Automatic Verification of Behavior Preservation at the Transformation Level for Relational Model Transformation

Johannes Dyck, Holger Giese, Leen Lambers

Technische Berichte Nr. 112

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 112

Johannes Dyck | Holger Giese | Leen Lambers

Automatic Verification of Behavior Preservation at the Transformation Level for Relational Model Transformation

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2017
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URN urn:nbn:de:kobv:517-opus4-100279
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-100279

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-391-6

The correctness of model transformations is a crucial element for model-driven engineering of high quality software. In particular, behavior preservation is the most important correctness property avoiding the introduction of semantic errors during the model-driven engineering process. Behavior preservation verification techniques either show that specific properties are preserved, or more generally and complex, they show some kind of behavioral equivalence or refinement between source and target model of the transformation. Both kinds of behavior preservation verification goals have been presented with automatic tool support for the instance level, i.e. for a given source and target model specified by the model transformation. However, up until now there is no automatic verification approach available at the transformation level, i.e. for all source and target models specified by the model transformation.

In this report, we extend our results presented in [27] and outline a new sophisticated approach for the automatic verification of behavior preservation captured by bisimulation resp. simulation for model transformations specified by triple graph grammars and semantic definitions given by graph transformation rules. In particular, we show that the behavior preservation problem can be reduced to invariant checking for graph transformation and that the resulting checking problem can be addressed by our own invariant checker even for a complex example where a sequence chart is transformed into communicating automata. We further discuss today's limitations of invariant checking for graph transformation and motivate further lines of future work in this direction.

Contents

1. Introduction

The correctness of model transformations is a crucial element for model-driven engineering of high quality software. Many quality related activities are obtained using the source models of the transformations rather than the results of a single transformation or chains of transformations. Therefore, only if the model transformations work correctly and introduce no faults, the full benefits of working with the higher-level source models can be realized.

In this context in particular *behavior preservation* is the most important correctness property avoiding the introduction of semantic errors during the model-driven engineering process. Behavior preservation verification techniques either show that specific properties are preserved, or more generally and complex, they show some kind of behavioral equivalence or refinement (e.g., bisimulation or simulation [41]) between source and target model of the transformation.

For both kinds of behavior preservation, verification goals have been presented with automatic tool support for the instance level [53, 22, 5, 42, 43, 14], i.e. for a given source and target model specified by the model transformation. Nevertheless, up until now there is no automatic verification approach available at the transformation level, i.e. for *all* source and target models specified by the model transformation. However, as usually the transformation development and the application development that employs the developed transformation are separate activities that are addressed by different people or even different organizations, detecting that the transformation is not correct during application development time is thus usually too late.

Consequently, ensuring behavior preservation for the transformation in general already during the development of the transformation is highly desirable, but to our best knowledge so far no work exists that promises to solve the problem in a fully automated manner. We presented a first approach [25] attacking this problem on the transformation level in a semi-automated manner in form of a verification technique based on interactive theorem proving. Also [36] presented and compared different proof strategies for manual proofs on the transformation level without solving the problem of automation. Some first approaches to automating the verification of behavior preservation for the special case of model refactorings are present [46, 6], but none of them covers the general case of model transformations.

In this report, we present a first overall approach towards automatic verification of behavior preservation for the general case of model transformations specified by triple graph grammars (TGG) and semantic definitions given by graph transformation systems (GTS). In particular, we show that the behavior preservation problem can be reduced to invariant checking for GTS, which in restricted cases can be automatically verified using the existing verification technique [2], similar to [3]

were we reduced the problem of consistency preservation of refactorings accordingly. Due to a mapping of TGGs on specially typed graph transformations [26, 28] both the transformation and the semantics are captured in a homogeneous manner, which greatly facilitates mapping the problem on invariants for GTS. We demonstrate with the complex example of a model transformation from sequence charts to communicating automata how sophisticated the presented approach is. Besides this demonstration for the degree of automation that can be achieved today, we further discuss today's limitations of invariant checking for graph transformation and motivate further lines of future work in this direction.

The report extends the basic first approach presented in [27] (and repeated in Section 3) in several directions: At first, we support not only behavioral equivalences in form of bisimulation, but also behavioral refinement in form of simulation (see later Section 5). Secondly, the improved verification scheme covers not only deterministic semantics but also non-deterministic ones that are necessary to cover non-deterministic sequential behavior or concurrent behavior (see discussion in Section 3.4). Furthermore, we have added an extensive evaluation of our approach as described in Section 7. Finally, compared to the basic first approach in [27] the required amount of manual specification efforts can be substantially reduced (see discussion in Section 6). Moreover in [17] we showed already that the basic verification scheme for relational model transformations presented in [27] and extended here can be adapted such that it is applicable also to operational model transformations.

1.1. Behavior Preservation at the Transformation Level

In this section we introduce the notion of behavior preservation at the transformation level for model transformation in a generic way. We clarify which artefacts are necessary to describe the problem such that later in Section 2 we can formalize these artefacts to tackle the corresponding verification problem. In particular, we need the notions of modeling language, a corresponding semantics for each model in this language, a notion of model transformations, and some notion for behavior comparison to describe the problem.

Definition 1 (Modeling Language $\mathcal{L}$)**.** *A* modeling language $\mathcal{L}$ *consists of a possibly infinite set of models.*

A modeling language can, for example, be defined by a grammar, or by a metamodel enriched with constraints.

Definition 2 (Model Semantics, Semantic Domain $\mathcal{D}$, Semantic Mapping $sem(\cdot)$)**.** *Given a modeling language* $\mathcal{L}$ *and a* semantic domain $\mathcal{D}$*, a* semantic mapping $sem : \mathcal{L} \to \mathcal{D}$ *defines the* semantics $sem(M)$ *of each model* M *in* $\mathcal{L}$*.*

A semantic domain can, for example, be a set of labeled transition systems, but it can also be any other formalism or modeling language, for which the behavior is well-defined.

Definition 3 (Model Transformation MT, Model Transformation Instance $(M_s, M_t) \in \text{MT}$)**.** *Given a source and target modeling language $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$, respectively, a* model transformation MT *is a relation over $\mathcal{L}_\mathcal{S} \times \mathcal{L}_\mathcal{T}$. Each pair (M_s, M_t) of source and target models in* MT *is a* model transformation instance *of the model transformation* MT*.*

A model transformation can, for example, be defined by operational model transformation techniques [44, 37, 12, 24, 49], relational model transformation techniques [47, 44, 40], or hybrid model transformation techniques [4, 44]. Such a definition for a model transformation then implicitly defines all model transformation instances.

In order to be able to compare the semantics of a source and target model, we first have to map the corresponding source and target semantics to the same semantic domain $\mathcal{D}$ by a so-called semantic remapping as introduced in the following definition.

Definition 4 (Semantic Remapping l)**.** *Given a semantic domain $\mathcal{D}_1$ and a semantic domain $\mathcal{D}_2$, a* semantic remapping $l : \mathcal{D}_1 \to \mathcal{D}_2$ *is a mapping from semantic domain $\mathcal{D}_1$ to semantic domain $\mathcal{D}_2$.*

If source and target semantics for a pair (M_s, M_t) of source and target models of modeling language $\mathcal{L}_s$ and $\mathcal{L}_t$ are mapped to the same semantic domain $\mathcal{D}$, we can employ several alternative notions for the behavioral equivalence and refinement (cf. [51, 52] for a plethora of such notions for the case of labeled transition systems).

Definition 5 (Behavioral Equivalence $=_\mathcal{D}$, Behavioral Refinement $\leq_\mathcal{D}$)**.** *Given a shared semantic domain $\mathcal{D}$, we can distinguish two behavioral relations:*

- *A* behavioral equivalence $=_\mathcal{D} \subseteq \mathcal{D} \times \mathcal{D}$ *is a reflexive, symmetric, and transitive relation.*
- *A* behavioral refinement $\leq_\mathcal{D} \subseteq \mathcal{D} \times \mathcal{D}$ *is a reflexive and transitive relation (preorder).*

Remark 1. *The inverse of a refinement in form of a* behavioral abstraction *is also a reflexive and transitive relation (preorder).*

For the in the following outlined notion of behavior preservation for model transformation at the transformation level, we assume that a suitable shared semantic domain $\mathcal{D}$ and a suitable behavioral relation $=_\mathcal{D}$ or $\leq_\mathcal{D}$ will be identified.

Definition 6 (Behavior Preservation – Transformation Level)**.** *Given a model transformation* MT $\subseteq \mathcal{L}_\mathcal{S} \times \mathcal{L}_\mathcal{T}$*, semantic mappings* $sem_\mathcal{S} : \mathcal{L}_\mathcal{S} \to \mathcal{D}_\mathcal{S}$ *and* $sem_\mathcal{T} : \mathcal{L}_\mathcal{T} \to \mathcal{D}_\mathcal{T}$ *for source and target language* $\mathcal{L}_\mathcal{S}$ *and* $\mathcal{L}_\mathcal{T}$*, and semantic remappings* $l_s : \mathcal{D}_\mathcal{S} \to \mathcal{D}$ *and* $l_t : \mathcal{D}_\mathcal{T} \to \mathcal{D}$*, we say that the* model transformation MT *is (2.1)* behavior preserving in an equivalent manner *if for each pair of source and target models* $(M_s, M_t) \in$ MT*, it holds that* $l_s(sem_\mathcal{S}(M_s)) =_\mathcal{D} l_t(sem_\mathcal{T}(M_t))$*. We say that* MT *is (2.2)* behavior preserving in a refining manner *if for each pair of source and target models* $(M_s, M_t) \in$ MT*, it holds that* $l_t(sem_\mathcal{T}(M_t)) \leq_\mathcal{D} l_s(sem_\mathcal{S}(M_s))$*.*

Remark 2 (Behavior Preservation – Instance Level)**.** *In contrast to behavior preservation at the transformation level applying to the* transformation as a whole *consisting in general of infinitely many model transformation instances, for behavior preservation at the instance level it is enough that condition (2.1) or (2.2) holds for* just one *such a pair of source and target models* $(M_s, M_t) \in$ MT*. In this case we say that the model transformation* instance (M_s, M_t) *is behavior preserving in an equivalent or refining manner, respectively.*

Remark 3 (Abstraction)**.** *Since abstraction is the opposite case of refinement, we also say that a model transformation* MT *represents an abstraction if its inverse defines a model transformation, which is behavior preserving in a refining manner. In particular, this means that for each pair of source and target models* $(M_s, M_t) \in$ MT *it needs to hold that* $l_s(sem_\mathcal{S}(M_s)) \leq_\mathcal{D} l_t(sem_\mathcal{T}(M_t))$*.*

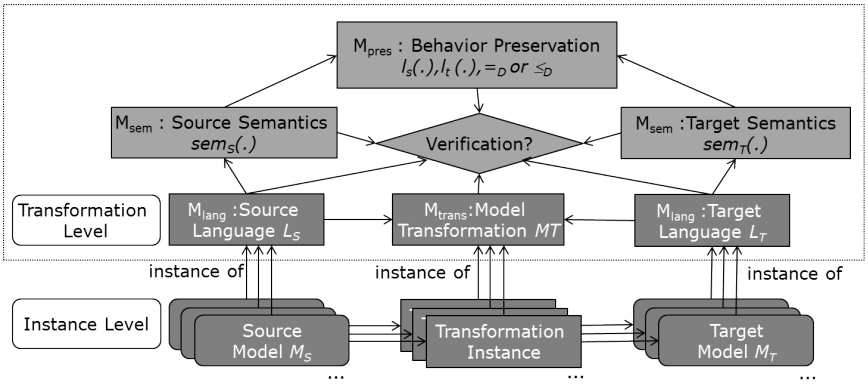


Figure 1: Behavior Preservation – Transformation Level (see Definition 6)

In Figure 1 it is summarized which general concepts are employed in Def. 6 that need to be further formalized in some corresponding modeling step M_x such that

a verification technique can be developed on the transformation level. Note that a model transformation MT is behavior preserving in an equivalent or refining manner if and only if *all* model transformation instances $(M_s, M_t) \in \text{MT}$ are behavior preserving in an equivalent or refining manner, respectively.

Example 1 (Running example)**.** *For the purpose of demonstrating our verification techniques for behavior preservation, we will in the following consider model transformations (*MT*) involving different variants of sequence charts as the source modeling language ($\mathcal{L}_\mathcal{S}$) and automata as the target modeling language ($\mathcal{L}_\mathcal{T}$) as our running examples. We will later extend this example by concrete definitions for source and target languages $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$ (modeling step M_{lang}), for semantic mappings $sem_\mathcal{S}$ and $sem_\mathcal{T}$ (modeling step M_{sem}), for a model transformation* $\text{MT} \subseteq \mathcal{L}_\mathcal{S} \times \mathcal{L}_\mathcal{T}$ *(modeling step M_{trans}), and for remappings l_s and l_t and behavioral equivalence and refinement (modeling step M_{pres}) such that all concepts occurring in Definition 6 are explicitly modeled according to our proposed formalization.*

1.2. State of the Art

The verification of model transformations is an active area of research and consequently a number of approaches have been developed. As we approach in this report the case of exogenous model transformations, we will further limit our discussion mainly to related work for this kind of transformations.

In general the properties that are verified for model transformations could be of syntactical or semantical nature. Syntactical properties are, for example, well-formedness constraints that are guaranteed for a transformation result in case other well-formedness constraints are guaranteed for the input. Semantical properties refer to the semantics of the source and target model and include as the most general case behavior preservation as studied in this report.

Finally, we can target the verification of properties at the instance level or at the transformation level (as in Definition 6). While in the former we study the problem for a pair of source and target model, in the latter we consider all potentially infinitely many source and target models that are linked by the transformation at once.

Ensuring syntactical properties for model transformation instances is usually rather straight forward, as we can simply check them inspecting the source and target model.

The correctness of semantical properties for model transformation instances in contrast is more complicated. The techniques for the verification of model transformation instances [5, 22, 42, 43, 14, 53] assign the source and target model a formal semantics and then proof via model checking (cf. [53, 22]) or bisimulation checking (cf. [5, 42, 43]) the semantic equivalence.

The verification of properties at the transformation level as in Definition 6 is more demanding than at the instance level as we have to consider all potentially infinitely many source and target models that are linked by the transformation at once.

For syntactical properties there exist, however, already first results: In [9] an approach for ATL has been developed that based on bounded verifiers check if constraints on the source and target model can be violated by the transformation. For constraints in form of OCL invariants [10] permits to check for relational QVT und TGG based on a bounded verifiers check if constraints on the source and target model can be violated by the transformation. However, for all approaches it holds that the results are incomplete in the sense that due to the bounded search space counter examples may be missed. In our own related work [3] we developed an approach that preserves a specific kind of graph constraints for refactorings described by graph transformations that is in contrast complete. For model transformations with triple graph grammars we also present related results concerning the preservation of syntactical properties in [35].

For semantical properties including the in this report studied behavior preservation for model transformation at the transformation level (see Definition 6), only first approaches exist: In [25] we developed an approach to verify behavior preservation for model transformations specified by TGGs with the theorem prover Isabelle/HOL where the elements as outlined in Section 1.1 are encoded in the logic of the theorem prover. In [36] different proof strategies for the verification of model transformations at the model transformation level are suggested, but these strategies do no solve the question how to automate the verification.

Some initial ideas for the automation of behavior preservation for refactorings exist [46, 6], however, these ideas do not cover the more general exogenous model transformations as studied in this report.

Table 1: Summary of existing fully automated approaches

property \ level	instance level	transformation level
syntactical	trivial	[9, 10][3, 35][1]
semantical	[53, 22, 5, 42, 43, 14]	[27][1]

The review of the related work revealed as depicted in Table 1 that for verification of the behavior preservation for model transformations at the transformation level no fully automatic and sound approach besides our own work for relational model

transformations [27] as well as operational model transformations [17] exists so far. This transformation level approach however is still rather limited compared to the presented solution for relational model transformations in this report as outlined on page 10. Not even an incomplete approach that is fully automatic has been suggested so far, which demonstrates how difficult behavior preservation verification on the transformation level is.

1.3. Outline

The rest of the report is structured as follows: In Section 2, we introduce the basic formal notions that we require to tackle the behavior preservation problem on the transformation level. In particular, for formalizing modeling languages (modeling step M_{lang}) and its behavioral semantics (modeling step M_{sem}) as well as model transformations (modeling step M_{trans}) we will rely on so-called (typed) graph constraints, GTSs and TGGs. Moreover, we will reintroduce the notion of bisimulation and simulation as a formal notion for behavioral equivalence and refinement (modeling step M_{pres}). Modeling steps M_{lang}, M_{sem}, M_{trans} and M_{pres} compose into a so-called *modeling scheme* such that when this scheme is complete the notion of behavior preservation at the transformation level is formalized in such a way that a corresponding *verification scheme* can be developed. Indeed as described in our previous work [27] already, in Section 3 we present such a verification scheme. In particular, we describe how the problem of verifying behavior preservation for relational model transformations at the transformation level can be reduced to invariant checking GTSs and prove its correctness. We moreover explain why our previous work [27] can only handle modeling languages with a deterministic semantics that only distinguishes finitely many labels and what limitations this implies. In Section 4 we state the problem of behavior preservation for modeling languages that do not have these limitations on their semantics and we show how the verification scheme can be generalized. Moreover, in Section 5 we handle the case of model transformations for which behavior is refined by the model transformation rather than kept equivalent by requiring a simulation relation between source and target semantics. Afterwards, the automation for a restricted class of the three introduced variants of the problem is presented in Section 6 and thereafter evaluated in Section 7. In Section 8 we discuss the appropriateness, applicability, and limitations concerning the automation of the approach. The report closes with a final conclusion and outlook on future work.

[1]This particular work is a predecessor of the presented work.

2. Formalization

In order to formalize the artifacts modeling language (Section 2.1), model semantics (Section 2.2) and model transformation (Section 2.3) of Definition 6, we reintroduce graph conditions, graph transformation and TGGs in a compact way and refer to [32, 19, 28] for more detailed definitions and explanations. We also reintroduce the notion of labeled transition systems, relabelings of labeled transition systems, and bisimulation/simulation (Section 2.4), since it will be employed to formalize the semantic domain and the notion of behavioral equivalence and refinement. In Section 2.5 we then summarize how to refine Definition 6 with these formal concepts.

2.1. Modeling Language

To formalize the modeling language according to Definition 1, we will assume in this report that each model of a modeling language is represented by a graph. Graphs can be equipped with *typing* over a given type graph TG as usual [18] by adding a so-called typing morphism from each graph to TG. Such a typing morphism is a regular graph morphism from the graph G to be typed into the type graph TG, expressing to which type node/edge in TG each node/edge in G, resp., is being mapped. In the following, we formally define the notions of graphs, graph morphisms and type graphs.

Definition 7 (Graph [18])**.** *A* graph $G = (V, E, s, t)$ *consist of a set V of nodes (also called vertices), a set E of edges, and two mappings $s, t : E \to V$, the source and target mappings, respectively.*

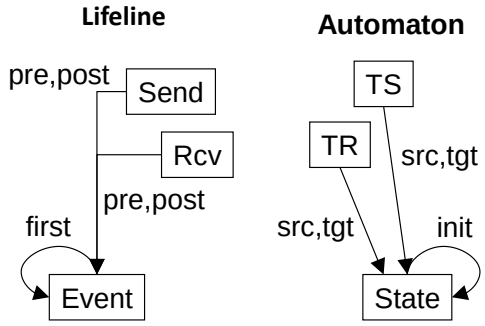
Definition 8 (Graph morphism [18])**.** *Given graphs G_1, G_2 with $G_i = (V_i, E_i, s_i, t_i)$ for $i = 1, 2$, a* graph morphism $f : G_1 \to G_2$, $f = (f_V, f_E)$ *consists of two mappings $f_V : V_1 \to V_2$ and $f_E : E_1 \to E_2$ that preserve the source and target mappings, i.e. $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$.*

Definition 9 (Typed graph and typed graph morphism [18])**.** *A* type graph *is a distinguished graph $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG})$. V_{TG} and E_{TG} are called the vertex and the edge type alphabets, respectively. A tuple $(G, type)$ of a graph G together with a graph morphism $type : G \to TG$ is then called a* typed graph*. Given typed graphs $G_1^T = (G_1, type_1)$ and $G_2^T = (G_2, type_2)$, a* typed graph morphism $f : G_1^T \to G_2^T$ *is a graph morphism $f : G_1 \to G_2$ such that $type_2 \circ f = type_1$.*

With these definitions, a graph language consisting of graphs typed over a common type graph can be defined.

Definition 10 (Graph language)**.** *Given a type graph* TG*, the* graph language $\mathcal{L}(TG)$ *denotes the set of all graphs typed over TG, i.e.* $\mathcal{L}(\text{TG}) = \{G \mid \exists\, type\ (type : G \to \text{TG})\}$.

In the following, all specific graphs encountered throughout this report will be typed graphs, unless noted otherwise, and the typing morphism will be omitted. Likewise, all specific graph morphisms will be typed graph morphisms.



(a) Type graph S_{TT} (b) Type graph T_{TT}

Figure 2: Type graphs S_{TT} and T_{TT}

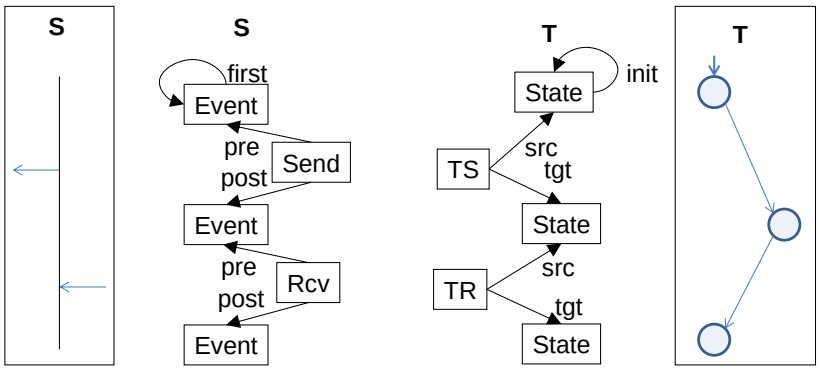


Figure 3: S and T in concrete and abstract syntax

Example 2 (Graph language as modeling language)**.** *In Fig. 2(a), a type graph S_{TT} is depicted used to describe the language of message sequence charts with one lifeline being able to send and receive messages, which is the source modeling language for an example model transformation introduced later. The graph S in Fig. 3 describes a concrete sequence chart with one lifeline, depicted on its left-hand side also in concrete syntax. This graph S is typed over S_{TT} via a typing morphism mapping nodes and edges with a specific label in S to a node type and edge in S_{TT} with the same label, respectively.*

Analogously, the graph T in Fig. 3, which is typed over T_{TT} (Fig. 2(b)), describes a specific automaton with three states and two different types TS and TR of transitions. The language of automata is the target modeling language for the example model transformation.

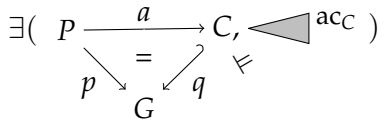
The language $\mathcal{L}(S_{TT})$ of all sequence charts with one lifeline still contains malformed models. In particular, charts are allowed with events that are connected to more than one previous message. Instead, we only want to allow charts such that for each event there exists at most one unique previous (or subsequent) message.

A *modeling language* could be defined as $\mathcal{L}(\mathrm{TG})$ for a given type graph TG, but as shown in Example 2 we usually need a possibility to further constrain this set of typed graphs to describe the modeling language more precisely. Graph constraints, derived from graph conditions as explained in the following, are the right formalism to further constrain $\mathcal{L}(\mathrm{TG})$.

Graph conditions [32, 19] generalize the corresponding notions in [31], where a negative (positive) application condition, NAC (PAC) for short, over a graph P, denoted $\neg\exists a$ ($\exists a$) is defined in terms of a graph morphism. Informally, a morphism $p : P \to G$ satisfies $\neg\exists a$ ($\exists a$) if there does not exist a morphism $q : C \to G$ extending p (if there exists q extending p). Then, a *(nested) graph condition AC* is either the special condition true or a pair of the form $\neg\exists(a, \mathrm{ac}_C)$ or $\exists(a, \mathrm{ac}_C)$, where the first case corresponds to a NAC and the second to a PAC, and in both cases ac_C is an additional AC on C. Intuitively, a morphism $p : P \to G$ satisfies $\exists(a, \mathrm{ac}_C)$ if p satisfies a and the corresponding extension q satisfies ac_C. ACs (and also NACs and PACs) may be combined with the usual logical connectors. A morphism $p\colon P \to G$ satisfies $\neg c$ if p does not satisfy c and satisfies $\wedge_{i\in I} c_i$ if it satisfies each c_i ($i \in I$).

Definition 11 (graph condition [19])**.** *A* graph condition*, also called* nested graph condition*, is inductively defined as follows:*

1. *For every graph P,* true *is a graph condition over P.*
2. *For every morphism $a : P \to C$ and every graph condition ac_C over C, $\exists(a, \mathrm{ac}_C)$ is a graph condition over P.*
3. *For graph conditions* ac, ac_i *over P with i in an index set I,* $\neg$ac *and* $\bigwedge_{i\in I} \mathrm{ac}_i$ *are graph conditions over P.*



Satisfiability *of graph conditions is inductively defined as follows:*

1. *Every morphism satisfies true.*
2. *A morphism* $p : P \to G$ *satisfies* $\exists(a, \mathrm{ac}_C)$ *over* P *with* $a : P \to C$ *if there exists an injective morphism* $q : C \hookrightarrow G$ *such that* $q \circ a = p$ *and* q *satisfies* ac_C.
3. *A morphism* $p : P \to G$ *satisfies* $\neg\mathrm{ac}$ *over* P *if* p *does not satisfy* ac *and* p *satisfies* $\bigwedge_{i\in I} \mathrm{ac}_i$ *over* P *if* p *satisfies each* ac_i $(i \in I)$.

We write $p \vDash \mathrm{ac}$ *to denote that the morphism* p *satisfies* ac.

Graph conditions can be equipped with typing *over a given type graph* TG *as usual [18] by adding typing morphisms from each graph to* TG *and by requiring type-compatibility with respect to* TG *for each graph morphism.*

Graph conditions over the empty graph I *are also called* graph constraints. *A graph* G *satisfies a graph constraint* ac_I, *written* $G \vDash \mathrm{ac}_I$, *if the initial morphism* $i_G : I \to G$ *satisfies* ac_I.

Notation. *Note that* $\exists a$ *abbreviates* $\exists(a, \mathrm{true})$, $\forall(a, \mathrm{ac}_C)$ *abbreviates* $\neg\exists(a, \neg\mathrm{ac}_C)$ *and* $\exists(C, \mathrm{ac}_C)$ *abbreviates* $\exists(i_C, \mathrm{ac}_C)$ *with the initial morphism* $i_C : I \to C$. *More in general, a graph condition* $\exists(a, \mathrm{ac}_C)$ *over* P *with* $a : P \to C$, *is abbreviated also as* $\exists(C, \mathrm{ac}_C)$ *if* a *denotes an inclusion morphism and if it is clear from the context that* $\exists(C, \mathrm{ac}_C)$ *is a condition over* P. *Moreover, the depiction of specific constraints shown in this report (cf. Figure 4) takes into account that they may relate specifically to the source or target modeling language or traceability information of a model transformation. Hence, dashed vertical lines will be used to separate elements of the respective languages, with elements to the left (right, middle) being part of the source (target, traceability information), respectively. In particular, for our formalization of model transformations as triple graph grammars (see Section 2.3), this traceability information will be represented by correspondence nodes and links between source and target model elements.*

The idea behind graph constraints, as opposed to graph conditions, is to have a way of describing properties on the level of graphs instead of morphisms. For example, we can restrict the set of graphs in a graph language $\mathcal{L}(\mathrm{TG})$ defined by a type graph to those graphs also satisfying a constraint $\mathcal{C}$.

Definition 12 (Graph language with constraint)**.** *Given a type graph* TG *and a graph constraint* $\mathcal{C}$ *typed over* TG, *the* graph language with constraint $\mathcal{L}(\mathrm{TG}, \mathcal{C})$ *denotes the set of all graphs typed over* TG *that satisfy* $\mathcal{C}$, *i.e.* $\mathcal{L}(\mathrm{TG}, \mathcal{C}) = \{G \mid \exists\, type\ (type : G \to \mathrm{TG}) \wedge G \vDash \mathcal{C}\}$.

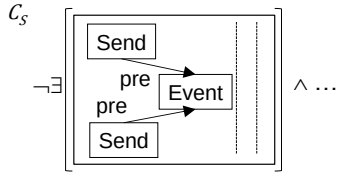


Figure 4: Fragment of $\mathcal{C}_\mathcal{S}$

Example 3 (Graph language with constraint as modeling language)**.** *In Fig. 4, a graph constraint is depicted expressing that an event is connected with at most one unique previous* Send *message. The dashed lines show that the constraint only contains and concerns elements relevant for the source modeling language. Analogously, we can define constraints for subsequent as well as* Rcv *messages. We denote the conjunction of all these constraints with $\mathcal{C}_\mathcal{S}$. The language $\mathcal{L}(S_{TT}, \mathcal{C}_\mathcal{S})$ then describes the (source modeling) language of sequence charts consisting of only one lifeline more accurately than as introduced without constraint in Example 2.*

Likewise, the language $\mathcal{L}(T_{TT}, \mathcal{C}_\mathcal{T})$ for a similar constraint $\mathcal{C}_\mathcal{T}$ and the type graph T_{TT} describes the (target modeling) language of automata.

2.2. Model Semantics

In this report, we use graph transformation and induced labeled transition systems to formalize model semantics according to Definition 2. We start with reintroducing graph transformation and thereby assume the double-pushout approach (DPO) to graph transformation with injective matching [18]. In particular, we allow rules to be equipped with application conditions (AC) [32, 19], allowing to apply a given rule to a graph G only if the corresponding match morphism satisfies the AC of the rule.

Definition 13 (graph transformation)**.** *A plain graph transformation rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$ consists of a span of injective graph morphisms. We say that the graphs L and R are the left-hand side (LHS) and right-hand side (RHS) of the rule, respectively. A* graph transformation rule *$\rho = \langle p, \mathrm{ac}_L \rangle$ consists of a plain rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$ and an application condition ac_L over L.*

$$\begin{array}{ccccc} \mathrm{ac}_L & L & \hookleftarrow & I & \hookrightarrow & R \\ & \Vdash & & & & \\ & m \downarrow & (1) & \downarrow & (2) & \downarrow m^* \\ & G & \hookleftarrow & D & \hookrightarrow & G' \end{array}$$

A direct graph transformation *via rule* $\rho = \langle p, \mathrm{ac}_L \rangle$ *consists of two pushouts (1) and (2), called DPO, with injective match m and comatch* m^* *such that* $m \vDash \mathrm{ac}_L$*. If there exists a direct transformation from G to* G' *via rule* ρ *and match m, we write* $G \Rightarrow_{m,\rho} G'$*. If we are only interested in the rule* ρ*, we write* $G \Rightarrow_{\rho} G'$*. If a rule* ρ *in a set of rules* $\mathcal{R}$ *exists such that there exists a direct transformation via rule* ρ *from G to* G'*, we write* $G \Rightarrow_{\mathcal{R}} G'$*. A* graph transformation*, denoted as* $G_0 \Rightarrow^* G_n$*, is a sequence* $G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n$ *of* $n \geq 0$ *direct graph transformations.*

Rules and transformations as described before can be equipped with *typing* over a given type graph TG as usual [18] by adding typing morphisms from each graph to TG and by requiring type-compatibility with respect to TG for each graph morphism.

Definition 14 (graph transformation system (with constraint), set of reachable graphs)**.** *A* graph transformation system *(GTS)* $\mathrm{gts} = (\mathcal{R}, \mathrm{TG})$ *consists of a set of rules* $\mathcal{R}$ *typed over a type graph* TG*. A graph transformation system may be equipped with an initial graph* G_0 *or a set of initial graphs I being graphs typed over* TG*. If a rule* ρ *in the set of rules* $\mathcal{R}$ *of* gts *exists such that there exists a typed direct transformation via rule* ρ *from G to* G'*, we write* $G \Rightarrow_{\mathrm{gts}} G'$*. For a GTS* $\mathrm{gts} = (\mathcal{R}, \mathrm{TG})$ *and an initial graph* G_0 *the* set of reachable graphs $\mathit{REACH}(\mathrm{gts}, G_0)$ *is defined as* $\{G \mid G_0 \overset{*}{\Rightarrow}_{\mathrm{gts}} G\}$*. A* GTS with constraint $\mathrm{gts}^{\mathcal{C}} = (\mathcal{R}, \mathrm{TG}, \mathcal{C})$ *consists of a GTS* $\mathrm{gts} = (\mathcal{R}, \mathrm{TG})$ *and a constraint* $\mathcal{C}$ *typed over* TG*. If a rule* ρ *in the set of rules* $\mathcal{R}$ *of* $\mathrm{gts}^{\mathcal{C}}$ *exists such that there exists a typed direct transformation via rule* ρ *from G to* G' *both satisfying* $\mathcal{C}$*, we write* $G \Rightarrow_{\mathrm{gts}^{\mathcal{C}}} G'$*. For a GTS with constraint* $\mathrm{gts}^{\mathcal{C}} = (\mathcal{R}, \mathrm{TG}, \mathcal{C})$ *and an initial graph* G_0 *satisfying* $\mathcal{C}$ *the* set of reachable graphs $\mathit{REACH}(\mathrm{gts}^{\mathcal{C}}, G_0)$ *is defined as* $\{G \mid G_0 \overset{*}{\Rightarrow}_{\mathrm{gts}^{\mathcal{C}}} G\}$*.*

The satisfaction of graph constraints can be invariant with respect to a GTS. In particular, in our verification approach, we reduce the problem of behavior preservation to invariant checking. In Section 6, we explain how and with which restrictions automatic invariant checking can be performed statically.

Definition 15 (inductive invariant [11])**.** *A graph constraint* ac_I *is an* inductive invariant *of the GTS* $\mathrm{gts} = (\mathcal{R}, \mathrm{TG})$*, if for all graphs G in* $\mathcal{L}(TG)$ *such that* $G \vDash \mathrm{ac}_I \wedge\ G \Rightarrow_{\mathrm{gts}} G'$ *it holds that* $G' \vDash \mathrm{ac}_I$*. A graph constraint* ac_I *is an* inductive invariant *of the GTS* with constraint $\mathrm{gts}^{\mathcal{C}} = (\mathcal{R}, \mathrm{TG}, \mathcal{C})$*, if for all graphs G in* $\mathcal{L}(TG, \mathcal{C})$ *such that* $G \vDash \mathrm{ac}_I \wedge\ G \Rightarrow_{\mathrm{gts}^{\mathcal{C}}} G'$ *it holds that* $G' \vDash \mathrm{ac}_I$*.*

Remark 4. *In the latter case it holds that* $G' \in \mathcal{L}(\mathrm{TG}, \mathcal{C})$*, since it results as a graph from a rule application via the GTS with constraint* $\mathrm{gts}^{\mathcal{C}}$ *from a graph G satisfying* $\mathcal{C}$*.*

Analogous to the work of Hülschbusch et al. [36], we will define a semantic mapping for a modeling language defined by a graph language using graph transformation systems. The basic idea here is to define the semantics by means of an

interpreter which is described by graph transformation rules, which operate on the basis of the model and additional runtime information capturing the current state in the execution of the model. The idea to employ GTS to define some form of interpreter for diagrams has been also suggested by the dynamic metamodeling approach [21, 33] and has been used for the instance-level verification in [22]. In order to be able to encode runtime information into a graph language the according type graphs can be enriched with so-called *dynamic types* allowing to define dynamic elements and possible changes of dynamic elements in instances of this *runtime type graph*. In this context, we say that a type (or corresponding instance element) is *static* if it is not a dynamic type (or element), respectively.

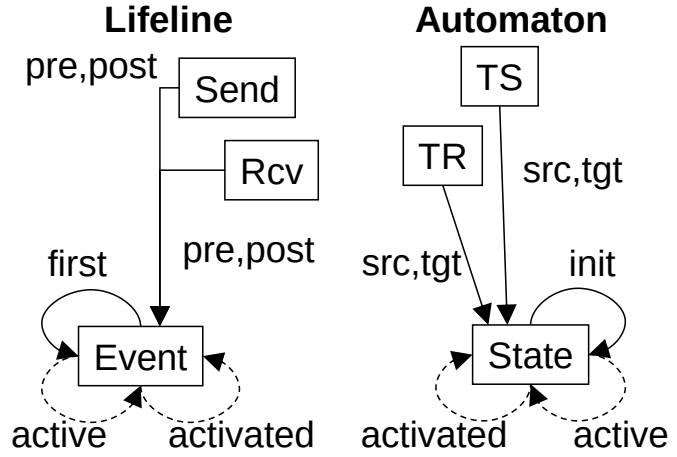
Definition 16 (runtime and static type graph, dynamic and static node/edge (type))**.** *Given a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, then a* runtime type graph *TG' with respect to $\mathcal{L}(TG, \mathcal{C})$ is a graph having TG as a subgraph. We say that TG is a* static type graph *w.r.t. TG' and that $\mathcal{C}$ is a* static constraint *w.r.t. TG'. All node and edge types in TG' but not in TG are called* dynamic node and edge types*, respectively. All node and edge types in TG but not in TG' are called* static node and edge types*, respectively. All nodes/edges typed over a static or dynamic node/edge type, resp., are called* static or dynamic nodes/edges*, respectively.*

Given a graph language with constraint and a corresponding runtime type graph we can now define a so-called runtime graph language with dynamic constraint. Each graph belonging to this language describes a potential runtime state. The dynamic constraint is thereby used in addition to the runtime type graph and the static constraint to restrict this language to well-formed potential runtime states.

