Software & Systems Modeling (2019) 18:2937–2972
https://doi.org/10.1007/s10270-018-00706-9

REGULAR PAPER

Automatic verification of behavior preservation at the transformation level for relational model transformation

Johannes Dyck[1] · **Holger Giese**[1] · **Leen Lambers**[1]

Received: 28 July 2017 / Revised: 26 November 2018 / Accepted: 4 December 2018 / Published online: 22 December 2018

Abstract
The correctness of model transformations is a crucial element for model-driven engineering of high-quality software. In particular, behavior preservation is an important correctness property avoiding the introduction of semantic errors during the model-driven engineering process. Behavior preservation verification techniques show some kind of behavioral equivalence or refinement between source and target model of the transformation. Automatic tool support is available for verifying behavior preservation at the instance level, i.e., for a given source and target model specified by the model transformation. However, until now there is no sound and automatic verification approach available at the transformation level, i.e., for all source and target models. In this article, we extend our results presented in earlier work (Giese and Lambers, in: Ehrig et al (eds) Graph transformations, Springer, Berlin, 2012) and outline a new transformation-level approach for the sound and automatic verification of behavior preservation captured by bisimulation resp. simulation for outplace model transformations specified by triple graph grammars and semantic definitions given by graph transformation rules. In particular, we first show how behavior preservation can be modeled in a symbolic manner at the transformation level and then describe that transformation-level verification of behavior preservation can be reduced to invariant checking of suitable conditions for graph transformations. We demonstrate that the resulting checking problem can be addressed by our own invariant checker for an example of a transformation between sequence charts and communicating automata.

Keywords Relational model transformation · Formal verification of behavior preservation · Behavioral equivalence and refinement · Bisimulation and simulation · Graph transformation · Triple graph grammars · Invariant checking

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, *behavior preservation* in particular is an 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 [43]) between source and target model of the transformation. While other notions for correctness exist and are applicable to different model transformations and scenarios, this article focuses on behavioral equivalence and refinement.

Since we use a number of formalizations in our approach, a certain familiarity with the concepts of graph transformation systems (GTSs), triple graph grammars (TGGs), nested

Communicated by Mr. Juan de Lara.

This work was developed mainly in the course of the projects Correct Model Transformations II (GI 765/1-2) and III (GI 765/1-3) funded by the Deutsche Forschungsgemeinschaft. See https://hpi.de/en/giese/research/projects/cormorant.html.

✉ Johannes Dyck
johannes.dyck@hpi.de

Holger Giese
holger.giese@hpi.de

Leen Lambers
leen.lambers@hpi.de

[1] Hasso Plattner Institute, University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

graph conditions, and bisimulation is recommended to understand this article, although we have provided definitions for all required concepts. With knowledge of those concepts, we hope that our approach and contribution will be useful to both developers and users of model transformations intended to be behavior preserving.

Verification of behavior preservation has been presented with automatic tool support for the instance level [19,44,45], i.e., for a given source and target model specified by the model transformation. Nevertheless, until now there is no sound and fully automatic verification approach available at the transformation level such that its results cover *all* source and target models specified by the model transformation. Since development and application of the transformation usually happen in separate stages and are performed by different people or different organizations, finding errors in the transformation during the application stage (instance level) is usually too late.

Consequently, ensuring behavior preservation for the transformation during the development of the transformation (transformation level) is highly desirable, but to our knowledge so far no work exists that promises to solve the related verification problem in a sound and automatic manner. We presented a sound approach [22] addressing this problem on the transformation level in a semi-automatic manner in form of a verification technique based on interactive theorem proving. Also, Hülsbusch et al. [35] presented and compared different sound proof strategies for manual proofs on the transformation level without solving the problem of automation. Some sound approaches to automating the verification of behavior preservation for the endogenous case of model refactorings exist [7,50], but none of them covers the case of outplace model transformations, being able to specify also exogenous model transformations.

In this article, we present a first sound and automatic approach for the verification of behavior preservation for the case of outplace model transformations—where, specifically, model transformations are specified by triple graph grammars (TGGs) and semantics are defined by graph transformation systems (GTSs). In particular, we show how to encode the behavior preservation problem symbolically at the transformation level (in a so-called modeling scheme) and then present how to verify it automatically by reducing it to invariant checking for GTSs (presented in our verification scheme). We show that in restricted cases the invariant checking can be performed automatically using our existing invariant checking technique for GTSs [4]. We used a similar basic idea for verifying consistency preservation of refactorings by reducing it to an invariant checking problem for GTSs accordingly [5]. Due to a mapping of TGGs on specially typed graph transformations [23,28], both the transformation and the semantics are captured in a homogeneous manner, which facilitates mapping the problem to invariants for GTSs. We demonstrate our approach with the example of a model transformation from sequence charts to communicating automata.

Although we address tooling and the potential for automation, our focus lies more on the approach itself and less on a particular tool or toolchain integrating the different steps of our modeling and verification schemes.

This article extends our earlier work and first approach [24] in several directions: First, we support not only behavioral equivalence in form of bisimulation, but also behavioral refinement in form of simulation. 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. Thirdly, we have applied our approach (Sect. 5.2) to an example extending our example in earlier work [24] in order to demonstrate the aforementioned features, i.e., non-determinism and simulation. Lastly, the required amount of manual specification efforts for the modeling of the behavior preservation problem in a symbolic manner has been reduced (see the related discussion in Sect. 5).

Our sound and automatic approach for operational model transformations [14]—as opposed to relational model transformations—only applies the limited concepts developed in our first approach [24] to establish bisimulation via invariant checking on the transformation level. Only the first phase of the related verification scheme, which establishes the required invariant for each pair of source and target model of an operational transformation, is different from the basic approach [24]. Therefore, the approach can be seen as orthogonal and includes the same limitations. Accordingly, the approach could be improved by combining it with the improvements in the modeling and verification scheme as presented in this article.

1.1 Behavior preservation at the transformation level

To introduce the considered notion of behavior preservation for model transformations at the transformation level, we clarify in the following which artifacts are necessary to describe the problem such that we can later (Sect. 2) refine and formalize these artifacts as a basis for tackling the corresponding verification problem.

Example 1 (Running example) In order to demonstrate our verification technique for behavior preservation, we will consider a model transformation between sequence charts and communicating automata as our running example. When considering how to formalize (and later verify) behavior preservation with respect to that specific example, we require formalizations capable of describing

1. sequence charts and communicating automata, i.e., *modeling languages* for source and target models,
2. the behavior of sequence charts and communicating automata, i.e., *model semantics* for source and target models,
3. how sequence charts should be transformed to communicating automata, i.e., *model transformations* between source and target models,
4. how behavior of sequence charts maps to behavior of communicating automata (and vice versa), which is established via *semantic remappings*, and
5. when a sequence chart is or is not behaviorally equivalent to a system of communicating automata, i.e., how we define *behavioral equivalence* (and *behavioral refinement*).

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* $(M_s, M_t) \in MT$) Given a source and target modeling language $\mathcal{L}_S$ and $\mathcal{L}_T$, respectively, a *model transformation MT* is a relation over $\mathcal{L}_S \times \mathcal{L}_T$. Each pair (M_s, M_t) of source and target models in MT is a *model transformation instance* of the model transformation MT.

In the case of an outplace model transformation, the source and target modeling languages $\mathcal{L}_S$ and $\mathcal{L}_T$ can be different (in this case the model transformation is exogenous), while in the endogenous case the source and target modeling languages $\mathcal{L}_S$ and $\mathcal{L}_T$ must be the same.

A model transformation can, for example, be defined by operational model transformation techniques [11,21,36,47,55], relational model transformation techniques [41,47,51], or hybrid model transformation techniques [6,47]. Such a definition for a model transformation then implicitly defines all model transformation instances. Besides unidirectional model transformations that only support to derive a target model for a given source model, there are also bidirectional model transformations that also support the inverse direction [53]. In this article, we consider the general case of outplace model transformation, focus on relational model transformation techniques, and do not assume that the model transformations are unidirectional or bidirectional. 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 behavioral equivalence and refinement [26,27].

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 notion of behavior preservation for model transformation at the transformation level outlined in the following, 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 $MT \subseteq \mathcal{L}_S \times \mathcal{L}_T$, semantic mappings $sem_S : \mathcal{L}_S \to \mathcal{D}_S$ and $sem_T : \mathcal{L}_T \to \mathcal{D}_T$ for source and target language $\mathcal{L}_S$ and $\mathcal{L}_T$, and semantic remappings $l_s : \mathcal{D}_S \to \mathcal{D}$ and $l_t : \mathcal{D}_T \to \mathcal{D}$, we say that

1. the *model transformation MT* is *behavior preserving in an equivalent manner* if for each pair of source and target models $(M_s, M_t) \in MT$, it holds that $l_s(sem_S(M_s)) =_{\mathcal{D}} l_t(sem_T(M_t))$.
2. the *model transformation MT* is *behavior preserving in a refining manner* if for each pair of source and target models $(M_s, M_t) \in MT$, it holds that $l_t(sem_T(M_t)) \leq_{\mathcal{D}} l_s(sem_S(M_s))$.

Remark 2 (Behavior Preservation—Instance Level) Behavior preservation at the transformation level applies to the *transformation as a whole*, which, in general, consists of infinitely many model transformation instances. For behavior

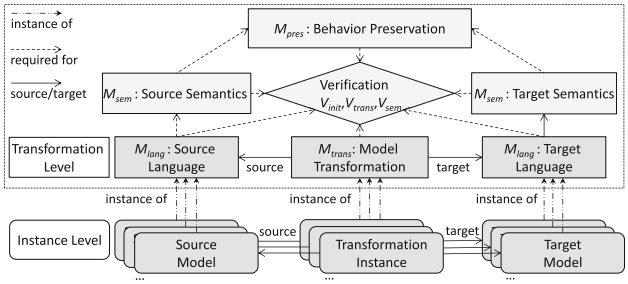


Fig. 1 Behavior preservation—transformation level (see Definition 6)

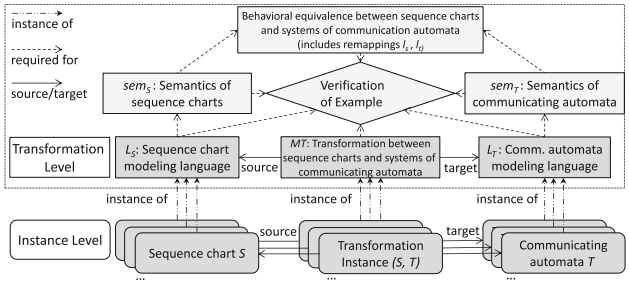


Fig. 2 Elements of the running example mapped to the abstract overview

preservation at the instance level, it is enough that condition 1 or 2 holds for *just one* such pair of source and target models $(M_s, M_t) \in MT$. Then, 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 is 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 $l_s(sem_S(M_s)) \preceq_{\mathcal{D}} l_t(sem_T(M_t))$.

Figure 1 summarizes the general concepts from Definition 6 also enumerated in the first row of Table 1, which need to be further refined and formalized in corresponding modeling steps $M_{\square}$ (together they form our so-called modeling scheme). Based on this modeling of behavior preservation at the transformation level in a symbolic manner, we will be able to develop a corresponding verification scheme on the transformation level. This verification scheme, which will be explained later, consists of multiple verification steps $V_{\square}$. In Figs. 1 and 2, rounded rectangles depict elements located at the instance level (here: models and transformation instances) and rectangles depict elements that are part of the modeling scheme and located at the transformation level. The verification scheme is depicted by a rhombus. Colors classify elements as belonging to model semantics (upper part, lighter) or not (lower part, darker).

Example 2 (Running example) In Fig. 2, we show the elements of our running example (introduced in Example 1) mapped to our abstract overview. As explained above, our example considers a model transformation (MT) between sequence charts as the source modeling language ($\mathcal{L}_S$) and communicating automata as the target modeling language ($\mathcal{L}_S$). We will later extend this example by specific definitions for source and target languages $\mathcal{L}_S$ and $\mathcal{L}_T$ (modeling step M_{lang}), for semantic mappings sem_S and sem_T (modeling step M_{sem}), for a model transformation $MT \subseteq \mathcal{L}_S \times \mathcal{L}_T$ (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. Since this article approaches behavior preservation for outplace model transformations, we will further limit our discussion mainly to related work for this problem and this kind of transformations.

We can target the verification of behavior preservation for outplace model transformations at the instance level or at the transformation level (see Definition 6 and Remark 2). While in the former, we study the problem for a pair of a source and a target model, in the latter we consider at once all potentially infinitely many source and target models that are linked by the transformation.

The techniques for verification of behavior preservation for model transformation instances [19,44,45] assign the

Table 1 Generic concepts from Definition 6 and corresponding modeling steps as well as their refinement

Concept (modeling step)	Modeling languages (M_{lang})	Model semantics (M_{sem})	Model transformation (M_{trans})	Behavior preservation (M_{pres})	
				Equivalence	Refinement
Definition 6	$\mathcal{L}_S, \mathcal{L}_T$	$sem_S(\cdot), sem_T(\cdot)$	MT	$l_s(.) =_{\mathcal{D}} l_t(.)$	$l_t(.) \preceq_{\mathcal{D}} l_s(.)$
Definition 25	$\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T)$	$LTS(gts_s, \cdot), LTS(gts_t, \cdot)$	$MT(tgg, \mathcal{C}_{tgg})$	$l_s(.) =_{bsim} l_t(.)$	$l_t(.) \preceq_{sim} l_s(.)$
Section 3				$\mathcal{C}_{Bis}^{Cor}$	$\mathcal{C}_{Sim}^{Cor,b}$

source and target model a formal semantics and then proof semantic equivalence via bisimulation checking.

The formal modeling and verification of properties at the transformation level (see 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.

Barroca et al. [3] describe how the typical instance-level approaches are extended by checking bisimulation not only for a single pair of a source and target model, but for all input and related output models for a symbolically encoded finite number of combinations of possibly overlapping transformation rules. However, like the work by Lúcio et al. [42] not only a single such step or sequences of a bounded number of steps but sequences of arbitrary number of steps have to be considered and thus the approach is unsound. It can only find errors if they are present within the considered sequences of bounded number of steps. It cannot exclude errors.

For the verification of behavior preservation for model transformations at the transformation level (see Definition 6), only rudimentary sound approaches exist: In our earlier work [22], 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 Sect. 1.1 are encoded in the logic of the theorem prover. Hülsbusch et al. [35] suggest different proof strategies for the verification of model transformations at the model transformation level, but these strategies do not solve the question how to automate the verification. However, our approach will make use of a number of concepts described in their work, including distinguishing static and runtime elements (the latter involving model semantics), semantic (re)mappings to labeled transition systems, and bisimulation for labeled transitions systems.

Some initial ideas for the automation of behavior preservation at the transformation level for endogenous model transformations in form of in-place refactorings exist [7,50]; however, these ideas do not cover the case of outplace (and therefore also exogenous) model transformations that is studied in this article. Furthermore, there is work that presents behavior preservation verification techniques for model transformation instances concentrating on checking the preservation of specific behavioral properties [1,40,57,58] as opposed to checking some kind of bisimulation, which is more general and complex.

The review of the related work as depicted in Table 2 shows that for the verification of behavior preservation for outplace model transformations at the transformation level no automatic and sound verification approach besides our own work for relational model transformations [24] as well as operational model transformations [14] exists so far. These transformation-level approaches are, however, still rather limited compared to the solution for relational model transformations presented in this article as outlined in the introduction.

Table 2 Summary of automatic verification approaches for behavior preservation of outplace model transformations

Instance level	Transformation level	
Sound	Unsound	Sound
[19,44,45]	[3]	([14,24][a])

[a] These two approaches are predecessors of the approach presented in this article

1.3 Outline

The rest of the article is structured as follows: In Sect. 2, we introduce the basic formal notions that we require to tackle the description of the behavior preservation problem on the transformation level. In particular, for formalizing modeling languages (modeling step M_{lang}), the behavioral semantics of the modeling languages (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}). M_{pres} is further specified in its own section—Sect. 3—for both behavioral equivalence and refinement. Together, modeling steps M_{lang}, M_{sem}, M_{trans}, and M_{pres} constitute a so-called *modeling scheme* such that when this scheme is complete the notion of behavior preservation at the transformation level is formalized relying on a symbolic encoding allowing us to develop a corresponding *verification scheme*. In Sect. 4, we present such a verification scheme, which consists of the three steps V_{init}, V_{trans}, and V_{sem}. In particular, we describe how the problem of verifying behavior preservation for relational model transformation at the transformation level can be reduced to invariant checking GTSs and prove its correctness. This extends to both behavioral equivalence and behavioral refinement. Afterward, automation and evaluation for the two introduced variants of our problem are discussed in Sect. 5. In Sect. 6, we discuss the appropriateness, applicability, and limitations concerning the automation of the approach. The article closes with a conclusion and outlook on future work. All details and examples omitted in this article can be found in a technical report [13].

2 Formalization

In order to formalize the artifacts modeling language (modeling step M_{lang}, Sect. 2.1), model semantics (modeling step

M_{sem}, Sect. 2.2), and model transformation (modeling step M_{trans}, Sect. 2.3) of Definition 6, we reintroduce graph conditions, graph transformation and TGGs in a compact way and refer to the literature [16,28,32] for more detailed definitions and explanations. We also reintroduce the notion of labeled transition systems, relabelings of labeled transition systems, and bisimulation/simulation in order to formalize the semantic domain and the notion of behavioral equivalence and refinement (modeling step M_{pres}, Sect. 2.4). In particular, Sect. 2.4 summarizes how to refine Definition 6 with these formal concepts.

Notation 1 *The general principles behind our notation are as follows: We always denote simple graphs with capitals. Triple graphs will be denoted as $S_G C_G T_G$, to make the source S_G, correspondence C_G, and target component T_G of the triple graph explicit. If we refer to a language, we use $\mathcal{L}$. We use $\mathcal{C}$ to refer to specific conjunctions of constraints used to constrain a language or to define invariants to be checked for behavior preservation (with indices to make distinctions between the different types of constraints). We use LTS as well as MT to derive labeled transition systems or model transformations from graph transformation systems (denoted gts) or triple graph grammars (denoted tgg) with constraints. We use S and T to denote source and target models, respectively. We use indices S or T if we refer to artifacts related to source or target models—and indices s and t if they belong to identifiers in small letters, such as gts_s or gts_t.*

2.1 Modeling language

To formalize the modeling language according to Definition 1, we will assume in this article that each model of a modeling language is represented by a graph. Graphs can be *typed* over a given type graph *TG* as usual [15] by adding a 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 is being mapped.

In the following, we formally define the notions of graphs, graph morphisms, and type graphs: A *graph* $G = (V, E, s, t)$ consists 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. 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 $f_V \circ t_1 = t_2 \circ f_E$. 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. This is similar to a language defined by a metamodel where the language consists of the metamodel's instances. Note that we omit attributes in our formal framework and we refer to Sect. 6 for a discussion with respect to the extension of our framework for cases with attributes.

Definition 7 (*Graph language*) Given a type graph *TG*, the *graph language* $\mathcal{L}(TG)$ denotes the set of all graphs typed over TG, i.e., $\mathcal{L}(TG) = \{(G, type) \mid type : G \to TG\}$.

