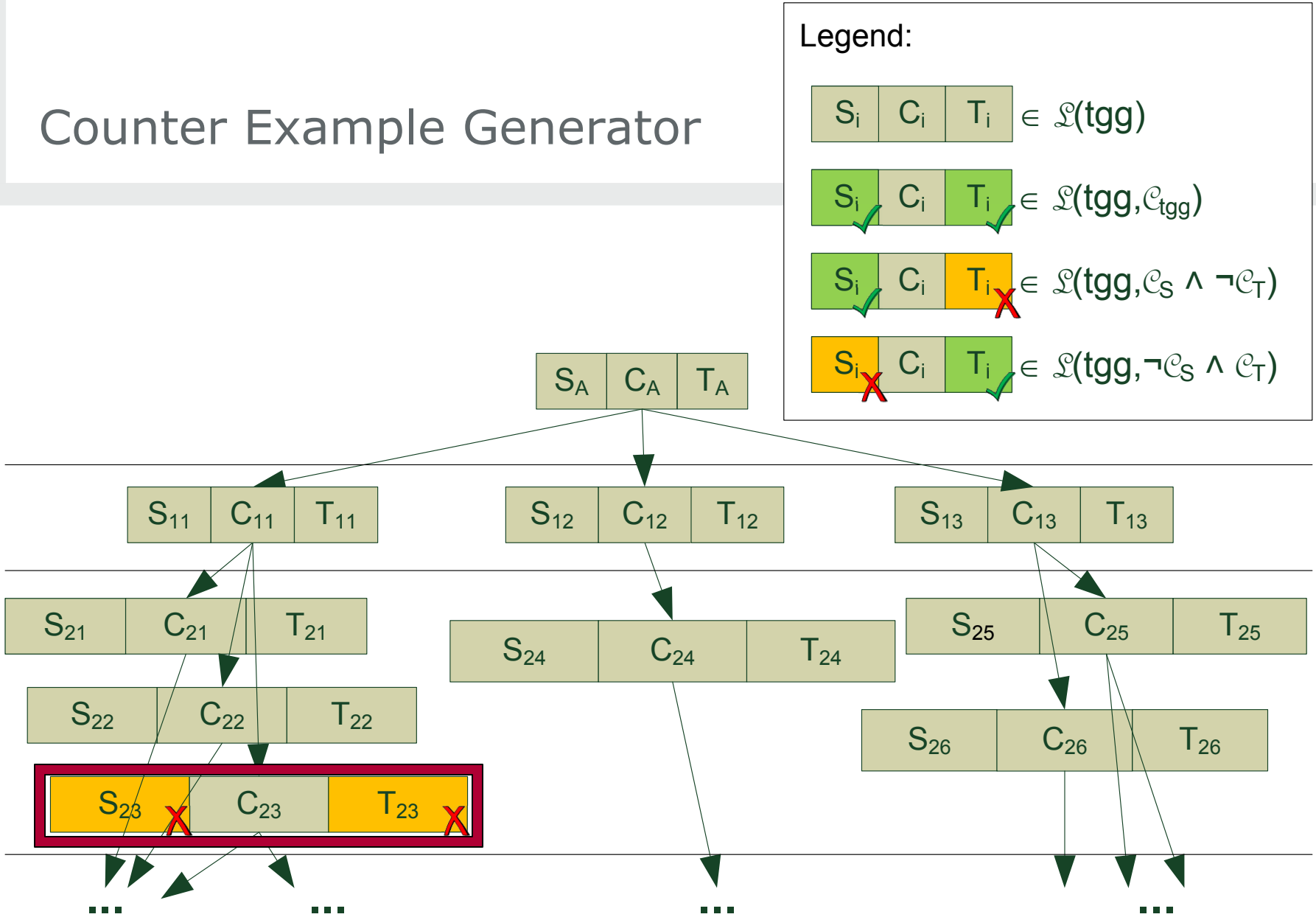
- validate models

Counter Example Generator



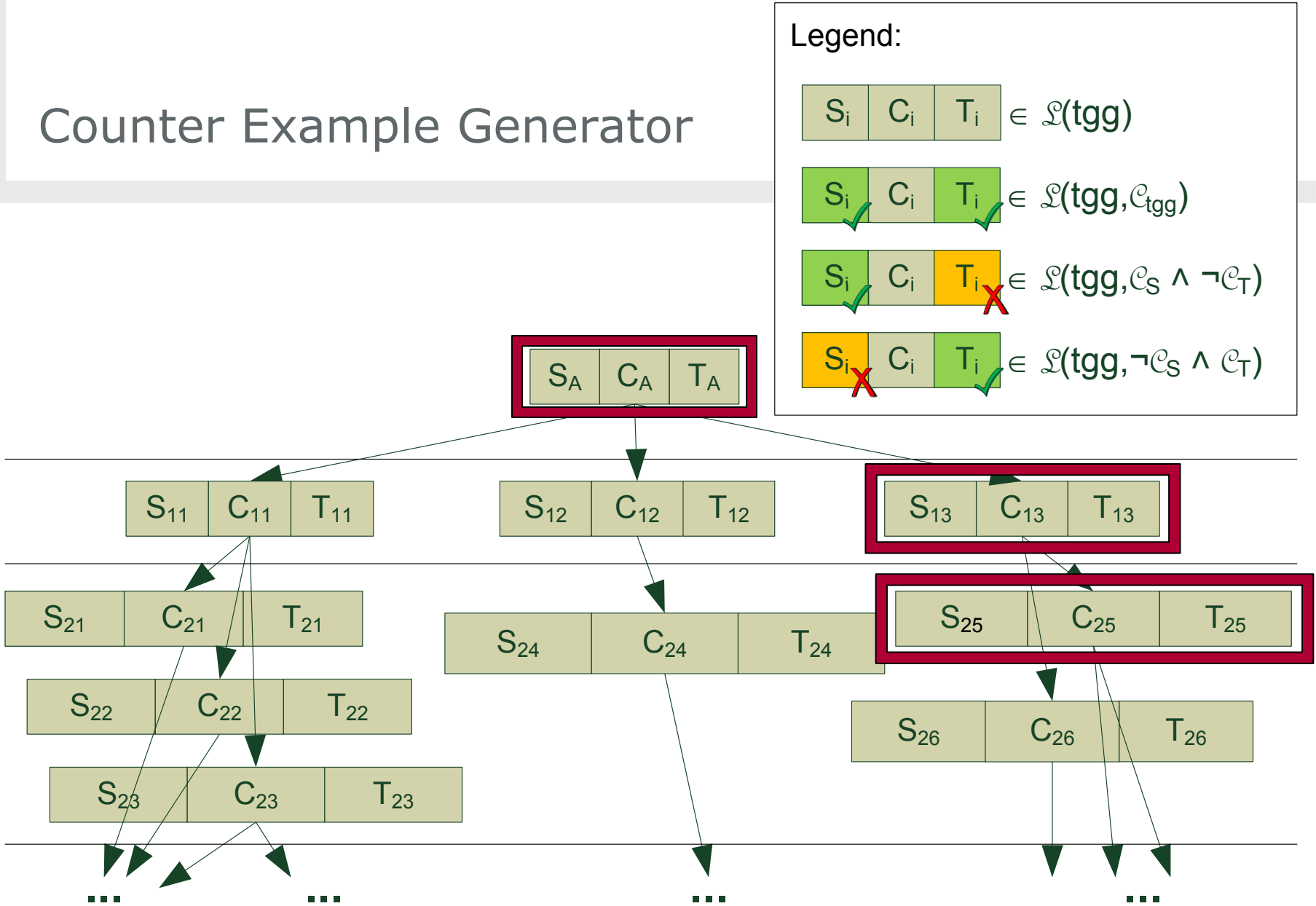
