Towards the Automatic Verification of Behavior Preservation at the Transformation Level for Operational Model Transformations[*]

Johannes Dyck[1], Holger Giese[1], Leen Lambers[1], Sebastian Schlesinger[2], and Sabine Glesner[2]

[1]System Analysis and Modeling Group, Hasso Plattner Institute at the University of Potsdam, Germany
[2]Software Engineering for Embedded Systems, Technical University of Berlin, Germany

Abstract

The correctness of model transformations and, in particular, behavior preservation is important for model-driven engineering of high quality software. Behavior preservation verification techniques have been presented with automatic tool support for the instance level, i.e. for a given source and target model specified by the model transformation. However, up until now there is no automatic verification approach available for operational model transformations at the transformation level, i.e. for all source and target models specified by an operational model transformation. In this paper, we outline a first approach towards the automatic verification of behavior preservation for operational model transformations at the transformation level extending our previous results for relational model transformations. In particular, we show that in restricted cases the behavior preservation problem for an operational model transformation can be reduced to invariant checking for graph transformation with priorities illustrated by a simple example.

1 Introduction

The correctness of model transformations and, in particular, *behavior preservation* is a crucial element for model-driven engineering of high quality software. In this paper we focus on behavior preservation for *operational model transformations* (cf. QVT operational [1]). Behavior preservation verification techniques either show that specific properties are preserved, or more generally and complex, they show some kind of behavioral equivalence (e.g., bisimulation $\approx_{bsim}$) between source and target model of the transformation. For both kinds of behavior preservation, verification goals have been presented with automatic tool support for the instance level [2, 3, 4], i.e. for a given source and target model specified by the model transformation. However, the development of the transformation and its application are separate activities and therefore detecting that the transformation is not correct during application is thus too late. Instead means for the automatic verification at the transformation level, i.e. for *all* source and target models specified by the model transformation, are required.

We presented a first approach [5] attacking this problem on the transformation level in a semi-automated manner in form of a verification technique based on interactive theorem proving for *relational model transformations* [6]. Based on this former result we developed an automatic verification approach at the transformation level for relational model transformations [7, 8], but up until now such an approach for operational model transformations is not available.

[*]This work was developed in the course of the project Correct Model Transformations II (GI 765/1-2) at Hasso Plattner Institute at the University of Potsdam and Technical University of Berlin and was funded by the Deutsche Forschungsgemeinschaft and published on its behalf. See `http://www.hpi.uni-potsdam.de/giese/projekte/kormoran.html?L=1` and `http://www.pes.tu-berlin.de/menue/forschung/projekte/kormoran/parameter/en/`.

The main result presented in this paper is a scheme for the automatic verification of behavior preservation at the transformation level for *operational model transformations* given in the form of a restricted *Story Diagram (SD)* [9]. In particular, the verification scheme employs graph transformation with priorities and inductive invariant checking [10, 11].[1] To this end we moreover present a formalization of restricted SDs with a corresponding mapping to graph transformation systems with priorities. Given a model transformation MT $\subseteq \mathcal{L}_S \times \mathcal{L}_T$ for a source $\mathcal{L}_S$ and target language $\mathcal{L}_T$, we require that correspondences between the source and target are captured leading to a model transformation with correspondences MTC $\subseteq \mathcal{L}_{SCT}$, where the added correspondence elements capture the traceability between source and target elements. Our *running example* describes a transformation from lifelines to automata. Fig. 1(a) shows the metamodel for this transformation with correspondences. The dashed edges denote dynamic elements of the source and target metamodel of the transformation.

The *verification scheme* presented in this paper shares some elements with our former work on checking behavior preservation for relational model transformations [7, 8] by bisimulation. The verification scheme there is based on a so-called *bisimulation constraint* $\mathcal{C}_{\mathrm{Bis}}$, being a sufficient condition for the membership of a source and target semantics state of a model transformation instance to an induced bisimulation relation, and it consists of two parts. (1) Checking the satisfaction of $\mathcal{C}_{\mathrm{Bis}}$ on all model transformation instances with correspondences gathered in MTC(REL) $\subseteq \mathcal{L}_{SCT}$ being conform to the relational model transformation REL. (2) Checking that based on the result in (1) all model transformation instances satisfying the bisimulation constraint $\mathcal{C}_{\mathrm{Bis}}$ have bisimilar semantics. For our running example this means informally that, (1) considers all possible model transformation instances consisting of a lifeline and corresponding automaton and (2) then compares the actual behavior of all these instances. In [7] a semantics definition for the lifeline/automaton is given based on graph transformation systems describing how dynamic elements may be changed in each runtime step of the behavior. In Fig. 2(a) an example lifeline and corresponding automaton are shown in abstract (top, left and right) and concrete (bottom) syntax with preserved behavior as described by their respective LTSs.

$$\forall\, \mathit{SCT} \in \mathrm{MTC(REL)} : \mathit{SCT} \vDash \mathcal{C}_{\mathrm{Bis}}\ (1) \qquad \forall\, \mathit{SCT} \vDash \mathcal{C}_{\mathrm{Bis}} : \mathit{sem}(S) \approx_{bsim} \mathit{sem}(T)\ (2).$$

In this work we have to adapt check (1) for all instances in MTC(REL) into a check (1*) for all instances in MTC(OP) conforming to our operational model transformation *OP*. In contrast check (2) can be reused as described in [7, 8], since it is independent from the model transformation type.

$$\forall\, \mathit{SCT} \in \mathrm{MTC(OP)} : \mathit{SCT} \vDash \mathcal{C}_{\mathrm{Bis}}\ (1^*) \qquad \forall\, \mathit{SCT} \vDash \mathcal{C}_{\mathrm{Bis}} : \mathit{sem}(S) \approx_{bsim} \mathit{sem}(T)\ (2).$$

The paper is structured as follows: In Section 2 we further describe our running example and explain our notion of operational model transformations. In Section 3, we reintroduce briefly all prerequisites for formalizing our notion of operational model transformations presented in Section 4 and the verification scheme presented in Section 5. This verification scheme is applied and evaluated by means of the running example in Section 6. We close the paper with some final conclusions and an outlook on future work.

2 Operational Model Transformations with Story Diagrams

We rely on Story Diagrams [9] as representatives for defining operational model transformations. We further restrict ourselves to model transformations describing outplace model transformations (i.e. transformations not changing the source model) with traceability information. The latter is encoded in a so-called correspondence model, storing traceability information explicitly between source and target model. The *metamodel* of a model transformation with traceability information MTC therefore consists of the following three parts: the source metamodel S_{TT} of the source modeling language $\mathcal{L}_S$ of the transformation linked with a correspondence metamodel C_{TT} for the correspondences to the target metamodel T_{TT} of the target modeling language $\mathcal{L}_T$. We moreover allow constraints to further restrict the metamodel.