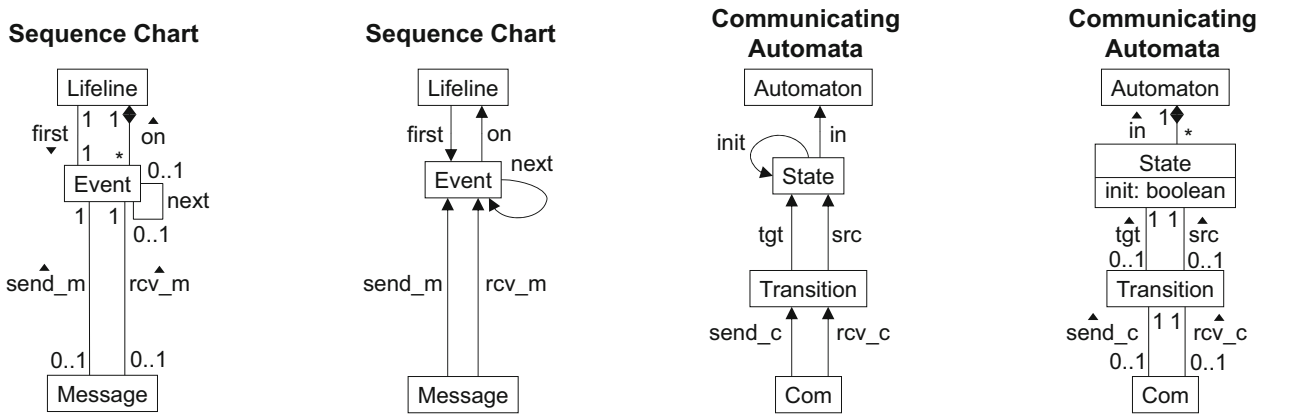
In the following, all graphs encountered in our examples will be typed graphs, unless noted otherwise, and the typing morphism will be omitted. Likewise, all specific graph morphisms in our examples will be typed graph morphisms. However, for certain concepts, typing—specifically, the type graph the graphs are typed over—plays an important role and will still appear in the respective definitions.

Example 3 (Graph language as modeling language) Figure 3a depicts a UML class diagram serving as a metamodel for our source modeling language introduced in Example 2—the language of message sequence charts able to send and receive messages. We also need additional constraints not shown here; for instance, an event cannot correspond to both receiving and sending the same message.

Figure 3b depicts the metamodel's implementation as a type graph S_{TT}. The graph S in Fig. 4a then describes a sequence chart, which is also depicted in concrete syntax on the left. The chart has two lifelines with two events each: one happens when the message *ms* is sent/received, the other afterward. In general, events occur either as part of sending or receiving messages or to denote the end of a lifeline; the latter is the case for events $e3$ and $e4$ here. The graph S is typed over S_{TT} via a typing morphism mapping nodes and edges with a specific label in S to a node and edge type in S_{TT} with the same label, respectively.

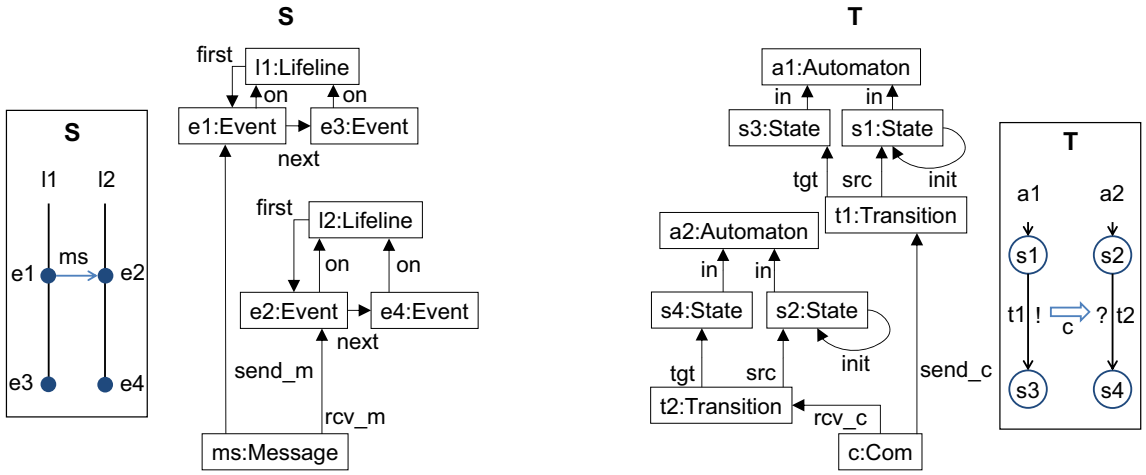
However, the language $\mathcal{L}(S_{TT})$ of all sequence charts still contains malformed models because no cardinality constraints or other restrictions are included in the type graph. For example, charts could have more than one event attached to receiving the same message or events could be placed on multiple lifelines. Instead, we only want to allow charts where there is at most one event corresponding to receiving a message and at most one corresponding to sending it—and one event should not belong to more than one lifeline. We can implement these cardinality restrictions as graph constraints, which will be explained below.

Similar to our source modeling language, Fig. 3d depicts a metamodel for systems of communicating automata and



(a) Source metamodel **(b)** Type graph S_{TT} **(c)** Type graph T_{TT} **(d)** Target metamodel

Fig. 3 Source and target meta-models and type graphs S_{TT} and T_{TT}



(a) S in concrete and abstract syntax **(b)** T in concrete and abstract syntax

Fig. 4 Example graphs S and T with $S \in \mathcal{L}(S_{TT})$ and $T \in \mathcal{L}(T_{TT})$ in concrete and abstract syntax

Fig. 3c shows the corresponding type graph (again, without cardinalities). The graph T in Fig. 4b, which is typed over T_{TT} describes a system of two communicating automata with two states and a transition for each automaton. The language of communicating automata is the target modeling language for the example model transformation; again, we still need to implement cardinality restrictions.

A *modeling language* could be defined as $\mathcal{L}(TG)$ for a given type graph TG, but as shown in Example 3 we usually need means 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 an appropriate formalism to further constrain $\mathcal{L}(TG)$. This is demonstrated in related work, where a formal relation has been established between the notions of meta-models with OCL invariants and type graphs with graph constraints for language definition [2].

Graph conditions [16,32] generalize corresponding earlier notions [31], where a negative application condition (NAC), over a graph P is defined in terms of a graph morphism $a : P \to C$ and denoted $\neg\exists a$. Informally, a morphism $p : P \to G$ satisfies $\neg\exists a$ if there does not exist a morphism $q : C \to G$ extending p. Positive application conditions (PACs), i.e., $\exists a$, follow the same scheme. Then, a *(nested) graph condition* AC is either the special condition *true* or a pair of the form $\neg\exists(a, ac_C)$ or $\exists(a, ac_C)$, 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 $\exists(a, ac_C)$ 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 $p : P \to G$ 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 8 (*graph condition* [16]) 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, $\exists(a, ac_C)$ 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 $\bigwedge_{i \in I} ac_i$ are graph conditions over P.

$$\exists\Big(\ P \xrightarrow{a} C,\ ac_C\ \Big) \qquad p \searrow G \swarrow q \quad (=,\ \not\models)$$

Satisfiability of graph conditions is inductively defined as follows:

1. Every morphism satisfies true.
2. A morphism $p : P \to G$ satisfies $\exists(a, ac_C)$ 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 $\neg ac$ over P if p does not satisfy ac and p satisfies $\bigwedge_{i \in I} ac_i$ over P if p satisfies each ac_i $(i \in I)$.

We write $p \models ac$ when the morphism p satisfies ac.

Graph conditions can be equipped with *typing* over a given type graph TG as usual [15] 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 $G \models ac_I$, if the initial morphism $i_G : I \to G$ satisfies ac_I.

Notation 2 *Note that $\exists a$ abbreviates $\exists(a, true)$, $\forall(a,\ ac_C)$ abbreviates $\neg\exists(a, \neg ac_C)$ and $\exists(C, ac_C)$ abbreviates $\exists(i_C,\ ac_C)$ with the initial morphism $i_C : I \to C$. More generally, a graph condition $\exists(a, ac_C)$ over P with $a : P \to C$, is abbreviated also as $\exists(C, ac_C)$ if a denotes an inclusion morphism and if it is clear from the context that $\exists(C, ac_C)$ is a condition over P. Moreover, the depiction of specific constraints shown in this paper (cf. Fig. 5) 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: elements to the left are part of the source, elements to the right are part of the target, elements in the middle model traceability information. In particular, for our formalization of model transformations as triple graph grammars (see Sect. 2.3), this traceability information will be represented by correspondence nodes and links between source and target model elements.*

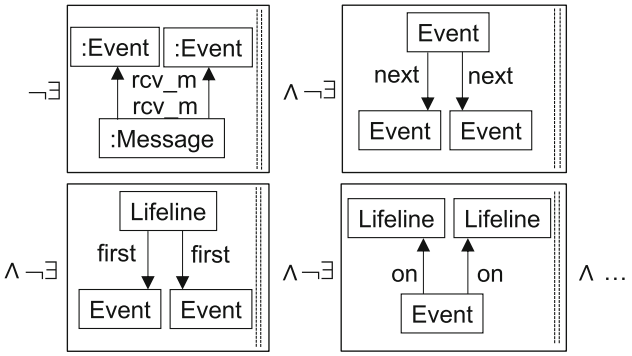


Fig. 5 Fragment of $\mathcal{C}_S$

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 $\mathcal{L}(TG)$ defined by a type graph to those graphs also satisfying a constraint $\mathcal{C}$.

Definition 9 (*Graph language with constraint*) Given a type graph TG and a graph constraint $\mathcal{C}$ typed over TG, the *graph language with constraint* $\mathcal{L}(TG, \mathcal{C})$ denotes the set of all graphs typed over TG that satisfy $\mathcal{C}$, i.e., $\mathcal{L}(TG, \mathcal{C}) = \{G \mid G \in \mathcal{L}(TG) \wedge G \models \mathcal{C}\}$.

Example 4 (Graph language with constraint as modeling language) Figure 5 depicts a fragment of a graph constraint intended to restrict our source modeling language, mostly implementing cardinality constraints of the metamodel. The dashed lines show that the existential constraints only contain and concern elements relevant for the source modeling language. In particular, the following restrictions are depicted:

- there is at most one event attached to receiving the same message,
- an event may at most have one subsequent event,
- a lifeline may not have more than one first event,
- an event may not belong to more than one lifeline.

We denote this conjunction of negated existential constraints, which includes several not depicted here, as $\mathcal{C}_S$. The language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ then describes the (source modeling) language of sequence charts more accurately than as introduced without constraints in Example 3. Likewise, the language $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ for a similar constraint $\mathcal{C}_T$ and the type graph T_{TT} describes the (target modeling) language of communicating automata. For a complete description of $\mathcal{C}_S$ and $\mathcal{C}_T$, we refer to our technical report [13].

2.2 Model semantics

In this article, 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 [15]. In particular, we allow rules to be equipped with application conditions (AC) [16,32], allowing to apply a given rule to a graph G only if the corresponding match morphism satisfies the AC of the rule.

Definition 10 (*graph transformation*) A plain graph transformation rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$ is 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* $\rho = \langle p, ac_L \rangle$ consists of a plain rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$ and an application condition ac_L over L.

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

A *direct graph transformation* via rule $\rho = \langle p, ac_L \rangle$ consists of two pushouts (1) and (2), called DPO, with injective match m and comatch m^* such that $m \vDash ac_L$. 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 [15] by adding typing morphisms from each graph to TG and by requiring type-compatibility with respect to TG for each graph morphism.

The applicability of a graph transformation rule can be expressed as a graph constraint. We exploited this feature already [5,20] 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 $\rho = \langle p, ac_L \rangle$ with $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$, expresses that an injective match m exists such that the application condition ac_L and the deletable condition $Deletable(p)$,[1] guaranteeing the existence of a PO-complement for $m \circ l$, are fulfilled. Then it is obvious that rule $\rho = \langle p, ac_L \rangle$ is applicable via some injective match m to a graph G if and only if G fulfills the rule applicability constraint.

Definition 11 (*rule applicability constraint*) For a rule $\rho = \langle p, ac_L \rangle$ with *plain* rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$, $App(\rho) = \exists(i_L, ac_L \wedge Deletable(p))$ is the rule applicability constraint of ρ. We moreover write $ac_{App(\rho)}$ to refer to $ac_L \wedge Deletable(p)$ for ρ.

Example 5 (rule applicability constraint) The applicability of the rule *send*, depicted in Fig. 8, can be expressed as graph constraint $\exists(i_L, true)$, 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 and therefore no dangling edges can ever arise (guaranteeing that the PO-complement always exists).

The rule applicability constraint will later be required to express behavior preservation in a symbolic way (in Sect. 3). In particular, we will use this rule applicability constraint to compose a more complex constraint, the so-called pair constraint. This pair constraint expresses that the applicability of related rules from the source and target semantics of the model transformation is connected in such a way that it implies behavior preservation for each model transformation instance.

Definition 12 (*graph transformation system (w. constraint), set of reachable graphs*) A *graph transformation system* (GTS) $gts = (\mathcal{R}, TG)$ 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 $G \Rightarrow_{gts} G'$. For a GTS $gts = (\mathcal{R}, TG)$ and an initial graph G_0 the *set of reachable graphs* $\mathsf{REACH}(gts, G_0)$ is defined as $\{G \mid G_0 \overset{*}{\Rightarrow}_{gts} G\}$. A *GTS with constraint* $gts^{\mathcal{C}} = (\mathcal{R}, TG, \mathcal{C})$ consists of a GTS $gts = (\mathcal{R}, TG)$ and a constraint $\mathcal{C}$ typed over TG. If a rule ρ in the set of rules $\mathcal{R}$ of $gts^{\mathcal{C}}$ exists such that there exists a typed direct transformation via rule ρ from G to G' both satisfying $\mathcal{C}$, we write $G \Rightarrow_{gts^{\mathcal{C}}} G'$. For a GTS with constraint $gts^{\mathcal{C}} = (\mathcal{R}, TG, \mathcal{C})$ and an initial graph G_0 satisfying $\mathcal{C}$ the *set of reachable graphs* $\mathsf{REACH}(gts^{\mathcal{C}}, G_0)$ is defined as $\{G \mid G_0 \overset{*}{\Rightarrow}_{gts^{\mathcal{C}}} G\}$.

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 Sect. 5, we explain how and with which restrictions automatic invariant checking can be performed statically.

[1] In Lemma 5.9 of [49], 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.

Definition 13 (*inductive invariant* [10]) A graph constraint ac_I is an *inductive invariant* of the GTS $gts = (\mathcal{R}, TG)$, if for all graphs G in $\mathcal{L}(TG)$ such that $G \vDash ac_I \wedge G \Rightarrow_{gts} G'$ it holds that $G' \vDash ac_I$. A graph constraint ac_I is an *inductive invariant* of the GTS *with constraint* $gts^{\mathcal{C}} = (\mathcal{R}, TG, \mathcal{C})$, if for all graphs G in $\mathcal{L}(TG, \mathcal{C})$ such that $G \vDash ac_I \wedge G \Rightarrow_{gts^{\mathcal{C}}} G'$ it holds that $G' \vDash ac_I$.

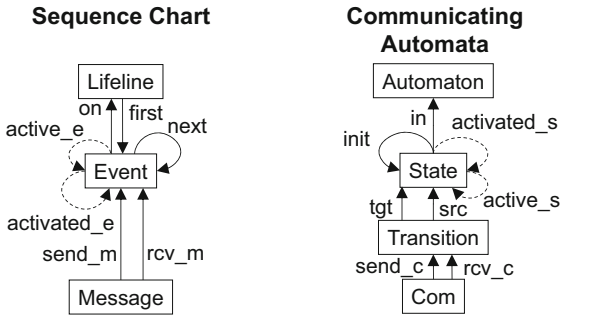
Remark 4 In the latter case, it holds that $G' \in \mathcal{L}(TG, \mathcal{C})$, since it results as a graph from a rule application via the GTS with constraint $gts^{\mathcal{C}}$ from a graph G satisfying $\mathcal{C}$.

Analogous to the work of Hülsbusch et al. [35], 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. 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 14 (*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 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 15 (*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 typed over elements from TG only, then the graph language with constraint $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ is called a *runtime graph language* with *dynamic constraint* $\mathcal{C}_{dyn}$.



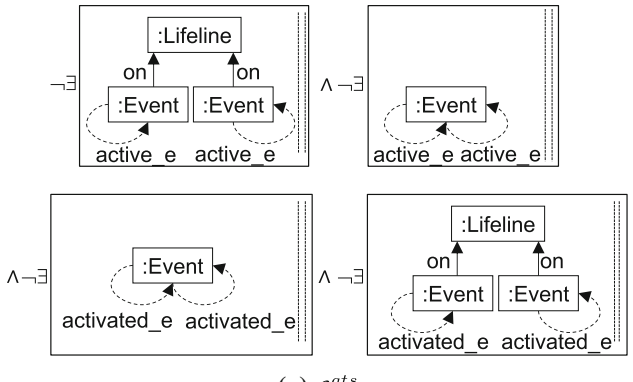
(a) Runtime type graph S_{RT} **(b)** Runtime type graph T_{RT}

Fig. 6 Type graphs S_{TT} and T_{TT} enriched with dynamic types

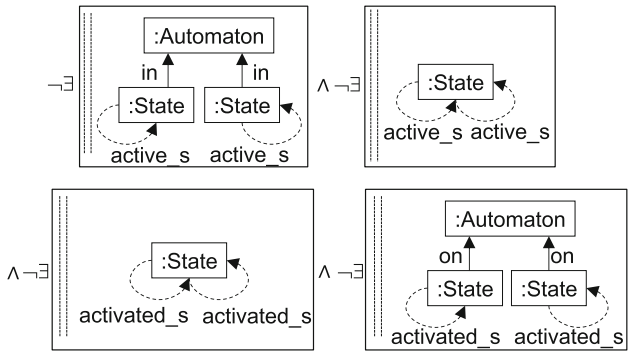
Example 6 (runtime graph language with dynamic constraint) We have a runtime type graph S_{RT} for the source language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ and a runtime type graph T_{RT} for the target language $(\mathcal{L}(T_{TT}, \mathcal{C}_T))$. Both type graphs are depicted in Fig. 6a and b, respectively. The dynamic edge types—active and activated—are denoted with dashed edges. Active edges mark events and states that are currently operating or being operated on. Activated edges denote that an event has been initialized; their purpose will be explained in more detail in the next example.

Moreover, there is a dynamic constraint $\mathcal{C}_s^{gts}$ (Fig. 7a) typed over the runtime type graph S_{RT}. Similarly, there is a dynamic constraint $\mathcal{C}_t^{gts}$ (Fig. 7b) typed over the runtime type graph T_{RT}. In summary, we have the source runtime graph language with dynamic constraint $\mathcal{L}(S_{RT}, \mathcal{C}_S \wedge \mathcal{C}_s^{gts})$ and the target runtime graph language with dynamic constraint $\mathcal{L}(T_{RT}, \mathcal{C}_T \wedge \mathcal{C}_t^{gts})$.