- validate models

Counter Example Generator



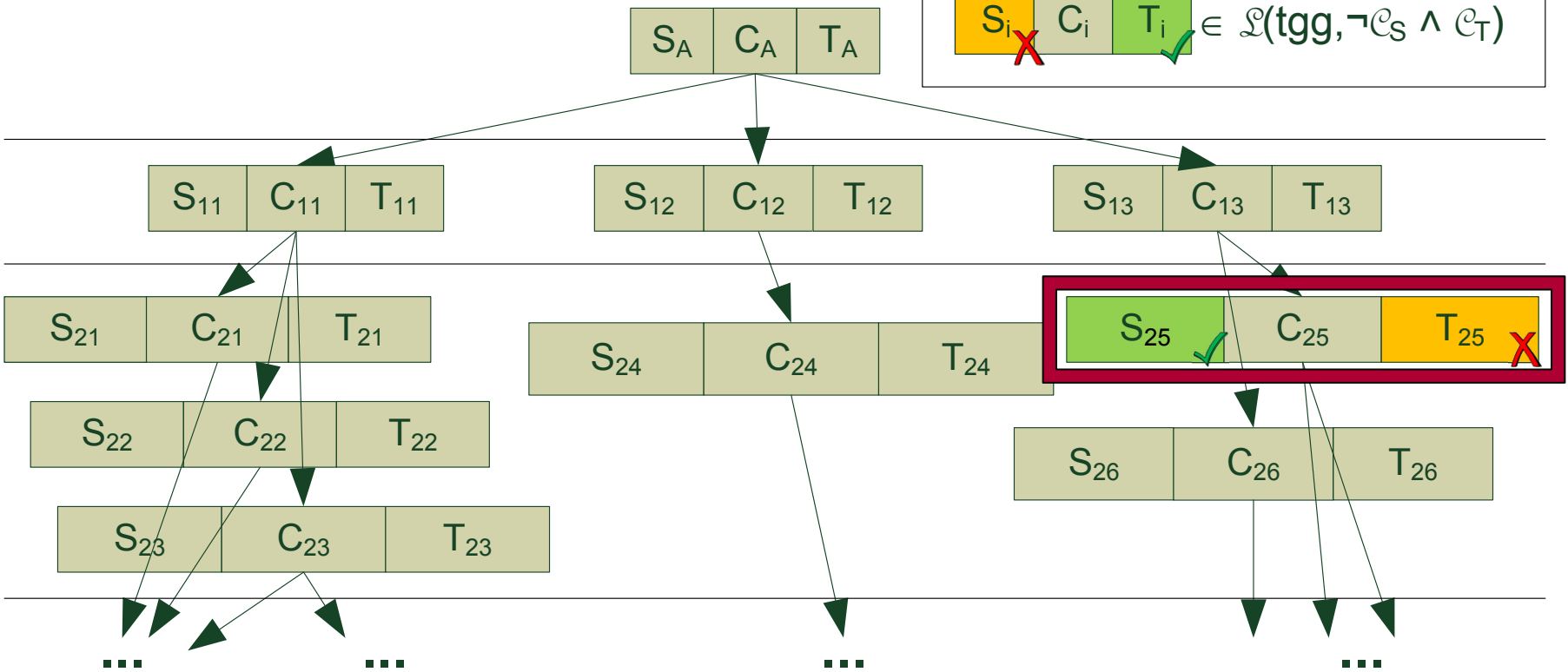
- if both models valid or invalid, generate new models