Example 1 (Metamodel for Lifeline2Automaton)**.** *Fig. 1(a) shows the metamodel of our running example transformation from lifelines to automata. A lifeline describes the sending/receiving of messages on a timeline. It consists of* Event*s and one distinguished event is marked as* first *event. Events are connected with* Send *or* Rcv *objects. An automaton consists of* State*s and one distinguished State is marked as* init*ial state. States are connected with* TR *or* TS *objects. Events correspond to states, Send (Rcv) objects to TS (TR) objects. Fig. 1(b) shows a constraint forbidding the existence of two Events marked with first.*

[1]Note that other approaches for checking consistency of graph transformation systems w.r.t. graph constraints such as [12] may be in principle also employed here. For a discussion of the differences see [11].

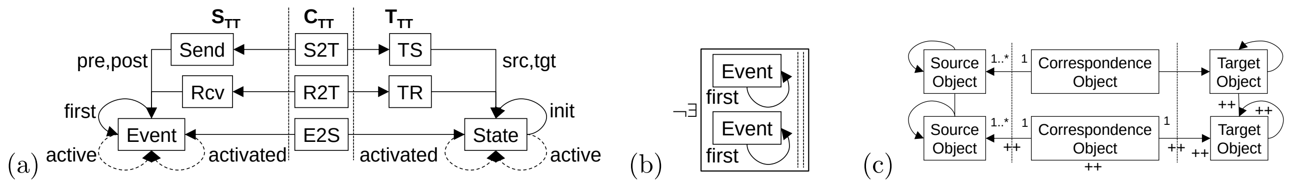


Figure 1: (a) Metamodel of Lifeline2Automaton, (b) metamodel constraint, and (c) prototypical structure of story patterns

Story diagrams originate from the Unified Modeling Language (UML) [9]. As enhanced activity diagrams they offer a way of operationally defining model transformations. Story diagrams consist of *activity nodes* containing *story patterns*. Activity nodes are connected via edges, which, along with conditional branches and loops, define control flow of a story diagram. A story diagram has an initial (final) node, where control flow starts (ends).

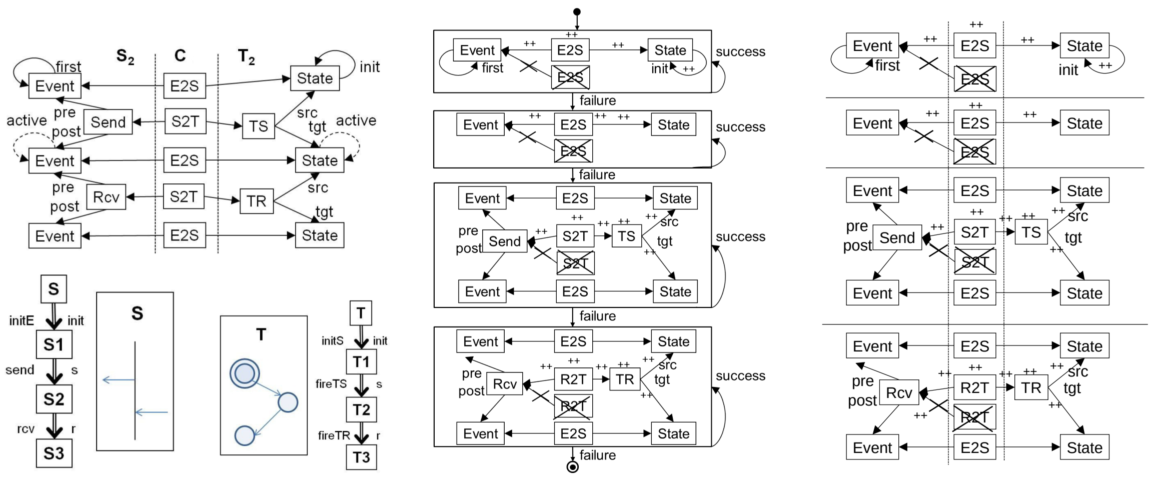
A *story pattern* describes a transformation rule. Each story pattern specifies in a condensed format the types of objects and links that need to be matched or added (marked with `++`) when applied to a model. These types correspond to a given metamodel containing classes and associations. Story patterns may have negative application conditions specifying which elements are forbidden for a valid match; those are crossed out.

In this paper we assume *restricted SDs* in particular consisting of a sequence of $n > 0$ particular `WHILE` loops containing one non-deleting story pattern without nesting: $\texttt{WHILE}(\rho_1)\texttt{;WHILE}(\rho_2)\texttt{;}\ldots\texttt{WHILE}(\rho_n)\texttt{.}$ The loop condition coincides with the applicability of the contained story pattern. This is described in story diagrams by a `success` loop edge attached to the story pattern node and an outgoing `failure` edge. Finally we restrict to story patterns that obey to the *prototypical structure* as depicted in Fig. 1(c) (conditions a)-c) are depicted). They a) create a unique correspondence object c, b) have a non-empty fragment of source objects not linked to a correspondence object yet (i.e. to be translated) which will be linked to c (i.e. will be translated), and c) include at least the unique correspondence object for each already translated fragment that is referred to. Furthermore, we require d) that each story pattern contains a NAC forbidding for each source object to be translated by this story pattern that it is already translated. Finally, we require e) that the type of the created correspondence object c of each story pattern ρ_i is not present as a type required by some story pattern ρ_j occurring earlier in the story diagram. Note that story patterns of this prototypical kind preserve all elements of the source model and thus describe only outplace model transformations. These story patterns also relate source patterns via a unique correspondence objects to its translated target pattern. Each source pattern is only translated once and a story pattern ρ_i occurring after a story pattern ρ_j in the story diagram does not create new matches for ρ_j.

A restricted story diagram is then *executed* starting with the first activity node succeeding the initial node. If a match for the story pattern of the current activity node is found, the story pattern is applied and the activity node linked via a `success` edge becomes the current one. Otherwise, the activity node linked via a `failure` is chosen. Due to the prototypical structure of the story pattern thus at first a valid match for the source model is searched and then the prescribed correspondence and target elements are added while the source model is always preserved. Note that the prototypical structure of story patterns ensures *termination* for each finite source model, since each story pattern connects at least one source object to a correspondence object (i.e. at least one objects is translated) and it forbids source objects to be connected to more than one correspondence object (i.e. each object is translated only once).