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 graph language. Therefore, we will also call the rules of such a GTS semantics rules and call corresponding rule applications semantics steps. In order for the semantics to be well defined, we assume some extra conditions on the GTS. (1.) 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. (2.) 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. (3.) Lastly, we require that application conditions in the GTS's graph rules deal with additional dynamic elements only. If all three conditions are fulfilled, we say that a GTS *conforms to the given runtime graph language*. Note that conditions (1.) and (2.) ensure that each graph that will



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



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

Fig. 7 $\mathcal{C}_s^{gts}$ and $\mathcal{C}_t^{gts}$

be reachable by applying the semantics rules starting from a given graph G from $\mathcal{L}(TG, \mathcal{C})$ belongs to the runtime graph language $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ and preserves G.

Definition 16 (*conformity to runtime graph language*) Given a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, a corresponding runtime graph language $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ and a GTS $gts = (\mathcal{R}, TG')$ typed over the runtime type graph TG'. We say that *gts conforms to the runtime graph language* $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ if each of the following conditions holds:

1. For each rule $\rho = \langle p, ac_L \rangle$ in $\mathcal{R}$ with $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$, it holds that all static nodes/edges in L and R have a pre-image in I.
2. The GTS with constraint $gts^{\mathcal{C}} = (\mathcal{R}, TG', \mathcal{C})$ has the dynamic constraint $\mathcal{C}_{dyn}$ as inductive invariant.
3. For each rule $\rho = \langle p, ac_L \rangle$ in $\mathcal{R}$ with $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$, it holds that ac_L is not equivalent to false and for each morphism $a : P \to C$ occurring in ac_L it holds that $C \backslash a(P)$ consists of dynamic nodes/edges only.

When the runtime graph language for the respective GTS is clear from the context, we will also speak only of a *runtime-conforming* GTS.

Establishing the notion of runtime conformity allows for a later separation of our verification scheme into a phase where runtime (dynamic) elements are not changed and a phase where static elements are not changed. If, in particular, the graph transformation system modeling semantics does not change static types, any static constraints valid for a model transformation instance cannot be violated by execution of the semantics. That knowledge is exploited during the verification process.

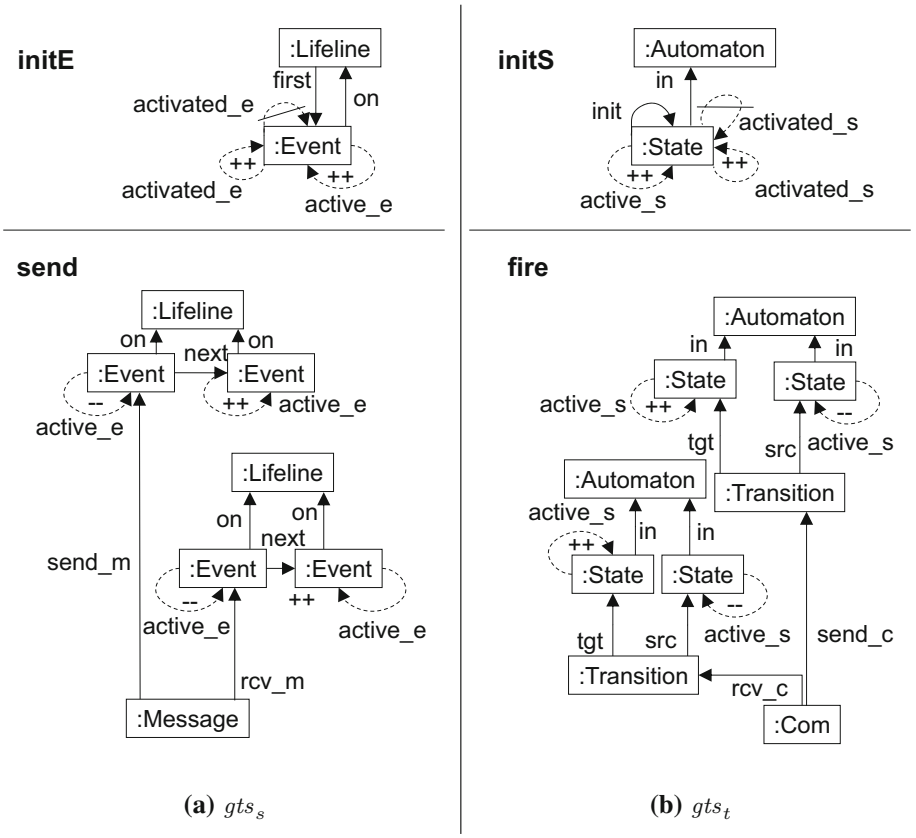
Note that the restriction on application conditions in rules (only dynamic types) limits the expressiveness of the respective graph transformation systems. However, it will later allow us to provide an automatic derivation for constraints required in our verification scheme. If these constraints are instead specified manually, this restriction can be dropped.

Example 7 (conformity to runtime graph language) For our runtime source language $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ with runtime type graph S_{RT} (Fig. 6a), we can define a runtime-conforming source GTS $gts_s = (\mathcal{R}_s, S_{RT})$ with $\mathcal{R}_s = \{initE, send\}$ as depicted in Fig. 8a. Likewise, for our runtime target language $\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$ with the runtime type graph T_{RT} (Fig. 6b), we define a target GTS $gts_t = (\mathcal{R}_t, T_{RT})$ with $\mathcal{R}_t = \{initS, fire\}$ as depicted in Fig. 8b. Elements created or deleted by a rule are marked with "**++**" or "**−**", respectively. We depict a NAC by crossing out the elements that it forbids.

Consider the two rules in the source GTS $gts_s = (\{initE, send\}, S_{RT})$. The rule *initE* describes initialization of a lifeline: the first event of the lifeline is activated—by creating an active edge—and marked accordingly with an activated edge. The rule also has a NAC that forbids its application if the event in question already has an activated edge. This prevents multiple initializations of the same lifeline, which would result in multiple active edges on the same lifeline or even the same event. For the notion of sequence charts used here, this scenario has no useful semantic interpretation; in fact, $\mathcal{C}_s^{gts}$ forbids the existence of two active edges on events on the same lifeline (see Fig. 7a). As explained, active edges mark events currently operating and will be removed and created by rule applications; activated edges, on the other hand, cannot be removed by either *initE* or *send*.

Rule *send* describes that a message is sent and received—denoted by corresponding events—between two different lifelines. Before rule application, the events denoting the sending and receiving of a message are active; afterward, their subsequent events become active.

The target GTS $gts_t = (\{initS, fire\}, T_{RT})$ is similar. First, $initS$, initializes an automaton by creating an active edge and an activated edge on the automaton's initial state. Again, a NAC prevents multiple initializations of an initial state. Second, *fire* describes firing of linked transitions in two communicating automata.



(a) gts_s **(b)** gts_t

Fig. 8 GTSs conforming to the source (target) runtime graph language

Note that gts_s indeed conforms to $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ and gts_t conforms to $\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$. In both cases, the conditions of Definition 16 are satisfied:

1. The rules preserve all elements with static types—only dynamic edges are created or removed.
2. The corresponding GTS with constraint $gts_s^{\mathcal{C}_S}$ has the dynamic constraint $\mathcal{C}_s^{gts}$ (Fig. 7a) as an inductive invariant. Likewise, the GTS with constraint $gts_t^{\mathcal{C}_T}$ has the dynamic constraints $\mathcal{C}_t^{gts}$ (Fig. 7b) as an inductive invariants. In particular, the NACs in *initE* and *initS* are instrumental here: in their absence, multiple initializations of a lifeline or automaton would lead to two active edges on that lifeline or automaton, violating the dynamic constraint. In addition, the static constraints prevent the existence of two first events on a lifeline and two states in an automaton, which could also lead to a violation of the dynamic constraints.
3. Both NACs forbid the existence of activated edges; hence, for each rule, the set difference between NAC elements and regular rule elements in the left side consists of one activated edge only—which is a dynamic element.

The semantic mapping of a source and target graph language is based on the previously introduced graph transformation systems conforming to 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 conforming source (or target) GTS, respectively.

Definition 17 (*labeled transition system*) A *labeled transition system* (LTS) $lts = \langle Q, L, \rightarrow, i \rangle$ consists of a set of states Q, a label alphabet L, a labeled transition relation $\rightarrow \subseteq Q \times L \times Q$, and an initial state i.

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 limit the information captured by the labeling upfront, we employ both here for the induced LTS.

Definition 18 (*induced LTS*) Given a type graph TG, a graph transformation system $gts = (\mathcal{R}, TG)$, and a graph G_0 typed over TG, then the *LTS induced by gts and* G_0, denoted by $LTS(gts, G_0)$, equals $\langle Q_{gts}, \mathcal{M}_{\mathcal{R}}, \rightarrow_{gts}, G_0 \rangle$ with

- $Q_{gts} = \mathsf{REACH}(gts, G_0)$,
- $\mathcal{M}_{\mathcal{R}} = \{(\rho, m) \mid \rho \in \mathcal{R} \wedge m \in \mathcal{M}_{\text{æ}}\}$ with $\mathcal{M}_\rho$ the set of injective graph morphisms having as domain the LHS of ρ, and

– $\rightarrow_{gts} = \{(G, (\rho, m), G') | G, G' \in Q_{gts} \wedge (\rho, m) \in \mathcal{M}_{\mathcal{R}} \wedge G \Rightarrow_{m,\rho} G'\}$.

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 $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ with runtime-conforming GTS $gts = (\mathcal{R}, TG')$, then the semantic mapping maps each G in $\mathcal{L}(TG, \mathcal{C})$ to the induced labeled transition system $LTS(gts, G)$, the semantics of G. The semantic domain thereby consists of the set of all LTSs that are induced from the graph transformation system gts and some graph in $\mathcal{L}(TG, \mathcal{C})$. This is illustrated by the following example.

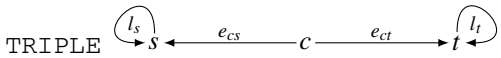
Example 8 (induced LTS as model semantics) Given the graph transformation system gts_s conforming to the runtime source graph language as introduced in Example 7, the induced labeled transition system $LTS(gts_s, S)$ describes for each graph S in $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ the corresponding semantics. Similarly, given gts_t conforming to the runtime target graph language, $LTS(gts_t, T)$ describes for each graph T in $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ the corresponding semantics.

2.3 Model transformation

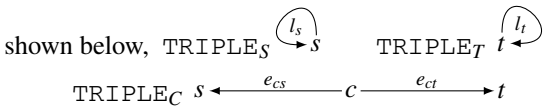
In this article, we will formalize model transformations according to Definition 3 using triple graph grammars. Triple graph grammars (TGGs) define model transformations in a relational (declarative) way by combining three conventional graph grammars for the source, target and correspondence models. 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. TGGs can describe unidirectional model transformations that only support to derive a target model for a given source model using the forward transformation or bidirectional model transformations that also support the inverse direction via the backward transformation [23]. However, for our modeling scheme and verification approach we do not make any special assumption concerning the unidirectional or bidirectional nature of the model transformations described by TGGs.

We use a TGG formalization [23,28] more suitable for the current practice for TGGs than the one originally introduced [51]. The main idea is to use a distinguished, fixed graph TRIPLE which all triple graphs—including the type triple graph $TT = S_{TT}C_{TT}T_{TT}$—are typed over.



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



are the *source*, *target*, and *correspondence 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 that are 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 paper, 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 is a *source graph* belonging to the language $\mathcal{L}(S_{TT})$. Similarly, T_G is a *target graph* belonging to the language $\mathcal{L}(T_{TT})$; C_G is a *correspondence graph* belonging to the language $\mathcal{L}(C_{TT})$.

Notation 3 *Note that each source graph corresponds uniquely to a triple graph with empty correspondence and target components; each target graph corresponds uniquely to a triple graph with empty source and correspondence components. Therefore, if it is clear from the context that we are dealing with triple graphs, we denote triple graphs $S_G\varnothing\varnothing$ with empty correspondence and target components also as S_G; likewise, triple graphs $\varnothing\varnothing T_G$ can be denoted as T_G.*

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 $S_{G_0} C_{G_0} T_{G_0} \overset{*}{\Rightarrow} S_{G_n} C_{G_n} T_{G_n}$, 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. Further, we do not allow TGGs to be equipped with application conditions. However, we allow grammars to be equipped with a so-called *TGG constraint* $\mathcal{C}_{tgg}$ 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 $\mathcal{C}_{tgg}$.

Definition 19 (*Triple graph grammar, language*) A *triple graph grammar* (TGG) $tgg = (\mathcal{R}, S_{TT} C_{TT} T_{TT}, S_A C_A T_A)$ 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* $\mathcal{L}(tgg)$ 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 $\mathcal{C}_{tgg}$ typed over the type triple graph $S_{TT} C_{TT} T_{TT}$ such that $S_A C_A T_A \models \mathcal{C}_{tgg}$. The *triple graph language* $\mathcal{L}(tgg, \mathcal{C}_{tgg})$ equals $\mathsf{REACH}((\mathcal{R}, S_{TT} C_{TT} T_{TT}, \mathcal{C}_{tgg}), S_A C_A T_A)$.

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 article.

Example 9 (Triple Graph Grammar) In Fig. 9, we show an example TGG tgg with an empty 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 Fig. 9b 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 (where messages can be sent and received between different lifelines) and a system of communicating automata; both the source and target modeling languages have been described in Examples 3 and 4. The events, which denote the sending/receiving of a message between lifelines, correspond to states before and after firing of the respective transitions. On each lifeline, there is one first event which corresponds to an initial state in exactly one corresponding automaton.

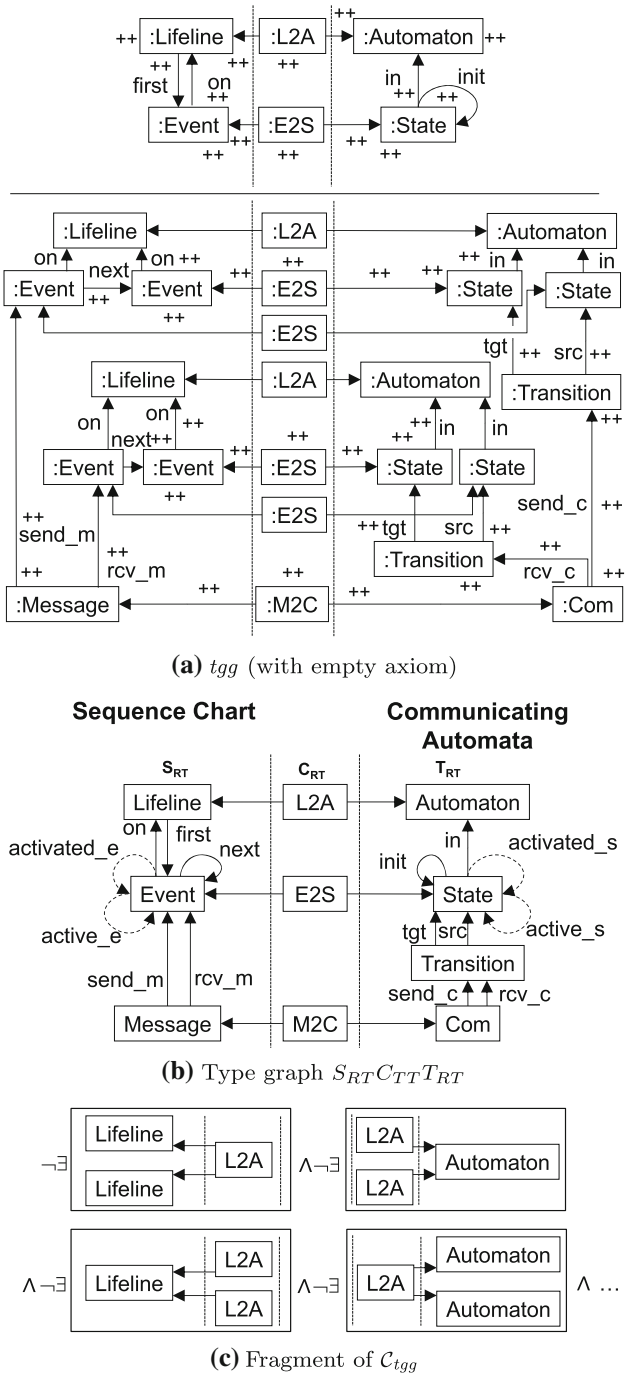


Fig. 9 *tgg* with an empty axiom and two rules, type graph $S_{RT} C_{TT} T_{RT}$, fragment of $\mathcal{C}_{tgg}$

Figure 9c shows a fragment of the TGG constraint $\mathcal{C}_{tgg}$. This fragment is concerned specifically with the cardinalities between correspondence nodes of type L2A and lifelines

and automata. Other constraint fragments follow a similar scheme. This type of cardinality constraint restricts the number of valid triple graphs allowed by the model transformation.

Note that the triple graph grammar does not consider automata with cycles in their transition systems. A transition can only be created along with a new target state and may not have an existing state as a target. This is intended: the variant of sequence charts we use in our example does not allow cyclic behavior either.

Analogous to our earlier work [22] and the work of Hülsbusch et al. [35], we derive a model transformation over $\mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ from a given TGG *tgg* typed over $S_{TT}C_{TT}T_{TT}$. Additionally, we allow *tgg* to be equipped with a TGG constraint $\mathcal{C}_{tgg}$ such that the model transformation is based on the language $\mathcal{L}(tgg, \mathcal{C}_{tgg})$. Moreover, w.l.o.g. we assume that $\mathcal{C}_{tgg}$ comprises the source and target constraints $\mathcal{C}_S$ and $\mathcal{C}_T$ of the source and target graph language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ and $\mathcal{L}(T_{TT}, \mathcal{C}_T)$, respectively. In particular, this assumption ensures that the following definition is well defined.

Definition 20 (*induced model transformation* $MT(tgg, \mathcal{C}_{tgg})$) Given a source and target graph language with constraint $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ and $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ as given in Definition 9 and a TGG *tgg* with TGG constraint $\mathcal{C}_{tgg}$ typed over $S_{TT}C_{TT}T_{TT}$ such that it comprises the source and target constraints $\mathcal{C}_S$ and $\mathcal{C}_T$, then the *induced model transformation* $MT(tgg, \mathcal{C}_{tgg}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ consists of pairs of source and target graphs (S, T) such that there exists some triple graph $SCT \in \mathcal{L}(tgg, \mathcal{C}_{tgg})$ having S and T as source and target component, respectively.

Example 10 (induced $MT(tgg, \mathcal{C}_{tgg})$ as model transformation) In Fig. 10, a source graph S_2 and target graph T_2 is depicted in abstract (Fig. 10a) as well as concrete syntax (Fig. 10b, c) describing a runtime state for the source graph S and target graph T belonging to $MT(tgg, \mathcal{C}_{tgg})$ with *tgg* and $\mathcal{C}_{tgg}$ as described in Example 9 and induced labeled transition systems $LTS(gts_s, S)$ and $LTS(gts_t, T)$ for runtime-conforming gts_s and gts_t as described in Example 8. A triple graph SCT (depicted in Fig. 4 of Example 3) fulfilling $\mathcal{C}_{tgg}$ can be generated by *tgg*, being the subgraph of S_2CT_2 depicted in Fig. 10 obtained by not considering the dynamic elements (*activated* and *active* loops). The currently active events and states on the lifelines and in the automata are $e1$, $e2$, $s1$, and $s2$, respectively.