Definition 17 (runtime graph language, dynamic constraint)**.** *Given a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, a runtime type graph TG' w.r.t. $\mathcal{L}(TG, \mathcal{C})$ and a graph constraint $\mathcal{C}_{dyn}$ typed over TG', but not only typed over TG, then the graph language with constraint $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ is called a* runtime graph language *with* dynamic constraint *$\mathcal{C}_{dyn}$.*

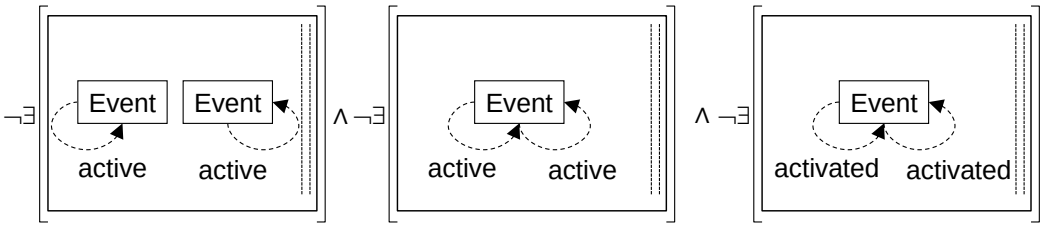
Example 4 (runtime graph language with dynamic constraint)**.** *We have runtime type graphs S_{RT} (T_{RT}) for the source (target) language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ ($\mathcal{L}(T_{TT}, \mathcal{C}_T)$) as depicted in Fig. 5(a) and 5(b), respectively. The dynamic edge types, in particular active and activated, are denoted with dashed edges. Moreover, they have the dynamic constraints $\mathcal{C}_s^{gts}$ (Figure 6(a)) and $\mathcal{C}_t^{gts}$ (Figure 6(b)) typed over the runtime type graphs S_{RT} and T_{RT}, respectively. Summarizing, we have the source and target runtime graph language with dynamic constraint $\mathcal{L}(S_{RT}, \mathcal{C}_S \wedge \mathcal{C}_s^{gts})$ ($\mathcal{L}(T_{RT}, \mathcal{C}_T \wedge \mathcal{C}_t^{gts})$), respectively.*

Given a graph language and a corresponding runtime graph language, a GTS typed over the runtime type graph will serve as the basis for the semantics of the

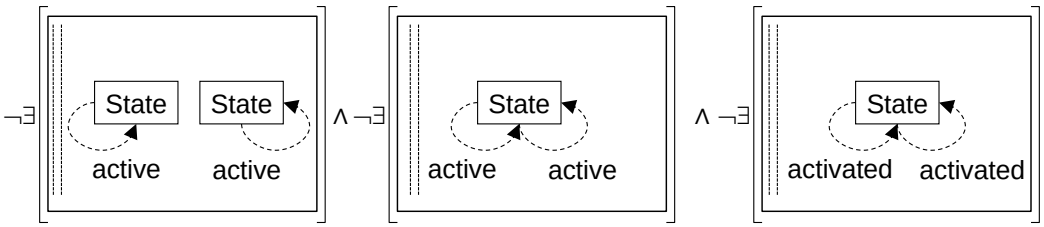


(a) Runtime type graph S_{RT} (b) Runtime type graph T_{RT}

Figure 5: Type graphs S_{TT} and T_{TT} enriched with dynamic types



(a) $\mathcal{C}_s^{gts}$



(b) $\mathcal{C}_t^{gts}$

Figure 6: $\mathcal{C}_s^{gts}$ and $\mathcal{C}_t^{gts}$

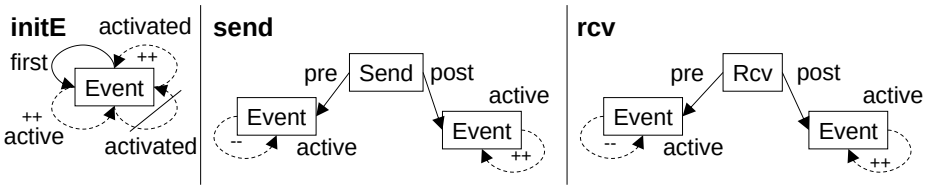
graph language. We therefore in the following will call the rules of such an GTS also semantics rules and corresponding rule applications semantics steps. In order for the semantics to be well-defined we assume some extra conditions on the GTS. First, it has the property that it does not change elements with static type, since the GTS merely models the change of runtime information for each graph in the graph language. Note that this implies that the GTS preserves the satisfaction of the static constraints of each runtime graph. Moreover, we assume that the GTS with static constraint has the given dynamic constraint of the runtime graph language as inductive invariant such that the corresponding graph transformation rules preserve also the satisfaction of the dynamic constraints of each runtime graph. If both conditions are fulfilled, we say that a GTS is *runtime conform* w.r.t. the given runtime graph language.

Definition 18 (runtime conform GTS)**.** *Given a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, a corresponding runtime graph language $\mathcal{L}(TG', C \wedge C_{dyn})$ and a GTS* $\mathrm{gts} = (\mathcal{R}, TG')$ *typed over the runtime type graph TG'. We say that* gts *is* runtime conform *w.r.t. the runtime graph language $\mathcal{L}(TG', \mathcal{C} \wedge C_{dyn})$ if (1) the rules $\mathcal{R}$ in* gts *preserve all static types of TG' and if (2) the GTS with constraint* $\mathrm{gts}^{\mathcal{C}} = (\mathcal{R}, TG', \mathcal{C})$ *has the dynamic constraint $\mathcal{C}_{dyn}$ as inductive invariant.*

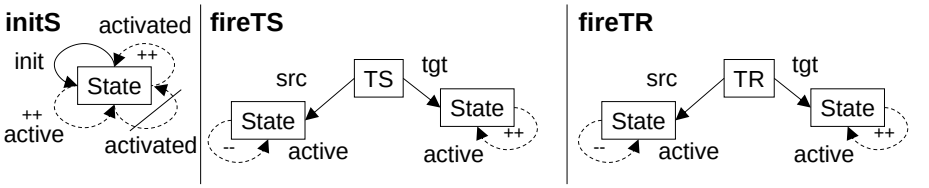
Remark 5. *Each GTS with constraint* $\mathrm{gts}^{\mathcal{C}} = (\mathcal{R}, \mathrm{TG}, \mathcal{C})$ *can be translated into an equivalent GTS without constraint* $\mathrm{gts}' = (\mathcal{R}', \mathrm{TG})$ *having $\mathcal{C}$ as inductive invariant. In particular, this means that, as proven in [34, 32], the constraint $\mathcal{C}$ can be translated into LHS application conditions for each rule ρ in $\mathcal{R}$ such that such a translated rule ρ' in $\mathcal{R}'$ is applicable to a graph satisfying $\mathcal{C}$ if and only if the resulting graph after application of ρ' satisfies $\mathcal{C}$ as well.*

Note that we can assume condition 2 of Def. 18 without loss of generality, because of the following reasoning: suppose that the GTS with static constraint does not have the dynamic constraint as inductive invariant. Then, we could consider the GTS with static and dynamic constraint as a basis for the semantics such that the satisfaction of the dynamic constraint is trivially always fulfilled. However, according to Remark 5 the GTS with static and dynamic constraint can be translated into an equivalent GTS with static constraint – but without dynamic constraint – that has this dynamic constraint as an inductive invariant.

Example 5 (runtime conform GTS)**.** *For the runtime source language $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ with runtime type graph S_{RT} and runtime target language $\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$ with runtime type graph T_{RT}, we can define a runtime conform source GTS* $\mathrm{gts}_s = (\mathcal{R}_s = \{\rho_s^i \mid i \in I\}, S_{RT})$ *and target GTS* $\mathrm{gts}_t = (\mathcal{R}_t = \{\rho_t^j \mid j \in J\}, T_{RT})$ *as depicted in Fig. 7(a) and 7(b), respectively. Note that the rules of these GTSs indeed preserve all elements with static type. Moreover,*



(a) gts_s



(b) gts_t

Figure 7: runtime conform GTSs for source and target runtime graph language

the corresponding GTSs with constraint $\mathrm{gts}_s^{\mathcal{C}_S}$ *and* $\mathrm{gts}_t^{\mathcal{C}_T}$ *have the dynamic constraints* $\mathcal{C}_s^{gts}$ *(Figure 6(a)) and* $\mathcal{C}_t^{gts}$ *(Figure 6(b)) typed over the runtime type graphs* S_{RT} *and* T_{RT}*, respectively, as inductive invariants. The core idea is that an event on (state in) a lifeline (automaton) cannot be activated twice depicted by the activated edge that is crossed out. This is enforced by corresponding NACs in initE (initS, respectively). Moreover, the static constraints take care of the fact that no two first events or initial states can occur in the first place. Similarly, there cannot be two active events (states) belonging to a single lifeline (automaton). The GTS* $\mathrm{gts}_s = (\{initE, send, rcv\}, S_{RT})$ *depicted in Fig. 7(a) typed over* S_{RT} *(see Fig. 5(a)) is then the basis for a semantic mapping for the source graphs. We use a notation marking elements that are created or deleted by the rule with "++" or "- -", respectively. We depict a NAC by crossing out the elements that it forbids. The rule* initE *holds a NAC, which forbids the occurrence of an activated edge. The rule describes that the first event of the lifeline is made active and marked as activated. The NAC ensures that if this first event was activated already, then it can't be activated again. Rule send describes the sending of a message from the lifeline between two events. Before applying the rule, the event previous to the message is active and afterwards the event after the message is active. The rule rcv analogously describes the receiving of a message. For the target graphs (depicted in Fig. 7(b)) we have* $\mathrm{gts}_t = (\{initS, fireTS, fireTR\}, T_{RT})$ *typed over* T_{RT} *(see Fig. 5(b)). Rule initS describes the activation of the initial state of the automaton. The NAC of this rule describes that if this initial state was activated already, then it cannot be activated again. Rule* $fireTS$ *describes the firing of a transition of type TS. Analogously,* $fireTR$ *describes the firing of a transition of type TR.*

The semantic mapping of a source and target graph language is based on the previously introduced graph transformation systems being runtime conform w.r.t. the corresponding runtime graph languages. In particular, each source (or target) graph is mapped to the labeled transition system that is induced by this graph and the runtime conform source (or target) GTS, respectively.

Definition 19 (Labeled transition system)**.** *A* labeled transition system *(LTS)* $lts = \langle i, \rightarrow, Q, L \rangle$ *consists of the initial state* i*, the labeled transition relation* $\rightarrow \subseteq Q \times L \times Q$ *over the label alphabet* L *and the set of states* Q*.*

For the formalism of typed graph transformation systems employed here, the most fine-grained labeling possible for a labeled transition system induced by some GTS, consists of the rule applied to reach a new state, but also of the match via which this rule is applied. To not upfront limit the information captured by the labeling, we thus employ both here for the induced LTS.

Definition 20 (induced LTS(gts, G_0))**.** *The* labeled transition system $\text{LTS}(\text{gts}, G_0)$ induced by $\text{gts} = (\mathcal{R}, TG)$ *and the initial graph* G_0 *equals* $\langle G_0, \rightarrow_{\text{gts}}, Q_{\text{gts}}, \mathcal{R} \times \mathcal{M} \rangle$ *with* $\rightarrow_{\text{gts}} = \{(G, (\rho, m), G') | G, G' \in Q_{\text{gts}}, \rho \in \mathcal{R} \wedge G \Rightarrow_{m,\rho} G'\}$*, and* $Q_{\text{gts}} = \textit{REACH}(\text{gts}, G_0)$ *and* $\mathcal{M}$ *the set of injective graph morphisms with as domain the LHS of some rule in* $\mathcal{R}$*.*

This definition of induced LTS enables us to refine the definition of model semantics, semantic domain and semantic mapping (see Definition 2): Given a modeling language in the form of a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$ and corresponding runtime graph language $\mathcal{L}(TG', C \wedge C_{dyn})$ with runtime conform GTS $\text{gts} = (\mathcal{R}, TG')$, then the semantic mapping maps each G in $\mathcal{L}(TG, \mathcal{C})$ to the induced LTS $\text{LTS}(\text{gts}, G)$, the semantics of G. The semantic domain thereby consists of the set of all LTSs that are induced from the GTS gts and some graph in $\mathcal{L}(TG, \mathcal{C})$. This is illustrated by the following example.

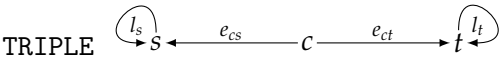
Example 6 (induced LTS as model semantics)**.** *Given the runtime conform graph transformation systems* gts_s *and* gts_t *as introduced in Example 5, then the induced LTSs* $\text{LTS}(\text{gts}_s, S)$ *or* $\text{LTS}(\text{gts}_t, T)$*, respectively, describe for each graph* S *or* T *in* $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ *or* $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ *the corresponding semantics, respectively.*

2.3. Model Transformation

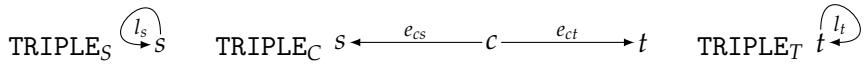
In this report, we will formalize model transformations according to Definition 3 using triple graph grammars. Triple graph grammars (TGGs) define model transformations in a relational (declarative) way by combining three conventional graph grammars for the source, target and correspondence models. The correspondence model explicitly stores correspondence relationships between source and target

model elements. A TGG consists of an axiom and several non-deleting graph rules. This grammar creates all so-called triple graphs describing a valid model transformation instance. In particular, the source (target) component of such a triple graph describes the source (target) model of each model transformation instance and the correspondence component relates them by correspondence relationships. TGGs are relational model transformations that cannot be executed directly to transform a given source model to a target model. Instead, operational rules have to be derived for each transformation direction: A forward/backward transformation takes a source/target model and creates the correspondence and target/source models. A model integration creates the correspondence model for given source and target models.

We use [26, 28] a TGG formalization more suitable for the current practice for TGGs than the one introduced originally in [47]. Thereby, the main idea is to use a distinguished, fixed graph TRIPLE which all triple graphs, including the type triple graph $S_{TT}C_{TT}T_{TT}$, are typed over.



We say that $\texttt{TRIPLE}_S$, $\texttt{TRIPLE}_C$, and $\texttt{TRIPLE}_T$, as shown below,



are the *source, correspondence*, and *target component* of TRIPLE, respectively. Analogously to the aforementioned case, the projection of a graph G typed over TRIPLE to $\texttt{TRIPLE}_S$, $\texttt{TRIPLE}_C$, or $\texttt{TRIPLE}_T$ selects the corresponding component of this graph.

We denote a triple graph as a combination of three indexed capitals, as for example $G = S_G C_G T_G$, where S_G denotes the *source* and T_G denotes the *target component* of G, while C_G denotes the *correspondence component*, being the smallest subgraph of G such that all c-nodes as well as all e_{cs}- and e_{ct}-edges are included in C_G. Note that C_G has to be a proper graph, i.e. all target nodes of e_{cs} and e_{ct}-edges have to be included. The category of triple graphs and triple graph morphisms is called **TripleGraphs**.

Analogously to typed graphs, *typed triple graphs* are triple graphs typed over a distinguished triple graph $S_{TT}C_{TT}T_{TT}$, called type triple graph. The category of typed triple graphs and morphisms is called $\textbf{TripleGraphs}_{\texttt{TT}}$. In the remainder of this report, we assume every triple graph $S_G C_G T_G$ and triple graph morphism f to be typed over $S_{TT}C_{TT}T_{TT}$, even if not explicitly mentioned. In particular, this means that S_G is typed over S_{TT}, C_G is typed over C_{TT}, and T is typed over T_{TT}. We say that S_G (T_G or C_G) is a *source graph* (*target graph* or *correspondence graph*, respectively) belonging to the language $\mathcal{L}(S_{TT})$ ($\mathcal{L}(T_{TT})$ or $\mathcal{L}(C_{TT})$, respectively).

Notation. *Note that each source graph (target graph) corresponds uniquely to a triple graph with empty correspondence and target (source and correspondence) components, respectively. Therefore, if it is clear from the context that we are dealing with triple graphs, we denote triple graphs $S_G\varnothing\varnothing$ ($\varnothing\varnothing T_G$) with empty correspondence and target components (source components) also as S_G (T_G), respectively.*

A *triple graph rule* $p : S_L C_L T_L \xrightarrow{r} S_R C_R T_R$ consists of a triple graph morphism r, which is an inclusion. A *direct triple graph transformation* $S_G C_G T_G \Rightarrow_{p,m} S_H C_H T_H$ from $S_G C_G T_G$ to $S_H C_H T_H$ via p and m consists of the pushout (PO) in $\mathbf{TripleGraphs}_{\mathrm{TT}}$.

$$\begin{array}{ccc} S_L C_L T_L & \xrightarrow{r} & S_R C_R T_R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow{\scriptstyle n} \\ S_G C_G T_G & \xrightarrow[h]{} & S_H C_H T_H \end{array}$$

A *triple graph transformation*, denoted as $S_{G_0} C_{G_0} T_{G_0} \overset{*}{\Rightarrow} S_{G_n} C_{G_n} T_{G_n}$, is a sequence $S_{G_0} C_{G_0} T_{G_0} \Rightarrow S_{G_1} C_{G_1} T_{G_1} \Rightarrow \cdots \Rightarrow S_{G_n} C_{G_n} T_{G_n}$ of direct triple graph transformations. As in the context of classical triple graphs, we consider triple graph grammars (TGGs) with non-deleting rules. Moreover, we allow grammars to be equipped with a so-called *TGG constraint* $\mathcal{C}_{\mathrm{tgg}}$ typed over $S_{TT} C_{TT} T_{TT}$, restricting the language of triple graphs generated by the TGG to those triple graphs generated via transformations of triple graphs satisfying $\mathcal{C}_{\mathrm{tgg}}$.

Definition 21 (Triple graph grammar, language)**.** *A* triple graph grammar *(TGG)* $\mathrm{tgg} = (\mathcal{R}, S_{TT} C_{TT} T_{TT}, S_A C_A T_A)$ *consists of a set of triple graph rules $\mathcal{R}$ typed over $S_{TT} C_{TT} T_{TT}$ and a triple start graph $S_A C_A T_A$, called axiom, also typed over $S_{TT} C_{TT} T_{TT}$. The* triple graph language $\mathcal{L}(\mathrm{tgg})$ *equals* $\mathsf{REACH}((\mathcal{R}, S_{TT} C_{TT} T_{TT}), S_A C_A T_A)$*. A triple graph grammar* tgg *can be equipped with a so-called TGG constraint $\mathcal{C}_{\mathrm{tgg}}$ typed over $S_{TT} C_{TT} T_{TT}$ such that $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{tgg}}$. The* triple graph language $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *equals* $\mathsf{REACH}((\mathcal{R}, S_{TT} C_{TT} T_{TT}, \mathcal{C}_{\mathrm{tgg}}), S_A C_A T_A)$.

The type graph $S_{TT} C_{TT} T_{TT}$ enriched with dynamic types for source and target languages is denoted as $S_{RT} C_{TT} T_{RT}$. Note that if some graph $S_G C_G T_G$, morphism m, rule ρ, or condition ac is typed over a subgraph $S_{SG} C_{SG} T_{SG}$ of $S_{RT} C_{TT} T_{RT}$, then it is straightforward to extend the codomain of the corresponding typing morphisms to $S_{RT} C_{TT} T_{RT}$ such that $S_G C_G T_G$, m, ac, or ρ are actually typed over $S_{RT} C_{TT} T_{RT}$. We therefore do not explicitly mention this anymore in the rest of this report.

Example 7 (Triple Graph Grammar)**.** *In Fig. 8, an example TGG* tgg *is depicted with an axiom $S_A C_A T_A$ and two rules typed over the type graph $S_{TT} C_{TT} T_{TT}$. The type graph $S_{TT} C_{TT} T_{TT}$ is the subgraph of the type graph $S_{RT} C_{TT} T_{RT}$ shown in the upper right part of Fig. 8, obtained by deleting the dynamic node and edge types in the source and target*

component. The TGG rules describe a model transformation between a sequence chart with one lifeline being able to send and receive messages and an automaton with two different types of transitions, one for the sending and one for the receiving of messages as described already in Example 2 and 3. The events before and after a send/receive message on the lifeline correspond to states before and after send/receive transitions in the automaton. On the lifeline, there is one first event which corresponds to an initial state of the automaton. The TGG constraint $\mathcal{C}_{\text{tgg}}$ is shown in Figures 27, 28 and 29. A subset is shown in the lower right part of Fig. 8. It is in particular a constraint for the source language and expresses that each event should be connected with at most one previous message (subsequent message) of type Send/Rcv. An analogous condition holds for states and transitions in the automaton.

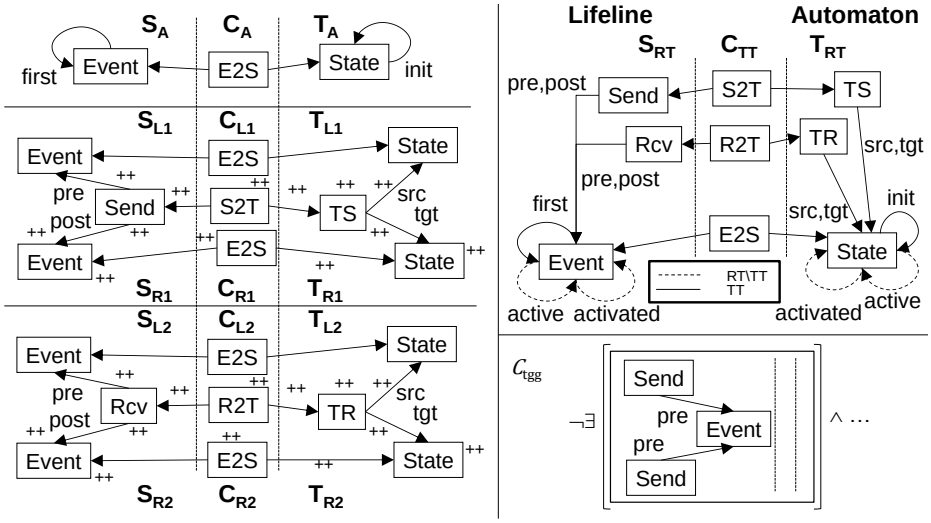


Figure 8: tgg with axiom and two rules, type graph $S_{RT}C_{TT}T_{RT}$, fragment of $\mathcal{C}_{\text{tgg}}$

Analogous to [25, 36], we derive a model transformation over $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}) \times \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ from a given TGG tgg typed over $S_{TT}C_{TT}T_{TT}$. Additionally, we allow tgg to be equipped with a TGG constraint $\mathcal{C}_{\text{tgg}}$ such that the model transformation is based on the language $\mathcal{L}(\text{tgg}, \mathcal{C}_{\text{tgg}})$. Moreover, w.l.o.g. we assume that $\mathcal{C}_{\text{tgg}}$ comprises the source and target constraints $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{T}}$ of the source and target graph language $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ and $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$, respectively. In particular, this assumption ensures that the following definition is well-defined.

Definition 22 (induced $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$)**.** *Given a source and target graph language with constraint $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ and $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ as given in Def. 12 and a TGG* tgg *with TGG constraint $\mathcal{C}_{\mathrm{tgg}}$ typed over $S_{TT}C_{TT}T_{TT}$ such that it comprises the source and target constraints $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{T}}$, then the* induced model transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}) \times \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ *consists of pairs of source and target graphs (S, T) such that there exists some triple graph $SCT \in \mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ having S and T as source and target component, respectively.*

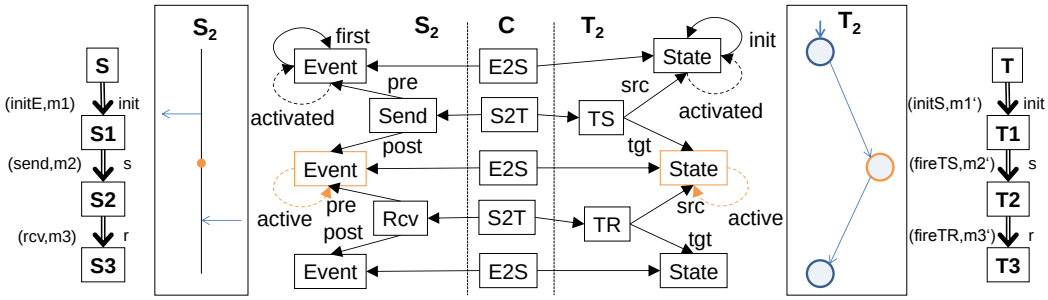


Figure 9: Model transformation instance (S, T) with $SCT \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ and induced LTSs $\mathrm{LTS}(\mathrm{gts}_s, S)$ and $\mathrm{LTS}(\mathrm{gts}_t, T)$ for runtime conform gts_s and gts_t such that $SCT \subseteq S_2CT_2$

Example 8 (induced $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ as model transformation)**.** *In Fig. 9, a source graph S_2 and target graph T_2 is depicted in abstract as well as concrete syntax describing a runtime state for the source graph S and target graph T belonging to* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *with* tgg *and $\mathcal{C}_{\mathrm{tgg}}$ as described in Example 7 and induced LTSs* $\mathrm{LTS}(\mathrm{gts}_s, S)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, T)$ *for runtime conform* gts_s *and* gts_t *as described in Example 6. A triple graph SCT (depicted in Figure 3 of Example 2) fulfilling $\mathcal{C}_{\mathrm{tgg}}$ can be generated by* tgg*, being the subgraph of S_2CT_2 depicted in Fig. 9 obtained by not considering the dynamic elements (*activated *and* active *loops). The currently active event (state) on the lifeline (in the automaton) is depicted in orange.*

2.4. Behavioral Equivalence and Refinement

In order to adequately compare the induced labeled transition systems of a source and target model of a model transformation, we introduce a so-called relabeling such that the different alphabets of the source and target LTSs can be mapped to a

common one. Note that this relabeling is the concrete realization of the notion of remapping of semantic domains as introduced in Def. 4.

Definition 23 (Relabeling)**.** *Given a modeling language in the form of a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, a corresponding runtime graph language $\mathcal{L}(TG', C \wedge C_{dyn})$ with runtime conform GTS* $\text{gts} = (\mathcal{R}, TG')$*, an induced LTS $LTS(\text{gts}, G)$ with G some graph in $\mathcal{L}(TG, \mathcal{C})$ and label alphabet $\mathcal{R} \times \mathcal{M}$ and a total function $l : \mathcal{R} \times \mathcal{M} \to L'$ to some label alphabet L', then the* relabeling *induced from l maps each LTS $LTS(\text{gts}, G)$ to the LTS where each label (ρ, m) has been replaced by $l(\rho, m)$.*

Remark 6. *In the rest of the report we do not make a distinction between the total function $l : \mathcal{R} \times \mathcal{M} \to L'$ and the derived relabeling mapping each given $LTS(\text{gts}, G)$ to $l(LTS(\text{gts}, G))$ as described above if it is clear from the context.*

To formalize behavioral equivalence and behavioral refinement according to Definition 5, we will further employ the notions of bisimulation and simulation [41], two particular flavors of behavior preservation for a model transformation that require a relation between the states of different labeled transition systems (cf. [51, 52]). The behavioral equivalence of bisimilarity of two LTSs over the same alphabet is defined as follows:

Definition 24 (bisimulation relation [41])**.** *A* bisimulation relation *between two labeled transition systems* $\text{lts}_1 = \langle i_1, \to, Q_1, A \rangle, \text{lts}_2 = \langle i_2, \to, Q_2, A \rangle$ *over the same alphabet A is a relation $B \subseteq Q_1 \times Q_2$ such that whenever $(q_1, q_2) \in B$*

1. *If $q_1 \xrightarrow{\alpha} q_1'$, then $q_2 \xrightarrow{\alpha} q_2'$ and $(q_1', q_2') \in B$.*
2. *If $q_2 \xrightarrow{\alpha} q_2'$, then $q_1 \xrightarrow{\alpha} q_1'$ and $(q_1', q_2') \in B$.*

We say that lts_1 *and* lts_2 *are* bisimilar *and write* $\text{lts}_1 =_{bsim} \text{lts}_2$ *if there exists a bisimulation relation B between them such that $(i_1, i_2) \in B$.*

Note that if $\text{lts}_1 =_{bsim} \text{lts}_2$ holds, we can conclude, for example, that all traces of lts_1 can also be found in lts_2 and vice versa. Consequently, $\text{lts}_1 =_{bsim} \text{lts}_2$ describes that lts_1 somehow preserves all possible behavior of lts_2 (equivalence).

In some cases, behavioral refinement a weaker kind of behavior preservation is enough and thus we can consider the simulation relation instead of bisimulation relation.[2]

[2]Note that a simulation relation S is also a bisimulation relation if S^{-1} is a simulation relation as well.

Definition 25 (simulation relation [41]). *A* simulation relation *between two labeled transition systems* $\text{lts}_1 = \langle i_1, \rightarrow, Q_1, A \rangle, \text{lts}_2 = \langle i_2, \rightarrow, Q_2, A \rangle$ *over the same alphabet* A *is a relation* $S \subseteq Q_1 \times Q_2$ *such that whenever* $(q_1, q_2) \in S$

- *If* $q_1 \xrightarrow{\alpha} q_1'$, *then* $q_2 \xrightarrow{\alpha} q_2'$ *and* $(q_1', q_2') \in S$.

We say that lts_1 *is* simulated by lts_2 (lts_2 simulates lts_1) *and write* $\text{lts}_1 \leq_{sim} \text{lts}_2$ *if there exists a simulation relation* S *between them such that* $(i_1, i_2) \in S$.

Note that if $\text{lts}_1 \leq_{sim} \text{lts}_2$ holds we can conclude, for example, that all traces of lts_1 can also be found in lts_2, while the opposite may in general not be the case. Consequently, $\text{lts}_1 \leq_{sim} \text{lts}_2$ describes that lts_1 somehow preserves some of the possible behavior of lts_2 (refinement), but does not preserve all possible behavior (equivalence). As a consequence, lts_2 can also be seen as an abstraction of lts_1 since lts_2 still allows more behavior than lts_1.

2.5. Behavior Preservation at the Transformation Level: Formalized

Now we have the means to model formally each concept of behavior preservation at the transformation level as presented generically in Definition 6 using the notions of graph transformation, triple graph grammars, and labeled transition systems as summarized in Table 2. In particular, this table shows in a compact way how to formalize modeling languages in modeling step M_{lang}, model semantics in modeling step M_{sem}, and model transformation in modeling step M_{trans}. Moreover, in Tables 3, 4, and 5, an overview is given of these modeling steps for our current example and subsequent more elaborate examples. In the rest of the report, we will often need all artefacts involved for the modeling steps M_{lang}, M_{sem}, and M_{trans} as a basis for further concepts and definitions. Therefore in the following definition, we first introduce their composition as a formally covered model transformation to be used in the rest of the report.

Table 2: Generic concepts from Definition 6 and corresponding modeling steps

concept (modeling step)	modeling languages (M_{lang})	model semantics (M_{sem})	model transformation (M_{trans})	behavior preservation (M_{pres})	
Def. 6	$\mathcal{L}_\mathcal{S}, \mathcal{L}_\mathcal{T}$	$sem_\mathcal{S}(\cdot)$, $sem_\mathcal{T}(\cdot)$	MT	equivalence	refinement
				$l_s(.) =_\mathcal{D} l_t(.)$	$l_t(.) \leq_\mathcal{D} l_s(.)$
Def. 27	$\mathcal{L}(S_{TT}, \mathcal{C}_S)$, $\mathcal{L}(T_{TT}, \mathcal{C}_T)$	$\text{LTS}(\text{gts}_s, \cdot)$, $\text{LTS}(\text{gts}_t, \cdot)$	$MT(tgg, \mathcal{C}_{tgg})$	$l_s(.) =_{bsim} l_t(.)$	$l_t(.) \leq_{sim} l_t(.)$

Definition 26 (Formally Covered Model Transformation)**.** *Given source and target graph language with constraint* $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ *and* $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ *(see Def.12), a corresponding model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}) : \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ *for a* $\mathrm{tgg} = ((\mathcal{R}, S_{TT}C_{TT}T_{TT}), S_AC_AT_A)$ *with TGG constraint* $\mathcal{C}_{\mathrm{tgg}}$ *(see Def. 22), corresponding runtime source and target graph language with dynamic constraint* $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ *and* $\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$ *(see Def. 17) together with runtime conform source and target GTSs* $\mathrm{gts}_s = (\mathcal{R}_s, S_{RT})$ *and* $\mathrm{gts}_t = (\mathcal{R}_t, T_{RT})$ *(see Def. 18) and the induced LTSs* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *(see Def. 20) for each element of* $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ *and* $\mathcal{L}(T_{TT}, \mathcal{C}_T)$*, respectively, we say that* $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .))$ *is a* formally covered model transformation.

Based on this formally covered model transformation together with the notion of relabeling as introduced in the previous section as well as the classical notion of bisimulation or simulation, we obtain the following formalized definition of behavior preservation at the transformation level (see also Fig.10 and Table 2). As illustrated in the figure, the semantics of source and target language is defined by the induced LTSs of runtime conform GTSs for each source and target model of the model transformation. Behavior preservation in an equivalent or refining manner then holds if in between all these pairs of source and target LTSs after some proper relabeling a bisimulation or simulation relation can be constructed, respectively.

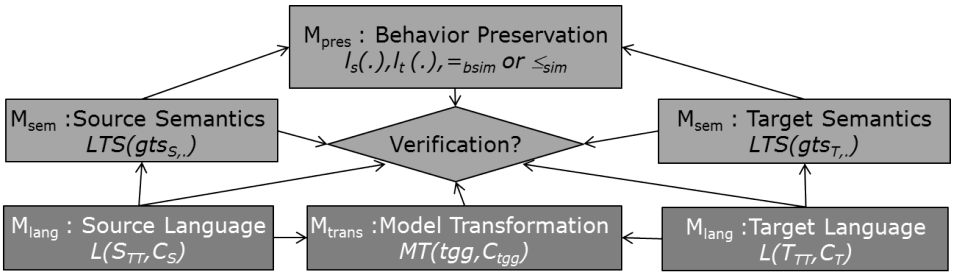


Figure 10: Behavior Preservation – Transformation Level (see Fig. 1) – Formalized (see Definition 27)

Definition 27 (Behavior Preservation – Transformation Level – Formalized)**.** *Given a formally covered model transformation (i.e.,* $\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .)$*) as introduced in Def. 26 together with relabelings* l_s *and* l_t *for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *into a common alphabet A as given in Def. 23, respectively, we say that*

3.1 *the model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is* behavior preserving in an equivalent manner *if for each pair of source and target models* $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *it holds that* $l_s(\mathrm{LTS}(\mathrm{gts}_s, S)) =_{bsim} l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$.

3.2 *the model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is* behavior preserving in a refining manner *if for each pair of source and target models* $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *it holds that* $l_t(\mathrm{LTS}(\mathrm{gts}_t, T)) \leq_{sim} l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$.

Example 9 (Behavior Preservation – Transformation Level – Formalized)**.** *The example for Definition 27 consists of the TGG* tgg *with TGG constraint* $\mathcal{C}_{\mathrm{tgg}}$ *of Example 7 shown in Fig. 8, describing a model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}) : \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}) \times \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ *between sequence charts (source modeling language* $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$*) and automata (target modeling language* $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$*). Both source and target language are equipped with a runtime language with dynamic constraints* $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ *(*$\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$*) as described in Example 4 and corresponding runtime conform graph transformation systems* $\mathrm{gts}_s = (\{initE, send, receive\}, S_{RT})$ *and* $\mathrm{gts}_t = (\{initS, fireTS, fireTR\}, T_{RT})$ *(see Example 5) describing the possible behavior of sequence charts and automata, respectively. The GTSs* $\mathrm{gts}_s^{\mathcal{C}_S}$ *and* $\mathrm{gts}_t^{\mathcal{C}_T}$ *have as inductive invariants the source dynamic constraint* $\mathcal{C}_s^{gts}$ *and the target dynamic constraint* $\mathcal{C}_t^{gts}$*, respectively. For source and target models S and T, the semantic mappings assign induced labeled transition systems* $\mathrm{LTS}(\mathrm{gts}_s, S)$ *and* $\mathrm{LTS}(\mathrm{gts}_s, T)$ *to S and T, respectively.*

To summarize, we have now established a formalization for the general definition of behavior preservation (see Definition 6) such that most concepts are at hand. We have clarified how the modeling language, model semantics, and model transformation have to be formalized and have defined what we consider as behavioral equivalence and as behavioral refinement. However, two issues remain still to be addressed:

- The relabelings l_s and l_t to establish a shared semantic domain and to be able to apply bisimulation/simulation remain to be further refined. As we will see in the following sections, this is not trivial and has great impact on how behavior preservation can be verified.

- The even more challenging remaining problem is that our mapping to LTSs and the use of bisimulation and simulation for LTSs would directly enable us to approach the verification of behavior preservation if we would work at the instance level. However, we want to address the verification at the transformation level (see Definition 6) and this cannot be done straight forwardly at the level of the LTSs as there are potentially infinitely many source and target models that have to be considered.