Example 2 (SD for Lifeline2Automaton)**.** *Fig. 2(b) shows a Story Diagram that describes our example transformation. The story diagram starts with a lifeline and sets up a corresponding automaton related with correspondences step by step. The first activity node creates an initial state for the first event. Since there is only one first event, this is done only once. Afterwards all other events are translated. Activity node three connects the states that belong to already translated events that are connected via a Send object properly. The fourth activity node is the same as the third, but for Rcv obejcts instead of send objects. In the top of Fig. 2(a) an example model transformation instance (with* active *edges as dynamic elements) is depicted computed by this SD.*

The prototypical structure, while appearing to only impose additional restrictions, is actually also a means of explicitly capturing traceability information required for the transformation. While other approaches and model transformation languages may do this in an implicit way, they still require similar mechanisms. Moreover, restricted SDs roughly correspond to the phased construction design pattern [13] based on the investigation of leading model transformation languages such as ATL, QVT and graph transformation based approaches. In



(a) MT instance for Lifeline2Automaton from [7]
(b) SD for Lifeline2Automaton
(c) Rules for Lifeline2Automaton ordered by priority (descending)

Figure 2: Lifeline2Automaton Running Example

general, considering restricted forms of model transformations to guarantee important transformation properties and to enable verification at the transformation level is a common approach (see e.g. [14]).

3 Formal Model

In order to formalize operational model transformations and our verification scheme in the subsequent sections, we reintroduce the notions: *typed graph, morphism, transformation, transformation system* and *constraint* as well as *graph language*. Moreover we revisit *control conditions* for a transformation system and *inductive invariants*. Finally, we reintroduce how these notions can be generalized to so-called triple graphs, consisting of the following parts: the source (target) component describing the source (target) of a model transformation related by correspondence relationships in the correspondence component.

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. Graphs can be equipped with *typing* over a given type graph TG as usual [15] by adding a so-called typing morphism from each graph to TG. Such a typing morphism is a regular graph morphism from the graph G to be typed into the type graph TG, expressing to which type node/edge in TG each node/edge in G, resp., is being mapped. A *graph morphism* $f = (f_V, f_E)$ consists of a node mapping f_V and edge mapping f_E preserving source and target mappings. A *graph language* $\mathcal{L}(TG)$ consists of all graphs typed over TG. The category of typed graphs and morphisms is called $\mathbf{Graphs}_{\mathtt{TG}}$.

We can further constrain this set of typed graphs using *graph constraints*. We restrict to a specific kind of constraints that can be handled by our invariant checker [10, 11], but in general they may be more expressive [16]. *true* and *false* are graph constraints. For every graph N, $\neg\exists N$ is a graph constraint. For every graph P, C_i such that P is included in C_i via some inclusion morphism $c_i : P \to C_i$ with i in an index set I, $\forall(P, \vee_{i\in I}\exists c_i)$ is a graph constraint. If the inclusion morphism c_i is clear from the context we also write $\forall(P, \vee_{i\in I}\exists C_i)$. Every conjunction of graph constraints is a graph constraint. *Satisfiability* of graph constraints is inductively defined as follows: A graph G satisfies $\neg\exists N$ if there does not exist an injective morphism $q : N \to G$. A graph G satisfies $\forall(P, \vee_{i\in I}\exists c_i)$ if there exists for each injective morphism $q : P \to G$ at least one $i \in I$ such that an injective morphism $q' : C_i \to G$ exists with $q' \circ c_i = q$. A graph constraint satisfies a conjunction of graph constraints if it satisfies each graph constraint in the conjunction. Graph constraints can be equipped with *typing* over a type graph TG as usual [15] by adding typing morphisms from each graph to TG and by requiring type-compatibility w.r.t. TG for each morphism. We can now restrict the language $\mathcal{L}$(TG) to a so-called *graph language with constraint* $\mathcal{L}(\text{TG}, \mathcal{C})$ comprising those graphs also satisfying a constraint $\mathcal{C}$ typed over TG.

A *graph rule* $\rho : (L \xrightarrow{r} R, NAC_\rho)$ consists of a graph morphism r, which is an inclusion and a so-called negative application condition (NAC) $NAC_\rho = \wedge_{i\in I}(\neg\exists n_i)$ with $n_i : L \to N_i$. Given a graph rule $\rho : (L \xrightarrow{r} R, NAC_\rho)$ and a graph G, ρ can be applied to G if there is an occurrence of L in G i.e. an injective graph morphism $m : L \to G$, called *match*, such that $m \vDash NAC_\rho$. The latter is the case if for each n_i in NAC_ρ it holds that there does not exist an injective morphism $q_i : N_i \to G$ such that $q_i \circ n_i = m$.

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\ G \arrow[r, "h"'] & H \end{tikzcd}$$

Note that we restrict to non-deleting rules in this paper. A *direct graph transformation* $G \Rightarrow_{\rho,m} H$ from G to H or short $G \Rightarrow_{\rho} H$ via ρ and m consists of the pushout (PO) in $\mathbf{Graphs}_{\mathtt{TG}}$.

For a set of rules $\mathcal{R}$ a *direct graph transformation* $G \Rightarrow_{\mathcal{R}} G'$ is given if a rule $\rho \in \mathcal{R}$ with $G \Rightarrow_{\rho} G'$ exists. A *graph transformation*, denoted as $G_0 \Rightarrow^*_{\mathcal{R}} G_n$, is a sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \cdots \Rightarrow_{\mathcal{R}} G_n$ of direct graph transformations. Graph rules and transformations 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. A *graph transformation system* (GTS) gts $= (\mathcal{R}, \mathrm{TG})$ consists of a set of rules $\mathcal{R}$ typed over a type graph TG.

The application of graph transformation (GT) rules from a GTS can be restricted via a so-called *control condition* [17] restricting the non-determinism of rule application during the transformation process. A graph transformation $G \Rightarrow^*_{\mathcal{R}} G'$ via rules in $\mathcal{R}$ that is allowed by some control condition $\mathcal{CC}$ is denoted $G \Rightarrow^*_{\mathcal{R},\mathcal{CC}} G'$. A *GTS with control condition* (gts, $\mathcal{CC}$) consists of a GTS gts $= (\mathcal{R}, \mathrm{TG})$ and control condition $\mathcal{CC}$ over $\mathcal{R}$. Priorities are a well-known possible control condition for graph transformation systems, where the rule with the highest priority is always applied first. Given a set of graph transformation rules $\mathcal{R} = \{\rho_1, \ldots, \rho_n\}$ then $p : \mathcal{R} \to \mathbb{N}$ defines *priorities* over $\mathcal{R}$. Thereby, $G \Rightarrow_{\mathcal{R},p} G'$ is a direct graph transformation allowed by p if $\exists \rho \in \mathcal{R} : G \Rightarrow_{\rho} G'$ and $\neg\exists \rho' \in \mathcal{R} : G \Rightarrow_{\rho'} G'' \wedge p(\rho) < p(\rho')$. $G \overset{*}{\Rightarrow}_{\mathcal{R},p} G'$ is a sequence of direct transformations allowed by p. A graph constraint $\mathcal{C}$ is an *inductive invariant* of the graph transformation system with priorities gts $= ((\mathcal{R}, \mathrm{TG}), p)$, if for all graphs G in $\mathcal{L}(TG)$, it holds that $G \vDash \mathcal{C} \wedge\ G \Rightarrow_{\mathcal{R},p} G'$ implies $G' \vDash \mathcal{C}$.