2.4 Behavior preservation

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 Definition 4.

Definition 21 (*Relabeling*) Given a modeling language in the form of a graph language with constraint $\mathcal{L}(TG, \mathcal{C})$, a corresponding runtime graph language $\mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$ with a conforming GTS $gts = (\mathcal{R}, TG')$, an induced labeled transition system $LTS(gts, G)$ 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 labeled transition system $LTS(gts, G)$ to the LTS where each label (ρ, m) has been replaced by $l(\rho, m)$.

Remark 5 In the rest of the paper, 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 $LTS(gts, G)$ to $l(LTS(gts, G))$ 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 [43], two particular flavors of behavior preservation for a model transformation that require a relation between the states of different labeled transition systems [26,27]. The behavioral equivalence of bisimilarity of two LTSs over the same alphabet is defined as follows:

Definition 22 (*bisimulation relation* [43]) A *bisimulation relation* between two labeled transition systems $lts_1 = \langle Q_1, A, \to, i_1 \rangle$, $lts_2 = \langle Q_2, A, \to, i_2 \rangle$ over the same alphabet A is a relation $BSIM \subseteq Q_1 \times Q_2$ if $(q_1, q_2) \in BSIM$ implies, for all $\alpha \in A$,

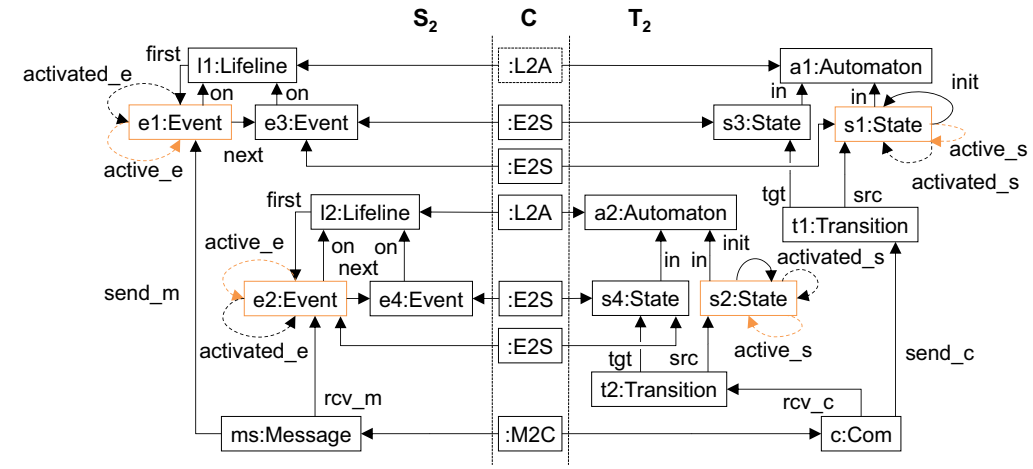
1. Whenever $q_1 \xrightarrow{\alpha} q_1'$ then, for some q_2', $q_2 \xrightarrow{\alpha} q_2'$ and $(q_1', q_2') \in BSIM$.
2. Whenever $q_2 \xrightarrow{\alpha} q_2'$ then, for some q_1', $q_1 \xrightarrow{\alpha} q_1'$ and $(q_1', q_2') \in BSIM$.

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

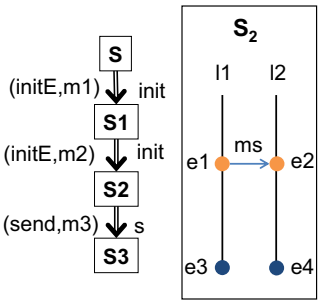
Note that if $lts_1 =_{bsim} lts_2$ holds, we can conclude, for example, that all traces of lts_1 can also be found in lts_2 and vice versa. Consequently, $lts_1 =_{bsim} lts_2$ describes that lts_1 somehow preserves all possible behavior of lts_2 (equivalence).

In some cases, behavioral refinement—a weaker kind of behavior preservation—is sufficient. Then, we can consider a simulation relation instead of a bisimulation relation.[2]

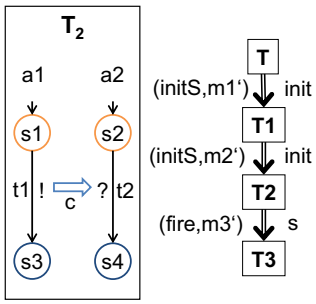
[2] Note that a simulation relation SIM is also a bisimulation relation if SIM^{-1} is a simulation relation as well.



(a) S_2CT_2 with SCT S_2CT_2 and SCT $MT(tgg, \mathcal{C}_{tgg})$; abstract syntax



(b) S_2 (runtime state) in concrete syntax and induced labelled transition systems $LTS(gts_t, T)$ for gts_s conforming to the runtime graph languages



(c) T_2 (runtime state) in concrete syntax and induced labelled transition system $LTS(gts_t, T)$ for gts_t conforming to the runtime graph language

Fig. 10 Induced MT($tgg, \mathcal{C}_{tgg}$) as model transformation and runtime states

Definition 23 (*simulation relation* [43]) A *simulation relation* between two labeled transition systems $lts_1 = \langle Q_1, A, \rightarrow, i_1 \rangle$, $lts_2 = \langle Q_2, A, \rightarrow, i_2 \rangle$ over the same alphabet A is a relation $SIM \subseteq Q_1 \times Q_2$ if $(q_1, q_2) \in SIM$ implies, for all $\alpha \in A$,

– Whenever $q_1 \xrightarrow{\alpha} q_1'$, then, for some q_2', $q_2 \xrightarrow{\alpha} q_2'$ and $(q_1', q_2') \in SIM$.

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

Note that if $lts_1 \leq_{sim} lts_2$ 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, $lts_1 \leq_{sim} lts_2$ 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.

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 1. 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}. In the rest of the article, we will often need all artifacts 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 article.

Definition 24 (*Formally Covered Model Transformation*) A *formally covered model transformation* is a tuple $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ with

– $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ and $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ the source and target graph language with constraint (see Definition 9), respectively,

- $MT(tgg, \mathcal{C}_{tgg}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ the induced model transformation for a $tgg = ((\mathcal{R}, S_{TT}C_{TT}T_{TT}), S_AC_AT_A)$ with TGG constraint $\mathcal{C}_{tgg}$ (see Definition 20), and
- $LTS(gts_s, .)$ and $LTS(gts_t, .)$ (see Definition 18) the induced LTSs for each element of $\mathcal{L}(S_{TT}, \mathcal{C}_S)$ and $\mathcal{L}(T_{TT}, \mathcal{C}_T)$ such that $\mathcal{L}(S_{RT}, \mathcal{C}_s^{gts})$ and $\mathcal{L}(T_{RT}, \mathcal{C}_t^{gts})$ are runtime source and target graph languages with dynamic constraint (see Definition 15) together with runtime-conforming source and target GTSs $gts_s = (\mathcal{R}_s, S_{RT})$ and $gts_t = (\mathcal{R}_t, T_{RT})$ (see Definition 16), respectively.

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. 1 and Table 1). As illustrated in the figure, the semantics of source and target language is defined by the induced LTSs of runtime-conforming 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.

Definition 25 (*Behavior Preservation—Transformation Level—Formalized*) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24 together with relabelings l_s and l_t for $LTS(gts_s, .)$ and $LTS(gts_t, .)$ into a common alphabet A as given in Definition 21, respectively, we say that

1. the model transformation $MT(tgg, \mathcal{C}_{tgg})$ is *behavior preserving in an equivalent manner* if for each pair of source and target models $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ it holds that $l_s(LTS(gts_s, S)) =_{bsim} l_t(LTS(gts_t, T))$.
2. the model transformation $MT(tgg, \mathcal{C}_{tgg})$ is *behavior preserving in a refining manner* if for each pair of source and target models $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ it holds that $l_t(LTS(gts_t, T)) \leq_{sim} l_s(LTS(gts_s, S))$.

Example 11 (Behavior Preservation—Transformation Level—Formalized) The example for Definition 25 consists of the TGG tgg with TGG constraint $\mathcal{C}_{tgg}$ of Example 9 shown in Fig. 9, describing a model transformation $MT(tgg, \mathcal{C}_{tgg}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ between sequence charts (source modeling language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$) and systems of communicating automata (target modeling language $\mathcal{L}(T_{TT}, \mathcal{C}_T)$). 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 6 and corresponding conforming graph transformation systems $gts_s = (\{initE, send\}, S_{RT})$ and $gts_t = (\{initS, fire\}, T_{RT})$ (see Example 7) describing the possible behavior of sequence charts and automata, respectively. The GTSs $gts_s^{\mathcal{C}_S}$ and $gts_t^{\mathcal{C}_T}$ 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 $LTS(gts_s, S)$ and $LTS(gts_s, T)$ to S and T, respectively.

To summarize, we have now established with Definition 25 as depicted in Table 1 a formalization for the general definition of behavior preservation (see Definition 6) such that most concepts are at hand. We have explained how the modeling language (modeling step M_{lang}), model semantics (modeling step M_{sem}), and model transformation (modeling step M_{trans}) have to be formalized and have defined our notion of behavioral equivalence and behavioral refinement (modeling step M_{pres}). However, while 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 worked at the instance level, we want to address the verification at the transformation level (see Definition 6). Therefore, 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. Consequently, 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 section, this is not trivial as we have to define a *symbolic relabeling* covering all pairs of semantics of the source and target at once.
- The bisimulation/simulation relations for the LTS of potentially infinitely many source and target models have to be considered. Therefore, we need to define *symbolic relations* for bisimulation resp. simulation that cover all pairs of the semantics of the source and target at once.

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 sections—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 using a symbolic encoding.

3 Modeling behavior preservation symbolically

This section is concerned with refining modeling step M_{pres} (of our modeling scheme), which describes behavior preservation and has been established as a general concept in Sect. 2.4, in a symbolic way. This symbolic way will allow us to define the relabelings and the bisimulation/simulation relation for all source and target models at the same time and consequently overcome the limitations as discussed at the end of Sect. 2.4.

First (Sect. 3.1), we present a relabeling (cf. Definition 21) that is defined symbolically such that it applies to the labeled transition systems of all pairs of source and target model of a given model transformation at once. Then (Sect. 3.2), we introduce some formal concepts required for specifying M_{pres} in a symbolic way such that it covers the labeled transition systems of all pairs of source and target model of a given model transformation at once—and also permits to exploit our knowledge about correspondences of the states in the source and target model semantics. Thereafter, we define bisimulation (Sect. 3.3) and simulation (Sect. 3.4) for the labeled transition systems of all pairs of source and target models of a given model transformation at once in a symbolic way. Our verification scheme will follow in the next section.

3.1 Symbolic relabeling

We first describe how the relabelings l_s and l_t can be defined symbolically for the source and target labeled transition systems of each proper pair of source and target model based on correspondence structures generated by the TGG. 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.

We can as a first step relate the rules of source and target semantics using two bijective mappings $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$, mapping source and target rules to the same alphabet A. This results in a set of equivalent rule pairs $Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$.

Definition 26 (*set of pairs* $Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as given in Definition 24 and two 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* $Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}}) = \{(\rho_s, \rho_t) | l_s^{\mathcal{R}}(\rho_s) = l_t^{\mathcal{R}}(\rho_t) \wedge \rho_s \in \mathcal{R}_s, \rho_t \in \mathcal{R}_t\}$.

In addition to relating equivalent rules, 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 between their co-domains.

Definition 27 (*correspondences between source and target rules* $C_{(\rho_s,\rho_t)}$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24 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 Definition 26, 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 $L_s C_{(\rho_s,\rho_t)} L_t$ is a well-defined triple graph over $S_{RT} C_{TT} T_{RT}$.

We believe it is reasonable to leave the specification of remappings and correspondences between mapped source and target rules to the user: in order to verify behavior preservation, we require the user to specify his or her notion of how source and target semantics rules map to each other (remappings) and how source and target model elements in those rules relate to each other by correspondences.

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

Definition 28 ($\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t, \mathcal{P}(l_s, l_t)^{Cor}$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24 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 Definition 26 and $C_{(\rho_s,\rho_t)}$ as given in Definition 27, the *pair rule with correspondences* $\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t$ equals the E-concurrent rule [16,17] of ρ_s and ρ_t via the E-dependency relation $R_s \to R_s C_{(\rho_s,\rho_t)} L_t \leftarrow L_t$ with underlying plain rule $p_s *_{R_s C_{(\rho_s,\rho_t)} L_t} p_t : L_s C_{(\rho_s,\rho_t)} L_t \leftarrow I_s C_{(\rho_s,\rho_t)} I_t \to R_s C_{(\rho_s,\rho_t)} R_t$ as depicted in the diagram in Fig. 11 where all morphisms are inclusions.

Vice versa, the analog *pair rule with correspondences* $\rho_t *_{R_t C_{(\rho_s,\rho_t)} L_s} \rho_s$ equals the E-concurrent rule [16,17] of ρ_t and ρ_s via the E-dependency relation $R_t \to R_t C_{(\rho_s,\rho_t)} L_s \leftarrow L_s$ consisting of inclusions.

We define $\mathcal{P}(l_s, l_t)^{Cor} = \{\rho_s *_{R_s C_{(\rho_s,\rho_t)} L_t} \rho_t | (\rho_s, \rho_t) \in Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})\}$ as the set of *pair rules with correspondences*.

The idea behind E-concurrent rules [16,17] is to encode subsequent application of two rules in one (concurrent) rule application while taking possible mutual dependencies (e.g., overlappings, application conditions) into account. Those dependencies are encoded in the E-dependency relation (and a new combined application condition). In our case, the E-dependency relation is two inclusions of the respective source

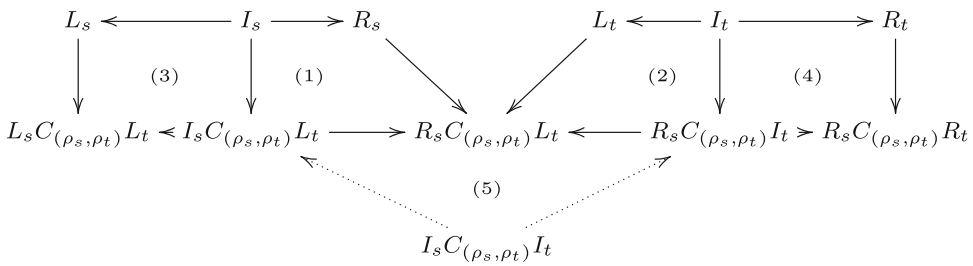
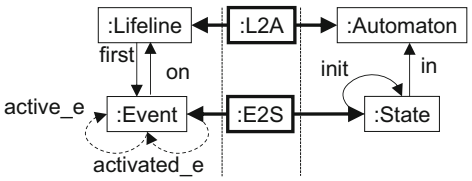
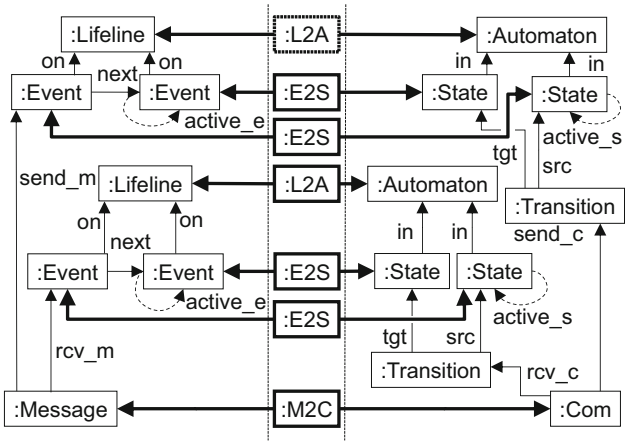


Fig. 11 Construction of pair rule with correspondences



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



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

Fig. 12 Correspondences between source and target rules

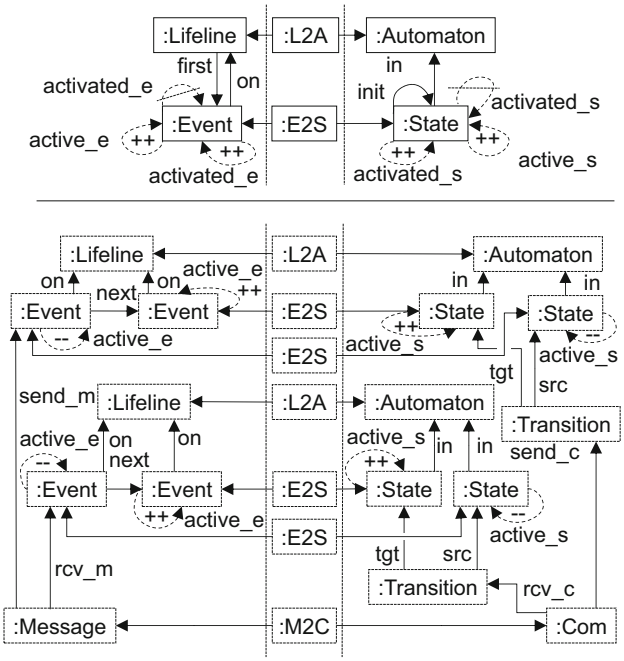


Fig. 13 Pair rules with correspondences $\mathcal{P}(l_s, l_t)^{Cor}$

(target) semantics rule into both rules glued together by the formerly defined correspondences. With respect to our pair rules and bisimulation, the intent is to encode pairwise rule application of equivalent and corresponding rules: an application of a rule on the source model must be followed by subsequent rule application of an equivalent rule on the target model.

Note that the construction of E-concurrent rules involves combining a right-hand side (source) with a left-hand side (target). This is just a feature of the construction [16,17]. Its result is still a rule that, intuitively, replaces the left sides by the right sides.

Example 12 (correspondences between source and target rules, pair rules with correspondences $\mathcal{P}(l_s, l_t)^{Cor}$) For the current example, Fig. 12 shows 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 $S_{RT}C_{TT}T_{RT}$.

Further, Fig. 13 shows the pair rules with correspondences for the example. Note that the crossed edges depict two negative application conditions that forbid the existence of the respective edge. Both conditions, conjunctively joined, constitute the left application condition of the pair rule.

For the rule pair $(initE, initS)$, the result is a rule representing subsequent application of $initE$ and $initS$, with a similar outcome for $(send, fire)$. In addition to the equivalence of both rules in a pair, we also compare matches. Thus, we also require the existence of the established correspondences (Fig. 12a, b) between the source and target elements and consequently, the correspondences are part of the pair rule construction.

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 29 (*relabelings* l_s and l_t) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24 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 $Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ as given in Definition 26 and $C_{(\rho_s,\rho_t)}$ 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 Definition 27, 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) $(\rho_s, \rho_t) \in Pair(l_s^{\mathcal{R}}, l_t^{\mathcal{R}})$ and (2) $S_1 \Rightarrow_{\rho_s,m_s} S_2$ in $LTS(gts_s, S)$ and $T_1 \Rightarrow_{\rho_t,m_t} T_2$ in $LTS(gts_t, T)$ for some $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ because there exists SCT in $\mathcal{L}(tgg, \mathcal{C}_{tgg})$ 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.