Counter Example Generator



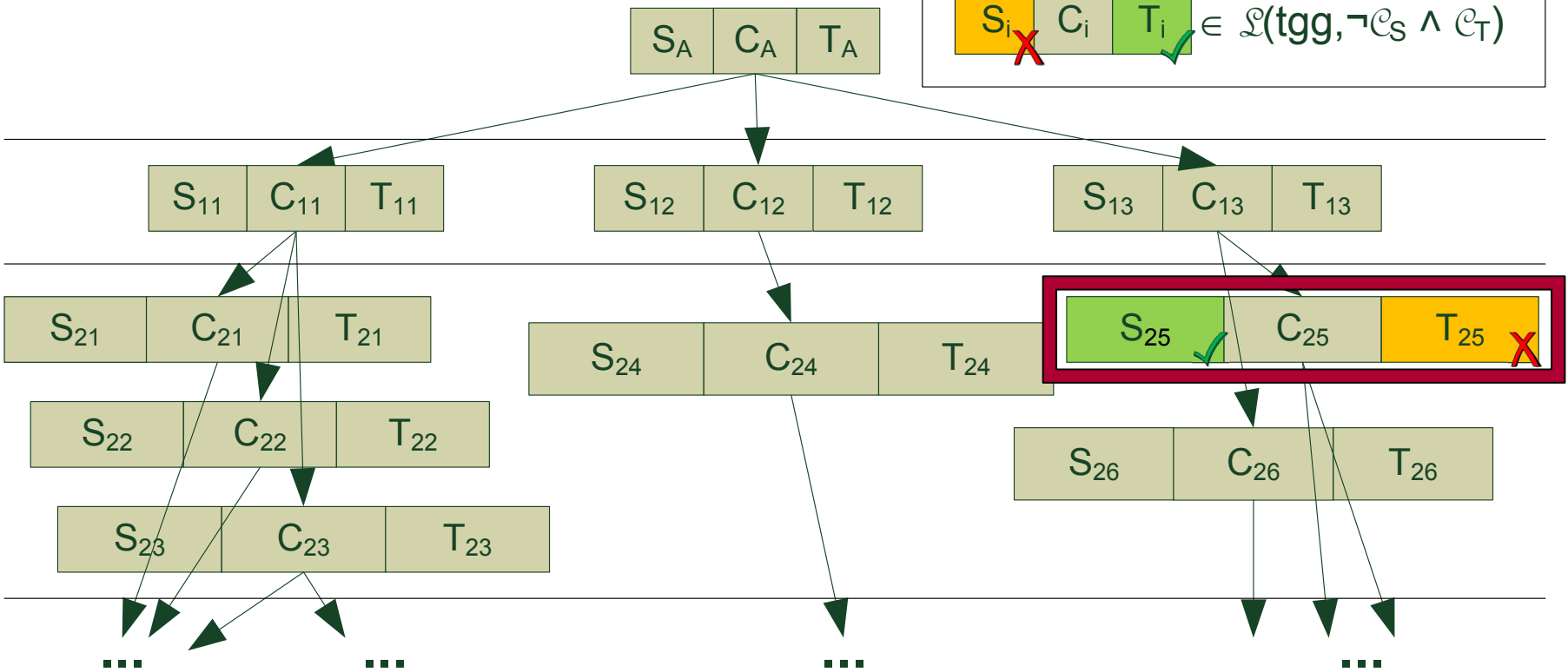
- if both models valid or invalid, generate new models