Triple graphs are defined as introduced in [18, 19], a particular formalization different from the original one introduced in [6]. Thereby, the main idea is to use a distinguished, fixed graph $\mathtt{TR}$ which all triple graphs, including the type triple graph $S_{TT}C_{TT}T_{TT}$, are typed over. It defines three node types s, c, and t representing the source, correspondence, and target nodes, and corresponding edge types l_s and l_t for source and target graph edges. Moreover, for the connections from correspondence to source or target nodes the edge types e_{cs} and e_{ct} are available.

$$\begin{tikzcd} \mathtt{TR} & s \arrow[loop above, "l_s"] & c \arrow[l, "e_{cs}"'] \arrow[r, "e_{ct}"] & t \arrow[loop above, "l_t"] \end{tikzcd}$$

We say that $\mathtt{TR}_S$, $\mathtt{TR}_C$, $\mathtt{TR}_T$, and $\mathtt{TR}_{SC}$ as shown below,

$$\begin{tikzcd} \mathtt{TR}_S & s \arrow[loop above, "l_s"] & \mathtt{TR}_C & s & c \arrow[l, "e_{cs}"'] \arrow[r, "e_{ct}"] & t & \mathtt{TR}_T & t \arrow[loop above, "l_t"] & \mathtt{TR}_{SC} & s & c \arrow[l, "e_{cs}"'] \end{tikzcd}$$

are the *source, correspondence, target* and *source-correspondence component* of $\mathtt{TR}$, respectively.

Analogously, the projection of a graph G typed over $\mathtt{TR}$ to $\mathtt{TR}_S$, $\mathtt{TR}_C$, $\mathtt{TR}_{SC}$, or $\mathtt{TR}_T$ selects the corresponding component of this graph. A *triple graph* (G, triple_G) is a graph G equipped with a morphism $\mathrm{triple}_G : G \to \mathtt{TR}$. We denote a triple graph as a combination of three indexed capitals, as for example $S_GC_GT_G$, 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. Moreover, C^S_G denotes the *source-correspondence* component of G.

Analogously to typed graphs, *typed triple graphs* are triple graphs typed over a distinguished triple graph $S_{TT}C_{TT}T_{TT}$, called *type triple graph.* In the remainder of this paper, we assume every triple graph $S_GC_GT_G$ and triple graph morphism f to be typed over $S_{TT}C_{TT}T_{TT}$, even if not explicitly mentioned. In particular, this means that S_G is typed over S_{TT}, C_G is typed over C_{TT}, and T is typed over T_{TT}. We say that S_G (T_G or C_G) is a *source graph* (*target graph* or *correspondence graph*, respectively) belonging to the language $\mathcal{L}(S_{TT})$ ($\mathcal{L}(T_{TT})$ or $\mathcal{L}(C_{TT})$, respectively). Note that each source graph (target graph) corresponds uniquely to a triple graph with empty correspondence and target (source and correspondence) components, respectively. Therefore, if it is clear from the context that we are dealing with triple graphs, we denote triple graphs $S_G\varnothing\varnothing$ ($\varnothing\varnothing T_G$) with empty correspondence and target components (source components) also as S_G (T_G), respectively. Typing a graph over a type triple graph $S_{TT}C_{TT}T_{TT}$ already defines the triples, i.e. any graph and any morphism typed over the type triple graph $S_{TT}C_{TT}T_{TT}$ corresponds uniquely to a typed triple graph and a typed triple graph morphism, respectively. Analogously [18, 19], also graph rules, graph transformations as well as graph constraints can be generalized to *typed triple graph rules, transformations* as well as *constraints* by requiring that each graph and morphism therein is typed over $S_{TT}C_{TT}T_{TT}$.

The type graph $S_{TT}C_{TT}T_{TT}$ can be enriched with dynamic types for source and target languages and is then denoted as $S_{RT}C_{TT}T_{RT}$. In particular, the *bisimulation constraint* $\mathcal{C}_{\mathrm{Bis}}$ mentioned in the introduction and used throughout the paper is a graph constraint typed over $S_{RT}C_{TT}T_{RT}$. If some graph $S_GC_GT_G$, 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 $S_GC_GT_G$, 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 paper.

4 Operational Model Transformations Formalized

We formalize the *source and target modeling language* $\mathcal{L}_S$ and $\mathcal{L}_T$ of restricted SDs describing our operational model transformations (see Section 2) by a graph language with constraint. Thus we have a source and target graph language $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ and $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$ with source and target type graph S_{TT} and T_{TT} and source and target constraints $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{T}}$, respectively. For example, Fig. 1(b) shows a fragment of the source constraint $\mathcal{C}_{\mathcal{S}}$ of our running example. The *model transformation instances with correspondences* belonging to $\mathcal{L}_{SCT}$ are formalized by triple graphs. These instances conform to the *metamodel of our model transformation* that is formalized by a type triple graph $S_{TT}C_{TT}T_{TT}$. To further restrict the set of model transformation instances with correspondences, we allow a graph constraint $\mathcal{C}_{\mathcal{SCT}}$ typed over the type triple graph $S_{TT}C_{TT}T_{TT}$ expressing restrictions on the correspondences between source and target models. W.l.o.g. we assume that $\mathcal{C}_{\mathcal{SCT}}$ comprises the source and target constraints $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{T}}$ of the source and target graph language $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ and $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$, respectively, such that the target component T of each SCT in $\mathcal{L}(S_{TT}C_{TT}T_{TT}, \mathcal{C}_{\mathcal{SCT}})$ automatically belongs to $\mathcal{L}(T_{TT}, \mathcal{C}_{\mathcal{T}})$.

For our verification scheme it will be important that *traceability information* for source elements is *complete* as well as *unique* in the following sense. Note that restricted story diagrams as described in Section 2 only create uniquely traceable model transformation instances.

Definition 1 (Uniquely/Complete Traceability)**.** *Given a triple graph SCT, then SCT is* completely traceable*, if each node s of S is connected via a correspondence edge to at least one correspondence node c of C. Moreover, SCT is* uniquely traceable*, if each node s of S is connected via a correspondence edge to at most one correspondence node c of C.*

We formalize the *story patterns* present in a restricted SD by a set $\mathcal{R}$ of non-deleting triple graph transformation rules and the *control flow* by a control program P, being a specific control condition over $\mathcal{R}$. Note that as explained in the previous section, the prototypical structure of SPs in restricted SDs ensures that a control program over rules derived from these patterns terminates. Moreover, w.l.o.g. we assume that each control program P over $\mathcal{R}$ is complete meaning that each rule ρ of $\mathcal{R}$ is used in P.