Example 13 (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, the upper and middle part of Fig. 14 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. The lower and middle part of Fig. 14 shows how these matches can be extended to matches for $initE *_{R_sC_{(initE,initS)}L_t} initS$ in the triple graph SCT such that we have $m = m'$ and consequently $l_s(initE, m_s) = (init, m) = l_s(initS, m_t)$. After applying the relabeling, i.e., the extension to m, the individual matches m_s and m_t are no longer relevant; hence, they are grayed out in the figure.

3.2 Symbolic relations

According to Definition 25, 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 of a relation over $\mathsf{REACH}(gts_s, S) \times \mathsf{REACH}(gts_t, T)$ between the respective states of the transition systems for each (S, T). We know that for each (S, T) in $MT(tgg, \mathcal{C}_{tgg})$, there exists some SCT in $\mathcal{L}(tgg, \mathcal{C}_{tgg})$. Then a key idea of our approach is

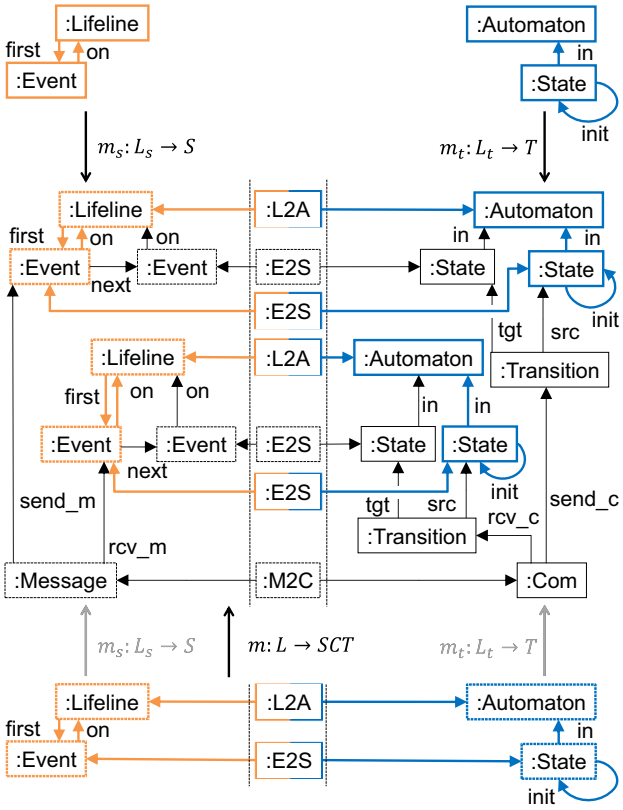


Fig. 14 Matches for $initE$ and $initS$

to require that (S, T) belongs to the relation if the corresponding SCT fulfills some particular constraint $\mathcal{C}$. So we derive the bisimulation resp. 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 for each (S, T) a relation over $\mathsf{REACH}(gts_s, S) \times \mathsf{REACH}(gts_t, T)$ from a constraint $\mathcal{C}$ typed over $S_{RT}C_{TT}T_{RT}$.

Definition 30 (*induced relation* $\mathcal{R}(\mathcal{C}, .)$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24 and given a graph constraint $\mathcal{C}$ typed over $S_{RT}C_{TT}T_{RT}$, then for each (S, T) in $MT(tgg, \mathcal{C}_{tgg})$ and a corresponding SCT in $\mathcal{L}(tgg, \mathcal{C}_{tgg})$ the *induced relation* $\mathcal{R}(\mathcal{C}, SCT) \subseteq \mathsf{REACH}(gts_s, S) \times \mathsf{REACH}(gts_t, T)$ 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 $S' \in \mathsf{REACH}(gts_s, S)$ and $T' \in \mathsf{REACH}(gts_t, T)$ as given in Definition 30 is a well-defined triple graph.*

Proof idea Since gts_s and gts_t only change dynamic elements the correspondences given by C remain valid in $S'CT'$. The detailed proof can be found in our technical report [13]. □

In Definitions 22 and 23 for bisimulation and simulation, we refer to the explicit state sets Q_1 and Q_2, while in the symbolic setting here we have to consider all state sets of all possible models at once. As not really all possible state pairs are proper elements of a bisimulation resp. simulation relation ($B \subseteq Q_1 \times Q_2$), but only those where the runtime structures of source and target language should be related to each other via correspondences in pairs of equivalent states, we may employ a constraint typed over $S_{RT}C_{TT}T_{RT}$ to restrict the symbolic encoding of the bisimulation resp. simulation relation to suitable pairs of states only.

3.3 Symbolic bisimulation relation

Having defined relabelings taking into account correspondences between source and target models, we now describe the desired bisimulation relation (to establish behavioral equivalence) between the relabeled transition systems. In particular, it is derived as the relation induced (see Definition 30) by the *bisimulation constraint* $\mathcal{C}^{Cor}_{Bis} = \mathcal{C}_{RT} \wedge \mathcal{C}^{Cor}_{Pair} \wedge \mathcal{C}^{gts}_{s} \wedge \mathcal{C}^{gts}_{t} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}^{Cor}_{MT}$, which represents our symbolic encoding of behavior preservation. Besides the dynamic constraints (Definition 15) $\mathcal{C}^{gts}_{s}$ and $\mathcal{C}^{gts}_{t}$ and the TGG constraint (Definition 19) $\mathcal{C}_{tgg}$, the bisimulation constraint has the following components, which are explained below:

- the runtime constraint $\mathcal{C}_{RT}$, which is specified manually,
- the pair constraint with correspondences $\mathcal{C}^{Cor}_{Pair}$, which is derived from the pair rules with correspondences $\mathcal{P}(l_s, l_t)^{Cor}$, and
- the model transformation constraint $\mathcal{C}^{Cor}_{MT}$, which is derived from the pair constraint $\mathcal{C}^{Cor}_{Pair}$.

The following definitions for runtime constraint, pair constraint, and model transformation constraint always appear in the context of a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24.

First, the runtime constraint $\mathcal{C}_{RT}$ is a constraint typed over $S_{RT}C_{TT}T_{RT}$ that will be specified manually. It expresses how runtime structures of source and target language should be related to each other via correspondences in pairs of equivalent states. As part of the bisimulation constraint, it restricts possible bisimulation relations to those that fulfill this user-defined notion for behavioral equivalence.

Definition 31 (*runtime constraint* $\mathcal{C}_{RT}$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24, then the *runtime constraint* $\mathcal{C}_{RT} = \mathcal{C}^{f}_{RT} \wedge \mathcal{C}^{b}_{RT}$ is a graph constraint typed over $S_{RT}C_{TT}T_{RT}$.

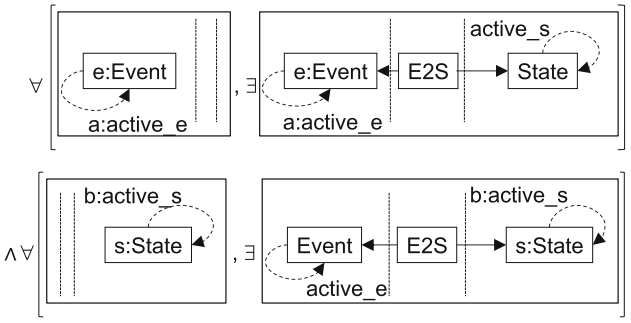


Fig. 15 Runtime constraint $\mathcal{C}_{RT}$

Since we cover bisimulation (instead of simulation only), the runtime constraint is split into a forward runtime constraint $\mathcal{C}^{f}_{RT}$ and backward runtime constraint $\mathcal{C}^{b}_{RT}$. The forward runtime constraint is intended to describe which runtime structures of the source model should imply which corresponding runtime structures in the target model, and vice versa for the backward runtime constraint. The rationale behind the introduction of the runtime constraint lies in its capability to further restrict the notion of behavioral equivalence beyond the pairwise application of semantics rules. Since it is specified manually, it allows for a more fine-grained and rule-independent declaration of which states should be considered as equivalent: equivalence can not only be defined at the level of pairwise rule applications, but also based on comparisons of states, e.g., by comparing flags between corresponding source and target model elements.

Example 14 (runtime constraint) Figure 15 shows the runtime constraint $\mathcal{C}_{RT} = \mathcal{C}^{f}_{RT} \wedge \mathcal{C}^{b}_{RT}$ of our running example. In particular, in order to consider the states of a sequence chart and the corresponding communicating automata as equivalent, we require each active event (on a lifeline) to have a corresponding state (in an automaton) that is also marked as active. Conversely, each active state should have a corresponding active event.

While these structures can also be observed as parts of the pair rules with correspondences, the constraint adds a reasonable and more fine-grained requirement of behavioral equivalence for our running example. Since, in our example, states are the corresponding entities to events—and vice versa—an active event without a corresponding active state should be considered a violation of behavioral equivalence, regardless of the validity (or absence thereof) of other parts of the bisimulation constraint. In other words, corresponding events and states should, at all times, have the same status with respect to being active or not.

Secondly, the pair constraint with correspondences $\mathcal{C}^{Cor}_{Pair} = \mathcal{C}^{Cor,f}_{Pair} \wedge \mathcal{C}^{Cor,b}_{Pair}$ consists of the forward pair constraint

$\mathcal{C}^{Cor,f}_{Pair}$ and the backward pair constraint $\mathcal{C}^{Cor,b}_{Pair}$. The forward pair constraint 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 vice versa for the backward pair constraint.

Definition 32 (*pair constraint $\mathcal{C}^{Cor}_{Pair}$ with correspondences*) Given a formally covered model transformation 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 $Pair(l^{\mathcal{R}}_s, l^{\mathcal{R}}_t)$ as given in Definition 26 and corresponding pair rules with correspondences as given in Definition 28, the *pair constraint $\mathcal{C}^{Cor}_{Pair}$ with correspondences* is defined as $\mathcal{C}^{Cor}_{Pair} = \mathcal{C}^{Cor,f}_{Pair} \wedge \mathcal{C}^{Cor,b}_{Pair}$ with

$$\mathcal{C}^{Cor,f}_{Pair} = \wedge_{(\rho_s,\rho_t)\in Pair(l^{\mathcal{R}}_s,l^{\mathcal{R}}_t)}(\forall(L_s, ac_{App(\rho_s)} \Rightarrow$$

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

and

$$\mathcal{C}^{Cor,b}_{Pair} = \wedge_{(\rho_s,\rho_t)\in Pair(l^{\mathcal{R}}_s,l^{\mathcal{R}}_t)}(\forall(L_t, ac_{App(\rho_t)} \Rightarrow$$

$$\exists(j_t : L_t \to L_s C_{(\rho_s,\rho_t)} L_t, ac_{App(\rho_t * {}_{R_t C_{(\rho_s,\rho_t)} L_s} \rho_s)}))$$

typed over $S_{RT}C_{TT}T_{RT}$ with $\rho_s = \langle p_s : \langle L_s \hookleftarrow I_s \hookrightarrow R_s\rangle, ac_{L_s}\rangle$, $\rho_t = \langle p_t : \langle L_t \hookleftarrow I_t \hookrightarrow R_t\rangle, ac_{L_t}\rangle$, j_s and j_t inclusion morphisms and $ac_{App(\rho)}$ for a given rule ρ as introduced in Definition 11.

With this definition, the forward pair constraint (and, analogously, the backward pair constraint) consists of one constraint $\forall(L_s, \ldots \Rightarrow \exists(j_s \ldots))$ per rule pair (ρ_s, ρ_t); all those constraints are conjunctively joined. The first part of these constraints—the left rule side L_s of the respective rule ρ_s and its application condition and deletable condition $ac_{App(\rho_s)}$—express the applicability of the (source semantics) rule. Further, the morphism j_s is an inclusion of L_s into the left side of the respective pair rule with correspondences. Together with the pair rule's application condition and deletable condition, it guarantees the applicability of the equivalent target semantics rule via correspondences. The latter is of particular importance because our notion of label equivalence and relabeling (Definition 29) requires both similarly labeled semantics rules and matching correspondences between source and target.

Example 15 (pair constraint) Figure 16 shows the forward part $\mathcal{C}^{Cor,f}_{Pair}$ of our running example's pair constraint $\mathcal{C}^{Cor}_{Pair}$. The upper half depicts the constraint fragment for the rule pair (*initE*, *initS*); the lower half shows the fragment for (*send*, *fire*). As explained, both fragments are conjunctively joined.

In particular, consider the first fragment: the first two graphs are the left rule side L_s (of $initE$) and the rule's application condition, respectively. Since the original $initE$ rule does not delete any elements, the deletable condition is (trivially) true and hence, $ac_{App(\rho_s)}$ is true as well. The implication's postcondition is then concerned with the respective pair rule with correspondences: the third graph describes the inclusion of L_s into the pair rule's left side, the fourth and fifth graph are the pair rule's application condition.

Finally, we introduce the *model transformation constraint* $\mathcal{C}^{Cor}_{MT}$ describing the static part from the pair constraint with correspondences $\mathcal{C}^{Cor}_{Pair}$. The intent of the model transformation constraint is to eliminate potential sources of non-behavior preserving model transformations (violation of the pair constraint) that occur because of lack of corresponding static structures required for executable behavior (violation of the model transformation constraint). Therefore, we want the model transformation constraint to be satisfied by each triple graph of the model transformation under consideration. If that were not the case, i.e., if the model transformation creates static structures required for executable behavior in the source model without corresponding structures in the target model, this may lead to the applicability of the respective source semantics rule without applicability of the corresponding pair rule because the target model lacks the static prerequisites. Technically, we use type restriction (denoted as $|S_{TT}C_{TT}T_{TT}$) of graphs and morphisms typed over $S_{RT}C_{TT}T_{RT}$ to their static counterparts omitting all dynamic elements.

Definition 33 (*model transformation constraint $\mathcal{C}^{Cor}_{MT}$*) Given a formally covered model transformation with $\mathcal{C}^{Cor}_{Pair}$ the pair constraint with correspondences as given in Definition 32, then the *model transformation constraint* $\mathcal{C}^{Cor}_{MT} = \mathcal{C}^{Cor,f}_{MT} \wedge \mathcal{C}^{Cor,b}_{MT}$ is a graph constraint typed over $S_{TT}C_{TT}T_{TT}$ with

$$\mathcal{C}^{Cor,f}_{MT} = \wedge_{(\rho_s,\rho_t)\in Pair(l^{\mathcal{R}}_s,l^{\mathcal{R}}_t)}\forall(L_{s|S_{TT}C_{TT}T_{TT}},$$

$$\exists(j_{s|S_{TT}C_{TT}T_{TT}}))$$

and

$$\mathcal{C}^{Cor,b}_{MT} = \wedge_{(\rho_s,\rho_t)\in Pair(l^{\mathcal{R}}_s,l^{\mathcal{R}}_t)}\forall(L_{t|S_{TT}C_{TT}T_{TT}},$$

$$\exists(j_{t|S_{TT}C_{TT}T_{TT}})).$$

The intent of the model transformation constraint, as described above informally, is formalized in Definition 40

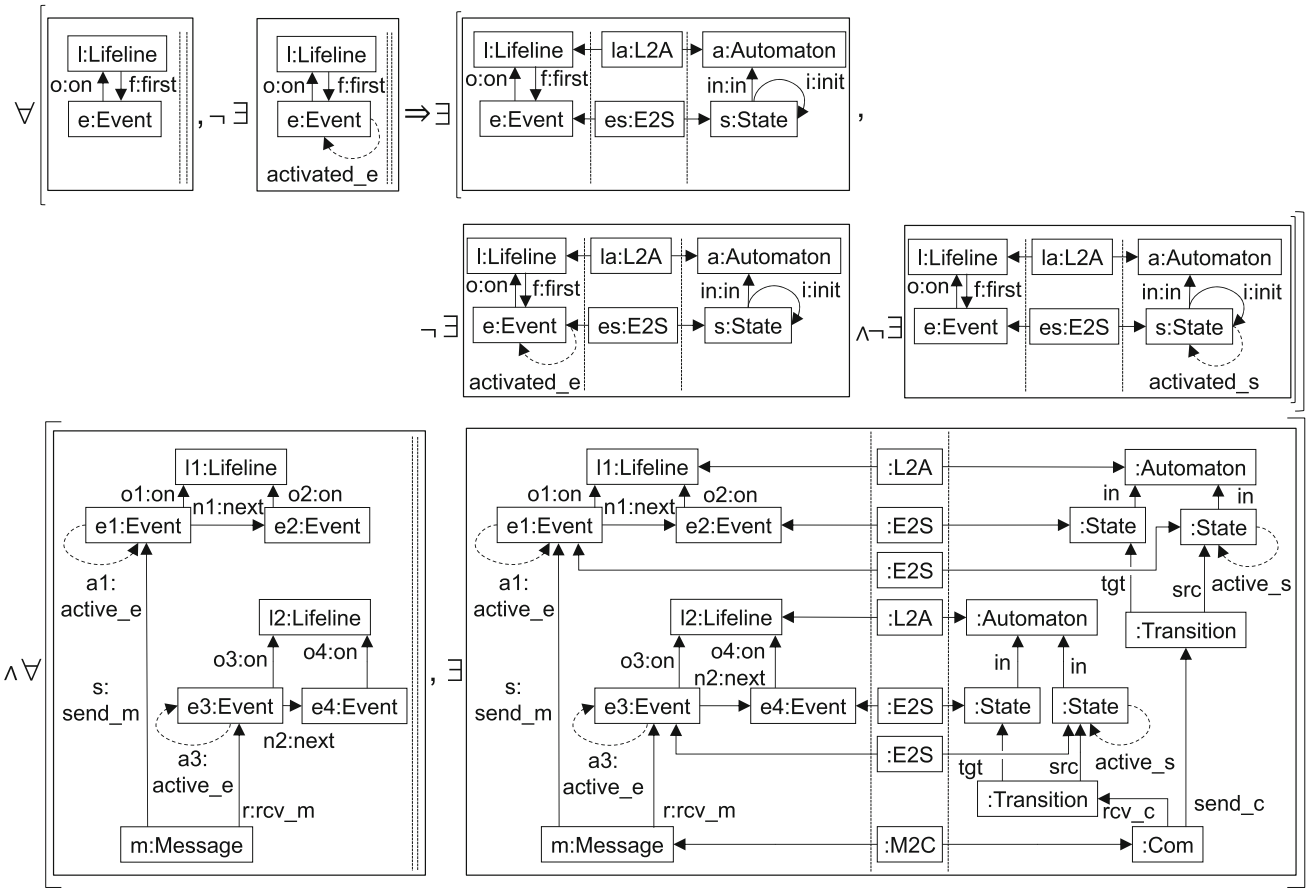


Fig. 16 Forward pair constraint $\mathcal{C}^{Cor,f}_{Pair}$

(Section 4) of our technical report [13]. There, we propose to establish the model transformation constraint over the static type graph such that its violation by a triple graph typed over the static type graph implies the violation of the pair constraint by the same triple graph with additional dynamic elements. We prove [13] that the construction of the model transformation constraint as given in Definition 33 indeed fulfills this declaratively described intent, provided that the semantics rules' application conditions are extending the respective rules by non-static elements only (i.e., elements typed over $S_{RT}C_{TT}T_{RT}$, but not over $S_{TT}C_{TT}T_{TT}$). [3] Given runtime-conforming GTSs, this is the case.