Counter Example Generator



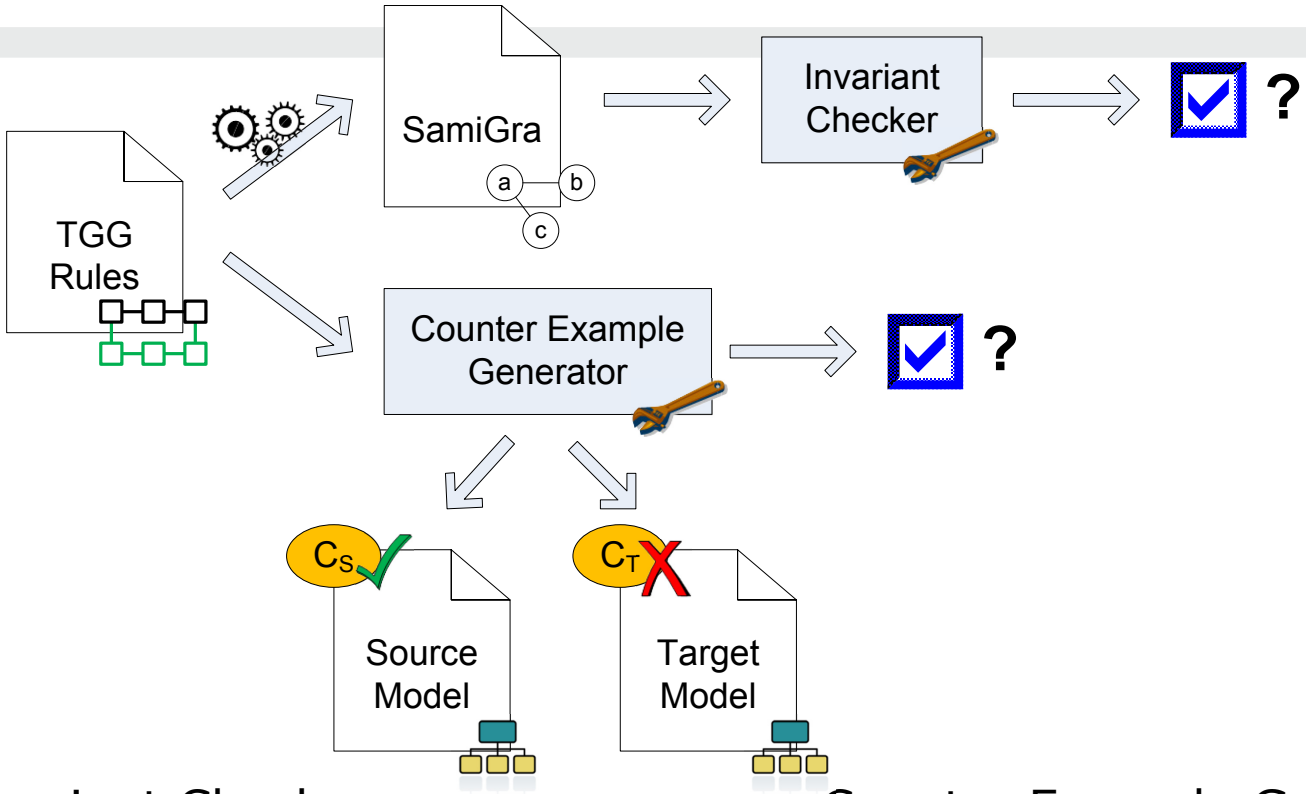
- if both models valid or invalid, generate new models