Definition 2 (Operational Rules & Control Program)**.** *An* operational rule *is a triple graph rule $\rho : (S_LC_LT_L \xrightarrow{r} S_LC_RT_R, NAC_\rho)$ typed over $S_{TT}C_{TT}T_{TT}$. Given a non-empty set of operational rules $\mathcal{R}$, then a* control program *P over $\mathcal{R}$ is defined recursively, as follows: $\texttt{WHILE(}\rho\texttt{)}$ is a control program over $\{\rho\}$, and $\texttt{WHILE(}\rho'\texttt{);}P'$ is a control program over $\rho' \cup \mathcal{R}'$ with P' a control program over $\mathcal{R}'$ such that ρ' is not in $\mathcal{R}'$ and any rule in $\mathcal{R}'$ is not allowed to create any new match for ρ'. Finally, $G \Rightarrow^*_{\{\rho\},\texttt{WHILE(}\rho\texttt{)}} G'$ if $G \Rightarrow^*_{\{\rho\}} G' \wedge G' \not\Rightarrow_{\{\rho\}}$, and $G \Rightarrow^*_{\rho' \cup \mathcal{R}',\texttt{WHILE(}\rho'\texttt{);}P'} G'$ if $\exists G'' : G \Rightarrow^*_{\{\rho'\},\texttt{WHILE(}\rho'\texttt{)}} G'' \wedge G'' \Rightarrow^*_{\mathcal{R}',P'} G'$.*

Now we are ready to formalize operational model transformations given as restricted SDs (see Section 2).

Definition 3 (Operational Model Transformation OP)**.** *An* operational model transformation OP $= ((\mathcal{R}, S_{TT}C_{TT}T_{TT}), P), \mathcal{C}_{\mathcal{SCT}})$ *consists of a GTS $(\mathcal{R}, S_{TT}C_{TT}T_{TT})$ with control program P over a non-empty finite set of operational rules $\mathcal{R}$ as given in Def. 2 typed over a uniquely traceable type triple graph $S_{TT}C_{TT}T_{TT}$ and a graph constraint $\mathcal{C}_{\mathcal{SCT}}$ typed over $S_{TT}C_{TT}T_{TT}$. The induced* model transformation with correspondences MTC(OP) *consists of the subset of $\mathcal{L}(S_{TT}C_{TT}T_{TT}, \mathcal{C}_{\mathcal{SCT}})$ containing exactly those triple graphs SCT for which there exists a triple graph transformation $S \Rightarrow^*_{\mathcal{R},P} SCT$ such that SCT is completely and uniquely traceable. A* model transformation instance *of* MTC(OP) *is an element of* MTC(OP).

5 Verification Scheme for Checking the Bisimulation Constraint

Recall that for solving the behavior preservation problem we want to be able to check for some operational model transformation OP that all its instances fulfill the bisimulation constraint $\mathcal{C}_{\text{Bis}}$, i.e. $\forall SCT \in \text{MTC(OP)} : SCT \models \mathcal{C}_{\text{Bis}}$ (1^*) (see Sect. 1). In our verification approach, we aim at reducing this problem to invariant checking. In principle, this means showing that $\mathcal{C}_{\text{Bis}}$ holds for each valid source model S (induction base) and that $\mathcal{C}_{\text{Bis}}$ is an inductive invariant of the operational model transformation OP (induction step). We conduct the verification with our invariant checker [10, 11], which is able to automatically perform inductive invariant checking for typed graph transformation systems with priorities and graph constraints as invariants as introduced in Section 3. Since the invariant checker does not directly support control programs P over rules as employed by OP, we first study how we can map the behavior of P to priorities as in the following Lemma. Note that this is correct since each rule in P is not allowed to create any new matches for an earlier rule.

Lemma 1 (Mapping of Control Programs to Priorities)**.** *Given a graph transformation system with control program $((\mathcal{R}, S_{TT}C_{TT}T_{TT}), P)$ with non-empty rule set $\mathcal{R}$ and control program P we can derive a GTS with priorities $((\mathcal{R}, S_{TT}C_{TT}T_{TT}), p)$ such that*

$$G \Rightarrow^*_{\mathcal{R},P} G' \qquad \Longrightarrow \qquad (\quad G \Rightarrow^*_{\mathcal{R},p} G' \quad \wedge \quad G' \not\Rightarrow^*_{\mathcal{R}} \quad).$$

Proof. We define priorities following the recursive definition of a control program in Def. 2. For $WHILE(\rho)$ a control program over $\{\rho\}$ we define the priority $p(\rho) := 1$. For $WHILE(\rho'); P'$ such that P' is a control program over $\mathcal{R}'$ we define $p(\rho') := max\{p(\rho)|\rho \in \mathcal{R}'\}+1$. More informally, we have that the priority $p(\rho')$ is the maximum of the priorities of all rules occurring in P' incremented by 1. Now we can prove the above condition. For $WHILE(\rho)$ the above condition is trivially true. For $WHILE(\rho'); P'$ we assume that the above condition holds for P' over $\mathcal{R}'$ $(*)$. Given $G \Rightarrow^*_{\rho' \cup \mathcal{R}', \mathtt{WHILE}(\rho');P'} G'$, then by definition $\exists G'' : G \Rightarrow^*_{\{\rho'\},\mathtt{WHILE}(\rho')} G'' \wedge G'' \Rightarrow^*_{\mathcal{R}',P'} G'$. By hypothesis $(*)$ for $G'' \Rightarrow^*_{\mathcal{R}',P'} G'$ it follows that $G'' \Rightarrow^*_{\mathcal{R}',p} G'$ and $G' \not\Rightarrow^*_{\mathcal{R}'}$. Consequently, since ρ' has by definition a higher priority than all rules in $\mathcal{R}'$ and they do not create new matches for ρ' it follows that $G \Rightarrow^*_{\rho' \cup \mathcal{R}',p} G'$ and $G' \not\Rightarrow^*_{\rho' \cup \mathcal{R}'}$. □

Example 3. *The priorities for our model transformation example are depicted in Figure 2(c) by the ordering.*

Two problems occur when now trying to check $\mathcal{C}_{\text{Bis}}$ as an invariant of our operational model transformation OP. On the one hand not every constraint in $\mathcal{C}_{\text{Bis}}$ holds for each valid source model S (*induction base*) and on the other hand $\mathcal{C}_{\text{Bis}}$ might be too weak to be established as an inductive invariant for the operational rules with priorities derived from OP (*induction step*). In the following theorem we therefore present a derived bisimulation constraint that we can check as an alternative invariant such that in the end $\forall SCT \in \text{MTC(OP)} : SCT \vDash \mathcal{C}_{\text{Bis}}$ (1*) holds. For a successful induction base we argue that $\mathcal{C}_{\text{Bis}}$ can be *weakened* to a constraint $\mathcal{C}^{\text{w}}_{\text{Bis}}$ because of completeness and uniqueness of the traceability information in each model transformation instance. For a successful induction step we introduce a so-called transformation constraint $\mathcal{C}_{\text{TR}}$ that can be used to *strengthen* $\mathcal{C}_{\text{Bis}}$.