Example 16 (model transformation constraint) Figure 17 shows the forward part $\mathcal{C}^{Cor,f}_{Pair}$ of our example model transformation constraint $\mathcal{C}^{Cor}_{Pair} = \mathcal{C}^{Cor,f}_{MT} \wedge \mathcal{C}^{Cor,b}_{MT}$. The upper fragment corresponds to the (forward) pair constraint fragment concerned with the $initE$ and $initS$ rules (see upper part of Fig. 16), the lower part relates to the rule pair (*send*, *fire*).

Note that, as defined above, any dynamic structures have been removed from the constraint. In particular, consider the part dealing with $initE$ and $initS$: existence of the static structure required for application of $initE$, which lacks the application condition typed over $S_{RT}C_{TT}T_{RT}$, implies the existence of the corresponding static structure required for the application of the respective pair rule with correspondences.

If, on the other hand, we were not to impose that constraint on the model transformation, we might have model transformation instances where the existence of the left graph would lead to a possible application of $initE$ when executing the semantics on the source model; without the corresponding graph required by the constraint (right graph), applicability of $initS$ would not be guaranteed either.

The dynamic constraints, TGG constraint, runtime constraint, pair constraint, and model transformation constraint are then conjunctively combined to form the bisimulation constraint.

[3] The proof assumes that the applicability constraints for semantics rules occurring in $\mathcal{C}^{Cor}_{Pair}$ are trivially true (see the respective Construction 1 [13]). However, the proof can be generalized for applicability constraints using only restrictions on dynamic elements. In this more general case, they can be safely omitted from the pair constraint when restricting it to its static part as given in Definition 33.

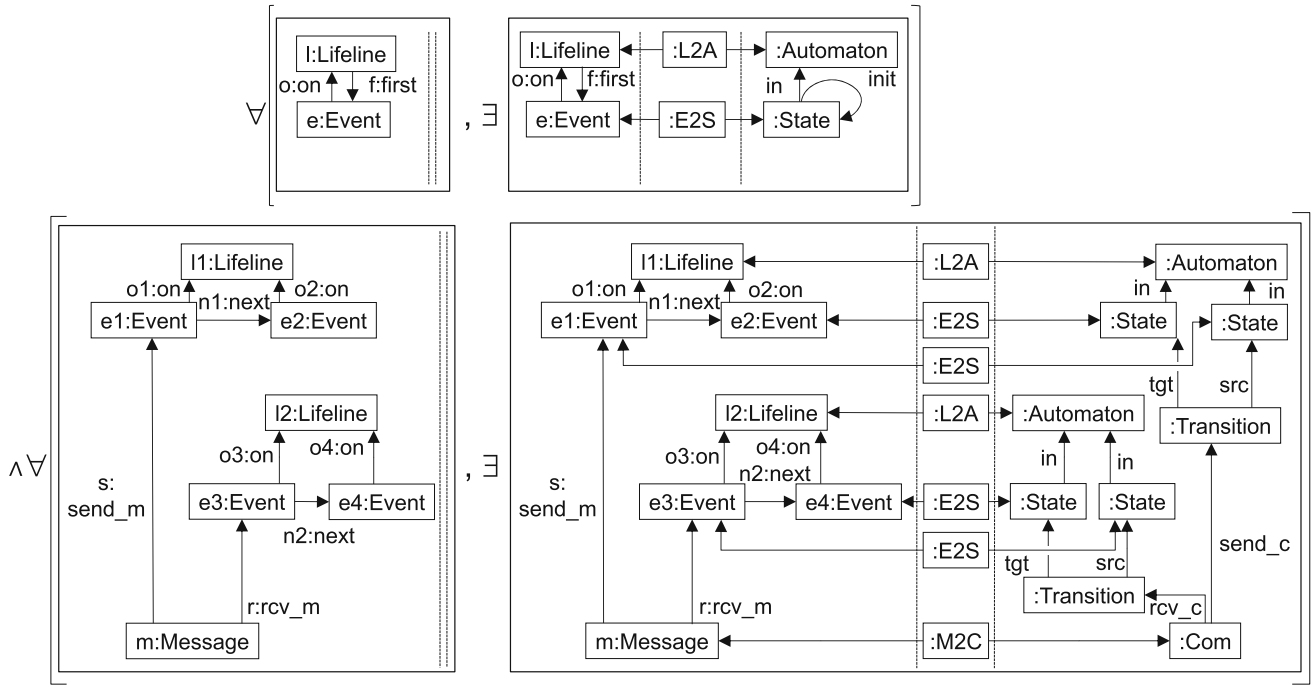


Fig. 17 Forward model transformation constraint $\mathcal{C}_{MT}^{Cor,f}$

Definition 34 (*bisimulation constraint* $\mathcal{C}_{Bis}^{Cor}$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ as introduced in Definition 24. 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 $LTS(gts_s, .)$ and $LTS(gts_t, .)$ derived from $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ as given in Definition 29, then the *bisimulation constraint* $\mathcal{C}_{Bis}^{Cor} = \mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor}$ with $\mathcal{C}_{RT}$ a runtime constraint as given in Definition 31, $\mathcal{C}_{Pair}^{Cor}$ the pair constraint with correspondences as given in Definition 32 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 Definition 28 and $\mathcal{C}_{MT}^{Cor}$ a model transformation constraint as given in Definition 33.

Example 17 (Equivalence) This example describes the problem of verification of behavior preservation in an equivalent manner for model transformations according to Definition 25. In comparison with the simpler example [13,24], it considers more fine-grained notions of relabelings and rule equivalence. In particular, the correspondence extensions with respect to rule equivalence and pair rules allow for the handling of non-deterministic application of semantics rules.

For most artifacts of the example's different modeling steps, Table 3 provides references to the respective figures in this paper. All elements are depicted in full in Appendix B.2 of our technical report [13].

In summary, the triple graph grammar *tgg* with TGG constraint $\mathcal{C}_{tgg}$ describes a model transformation $MT(tgg, \mathcal{C}_{tgg}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}_T)$ between sequence charts with multiple lifelines (source modeling language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$) and systems of communicating automata (target modeling language $\mathcal{L}(T_{TT}, \mathcal{C}_T)$).

Both source and target language are equipped with model semantics specified by runtime-conforming graph transformation systems $gts_s = (\{initE, send\}, S_{RT})$ and $gts_t = (\{initS, fire\}, T_{RT})$ describing the possible behavior of sequence charts and communicating automata, respectively. For source and target models S and T, the semantic mappings $sem_S(S) = LTS(gts_s, S)$ and $sem_T(T) = LTS(gts_s, T)$ 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 $\mathcal{M}$.

$MT(tgg, \mathcal{C}_{tgg})$ is then behavior preserving in an equivalent manner, if for each pair of source and target models $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ it holds that $l_s(LTS(gts_s, S)) =_{bsim} l_t(LTS(gts_t, T))$. In particular, we specify the bisimulation relation as an induced relation $\mathcal{R}(\mathcal{C}_{Bis}^{Cor}, .)$ with the bisimulation constraint $\mathcal{C}_{Bis}^{Cor} = \mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor}$ 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,

Table 3 Figures for elements of Example 17

Step	Modeling languages (M_{lang})				Model semantics (M_{sem})					
Artifact	S_{TT}	$\mathcal{C}_S$	T_{TT}	$\mathcal{C}_T$	$\mathcal{R}_s$	S_{RT}	$\mathcal{C}_s^{gts}$	$\mathcal{R}_t$	T_{RT}	$\mathcal{C}_t^{gts}$
Figure	3b	5	3b	–	8a	6a	7a	8b	6b	7b

Step	Model transformation (M_{trans})				Behavior preservation (M_{pres})			
Artifact	$\mathcal{R}$	$S_{RT}\,C_{TT}\,T_{RT}$	$\mathcal{C}_{tgg}$	corres	$\mathcal{P}(l_s, l_t)^{Cor}$	$\mathcal{C}_{RT}$	$\mathcal{C}_{Pair}^{Cor}$	$\mathcal{C}_{MT}^{Cor}$
Figure	9a	9b	9c	12	13	15	16	17

the derived constraints $\mathcal{C}_{MT}^{Cor}$ and $\mathcal{C}_{Pair}^{Cor}$ also include correspondence information between the respective source and target elements.

More informally, 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 can be followed by an equivalent and corresponding rule application on the target model such that after application of the respective pair rule with correspondences the resulting model states are equivalent again—and vice versa; each rule application on the target model can be followed by a corresponding rule application on the source model such that the resulting model states are equivalent.

Summarizing, in this section we have introduced a symbolic encoding for expressing behavior preservation M_{pres} in an equivalent manner by introducing the bisimulation constraint $\mathcal{C}_{Bis}^{Cor}$ from which an induced bisimulation relation can be derived for all source and target models of a given model transformation (cf. Table 1).

3.4 Symbolic simulation relation

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

As established in Definitions 23 and 25, we require a simulation relation rather than a bisimulation relation. It is similarly derived as an induced relation from the simulation constraint $\mathcal{C}_{Sim}^{Cor,b}$, which considers the dynamic constraints, the TGG constraint and (for the forward transformation) the backward direction of the runtime, pair, and model transformation constraints. The simulation constraint is defined as follows:

Definition 35 (*Simulation constraint* $\mathcal{C}_{Sim}^{Cor,b}$ and $\mathcal{C}_{Sim}^{Cor,f}$) Given a formally covered model transformation $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), \quad LTS(gts_t, .))$ as introduced in Definition 24. 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 $LTS(gts_s, .)$ and $LTS(gts_t, .)$ derived from $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ as given in Definition 29, the simulation constraint $\mathcal{C}_{Sim}^{Cor,b} = \mathcal{C}_{RT}^{b} \wedge \mathcal{C}_{Pair}^{Cor,b} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor,b}$ with $\mathcal{C}_{RT}^{b}$ a backward runtime constraint as given in Definition 31, $\mathcal{C}_{Pair}^{Cor,b}$ the backward pair constraint with correspondences as given in Definition 32 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 Definition 28 and a backward model transformation constraint $\mathcal{C}_{MT}^{Cor,b}$ as given in Definition 33.

Example 18 (Refinement) This example describes the problem of verification of behavior preservation in a refining manner according to case 2 (behavioral refinement) of Definition 25. We consider a TGG tgg' with TGG constraint $\mathcal{C}'_{tgg}$, which induce a model transformation $MT(tgg', \mathcal{C}'_{tgg}) \subseteq \mathcal{L}(S_{TT}, \mathcal{C}_S) \times \mathcal{L}(T_{TT}, \mathcal{C}'_T)$ between sequence charts (modeling language $\mathcal{L}(S_{TT}, \mathcal{C}_S)$) and communicating automata with additional internal behavior (modeling language $\mathcal{L}(T_{TT}, \mathcal{C}'_T)$). In contrast to Example 17, the TGG tgg' now also allows automata with internal behavior—communicating transitions—which is not mirrored by corresponding messages in the sequence chart. The target modeling language's constraint $\mathcal{C}'_T$ and the TGG constraint $\mathcal{C}'_{tgg}$ have been adjusted accordingly. All details and figures of the full example can be found in our technical report [13].

Since messages in sequence charts are still created along with communications and transitions in automata by the TGG, we expect the backward transformation $MT(tgg', \mathcal{C}'_{tgg})^{-1} \subseteq \mathcal{L}(T_{TT}, \mathcal{C}'_T) \times \mathcal{L}(S_{TT}, \mathcal{C}_S)$ from communicating automata to sequence charts to be behavior preserving in a refining manner: all the behavior of a sequence chart is reflected in the corresponding system of communicating automata. However, with additional behavior in automata, the reverse will not be true.

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 with conforming graph transformation systems $gts_s = (\{initE, send\}, S_{RT})$ and $gts_t = (\{initS, fire\}, T_{RT})$, 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 $LTS(gts_s, S)$ and $LTS(gts_s, T)$, 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 equivalently renamed rules from gts_s and gts_t again leads to pair rules with correspondences. Similar to Example 17, 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 transformation $MT(tgg', \mathcal{C}'_{tgg})^{-1}$ is then behavior preserving in a refining manner, if for each pair of models $(T, S) \in MT(tgg', \mathcal{C}'_{tgg})^{-1}$ it holds that $l_s(LTS(gts_s, S)) \leq_{sim} l_t(LTS(gts_t, T))$. In particular, we specify the simulation relation as an induced relation of the simulation constraint $\mathcal{C}_{Sim}^{Cor,f} = \mathcal{C}_{RT}^{f} \wedge \mathcal{C}_{Pair}^{Cor,f} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor,f}$ being the conjunction of the forward part of the runtime constraint, pair constraint, source and target dynamic constraints, TGG constraint, and model transformation constraint.

Intuitively, 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 17, we do not require both directions. Thus, our simulation relation only needs to consider the forward part of the respective constraints.

Summarizing, in this section we have introduced a symbolic encoding for expressing behavior preservation M_{pres} in a refining manner by introducing the simulation constraint $\mathcal{C}_{Sim}^{Cor,b}$ from which an induced simulation relation can be derived for all source and target models of a given model transformation (cf. Table 1).

4 Verification of behavior preservation

Having specified M_{pres} as the final step of our modeling scheme, this section presents our verification scheme for both behavioral equivalence (see Definition 25, case 1) in Sect. 4.1 and behavioral refinement (see Definition 25, case 2) in Sect. 4.2.

4.1 Behavioral equivalence

Our verification scheme for behavioral equivalence is intended to show that 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. 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. We formally define our scheme and prove its correctness in the following theorem.

Theorem 1 (equivalence verification) *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ *as introduced in Definition 24. 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* $LTS(gts_s, .)$ *and* $LTS(gts_t, .)$ *derived from* $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ *and* $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ *as given in Definition 29, a bisimulation constraint* $\mathcal{C}_{Bis}^{Cor} = \mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor}$ *typed over* $S_{RT}C_{TT}T_{RT}$ *as given in Definition 34, then* $MT(tgg, \mathcal{C}_{tgg})$ *is* behavior preserving *in an equivalent manner (in particular, via the induced bisimulation relation* $\mathcal{R}(\mathcal{C}_{Bis}^{Cor}, .)$*) in the sense of case 1 of Definition 25 if the following conditions are fulfilled:*

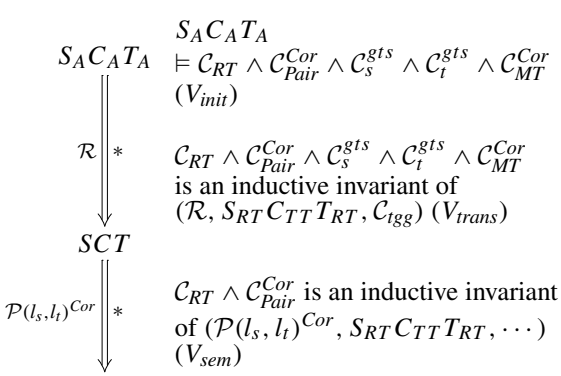
- V_{init}: $S_A C_A T_A \models \mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{MT}^{Cor}$.
- V_{trans}: $\mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts} \wedge \mathcal{C}_{MT}^{Cor}$ *is an inductive invariant of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg})$.
- V_{sem}: $\mathcal{C}_{RT} \wedge \mathcal{C}_{Pair}^{Cor}$ *is an inductive invariant of* $(\mathcal{P}(l_s, l_t)^{Cor}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor} \wedge \mathcal{C}_{s}^{gts} \wedge \mathcal{C}_{t}^{gts})$ *with* $\mathcal{P}(l_s, l_t)^{Cor}$ *as given in Definition 28.*

Proof idea. The detailed proof [13] consists of two parts.

In the first part, we show by induction that all model transformation instances $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ are contained in the induced relation $\mathcal{R}(\mathcal{C}_{Bis}^{Cor}, SCT)$. This follows since $\mathcal{C}_{Bis}^{Cor}$ holds for the triple graph grammar's axiom $S_A C_A T_A$ (V_{init}, base of induction) and since $\mathcal{C}_{Bis}^{Cor}$ is an inductive invariant for the triple graph grammar with constraint $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg})$ (V_{trans}, inductive step). This first part of the proof is depicted by the first arrow ($S_A C_A T_A \Rightarrow_{\mathcal{R}}^{*} SCT$) in the sketch below.

In the second part of the proof, we show that the induced relation $\mathcal{R}(\mathcal{C}_{Bis}^{Cor}, SCT)$ is indeed a bisimulation relation (see Definition 22): the invariance of the pair constraint $\mathcal{C}_{Pair}^{Cor}$ for the pair rules enforces that each application of a rule in the source model (i.e., a transition in the labeled transition system $LTS(gts_s, S)$) is followed by an equivalent and corresponding rule in the target model, and vice versa.

This is exemplified by the second arrow in the sketch below ($SCT \Rightarrow^*_{\mathcal{P}(l_s,l_t)^{Cor}}$). □



Example 19 (Equivalence: Verification) By Theorem 1, the model transformation described in Example 17 can be shown to be behavior preserving via the bisimulation relation induced by the bisimulation constraint $\mathcal{C}_{Bis}$.

Specifically, the forward part of the pair constraint ($\mathcal{C}^{Cor,f}_{Pair}$) of our example enforces two communicating and enabled transitions for each corresponding and executable message that connects two lifelines and their respective events. $\mathcal{C}^{Cor,f}_{Pair}$ also enforces the existence of a corresponding initial and unactivated state for each initializable lifeline event. Conversely, the backward part $\mathcal{C}^{Cor,b}_{Pair}$ establishes pairwise applicability in reverse direction, i.e., from target to source.

Summarizing, after having formalized all modeling steps of our behavior preservation problem in the previous sections as summarized in Table 1 we have now established a verification scheme that can be used to show on the transformation level as illustrated in Fig. 1 that a model transformation is behavior preserving in an equivalent manner.

4.2 Behavioral refinement

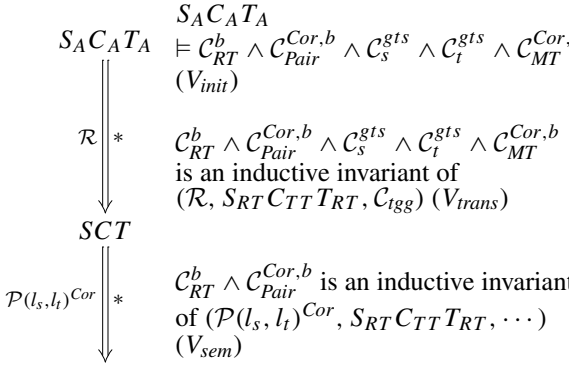
For the case of behavioral refinement, our verification scheme is required to show that the desired simulation relation as given in the previous section indeed leads to simulation of the semantics of each target model by the semantics of each source model of the model transformation. In particular, it 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. We formally define our scheme and proof its correctness in the following theorem.