Counter Example Generator



- until counter example found or maximum number of trials

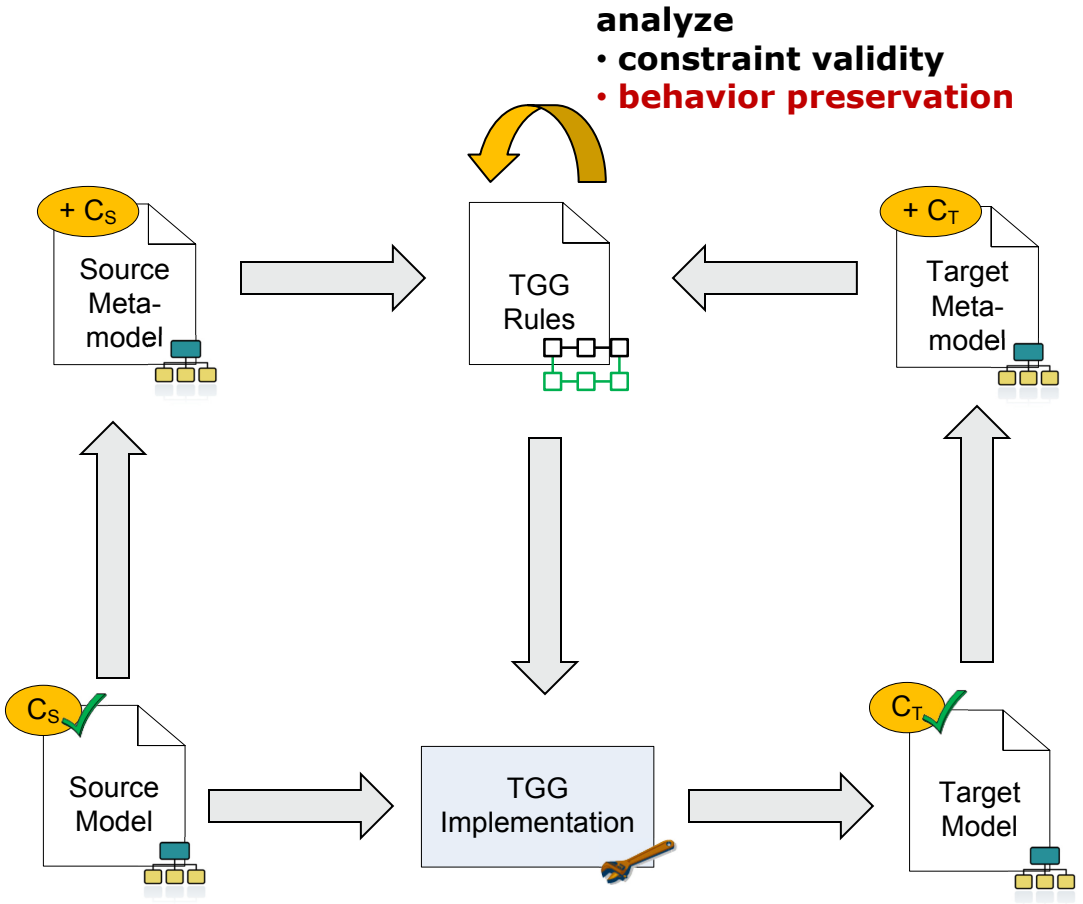
TGG Validation



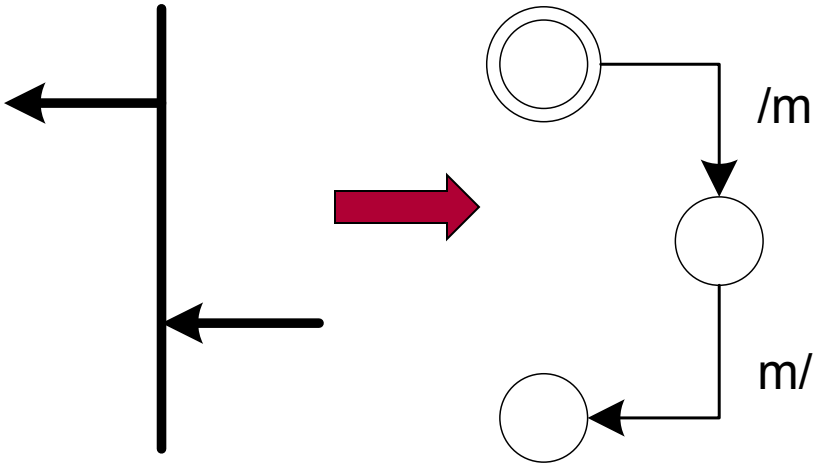
- Invariant Checker
  - complete static analysis
  - translate constrains to graph patterns
- Counter Example Generator
  - no translation of constraints
  - incomplete analysis

Outline

- Overview
- TGG Implementation
- TGG Conformance
- **TGG Analysis**
- Conclusion
- Future Work