Definition 4 (weakened bisimulation constraint, transformation constraint, derived bisimulation constraint)**.** *Given a bisimulation constraint $\mathcal{C}_{\text{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$, then a graph constraint $\mathcal{C}^{\text{w}}_{\text{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$ is a* weakened bisimulation constraint *for $\mathcal{C}_{\text{Bis}}$ if (1) for each completely and uniquely traceable triple graph $SCT \vDash \mathcal{C}^{\text{w}}_{\text{Bis}}$ also implies $SCT \vDash \mathcal{C}_{\text{Bis}}$ and if (2) each graph $S \in \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ satisfies $\mathcal{C}^{\text{w}}_{\text{Bis}}$. Moreover, a graph constraint $\mathcal{C}_{\text{TR}}$ typed over $S_{RT}C_{TT}T_{RT}$ is a* transformation constraint *for $\mathcal{C}_{\text{Bis}}$ if each graph $S \in \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ satisfies $\mathcal{C}_{\text{TR}}$. The graph constraint $\mathcal{C}'_{\text{Bis}} = \mathcal{C}^{\text{w}}_{\text{Bis}} \wedge \mathcal{C}_{\text{TR}}$ is then a* derived bisimulation constraint *for $\mathcal{C}_{\text{Bis}}$.*

Theorem 1. *Given an operational model transformation* $\text{OP} = (((\mathcal{R}, S_{TT}C_{TT}T_{TT}), P), \mathcal{C}_{\mathcal{SCT}})$ *as in Def. 3 and a bisimulation constraint $\mathcal{C}_{\text{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$, then $\forall SCT \in \text{MTC(OP)} : SCT \vDash \mathcal{C}_{\text{Bis}}$ if the graph transformation system with priorities $((\mathcal{R}, S_{TT}C_{TT}T_{TT}), p)$ derived from $((\mathcal{R}, S_{TT}C_{TT}T_{TT}), P)$ according to Lemma 1 has a derived bisimulation constraint $\mathcal{C}'_{\text{Bis}}$ as given in Def. 4 as inductive invariant.*

Proof. We prove first that each $SCT \in \text{MTC(OP)}$ satisfies $\mathcal{C}'_{\text{Bis}}$ by induction over the length of the graph transformation needed to create some triple $SCT \in \text{MTC(OP)}$. (Induction base) The weakened bisimulation constraint $\mathcal{C}^{\text{w}}_{\text{Bis}}$ and also the transformation constraint $\mathcal{C}_{\text{TR}}$ are fulfilled by construction. (Induction step) By construction and by Lemma 1. Then we know by definition of the weakened bisimulation constraint that also $\mathcal{C}_{\text{Bis}}$ holds. □

6 Application of Verification Scheme

We outline in this section that for a bisimulation constraint $\mathcal{C}_{\text{Bis}}$ fulfilling certain restrictions we can determine a related derived bisimulation constraint as required for Theorem 1. We exploit in the following that graph constraints may be trivially true for each potential model transformation instance ($\mathcal{C}^{True}_{\text{Bis}}$), that a graph constraint may hold for each source graph S in $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ ($\mathcal{C}^{src}_{\text{Bis}}$), and that for some constraints a weakened bisimulation

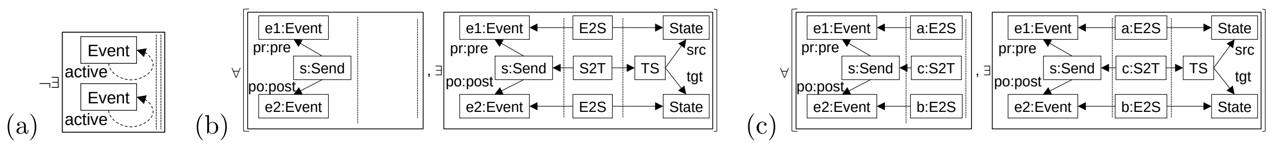


Figure 3: (a) Fragment of $\mathcal{C}_{\mathrm{Bis}}^{True}$, (b) fragment of $\mathcal{C}_{\mathrm{Bis}}$ to be weakened, and (c) weakened fragment of $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$

constraint ($\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$) can be derived. In addition, we will show how a suitable transformation constraint $\mathcal{C}_{\mathrm{TR}}$ for $\mathcal{C}_{\mathrm{Bis}}$ can be derived systematically.

The *trivially true constraint* $\mathcal{C}_{\mathrm{Bis}}^{True}$ summarizes all graph constraints in $\mathcal{C}_{\mathrm{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$ that are true for each potential model transformation instance $SCT \in \mathcal{L}(S_{TT}C_{TT}T_{TT})$. In particular, some of the conditions of $\mathcal{C}_{\mathrm{Bis}}$ refer to dynamic elements required for the semantics definitions as follows: Conditions of the form $\wedge_{i\in I}\forall(S_{P_i}C_{P_i}T_{P_i}, \exists S_{C_i}C_{C_i}T_{C_i})$ with $S_{P_i}C_{P_i}T_{P_i}$ typed over $S_{RT}C_{TT}T_{RT}$ but not $S_{TT}C_{TT}T_{TT}$ alone are true for all SCT typed over $S_{TT}C_{TT}T_{TT}$ since the precondition can never be fulfilled by a graph typed over $S_{TT}C_{TT}T_{TT}$. Moreover, conditions of the form $\wedge_{i\in I}\neg\exists S_{N_i}C_{N_i}T_{N_i}$ with $S_{N_i}C_{N_i}T_{N_i}$ typed over $S_{RT}C_{TT}T_{RT}$ but not $S_{TT}C_{TT}T_{TT}$ alone are true for all SCT typed over $S_{TT}C_{TT}T_{TT}$ since $S_{N_i}C_{N_i}T_{N_i}$ can never be found for a graph typed only over $S_{TT}C_{TT}T_{TT}$.