Table 3: Modeling step M_{lang}

concept	modeling language			
formalization	source: $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$		target: $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$	
artefact	type graph S_{TT}	constraint $\mathcal{C}_{\mathcal{S}}$	type graph T_{TT}	constraint $\mathcal{C}_{\mathcal{T}}$
simple equivalence (Ex. 9,15) Section 3	Fig. 2(a), 24	Fig. 4, 27	Fig. 2(b), 24	Fig. 28
equivalence (Example 23) Section 4	Fig. 14(a), 35	Fig. 39	Fig. 14(b), 35	Fig. 40
refinement (Example 25) Section 5	Fig. 47	Fig. 52	Fig. 47	Fig. 53

Table 4: Modeling step M_{sem}

concept	model semantics					
formalization	source: LTS($\text{gts}_s, \cdot$) with runtime conform $\text{gts}_s = (\mathcal{R}_s, S_{RT})$ and dynamic constraint $\mathcal{C}_s^{gts}$			target: LTS($\text{gts}_t, \cdot$) with runtime conform $\text{gts}_t = (\mathcal{R}_t, T_{RT})$ and dynamic constraint $\mathcal{C}_t^{gts}$		
artefact	rules $\mathcal{R}_s$	type graph S_{RT}	dynamic constraint $\mathcal{C}_s^{gts}$	rules $\mathcal{R}_t$	type graph T_{RT}	dynamic constraint $\mathcal{C}_t^{gts}$
simple equivalence (Ex. 9,15) Section 3	Fig. 7(a)	Fig. 5(a), 24	Fig. 6(a), 32	Fig. 7(b)	Fig. 5(b), 24	Fig. 6(b), 33
equivalence (Example 23) Section 4	Fig. 15(a), 37	Fig. 14(a), 35	Fig. 42	Fig. 15(b), 37	Fig. 14(b), 35	Fig. 43
refinement (Example 25) Section 5	Fig. 51	Fig. 47	Fig. 42	Fig. 57	Fig. 47	Fig. 43

Table 5: Modeling step M_{trans}

concept	model transformation		
formalization	MT(tgg, $\mathcal{C}_{\text{tgg}}$) with tgg = $((\mathcal{R}, S_{TT}C_{TT}T_{TT}), S_AC_AT_A)$		
artefact	TGG rules $\mathcal{R}$	type graph $S_{TT}C_{TT}T_{TT}$	TGG constraint $\mathcal{C}_{\text{tgg}}$
simple equivalence (Ex. 9,15) Section 3	Fig. 8, 25	Fig. 8, 24	Fig. 8, 29
equivalence (Example 23) Section 4	Fig. 16, 36	Fig. 14(c), 35	Fig. 41
refinement (Example 25) Section 5	Fig. 48, 49, 50	Fig. 47	Fig. 54

To overcome these two issues we will in the following exploit that the semantic mapping for each model is in fact provided by GTS rules that interpret the models and are thus the same for all models. Therefore, we can – as we will demonstrate in the next section – consider behavior preservation verification for all source and target models at the same time by also defining the relabelings and the bisimulation/simulation relation for all source and target models at once.

3. Approaching Behavioral Equivalence Verification

We first reintroduce some formal concepts in Section 3.1 that will be important for verifying behavior preservation. Then, we report on our first approach to tackle behavioral equivalence presented in [27]: In Section 3.2 we present the particular relabeling for the source and target labeled transition systems proposed in [27] and explain how to set up a relation between them that is a predestined candidate for expressing the bisimulation between these relabeled transition systems. In Section 3.3, we present a verification scheme of [27] for showing that this relation indeed defines a bisimulation relation for each source and target model of the model transformation. Finally, in Section 3.4 we discuss that the relabelings proposed in [27] and presented in this section with the corresponding predestined bisimulation relation is too limited for more complex cases.

3.1. Prerequisites

The applicability of a graph transformation rule can be expressed as a graph constraint. We exploited this feature already in [3, 23] for the consistency preservation verification of rule-based refactorings and for consistency verification of integrated behavior models, respectively. The rule applicability constraint for a rule $\rho = \langle p, \mathrm{ac}_L \rangle$ with $p = \langle L \overset{l}{\hookleftarrow} I \overset{r}{\hookrightarrow} R \rangle$, expresses that an injective match m exists such that the application condition ac_L and the so-called deletable condition $Deletable(p)$[3], guaranteeing the existence of a PO-complement for $m \circ l$, are fulfilled. Then it is obvious that the rule $\rho = \langle p, \mathrm{ac}_L \rangle$ is applicable with injective matching to a graph G if and only if G fulfills the rule applicability constraint.

Definition 28 (rule applicability constraint)**.** *Given a rule* $\rho = \langle p, \mathrm{ac}_L \rangle$ *with* plain *rule* $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$*, then* $\mathrm{App}(\rho) = \exists(i_L, \mathrm{ac}_L \wedge Deletable(p))$ *is the rule applicability constraint of* ρ*. We moreover write* $\mathrm{ac}_{\mathrm{App}(\rho)}$ *to refer to* $\mathrm{ac}_L \wedge Deletable(p)$ *for* ρ*.*

Example 10 (rule applicability constraint)**.** *The applicability of the rule* send*, depicted in Fig. 7, can be expressed as graph constraint* $\exists(i_L, \mathrm{true})$*, or abbreviated* $\exists L$*, with* L *the LHS of* send*. This is because* ac_L *is* true *and* $Deletable(p)$ *is* true*, since the rule does not delete any nodes. The same holds for the applicability of rule* fireTS*. In Fig. 11, the graph constraint* $\mathrm{App}(send) \Rightarrow \mathrm{App}(fireTS)$*, equivalent to* $\neg\mathrm{App}(send) \vee \mathrm{App}(fireTS)$*, is depicted.*

[3] In Lemma 5.9 of [45], it is described how to construct $Deletable(l)$ (we write $Deletable(p)$ instead of $Deletable(l)$). Basically, it prohibits the existence of additional adjacent edges, making use of additional NACs, for nodes that are to be deleted.

According to Definition 27, for showing behavior preservation at the transformation level we need to find a bisimulation or simulation between the relabeled source and target transition systems of each source and target model of the model transformation under consideration. It consists in particular of a relation over $\mathsf{REACH}(\mathrm{gts}_s, S) \times \mathsf{REACH}(\mathrm{gts}_t, T)$ between the respective states of the transition systems. We know that for each (S, T) in $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$, there exists some SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$. Then a key idea of our approach is to require that (S, T) belongs to the relation if the corresponding SCT fulfills some particular constraint $\mathcal{C}$. So we derive the (bi-)simulation relation from a given graph constraint $\mathcal{C}$ typed over $S_{RT}C_{TT}T_{RT}$, characterizing all triple graphs for which the source and target components belong to the relation. In the following definition we formalize this idea of deriving a relation over $\mathsf{REACH}(\mathrm{gts}_s, S) \times \mathsf{REACH}(\mathrm{gts}_t, T)$ from a constraint $\mathcal{C}$ typed over $S_{RT}C_{TT}T_{RT}$.

Definition 29 (induced relation $\mathcal{R}(\mathcal{C}, .)$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .))$ *as introduced in Def. 26 and given a graph constraint $\mathcal{C}$ typed over $S_{RT}C_{TT}T_{RT}$, then for each (S, T) in* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *and a corresponding SCT in* $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *the* induced relation $\mathcal{R}(\mathcal{C}, SCT) \subseteq \mathsf{REACH}(\mathrm{gts}_s, S) \times \mathsf{REACH}(\mathrm{gts}_t, T)$ *consists of all (S', T') such that $S'CT'$ fulfills $\mathcal{C}$.*

The above definition is well-defined, i.e. each $S'CT'$ is a valid triple graph, because of the following lemma:

Lemma 1 (induced relation $\mathcal{R}(\mathcal{C}, .)$ well-defined)**.** *Each $S'CT'$ with $S' \in \mathsf{REACH}(\mathrm{gts}_s, S)$ and $T' \in \mathsf{REACH}(\mathrm{gts}_t, T)$ as given in Def. 29 is a well-defined triple graph.*

Proof. In Def. 18, we required that gts_s and gts_t are runtime conform such that the rules preserve elements with static types. Therefore, it holds that S and T is included in each $S' \in \mathsf{REACH}(\mathrm{gts}_s, S)$ and $T' \in \mathsf{REACH}(\mathrm{gts}_t, T)$, respectively. Moreover, since the correspondence component C consists of all edges connecting S and T via correspondence nodes including incident nodes belonging to S and T, also $S'CT'$ is a well-defined triple graph. □

3.2. Modeling Step M_{pres}

As described after our formalized definition of behavior preservation at the transformation level (see Definition 27) the concrete modeling of the relabelings of source and target labeled transition systems as well as the concrete modeling of the bisimulation/simulation relation for each relabeled source and target transition system realizing a bisimulation/simulation are still two open issues (modeling step M_{pres}). In this section, we give a first idea of how to model these two remaining artefacts.

3.2.1. Relabeling

We relabel the source and target labeled transition systems by means of bijective mappings[4] from the source and target GTS rules to a common alphabet. In particular, as a first approach, we discard the match in each label.

Definition 30 (Relabelings l_s and l_t)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_\mathcal{S}), \mathcal{L}(T_{TT}, \mathcal{C}_\mathcal{T}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as given in Def. 26 together with two bijective mappings* $l_s^\mathcal{R} : \mathcal{R}_s \to A$ *and* $l_t^\mathcal{R} : \mathcal{R}_t \to A$ *from the source and target semantics rules into a common alphabet* A*, then the* relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A$ and $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A$ *for* $\text{LTS}(\text{gts}_s, .)$ *and* $\text{LTS}(\text{gts}_t, .)$*, resp., are defined as follows:* $l_s(\rho_s, m_s) = l_s^\mathcal{R}(\rho_s)$ *and* $l_t(\rho_t, m_t) = l_t^\mathcal{R}(\rho_t)$*.*

Example 11 ($l_s(\text{LTS}(\text{gts}_s, .))$ and $l_t(\text{LTS}(\text{gts}_t, .))$)**.** *For the source and target GTSs* gts_s *and* gts_t *from Example 5 and the induced LTSs* $\text{LTS}(\text{gts}_s, .)$ *and* $\text{LTS}(\text{gts}_t, .)$ *we define bijective mappings* $l_s^\mathcal{R} : \mathcal{R}_s \to A$ *and* $l_t^\mathcal{R} : \mathcal{R}_t \to A$ *with* $A = \{init, s, r\}$ *and* $l_s^\mathcal{R}(initE) = init, l_s^\mathcal{R}(send) = s, l_s^\mathcal{R}(rcv) = r$ *and* $l_t^\mathcal{R}(initS) = init, l_t^\mathcal{R}(fireTS) = s, l_t^\mathcal{R}(fireTR) = r$*. The application of the respective relabelings to the LTSs of the example source and target model is depicted in Fig. 9, where the original labels are depicted on the left side of a transition, while the relabeled labels are depicted on the right side of a transition.*

3.2.2. Bisimulation

The question remains how to specify the relation between the state sets of the relabeled transition systems $l_s(\text{LTS}(\text{gts}_s, S))$ and $l_t(\text{LTS}(\text{gts}_t, T))$ for any (S, T) in our model transformation $\text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}})$. In particular, as described already in Section 3.1 we will derive it as an induced relation (see Def. 29) from the so-called *bisimulation constraint* $\mathcal{C}_{\text{Bis}}$. In particular the bisimulation constraint $\mathcal{C}_{\text{Bis}}$ equals the conjunction of a so-called runtime constraint $\mathcal{C}_{\text{RT}}$, pair constraint $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$, the source and target dynamic constraints $\mathcal{C}_s^{gts}$ and $\mathcal{C}_t^{gts}$ as introduced in Def. 17, the TGG constraint $\mathcal{C}_{\text{tgg}}$ as introduced in Def. 21, and a so-called model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}}$.

First, the *pair constraint* $\mathcal{C}_{\text{Pair}}^{\text{Rule}} = \mathcal{C}_{\text{Pair}}^{\text{Rule,f}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Rule,b}}$, can be derived automatically from the source and target GTSs gts_s and gts_t and the corresponding relabelings l_s and l_t. It expresses that the applicability of a rule ρ_s of the source semantics gts_s implies the applicability of a rule ρ_t of the target semantics gts_t ($\mathcal{C}_{\text{Pair}}^{\text{Rule,f}}$) and the other way round ($\mathcal{C}_{\text{Pair}}^{\text{Rule,b}}$), whenever ρ_s and ρ_t are mapped by $l_s^\mathcal{R}$ and $l_t^\mathcal{R}$ to the same label in the common alphabet A. Trivially speaking, rules "with the same

[4] It is important that the mappings are bijective to guarantee that we get a clear correspondence between source and target rules of the semantics for the source and target models. See Section 8 for a more detailed discussion of this issue.

label" should be applicable pairwise, since this is exactly what we need for proving bisimulation. Rules with the same label are gathered into pairs of rules and their application is summarized into parallel rules. We call such a parallel rule also pair rule.

Definition 31 (set of pairs $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$, pair rules $\mathcal{P}(l_s, l_t)$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as given in Def. 26 and two relabelings* l_s *and* l_t *for* $\text{LTS}(\text{gts}_s, .)$ *and* $\text{LTS}(\text{gts}_t, .)$ *as given in Def. 30 with bijective mappings* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$*, the* set of pairs $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}}) = \{(\rho_s, \rho_t) | l_s^{\mathcal{R}}(\rho_s) = l_t^{\mathcal{R}}(\rho_t) \wedge \rho_s \in \mathcal{R}_s, \rho_t \in \mathcal{R}_t\}$*. We further define* $\mathcal{P}(l_s, l_t) = \{\rho_s + \rho_t | (\rho_s, \rho_t) \in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})\}$ *with* $\rho_s + \rho_t$ *being the parallel rule of* ρ_s *and* ρ_t *[20, 19] as the set of* pair rules*.*

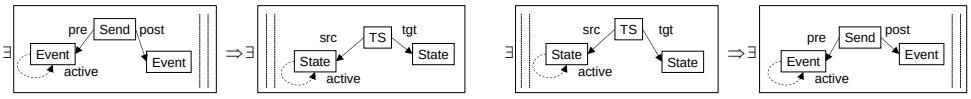
Definition 32 (pair constraint $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as given in Def. 26 and two relabelings* l_s *and* l_t *for* $\text{LTS}(\text{gts}_s, .)$ *and* $\text{LTS}(\text{gts}_t, .)$ *as given in Def. 30 with bijective mappings* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as well as the set of pairs* $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ *as given in Def. 31, the* pair constraint $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$ *is defined as* $\mathcal{C}_{\text{Pair}}^{\text{Rule}} = \mathcal{C}_{\text{Pair}}^{\text{Rule,f}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Rule,b}}$ *with*

$$\mathcal{C}_{\text{Pair}}^{\text{Rule,f}} = (\wedge_{(\rho_s,\rho_t)\in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})}(\text{App}(\rho_s) \Rightarrow \text{App}(\rho_t))) \qquad \textit{and}$$
$$\mathcal{C}_{\text{Pair}}^{\text{Rule,b}} = (\wedge_{(\rho_s,\rho_t)\in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})}(\text{App}(\rho_t) \Rightarrow \text{App}(\rho_s)))$$

typed over $S_{RT}C_{TT}T_{RT}$*.*

Note that as $l_s^{\mathcal{R}}$ and $l_t^{\mathcal{R}}$ are bijective mappings, we can conclude that the relation $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ in particular represents also a bijective mapping on $\mathcal{R}_s \times \mathcal{R}_t$.

Example 12 (Simple Equivalence: $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$)**.** *For the pair* $(send, fireTS) \in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ *of semantics rules of Example 9 the constraint* $\text{App}(send) \Rightarrow \text{App}(fireTS)$ *belonging to* $\mathcal{C}_{\text{Pair}}^{\text{Rule,f}}$ *depicted in Fig. 11 results.*



(a) $\text{App}(send) \Rightarrow \text{App}(fireTS)$ in $\mathcal{C}_{\text{Pair}}^{\text{Rule,f}}$ (b) $\text{App}(fireTS) \Rightarrow \text{App}(send)$ in $\mathcal{C}_{\text{Pair}}^{\text{Rule,b}}$

Figure 11: Fragment of pair constraint $\mathcal{C}_{\text{Pair}}^{\text{Rule}} = \mathcal{C}_{\text{Pair}}^{\text{Rule,f}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Rule,b}}$ for Example 9,15

Secondly, the *runtime constraint* $\mathcal{C}_{\text{RT}}$ is a constraint typed over $S_{RT}C_{TT}T_{RT}$ that can be specified manually, expressing – in addition to the automatically derived $\mathcal{C}^{\text{Rule}}_{\text{Pair}}$ – how the runtime structure of source and target language should be related to each other via the correspondences between them defined by the tgg in each pair of equivalent source and target states.

Definition 33 (Runtime constraint $\mathcal{C}_{\text{RT}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26, then the* runtime constraint $\mathcal{C}_{\text{RT}} = \mathcal{C}^{\text{f}}_{\text{RT}} \wedge \mathcal{C}^{\text{b}}_{\text{RT}}$ *is a graph constraint typed over* $S_{RT}C_{TT}T_{RT}$*.*

In particular, $\mathcal{C}^{\text{f}}_{\text{RT}}$ ($\mathcal{C}^{\text{b}}_{\text{RT}}$) expresses how the runtime structure of the source (target) language is related to the runtime structure of the target (source) language, respectively. Thus whereas the pair constraint comprises those source and target models that at least need to belong to the induced relation to become a bisimulation relation, the runtime constraint can be used to refine this minimal bisimulation relation manually according to the intuitive perception of behavioral equivalence for the specific model transformation.

Example 13 (Simple Equivalence: runtime constraint $\mathcal{C}_{\text{RT}}$)**.** *The runtime constraint for our Example 9, as depicted in Fig. 12, expresses that if an active loop on some event in the sequence chart domain occurs, then this event should be connected to a corresponding state with active loop in the automaton domain and the other way round. This runtime constraint is typed over the type graph* $S_{RT}C_{TT}T_{RT}$ *shown in Fig. 8.*

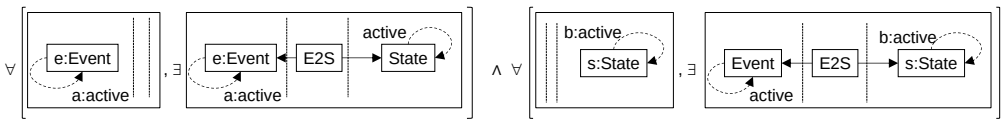


Figure 12: Runtime constraint $\mathcal{C}_{\text{RT}} = \mathcal{C}^{\text{f}}_{\text{RT}} \wedge \mathcal{C}^{\text{b}}_{\text{RT}}$ for Example 9,15

Finally, the *model transformation constraint* $\mathcal{C}^{\text{Man}}_{\text{MT}}$ is a constraint needed for verifying behavior preservation, but already satisfied by every model transformation instance. It manifests in the form of a graph constraint instead of in the form of a grammar which triple graphs belong to the triple graph grammar language and describe valid forward or backward model transformations w.r.t. behavior preservation, respectively.

Definition 34 (Model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26, then the* model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}} = \mathcal{C}_{\text{MT}}^{\text{Man,f}} \wedge \mathcal{C}_{\text{MT}}^{\text{Man,b}}$ *is a graph constraint typed over* $S_{TT}C_{TT}T_{TT}$*.*

The model transformation constraint should be chosen strong enough in order to be able to verify successfully behavior preservation as shown in the next section. There are different potential candidates for the model transformation constraint. In the following example, we illustrate one of these possibilities.

Example 14 (Simple Equivalence: model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}}$)**.** *For the current example, the model transformation constraint* $\mathcal{C}_{\text{MT}}^{\text{Man}}$ *is depicted in Fig. 13. It consists of a conjunction of constraints of the form* $\forall(P, \exists C)$ *such that* P *is the source component of the RHS of a TGG rule and* C *is the complete RHS of this TGG rule (in case of* $\mathcal{C}_{\text{MT}}^{\text{Man,f}}$*). When verifying the example for behavior preservation, as stated in [27], we interactively found out that this kind of model transformation constraint is sufficient for verifying successfully behavior preservation as will be described in Section 3.3.*

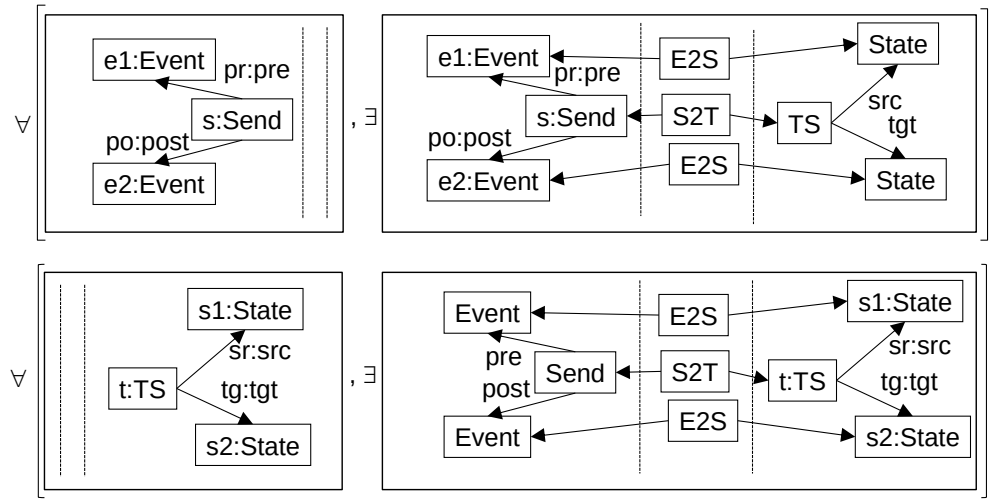


Figure 13: Fragment of model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}} = \mathcal{C}_{\text{MT}}^{\text{Man,f}} \wedge \mathcal{C}_{\text{MT}}^{\text{Man,b}}$ for Example 9,15

Note that in [27], we did not explicitly mention the model transformation constraint. Instead we assumed that the TGG constraint $\mathcal{C}_{\text{tgg}}$ is already strong enough and contains every constraint we need for the verification phase. If this was not

the case, then we interactively strengthened the regular TGG constraint with a constraint that does not modify the set of model transformation instances defined by $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$. Here, we make this process explicit by distinguishing $\mathcal{C}_{\mathrm{tgg}}$, the constraint actually defining the model transformation instances, from the model transformation constraint $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$, the constraint needed for successful behavior preservation verification satisfied already by any model transformation instance anyway.

Definition 35 (Bisimulation constraint $\mathcal{C}_{\mathrm{Bis}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .))$ *as introduced in Def. 26. Moreover, given relabelings* $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as given in Def. 30, then the* bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$ *with* $\mathcal{C}_{\mathrm{RT}}$ *a runtime constraint as given in Def. 33,* $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}}$ *the pair constraint derived from* $l_s^{\mathcal{R}}$ *and* $l_t^{\mathcal{R}}$ *as given in Def. 32 and* $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$ *a model transformation constraint as given in Def. 34.*

Table 6: Modeling step M_{pres} for Example 9,15

concept	behavioral equivalence			
formalization	relabelings l_s, l_t	induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}, .)$ with (bi-)simulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$		
artefact	mappings $l_s^{\mathcal{R}}$, $l_t^{\mathcal{R}}$	runtime constraint $\mathcal{C}_{\mathrm{RT}}$	pair constraint $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}}$	model transformation constraint $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$
depicted in	Fig. 9	Fig. 12, 30	Fig. 11, 31	Fig. 13, 34

3.3. Verification Scheme

Now we have all the necessary concepts at hand to describe the problem of verification of behavior preservation for model transformations at the transformation level in the sense of Definition 27, with Tables 3, 4, 5, and 6 listing the specific artefacts needed and introduced earlier (see Table 2) for modeling steps M_{lang},M_{sem},M_{trans} and M_{pres}. The following example illustrates in a summarizing way all modeling steps needed for enabling behavior preservation verification as described in the subsequent theorem.

Example 15 (Simple Equivalence)**.** *In summary, we can complete the modeling steps for the running example for Definition 27 as described in Example 9 as follows: The relabelings l_s and l_t as described in Example 11 map transitions of the* LTS *to a common alphabet A via bijective mappings $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$, which map equivalent rules in source and target GTS to the same element.* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is then behavior preserving in an equivalent manner, if for each pair of source and target models $(S,T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ it holds that $l_s(\mathrm{LTS}(\mathrm{gts}_s, S)) =_{bsim} l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$. In particular, we specify the bisimulation relation as an induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}, .)$ with the bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$ being the conjunction of the runtime constraint (see Example 13), pair constraint (see Example 12), source and target dynamic constraints, TGG constraint and model transformation constraint (see Example 14).*

More informally, we require that for each pair of a sequence chart and an automaton related by the model transformation, their behavior is equivalent in the sense that each rule application on the source model (or target model, respectively) can be followed by an equivalent rule application on an equivalent target model (or source model, respectively) such that the resulting source and target models are equivalent again.

In the following theorem, we present a *verification scheme* for showing that the desired bisimulation relation $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}, .)$ with (bi-)simulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$ as given in the previous section indeed defines a bisimulation between source and target semantics of each source and target model of the model transformation. In addition, we prove the correctness of the verification scheme. In particular, it consists of three steps: (V_{init}) one simple constraint satisfaction check on the axiom of the triple graph grammar defining the model transformation, (V_{trans}) one invariant check on the triple graph grammar rules and (V_{sem}) one invariant check on pair rules of source and target GTSs defining the source and target model semantics, respectively.

Theorem 1 (bisimulation verification[27][5])**.** *Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .))$ as introduced in Def. 26. Moreover, given relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ and $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *derived from $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ as given in Def. 30, a bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Man}}$*

[5]This theorem is a slightly revised version of Theorem 3 in [27]. In [27], we considered a bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Rule}}$, which was then interactively strengthened by additional constraints holding anyway for all model transformation instances, but necessary for successful verification with our invariant checking tool [2, 15]. Here, our approach establishes these additional constraints explicitly such that they are considered directly in the verification theorem.

typed over $S_{RT}C_{TT}T_{RT}$ as given in Def. 35, then the model transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is behavior preserving in an equivalent manner via the induced bisimulation relation* $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}, .)$ *(see Def. 29) in the sense of case 3.1 of Definition 27 if the following conditions are fulfilled:*

V_{init}: $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$.

V_{trans}: $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}})$.

V_{sem}: $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{P}(l_s, l_t), S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t})$ *with* $\mathcal{P}(l_s, l_t)$ *as given in Def. 31.*

Proof. Given some $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$, then we know that there exists some SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ such that it suffices to show that $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ and $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ are bisimilar via the induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$. We therefore prove (1) that the pair of initial states (S, T) of $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ and $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ is always in $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ (2) and that $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ is indeed a bisimulation relation according to conditions 1 and 2 of Def. 25.

(1) $(S, T) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$*:* Each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills $\mathcal{C}_{\mathrm{tgg}}$ by construction. We further prove by induction over the number of TGG rule applications that each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills also $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$ such that according to Def. 29 $(S, T) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$. The *base clause* for the axiom $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$ follows directly from condition (V_{init}) of the Theorem. Condition (V_{trans}) of the Theorem then provides the *induction step* that for any TGG rule application $S_n C_n T_n \Rightarrow_{\mathcal{R}} S_{n+1} C_{n+1} T_{n+1}$ it holds that $S_{n+1} C_{n+1} T_{n+1} \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$ assuming the *induction hypothesis* that $S_n C_n T_n \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$.

(2) $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ *is a bisimulation relation:* We first have to show for condition 1 of Def. 25 that for all $(S_1, T_1) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ (equivalent to $S_1 C T_1 \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}$ according to Def. 29), if $S_1 \xrightarrow{\alpha} S_2$, then $T_1 \xrightarrow{\alpha} T_2$ and $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ for $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ and $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$, respectively. This holds if $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$ implies $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ with $l_s(\rho_s, m_s) = l_t(\rho_t, m_t)$ and $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Rule}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Man}}_{\mathrm{MT}}, SCT)$ for ρ_s in gts_s and ρ_t in gts_t and m_s and m_t a match for ρ_s and ρ_t, respectively. We first prove that $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ if $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$ with $l_s(\rho_s, m_s) = l_t(\rho_t, m_t)$. If we have $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$, then we also

have $S_1CT_1 \Rightarrow_{\rho_s} S_2CT_1$. Because $S_1CT_1 \vDash \mathcal{C}^{\text{Rule}}_{\text{Pair}}$, and in particular $S_1CT_1 \vDash \mathcal{C}^{\text{Rule}^f}_{\text{Pair}}$, applicability of ρ_s to S_1CT_1 implies applicability of ρ_t to S_1CT_1 such that $S_1CT_1 \Rightarrow_{\rho_t} S_1CT_2$ with $l^{\mathcal{R}}_s(\rho_s) = l^{\mathcal{R}}_t(\rho_t)$. This means, in particular, that $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ via some match m_t such that $l_s(\rho_s, m_s) = l^{\mathcal{R}}_s(\rho_s) = l^{\mathcal{R}}_t(\rho_t) = l_t(\rho_t, m_t)$. We still need to prove that $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}, SCT)$. This is because, as ρ_s and ρ_t consist of disjoint types, they can only be applied in a parallel independent way to S_1CT_1. Due to the Parallelism Theorem [20], then it follows that $S_1CT_1 \Rightarrow_{\rho_s+\rho_t} S_2CT_2$ with $\rho_s + \rho_t \in \mathcal{P}(l_s, l_t)$. As gts_s and gts_t are runtime conform, they preserve static types, $\mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}$ typed over $S_{TT}C_{TT}T_{TT}$ are by construction inductive invariants for $\mathcal{P}(l_s, l_t)$ implying $S_2CT_2 \vDash \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}$. Moreover, $S_2CT_2 \vDash \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t$ by construction as well, since S_2CT_2 already satisfies $\mathcal{C}_S$ and $\mathcal{C}_T$ (comprised in $\mathcal{C}_{\text{tgg}}$) and again because of runtime conformity of gts_s and gts_t the GTSs with constraint $\text{gts}^{\mathcal{C}_S}_s$ and $\text{gts}^{\mathcal{C}_S}_t$ have the dynamic constraints $\mathcal{C}^{gts}_s$ and $\mathcal{C}^{gts}_t$ as inductive invariants, respectively. Since gts_s and gts_t preserve static types S_2CT_2 will in particular also satisfy $\mathcal{C}_S$ and $\mathcal{C}_T$ such that it is enough for the GTSs with constraint to have the dynamic constraints $\mathcal{C}^{gts}_s$ and $\mathcal{C}^{gts}_t$ as inductive invariants, respectively. Because of condition (V_{sem}) of the Theorem and the fact that $S_1CT_1 \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}$, then it follows that $S_2CT_2 \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}$. Thus, according to Def. 29 this means that $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Man}}_{\text{MT}}, SCT)$. Condition 2 of Def. 25 follows analogously to condition 1 as the roles of S and T are symmetric. □

$$
\begin{array}{ll}
S_AC_AT_A & S_AC_AT_A \\
 & \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}^{\text{Man}}_{\text{MT}} \ (V_{init}) \\
\mathcal{R} \Big\Downarrow * & \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}^{\text{Man}}_{\text{MT}} \\
 & \text{is an inductive invariant of} \\
 & (\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\text{tgg}}) \ (V_{trans}) \\
SCT & \\
\mathcal{P}(l_s,l_t) \Big\Downarrow * & \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \text{ is an inductive invariant} \\
 & \text{of } (\mathcal{P}(l_s, l_t), S_{RT}C_{TT}T_{RT}, \cdots) \ (V_{sem}) \\
S_*CT_* &
\end{array}
$$

The sketch above summarizes how the constraints are preserved via the conditions starting from the TGG axiom, via the TGG rules, and finally also during the execution of pair rules of the semantics. While condition V_{init} and V_{trans} are employed in step (1) of the proof via an induction, condition V_{sem} allows to show that the induced relation is indeed a bisimulation in step (2) of the proof.

Example 16 (Simple Equivalence: Verification)**.** *The model transformation described in Example 15 can be shown to be behavior preserving via the bisimulation relation induced*

by the bisimulation constraint $\mathcal{C}_{\text{Bis}}$, since the conditions V_{init},V_{trans} and V_{sem} as given in Theorem 1 hold.

Since the correspondence of runtime elements as described in $\mathcal{C}_{\text{RT}}$ and the applicability of equivalent rules as expressed in $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$ hold for the axiom of the triple graph grammar (condition V_{init}) and are invariant for each possible application of a triple graph rule (condition V_{trans}) from the example TGG tgg *(with TGG constraint $\mathcal{C}_{\text{tgg}}$), each possible triple graph generated by the triple graph grammar (describing a pair of a sequence chart and an automaton being a model transformation instance) also satisfies these constraints.*

Since the correspondence of runtime elements ($\mathcal{C}_{\text{RT}}$) and the applicability of equivalent rules ($\mathcal{C}_{\text{Pair}}^{\text{Rule}}$) remains invariant for each pairwise application of equivalent source and target semantics rules, all model transformation instances and thus, the model transformation as such, are behavior preserving. More informally, each rule application on the source model (or target model, respectively) can be followed by an equivalent rule application on an equivalent target model (or source model, respectively) such that the resulting source and target model are equivalent again.

In Section 6 we describe in detail how the checks of conditions V_{init},V_{trans} and V_{sem} can be automated.

3.4. Limitations

The results of [27] presented so far employ a rather coarse grain approach concerning the labeling. However, if we want to check bisimilarity between a source and target model different levels of abstraction may be necessary. The check enforced by $\mathcal{C}_{\text{Pair}}^{\text{Rule}}$ as supported in the presented verification scheme of [27] is quite coarse, since the derived bisimulation relation defines pairs of transitions as the same if GTS rules mapped to the same label are realizing it. In particular, the rule matches leading to the transitions is "forgotten" by the relabeling. Often this is too abstract since a lower level of abstraction may reflect details that go beyond the information which semantics rule was executed. E.g., a message name or even message parameter values etc, may be important as well. However, the relabeling supported by [27] excludes such information and we will study in the next section a relabeling approach that is not limiting us in this respect.

A particular example where the relabeling supported by [27] would be too coarse grain are when the semantics of a particular LTS is non-deterministic such that alternative rule applications of the same rule for different matches are possible. Then, there are multiple combinations of source and target steps mapped to the same label where some combinations may lead to a combined state, where equivalently labeled steps are applicable again, but at the same time other combinations may lead also to a combined state, where equivalently labeled steps are not both

applicable anymore. In the presented verification scheme of [27] this would result in refusing bisimilarity due to the too coarse grain relabeling when checking the conditions of Theorem 1, while if a more fine grain relabeling would have been used bisimilarity could be established.

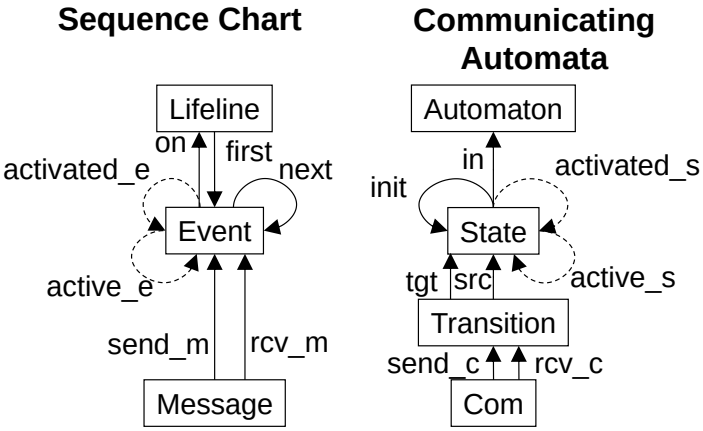
The following variant of our example shows that a more fine grain relabeling of the LTS based on the rule *and the match* is required in some cases.