Theorem 2 (refinement verification for forward transformation) *Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .), LTS(gts_t, .))$ *as introduced in Definition 24. Further, 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* $LTS(gts_s, .)$ *and* $LTS(gts_t, .)$ *derived from* $l^{\mathcal{R}}_s : \mathcal{R}_s \to A$ *and* $l^{\mathcal{R}}_t : \mathcal{R}_t \to A$ *as given in Definition 29, and a simulation constraint* $\mathcal{C}^{Cor,b}_{Sim} = \mathcal{C}^b_{RT} \wedge \mathcal{C}^{Cor,b}_{Pair} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}^{Cor,b}_{MT}$ *typed over* $S_{RT}C_{TT}T_{RT}$ *as given in Definition 35, then* $MT(tgg, \mathcal{C}_{tgg})$ *is* behavior preserving *in a refining manner (in particular, via the induced simulation relation* $\mathcal{R}(\mathcal{C}^{Cor,b}_{Sim}, .)^{-1}$*) in the sense of case 2 of Definition 25 if the following conditions are fulfilled:*

- V_{init}: $S_AC_AT_A \models \mathcal{C}^b_{RT} \wedge \mathcal{C}^{Cor,b}_{Pair} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}^{Cor,b}_{MT}$.
- V_{trans}: $\mathcal{C}^b_{RT} \wedge \mathcal{C}^{Cor,b}_{Pair} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t \wedge \mathcal{C}^{Cor,b}_{MT}$ *is an inductive invariant of* $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg})$.
- V_{sem}: $\mathcal{C}^b_{RT} \wedge \mathcal{C}^{Cor,b}_{Pair}$ *is an inductive invariant of* $(\mathcal{P}(l_s,l_t)^{Cor}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg} \wedge \mathcal{C}^{Cor,b}_{MT} \wedge \mathcal{C}^{gts}_s \wedge \mathcal{C}^{gts}_t)$ *with* $\mathcal{P}(l_s,l_t)^{Cor}$ *as given in Definition 28.*

Proof idea The detailed proof [13] consists of two parts.

In the first part, we show by induction that all model transformation instances $(S, T) \in MT(tgg, \mathcal{C}_{tgg})$ are contained in the induced relation $\mathcal{R}(\mathcal{C}^{Cor,b}_{Sim}, SCT)^{-1}$. This follows since $\mathcal{C}^{Cor,b}_{Sim}$ holds for the triple graph grammar's axiom $S_AC_AT_A$ (V_{init}, base of induction) and since $\mathcal{C}^{Cor,b}_{Sim}$ is an inductive invariant for the triple graph grammar with constraint $(\mathcal{R}, S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg})$ (V_{trans}, inductive step). This first part of the proof is depicted by the first arrow ($S_AC_AT_A \Rightarrow^*_{\mathcal{R}} SCT$) in the sketch below.



In the second part of the proof we show that the induced relation $\mathcal{R}(\mathcal{C}^{Cor,b}_{Sim}, SCT)^{-1}$ is indeed a simulation relation (see Definition 23): the invariance of the pair constraint $\mathcal{C}^{Cor,b}_{Pair}$ for the pair rules enforces that each application of a rule in the target model (i.e., a transition in the labeled transition system $LTS(gts_t, T)$) is followed by an equivalent and corresponding rule in the source model. This is exemplified by the second arrow in the sketch above ($SCT \Rightarrow^*_{\mathcal{P}(l_s,l_t)^{Cor}}$). □

Corollary 1 *(refinement verification for backward transformation) Given a formally covered model transformation* $(\mathcal{L}(S_{TT}, \mathcal{C}_S), \mathcal{L}(T_{TT}, \mathcal{C}_T), MT(tgg, \mathcal{C}_{tgg}), LTS(gts_s, .),$

$LTS(gts_t, .)$) as introduced in Definition 24. Further, 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 $LTS(gts_s, .)$ and $LTS(gts_t, .)$ derived from $l_s^{\mathcal{R}} : \mathcal{R}_s \to A$ and $l_t^{\mathcal{R}} : \mathcal{R}_t \to A$ as given in Definition 29, and a simulation constraint $\mathcal{C}_{Sim}^{Cor,f} = \mathcal{C}_{RT}^{f} \wedge \mathcal{C}_{Pair}^{Cor,f} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor,f}$ typed over $S_{RT}C_{TT}T_{RT}$ as given in Definition 35, then the backward *model transformation $MT(tgg, \mathcal{C}_{tgg})^{-1} : \mathcal{L}(T_{TT}, \mathcal{C}_T) \times \mathcal{L}(S_{TT}, \mathcal{C}_S)$ is* behavior preserving *in a refining manner (in particular, via the relation $\mathcal{R}(\mathcal{C}_{Sim}^{Cor,f}, .)$) in the sense of case 2 of Definition 25 if the following conditions are fulfilled:*

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

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

V_{sem}: $\mathcal{C}_{RT}^{f} \wedge \mathcal{C}_{Pair}^{Cor,f}$ *is an inductive invariant of* $(\mathcal{P}(l_s, l_t)^{Cor}), S_{RT}C_{TT}T_{RT}, \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor,f} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts})$ *with $\mathcal{P}(l_s, l_t)^{Cor}$ as given in Definition 28.*

Proof The claimed result follows from Theorem 2 and the symmetry of the TGG and the constructed constraints. □

Remark 6 (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 transformation derived from some TGG is behavior preserving in a refining manner, then the corresponding backward transformation represents an abstraction—and vice versa.

Example 20 (Refinement verification for backward transformation) By Corollary 1, the (backward) model transformation described in Example 18 can be shown to be behavior preserving in a refining manner via the simulation relation induced by the simulation constraint $\mathcal{C}_{Sim}^{Cor,f}$.

Specifically, the forward pair constraint ($\mathcal{C}_{Pair}^{Cor,f}$) of our example enforces two communicating and enabled transitions for each corresponding executable message that connects two lifelines and their respective events. $\mathcal{C}_{Pair}^{Cor,f}$ also enforces the existence of a corresponding initial and unactivated state for each initializable lifeline event. Conversely, as opposed to behavioral equivalence, the backward pair constraint $\mathcal{C}_{Pair}^{Cor,b}$ does not need to be considered. In fact, given additional internal behavior in the target model (the systems of communicating automata), $\mathcal{C}_{Pair}^{Cor,b}$ is not an inductive invariant of $(\mathcal{P}(l_s, l_t)^{Cor}, \ldots)$ which would be required by V_{sem} of the verification scheme for behavioral equivalence.

Summarizing, after having formalized all modeling steps of our behavior preservation problem in the previous sections as summarized in Table 1 we have now established a verification scheme that can be used to show on the transformation level as illustrated in Fig. 1 that a model transformation is behavior preserving in a refining manner.

5 Application

This section iterates over the different steps of our modeling and verification schemes, discusses automation of those steps at a general level, and presents results for our examples for behavioral equivalence (Examples 17 and 19) and refinement (Examples 18 and 20).

5.1 Automation

Our modeling and verification schemes for behavioral equivalence and behavioral refinement share a number of common elements. Thus, we will discuss automation on a generic level applicable to both cases and examples. All steps of our generic modeling scheme (see Fig. 1 and Table 1), including the respective artifacts to be established by them, are listed below:

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): $LTS(gts_s, \cdot)$, based on runtime-conforming $gts_s = (\mathcal{R}_s, S_{RT})$ and dynamic constraint $\mathcal{C}_s^{gts}$ and $LTS(gts_t, \cdot)$, based on runtime-conforming $gts_t = (\mathcal{R}_t, T_{RT})$ and dynamic constraint $\mathcal{C}_t^{gts}$.

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

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

In M_{lang}, we expect that the source and target modeling languages have been established manually by specifying the respective type graphs and constraints.

Similarly, we expect existing (manual) specifications of model semantics (M_{sem}), i.e., of graph transformation systems gts_s and gts_t and dynamic constraints for source and target models. The runtime conformity of gts_s and gts_t with respect to their runtime graph language can be verified automatically by tools capable to perform the required checks. Automatic 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 $gts_s^{\mathcal{C}_S}$ and $gts_t^{\mathcal{C}_T}$. This capability of performing inductive invariant checks will also be required for the automation of our verification scheme discussed below.

M_{trans} includes manual specification of the model transformation and the TGG constraint $\mathcal{C}_{tgg}$. We require $\mathcal{C}_{tgg}$ to comprise both $\mathcal{C}_S$ and C_T. We can therefore assume that $\mathcal{C}_{tgg}$ is automatically extended with $\mathcal{C}_S$ and C_T to fulfill this condition.

For the formalization and artifacts of step M_{pres}, 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. Both remappings and correspondences are to be specified manually. In general, those correspondences might 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 $C_{(\rho_s,\rho_t)}$.

However, we also believe it is reasonable to leave the specification of correspondences between source and target rules to the user: in order to verify behavior preservation, we require the user to specify his or her notion of how source and target semantics rules map to each other and how source and target model elements in those rules relate to each other by correspondences. Given that information, pair rules with correspondences and the pair constraint can be derived automatically.

Additionally, we require the user to specify the runtime constraint $\mathcal{C}_{RT}$ in order to further restrict which states should be considered equivalent. We think it is reasonable to characterize equivalent behavior not only based on the level of pairwise rule application, but also based on a user-chosen comparison of states and corresponding elements; for instance, our example constraint requires corresponding events and states to share the same status with respect to being active or not.

With respect to the model transformation constraint $\mathcal{C}_{MT}^{Cor\square}$, automation is a challenging issue. Due to the highly flexible nature of the pair constraint (derived from the semantics rules) and the expressive power of nested application conditions, an appropriate automatic derivation for the model transformation constraint is difficult to establish for the general case. However, given the restriction to rules' application conditions with respect to their runtime conformity, namely the absence of additional statically typed elements, we have established an automatic derivation (cf. Definition 33). If the model transformation constraint is instead specified manually, this restriction can be dropped.

For both equivalence and refinement as discussed in this article, the proposed verification scheme consists of the three steps described by Theorems 1 and 2 and Corollary 1, respectively. All involved elements can be represented in a generic way for demonstration purposes as follows:

V_{init}: $S_A C_A T_A \vDash \mathcal{C}_{RT}^{\square} \wedge \mathcal{C}_{Pair}^{Cor\square} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{MT}^{Cor\square}$.
V_{trans}: $\mathcal{C}_{RT}^{\square} \wedge \mathcal{C}_{Pair}^{Cor\square} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts} \wedge \mathcal{C}_{MT}^{Cor\square}$ is an inductive invariant of $(\mathcal{R}, S_{RT} C_{TT} T_{RT}, \mathcal{C}_{tgg})$.
V_{sem}: $\mathcal{C}_{RT}^{\square} \wedge \mathcal{C}_{Pair}^{Cor\square}$ is an inductive invariant of $(\mathcal{P}(l_s, l_t)^{Cor}, S_{RT} C_{TT} T_{RT}, \mathcal{C}_{tgg} \wedge \mathcal{C}_{MT}^{Cor\square} \wedge \mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts})$.

Since the axiom $S_A C_A T_A$ is just a specific typed graph, the first check (V_{init}) can be done by most graph transformation tools that support typed graphs and satisfiability checks of graph constraints for a particular typed graph. This type of check is usually not computationally challenging. In addition, 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 nested conditions in rules or constraints, the related verification problem is undecidable [32], since it can be reduced to the implication problem for first-order logic formulas on graphs. 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 work related to our verification approach [12], 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 [38], Augur [37], RAVEN [8], and the model checking approaches by Steenken [52] and Boneva et. al. [9]. While the approach of Habel and Pennemann [32,49] is expressive enough, their corresponding tool does not yield results for our examples in reasonable time. While our own tool [4,12] does, it requires special considerations and simplifications (see Sect. 5.2 below) to specifically cope with the pair constraints in our examples.

As established in discussions of our verification tool [12], it may yield false negatives in the form of spurious counterexamples. This comes as a result of employing a symbolic encoding representing (possibly) infinitely many cases in a finite fashion, which is usually necessary for verification approaches working at the transformation level. Depending on the transformation and semantics in question, it may be possible to manually enhance or add to the input artifacts such

that the verification succeeds. However, since the underlying verification problem is undecidable in general [32], the possibility of false negatives is a compromise to ensure termination of our verification tasks. In any case, symbolic counterexamples produced by our tool can be inspected by hand in order to identify them as spurious or not.

5.2 Evaluation

After our generic discussion about possible automation of our modeling and verification schemes, this section now explains and evaluates the specific modeling and verification steps of our two examples. For the verification part, we employed our own invariant checking tool [4,12]. While we have discussed—in Sect. 5.1—that several steps in our modeling and verification schemes can be automated, we do not provide a fully functional toolchain. With respect to automation, our focus in the application of our approach to the example lies on the second and third step of the verification scheme.

Table 4 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 artifacts to be specified in the modeling scheme, the resulting effort required for their manual specification and comprehension, and the effect on verification times.

Considering the manual specification effort, the high number of fragments in the TGG constraint stands out. All those fragments implement cardinality restrictions similar to the one shown in Fig. 9c. In a different context, those constraints could also be encoded directly in the type graph. Our approach to TGGs and the corresponding verification tool [4,12] does not support cardinalities as part of a type graph. Hence, we require their explicit representation as graph constraints. However, in principle, the constraints could be generated automatically from cardinalities in a type graph.

All verification tasks were conducted on a machine with two cores at 2.8 GHz and 8 GB of main memory while running Windows 7, Java 8 with a heap space limit of 1 GB, and our tool as an Eclipse (4.5.1) plugin. In comparison with earlier evaluations of our example [12,13], we used a newer and more efficient version of our tool.

The initial verification step for the TGG rules (V_{init}) did not require any automatic verification: since both cases have an empty graph as the axiom and since, because of their structure, all constraints require the existence of specific elements to be violated, the axioms trivially satisfy the respective constraints.

The inductive invariant checks for the TGG rules (V_{trans}) and pair rules (V_{sem}) can be performed automatically by our invariant checking tool [4,12], which has also already been applied for the more restricted case of simple equivalence described in our first approach [24]. Similarly, runtime conformity with respect to dynamic constraints was also checked. However, the rules and constraints that can be verified 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 $\rho = (\langle L \hookleftarrow I \hookrightarrow R, \rangle, \bigwedge_{i\in I} \neg\exists n_i)$ with $n_i : L \hookrightarrow N_i$ a morphism.
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 $\mathcal{C}^{Cor\square}_{Pair}$ and hence, for the pair rules $\mathcal{P}(l_s, l_t)^{Cor}$ and for the source and target graph transformation systems gts_s and gts_t. Since the nesting level in the pair constraint is restricted, the rules in gts_s and gts_t may only have the trivial condition true as nested application conditions. If, however, any negative application conditions of the form $\neg\exists a$ exist and if these conditions only prohibit the existence of one edge, they can be emulated by additional constructions as explained in Appendix A of our technical report [13]. This is, in fact, the case for the $initE$ and $initS$ rules of both examples and the scheme is applied for all the verification steps involved.

This restriction and the resulting triviality of nested application conditions in semantics rules leads to simple $\forall(P, \exists N)$ fragments for our pair constraints and by definition, for the model transformation constraint, which can be derived automatically from the pair constraint for both behavioral equivalence and refinement (Examples 17 and 18).

We also employ a simplification scheme to replace one-to-one correspondence nodes and connected edges by simple edges as explained in Appendix A of our technical report [13]. This lowers the size of rules and constraints in our examples considerably and allows for a verification in reasonable time and memory consumption.

While our tool requires 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 (see Table 4). For both behavioral equivalence and refinement, V_{trans} is the costlier verification step, with refinement verification requiring much more time than equivalence verification. This difference results from the higher number of TGG rules for the refinement example. Two additional TGG rules for the model transformation allow automata to have internal behavior: communicating transitions that do not have a linked message counterpart in the corresponding sequence chart.

Table 4 Evaluation data with numbers of subcomponents and maximum node size for artifacts involved in modeling and verification schemes

Modeling step/artefact		Equivalence (Example 17)		Refinement (Example 18)	
		No. of subcomponents	Max. size	No. of subcomponents	Max. size
M_{lang}	S_{TT}	1	3	1	3
	$\mathcal{C}_S$	11	3	11	3
	T_{TT}	1	4	1	4
	$\mathcal{C}_T$	8	3	6	3
M_{sem}	$\mathcal{R}_s$	2	7	2	7
	S_{RT}	1	3	1	3
	$\mathcal{C}_s^{gts}$	6	3	6	3
	$\mathcal{R}_t$	2	9	2	9
	T_{RT}	1	4	1	4
	$\mathcal{C}_t^{gts}$	6	3	6	3
M_{trans}	$\mathcal{R}$	2	16	4	16
	$S_{RT}C_{TT}T_{RT}$	1	7	1	7
	$\mathcal{C}_{tgg}$	28	3	26	3
M_{pres}	$C_{(\rho_s,\rho_t)}$	2	0	2	0
	$\mathcal{C}_{RT}^{\square}$	2	2	1	2
M_{pres} (derived)	$\mathcal{P}(l_s,l_t)^{Cor}$	2	16	2	16
	$\mathcal{C}_{Pair}^{Cor\square}$	4	16	2	16
	$\mathcal{C}_{MT}^{Cor\square}$	4	16	2	16
Verification step		Result	Time	Result	Time
V_{init}		True	Trivial	True	Trivial
V_{trans}		True	103 s	True	870 s
V_{sem}		True	< 1 s	True	< 1 s

When comparing node and edge types, these rules share all elements with the larger fragment of the model transformation constraint, whose verification is the most complex part of the verification step V_{trans}. As a result, verification of our example for behavioral refinement takes much more time.

Verification tools of higher expressive power may struggle with performance for the verification of behavioral equivalence and refinement in systems of comparable complexity [12]. In contrast to interactive verification, our tool has the advantage of running completely automated. Finally, if our examples were to fail the verification steps, the tool would yield symbolic counterexamples, which could be investigated by hand to find the reason for the failure of the respective verification step.

6 Discussion

In the following, we will discuss the applicability of our approach and results for cases beyond the examples used in this article.

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 (modeling step M_{lang}) concerning the definition of the modeling languages, we chose typed graphs enriched with a constraint as formalization. The expressiveness of nested graph constraints is equivalent to first-order logic [32,49]; additionally, there is work on converting OCL invariants to nested graph constraints [2]. Consequently, we believe that the idea of type graphs with graph constraints fits well to commonly used meta-models. While we omit, e.g., attributes and inheritance, there exist extensions for typed graphs covering attributes [15], inheritance for node types [29,39] and edge types [56], and other concepts. Given a formal approach for modeling

attribute computations and conditions with a constraint-based approach (as in the work of Orejas and Lambers [48]), we expect that conceptually our verification algorithm based on invariant checking can be generalized. From a practical perspective, the invariant checking tooling then needs to be able to cope with attributes accordingly. Basic ideas and a prototypical implementation given certain restrictions have been described [46].

Model semantics according to Definition 2 (modeling step M_{sem}) 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 related work [18,33]). 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.

Our formalization requires that each static graph trivially satisfies each dynamic constraint: $\mathcal{L}(TG, \mathcal{C}) \subseteq \mathcal{L}(TG', \mathcal{C} \wedge \mathcal{C}_{dyn})$. However, the established and proven verification scheme can be 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 $S_A C_A T_A \vDash \mathcal{C}_{RT}^{\square} \wedge \mathcal{C}_{Pair}^{Cor\square} \wedge \mathcal{C}_{MT}^{Cor\square}$ and a check that the initialization phase then preserves all the other constraints $S_A C_A T_A \vDash \mathcal{C}_{RT}^{\square} \wedge \mathcal{C}_{Pair}^{Cor\square} \wedge \mathcal{C}_{MT}^{Cor\square}$ and guarantees the dynamic constraint $\mathcal{C}_s^{gts} \wedge \mathcal{C}_t^{gts}$ after termination.