Example 4. *The constraint fragment shown in Figure 3(a) is part of $\mathcal{C}_{\mathrm{Bis}}^{True}$ for our example. It is true for all triple graphs SCT typed over $S_{TT}C_{TT}T_{TT}$, since it describes the non-existence of dynamic elements and $S_{TT}C_{TT}T_{TT}$ does not contain dynamic types.*

The *invariant source constraint* $\mathcal{C}_{\mathrm{Bis}}^{src}$ further summarizes all graph constraints in $\mathcal{C}_{\mathrm{Bis}}$ typed over $S_{RT}C_{TT}T_{RT}$ that hold for each source graph S in $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ and thus can be directly approached by the invariant checker.

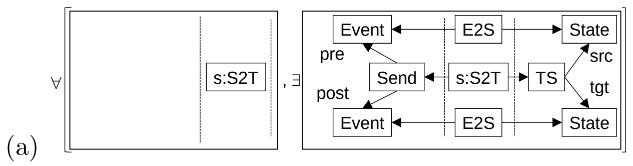
Employing the introduced $\mathcal{C}_{\mathrm{Bis}}^{True}$ and $\mathcal{C}_{\mathrm{Bis}}^{src}$, we assume that $\mathcal{C}_{\mathrm{Bis}}$ has the form $\mathcal{C}_{\mathrm{Bis}}^{True} \wedge \mathcal{C}_{\mathrm{Bis}}^{src} \wedge \wedge_{i\in I}\forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$ and we can observe that in this case only the constraints of the form $\forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$ (typed over $S_{TT}C_{TT}T_{TT}$) may be false for each valid source model. This is because they may imply the existence of a non-empty correspondence and target pattern for a specific source pattern. Obviously, no graph typed only over S_{TT} can fulfill this requirement. Hence, these constraints have to be weakened to fulfill the induction base for our inductive approach. We need a constraint $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$ such that each completey and uniquely traceable triple graph SCT fulfilling $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$ also fulfils the original constraints and that each graph in the source language $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ fulfils $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$ (cf. Def. 4).

Lemma 2 (Weakened Bisimulation Constraint $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$)**.** *Given the bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{Bis}}^{True} \wedge \mathcal{C}_{\mathrm{Bis}}^{src} \wedge \wedge_{i\in I}\forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$ then the constraint $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}} = \mathcal{C}_{\mathrm{Bis}}^{True} \wedge \mathcal{C}_{\mathrm{Bis}}^{src} \wedge \wedge_{i\in I}\forall(S_{P_i}C_{P_i}^{S}, \exists S_{P_i}C_{P_i}T_{P_i})$ is a weakened bisimulation constraint for $\mathcal{C}_{\mathrm{Bis}}$ as given in Def. 4.*

Proof. We have to show that (1) for each completely and uniquely traceable triple graph SCT with $SCT \vDash \mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$, $SCT \vDash \mathcal{C}_{\mathrm{Bis}}$ also holds and that (2) each graph $S \in \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$ satisfies $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$. (2) holds by construction of $\mathcal{C}_{\mathrm{Bis}}^{\mathrm{w}}$. For (1) we in particular have to show that if $SCT \vDash \wedge_{i\in I}\forall(S_{P_i}C_{P_i}^{S}, \exists S_{P_i}C_{P_i}T_{P_i})$ then $SCT \vDash \forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$. First, it follows from the complete and unique traceability of the type triple graph $S_{TT}C_{TT}T_{TT}$ in OP and the fact that our constraint to be weakened is typed accordingly that for S_{P_i} only a unique extension C^S can exist and due to the guaranteed type conformance $C_{P_i}^{S}$ must be a subgraph of that. Consequently, any completely and uniquely traceable SCT that matches S_{P_i} also matches $S_{P_i}C^S$ and thus $S_{P_i}C_{P_i}^{S}$ such that condition (1) must be fulfilled as well. □

Example 5. *The constraint fragment in Figure 3(b) is part of $\wedge_{i\in I}\forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$ for our example. Figure 3(c) shows how this fragment can be weakened such that it holds for each graph S in $\mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$, i.e. for each valid source model, because no such graph S can contain correspondence nodes.*

We further use a *transformation constraint* typed over $S_{TT}C_{TT}T_{TT}$ to strengthen the bisimulation constraint $\mathcal{C}_{\mathrm{Bis}} = \mathcal{C}_{\mathrm{Bis}}^{True} \wedge \mathcal{C}_{\mathrm{Bis}}^{src} \wedge \wedge_{i\in I}\forall(S_{P_i}, \exists S_{P_i}C_{P_i}T_{P_i})$ describing that the existence of a correspondence or target model node in a graph SCT implies that one of the rules capable of creating the respective correspondence or target model node has been applied and that one of those rules' side effects (right-hand rule side) is present in SCT. By construction, this transformation constraint is fulfilled by all source graphs S typed over S_{TT} as these graphs contain no correspondence nodes and can thus never fulfill the precondition. Since the operational rules are derived from the operational rules, they will usually be inductive invariants.



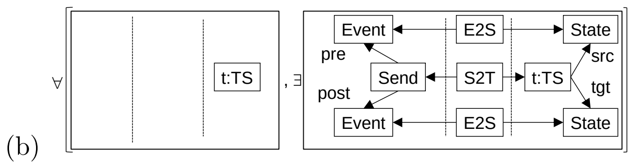


Figure 4: Fragments of the transformation contraint $\mathcal{C}_{TR}$ for (a) correspondence nodes and (b) target nodes

Lemma 3 (Transformation Constraint $\mathcal{C}_{TR}$)**.** *Given a set of operational rules $\mathcal{R} = \{\rho_i | i \in I\}$ with $\rho_i : (S_{L_i} C_{L_i} T_{L_i} \xrightarrow{r_i} S_{L_i} C_{R_i} T_{R_i}, NAC_{\rho_i})$, then the related* transformation constraint for correspondence nodes *is $\mathcal{C}^C_{\rho_i} = \forall(c_i, \vee_{j \in J_i} \exists S_{L_j} C_{R_j} T_{R_j})$ with c_i a correspondence node in C_{R_i} but not in $r_i(C_{L_i})$ created by ρ_i and $J_i = \{j | \rho_j$ creates a correspondence node of the same type in $S_{TT} C_{TT} T_{TT}$ as $c_i\} \subseteq I$ and the related* transformation constraint for target nodes *is $\mathcal{C}^T_{\rho_i} = \forall(t_i, \vee_{j \in J_i} \exists S_{L_j} C_{R_j} T_{R_j})$ with t_i a target node in T_{R_i} but not in $r_i(T_{L_i})$ created by ρ_i and $J_i = \{j | \rho_j$ creates a target node of the same type in $S_{TT} C_{TT} T_{TT}$ as $t_i\} \subseteq I$. Given the bisimulation constraint $\mathcal{C}_{\text{Bis}}$, then the constraint $\mathcal{C}_{TR} = \wedge_{i \in I} \mathcal{C}^C_{\rho_i} \wedge \mathcal{C}^T_{\rho_i}$ is a transformation constraint for $\mathcal{C}_{\text{Bis}}$ as given in Def. 4.*