Example 17 (Complex example: Sequence Chart to Communicating Automata)**.** *In the example of the previous section, the induced labeled transition systems for source and target semantics were deterministic. In each state only one rule via one particular match is applicable. However, in general a sequence chart consists of more than one lifeline such that the communication between different objects can be modeled.*

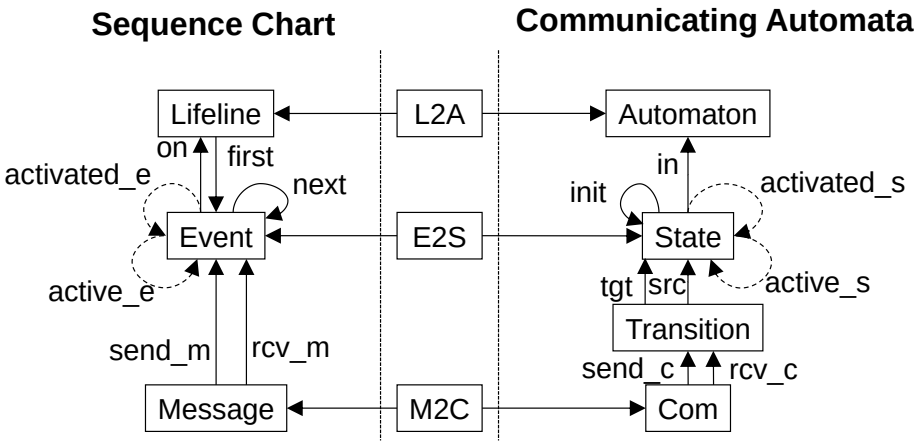
Using a model transformation we can synthesize so-called communicating automata from these sequence charts, describing this inter-object communication in a state-based way. Fig. 14(a) depicts the type graph with dynamic types active-e *and* activated-e *of the sequence chart language. A chart consists of several lifelines on which events occur from which one of them is the first event and between each pair of subsequent events connected by a* next *edge messages are being sent or received. In Fig. 14(b) the type graph with dynamic types* active-s *and* activated-s *of the communicating automata language is depicted. It consists of several automata that contain states and one of these states is the initial state. States can be connected via transitions and transitions of different automata are connected via a node of type* Com *if they can only be triggered together.*

The operational semantics of each sequence chart (communicating automaton) is defined by the GTS gts_s *(*gts_t*) as depicted in Fig. 15(a) (15(b)). Thereby* gts_s *consists of the* initE *and* send *rule, whereas* gts_t *consists of the* initS *and* fire *rule. Finally, the TGG* $\mathcal{R}$ *depicted in Fig. 16 defines a model transformation between life sequence charts and communicating automata. It is typed over* $S_{RT}C_{TT}T_{RT}$ *as depicted in Fig. 14(c).*

In Fig.17, an example source sequence chart S *and target communicating automaton* T *is shown together with their respective induced labeled transition systems. Note that these LTSs are non-deterministic. Suppose that we map* initE *and* initS *plus* send *and* fire *to the same labels in a common label alphabet. As illustrated in Fig.17,* $\mathcal{C}^{\text{Rule}}_{\text{Pair}}$ *now would not represent an inductive invariant for* $(\mathcal{P}(l_s, l_t), S_{RT}C_{TT}T_{RT})$*. It is possible to reach state* $(S4, T5)$ *with equivalently labeled rules, because the relabeling notion introduced earlier does not take matches into account. However, in said state* $(S4, T5)$ *a problem arises, since in* $S4$ send *can be applied, but in* $T5$ *the equivalently labeled rule* fire *is not applicable. In fact, the real problem w.r.t. behavior preservation arises already during the first parallel rule application with equivalently labeled rules* initE *and* initS *from* (S, T) *to* $(S1, T3)$*, which leads to a violation of our runtime constraint. This is because* initE *initializes the first lifeline and* initS *initializes the automaton corresponding to the last lifeline instead. More generally,* initS *(*initE*) can, at first, be applied at three different lifelines (automata).*

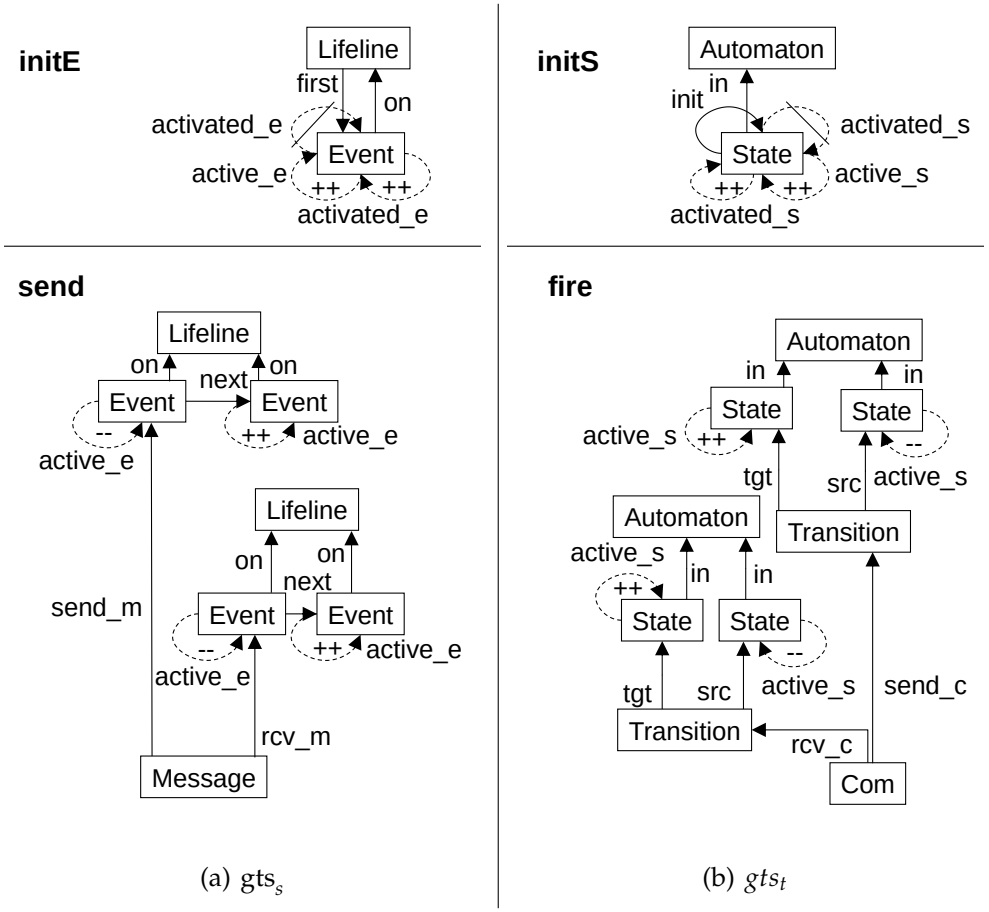


(a) Type graph S_{RT} (b) Type graph T_{RT}



(c) Type graph $S_{RT}C_{TT}T_{RT}$

Figure 14: Type graphs for complex example



(a) gts_s (b) gts_t

Figure 15: Operational semantics: gts_s and gts_t for complex example

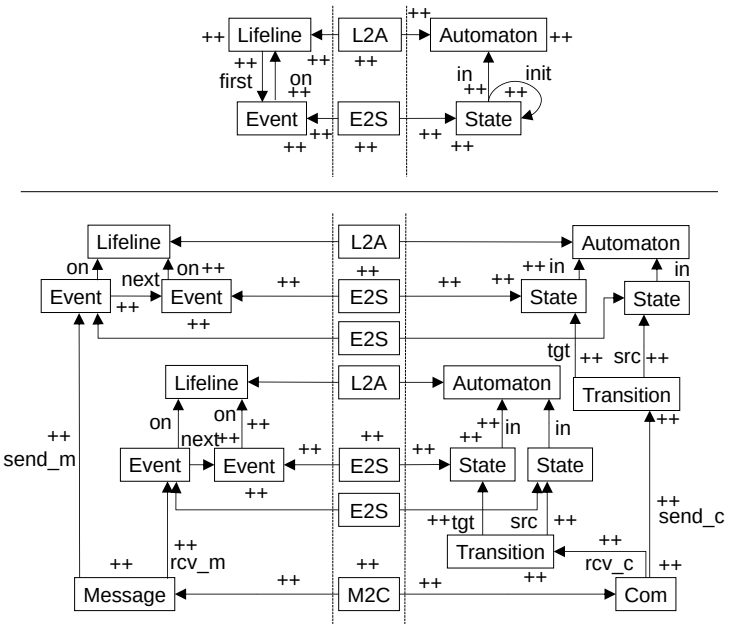


Figure 16: TGG rules $\mathcal{R}$ for complex example

We can thus observe that the approach to equivalence verification presented in [27] is only able to handle model transformations from source to target languages with semantics that do not include non-deterministic behavior w.r.t. the choice of rule matches.

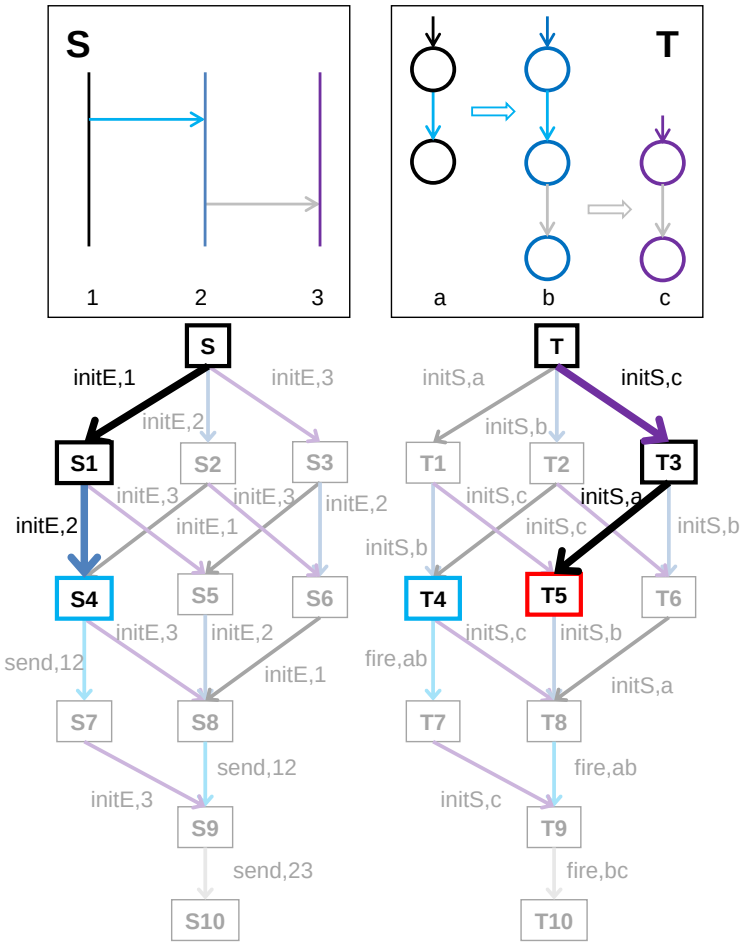


Figure 17: S and T in concrete syntax, $\mathrm{LTS}(\mathrm{gts}_s, S)$ and $\mathrm{LTS}(\mathrm{gts}_t, T)$ with two highlighted traces that have equivalent labels but lead to a state pair that is not bisimilar

4. Behavioral Equivalence Verification

As discussed, the results presented for equivalence in the former section are somehow limited. Therefore, we will in this section consider an approach to overcome these limitations.

4.1. Modeling Step M_{pres}

One possibility to solve the problem described in Section 3.4 is to take into account the rule matches when defining the relabelings in contrast to discarding them as done by the relabelings defined in Section 3. A first obvious option would be to relate matches by comparing concrete attribute values for matched elements. However, to study all pairs of rules including such attribute values requires a symbolic treatment with attribute conditions (equivalence conditions) as otherwise all the usually infinitely many possible attributed values would require an explicit consideration. A less demanding and more straightforward option is to exploit the correspondence structures generated by the TGG, encoding traceability between source and target models, which allow a very natural differentiation in contrast to only externally visible details as present in any kind of labeling. In particular, we will consider source and target semantics steps to be equivalent if not only the rules are equivalent, but the corresponding matches are connected via specific correspondence structures generated by the TGG.

4.1.1. Relabeling

We therefore first describe how the relabelings l_s and l_t based on correspondence structures generated by the TGG can be defined. Then we derive the desired bisimulation relation between relabeled source and target transition systems from a given graph constraint $\mathcal{C}^{\text{Cor}}_{\text{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$, called the *bisimulation constraint*.

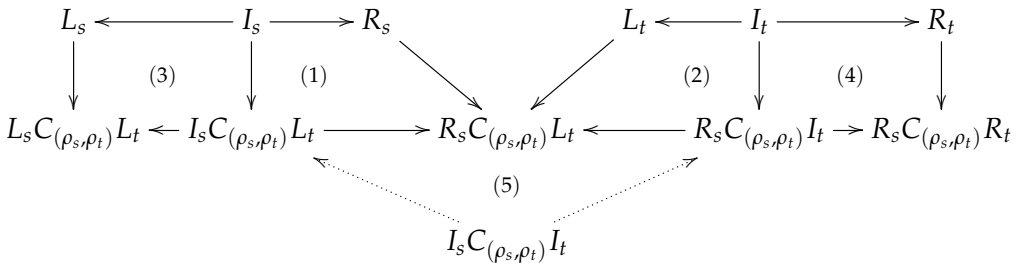
We can relate the rules of source and target semantics using two bijective mappings $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ and $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$, mapping source and target rules to the same alphabet A. In addition we want to compare matches. In particular, two matches for equivalent rules ρ_s and ρ_t will be said to be equivalent, if specific correspondences $C_{(\rho_s,\rho_t)}$ can be found in between their co-domains. In general, those correspondences can be found automatically (see Section 6.1).

Definition 36 (correspondences between source and target rules $C_{(\rho_s,\rho_t)}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT},\mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT},\mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg},\mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s,.), \text{LTS}(\text{gts}_t,.))$ *as introduced in Def. 26 as well as two bijective mappings* $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ *and* $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$*, then for each source and target rule* ρ_s *and* ρ_t *belonging to* $\text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)$ *as given in Def. 31, the* correspondences $C_{(\rho_s,\rho_t)}$ *between* ρ_s *and* ρ_t *consist of a fixed number*

of correspondence edges from correspondence nodes to source nodes of static types in the LHS L_s of ρ_s and to target nodes of static types in the LHS L_t of ρ_t such that $L_sC_{(\rho_s,\rho_t)}L_t$ is a well-defined triple graph over $S_{RT}C_{TT}T_{RT}$.

We then require that equivalent source and target semantics rules need to be applicable together along these correspondence structures. The effect of such a source and target semantics step can be described using a so-called pair rule with correspondences.

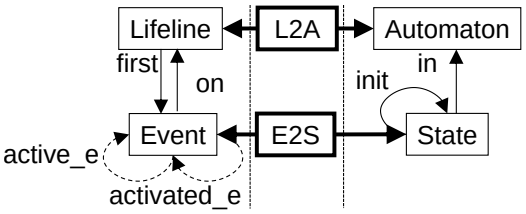
Definition 37 ($\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$, $\mathcal{P}(l_s,l_t)^{\mathrm{Cor}}$)**.** *Given a formally covered model transformation $(\mathcal{L}(S_{TT},\mathcal{C}_S),\mathcal{L}(T_{TT},\mathcal{C}_T),\mathrm{MT}(\mathrm{tgg},\mathcal{C}_{\mathrm{tgg}}),\mathrm{LTS}(\mathrm{gts}_s,.),\mathrm{LTS}(\mathrm{gts}_t,.))$ as introduced in Def. 26 as well as two bijective mappings $l_s^{\mathcal{R}}:\mathcal{R}_s\to A$ and $l_t^{\mathcal{R}}:\mathcal{R}_t\to A$, then for each source and target rule ρ_s and ρ_t belonging to* Pair$(l_s^{\mathcal{R}},l_t^{\mathcal{R}})$ *as given in Def. 31 and $C_{(\rho_s,\rho_t)}$ as given in Def. 36, the* pair rule with correspondences *$\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ equals the E-concurrent rule [20, 19] of ρ_s and ρ_t via the E-dependency relation $R_s\to R_sC_{(\rho_s,\rho_t)}L_t\leftarrow L_t$ with underlying plain rule $p_s *_{R_sC_{(\rho_s,\rho_t)}L_t} p_t : L_sC_{(\rho_s,\rho_t)}L_t \leftarrow I_sC_{(\rho_s,\rho_t)}I_t \to R_sC_{(\rho_s,\rho_t)}R_t$ as depicted in the diagram below where all morphisms are inclusions.*



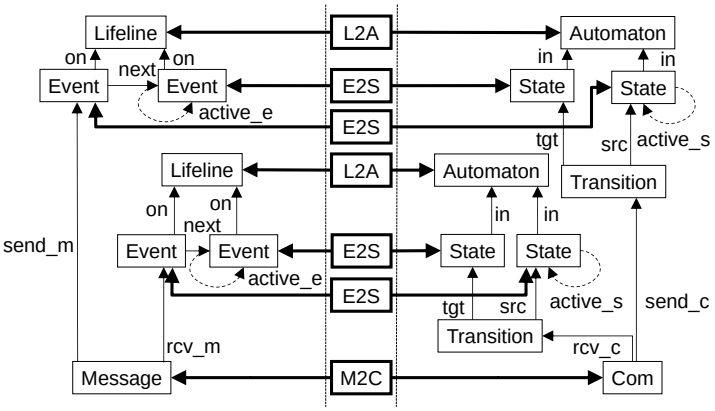
Analogously, the pair rule with correspondences *$\rho_t *_{R_tC_{(\rho_s,\rho_t)}L_s} \rho_s$ equals the E-concurrent rule [20, 19] of ρ_t and ρ_s via the E-dependency relation $R_t\to R_tC_{(\rho_s,\rho_t)}L_s\leftarrow L_s$ consisting of inclusions.*
We define $\mathcal{P}(l_s,l_t)^{\mathrm{Cor}} = \{\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t | (\rho_s,\rho_t)\in \mathrm{Pair}(l_s^{\mathcal{R}},l_t^{\mathcal{R}})\}$ as the set of pair rules with correspondences*.*

Remark 7. *Since $C_{(\rho_s,\rho_t)}$ only connects nodes of static types and source and target rules preserve static types, it follows that all graphs in the concurrent rule construction illustrated in Def. 37 are well-defined triple graphs.*

Example 18 (Equivalence: correspondences between source and target rules $C_{(\rho_s,\rho_t)}$)**.** *For the current example, Fig. 18 depicts the triple graphs $R_sC_{(initE,initS)}L_t$ and $R_sC_{(send,fire)}L_t$ where R_s and L_t are the RHS and LHS of initE and initS or send and fire, respectively, and $C_{(initE,initS)}$ and $C_{(send,fire)}$ (drawn with bold lines) are the correspondences between the respective source and target rules. Both triple graphs are typed over $S_{RT}C_{TT}T_{RT}$.*



(a) $R_sC_{(initE,initS)}L_t$



(b) $R_sC_{(send,fire)}L_t$

Figure 18: Correspondences between source and target rules

Remark 8. *Following the terminology of [20, 19], we would call the rule* $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ *actually concurrent rule via the E-dependency relation with* $E = R_sC_{(\rho_s,\rho_t)}L_t$*. However, since* ρ_s *and* ρ_t *have disjoint types, we opted to choose the more intuitive name* pair rule with correspondences*. It expresses that* $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ *is applicable if and only if* ρ_s *and* ρ_t *are actually applicable also in parallel with matchings following the correspondences expressed in* $C_{(\rho_s,\rho_t)}$*. In other words,* $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ *is isomorphic to* $\rho_t *_{R_tC_{(\rho_s,\rho_t)}L_s} \rho_s$ *since it does not matter if the source/target rule is applied first along the correspondences. As a consequence, we could have defined* $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ *also as the set* $\{\rho_t *_{R_tC_{(\rho_s,\rho_t)}L_s} \rho_s | (\rho_s, \rho_t) \in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})\}$*, since the respective rules therein would have the same effect.*

Example 19 (Equivalence: $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$, $\mathcal{P}(l_s, l_t)^{\text{Cor}}$)**.** *For the pairs* $(initE, initS)$ *and* $(send, fire)$ *in* $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$*, Fig. 38 depicts the resulting set of pair rules with correspondences* $\mathcal{P}(l_s, l_t)^{\text{Cor}}$*, constructed as* $initE *_{R_sC_{(initE,initS)}L_t} initS$ *and* $send *_{R_sC_{(send,fire)}L_t} fire$ *via the correspondences described in Example 18.*

Now we can define the relabelings l_s and l_t taking into account correspondence structures between matches of equivalent source and target rules in source and target models, respectively.

Definition 38 (relabelings l_s and l_t)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26 as well as two bijective mappings* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *into some common label alphabet* A *with* $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ *as given in Def. 31 and* $C_{(\rho_s,\rho_t)}$ *for each source and target rule* ρ_s *and* ρ_t *belonging to* $\text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ *as given in Def. 36, then the* relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to \mathcal{A} \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_s \to \mathcal{A} \times \mathcal{M}$ *with* $\mathcal{M}$ *a common alphabet for matches are defined as follows:*

$$l_s(\rho_s, m_s) = (l_s^{\mathcal{R}}(\rho_s), m) \text{ and } l_t(\rho_t, m_t) = (l_t^{\mathcal{R}}(\rho_t), m')$$

with

$$m = m'$$

if (1) $(\rho_s, \rho_t) \in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ *and (2)* $S_1 \Rightarrow_{\rho_s,m_s} S_2$ *in* $\text{LTS}(\text{gts}_s, S)$ *and* $T_1 \Rightarrow_{\rho_t,m_t} T_2$ *in* $\text{LTS}(\text{gts}_t, T)$ *for some* $(S, T) \in \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}})$ *because there exists some SCT in* $\mathcal{L}(\text{tgg}, \mathcal{C}_{\text{tgg}})$ *such that a concurrent transformation* $S_1CT_1 \Rightarrow_{\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t, m} S_2CT_2$ *via the extended match m exists with the source and target components of m equal to* m_s *and* m_t*, resp., and*

$$m \neq m'$$

otherwise.

By requiring a match for $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ (including the correspondences $C_{(\rho_s,\rho_t)}$) for label equivalence, the relabeling functions exploit the correspondence structures

created by the TGG to compare matches and applications of source and target rules. This is a fundamental difference to the relabeling functions introduced in Definition 30 of Section 3 ignoring the matches completely.

Example 20 (Equivalence: relabelings l_s and l_t)**.** *For specific source and target models S and T being the source and target component of a triple graph SCT, Figure 19 depicts two different example matches $m_s : L_s \to S$ and $m_t : L_t \to T$ for rules initE and initS with $l_s^{\mathcal{R}}(initE) = init = l_t^{\mathcal{R}}(initS)$ and left hand sides L_s and L_t, respectively. Figure 20 shows how these matches can be extended to matches for $initE *_{R_s C_{(initE,initS)} L_t} initS$ (Example 19) in the triple graph SCT such that we have $m = m'$ and consequently $l_s(initE, m_s) = (init, m) = l_s(initS, m_t)$.*

Note that this inclusion of matches and correspondence structures in the relabelings serves to avoid the limitation described in Example 17 and Figure 17. The transitions $(S, initE, S1)$ and $(T, initS, T1)$ were considered equivalent due to the relabeling functions defined in Definition 30 previously discarding matches, which resulted in $l_s(initE, m_s) = init = l_t(initS, m_t)$. With Definition 38 taking matches into account, the result is $l_s(initE, m_s) \neq l_t(initS, m_t)$ because the required extension of m_s and m_t along the correspondence structure $C_{(initE,initS)}$ is not possible.

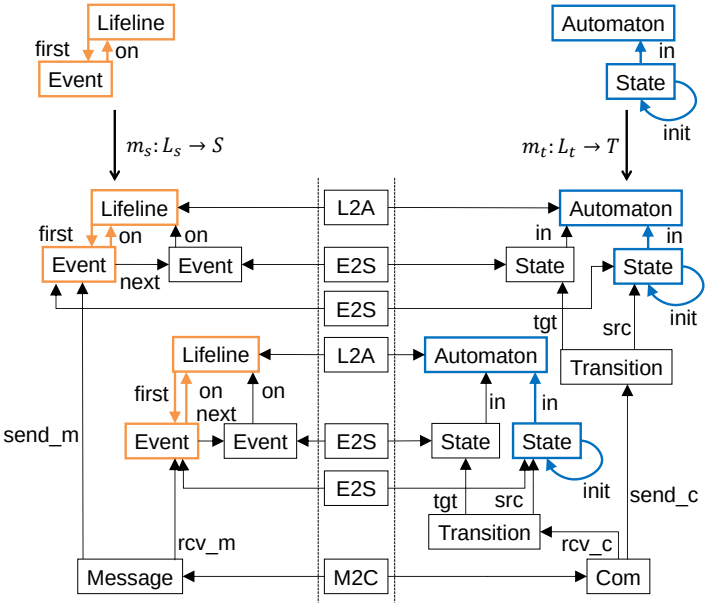


Figure 19: Matches for *initE* and *initS*

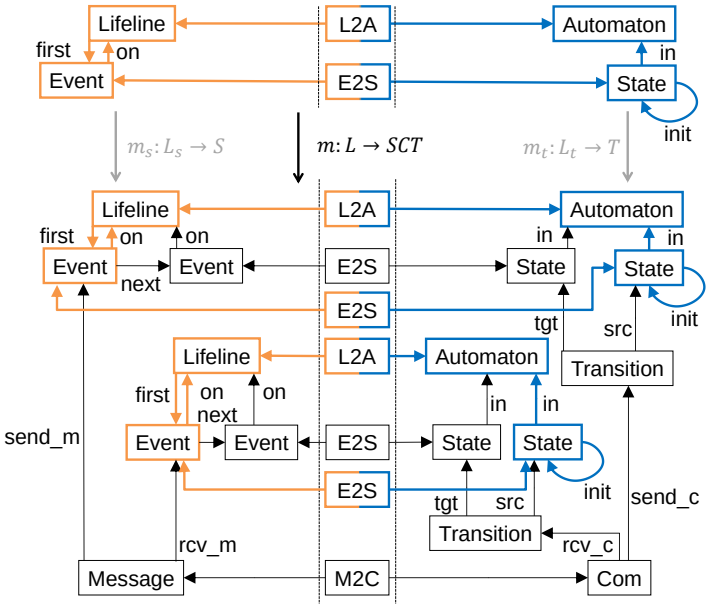


Figure 20: Extended matches for $initE *_{R_s C_{(initE,initS)} L_t} initS$

4.1.2. Bisimulation

Having defined relabelings taking into account correspondences between source and target models, we now describe the desired bisimulation relation between the relabeled transition systems. In particular, it is derived as induced relation (see Def. 29) from the bisimulation constraint $\mathcal{C}^{\text{Cor}}_{\text{Bis}} = \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$. The runtime constraint $\mathcal{C}_{\text{RT}}$ is as given in Def. 33. $\mathcal{C}^{gts}_{s}$ and $\mathcal{C}^{gts}_{t}$ are the dynamic constraints as introduced in Def. 17 and the TGG constraint $\mathcal{C}_{\text{tgg}}$ is as introduced in Def. 21. The pair constraint with correspondences $\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ expresses that the applicability of a source semantics rule implies that an equivalent target semantics rule is applicable via an extended match that links the predefined correspondences between equivalent source and target semantics rules and the other way round.

Definition 39 (pair constraint $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ with correspondences)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26 as well as two bijective mappings* $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ *and* $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$ *with* $\text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)$ *as given in Def. 31 and corresponding pair rules with correspondences as given in Def. 37, the* pair constraint $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ with correspondences *is defined as*

$\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ *with*

$$\mathcal{C}^{\text{Cor,f}}_{\text{Pair}} = \wedge_{(\rho_s,\rho_t)\in \text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)}(\forall(L_s, \text{ac}_{\text{App}(\rho_s)} \Rightarrow \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t, \text{ac}_{\text{App}(\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t)}))$$

and

$$\mathcal{C}^{\text{Cor,b}}_{\text{Pair}} = \wedge_{(\rho_s,\rho_t)\in \text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)}(\forall(L_t, \text{ac}_{\text{App}(\rho_t)} \Rightarrow \exists(j_t : L_t \to L_s C_{(\rho_s,\rho_t)} L_t, \text{ac}_{\text{App}(\rho_t *_{R_t C_{(\rho_s,\rho_t)} L_s} \rho_s)}))$$

typed over $S_{RT}C_{TT}T_{RT}$ *with* $\rho_s = \langle p_s : \langle L_s \hookleftarrow I_s \hookrightarrow R_s\rangle, \text{ac}_{L_s}\rangle$, $\rho_t = \langle p_t : \langle L_t \hookleftarrow I_t \hookrightarrow R_t\rangle, \text{ac}_{L_t}\rangle$, j_s *and* j_t *inclusion morphisms and* $\text{ac}_{\text{App}(\rho)}$ *for a given rule* ρ *as introduced in Def. 28.*

Example 21 (Equivalence: pair constraint with correspondences)**.** *In Fig. 45 the pair constraint with correspondences for Example 17 is depicted. In particular, for the* initE *and* initS *rules it is required that they are applicable in parallel via the correspondences* L2A *and* E2S *relating the lifeline to the corresponding automaton and the first event to the initial state. Analogously for rules* send *and* fire *it is required that they are applicable in parallel via the correspondences* L2A*,* E2S *and* M2C *connecting the events in each lifeline with corresponding states in the corresponding automaton and a message between lifelines with a corresponding communication in the communicating automaton. Note that the constraint depicted is simplified already according to the fact that the Deletable condition as given in Def. 28 is trivially true for each rule, the construction of concurrent rules with application conditions as explained more in detail in [20, 19] and the following equivalence:*

$$\text{ac}_{\text{App}(\rho_s)} \Rightarrow \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t, \text{ac}_{\text{App}(\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t)})$$

equivalent to

$$\neg\text{ac}_{\text{App}(\rho_s)} \vee \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t, \text{ac}_{\text{App}(\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t)}).$$

For Example 17 this means that only the pairs (Si, Ti) *with the same index in Fig. 17 fulfill the pair constraint with correspondences* $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$*, whereas pairs such as* $(S3, T1)$ *only fulfill the pair constraint* $\mathcal{C}^{\text{Rule}}_{\text{Pair}}$ *without correspondences.*

Finally, we have to specify the *model transformation constraint* $\mathcal{C}^{\text{Cor}}_{\text{MT}}$. In general, we want the static part of the pair constraint with correspondences to be satisfied by each triple graph of the model transformation under consideration. This is a prerequisite for the dynamic structures to correspond as required by the pair constraint. Therefore, the model transformation constraint $\mathcal{C}^{\text{Cor}}_{\text{MT}}$ is defined as a graph constraint typed over the static type graph $S_{TT}C_{TT}T_{TT}$. In order to choose a useful and suitable model transformation constraint that establishes said prerequisite for the dynamic structures, we propose to impose the following requirement on the model transformation constraint.

Definition 40 (Model transformation constraint $\mathcal{C}^{\text{Cor}}_{\text{MT}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26 and given static and runtime triple type graphs* $S_{TT}C_{TT}T_{TT}$ *and* $S_{RT}C_{TT}T_{RT}$*, the pair constraint* $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ *(Def. 39), and the type restriction operation (here: to the static triple type graph), denoted* $_{|S_{TT}C_{TT}T_{TT}}$*, then the* model transformation constraint $\mathcal{C}^{\text{Cor}}_{\text{MT}} = \mathcal{C}^{\text{Cor,f}}_{\text{MT}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{MT}}$ *is a graph constraint typed over* $S_{TT}C_{TT}T_{TT}$ *such that the following holds:*

$$\begin{aligned}\forall G(\ &(G \in \mathcal{L}(S_{TT}C_{TT}T_{TT}) \wedge G \not\models \mathcal{C}^{\text{Cor}}_{\text{MT}}) \\ &\Rightarrow \exists G'(G'_{|S_{TT}C_{TT}T_{TT}} = G \wedge G' \in \mathcal{L}(S_{RT}C_{TT}T_{RT}) \wedge G' \not\models \mathcal{C}^{\text{Cor}}_{\text{Pair}})\ ).\end{aligned}$$

The rationale behind this declarative definition is to eliminate a source of errors leading to non-behavior preserving model transformations by linking the model transformation constraint to the static part of the pair constraint. All triple graphs that are a result of the model transformation should satisfy the static restriction of the pair constraint. If that is not the case, i.e. if there is a triple graph violating the respective constraint ($G \not\models \mathcal{C}^{\text{Cor}}_{\text{MT}}$), the above definition ensures the existence of a triple graph G' isomorphic to G except for additional runtime elements ($G'_{|S_{TT}C_{TT}T_{TT}} = G \wedge G' \in \mathcal{L}(S_{RT}C_{TT}T_{RT})$) such that G' violates the pair constraint ($G' \not\models \mathcal{C}^{\text{Cor}}_{\text{Pair}}$), which may constitute a non-behavior preserving transformation. In short, the violation of the model transformation constraint leads to a possible violation of the pair constraint.

It should be noted that with this definition, the requirement on the model transformation constraint's validity for all model transformation instances (or rather, their triple graphs) is an over-approximiation. In some cases, a violation of the model transformation constraint by a static triple graph from a model transformation will, by the requirement above, lead to a dynamic triple graph violating the pair constraint – however, this triple graph may not occur in any execution trace of the source and target model's behavior and, hence, may not actually constitute a violation of the pair constraint and the bisimulation requirement. In those cases, the source (or) target model contains dead code – the static part required for executable behavior is present, but the execution is prevented by additional constraints or unreachable states.

Example 22 (Equivalence: model transformation constraint $\mathcal{C}^{\text{Cor}}_{\text{MT}}$)**.** *Fig.46 depicts the model transformation constraint chosen for our running example. It is the restriction of the pair constaint with correspondences* $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ *to the type graph* $S_{TT}C_{TT}T_{TT}$ *without runtime elements. Note that both directions are included in the model transformation constraint, similar to the pair constraints with correspondences. The core idea is that in order for the pair constraint with correspondences to be satisfied, the triple graphs in question also*

have to satisfy the static parts of the constraints. In contrast to the first notion of a model transformation constraint $\mathcal{C}^{\text{Man}}_{\text{MT}}$ as in Example 14, $\mathcal{C}^{\text{Cor}}_{\text{MT}}$ is derived from the pair constraint, which establishes a more intuitive relation to the model semantics.

Due to the highly variable (for different model transformations) nature of the pair constraint, which is derived from the semantics rules, and the expressiveness of nested application conditions, an automatic derivation for the model transformation constraint in the general case is difficult to establish. We discuss the approach applied for our example in Sections 6 and 7; particularly in Construction 1 of Section 7.1.

Definition 41 (Bisimulation constraint $\mathcal{C}^{\text{Cor}}_{\text{Bis}}$)**.** *Given a formally covered model transformation $(\mathcal{L}(S_{TT},\mathcal{C}_{\mathcal{S}}),\mathcal{L}(T_{TT},\mathcal{C}_{\mathcal{T}}),\text{MT}(\text{tgg},\mathcal{C}_{\text{tgg}}),\text{LTS}(\text{gts}_s,.),\text{LTS}(\text{gts}_t,.))$ as introduced in Def. 26. Moreover, given relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ and $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ for $\text{LTS}(\text{gts}_s,.)$ and $\text{LTS}(\text{gts}_t,.)$ derived from $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ and $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$ as given in Def. 38, then the* bisimulation constraint $\mathcal{C}^{\text{Cor}}_{\text{Bis}} = \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$ *with $\mathcal{C}_{\text{RT}}$ a runtime constraint as given in Def. 33, $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ the pair constraint with correspondences as given in Def. 39 derived from $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ and $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$ and corresponding pair rules with correspondences as given in Def. 37 and $\mathcal{C}^{\text{Cor}}_{\text{MT}}$ a model transformation constraint as given in Def. 40.*

Example 23 (Equivalence)**.** *Similar to Example 15, this example describes the problem of verification of behavior preservation for model transformations according to Definition 27, but for the more complex case of more fine-grained notions of relabelings and rule equivalence. In particular, the correspondence extensions with respect to rule equivalence and parallel rule application allow for the handling of non-deterministic application of semantics rules. Tables 3, 5, 4 and 7 list the specific elements and their depictions corresponding to the concepts shown in Table 2.*

In summary, the TGG tgg *with TGG constraint $\mathcal{C}_{\text{tgg}}$ describes a model transformation $\text{MT}(\text{tgg},\mathcal{C}_{\text{tgg}}) : \mathcal{L}(S_{TT},\mathcal{C}_{\mathcal{S}}) \times \mathcal{L}(T_{TT},\mathcal{C}_{\mathcal{T}})$ between sequence charts with multiple lifelines (source modeling language $\mathcal{L}(S_{TT},\mathcal{C}_{\mathcal{S}})$) and systems of communicating automata (target modeling language $\mathcal{L}(T_{TT},\mathcal{C}_{\mathcal{T}})$). In contrast to Example 15, this case includes multiple lifelines and automata.*

Both source and target language are equipped with semantic definitions in the form of runtime conform graph transformation systems $\text{gts}_s = (\{initE, send\}, S_{RT})$ and $\text{gts}_t = (\{initS, fire\}, T_{RT})$ describing the possible behavior of sequence charts and communicating automata, respectively. For source and target models S and T, the semantic mappings $sem_{\mathcal{S}}(S) = \text{LTS}(\text{gts}_s, S)$ and $sem_{\mathcal{T}}(T) = \text{LTS}(\text{gts}_s, T)$ assign labeled transition systems induced by the graph transformation systems gts_s and gts_t, respectively. In addition, relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ and $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ map transitions of the LTS *to tuples consisting of elements of common label and match alphabets A and $\mathcal{M}$.*

In contrast to Example 15, the relabelings take matches into account, while still being based on bijective functions $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_s \to A$ renaming equivalent rules in source and target GTS to the same name. In addition, the combination of thusly equivalently renamed rules from gts_s and gts_t leads to so-called pair rules with correspondences, which, in contrast to the notion of plain pair rules in Example 15, require the existence of correspondences between source and target elements. As with other extensions of the previous concept and example, this represents the correct application of semantics rules relating source and target matches by respecting correspondences between source and target elements (see Example 17).

$\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is then behavior preserving in an equivalent manner, if for each pair of source and target models $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ it holds that $l_s(\mathrm{LTS}(\mathrm{gts}_s, S)) =_{bsim} l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$. In particular, we specify the bisimulation relation as an induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}}, .)$ with the bisimulation constraint $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ being the conjunction of the runtime constraint, pair constraint, source and target dynamic constraints, TGG constraint and model transformation constraint. In order to correctly describe behavioral equivalence by bisimulation for the case of the more fine-grained notion of rule equivalence, the derived constraints $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ and $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}}$ also include correspondence information between the respective source and target elements.*

More informally, as in Example 15, we require that for each pair of a sequence chart and a system of communicating automata related by the model transformation, their behavior is equivalent in the sense that each rule application on the source model (or target model, respectively) can be followed by an equivalent and corresponding rule application on the target model (or source model, respectively) such that after application of the respective pair rule with correspondences the resulting model states are equivalent again.