The restriction of application conditions in semantics rules to additional dynamic elements only (see runtime conformity, Definition 16) is directly linked to the problem of automatically deriving the model transformation constraint. If this limitation is dropped, the model transformation constraint cannot be derived as in Definition 33—or if it still is, the resulting constraint may be too strict. Depending on whether automation of the model transformation constraint (instead of manual specification) or more expressive application conditions for the model semantics are more important, a decision can be made on a case-by-case basis.

For the formalization of model transformations (Definition 3, modeling step M_{trans}), we chose triple graph grammars (TGGs) enriched with graph constraints. In earlier work [23,34], we showed for TGGs enriched with a constraint that they conform to our TGG implementation [25]. Thus we can guarantee for that implementation that forward and backward transformation implementations are indeed behavior preserving.

Relational model transformations usually (explicitly or implicitly) keep track between the source model and transformed elements in the target model to properly execute transformations. TGGs use correspondences to represent that relation in the transformation definition itself; using those correspondences is central to our approach and handling of non-deterministic semantics. The explicit representation of source-target relations is a defining feature of relational model transformation approaches. Hence, for relational model transformation, we expect to find suitable concepts to be transformed to or exploited similarly to correspondences as described in our approach. In particular, there is work describing similarities between TGGs and QVT. While QVT Core can be transformed to TGG rules [30], this may not be the case for QVT Relational [54]. Consequently, we can assume that our results conceptually cover a segment of the class of relational model transformations.

Our results can also be adapted to the case of operational model transformations [14]. In this case the verification part for checking bisimulation or simulation (V_{sem}) can be reused after the establishment of condition $\mathcal{C}_{MT}^{Cor\square}$ for all model transformation instances by other means. However, for operational model transformations, fragments of the bisimulation constraint that require the existence of dynamic elements might not hold for a source and target model on which no semantics rule has been applied yet and hence, where no dynamic elements exist. In this case, an initialization phase $V_{sem,init}$ for the semantics could be a possible workaround; during that initialization, the required dynamic elements would be established. After that the regular verification part described in V_{sem} can be applied.

As outlined, from a theoretical perspective on modeling languages, model semantics, and model transformations, the limitations of the considered variant of graph transformation systems are not a limiting factor concerning expressiveness of the specifications necessary in the modeling steps of our approach, as the results also apply for extensions such as attributes, inheritance for node types and edge types, other extensions, and the general concept of $\mathcal{M}$-adhesive replacement systems [16]. However, our results concerning automatic checking are only feasible for models of limited expressiveness.

The choice of a particular behavioral equivalence or refinement (Definition 5, modeling step M_{pres}) is different in that we do not only chose a particular formalization but also have to select one option from a large number of alternative notions of equivalences and refinement for labeled transition systems that have been studied in the literature [26,27].

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. In contrast to that, other notions consider traces and/or refusals to establish the comparison on a more abstract level and avoid taking into account the state space of the two models involved [26,27]. 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 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.

Within the class of behavioral equivalence and refinement where the comparison is based on pairs of states a number of alternatives to bisimulation and simulation exist. We discuss alternative forms such as weak bisimulation/simulation, simulation without bijective relabelings, and *ready simulation* in Section 8 of our technical report [13].

7 Conclusion and future work

We present the first automatic and sound verification approach for behavior preservation of model transformations at the transformation level. For model transformations specified by TGGs and semantic definitions for the input models and output models given by GTSs, we were able to reduce the behavior preservation problem to an inductive invariant checking problem of nested graph conditions for GTSs. In particular, the GTSs are derived from the TGG rules and semantics rules, respectively; graph conditions encode bisimulation or simulation and the applicability of equivalent steps in the source and target models. Given manual specification of the modeling languages, model semantics, model transformation, and of what constitutes equivalent (or refining) behavior, the verification phase can be performed automatically.

We have shown that our approach is applicable to an example transformation between sequence charts and communicating automata, which preserves behavior in an equivalent manner, and to a slightly modified transformation, which preserves behavior in a refining manner. In particular, the verification phase of those examples can be performed automatically by our verification technique concerned with inductive invariants for graph transformation systems.

Given the expressiveness of the formal concepts employed, we believe that our formalization of the behavior preservation problem—consisting of the different steps of our modeling scheme—is applicable to similar examples. In comparison with our earlier work [24], we expect our extensions—non-deterministic semantics and refinement in addition to equivalence—to widen the range of examples that can be covered with our approach. However, in the absence of a sufficient number of case studies, we cannot confirm or disprove that. Unfortunately, applicability is not necessarily given when it comes to executing the verification scheme. While we have proven (Theorem 1) that the verification scheme is sound given our formalization of the behavior preservation problem and while there exist specific tools capable of automatically executing the steps of our verification scheme given limited [4,12] and—with respect to graph constraints—more general expressiveness [49], the required expressive power [4,12] or the undecidability and exponential complexity [49] of the underlying problems makes verification infeasible for larger examples.

In future work, we plan to investigate larger examples and case studies and identify which characteristics may prevent a successfull application of our approach. We would like to operationalize these insights to extend our approach's applicability to more cases. Directions may be more expressive models, different kind of semantics, or alternative notions for behavior equivalence and refinement. Other extensions of the scope of our approach might be dealing not only with operational model transformations [14] but also hybrid ones.

Acknowledgements We would 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 results in this article in a more comprehensible manner. Furthermore, we are grateful to the anonymous reviewers for their helpful and detailed comments.

References

1. Ab Rahim, L., Whittle, J.: Verifying semantic conformance of state machine-to-java code generators. In: 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10), pp. 166–180 (2010)
2. Arendt, T., Habel, A., Radke, H., Taentzer, G.: From core OCL invariants to nested graph constraints. In: Giese, H., König, B. (eds.) Graph Transformation. Lecture Notes in Computer Science, vol. 8571, pp. 97–112. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09108-2_7
3. Barroca, B., Amaral, V., Buchs, D.: Semantic languages for developing correct language translations. Softw. Qual. J. **26**, 417–453 (2017)
4. Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic invariant verification for systems with dynamic structural adaptation. In: Proceedings of the 28th International Conference on Software Engineering, pp. 72–81. ACM, New York (2006)
5. Becker, B., Lambers, L., Dyck, J., Birth, S., Giese, H.: Iterative development of consistency-preserving rule-based refactorings. In: Cabot, J., Visser, E. (eds.) Theory and Practice of Model Transformations. Lecture Notes in Computer Science, vol. 6707, pp. 123–137. Springer, Berlin (2011)
6. Bezivin, J., Dupe, G., Jouault, F., Pitette, G., Rougui, J.E.: First experiments with the ATL model transformation language: transforming XSLT into XQuery. In: 2nd OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture (2003)

7. Bisztray, D., Heckel, R., Ehrig, H.: Compositional verification of architectural refactorings. In: de Lemos, R., Fabre, J.C., Gacek, C., ter Beek, M. (eds.) Architecting Dependable Systems VI. Lecture Notes in Computer Science, vol. 5835, chap. 13, pp. 308–333. Springer, Berlin (2009)
8. Blume, C., Bruggink, H.J.S., Engelke, D., Knig, B.: Efficient symbolic implementation of graph automata with applications to invariant checking. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Graph Transformations. Lecture Notes in Computer Science, vol. 7562, pp. 264–278. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-33654-6_18
9. Boneva, I.B., Kreiker, J., Kurban, M.E., Rensink, A., Zambon, E.: 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)
10. Charpentier, M.: Composing invariants. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003: Formal Methods. Lecture Notes in Computer Science, vol. 2805, pp. 401–421. Springer, Berlin (2003)
11. Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D.: VIATRA—visual automated transformations for formal verification and validation of UML models. In: Richardson, J., Emmerich, W., Wile, D. (eds.) ASE 2002: 17th IEEE International Conference on Automated Software Engineering, pp. 267–270. IEEE Press (2002)
12. Dyck, J., Giese, H.: Inductive invariant checking with partial negative application conditions. In: Parisi-Presicce, F., Westfechtel, B. (eds.) Graph Transformation. Lecture Notes in Computer Science, vol. 9151, pp. 237–253. Springer, Cham (2015)
13. Dyck, J., Giese, H., Lambers, L.: Automatic verification of behavior preservation at the transformation level for relational model transformation. Technical Report 112, Hasso Plattner Institute, University of Potsdam (2017)
14. Dyck, J., Giese, H., Lambers, L., Schlesinger, S., Glesner, S.: Towards the automatic verification of behavior preservation at the transformation level for operational model transformations. In: Dingel, J., Kokaly, S., Lúcio, L., Salay, R., Vangheluwe, H. (eds.) Analysis of Model Transformations. CEUR Workshop Proceedings, vol. 1500, pp. 36–45 (2015). http://ceur-ws.org/Vol-1500/paper5.pdf
15. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, New York (2006)
16. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: $\mathcal{M}$-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation. Math. Struct. Comput. Sci. **24**, 1–48 (2014)
17. Ehrig, H., Habel, A., Lambers, L.: Parallelism and concurrency theorems for rules with nested application conditions. Electron. Commun. EASST **26** (2010). http://journal.ub.tu-berlin.de/index.php/eceasst/article/viewFile/363/333
18. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000—The Unified Modeling Language. Lecture Notes in Computer Science, vol. 1939, pp. 323–337. Springer, Berlin (2000)
19. Engels, G., Kleppe, A., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: From UML activities to TAAL—towards behaviour-preserving model transformations. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture—Foundations and Applications. Lecture Notes in Computer Science, vol. 5095, pp. 94–109. Springer, Berlin (2008)
20. Ermel, C., Gall, J., Lambers, L., Taentzer, G.: Modeling with plausibility checking: inspecting favorable and critical signs for consistency between control flow and functional behavior. In: Giannakopoulou, D., Orejas, F. (eds.) Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, vol. 6603, pp. 156–170. Springer, Berlin (2011)
21. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: a new graph rewrite language based on the unified modeling language and java. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Theory and Application of Graph Transformations. Lecture Notes in Computer Science, vol. 1764, pp. 296–309. Springer, Berlin (2000)
22. Giese, H., Glesner, S., Leitner, J., Schäfer, W., Wagner, R.: Towards verified model transformations. In: Hearnden, D., Süß, J.G., Baudry, B., Rapin, N. (eds.) Proceedings of the 3rd International Workshop on Model Development, Validation and Verification (MoDeV^{2}a), Genova, Italy, pp. 78–93. Le Commissariat à l'Energie Atomique (2006)
23. Giese, H., Hildebrandt, S., Lambers, L.: Bridging the gap between formal semantics and implementation of triple graph grammars—ensuring conformance of relational model transformation specifications and implementations. Softw. Syst. Model. **13**(1), 273–299 (2014). https://doi.org/10.1007/s10270-012-0247-y
24. Giese, H., Lambers, L.: Towards automatic verification of behavior preservation for model transformation via invariant checking. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Graph Transformations. Lecture Notes in Computer Science, vol. 7562, pp. 249–263. Springer, Berlin (2012)
25. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. **8**(1), 21–43 (2009). https://doi.org/10.1007/s10270-008-0089-9
26. van Glabbeek, R.J.: The linear time—branching time spectrum. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR'90, Theories of Concurrency: Unification and Extension. Lecture Notes in Computer Science, vol. 458, pp. 278–297. Springer, Berlin (1990)
27. van Glabbeek, R.J.: The linear time—branching time spectrum II. The semantics of sequential systems with silent moves. In: Best, E. (ed.) CONCUR'93. Lecture Notes in Computer Science, vol. 715, pp. 66–81. Springer, Berlin (1993)
28. Golas, U., Lambers, L., Ehrig, H., Giese, H.: Toward bridging the gap between formal foundations and current practice for triple graph grammars. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Graph Transformations. Lecture Notes in Computer Science, vol. 7562, pp. 141–155. Springer, Berlin (2012)
29. Golas, U., Lambers, L., Ehrig, H., Orejas, F.: Attributed graph transformation with inheritance: efficient conflict detection and local confluence analysis using abstract critical pairs. Theor. Comput. Sci. **424**, 46–68 (2012)
30. Greenyer, J., Kindler, E.: Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. Softw. Syst. Model. **9**(1), 21–46 (2010)
31. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundam. Inform. **26**(3/4), 287–313 (1996)
32. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**, 245–296 (2009)
33. Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling with time: specifying the semantics of multimedia sequence diagrams. Softw. Syst. Model. **3**(3), 181–193 (2004)
34. Hildebrandt, S., Lambers, L., Becker, B., Giese, H.: Integration of triple graph grammars and constraints. Electron. Commun. EASST **54**, 1–12 (2012)
35. Hülsbusch, M., König, B., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: Showing full semantics preservation in model transformation—a comparison of techniques. In: Méry, D., Merz,

S. (eds.) Integrated Formal Methods. Lecture Notes in Computer Science, vol. 6396, pp. 183–198. Springer, Berlin (2010)
36. Karsai, G., Agrawal, A., Shi, F., Sprinkle, J.: On the use of graph transformation in the formal specification of model interpreters. J. Univ. Comput. Sci. **9**(11), 1296–1321 (2003)
37. König, B., Kozioura, V.: Augur 2—a new version of a tool for the analysis of graph transformation systems. Electron. Notes Theor. Comput. Sci. **211**, 201–210 (2008)
38. König, B., Stückrath, J.: A general framework for well-structured graph transformation systems. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014—Concurrency Theory. Lecture Notes in Computer Science, vol. 8704, pp. 467–481. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-44584-6_32
39. de Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed graph transformation with node type inheritance. Theor. Comput. Sci. **376**(3), 139–163 (2007). https://doi.org/10.1016/j.tcs.2007.02.001
40. de Lara, J., Taentzer, G.: Automated model transformation and its validation using AToM 3 and AGG. In: Blackwell, A.F., Marriott, K., Shimojima, A. (eds.) Diagrammatic Representation and Inference. Lecture Notes in Computer Science, vol. 2980, pp. 182–198. Springer, Berlin (2004)
41. Lawley, M., Steel, J.: Practical declarative model transformation with Tefkat. In: Bruel, J.M. (ed.) Satellite Events at the MoDELS 2005 Conference, Lecture Notes in Computer Science, vol. 3844, pp. 139–150. Springer, Berlin (2006)
42. Lúcio, L., Barroca, B., Amaral, V.: A technique for automatic validation of model transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) Model Driven Engineering Languages and Systems. LNCS, vol. 6394, pp. 136–150. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-16145-2_10
43. Milner, R.: Communication and Concurrency. Prentice Hall, Hertfordshire (1995)
44. Narayanan, A., Karsai, G.: Towards verifying model transformations. Electron. Notes Theor. Comput. Sci. **211**, 191–200 (2008). https://doi.org/10.1016/j.entcs.2008.04.041
45. Narayanan, A., Karsai, G.: Verifying model transformations by structural correspondence. Electron. Commun. EASST **10**, 1–14 (2008)
46. Nicolai, C.: Using exchangeable constraint solvers for invariant checking on attributed graph transformation systems. Master's thesis, Hasso-Plattner-Institut fr Softwaresystemtechnik, Universitt Potsdam (2016)
47. OMG: MOF QVT Final Adopted Specification, OMG Document ptc/05-11-01 (2005)
48. Orejas, F., Lambers, L.: Lazy graph transformation. Fundam. Inform. **118**(1–2), 65–96 (2012)
49. Pennemann, K.H.: Development of correct graph transformation systems. Ph.D. thesis, University of Oldenburg (2009)
50. Rangel, G., Lambers, L., König, B., Ehrig, H., Baldan, P.: Behavior preservation in model refactoring using DPO transformations with borrowed contexts. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) Graph Transformations. Lecture Notes in Computer Science, vol. 5214, pp. 242–256. Springer, Berlin (2008)
51. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, vol. 903, pp. 151–163. Springer, Berlin (1995)
52. Steenken, D.: Verification of infinite-state graph transformation systems via abstraction. Ph.D. thesis, University of Paderborn (2015)
53. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Softw. Syst. Model. **9**, 7–20 (2010)
54. Stevens, P.: A simple game-theoretic approach to checkonly QVT relations. Softw. Syst. Model. **12**(1), 175–199 (2013). https://doi.org/10.1007/s10270-011-0198-8
55. Taentzer, G.: AGG: a tool environment for algebraic graph transformation. In: Nagl, M., Schürr, A., Münch, M. (eds.) Applications of Graph Transformation with Industrial Relevance. Lecture Notes in Computer Science, vol. 1779, pp. 481–488. Springer, Berlin (2000)
56. Taentzer, G., Rensink, A.: Ensuring structural constraints in graph-based models with type inheritance. In: Cerioli, M. (ed.) Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, vol. 3442, pp. 64–79. Springer, Berlin (2005)
57. Tiso, A., Reggio, G., Leotta, M.: Early experiences on model transformation testing. In: Proceedings of the First Workshop on the Analysis of Model Transformations, AMT '12, pp. 15–20. ACM, New York, NY (2012)
58. Varró, D., Pataricza, A.: Automated formal verification of model transformations. In: Jürjens, J., Rumpe, B., France, R., Fernandez, E.B. (eds.) CSDUML 2003: Critical Systems Development in UML, pp. 63–78. Technische Universität München (2003)

Johannes Dyck is a Ph.D. student at Hasso Plattner Institute at the University of Potsdam, where he also received his bachelor's and master's degree in IT Systems Engineering. He is part of the institute's System Analysis and Modeling group led by Prof. Holger Giese. His research interests include graph transformation systems and formal verification with a particular focus on invariant checking.

Holger Giese is a full professor at the Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam. Prior to that, he was assistant professor for object-oriented specification of distributed systems in the Software Engineering Group of the University of Paderborn. He studied Technical Computer Science at the University of Siegen and received his engineering degree in October 1995. He received a doctorate in Computer Science at the Institute of Computer Science at the University of Münster in February 2001. His research focus is the model-driven development of software-intensive systems covering the specification of dynamic and flexible systems by services, collaborations, patterns, and components, approaches to analyze and formally verify such models, and approaches for model synthesis. The main focus are systems that are typically distributed systems, embedded real-time systems as well as systems that are capable to adapt and coordinate themselves. Furthermore, he does research on model transformation, concepts for generating source code for structure and behavior, and the general problem of model integration during the process of model-driven development. He is a member of the Association for Computing Machinery, the IEEE Computer Society, and the German Informatics Society.

Leen Lambers Since July 2015, Leen Lambers is working as a senior researcher in the group System Analysis and Modeling of Prof. Holger Giese at the Hasso Plattner Institute at the University of Potsdam. She worked from January 2010 until June 2015 as a postdoc on the DFG-project "Correct Model Transformations" affiliated with the same research group. Her main research focus is formal modeling and analysis in software engineering, in particular, using graph transformation. She has spent research periods in the group of Prof. Mauro Pezzé at the University of Milano Bicocca and in the group of Prof. Fernando Orejas at the Technical University of Catalonia. She served as PC chair/member for several international workshops and conferences related to modeling and analysis in software engineering. She received a Ph.D. for her dissertation "Certifying Rule-Based Models using Graph Transformation" at the Technical University of Berlin in December 2009, where she has been a scientific assistant in the group of Prof. Hartmut Ehrig from October 2003 until December 2009.