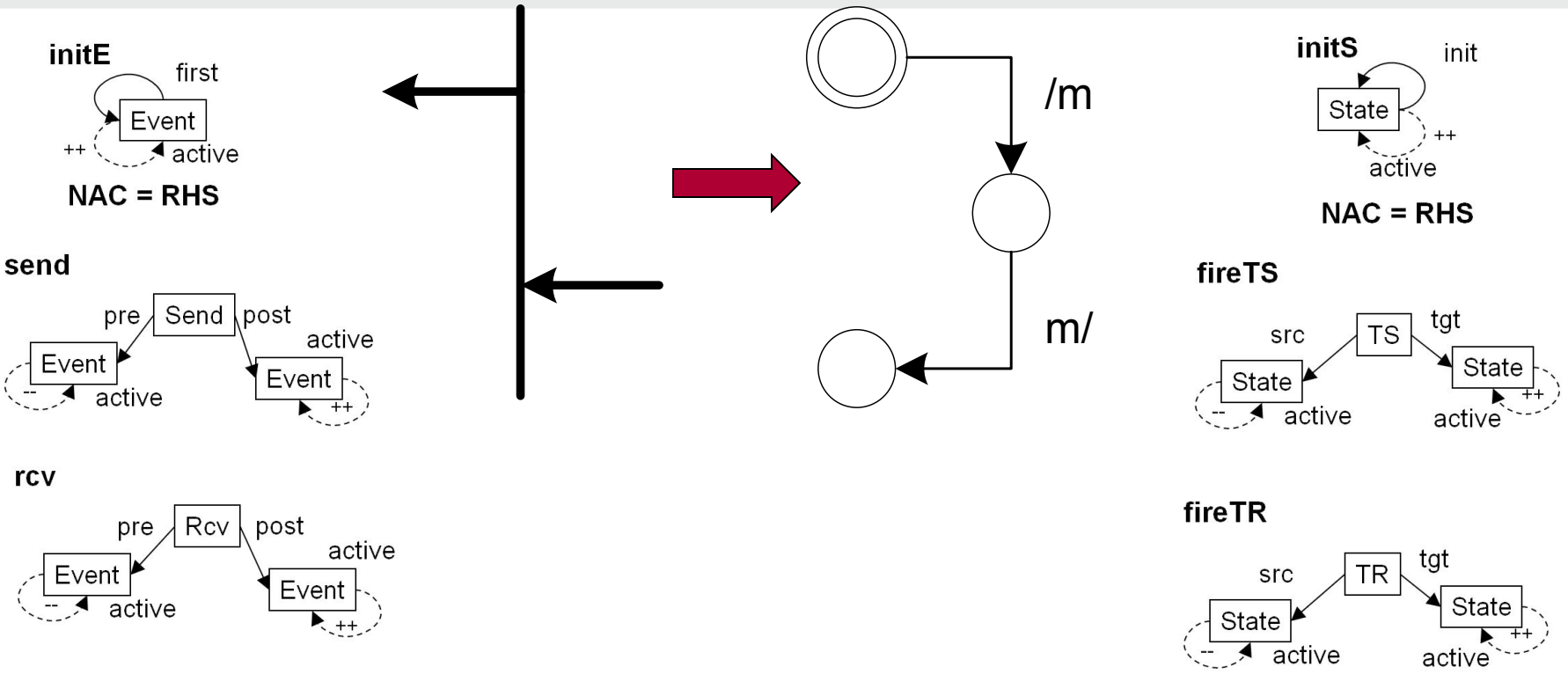


TGG Behavior Preservation Check



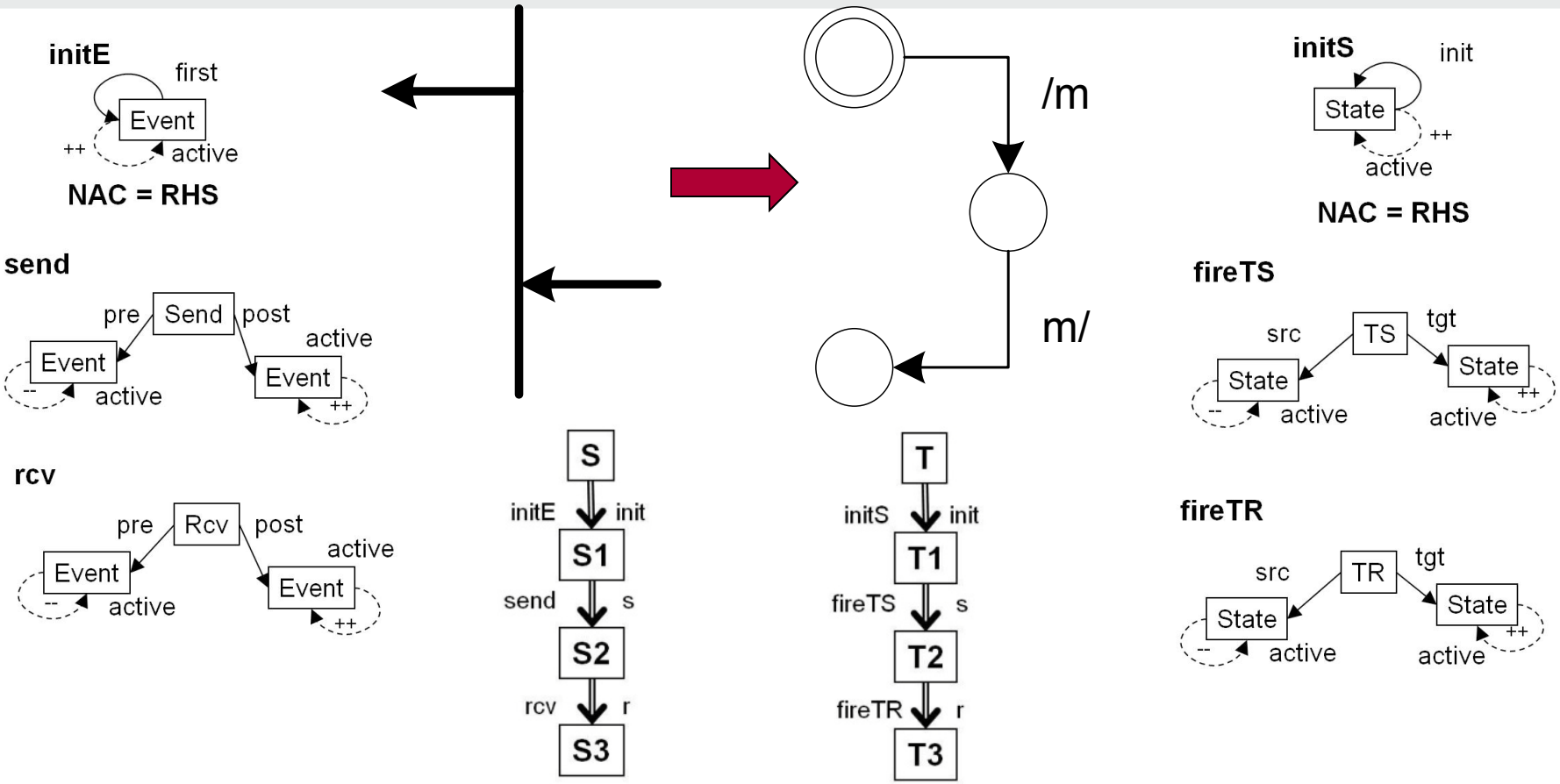
- model transformation should preserve behavior
- static behavior preservation check
- not completely implemented yet