Table 7: Modeling step M_{pres} of Example 23

concept	behavioral equivalence				
formalization	relabelings l_s, l_t (Fig. 19, 20)		induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}}, .)$ with bisimulation constraint $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$		
artefact	mappings $l_s^{\mathcal{R}}$, $l_t^{\mathcal{R}}$	correspondences	runtime constraint $\mathcal{C}_{\mathrm{RT}}$	pair constraint $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}}$	model transformation constraint $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$
depicted in		Fig. 18	Fig. 12, 44	Fig. 45	Fig. 46

4.2. Verification Scheme

In the following theorem, we present a verification scheme for showing that the desired bisimulation relation as given in the previous section indeed defines a bisimulation between source and target semantics of each source and target model of the model transformation and we proof its correctness. In particular, it consists of three steps: (V_{init}) one simple constraint satisfaction check on the axiom of the triple graph grammar defining the model transformation, (V_{trans}) one invariant check on the triple graph grammar rules and (V_{sem}) one invariant check on the pair rules with correspondences of equivalent rules in source and target GTSs.

Theorem 2 (equivalence verification)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_\mathcal{S}), \mathcal{L}(T_{TT}, \mathcal{C}_\mathcal{T}), \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}}), \mathrm{LTS}(\mathrm{gts}_s, .), \mathrm{LTS}(\mathrm{gts}_t, .))$ *as introduced in Def. 26. Moreover, given relabelings* $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *derived from* $l_s^\mathcal{R} : \mathcal{R}_s \to A$ *and* $l_t^\mathcal{R} : \mathcal{R}_t \to A$ *as given in Def. 38, a bisimulation constraint* $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}} = \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ *typed over* $S_{RT}C_{TT}T_{RT}$ *as given in Def. 41, then* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is* behavior preserving *in an equivalent manner (in particular, via the induced bisimulation relation* $\mathcal{R}(\mathcal{C}_{\mathrm{Bis}}^{\mathrm{Cor}}, .)$*) in the sense of case 3.1 of Definition 27 if the following conditions are fulfilled:*

V_{init}: $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$.

V_{trans}: $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}})$.

V_{sem}: $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts})$ *with* $\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}$ *as given in Def. 37.*

Proof. Given some $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$, then we know that there exists some SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ such that it suffices to show that $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ and $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ are bisimilar via the induced relation $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}, SCT)$. We therefore prove (1) that the pair of initial states (S, T) of $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ and $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ is always in $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}, SCT)$ and (2) that $\mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}, SCT)$ is indeed a bisimulation relation according to conditions 1 and 2 of Def. 24.

(1) $(S, T) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}, SCT)$*:* Each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills $\mathcal{C}_{\mathrm{tgg}}$ by construction. We further prove by induction over the number of TGG rule applications that each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills also $\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ such that according to Def. 29 $(S, T) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}, SCT)$. The *base clause* for the axiom $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor}}$ follows directly from condition (V_{init}) of the Theorem.

Condition (V_{trans}) of the Theorem then provides the *induction step* that for any TGG rule application $S_nC_nT_n \Rightarrow_{\mathcal{R}} S_{n+1}C_{n+1}T_{n+1}$ it holds that $S_{n+1}C_{n+1}T_{n+1} \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$ assuming the *induction hypothesis* that $S_nC_nT_n \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$.

(2) $\mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$ is a bisimulation relation: We first have to show for condition 1 of Def. 24 that for all $(S_1, T_1) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$ (equivalent to $S_1CT_1 \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$ according to Def. 29), if $S_1 \xrightarrow{\alpha} S_2$, then $T_1 \xrightarrow{\alpha} T_2$ and $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$ for $l_s(\text{LTS}(\text{gts}_s, S))$ and $l_t(\text{LTS}(\text{gts}_t, T))$, respectively. This holds if $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$ implies $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ with $l_s(\rho_s, m_s) = l_t(\rho_t, m_t)$ and $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Rule}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$. If we have $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$, then we also have $S_1CT_1 \Rightarrow_{(\rho_s, m_s)} S_2CT_1$. Because $S_1CT_1 \vDash \mathcal{C}^{\text{Cor}}_{\text{Pair}}$, and in particular $S_1CT_1 \vDash \mathcal{C}^{\text{Cor,f}}_{\text{Pair}}$, applicability of ρ_s to S_1CT_1 via m_s implies applicability of $\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t} \rho_t$ with (ρ_s, ρ_t) belonging to $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ to S_1CT_1 via an extended match m. It then follows that $S_1CT_1 \Rightarrow_{(\rho_s *_{R_sC_{(\rho_s,\rho_t)}L_t}\rho_t, m)} S_2CT_2$ such that because of the Concurrency Theorem [20] $S_1CT_1 \Rightarrow_{(\rho_s, m_s)} S_2CT_1 \Rightarrow_{(\rho_t, m_t)} S_2CT_2$. In particular, we then have that $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ such that $l_s(\rho_s, m_s) = l_t(\rho_t, m_t)$.We still need to prove that $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$. As gts_s and gts_t are runtime conform, they preserve static types, and therefore $\mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$ typed over $S_{TT}C_{TT}T_{TT}$ is by construction an inductive invariant for $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ implying $S_2CT_2 \vDash \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$. Moreover, $S_2CT_2 \vDash \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t}$ by construction as well, since S_2CT_2 already satisfies $\mathcal{C}_S$ and $\mathcal{C}_T$ (comprised in $\mathcal{C}_{\text{tgg}}$) and again because of runtime conformity of gts_s and gts_t the GTSs with constraint $\text{gts}_s^{\mathcal{C}_S}$ and $\text{gts}_t^{\mathcal{C}_S}$ have the dynamic constraints $\mathcal{C}^{gts}_{s}$ and $\mathcal{C}^{gts}_{t}$ as inductive invariants, respectively. Because of condition (V_{sem}) of the Theorem and the fact that $S_1CT_1 \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$, then it follows that $S_2CT_2 \vDash \mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}$. Thus, according to Def. 29, this means that $(S_2, T_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}} \wedge \mathcal{C}^{\text{Cor}}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\text{Cor}}_{\text{MT}}, SCT)$. Condition 2 of Def. 24 follows analogously to condition 1 because of Remark 8 as the roles of S and T are symmetric.

□

$$
\begin{array}{ll}
S_A C_A T_A & S_A C_A T_A \\
 & \models \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Cor}}_{\mathrm{MT}} \ (V_{init}) \\
\Big\Vert\, \mathcal{R} \,\Big\Vert\, * & \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\mathrm{Cor}}_{\mathrm{MT}} \\
 & \text{is an inductive invariant of} \\
\downarrow & (\mathcal{R}, S_{RT} C_{TT} T_{RT}, \mathcal{C}_{\mathrm{tgg}}) \ (V_{trans}) \\
SCT & \\
\Big\Vert\, \mathcal{P}(l_s, l_t)^{\mathrm{Cor}} \,\Big\Vert\, * & \mathcal{C}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}} \text{ is an inductive invariant} \\
 & \text{of } (\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}, S_{RT} C_{TT} T_{RT}, \cdots) \ (V_{sem}) \\
\downarrow & \\
S_* C T_* &
\end{array}
$$

The above sketch summarizes how the constraints are preserved via the conditions starting from the TGG axiom, via the TGG rules, and finally also during the execution of pair rules of the semantics. While condition V_{init} and V_{trans} are employed in step (1) of the proof via an induction, condition V_{sem} allows to show that the induced relation is indeed a bisimulation in step (2) of the proof.

Example 24 (Equivalence: Verification)**.** *The model transformation described in Example 23 can be shown to be behavior preserving via the bisimulation relation induced by the bisimulation constraint* $\mathcal{C}_{\mathrm{Bis}}$.

Since the correspondence of runtime elements as described in $\mathcal{C}_{\mathrm{RT}}$ *and the applicability of equivalent rules as expressed in* $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}}$ *hold for the axiom of the triple graph grammar (*V_{init}*) and are invariant for each possible application of a triple graph rule (*V_{trans}*) from the example TGG* tgg *(with TGG constraint* $\mathcal{C}_{\mathrm{tgg}}$*), each possible triple graph generated by the triple graph grammar (describing a pair of a sequence chart and a system of communication automata related by a model transformation instance) also satisfies these constraints.*

In contrast to Example 15, due to multiple possibilities to apply two equivalently named rules on the source and target model in combination, the pair rules with correspondences $\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}$ *and the derived constraints* $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}}$ *and* $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{MT}}$ *(which are part of the bisimulation constraint) include correspondences. For the case of pair rules without correspondences, as was shown in Example 17,* $\mathcal{C}_{\mathrm{RT}}$ *and* $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}}$ *would not be inductive invariants.*

Since the correspondence of runtime elements ($\mathcal{C}_{\mathrm{RT}}$*) and the applicability of equivalent and corresponding rules (*$\mathcal{C}^{\mathrm{Cor}}_{\mathrm{Pair}}$*) remains invariant for each pairwise application of equivalent and corresponding source and target semantics rules (*V_{sem}*), all model transformation instances and thus, the model transformation as such, are behavior preserving. More informally, each rule application on the source model (or target model, respectively) can be followed by an equivalent and corresponding rule application on the target model (or source model, respectively).*

5. Behavioral Refinement Verification

The results presented for equivalence in the former section can also be adjusted for the case we only want to ensure behavioral refinement (resp. behavioral abstraction) instead of behavioral equivalence.

5.1. Modeling Step M_{pres}

Instead of requiring a behavioral equivalence, for specific model transformations, it will be sufficient to require a behavioral refinement between the semantics of each target and source model of a model transformation. As refinement in contrast to equivalence is not symmetric, we further distinguish refinement for the forward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ and backward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1}$.

For showing a behavioral refinement for the forward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ we search for a simulation relation over $l_t(\mathrm{LTS}(\mathrm{gts}_t, .)) \times l_s(\mathrm{LTS}(\mathrm{gts}_s, .))$ between the semantics of each target T and source model S of the forward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$. We will derive it as the inverse relation of the induced relation (see Def. 29) from the *simulation constraint* $\mathcal{C}^{\mathrm{Cor,b}}_{\mathrm{Sim}} = \mathcal{C}^{\mathrm{b}}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Cor,b}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Cor,b}}_{\mathrm{MT}}$ (with $\mathcal{C}^{\mathrm{Cor,b}}_{\mathrm{MT}}$ chosen as the backward direction of the model transformation constraint $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{MT}}$ as introduced before), where all single components of this constraint have been introduced in the previous chapters. In particular, we will show in Theorem 3 that $l_t(\mathrm{LTS}(\mathrm{gts}_t, .))$ is simulated via this induced relation by $l_s(\mathrm{LTS}(\mathrm{gts}_s, .))$ (or $l_t(LTS(gts_t, .)) \leq_{sim} l_s(LTS(gts_s, .))$) for each target T and source model S of the forward transformation. More informally, all behavior in T can be found in S as well, but not necessarily vice versa, so that T is a refinement (in terms of behavior) of S.

If we want to demonstrate a behavioral refinement for the backward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1}$ we search for a simulation relation over $l_s(\mathrm{LTS}(\mathrm{gts}_s, .)) \times l_t(\mathrm{LTS}(\mathrm{gts}_t, .))$ between the semantics of each source model S and target model T of the backward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1}$. We will derive it as induced relation (see Def. 29) from the *simulation constraint* $\mathcal{C}^{\mathrm{Cor,f}}_{\mathrm{Sim}} = \mathcal{C}^{\mathrm{f}}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{Cor,f}}_{\mathrm{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}^{\mathrm{Cor,f}}_{\mathrm{MT}}$ (with $\mathcal{C}^{\mathrm{Cor,f}}_{\mathrm{MT}}$ being the forward part of $\mathcal{C}^{\mathrm{Cor}}_{\mathrm{MT}}$) , where all single components of this constraint have been introduced in the previous chapters. In particular, we will show in Corollary 1 that $l_s(\mathrm{LTS}(\mathrm{gts}_s, .))$ is simulated via this induced relation by $l_t(\mathrm{LTS}(\mathrm{gts}_t, .))$ (or $l_s(LTS(gts_s, .)) \leq_{sim} l_t(LTS(gts_t, .))$) for each source S and target model T of the backward transformation. More informally, all behavior in S can be found in T as well, but not necessarily vice versa, so that S is a refinement (in terms of behavior) of T.

Definition 42 (Simulation constraint $\mathcal{C}_{\text{Sim}}^{\text{Cor,b}}$ and $\mathcal{C}_{\text{Sim}}^{\text{Cor,f}}$)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}), \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}), \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}), \text{LTS}(\text{gts}_s, .), \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26. Moreover, given relabelings* $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *for* $\text{LTS}(\text{gts}_s, .)$ *and* $\text{LTS}(\text{gts}_t, .)$ *derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as given in Def. 38, the simulation constraint* $\mathcal{C}_{\text{Sim}}^{\text{Cor,b}} = \mathcal{C}_{\text{RT}}^{\text{b}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$ *with* $\mathcal{C}_{\text{RT}}^{\text{b}}$ *a backward runtime constraint as given in Def. 33,* $\mathcal{C}_{\text{Pair}}^{\text{Cor,b}}$ *the backward pair constraint with correspondences as given in Def. 39 derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *and corresponding pair rules with correspondences as given in Def. 37 and a backward model transformation constraint* $\mathcal{C}_{\text{MT}}^{\text{Cor,b}}$ *as given in Def. 40.*

Example 25 (Refinement)**.** *This example describes the problem of verification of behavior preservation in a refining manner according to case 3.2 (behavioral refinement) of Definition 27. In particular, we consider a model transformation induced by the TGG* tgg *with TGG constraint* $\mathcal{C}_{\text{tgg}}$ *describing a model transformation* $\text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}) : \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}) \times \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ *between sequence charts with multiple lifelines (modeling language* $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$*) and communicating automata with additional internal behavior (modeling language* $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$*) as described in Tables 3, 4, 5 and 8. In particular, we will show that the backward transformation derived from this TGG represents a refinement such that all the behavior of the sequence chart model is reflected by the communicating automaton, but not the other way round.*

Both source and target language are equipped with a runtime graph language $\mathcal{L}(S_{RT}, \mathcal{C}_S \wedge \mathcal{C}_s^{gts})$ *and* $\mathcal{L}(T_{RT}, \mathcal{C}_T \wedge \mathcal{C}_t^{gts})$*, respectively, as well as runtime conform graph transformation systems* $\text{gts}_s = (\{initE, send\}, S_{RT})$ *and* $\text{gts}_t = (\{initS, fire\}, T_{RT})$*, describing the possible behavior of sequence charts and communicating automata, respectively. For source and target models S and T, we have induced labeled transition systems* $\text{LTS}(\text{gts}_s, S)$ *and* $\text{LTS}(\text{gts}_s, T)$*, respectively. In addition, relabelings* $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *map transitions of the* LTS *to tuples consisting of elements of common label and match alphabets A and* $\mathcal{M}$*.*

As before, the relabeling functions take matches into account, while still being based on bijective mappings $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_s \to A$ *renaming equivalent rules in source and target GTS to the same name. In addition, the combination of thusly equivalently renamed rules from* gts_s *and* gts_t *again leads to so-called pair rules with correspondences. Similar to Example 23, correspondences are required to represent the correct application of source and target GTS rules relating source and target matches via correspondences between source and target elements.*

The backward model transformation $\text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}})^{-1}$ *is then behavior preserving in a refining manner, if for each pair of models* $(T, S) \in \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}})^{-1}$ *it holds that* $l_s(\text{LTS}(\text{gts}_s, S)) \leq_{sim} l_t(\text{LTS}(\text{gts}_t, T))$*. In particular, we specify the simulation relation as an induced relation of the simulation constraint* $\mathcal{C}_{\text{Sim}}^{\text{Cor,f}} = \mathcal{C}_{\text{RT}}^{\text{f}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,f}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge$

$\mathcal{C}_{\text{MT}}^{\text{Cor,f}}$ being the conjunction of the forward part of the runtime constraint, pair constraint, source and target dynamic constraints, TGG constraint and forward model transformation constraint ($\mathcal{C}_{\text{MT}}^{\text{Cor,f}}$).

More informally, we require that for each pair of a sequence chart and a system of communicating automata related by the model transformation, the behavior of the sequence chart refines the behavior of the related communicating automaton in the sense that each rule application on the sequence chart can be followed by an equivalent and corresponding rule application on the automaton. In contrast to behavioral equivalence as shown in Example 23, we do not require both directions. Thus, our simulation relation only needs to consider the forward part of the respective constraints.

Table 8: Modeling step M_{pres} of Example 25

concept	behavioral refinement				
formalization	relabelings l_s, l_t (Fig. 19, 20)		induced relation $\mathcal{R}(\mathcal{C}_{\text{Sim}}^{\text{Cor,f}},.)$ with bisimulation constraint $\mathcal{C}_{\text{Sim}}^{\text{Cor,f}} = \mathcal{C}_{\text{RT}}^{\text{f}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,f}} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,f}}$		
artefact	mappings $l_s^{\mathcal{R}}$, $l_t^{\mathcal{R}}$	correspondences	runtime constraint $\mathcal{C}_{\text{RT}}^{\text{f}}$	pair constraint $\mathcal{C}_{\text{Pair}}^{\text{Cor,f}}$	forward MT constraint $\mathcal{C}_{\text{MT}}^{\text{Cor,f}}$
depicted in		Fig. 18	Fig. 57	Fig. 58	Fig. 59

5.2. Verification Scheme

In the following theorem, we present a verification scheme for showing that the desired simulation relation as given in the previous section indeed leads to the fact that the semantics of each target model is simulated by the semantics of each source model of the model transformation and we proof the correctness of the verification scheme. In particular, it consists of three steps: (V_{init}) one simple constraint satisfaction check on the axiom of the triple graph grammar defining the model transformation, (V_{trans}) one invariant check on the triple graph grammar rules and (V_{sem}) one invariant check on the pair rules with correspondences of equivalent rules in source and target GTSs.

Theorem 3 (refinement verification for forward transformation)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_S)$, $\mathcal{L}(T_{TT}, \mathcal{C}_T)$, $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$, $\mathrm{LTS}(\mathrm{gts}_s, .)$, $\mathrm{LTS}(\mathrm{gts}_t, .))$ *as introduced in Def. 26. Moreover, given relabelings* $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as given in Def. 38, and a simulation constraint* $\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}} = \mathcal{C}_{\mathrm{RT}}^{\mathrm{b}} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ *typed over* $S_{RT}C_{TT}T_{RT}$ *as given in Def. 42, then* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *is* behavior preserving *in a refining manner (in particular, via the induced simulation relation* $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, .)^{-1}$*) in the sense of case 3.2 of Definition 27 if the following conditions are fulfilled:*

V_{init}: $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$.

V_{trans}: $\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}})$.

V_{sem}: $\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts})$ *with* $\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}$ *as given in Def. 37.*

Proof. We have to show that for any $(S, T) \in \mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$, it holds that $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ is simulated by $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ via the relation $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, SCT)^{-1}$. We therefore prove (1) that the pair of initial states (T, S) of $l_t(\mathrm{LTS}(\mathrm{gts}_t, T))$ and $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$ is always in $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, SCT)^{-1}$ and (2) that $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, SCT)^{-1}$ is indeed a simulation relation according to Def. 25.

(1) $(T, S) \in \mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, SCT)^{-1}$*:* Each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills $\mathcal{C}_{\mathrm{tgg}}$ by construction. We further prove by induction over the number of TGG rule applications that each triple graph SCT in $\mathcal{L}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ fulfills also $\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ such that according to Def. 29 $(T, S) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}, SCT)^{-1}$. The *base clause* for the axiom $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ follows directly from condition (V_{init}) of the Theorem. Condition (V_{trans}) of the Theorem then provides the *induction step* that for any TGG rule application $S_n C_n T_n \Rightarrow_{\mathcal{R}} S_{n+1} C_{n+1} T_{n+1}$ it holds that $S_{n+1} C_{n+1} T_{n+1} \vDash \mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ assuming the *induction hypothesis* that $S_n C_n T_n \vDash \mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$.

(2) $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,b}}, SCT)^{-1}$ *is a simulation relation:* We have to show according to Def. 25 that for all $(T_1, S_1) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}, SCT)^{-1}$ (equivalent to $S_1 C T_1 \vDash \mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ according to Def. 29), if $T_1 \overset{\alpha}{\to} T_2$, then $S_1 \overset{\alpha}{\to} S_2$ and $(T_2, S_2) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}, SCT)^{-1}$ for $l_t(\mathrm{LTS}(\mathrm{gts}_t, T)$ and $l_s(\mathrm{LTS}(\mathrm{gts}_s, S))$, respectively. This holds if $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$ implies $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$ with $l_t(\rho_t, m_t) = l_s(\rho_s, m_s)$ and $(T_2, S_2) \in \mathcal{R}(\mathcal{C}_{\mathrm{RT}}^{b} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,b}} \wedge$

$\mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,f}}, SCT)^{-1}$. If we have $T_1 \Rightarrow_{(\rho_t, m_t)} T_2$, then we also have $S_1CT_1 \Rightarrow_{(\rho_t, m_t)} S_1CT_2$. Because $S_1CT_1 \vDash \mathcal{C}_{\text{Pair}}^{\text{Cor,b}}$, applicability of ρ_t to S_1CT_1 via m_t implies applicability of $\rho_t *_{R_tC_{(\rho_s,\rho_t)}L_s} \rho_s$ with (ρ_s, ρ_t) belonging to $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ to S_1CT_1 via an extended match m. It then follows that $S_1CT_1 \Rightarrow_{(\rho_t *_{R_tC_{(\rho_s,\rho_t)}L_s}\rho_s, m)} S_2CT_2$ such that because of the Concurrency Theorem [20] $S_1CT_1 \Rightarrow_{(\rho_t, m_t)} S_1CT_2 \Rightarrow_{(\rho_s, m_s)} S_2CT_2$. In particular, we then have that $S_1 \Rightarrow_{(\rho_s, m_s)} S_2$ such that $l_s(\rho_s, m_s) = l_t(\rho_t, m_t)$. We still need to prove that $(T_2, S_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}}^{b} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,f}}, SCT)^{-1}$. As gts_s and gts_t are runtime conform they preserve static types. Since $\mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$ are typed over $S_{TT}C_{TT}T_{TT}$, it is by construction an inductive invariant for $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ implying $S_2CT_2 \vDash \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$. Moreover, $S_2CT_2 \vDash \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts}$ by construction as well, since S_2CT_2 already satisfies $\mathcal{C}_S$ and $\mathcal{C}_T$ (comprised in $\mathcal{C}_{\text{tgg}}$) and again because of runtime conformity of gts_s and gts_t the GTSs with constraint $\text{gts}_s^{\mathcal{C}_S}$ and $\text{gts}_t^{\mathcal{C}_S}$ have the dynamic constraints $\mathcal{C}_s^{gts}$ and $\mathcal{C}_t^{gts}$ as inductive invariants, respectively. Because of condition (V_{sem}) of the Theorem and the fact that $S_1CT_1 \vDash \mathcal{C}_{\text{RT}}^{b} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$, then it follows that $S_2CT_2 \vDash \mathcal{C}_{\text{RT}}^{b} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$. Thus, according to Def. 29, this means that $(T_2, S_2) \in \mathcal{R}(\mathcal{C}_{\text{RT}}^{b} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,f}}, SCT)^{-1}$.

□

$$
\begin{array}{ll}
S_AC_AT_A & S_AC_AT_A \\
 & \vDash \mathcal{C}_{\text{RT}}^{\text{b}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}} \ (V_{init}) \\
\Big\Vert\, {\scriptstyle \mathcal{R}}\,* & \mathcal{C}_{\text{RT}}^{\text{b}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}} \\
 & \text{is an inductive invariant of} \\
\downarrow & (\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\text{tgg}}) \ (V_{trans}) \\
SCT & \\
\Big\Vert\, {\scriptstyle \mathcal{P}(l_s,l_t)^{\text{Cor}}}\,* & \mathcal{C}_{\text{RT}}^{\text{b}} \wedge \mathcal{C}_{\text{Pair}}^{\text{Cor,b}} \text{ is an inductive invariant} \\
\downarrow & \text{of } (\mathcal{P}(l_s, l_t)^{\text{Cor}}, S_{RT}C_{TT}T_{RT}, \cdots) \ (V_{sem}) \\
S_*CT_* &
\end{array}
$$

The above sketch summarizes how the constraints are preserved via the conditions starting from the TGG axiom, via the TGG rules, and finally also during the execution of pair rules of the semantics. While condition V_{init} and V_{trans} are employed in step (1) of the proof via an induction, condition V_{sem} allows to show that the induced relation is indeed a simulation in step (2) of the proof.

Corollary 1 (refinement verification for backward transformation)**.** *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}}),\ \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}),\ \text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}}),\ \text{LTS}(\text{gts}_s, .),\ \text{LTS}(\text{gts}_t, .))$ *as introduced in Def. 26.*

Moreover, given relabelings $l_s : \mathcal{R}_s \times \mathcal{M}_s \to A \times \mathcal{M}$ *and* $l_t : \mathcal{R}_t \times \mathcal{M}_t \to A \times \mathcal{M}$ *for* $\mathrm{LTS}(\mathrm{gts}_s, .)$ *and* $\mathrm{LTS}(\mathrm{gts}_t, .)$ *derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as given in Def. 38, and a simulation constraint* $\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,f}} = \mathcal{C}_{\mathrm{RT}}^{f} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}$ *typed over* $S_{RT}C_{TT}T_{RT}$ *as given in Def. 42, then the* backward *model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1} : \mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}}) \times \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ *is* behavior preserving *in a refining manner (in particular, via the relation* $\mathcal{R}(\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,f}}, .)$*) in the sense of case 3.2 of Definition 27 if the following conditions are fulfilled:*

V_{init}: $S_A C_A T_A \vDash \mathcal{C}_{\mathrm{RT}}^{f} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}$.

V_{trans}: $\mathcal{C}_{\mathrm{RT}}^{f} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}})$.

V_{sem}: $\mathcal{C}_{\mathrm{RT}}^{f} \wedge \mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}}$ *is an inductive invariant (see Def. 15) of* $(\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}), S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\mathrm{tgg}} \wedge \mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,f}} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts})$ *with* $\mathcal{P}(l_s, l_t)^{\mathrm{Cor}}$ *as given in Def. 37.*

Proof. The claimed result follows from Theorem 3 and the symmetry of the TGG and the constructed constraints. □

Remark 9 (Abstraction)**.** *Since abstraction is the opposite of refinement, for verifying that a model transformation represents an abstraction, we can just check that the inverse model transformation is behavior preserving in a refining manner. In particular, this means that if a forward (backward) transformation derived from some TGG is behavior preserving in a refining manner, then the corresponding backward (forward) transformation represents an abstraction.*

Example 26 (Refinement verification for backward transformation)**.** *The model transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1}$ *described in Example 25 can be shown to be behavior preserving via the simulation relation induced by the simulation constraint* $\mathcal{C}_{\mathrm{Sim}}^{\mathrm{Cor,f}}$ *as described in Corollary 1.*

Since the correspondence of runtime elements as described in $\mathcal{C}_{\mathrm{RT}}^{\mathrm{f}}$ *and the applicability of equivalent rules as expressed in* $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}}$ *hold for the axiom of the triple graph grammar (*V_{init}*) and are invariant for each possible application of a triple graph rule (*V_{trans}*) from the example TGG* tgg *(with TGG constraint* $\mathcal{C}_{\mathrm{tgg}}$*), each possible triple graph generated by the triple graph grammar (describing a pair of a sequence chart and a system of communication automata related by a model transformation instance) also satisfies these constraints.*

Since the correspondence of runtime elements ($\mathcal{C}_{\mathrm{RT}}^{\mathrm{f}}$*) and the applicability of equivalent and corresponding rules (*$\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}}$*) remains invariant for each pairwise application of equivalent and corresponding source and target semantics rules (*V_{sem}*), all model transformation instances and thus, the model transformation as such, are behavior preserving in a refining manner. In contrast to Example 23, the respective constraints* $\mathcal{C}_{\mathrm{RT}}^{\mathrm{f}}$ *and* $\mathcal{C}_{\mathrm{Pair}}^{\mathrm{Cor,f}}$ *only describe*

implications in a direction from the source model to the target model. More informally, each rule application on the source model can be followed by an equivalent and corresponding rule application on the target model, but not vice versa.

In fact, the verification by the above theorem will fail for the forward transformation $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$*; in particular,* $\mathcal{C}_{\mathrm{MT}}^{\mathrm{Cor,b}}$ *is not an inductive invariant of* $(\mathcal{R}, S_{TT}C_{TT}T_{TT}, \mathcal{C}_{\mathrm{tgg}})$*. Because of two rules in* $\mathcal{R}$ *(Figures 49, 50),* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *relates source and target models where the target model contains the static components of additional behavior in the form of states and communicating transitions without matching components for behavior in the source model. Hence, there will be model transformation instances* (S, T)*, where a semantics rule application in the target model cannot be followed by a corresponding rule application in the source model.*

Finally note that since $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})^{-1}$ *is a model transformation that is behavior preserving in a refining manner, the corresponding forward transformation* $\mathrm{MT}(\mathrm{tgg}, \mathcal{C}_{\mathrm{tgg}})$ *from sequence charts to automata with additional internal behavior represents an abstraction.*

6. Automation

This section reiterates over the modeling steps M_{lang}, M_{trans}, M_{sem}, and M_{pres} of our modeling scheme and steps V_{init}, V_{trans}, and V_{sem} of our verification scheme for behavior preservation of model transformations and discusses if and how these steps can be executed automatically and which types of automated tools apply. Since all three variants of behavior preservation as discussed in Sections 3–5 share a significant number of common elements, automation is discussed generically for these steps. The steps of the generic modeling scheme are summarized in Fig. 1 and 10 and the generic verification scheme is summarized in Fig. 21. In the latter figure only the finite artefacts of the modeled concepts are depicted that can be automatically processed in the three steps V_{init}, V_{trans} and V_{sem} (as in Theorem 1, 2, and 3) of the verification scheme.

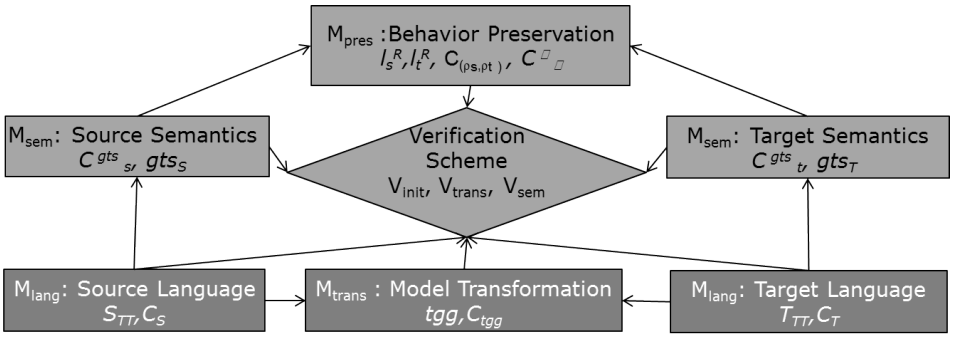


Figure 21: Verification Scheme V_{init}, V_{trans}, and V_{sem} for Behavior Preservation on the Transformation Level

6.1. Automation of Modeling Scheme

In all cases, our modeling scheme consists of the steps following below (as illustrated earlier in Fig. 1 and 10 and Table 2).

M_{lang} (Modeling Language): Source modeling language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ based on type graph S_{TT} and constraint $\mathcal{C}_S$ and target modeling language $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ based on type graph T_{TT} and constraint $\mathcal{C}_T$.

M_{sem} (Model Semantics): $\text{LTS}(\text{gts}_s, \cdot)$ based on runtime conform $\text{gts}_s = (\mathcal{R}_s, S_{RT})$ and dynamic constraint $\mathcal{C}_s^{gts}$ and $\text{LTS}(\text{gts}_t, \cdot)$ based on runtime conform $\text{gts}_t = (\mathcal{R}_t, T_{RT})$ and dynamic constraint $\mathcal{C}_t^{gts}$.

M_{trans} (Model Transformation): Induced model transformation $\text{MT}(\text{tgg}, \mathcal{C}_{\text{tgg}})$ based on TGG constraint $\mathcal{C}_{\text{tgg}}$ (which comprises $\mathcal{C}_S$ and $\mathcal{C}_T$) and $\text{tgg} = (\mathcal{R}, S_{RT}C_{TT}T_{RT}, S_AC_AT_A)$ with TGG rules $\mathcal{R}$, type graph $S_{RT}C_{TT}T_{RT}$, and axiom $S_AC_AT_A$.

M_{pres} (Behavior Preservation): Relabelings l_s, l_r based on mappings $l_s^{\mathcal{R}}$ and $l_t^{\mathcal{R}}$ and correspondences $C_{(\rho_s,\rho_t)}$ and induced (bi-)simulation relation $\mathcal{R}(\mathcal{C}_{\square}^{\square})$ based on (bi-)simulation constraint $\mathcal{C}_{\square}^{\square} = \mathcal{C}_{\text{RT}}^{\square} \wedge \mathcal{C}_{\text{Pair}}^{\square} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}_{\text{MT}}^{\square}$.

The required formalization and artefacts for the first three modeling steps can be found in Tables 3 (M_{lang}), 4 (M_{sem}), and 5 (M_{trans}). All required artefacts need to be specified manually. However, there are certain steps, that, depending on the specific example at hand, might have potential for automation.

First and foremost, our approach relies heavily on type graphs, typed graphs, and graph languages and GTSs defined by type graphs and graph constraints. Modeling tools that support these features will provide great help in the respective modeling steps involved in our approach. In particular, the use of modeling tools that can automatically check the adherence of a modeled artefact (graph rule, graph constraints, etc.) to such a modeling language or allow only the specification of correctly typed artefacts is advantageous. This applies to all steps of the modeling scheme.

Second, the runtime conformity of gts_s and gts_s (M_{sem}) with respect to their runtime graph language can be verified by tools capable to perform the required checks. Automated type checks of gts_s and gts_t are able to deduce whether their rules indeed preserve all static elements (i.e., nodes of types in S_{TT} and T_{TT}). Further, the dynamic constraints must be verified as inductive invariants of $\text{gts}_s^{\mathcal{C}_S}$ and $\text{gts}_t^{\mathcal{C}_T}$. This capability of performing inductive invariant checks will also be required for the automation of our verification scheme.

Third, in step (M_{trans}), we assume for the TGG constraint $\mathcal{C}_{\text{tgg}}$ to comprise both $\mathcal{C}_{\mathcal{S}}$ and $C_{\mathcal{T}}$. We can therefore assume that $\mathcal{C}_{\text{tgg}}$ is automatically extended with $\mathcal{C}_{\mathcal{S}}$ and $C_{\mathcal{T}}$ to fulfill this condition.

The generic view for the formalization and artefacts of step (M_{pres}) and their specific extensions for the three cases are shown in Tables 9 and 10, respectively. The relabelings l_s and l_t are based a) on mappings $l_s^{\mathcal{R}}$ and $l_t^{\mathcal{R}}$ of rules in the runtime GTSs to a common alphabet and b) on correspondences $C_{(\rho_s,\rho_t)}$ connecting source and target rules ρ_s, ρ_t assigned to the same element by the remappings (except for the simple equivalence case, where no correspondences are required). Both

remappings and correspondences are to be specified manually. In general, those correspondences could also be found automatically by trying to match the static part of the LHS of equivalently labeled source and target rules to the RHS of TGG rules. If the matching is successful, then the discovered correspondence relations prescribed by the TGG rules can be extracted as $C_{(\rho_s,\rho_t)}$.

For the specification of the bisimulation constraint $\mathcal{C}^{\square}_{\square}$, pair rules $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ are derived from the mappings $l_s^{\mathcal{R}}$ and $l_t^{\mathcal{R}}$. In the case of equivalence and refinement the pair constraint $\mathcal{C}^{\square}_{\text{Pair}}$ is derived from the mappings $l_s^{\mathcal{R}}$ and $l_t^{\mathcal{R}}$ and the correspondences between the equivalently labeled rules. For the model transformation constraint $\mathcal{C}^{\square}_{\text{MT}}$ there is a difference in the simple equivalence and equivalence (and refinement) approach: For the former, the constraint is based on the TGG rules as decribed in Section 3. For the latter cases, we propose to derive it from the static elements (i.e. in the type graph $S_{TT}C_{TT}T_{TT}$) of the pair constraint $\mathcal{C}^{\square}_{\text{Pair}}$.

For the equivalence and refinement approaches, we also suggest (in Section 7) an automatic derivation for the model transformation constraint from the pair constraint for cases where the semantics rules only have the trivial case true as application conditions and do not delete any nodes. In thoses cases, the pair constraint will always be a conjunction of nested constraints of the form $\forall(P, \exists N)$ without additional levels of nesting for which we can establish an automatic derivation fulfilling the requirements of Definition 40.

This approach can possibly be extended to cover more complex cases; in particular, we believe that there is an automatic derivation for cases with arbitrary nested application conditions in semantics rules, which would result in more complex nesting structures in the pair constraint. However, at this point, we cannot provide a general proof. Thus, automatic derivation of model transformation constraints from pair constraints as discussed in this report is limited to formally covered model transformations with the restrictions to semantics rules as described above and specified in Construction 1 of Section 7 below.

6.2. Automation of Verification Scheme

For all cases discussed in this report, the proposed verification scheme for behavior preservation consists of the three steps described by Theorems 1, 2, and 3, respectively. While between the different cases, different constraints and rules are involved, all such elements can be represented in a generic way for demonstration purposes as follows:

V_{init}: $S_A C_A T_A \vDash \mathcal{C}^{\square}_{\text{RT}} \wedge \mathcal{C}^{\square}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\square}_{\text{MT}}$.

V_{trans}: $\mathcal{C}^{\square}_{\text{RT}} \wedge \mathcal{C}^{\square}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}^{\square}_{\text{MT}}$ is an inductive invariant of $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\text{tgg}})$.

Table 9: Modeling step M_{pres} and its specification effort

concept	behavior preservation				
formalization	relabelings l_s, l_t		induced relation $\mathcal{R}(\mathcal{C}^{\square}_{\square}, .)$ with (bi-)simulation constraint $\mathcal{C}^{\square}_{\square} = \mathcal{C}^{\square}_{\text{RT}} \wedge \mathcal{C}^{\square}_{\text{Pair}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\square}_{\text{MT}}$		
artefact	mappings $l^{\mathcal{R}}_s$, $l^{\mathcal{R}}_t$	correspondences	runtime constraint $\mathcal{C}^{\square}_{\text{RT}}$	pair constraint $\mathcal{C}^{\square}_{\text{Pair}}$	MT constraint $\mathcal{C}^{\square}_{\text{MT}}$
specification	manual	semi-automatic	manual	automatic	man./autom., see text

Table 10: Modeling step M_{pres} and generic & specific artefacts

case	section	pair rules	runtime constraint	pair constraint	model transformation constraint
generic	Sect. 6	$\mathcal{P}(l_s, l_t)^{\square}$	$\mathcal{C}^{\square}_{\text{RT}}$	$\mathcal{C}^{\square}_{\text{Pair}}$	$\mathcal{C}^{\square}_{\text{MT}}$
simple equivalence	Sect. 3	$\mathcal{P}(l_s, l_t)$	$\mathcal{C}_{\text{RT}}$	$\mathcal{C}^{\text{Rule}}_{\text{Pair}}$	$\mathcal{C}^{\text{Man}}_{\text{MT}}$
equivalence	Sect. 4	$\mathcal{P}(l_s, l_t)^{\text{Cor}}$	$\mathcal{C}_{\text{RT}}$	$\mathcal{C}^{\text{Cor}}_{\text{Pair}}$	$\mathcal{C}^{\text{Cor}}_{\text{MT}}$
refinement, backward	Sect. 5	$\mathcal{P}(l_s, l_t)^{\text{Cor}}$	$\mathcal{C}^{\text{f}}_{\text{RT}}$	$\mathcal{C}^{\text{Cor,f}}_{\text{Pair}}$	$\mathcal{C}^{\text{Cor,f}}_{\text{MT}}$
refinement, forward	Sect. 5	$\mathcal{P}(l_s, l_t)^{\text{Cor}}$	$\mathcal{C}^{\text{b}}_{\text{RT}}$	$\mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$	$\mathcal{C}^{\text{Cor,b}}_{\text{MT}}$

V_{sem}: $\mathcal{C}^{\square}_{\text{RT}} \wedge \mathcal{C}^{\square}_{\text{Pair}}$ is an inductive invariant of $(\mathcal{P}(l_s, l_t)^{\square}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{\text{tgg}} \wedge \mathcal{C}^{\square}_{\text{MT}} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t})$.

For this generic form of the verification steps, Table 10 shows the respective specific steps. However, for the purpose of automatic verification, the general procedure remains the same for all cases.

Since the axiom $S_A C_A T_A$ is just a specifically typed graph, the first check (V_{init}) can be done by most graph transformation tools that support typed graphs and the checking of graph constraints for a particular typed graph. Checking the validity of a number of graph constraints is usually not computationally challenging, especially if the graph in question is rather small, as it is often the case with TGG axioms. Also, the expressiveness of the graph constraints does not need to be restricted, i.e. the nesting and combination of conditions can be arbitrarily deep and complex.

For steps V_{trans} and V_{sem} we can employ tools capable of verifying inductive invariants for graph transformation systems. For the general case of having nested conditions in rules or constraints, the related verification problem is, in general, undecidable [32]. While our formalization of steps V_{trans} and V_{sem} does not require restrictions on the rules or constraints, specific tools automating the corresponding invariant checks might impose specific restrictions on the structure of these rules and constraints. Also, depending on the nature, performance, and scalability of the tool, it may not be feasible to verify rules and constraints beyond a certain number or degree of complexity.

More specifically, as elaborated in [15], a number of verification tools are not applicable to our examples because they do not support negative application conditions to the required extent. This includes Uncover [39], Augur [38], RAVEN [7], and the model checking approaches by Steenken [48] and Boneva et al. [8]. While the approach of Habel and Pennemann [32, 45] is expressive enough, their corresponding tool does not yield results for our examples in reasonable time. While our own tool [2, 15] does, it requires special considerations and simplifications (see Section 7) to specifically cope with a fragment of the pair constraints in our examples.

7. Evaluation

After the repetition of the generic modeling and verification steps in the previous section and their potential for automation, this section now explains and evaluates the modeling and verification steps of our specific examples for the various cases. For the verification part, we employ our own invariant checking tool [2, 15].

7.1. Evaluation of Modeling Scheme

Table 11 lists both the results of the verification steps explained below and statistic information about the elements required for said verification and established in the steps of our modeling scheme. These numbers are intended to give an overview over the complexity of the example artefacts to be specified in the modeling scheme, the resulting effort required for their manual specification and comprehension, and the effect on verification times.

For all artefacts required for verification, the table lists both the number of contained subcomponents and the size (number of nodes) of the largest component. For a set of rules, the subcomponents are the contained graph rules. For a constraint, the subcomponents are the individual graph constraints joined conjunctively to form the complete constraint. It should be noted that in our examples, the constraints listed below always conform to the structure of a logical conjunction; in general, this does not need to be the case. In both categories, the maximum size is the maximum number of nodes encountered in such a subcomponent. Nodes in application conditions in a subcomponent (rule or constraint) also count towards the size of the subcomponent. For type graphs, there exists only one type graph of each kind; hence, the maximum size is the number of nodes of the respective type graph.

While some elements involved in the specification of the model transformation (e.g. tgg and the TGG constraint $\mathcal{C}_{\text{tgg}}$) grow to a certain size such that manual handling and analysis may be problematic, most elements concerned with the specification of behavior preservation itself are either small in size (such as the runtime constraint $\mathcal{C}^{\square}_{\text{RT}}$) or can be derived automatically (such as $\mathcal{C}^{\square}_{\text{MT}}$ or $\mathcal{C}^{\square}_{\text{Pair}}$). While we did not exploit this potential for automation in our modeled examples, it would certainly be required for even larger case studies.

However, since a number of constraints and graph rules are quite similar in structure and content, a lot of otherwise redundant work could be avoided by copying and adjusting constraints or rules. For instance, in Example 23, $\mathcal{C}^{\text{Cor}}_{\text{MT}}$ and $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ only differ in the four additional *active* edges of $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$, so our process saw heavy use of copying of existing elements. Because of the construction of $\mathcal{C}^{\square}_{\text{Pair}}$ from

$\mathcal{P}(l_s, l_t)^{\square}$ (on a general level), this will also apply to other instances of our problem statements.

Considering all elements in general, it should be noted that different elements may be specified by different parties in the process of the development of the source and target modeling languages, the model transformation, model semantics and behavior preservation. Indeed the specification effort listed in Table 11 will in practice often be distributed among various developers acting in different roles. It should also be noted that certain modeling steps will happen independently of other steps. For example, the specification of a model transformation can be based on existing source and target modeling languages that were established independently of an intent to design a model transformation.

For the cases of model transformation constraints in the equivalence (Example 23) and refinement (Example 25) approaches, as mentioned before, we can automatically derive the model transformation constraint $\mathcal{C}^{\text{Cor}}_{\text{MT}}$ from the pair constraint $\mathcal{C}^{\text{Cor}}_{\text{Pair}}$ for cases where the semantics rules only have the trivial case true as application conditions and do not delete any nodes. In those cases, our pair constraints are conjunctions of nested constraints of the form $\forall(P, \exists N)$ and our automatic derivation fulfills the requirements of Definition 40, which is proven below. (As will be explained in Section 7.2 below, this constraint form is also a requirement for our invariant checking tool [2, 15] to automatically perform the required inductive invariant checks.)

Construction 1. *Given a formally covered model transformation (Def. 26) with source and target semantics rules (in $\mathcal{R}_s$ and $\mathcal{R}_t$) that do not delete any nodes and have* true *as application condition, let $\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ be a pair constraint (Def. 39) with $\mathcal{C}^{\text{Cor,f}}_{\text{Pair}} = \bigwedge_{(\rho_s,\rho_t)\in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})} C_{(\rho_s,\rho_t)}$ where all $C_{(\rho_s,\rho_t)}$ are of the form $C_{(\rho_s,\rho_t)} = \forall(L_s, \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t))$. $\mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ is formed analogously.*

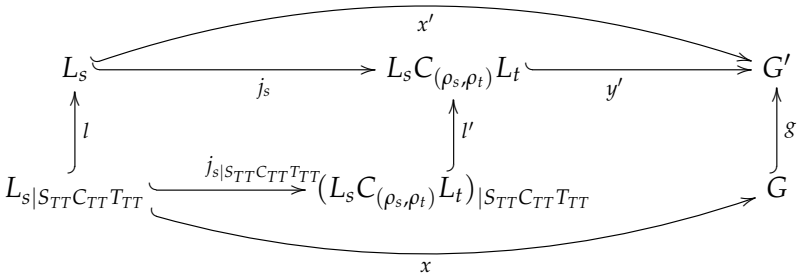
We define an operation rest_f *that derives the forward model transformation constraint from the forward pair constraint, i.e. $\mathcal{C}^{\text{Cor,f}}_{\text{MT}} = \text{rest}_f(\mathcal{C}^{\text{Cor,f}}_{\text{Pair}})$:*

$$
\begin{aligned}
& \text{rest}_f\Big(\bigwedge\nolimits_{(\rho_s,\rho_t)\in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})} C_{(\rho_s,\rho_t)}\Big) = \bigwedge\nolimits_{(\rho_s,\rho_t)\in \text{Pair}(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})} \text{rest}_f(C_{(\rho_s,\rho_t)}) \\
& \text{rest}_f(\forall(L_s, \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t))) \\
= \quad & \forall(L_{s|S_{TT}C_{TT}T_{TT}}, \exists(j_{s|S_{TT}C_{TT}T_{TT}} : L_{s|S_{TT}C_{TT}T_{TT}} \to (L_s C_{(\rho_s,\rho_t)} L_t)_{|S_{TT}C_{TT}T_{TT}})).
\end{aligned}
$$

There is an analogous construction rest_b *such that $\mathcal{C}^{\text{Cor,b}}_{\text{MT}} = \text{rest}_b(\mathcal{C}^{\text{Cor,b}}_{\text{Pair}})$ and $\mathcal{C}^{\text{Cor}}_{\text{MT}} = \text{rest}_f(\mathcal{C}^{\text{Cor,f}}_{\text{Pair}}) \wedge \text{rest}_b(\mathcal{C}^{\text{Cor,b}}_{\text{Pair}})$ fulfills the requirement specified in Definition 40:*

Proof. Given $\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ as specified above and given $\mathcal{C}^{\text{Cor}}_{\text{MT}} = \text{rest}_f(\mathcal{C}^{\text{Cor,f}}_{\text{Pair}}) \wedge \text{rest}_b(\mathcal{C}^{\text{Cor,b}}_{\text{Pair}})$, we have to show that for all graphs $G \in \mathcal{L}(S_{TT}C_{TT}T_{TT})$ with $G \not\models \mathcal{C}^{\text{Cor}}_{\text{MT}}$, there is a graph $G' \in \mathcal{L}(S_{RT}C_{TT}T_{RT})$ such that $G'_{|S_{TT}C_{TT}T_{TT}} = G$ and $G' \not\models \mathcal{C}^{\text{Cor}}_{\text{Pair}}$.

Hence, consider a graph $G \in \mathcal{L}(S_{TT}C_{TT}T_{TT})$ with $G \not\models \mathcal{C}^{\text{Cor}}_{\text{MT}}$. Then, we have $G \not\models \mathcal{C}^{\text{Cor,f}}_{\text{MT}}$ or $G \not\models \mathcal{C}^{\text{Cor,b}}_{\text{MT}}$. Since both cases can be treated analogously, we will consider $G \not\models \mathcal{C}^{\text{Cor,f}}_{\text{MT}}$ here, which leads to the existence of a $C_{(\rho_s,\rho_t)} = \forall(L_s, \exists(j_s : L_s \to L_s C_{(\rho_s,\rho_t)} L_t))$ such that $G \not\models C_{(\rho_s,\rho_t)}$ (where $(\rho_s, \rho_t) \in \text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)$).



Consequently, there exists a morphism $x : L_{s|S_{TT}C_{TT}T_{TT}} \hookrightarrow G$, such that there does not exist a morphism $y : (L_s C_{(\rho_s,\rho_t)} L_t)_{|S_{TT}C_{TT}T_{TT}} \hookrightarrow G$ with $y \circ j_{s|S_{TT}C_{TT}T_{TT}} = x$.

By constructing the pushout (G', x', g) over l and x, we get a graph $G' \in \mathcal{L}(S_{RT}C_{TT}T_{RT})$ with morphisms $x' : L_s \hookrightarrow G'$ such that $x = x'_{|S_{TT}C_{TT}T_{TT}}$ and $g : G \hookrightarrow G'$ such that $G' \setminus g(G)$ will only contain runtime elements and hence, $G'_{|S_{TT}C_{TT}T_{TT}} = G$.

We then need to show $G' \not\models \mathcal{C}^{\text{Cor}}_{\text{Pair}}$ and will do so by contradiction: Assume $G' \models \mathcal{C}^{\text{Cor}}_{\text{Pair}}$ and hence, $G' \models \mathcal{C}^{\text{Cor,f}}_{\text{Pair}}$, which, given $x' : L_s \hookrightarrow G'$ as above, implies the existence of a morphism $y' : L_s C_{(\rho_s,\rho_t)} L_t \hookrightarrow G'$ with $y' \circ j_s = x'$. Then, there exists a morphism $y = y'_{|S_{TT}C_{TT}T_{TT}}$ with $y : (L_s C_{(\rho_s,\rho_t)} L_t)_{|S_{TT}C_{TT}T_{TT}} \hookrightarrow G$ such that $y \circ j_{s|S_{TT}C_{TT}T_{TT}} = x$. This is a contradiction (to the non-existence of y as stated above); thus, we have $G' \not\models \mathcal{C}^{\text{Cor}}_{\text{Pair}}$, which concludes the proof. □

7.2. Evaluation of Verification Scheme

The inductive invariant checks for the TGG rules (V_{trans}) and pair rules (V_{sem}) can be performed automatically by our invariant checking tool [2, 15], which has also already been applied for the more restricted case of simple equivalence described in [27]. Similarly, runtime conformity with respect to dynamic constraints can also be checked. In particular, we check whether a graph constraint F is an inductive invariant for a typed GTS with constraint gts^C. However, the rules and constraints that can be checked are subject to the restrictions imposed by our tool:

1. Left application conditions can only have the form $\bigwedge_{i\in I} \neg\exists n_i$, such that rules must have the form $\rho = (\langle L \hookleftarrow I \hookrightarrow R, \rangle, \bigwedge_{i\in I} \neg\exists n_i)$ (for morphisms $n_i : L \hookrightarrow N_i$).

2. Graph constraints to be verified as inductive invariants (F) or to be part of the GTS (C) must have the form $\bigwedge_{i\in I} \neg\exists(i_{P_i}, \mathrm{ac}_i)$, for morphisms $i_{p_i} : \emptyset \hookrightarrow P_i$ and application conditions ac_i of the general form $\mathrm{ac}_i = \bigwedge_{j\in J} \neg\exists x_j$ for morphisms $x_j : P_i \hookrightarrow X_j$.

The latter restriction leads to a more specific condition for the pair constraint $\mathcal{C}^{\square}_{\mathrm{Pair}}$ and hence, for the pair rules $\mathcal{P}(l_s, l_t)^{\square}$ and for the source and target GTSs gts_s and gts_t. Since the nesting level in the pair constraint is restricted, the rules in gts_s and gts_t must not have any nested application conditions (except for the trivial case true). If, however, any unwanted negative application conditions exist and if these conditions only prohibit the existence of one edge, they can be simulated by additional constructions as explained in Appendix A. This is, in fact, the case for the *initE* and *initS* rules of all three examples and the scheme is applied for all the verification steps involved.

Given this restriction and the resulting absence of nested application conditions in semantics rules, which leads to a simple $\forall(P, \exists N)$ for our pair constraints, we can apply the automatic derivation (Construction 1) of the model transformation constraint $\mathcal{C}^{\square}_{\mathrm{MT}}$ from the pair constraint with correspondences $\mathcal{C}^{\square}_{\mathrm{Pair}}$ for the cases of behavioral equivalence and refinement (Examples 23 and 25).

We also employ a simplification scheme to replace one-to-one correspondence nodes and accompanying edges by simple edges as explained in Appendix A. This lowers the size of rules and constraints in our examples considerably and allows for a verification in reasonable time and memory consumption.

In the special case of Example 15, the pair constraint (before simplification) does not follow the required structure (see Figures 11 and 31). However, the implication $\mathrm{App}(\rho_s) \Rightarrow \mathrm{App}(\rho_t)$ which is part of the forward pair constraint can be written as $\neg(\mathrm{App}(\rho_s) \wedge \neg\mathrm{App}(\rho_t))$ and, for rules with a deletable condition that is trivially true, $\mathrm{App}(\rho_s) \wedge \neg\mathrm{App}(\rho_t)$ is equivalent to $\exists(L_s, ac_s \wedge \neg\exists(L_s \hookrightarrow L_s + L_t, ac_{s+t})$ with L_s and L_t being the LHS of ρ_s and ρ_t, respectively, and ac_{s+t} the resulting application condition for the parallel rule $\rho_s + \rho_t$. After negation, we have $\forall(L_s, \neg ac_s \vee \exists(L_s \hookrightarrow L_s + L_t, ac_{s+t})$, which, after applying the simplification procedure for simple negative application conditions, has the required structure.

It can easily be seen that verification step V_{init} – the satisfiability of the respective constraints by the respective axiom $S_A C_A T_A$ – holds, although automatic verification of this step would indeed be possible. Table 11 shows the results of the more elaborate verification processes for the verification steps V_{trans} and V_{sem} in our three examples: simple equivalence, equivalence, and refinement. Note that these numbers correspond to the examples adjusted by the simplification measures explained above: emulating negative application conditions containing single edges by positive edges and replacing correspondence nodes by edges. The only exception are

our examples for approaching behavioral equivalence – here, correspondence nodes have not been replaced by correspondence edges since the examples' complexity does not require it. All cases are behavior preserving in the sense of their respective problem statements.

We can observe a drastic difference between our examples for simple equivalence and equivalence/refinement due to the increase in size of rules and constraints. These differences are owed to the higher complexity of our more elaborate examples; in particular, the inclusion of correspondence structures leads to a higher number of entities.

While our tools require certain restrictions on graph rules and constraints and therefore requires the mentioned simplification step for application conditions in rules, it is able to yield a correct result in adequate time. On the other hand, more general tools of higher expressive power may struggle with performance for the verification of systems as complex as our examples for behavioral equivalence and refinement (see [15]). In contrast to tools with interactive verification, our tool has the advantage of running completely automated. Finally, while false negatives are possible in certain cases, our tool yields meaningful symbolic counterexamples for failed verification attempts. In case of doubt or to better understand the problem, these counterexamples can be investigated by hand.

Table 11: Evaluation data with numbers of subcomponents and maximum node size for artefacts involved in modeling and verification schemes

		simple equivalence (Example 15)		equivalence (Example 23)		refinement (Example 25)	
modeling step/artefact		number of subcompo-nents	max. size	number of subcompo-nents	max. size	number of subcompo-nents	max. size
M_{lang}	S_{TT}	1	3	1	3	1	3
	$\mathcal{C}_S$	10	3	11	3	11	3
	T_{TT}	1	3	1	4	1	4
	$\mathcal{C}_T$	10	3	8	3	6	3
M_{sem}	$\mathcal{R}_s$	3	3	2	7	2	7
	S_{RT}	1	3	1	3	1	3
	$\mathcal{C}_s^{gts}$	5	2	6	3	6	3
	$\mathcal{R}_t$	3	3	2	9	2	9
	T_{RT}	1	3	1	4	1	4
	$\mathcal{C}_t^{gts}$	5	2	6	3	6	3
M_{trans}	$\mathcal{R}$	3	9	2	16	4	16
	$S_{RT}C_{TT}T_{RT}$	1	9	1	7	1	7
	$\mathcal{C}_{\text{tgg}}$	32	3	28	3	26	3
M_{pres}	corres	0	0	2	0	2	0
	$\mathcal{P}(l_s, l_t)^{\square}$	3	6	2	16	2	16
	$\mathcal{C}_{\text{RT}}^{\square}$	2	3	2	2	1	2
	$\mathcal{C}_{\text{Pair}}^{\square}$	6	6	4	16	2	16
	$\mathcal{C}_{\text{MT}}^{\square}$	6	9	4	16	2	16
verification step		result	time	result	time	result	time
V_{init}		true	trivial	true	trivial	true	trivial
V_{trans}		true	1.0 s	true	43 min	true	50 min
V_{sem}		true	5.5 s	true	0.9 s	true	0.5 s

8. Discussion

In the following we will discuss whether the results described in this report are applicable for a large class of cases where we want to verify behavior preservation.

The applicability of our approach for verifying behavior preservation of model transformations at the transformation level (Definition 6), is mainly related to the question how well chosen our formalizations for the basic concepts such as the modeling language (Definition 1), the model semantics (Definition 2), the model transformation (Definition 3), and the notion for behavioral equivalence or refinement (Definition 5) are. Consequently, we will discuss in the following our choices, their pros and cons, as well as alternative choices.

To fulfill the needs of Definition 1 concerning the definition of the modeling languages, we chose typed graphs enriched with a constraint as formalization. As extensions for typed graphs such as attributes [18], inheritance on the node [13, 29] and edge types [50], and many other extensions exist, this is not really a limitation concerning the expressiveness and fits well to the usually employed meta models [1].

A model semantics according to Definition 2 has to be provided in our approach in form of an extension of the model by dynamic elements and graph transformation systems that describe how the state can change stepwise (as proposed already in [21, 33]). Consequently, the required formalization demands conceptually that an interpreter is defined that describes the behavior by means of a state transition system. Such a formalization can be done straightforward for most behavioral models related to software. However, if the considered models include continuous elements as in case of models of physical processes such a formalization may be problematic.

In our formalization we require that each static graph satisfy trivially each dynamic constraint: $\mathcal{L}(TG, \mathcal{C}) \subseteq \mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$. However, the proven verification scheme can be easily adjusted to cases where this is not true and where some initialization phase is required to establish the dynamic constraints $\mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts}$ by requiring that V_{init} is split into a first check for the outcome of the grammar not including the dynamic constraints $S_A C_A T_A \vDash \mathcal{C}_{\text{RT}}^{\square} \wedge \mathcal{C}_{\text{Pair}}^{\square} \wedge \mathcal{C}_{\text{MT}}^{\square}$ and a check that the initialization phase then preserves all the other constraints $S_A C_A T_A \vDash \mathcal{C}_{\text{RT}}^{\square} \wedge \mathcal{C}_{\text{Pair}}^{\square} \wedge \mathcal{C}_{\text{MT}}^{\square}$ and guarantees the dynamic constraint $\mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts}$ after termination.

For the formalization of model transformations (Definition 3) we chose triple graph grammars (TGGs). In [26, 35] we showed for TGGs enriched with a constraint that they are conform with our TGG implementation. Thus we can guarantee for that implementation that forward and backward transformation implementations are indeed behavior preserving. It has also been shown that TGGs are to some

extent similar to relational QVT [30] and thus we can conclude that the results conceptually cover the core of the class of relational model transformations. Our results can also be adapted to the case of operational model transformations as shown in [17]. In this case the verification part for checking bisimulation or simulation (V_{sem}) can be reused after the establishment of condition $\mathcal{C}^{\square}_{\mathrm{MT}}$ for the model transformation results by other means. In this context it may be noted also that in particular the constraints belonging to the bisimulation constraint including runtime elements might not hold for a source and target model on which no semantics rule has been applied yet. In particular, this could be the case if the presence of some runtime elements is required by the respective constraint. In this case a possible workaround would be to add an initialization phase for the semantics $V_{sem,init}$ in which these constraints with runtime elements being part of the bisimulation constraint are established first. After that the regular verification part as described in V_{sem} can be applied.

For modeling language, model semantics, and model transformation holds as outlined that the limitations of the considered variant of graph transformation system are not really a limiting factor concerning expressiveness of our approach, as the results obviously also apply for graph transformation extensions such as attributes, inheritance on the node and edge types, many other extensions, and even the general concept of $\mathcal{M}$-adhesive replacement systems [19]. It has to be noted, however, that the presented results concerning the automated checking are only feasible for models of rather limited expressiveness.

The choice of a particular behavioral equivalence or refinement (Definition 5) differs from the former choices as we do not only chose a particular formalization but also have to select one option from an overwhelming number of alternative notions of equivalences and refinement for labeled transition systems that have been studied in the literature (cf. [51, 52]).

The chosen behavioral equivalence bisimulation and behavioral refinement simulation are the main cases for a class of these alternatives where the comparison is based on pairs of states. Other notions in contrast look on traces and/or refusals to consider the comparison on a more abstract level not taking into account the state space of the two involved models (cf. [51, 52]). However, due to the fact that the two models are linked to each other via transformation steps (which is encoded in the correspondence model) the assumption that the state spaces are structurally similar is very well justified and thus it seems reasonable to limit our considerations to notions for behavioral equivalence and refinement where the comparison is based on pairs of states.

Note that our notion of behavioral equivalence and refinement implicitly assumes that unique correspondences exist between each source and target model of a model transformation. Otherwise the following exceptional case could arise: for one

correspondence variant of a model transformation instance a bisimulation relation or refinement can be found, whereas for another correspondence variant this is not the case. This exceptional case does not play a significant role within our framework, since for practical reasons as explained in [26] we assume the correspondence structure to be unique anyway in our model transformation approach. Moreover, in our particular verification approach we check the more severe case that for each correspondence variant a bisimulation/simulation relation can be found.

Within the class of behavioral equivalence and refinement where the comparison is based on pairs of states a number of alternatives other than bisimulation and simulation exist. One alternative, for example, would be to consider weak forms of bisimulation and simulation where internal steps are not taken into account. To do so we would have to join possibly multiple subsequent silent steps sequentially with one labeled rule such that bisimulation/simulation of the adjusted GTSs corresponds to weak bisimulation/simulation for the original GTSs. This can be done by gluing rules [18, 19] into so-called concurrent rules in case we have to join only a small finite number of rules. In particular, as described in our ongoing work on k-induction [16] the invariant checking approach needs to take into account up to k steps to establish an invariant. In case the number of silent steps is unbounded another option could be to allow the silent steps in addition to the usual corresponding pairs of source and target steps such that either the source or target can do silent steps or joint visible steps. Note however, that in this case an invariant has to be identified that also holds when applying silent steps on each side.

We can further check simulation also in cases where we do not have a bijective relabeling such that multiple target rules correspond to one rule ρ_s of the source model but to any rule ρ_t of the target modelonly one rule of the source model corresponds. We can do so by employing the original definition for the *pair constraint* $C^{\text{Cor,b}}_{\text{Pair}}$ *with correspondences* and the original pair rules. As the pair rules only covering the backward direction each rule in the target model has a unique rule in the source model and thus they guarantee that each behavior in the target model is also possible in the source model.

In particular concerning the simulation case holds that it is oftentimes a too weak refinement. This is because, for example, it does not necessarily preserve deadlock-freedom when the refined behavior of a target model is substituted for the refined behavior of the source model in an arbitrary context. The stronger notion of *ready-simulation* [51], which in contrast to simulation preserves deadlock-freedom for substitution, requires that the pair constraints have to be enriched to not only require that the target is simulated by the source, but also requires in the opposite direction that if a certain step with an external label is possible in the source model a step with the same label must be offered by the target model as well. Like for the above-discussed case for simulation we can support that multiple

target rules correspond to one rule ρ_s of the source model but only one source rule corresponds to any rule ρ_t of the target model. We can then adjust the definition for a *pair constraint* $\mathcal{C}^{\text{Cor,f}}_{\text{Pair},rs}$ *with correspondences* as follows to check ready-simulation using $\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair},rs} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ and the original pair rules with

$$\mathcal{C}^{\text{Cor,f}}_{\text{Pair},rs} = \wedge_{\rho_s \in \mathcal{R}_s}(\forall(L_s, \text{ac}_{\text{App}(\rho_s)} \Rightarrow \vee_{(\rho_s,\rho_t)\in \text{Pair}(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)} \exists(j_s : L_s \rightarrow L_s C_{(\rho_s,\rho_t)} L_t, \text{ac}_{\text{App}(\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t)}))).$$

Note that $\mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$ guarantees that each behavior in the target is also possible in the source. The additional constraint then guarantees that whenever a rule in the source is enabled at least one corresponding rule in the target is also enabled. Thus, a transition with the same label in the target always exists, which guarantees deadlock freedom for substitution.

Concerning the automation of the verification, besides the feasibility of the verification as such, other issues to be considered are how helpful the tools are in developing a correct transformation that is indeed behavior preserving. If the model transformation in question is not behavior preserving according to our verification scheme, then the currently employed tool outputs specific counter examples that can be checked to understand why the checking failed. Hence, improvements of the checking tool yielding more compact or less witnesses (specific or symbolic) in case of failed verification tasks would be another very useful direction for a model transformation developer interested in verifying behavior preservation.

9. Conclusion and Future Work

We present the first fully automated and complete verification approach for behavior preservation of model transformations at the level of the model transformations. For model transformations specified by TGGs and semantic definitions for the input models and output models given by GTS rules, we were able to reduce the behavior preservation problem to an inductive invariant checking problem for GTSs derived from the TGG rules and semantics rules and constraints encoding of bisimulation or simulation and the applicability of equivalent steps in the source and target models. Furthermore, we were able to show that the result is of relevance for a large class of model transformations and that even for complex examples the required checks can be done automatically.

In our future work we plan to investigate even larger examples and case studies and identify which characteristics may prevent to apply our approach. Based on these insights we want to develop extensions for our approach that cover more cases. Directions may be more expressive models, different kind of semantics, or alternative notions for behavior equivalence and refinement. Another direction to extend the scope of our approach will be to transfer the approach not only to operational model transformations as presented in [17] but also to hybrid ones.

The results presented in this report also imply that tools for checking inductive invariants for GTS that support more expressive GTS variants than the employed invariant checker or can handle more complex rules and constraints would be a valuable contribution. Thus we plan to investigate such improvements as well.

Finally, we plan to integrate our transformation level verification approach for behavior preservation into the overall model transformation development process. In particular, we aim to build debugging support w.r.t. behavior preservation.

Acknowledgements
This work was developed in the course of the project Correct Model Transformations II (GI 765/1-2) and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft. See `https://hpi.de/en/giese/research/projects/cormorant.html`.

We would also like to thank Jürgen Dingel, who contributes to the CorMorant II research project as a Mercator Fellow, for the valuable discussions about our approach and its relation to the general behavior preservation problem that helped us considerably to present our result in this report in a more comprehensible manner.

References

[1] T. Arendt, A. Habel, H. Radke, and G. Taentzer. From Core OCL Invariants to Nested Graph Constraints. In H. Giese and B. König, editors, *Graph Transformation*, volume 8571 of *Lecture Notes in Computer Science*, pages 97–112, Cham, 2014. Springer.

[2] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *Proceedings of the* 28^{th} *International Conference on Software Engineering*, pages 72–81, New York, 2006. ACM.

[3] B. Becker, L. Lambers, J. Dyck, S. Birth, and H. Giese. Iterative Development of Consistency-Preserving Rule-Based Refactorings. In J. Cabot and E. Visser, editors, *Theory and Practice of Model Transformations*, volume 6707 of *Lecture Notes in Computer Science*, pages 123–137, Berlin/Heidelberg, 2011. Springer.

[4] J. Bezivin, G. Dupe, F. Jouault, G. Pitette, and J. E. Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In 2^{nd} *OOPSLA Workshop on Generative Techniques in the context of Model-Driven Architecture*, 2003.

[5] D. Bisztray, R. Heckel, and H. Ehrig. Verification of Architectural Refactorings by Rule Extraction. In J. L. Fiadeiro and P. Inverardi, editors, *Fundamental Approaches to Software Engineering*, volume 4961 of *Lecture Notes in Computer Science*, pages 347–361, Berlin/Heidelberg, 2008. Springer.

[6] D. Bisztray, R. Heckel, and H. Ehrig. Compositional Verification of Architectural Refactorings. In R. de Lemos, J.-C. Fabre, C. Gacek, and M. ter Beek, editors, *Architecting Dependable Systems VI*, volume 5835 of *Lecture Notes in Computer Science*, chapter 13, pages 308–333. Springer, Berlin/Heidelberg, 2009.

[7] C. Blume, H. J. S. Bruggink, D. Engelke, and B. König. Efficient Symbolic Implementation of Graph Automata with Applications to Invariant Checking. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Transformations*, volume 7562 of *Lecture Notes in Computer Science*, pages 264–278, Berlin/Heidelberg, 2012. Springer.

[8] I. B. Boneva, J. Kreiker, M. E. Kurban, A. Rensink, and E. Zambon. Graph Abstraction and Abstract Graph Transformations (Amended Version). Technical Report TR-CTIT-12-26, Centre for Telematics and Information Technology, University of Twente, Enschede, 2012.

[9] F. Büttner, J. Cabot, and M. Gogolla. On Validation of ATL Transformation Rules by Transformation Models. In S. Weißleder, L. Lúcio, H. Cichos, and F. Fondement, editors, *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verificatin and Validation*, pages 9:1–9:8, New York, 2011. ACM.

[10] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara. Verification and validation of declarative model-to-model transformations through invariants. *Journal of Systems and Software*, 83(2):283–302, 2010.

[11] M. Charpentier. Composing Invariants. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 401–421, Berlin/Heidelberg, 2003. Springer.

[12] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varró. VIATRA – Visual Automated Transformations for Formal Verification and Validation of UML Models. In J. Richardson, W. Emmerich, and D. Wile, editors, *ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270. IEEE Press, 2002.

[13] J. de Lara, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Attributed Graph Transformation with Node Type Inheritance. *Theoretical Computer Science*, 376(3):139–163, 2007.

[14] J. de Lara and G. Taentzer. Automated Model Transformation and Its Validation Using AToM 3 and AGG. In A. F. Blackwell, K. Marriott, and A. Shimojima, editors, *Diagrammatic Representation and Inference*, volume 2980 of *Lecture Notes in Computer Science*, pages 182–198, Berlin/Heidelberg, 2004. Springer.

[15] J. Dyck and H. Giese. Inductive Invariant Checking with Partial Negative Application Conditions. In F. Parisi-Presicce and B. Westfechtel, editors, *Graph Transformation*, volume 9151 of *Lecture Notes in Computer Science*, pages 237–253, Cham, 2015. Springer.

[16] J. Dyck and H. Giese. K-Inductive Invariant Checking for Graph Transformation Systems. Technical report, Hasso Plattner Institute at the University of Potsdam, 2017. Forthcoming.

[17] J. Dyck, H. Giese, L. Lambers, S. Schlesinger, and S. Glesner. Towards the Automatic Verification of Behavior Preservation at the Transformation Level for Operational Model Transformations. In J. Dingel, S. Kokaly, L. Lúcio, R. Salay, and H. Vangheluwe, editors, *Analysis of Model Transformations*, volume 1500 of *CEUR Workshop Proceedings*, pages 36–45, 2015.

[18] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, New York, 2006.

[19] H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas. $\mathcal{M}$-adhesive transformation systems with nested application conditions. part 1: Parallelism, concurrency and amalgamation. *Mathematical Structures in Computer Science*, 24, 2014.

[20] H. Ehrig, A. Habel, and L. Lambers. Parallelism and Concurrency Theorems for Rules with Nested Application Conditions. *Electronic Communications of the EASST*, 26, 2010.

[21] G. Engels, J. H. Hausmann, R. Heckel, and S. Sauer. Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 – The Unified Modeling Language*, volume 1939 of *Lecture Notes in Computer Science*, pages 323–337, Berlin/Heidelberg, 2000. Springer.

[22] G. Engels, A. Kleppe, A. Rensink, M. Semenyak, C. Soltenborn, and H. Wehrheim. From UML Activities to TAAL – Towards Behaviour-Preserving Model Transformations. In I. Schieferdecker and A. Hartman, editors, *Model Driven Architecture – Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 94–109, Berlin/Heidelberg, 2008. Springer.