TGG Behavior Preservation Check



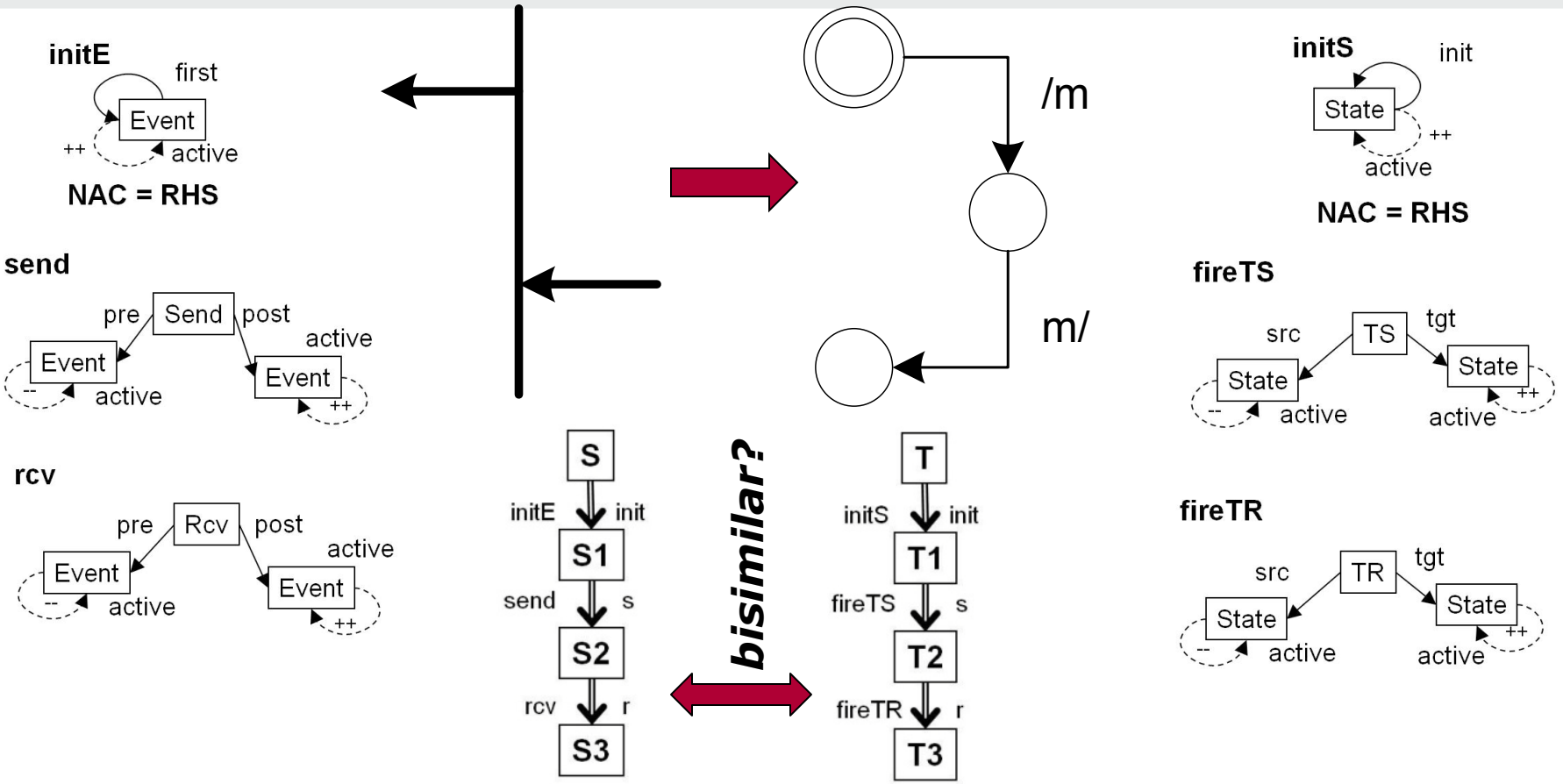
- extend metamodels with runtime information
- define semantics of models using graph transformation rules

TGG Behavior Preservation Check



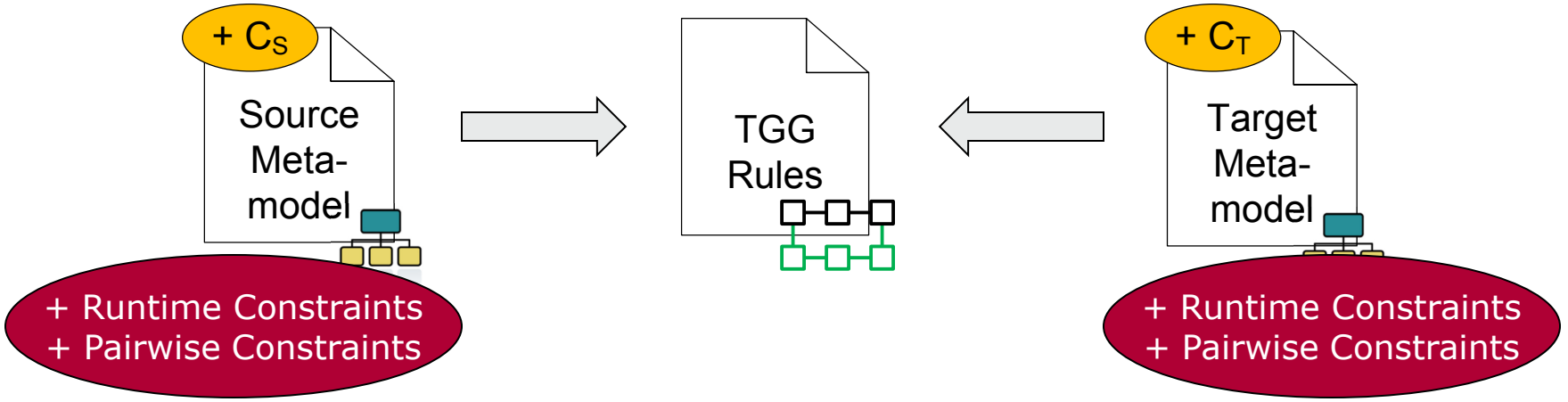
- derive labeled transition systems

TGG Behavior Preservation Check



- equivalent behavior if bisimulation relation exists

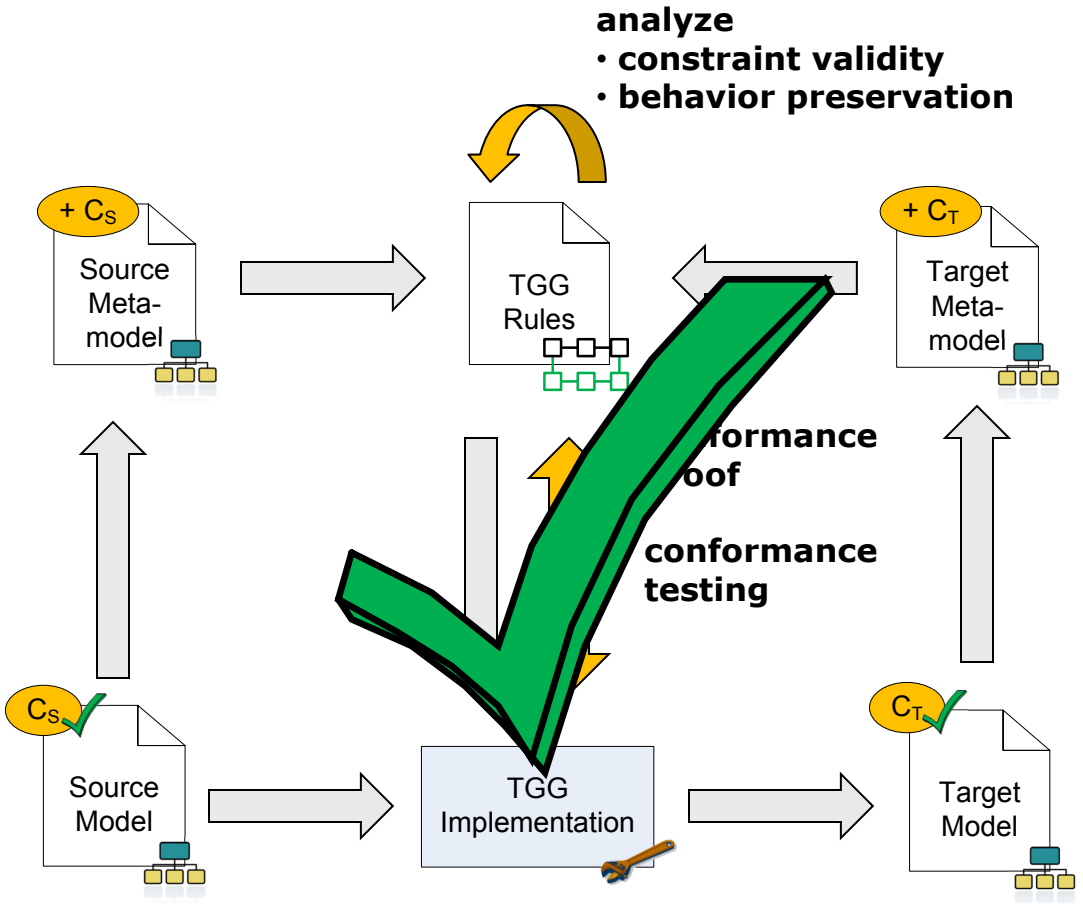
TGG Behavior Preservation Check



- show bisimulation for all LTSs of triple graphs producible by TGG
  - define additional constraints on metamodels
  - use *Invariant Checker* to check constraints
- for more information
  - Giese, Holger and Lambers, Leen *Towards Automatic Verification of Behavior Preservation for Model Transformation via Invariant Checking*, ICGT 2012

Outline

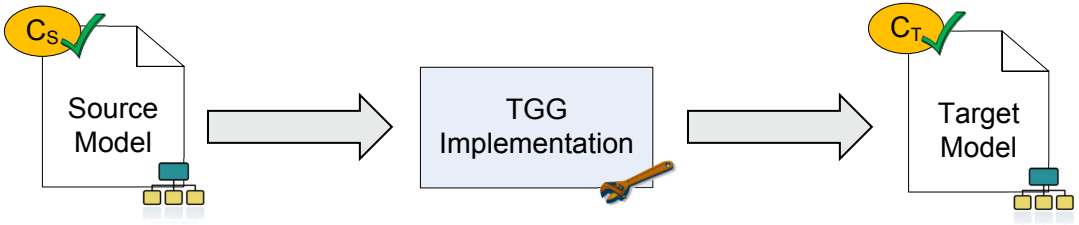
- Overview
- TGG Implementation
- TGG Conformance
- TGG Analysis
- **Conclusion**
- Future Work



Conclusion

- framework allows to
  - perform model transformation/synchronization
  - check/test TGG implementations for conformance
  - check forward/backward validity of TGGs
  - (check behavior preservation)

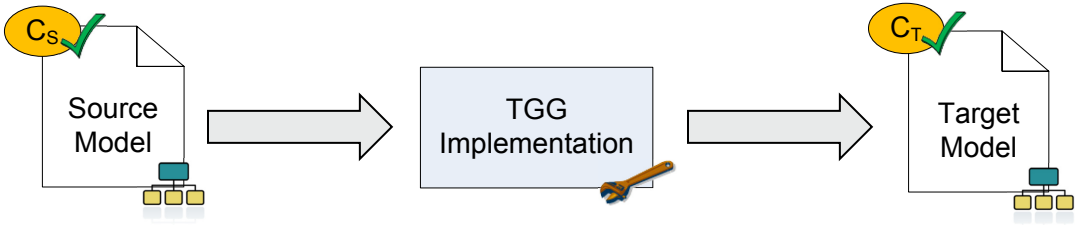
- support development of correct model transformations



Future Work

- TGG validation
  - automatic translation of OCL constraints to graph patterns
  - systematic generation of counter examples (also for test case generation)

- open research questions
  - How to make invalid TGGs valid?



Thank you!