[23] C. Ermel, J. Gall, L. Lambers, and G. Taentzer. Modeling with Plausibility Checking: Inspecting Favorable and Critical Signs for Consistency between Control Flow and Functional Behavior. In D. Giannakopoulou and F. Orejas, editors, *Fundamental Approaches to Software Engineering*, volume 6603 of *Lecture Notes in Computer Science*, pages 156–170, Berlin/Heidelberg, 2011. Springer.

[24] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language and Java. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 296–309, Berlin/Heidelberg, 2000. Springer.

[25] H. Giese, S. Glesner, J. Leitner, W. Schäfer, and R. Wagner. Towards Verified Model Transformations. In D. Hearnden, J. G. Süß, B. Baudry, and N. Rapin, editors, *Proc. of the 3rd International Workshop on Model Development, Validation and Verification (MoDeV^{2}a), Genova, Italy*, pages 78–93. Le Commissariat à l'Energie Atomique, 2006.

[26] H. Giese, S. Hildebrandt, and L. 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*, 13(1):273–299, 2014.

[27] H. Giese and L. Lambers. Towards Automatic Verification of Behavior Preservation for Model Transformation via Invariant Checking. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Transformations*, volume 7562 of *Lecture Notes in Computer Science*, pages 249–263, Berlin/Heidelberg, 2012. Springer.

[28] U. Golas, L. Lambers, H. Ehrig, and H. Giese. Toward Bridging the Gap between Formal Foundations and Current Practice for Triple Graph Grammars. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Graph Transformations*, volume 7562 of *Lecture Notes in Computer Science*, pages 141–155, Berlin/Heidelberg, 2012. Springer.

[29] U. Golas, L. Lambers, H. Ehrig, and F. Orejas. Attributed graph transformation with inheritance: Efficient conflict detection and local confluence analysis using abstract critical pairs. *Theoretical Computer Science*, 424:46 – 68, 2012.

[30] J. Greenyer and E. Kindler. Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. *Software and Systems Modeling*, 9(1):21–46, 2010.

[31] A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26(3/4):287–313, 1996.

[32] A. Habel and K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19:245–296, 2009.

[33] J. H. Hausmann, R. Heckel, and S. Sauer. Dynamic meta modeling with time: Specifying the semantics of multimedia sequence diagrams. *Software and System Modeling*, 3(3):181–193, 2004.

[34] R. Heckel and A. Wagner. Ensuring Consistency of Conditional Graph Grammars: A Constructive Approach. *Electronic Notes in Theoretical Computer Science*, 2:118–126, 1995.

[35] S. Hildebrandt, L. Lambers, B. Becker, and H. Giese. Integration of triple graph grammars and constraints. *Electronic Communication of the EASST*, 54, 2012.

[36] M. Hülsbusch, B. König, A. Rensink, M. Semenyak, C. Soltenborn, and H. Wehrheim. Showing Full Semantics Preservation in Model Transformation – A Comparison of Techniques. In D. Méry and S. Merz, editors, *Integrated Formal Methods*, volume 6396 of *Lecture Notes in Computer Science*, pages 183–198, Berlin/Heidelberg, 2010. Springer.

[37] G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the use of graph transformation in the formal specification of model interpreters. *Journal of Universal Computer Science*, 9(11):1296–1321, 2003.

[38] B. König and V. Kozioura. Augur 2 – A New Version of a Tool for the Analysis of Graph Transformation Systems. *Electronic Notes in Theoretical Computer Science*, 211:201–210, 2008.

[39] B. König and J. Stückrath. A General Framework for Well-Structured Graph Transformation Systems. In P. Baldan and D. Gorla, editors, *CONCUR 2014 – Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 467–481, Berlin/Heidelberg, 2014. Springer.

[40] M. Lawley and J. Steel. Practical Declarative Model Transformation with Tefkat. In J.-M. Bruel, editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 139–150, Berlin/Heidelberg, 2006. Springer.

[41] R. Milner. *Communication and Concurrency*. Prentice Hall, Hertfordshire, 1995.

[42] A. Narayanan and G. Karsai. Towards Verifying Model Transformations. *Electronic Notes in Theoretical Computer Science*, 211:191–200, 2008.

[43] A. Narayanan and G. Karsai. Verifying Model Transformations by Structural Correspondence. *Electronic Communications of the EASST*, 10, 2008.

[44] OMG. *MOF QVT Final Adopted Specification, OMG Document ptc/05-11-01*, 2005.

[45] K.-H. Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, University of Oldenburg, 2009.

[46] G. Rangel, L. Lambers, B. König, H. Ehrig, and P. Baldan. Behavior Preservation in Model Refactoring using DPO Transformations with Borrowed Contexts. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Graph Transformations*, volume 5214 of *Lecture Notes in Computer Science*, pages 242–256, Berlin/Heidelberg, 2008. Springer.

[47] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163, Berlin/Heidelberg, 1995. Springer.

[48] D. Steenken. *Verification of infinite-State Graph Transformation Systems via Abstraction*. PhD thesis, University of Paderborn, 2015.

[49] G. Taentzer. AGG: A Tool Environment for Algebraic Graph Transformation. In M. Nagl, A. Schürr, and M. Münch, editors, *Applications of Graph Transformation with Industrial Relevance*, volume 1779 of *Lecture Notes in Computer Science*, pages 481–488, Berlin/Heidelberg, 2000. Springer.

[50] G. Taentzer and A. Rensink. Ensuring Structural Constraints in Graph-Based Models with Type Inheritance. In M. Cerioli, editor, *Fundamental Approaches to Software Engineering*, volume 3442 of *Lecture Notes in Computer Science*, pages 64–79, Berlin/Heidelberg, 2005. Springer.

[51] R. J. van Glabbeek. The Linear Time – Branching Time Spectrum. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR'90. Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, Berlin/Heidelberg, 1990. Springer.

[52] R. J. van Glabbeek. The Linear Time – Branching Time Spectrum II. The Semantics of Sequential Systems with Silent Moves. In E. Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Berlin/Heidelberg, 1993. Springer.

[53] D. Varró and A. Pataricza. Automated Formal Verification of Model Transformations. In J. Jürjens, B. Rumpe, R. France, and E. B. Fernandez, editors, *CSDUML 2003: Critical Systems Development in UML*, pages 63–78. Technische Universität München, 2003.

A. Simplification

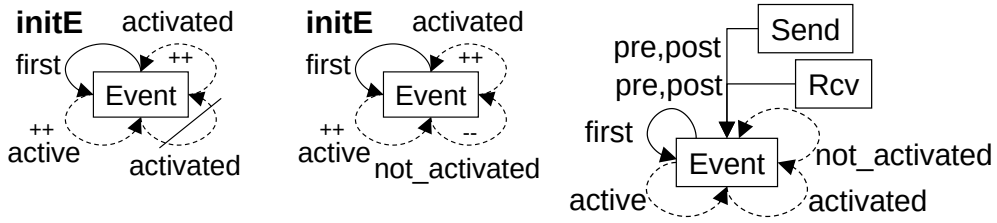
Depending on the various specific verification tools that may be employed to conduct the more complex verification steps V_{trans} and V_{sem}, the existence of nested application conditions in rules may pose a challenge. Most importantly, this influences a tool's capability to address the verification steps at all. However, even if a tool's formalism is expressive enough to support nested application conditions, the transformation of the verification task at hand into a system without such conditions may be desirable in terms of feasibility.

A restricted form of nested application conditions often encountered in rules of model semantics – and, in particular, in our examples – is a negative application condition prohibiting the existence of a single edge. More formally, given the LHS L of a rule, such application conditions have the form $\neg\exists(n : L \to N)$ with L and N being isomorphic with the exception of N having exactly one additional edge. For example, the *initE* and *initS* rules in Example 5 (Fig. 7(a) and 7(b)) have such negative application conditions.

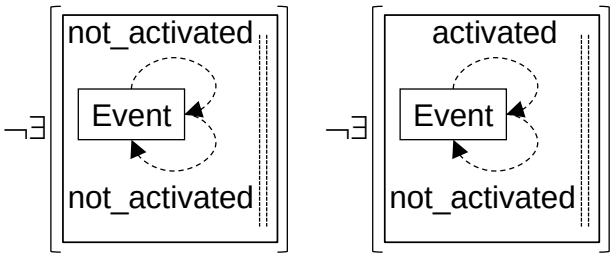
Assuming these negative edges have a uniform edge type, the occurrence of these very simple NACs in rules or graph constraints can be replaced with positive elements by replacing it by a new edge type (called here: *negative type*). This edge type represents the nonexistence of an edge of the original type (here: *positive type*) and hence, the satisfaction of the original NAC. Note that whenever an edge of the original type is created (deleted) in some rule, then also the rule itself must be changed such that it deletes (creates) a corresponding edge of the negative edge type. Then, a constraint C is an inductive invariant of the original GTS gts if the adapted GTS gts' has C' as an inductive invariant, where C' is C with two constraints joined conjunctively: one prohibiting the existence of two edges of the negative type on a node and one prohibiting the existence of edges of both the negative and positive type on a node.

Example 27. *Figure 22 shows the application of this simplification approach to the initE rule from Example 5, with the original rule depicted in Figure 22(a). Figure 22(b) shows the modified rule with an edge of type not_activated (the negative type to activated) instead of a negative application condition. Figures 22(c), 22(d), and 22(e) then show the addition of the negative type to the typegraph S_{RT} and the additional constraints to be conjunctively joined to C_s^{gts}. (Note that C_s^{gts} already contains a constraint forbidding two activated edges on one event.) Similarly, the procedure can be applied to the target model semantics.*

For the more general case of having a conjunction of such simple NACs, where single edges of different types are forbidden, the simplification approach can be applied by repeating the process described above for each such edge type.



(a) Original rule *initE* (b) Modified rule *initE* (c) Runtime type graph S_{RT}

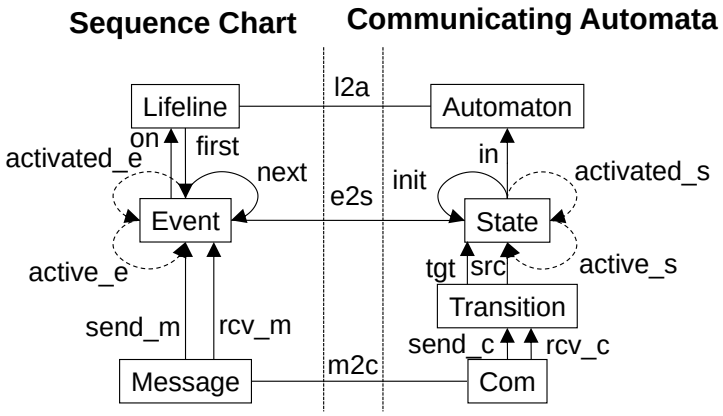


(d) Added constraint in C_s^{gts} (e) Added constraint in C_s^{gts}

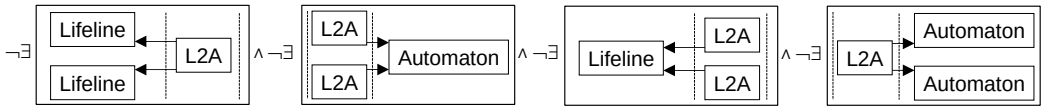
Figure 22: Original and adapted rule *initE*, runtime type graph S_{RT} and additional constraint in C_s^{gts}

Besides the simplification of negative application conditions, there is another possibility to reduce the artefacts' complexity. For the purpose of verification, TGG correspondence nodes with an edge to source and target nodes may be replaced by single edges. In particular, corresponding source and target nodes are connected only by such a single edge, which replaces the respective correspondence node together with its two outgoing edges. This procedure is correct if (for each artefact) there is exactly one connected source model node and exactly one connected target model node to which the respective correspondence node is connected. On the type graph level, for the respective correspondence node type, there must be only one connected source and target node type with unique connecting edge types. This conditions can be enforced by the type graph $S_{TT}C_{TT}T_{TT}$ and the TGG constraint $\mathcal{C}_{\text{tgg}}$, which is partly the case for our examples described in this report. More importantly, all correspondence nodes in our example constraints and rules follow the required structures. While we could extend this simplification process to a more general level and weaken the required conditions, this was not necessary for our examples and the TGG formalism we employ.

Example 28. *Figure 23(a) shows the type graph from Example 23, where the correspondence nodes and outgoing edges in the original type graph (Figure 14) have been replaced by correspondence edges. The TGG constraint $\mathcal{C}_{\text{tgg}}$ (Figure 41) restricts the cardinality of correspondence nodes and corresponding source and target elements as required. For example, the constraint fragment of $\mathcal{C}_{\text{tgg}}$ in Figure 23(b) enforces that one L2A correspondence node has at most one connected lifeline and at most one connected automata.*



(a) Simplified type graph



(b) Type graph constraints

Figure 23: Type graph with correspondence edges instead of nodes and type graph constraints enforcing cardinality restrictions

B. Models

B.1. All Figures of Example 15: Simple Equivalence (Tables 3, 4, 5, 6)

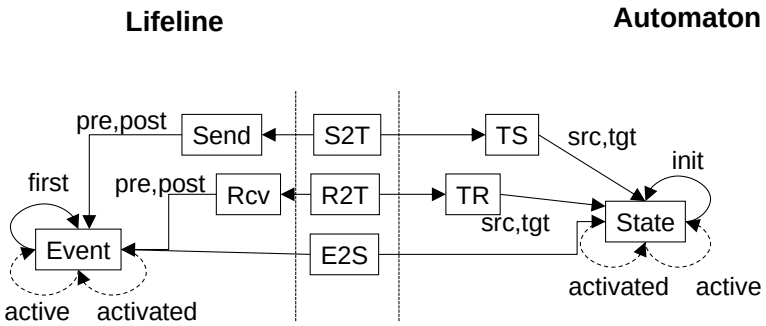


Figure 24: Type graph $S_{RT}C_{TT}T_{RT}$ (also shown in Figure 8)

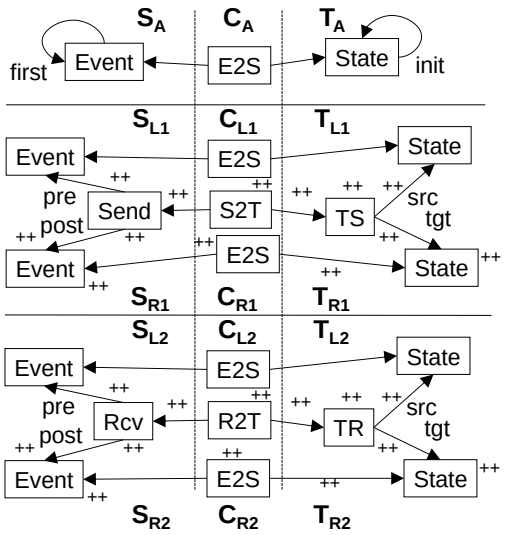


Figure 25: TGG rules $\mathcal{R}$ (also shown in Figure 8)

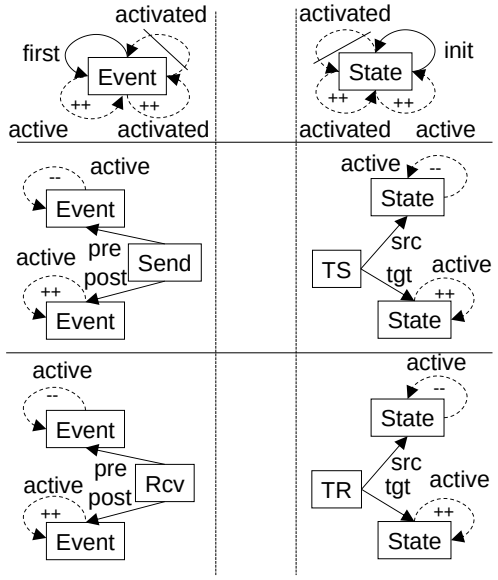


Figure 26: Pair rules $\mathcal{P}(l_s, l_t)$

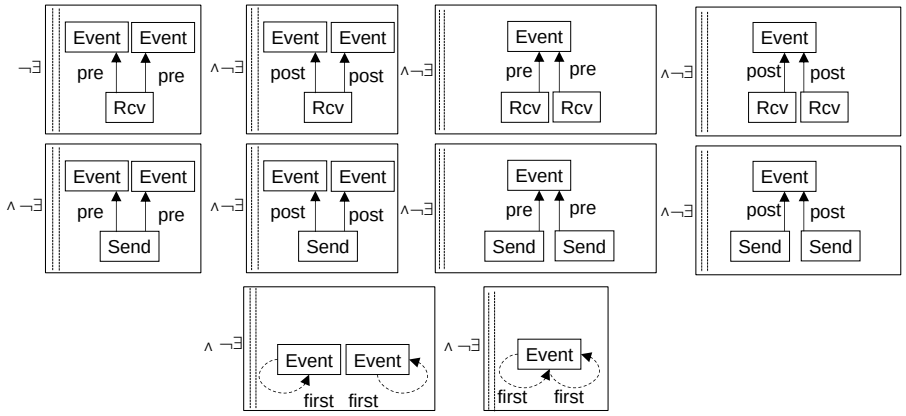


Figure 27: $\mathcal{C}_\mathcal{S}$ – source graph constraint

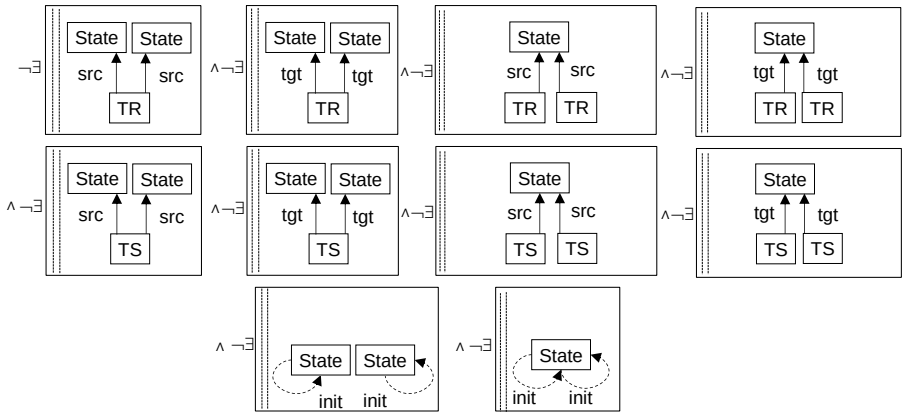


Figure 28: $\mathcal{C}_\mathcal{T}$ – target graph constraint

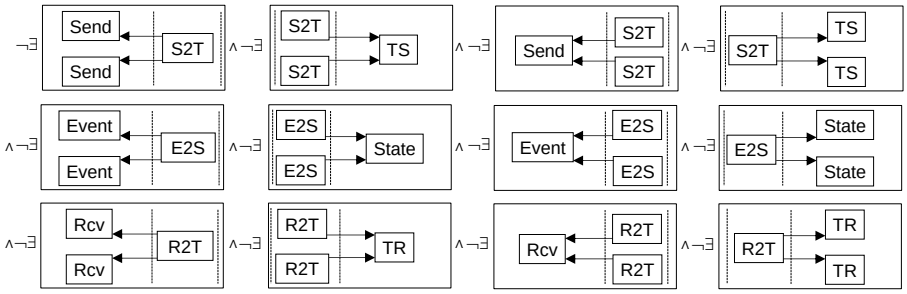
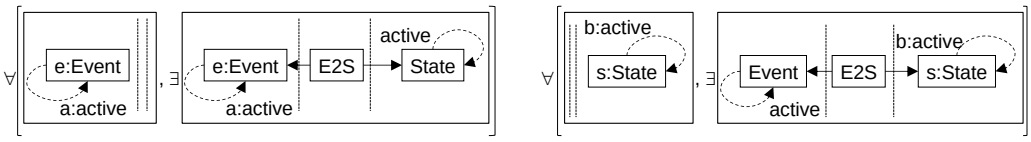
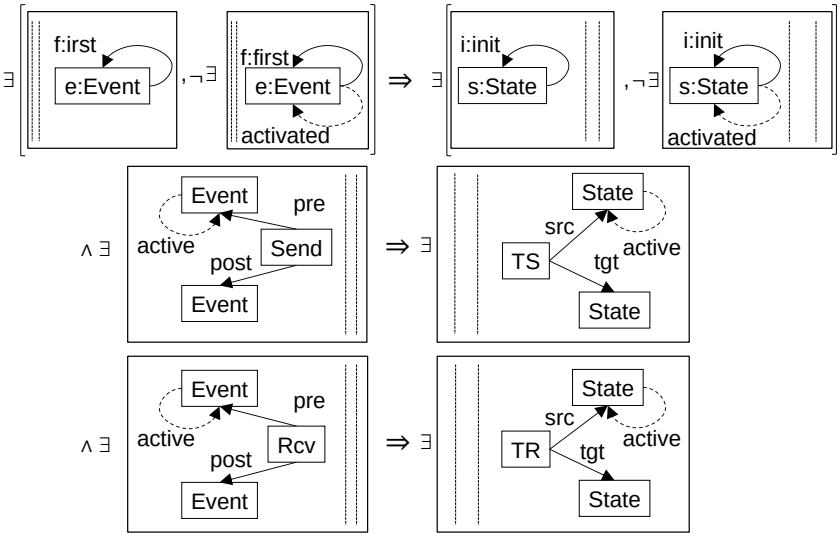


Figure 29: Fragment of $\mathcal{C}_{\text{tgg}}$

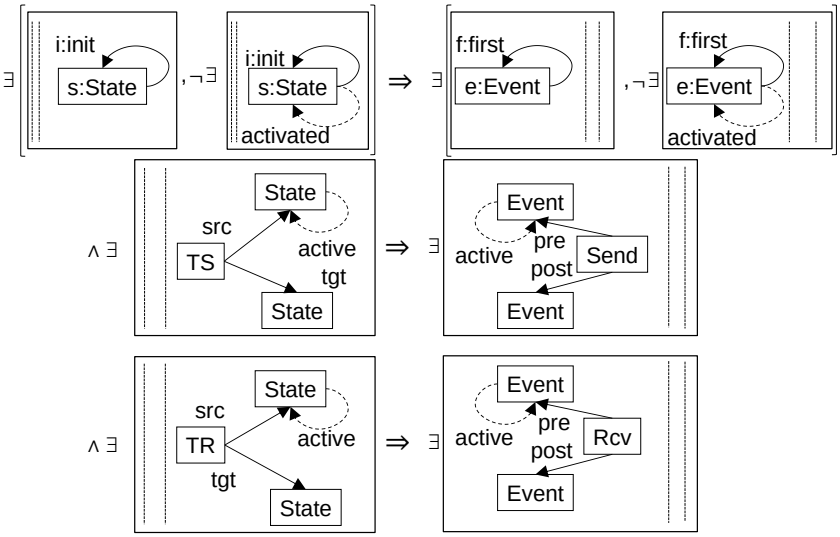


(a) $\mathcal{C}^{\mathrm{f}}_{\mathrm{RT}}$ (b) $\mathcal{C}^{\mathrm{b}}_{\mathrm{RT}}$

Figure 30: Runtime constraint $\mathcal{C}_{\mathrm{RT}} = \mathcal{C}^{\mathrm{f}}_{\mathrm{RT}} \wedge \mathcal{C}^{\mathrm{b}}_{\mathrm{RT}}$ (also shown in Figure 12)



(a) $\mathcal{C}^{\text{Rule,f}}_{\text{Pair}}$



(b) $\mathcal{C}^{\text{Rule,b}}_{\text{Pair}}$

Figure 31: Pair constraint $\mathcal{C}^{\text{Rule}}_{\text{Pair}} = \mathcal{C}^{\text{Rule,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Rule,b}}_{\text{Pair}}$ (cf. Figure 11)

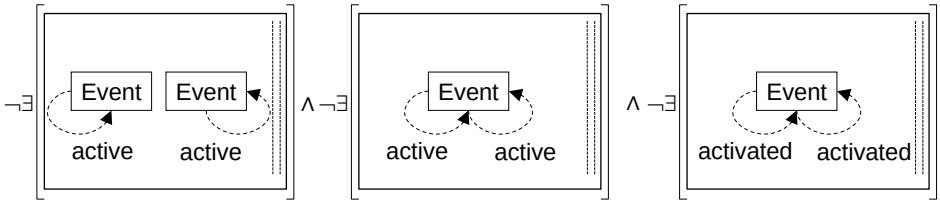


Figure 32: $\mathcal{C}_s^{gts}$ (also shown in Figure 6(a))

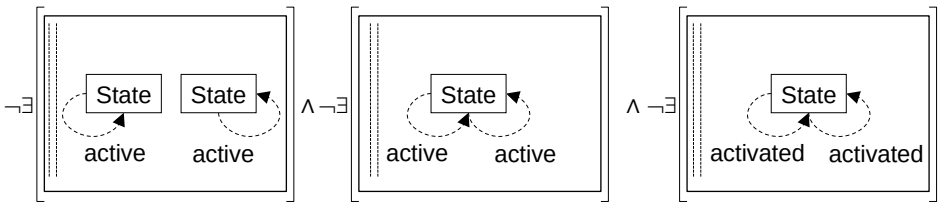
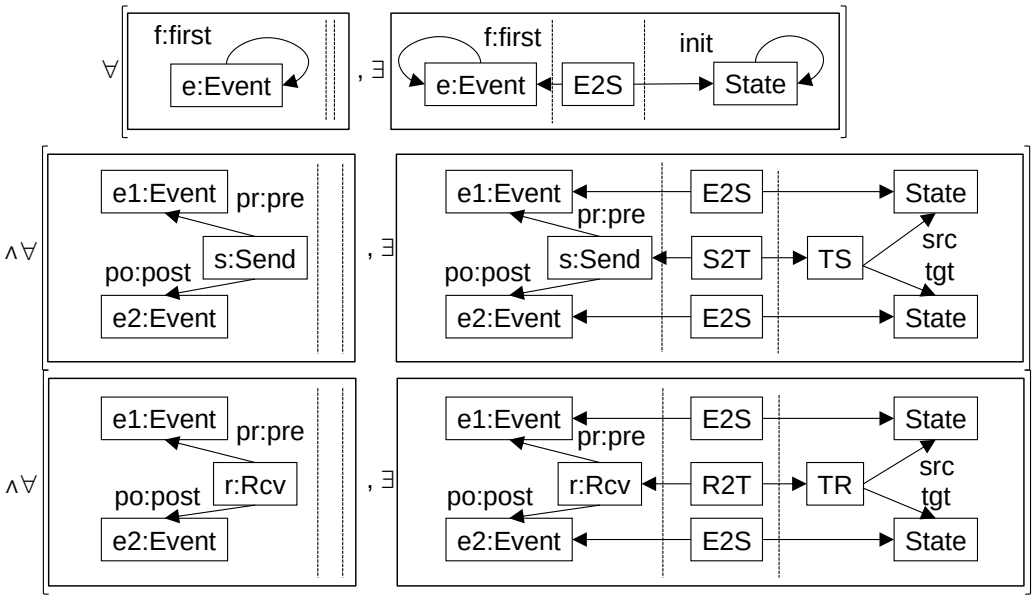
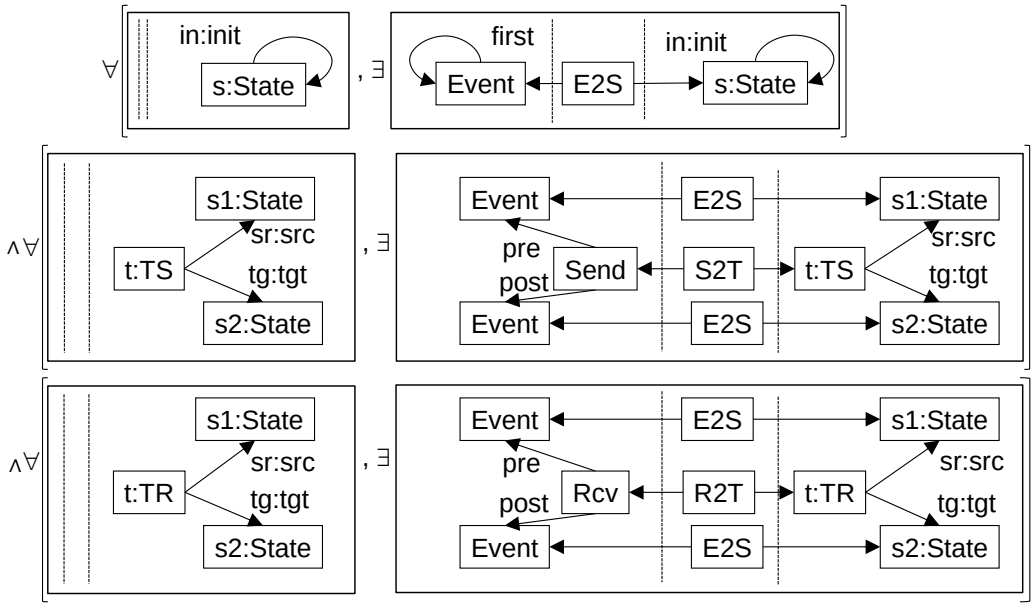


Figure 33: $\mathcal{C}_t^{gts}$ (also shown in Figure 6(b))



(a) $\mathcal{C}_{\text{MT}}^{\text{Man,f}}$



(b) $\mathcal{C}_{\text{MT}}^{\text{Man,b}}$

Figure 34: Model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Man}} = \mathcal{C}_{\text{MT}}^{\text{Man,f}} \wedge \mathcal{C}_{\text{MT}}^{\text{Man,b}}$ (cf. Figure 13)

B.2. All Figures of Example 23: Equivalence (Tables 3, 4, 5, 7)

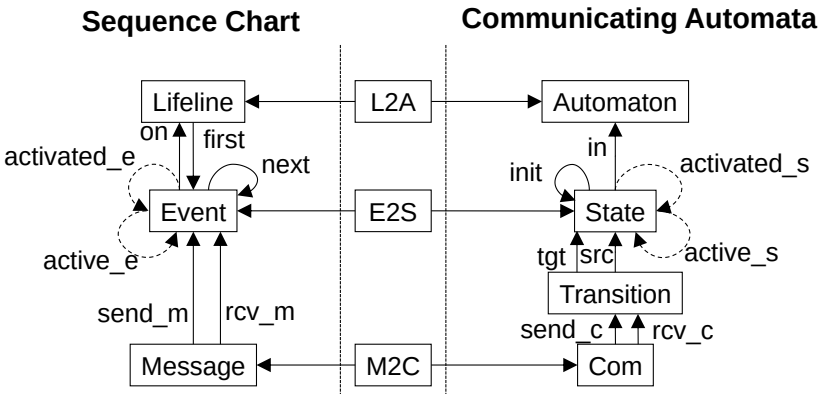


Figure 35: Type graph $S_{RT}C_{TT}T_{RT}$ for complex example (also shown in Figure 14)

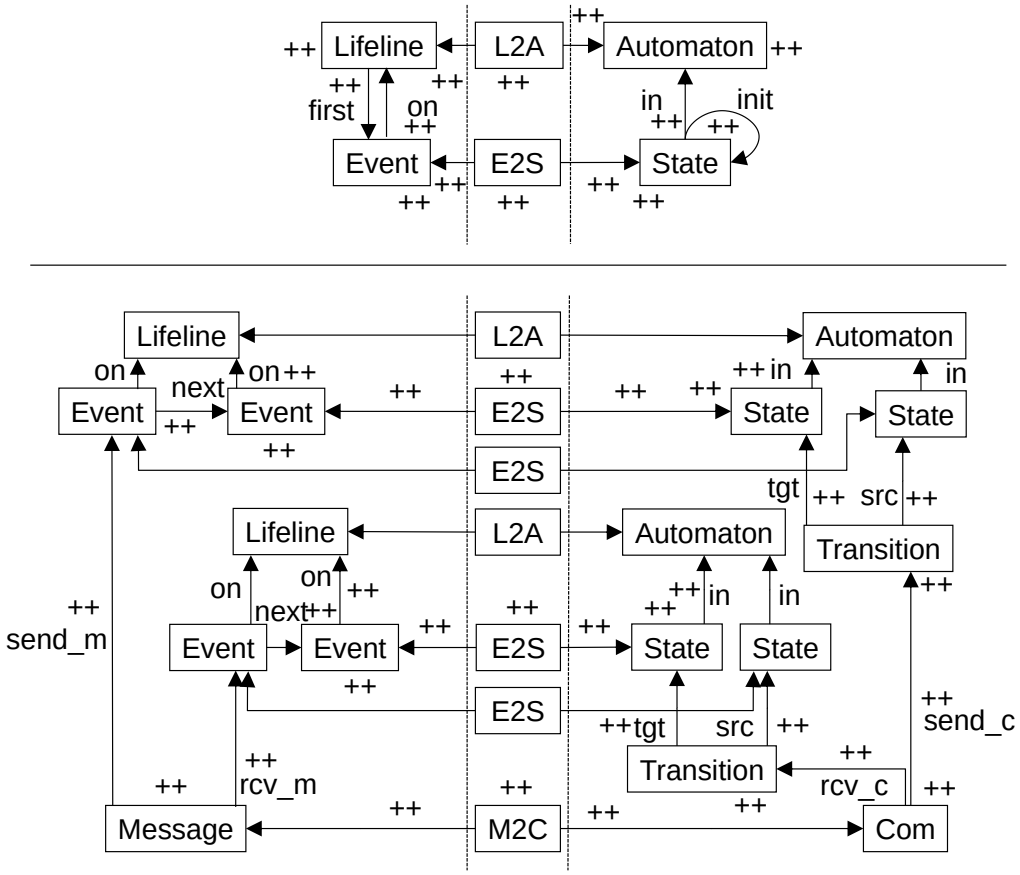


Figure 36: TGG rules $\mathcal{R}$ for complex example (also shown in Figure 16)

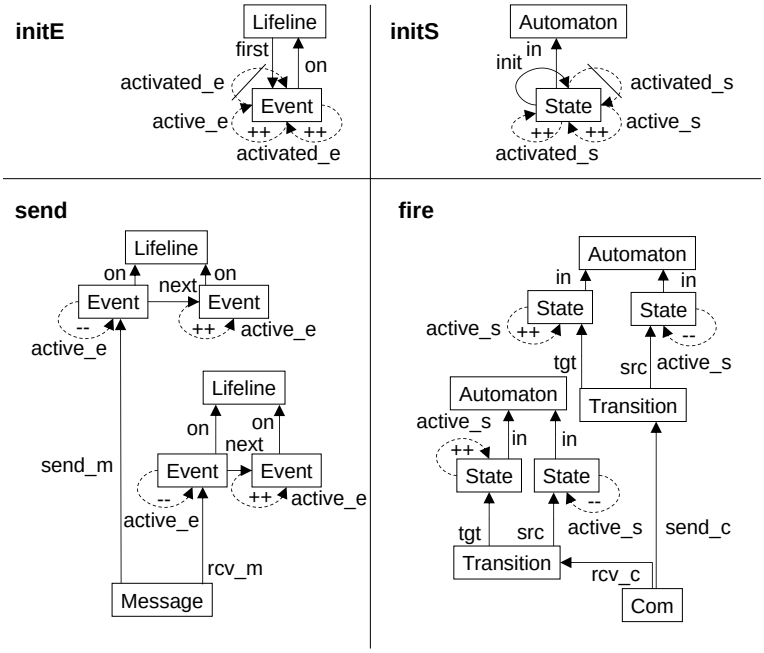


Figure 37: Operational semantics: gts_s and gts_t for complex example (also shown in Figure 15)

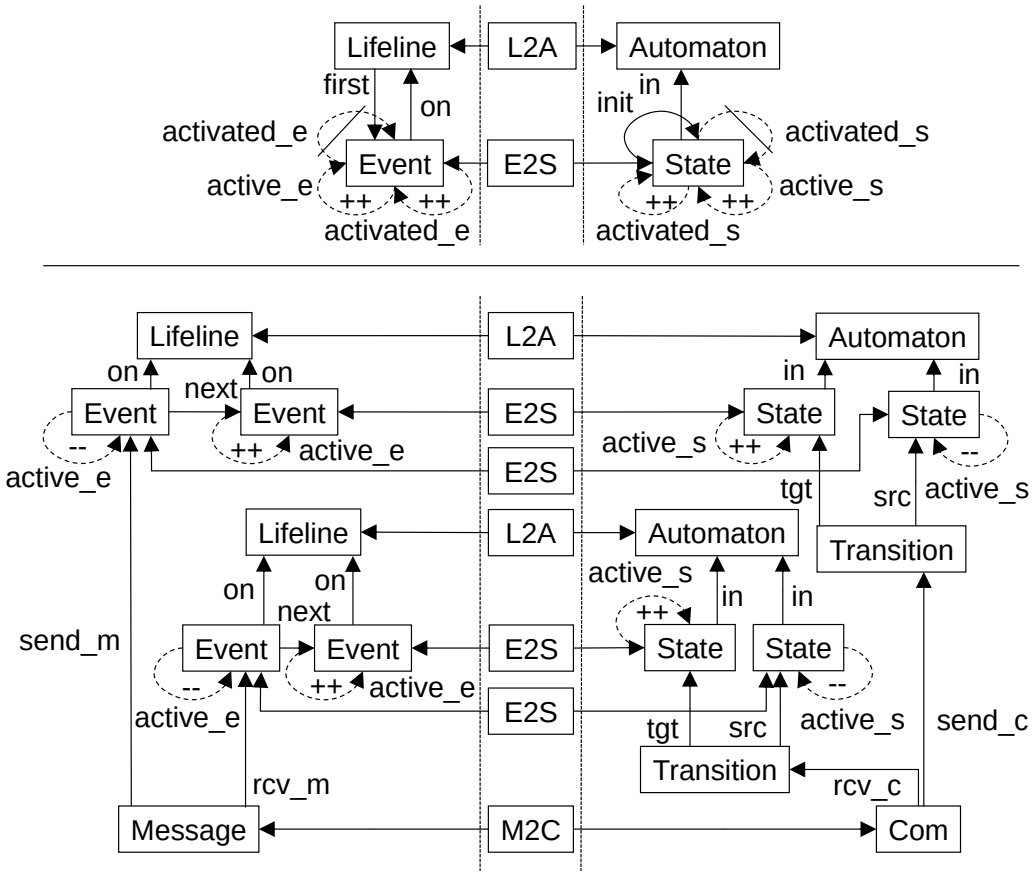


Figure 38: Pair rules $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ for complex example

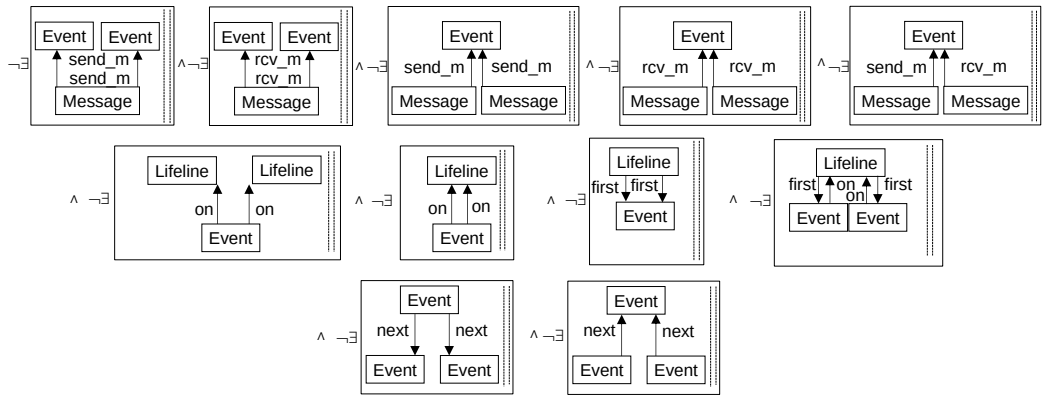


Figure 39: $\mathcal{C}_S$ – source graph constraint

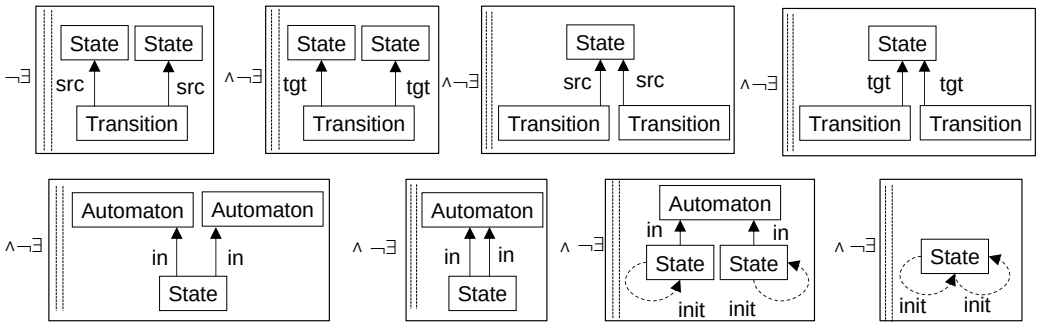


Figure 40: $\mathcal{C}_{\mathcal{T}}$ – target graph constraint

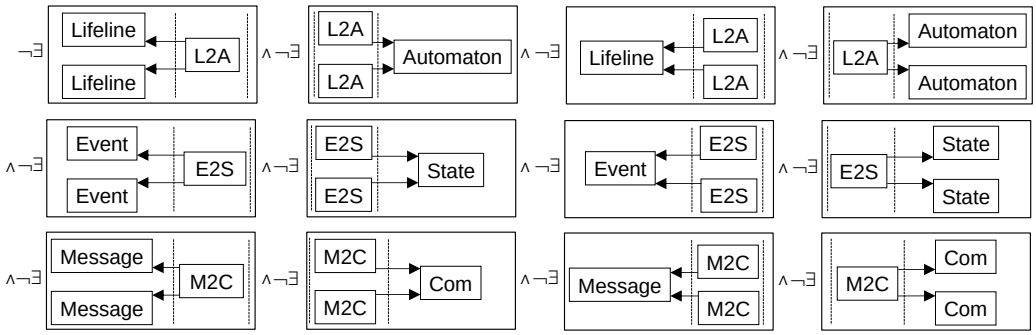


Figure 41: Fragment of $\mathcal{C}_{\text{tgg}}$

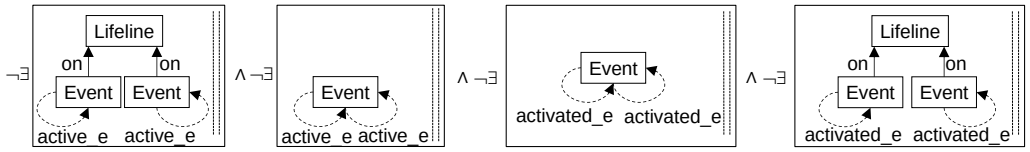


Figure 42: $\mathcal{C}_s^{gts}$

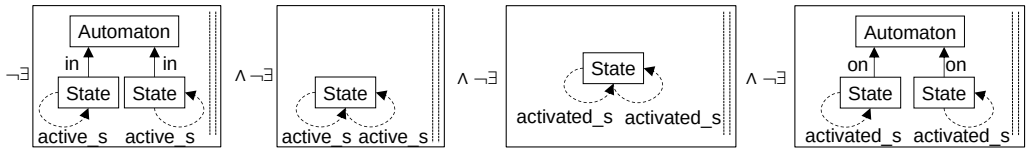
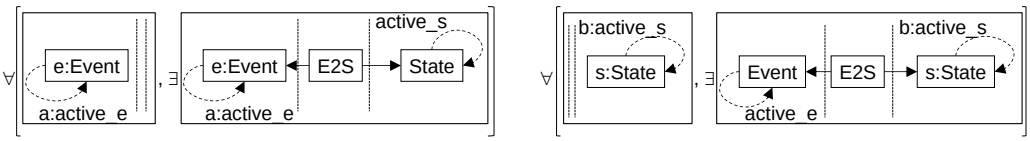
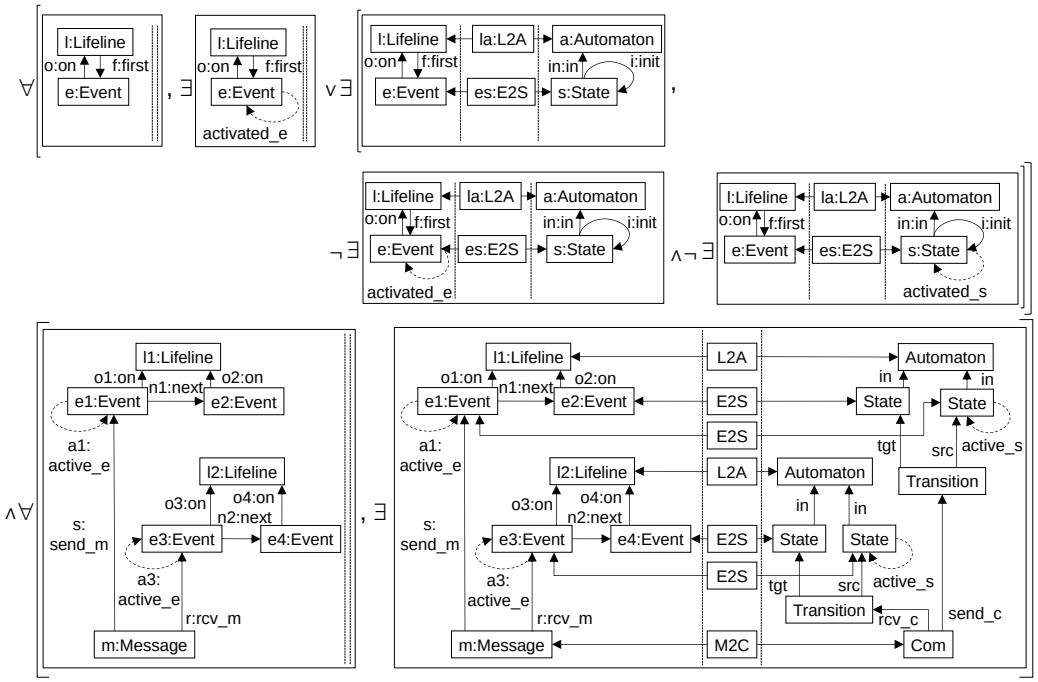


Figure 43: $\mathcal{C}_t^{gts}$

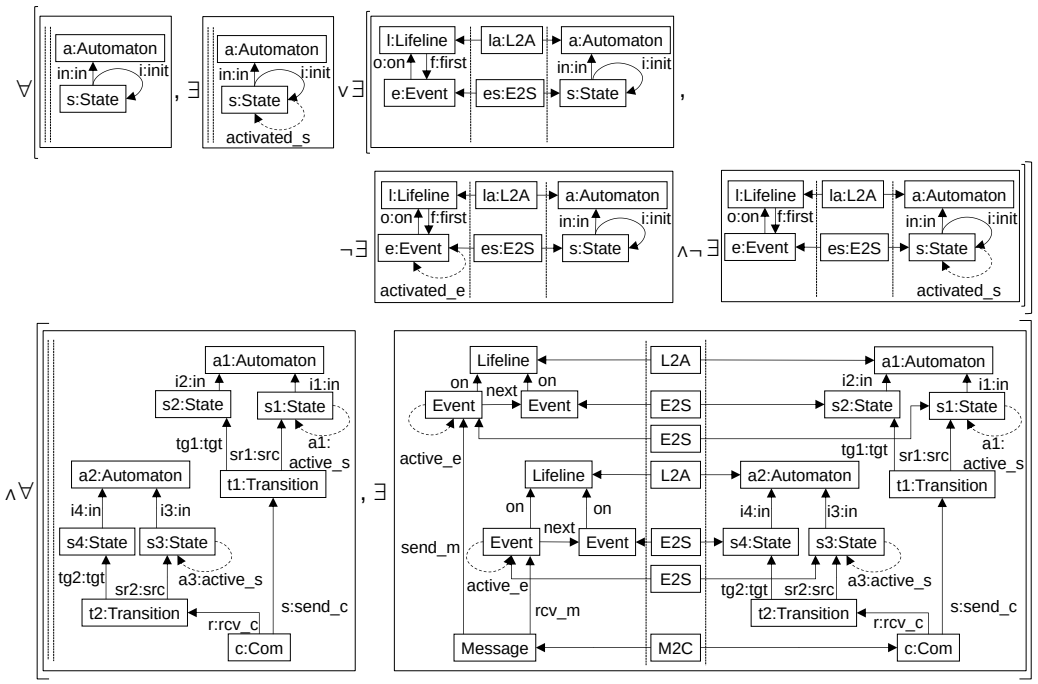


(a) $\mathcal{C}_{\mathrm{RT}}^{\mathrm{f}}$ (b) $\mathcal{C}_{\mathrm{RT}}^{\mathrm{b}}$

Figure 44: Runtime constraint $\mathcal{C}_{\mathrm{RT}} = \mathcal{C}_{\mathrm{RT}}^{\mathrm{f}} \wedge \mathcal{C}_{\mathrm{RT}}^{\mathrm{b}}$

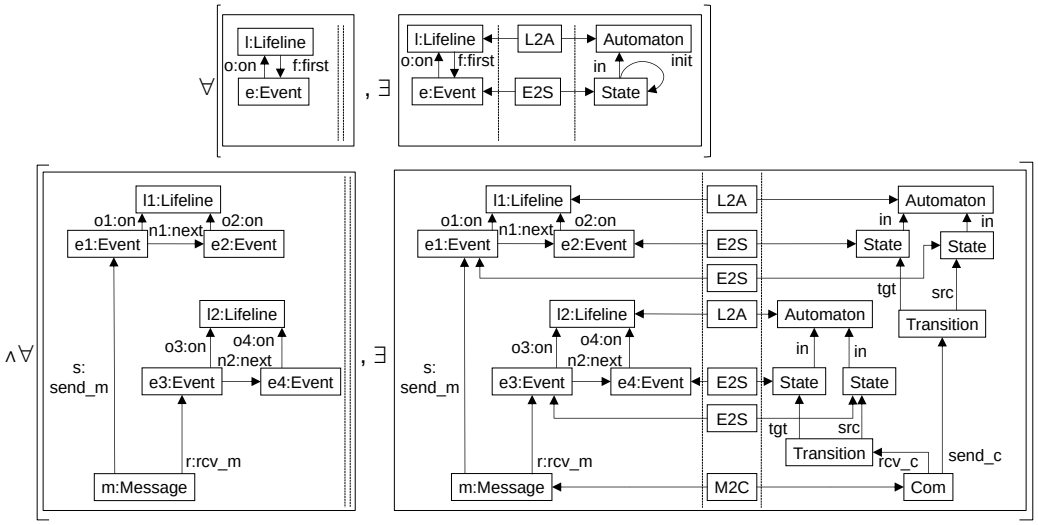


(a) $\mathcal{C}^{\text{Cor,f}}_{\text{Pair}}$

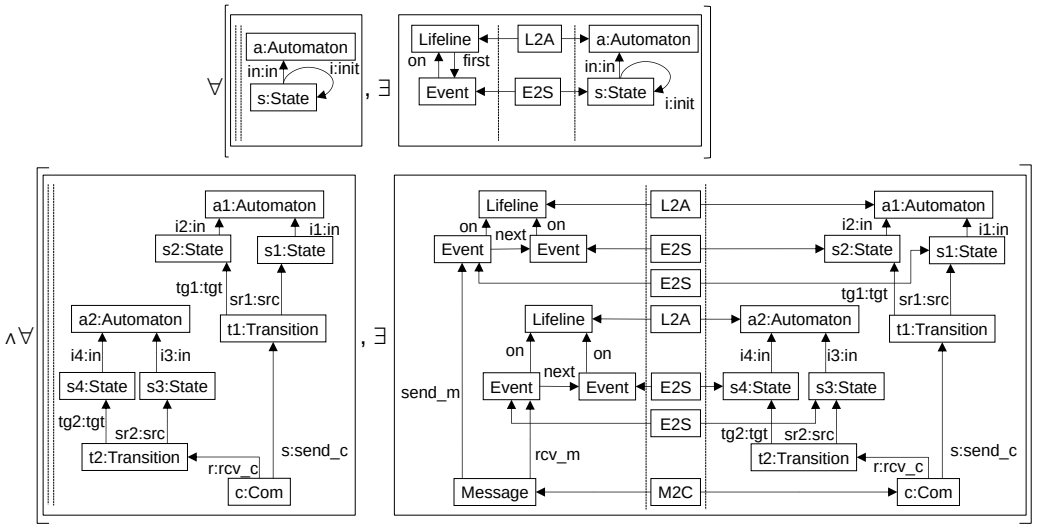


(b) $\mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$

Figure 45: Pair constraint with correspondences $\mathcal{C}^{\text{Cor}}_{\text{Pair}} = \mathcal{C}^{\text{Cor,f}}_{\text{Pair}} \wedge \mathcal{C}^{\text{Cor,b}}_{\text{Pair}}$



(a) $\mathcal{C}_{\text{MT}}^{\text{Cor,f}}$



(b) $\mathcal{C}_{\text{MT}}^{\text{Cor,b}}$

Figure 46: Model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Cor}} = \mathcal{C}_{\text{MT}}^{\text{Cor,f}} \wedge \mathcal{C}_{\text{MT}}^{\text{Cor,b}}$

B.3. All Figures of Example 25: Refinement (Tables 3, 4, 5, 8)

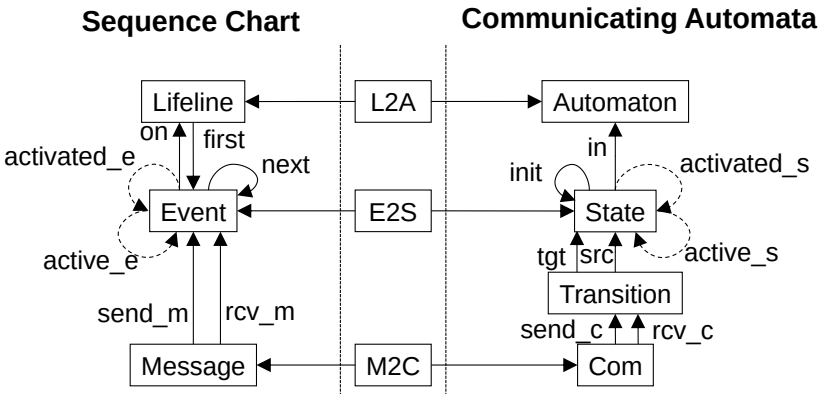


Figure 47: Type graph $S_{RT}C_{TT}T_{RT}$ for simulation example

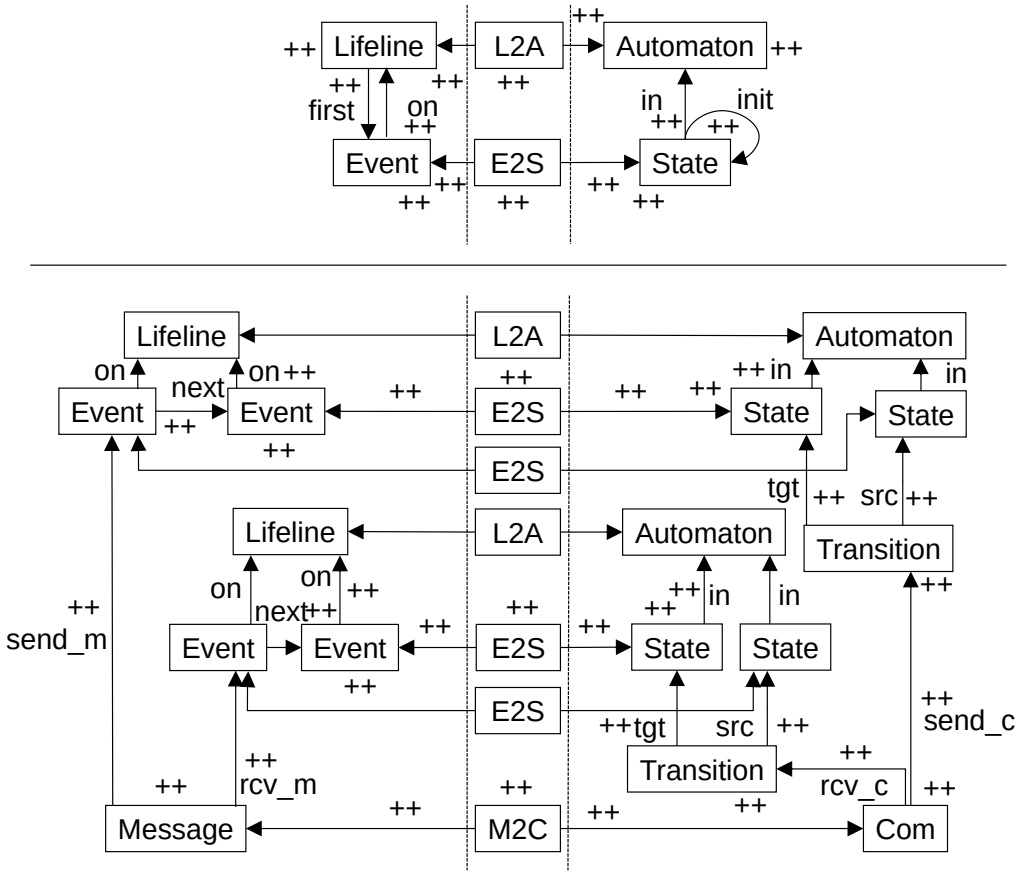


Figure 48: TGG rules $\mathcal{R}$ for simulation example, first part

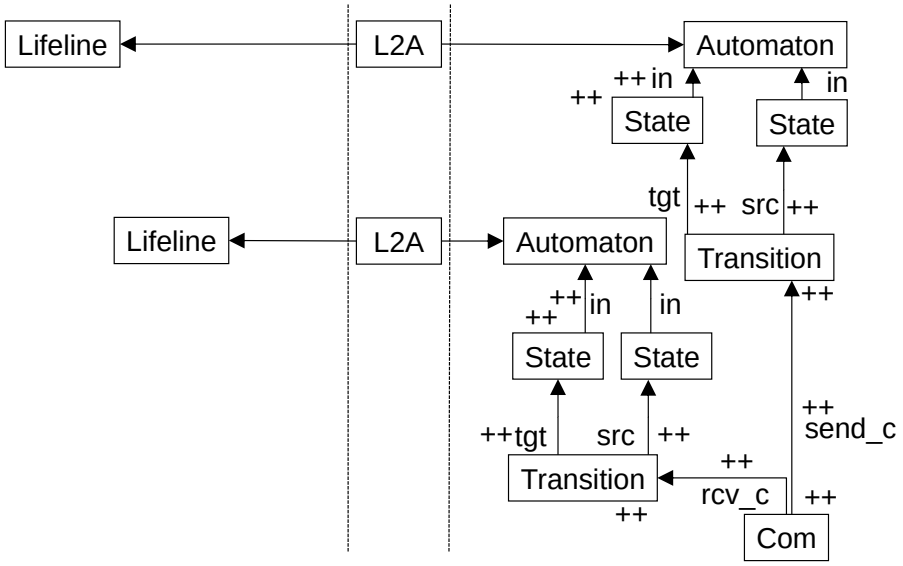


Figure 49: TGG rules $\mathcal{R}$ for simulation example, second part

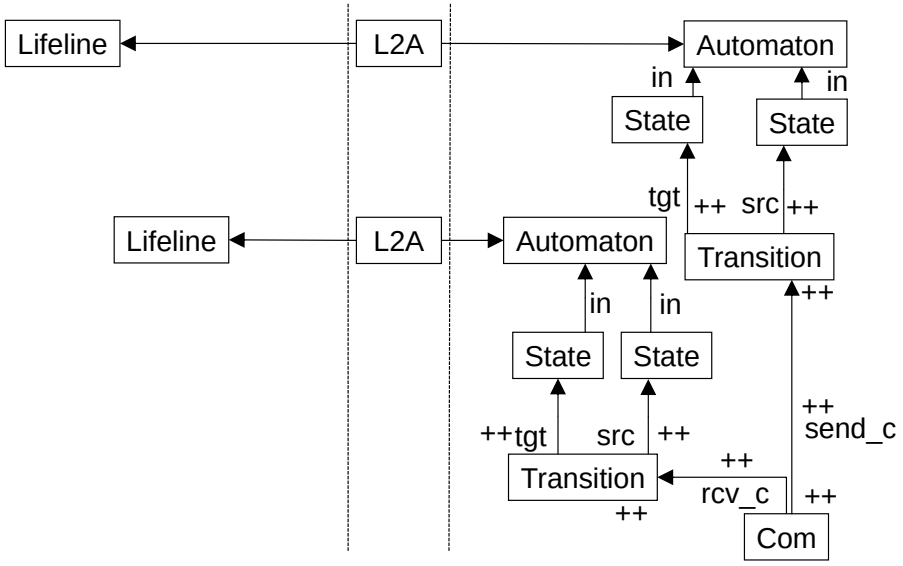


Figure 50: TGG rules $\mathcal{R}$ for simulation example, third part

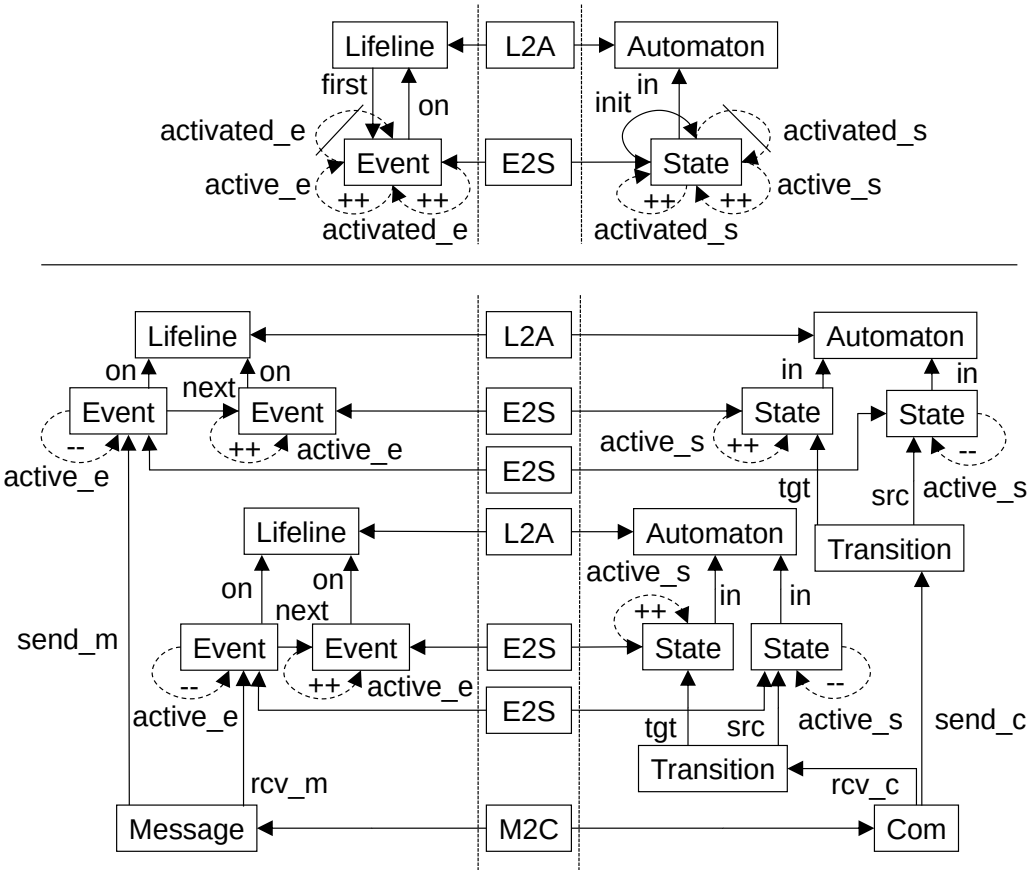


Figure 51: Pair rules $\mathcal{P}(l_s, l_t)^{\text{Cor}}$ for simulation example

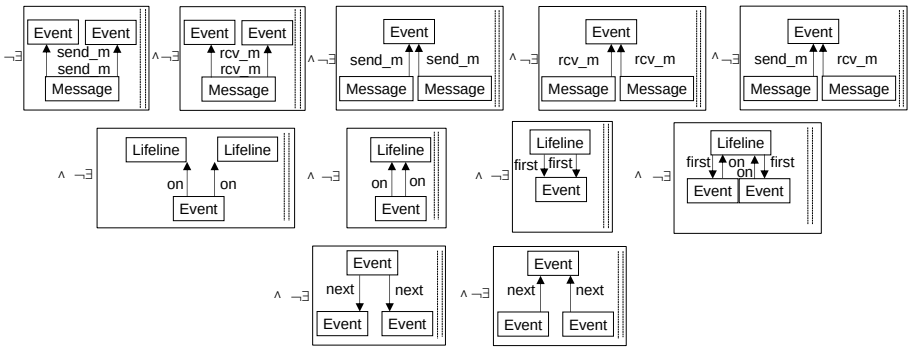


Figure 52: $\mathcal{C}_S$ – source graph constraint

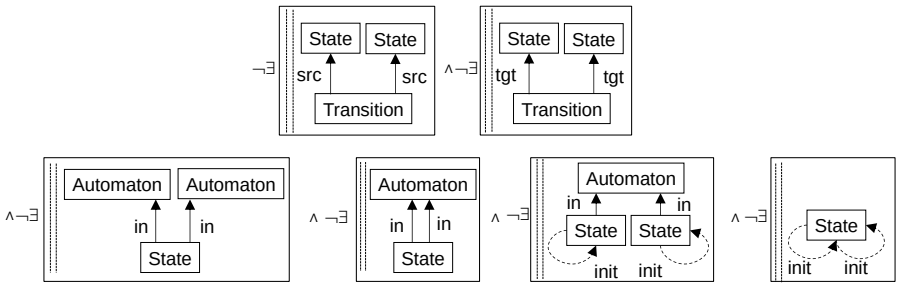


Figure 53: $\mathcal{C}_{\mathcal{T}}$ – target graph constraint

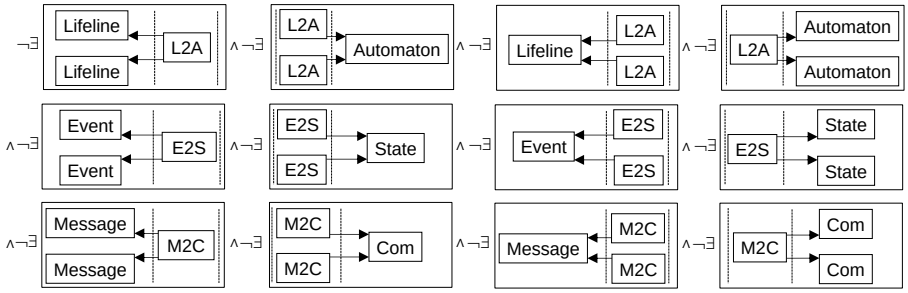


Figure 54: Fragment of $\mathcal{C}_{\text{tgg}}$

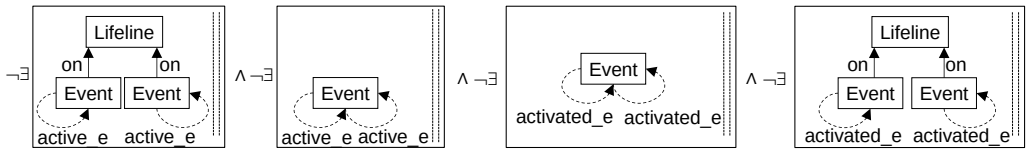


Figure 55: $\mathcal{C}_s^{gts}$

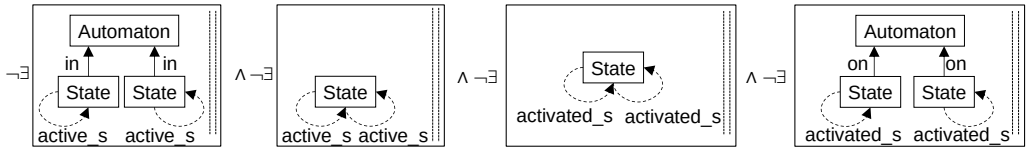


Figure 56: $\mathcal{C}_t^{gts}$

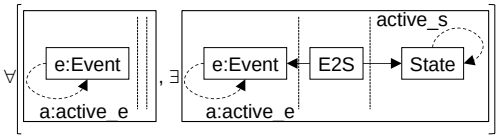


Figure 57: Forward runtime constraint $\mathcal{C}^{\mathrm{f}}_{\mathrm{RT}}$

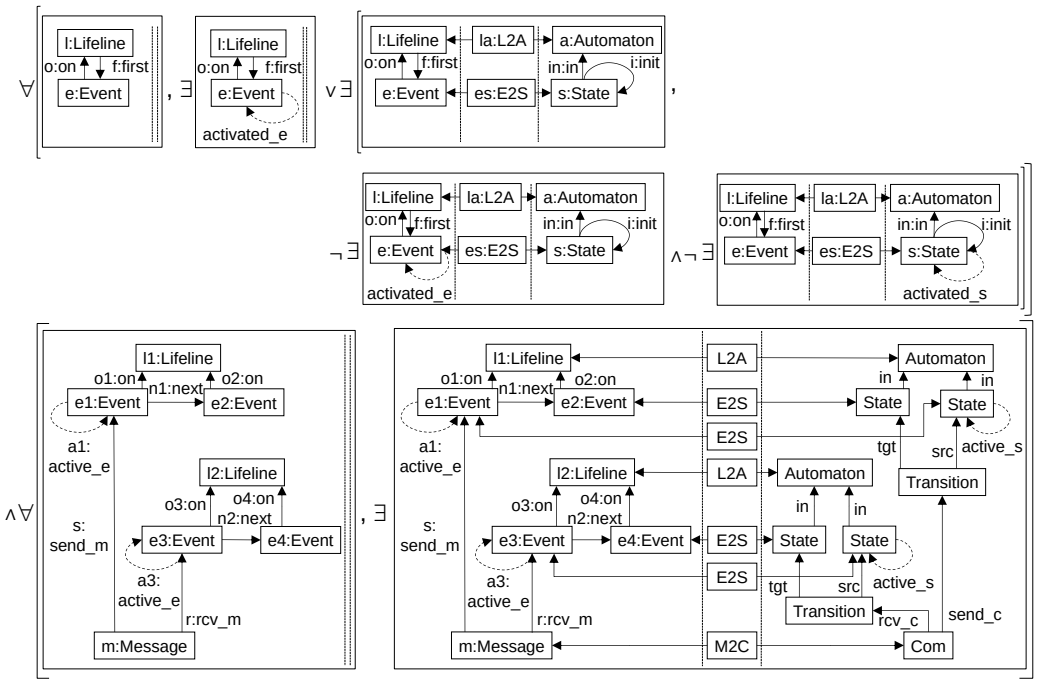


Figure 58: Forward pair constraint with correspondences $\mathcal{C}^{\mathrm{Cor,f}}_{\mathrm{Pair}}$

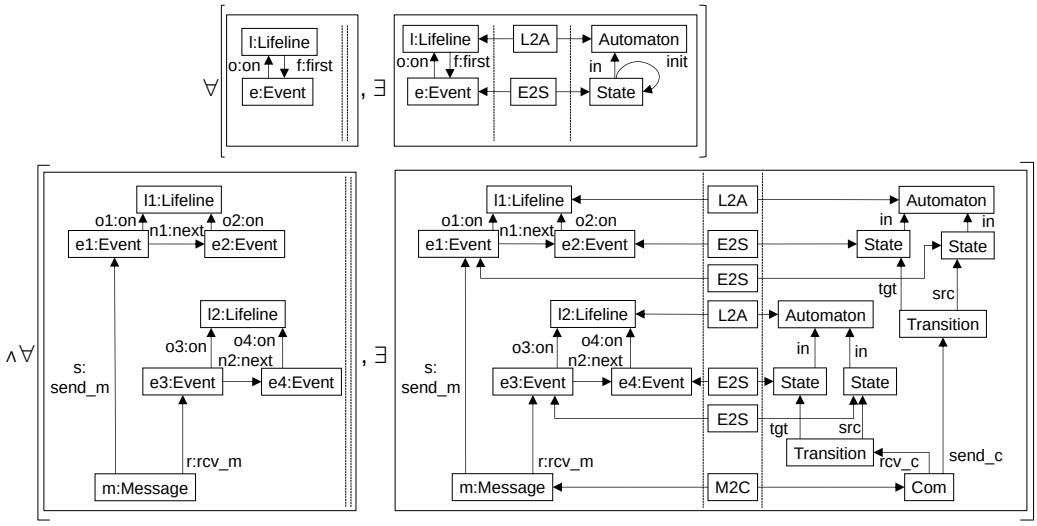


Figure 59: Forward model transformation constraint $\mathcal{C}_{\text{MT}}^{\text{Cor,f}}$

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
111	978-3-86956-390-9	**Proceedings of the 10th Ph.D. retreat of the HPI research school on service-oriented systems engineering**	Christoph Meinel, Hasso Plattner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch, Tobias Friedrich, Emmanuel Müller
110	978-3-86956-387-9	**Transmorphic : mapping direct manipulation to source code transformations**	Robin Schreiber, Robert Krahn, Daniel H. H. Ingalls, Robert Hirschfeld
109	978-3-86956-386-2	**Software-Fehlerinjektion**	Lena Feinbube, Daniel Richter, Sebastian Gerstenberg, Patrick Siegler, Angelo Haller, Andreas Polze
108	978-3-86956-377-0	**Improving Hosted Continuous Integration Services**	Christopher Weyand, Jonas Chromik, Lennard Wolf, Steffen Kötte, Konstantin Haase, Tim Felgentreff, Jens Lincke, Robert Hirschfeld
107	978-3-86956-373-2	**Extending a dynamic programming language and runtime environment with access control**	Philipp Tessenow, Tim Felgentreff, Gilad Bracha, Robert Hirschfeld
106	978-3-86956-372-5	**On the Operationalization of Graph Queries with Generalized Discrimination Networks**	Thomas Beyhl, Dominique Blouin, Holger Giese, Leen Lambers
105	978-3-86956-360-2	**Proceedings of the Third HPI Cloud Symposium "Operating the Cloud" 2015**	Estee van der Walt, Jan Lindemann, Max Plauth, David Bartok (Hrsg.)
104	978-3-86956-355-8	**Tracing Algorithmic Primitives in RSqueak/VM**	Lars Wassermann, Tim Felgentreff, Tobias Pape, Carl Friedrich Bolz, Robert Hirschfeld
103	978-3-86956-348-0	**Babelsberg/RML : executable semantics and language testing with RML**	Tim Felgentreff, Robert Hirschfeld, Todd Millstein, Alan Borning
102	978-3-86956-347-3	**Proceedings of the Master Seminar on Event Processing Systems for Business Process Management Systems**	Anne Baumgraß, Andreas Meyer, Mathias Weske (Hrsg.)

ISBN 978-3-86956-391-6
ISSN 1613-5652