Proof. Since $\mathcal{C}_{TR} = \wedge_{i \in I} \mathcal{C}^C_{\rho_i} \wedge \mathcal{C}^T_{\rho_i}$ is a constraint typed over $S_{TT} C_{TT} T_{TT}$ and it holds for each graph $S \in \mathcal{L}(S_{TT}, \mathcal{C}_{\mathcal{S}})$, it is a well-defined transformation constraint for $\mathcal{C}_{\text{Bis}}$. $\square$

Example 6 (Transformation constraint)**.** *Two fragments of the transformation constraint $\mathcal{C}_{TR}$ are depicted in Figure 4. Note that each valid source model satisfies such constraints. In particular, the fragments state that the existence of a correspondence node of type S2T (TS) requires the existence of the side effect of a rule capable of creating nodes of type S2T (TS, respectively). In both cases, there is only one such rule – the third rule in Figure 2(c), which creates TS and S2T nodes for Send nodes. Hence, the fragments' disjunctions ($\vee_{j \in J_i} \exists S_{L_j} C_{R_j} T_{R_j}$) contain only one condition each.*

We performed the inductive invariant check for $((\mathcal{R}, S_{TT} C_{TT} T_{TT}), p)$ with priorities p derived from the control program P in OP according to Lemma 1 and the derived bisimulation constraint $\mathcal{C}'_{\text{Bis}}$ of our running example Lifeline2Automaton introduced in Section 2 with our invariant checker [10, 11]. In particular, $((\mathcal{R}, S_{TT} C_{TT} T_{TT}), p)$ and $\mathcal{C}'_{\text{Bis}}$ are supported by our checker and the check was successful. Using Theorem 1 we can then conclude for our running example that indeed (1*) $\forall SCT \in \text{MTC(OP)} : SCT \models \mathcal{C}_{\text{Bis}}$ holds. The successful verification of (2) $\forall SCT \in \text{MTC(OP)} : sem(S) \approx_{bsim} sem(T)$ for our running example is explained in [7, 8] such that our running example transformation Lifeline2Automaton is behavior preserving.

For our example consisting of four transformation rules with at most 9 nodes and constraints with at most 9 nodes, the verification of condition (1*) takes about 10 seconds on an Intel Core-i7-2640M processor with two cores at 2,8 GHz, 8 GB of main memory and running Eclipse 4.2.2 and Java 8 with a limit of 2 GB on Java heap space, which is very acceptable. As can be inferred from [11], the size of rules and constraints is usually the limiting factor for feasibility of verification rather than the number of rules or constraints.

7 Conclusion and Future Work

In this paper, we presented a first approach towards the automatic verification at the transformation level of behavior preservation captured by bisimulation for operational model transformations. To achieve this, we also presented a formalization of restricted Story Diagrams with a mapping to graph transformation systems with priorities. Our main result has been obtained by reusing parts of our approach for relational model transformations [7, 8] and showing that required checks for the operational model transformation can be in case of a specific restricted form reduced to invariant checking for graph transformation with priorities. We further illustrate the feasibility by reporting on a simple example.

Besides the covered control constructs for operational model transformations such as while loops and sequence, operational model transformations may also employ as special cases single transformation steps, conditionals, general loops, nesting, or specific means to hand over binding from one rule to another. Also the type of language constraints as well as transformation rules might become more complex and may contain, for example, arbitrary levels of nesting. While in many cases these constructs can be mapped on the outlined restricted form, it remains an open question whether the approach can be extended to also cover these more general forms. Another open question is whether the required correspondence model (traceability information) is really necessarily an element that has to be specified explicitly as part of the operational model transformation or whether it can be synthesized systematically in a pre-processing step of the verification. As future work we plan to approach the outlined open questions and also study the feasibility by means of larger and more complex case studies.

References

[1] OMG: MOF QVT Final Adopted Specification, OMG Document ptc/05-11-01. (http://www.omg.org/)

[2] 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; Proceedings of the UML'03 Workshop. Number TUM-I0323 in Technical Report, Technische Universitat Munchen (2003) 63–78

[3] Engels, G., Kleppe, A., Rensink, A., Semenyak, M., Soltenborn, C., Wehrheim, H.: From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations. In I. Schieferdecker, A.H., ed.: Proceedings of the 4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2008), Berlin (Germany). LNCS, Berlin/Heidelberg, Springer (2008) 95–109

[4] Narayanan, A., Karsai, G.: Verifying Model Transformations by Structural Correspondence. Electronic Communications of the EASST: Graph Transformation and Visual Modeling Techniques 2008 **10** (2008)

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

[6] 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, 20^{th} International Workshop, WG '94. Volume 903 of LNCS., Herrsching, Germany (1994) 151–163

[7] Giese, H., Lambers, L.: Towards Automatic Verification of Behavior Preservation for Model Transformation via Invariant Checking. In Ehrig, H., Engels, G., Kreowski, H., Rozenberg, G., eds.: Proceedings of Intern. Conf. on Graph Transformation (ICGTÍ2). Volume 7562 of LNCS., Springer (2012) 249–263

[8] Dyck, J., Giese, H., Lambers, L.: Automatic Verification of Behavior Preservation at the Transformation-Level for Relational Model Transformation. Technical report, Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany (2015) (forthcoming).

[9] Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations. Volume 1764/2000 of Lecture Notes in Computer Science (LNCS)., London, UK, Springer-Verlag (2000) 296–309

[10] Becker, B., Beyer, D., Giese, H., Klein, F., Schilling, D.: Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In: Proc. of the 28^th International Conference on Software Engineering (ICSE), Shanghai, China, ACM Press (2006)

[11] Dyck, J., Giese, H.: Inductice Invariant Checking with Partial Negative Application Conditions. In Parisi-Presicce, F., Westfechtel, B., eds.: Graph Transformation. Volume 9151 of LNCS., Springer International Publishing (2015)

[12] Pennemann, K.-H.: Development of Correct Graph Transformation Systems. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg (2009)

[13] Lano, K., Kolahdouz-Rahimi, S.: Model-Transformation Design Patterns. Software Engineering, IEEE Transactions on **40**(12) (2014) 1224–1259

[14] Selim, G.M.K., Lúcio, L., Cordy, J.R., Dingel, J., Oakes, B.J.: Specification and Verification of Graph-Based Model Transformation Properties. In Giese, H., Knig, B., eds.: Graph Transformation. Volume 8571 of Lecture Notes in Computer Science. Springer International Publishing (2014) 113–129

[15] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)

[16] Habel, A., Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science **19** (2009) 1–52

[17] Kuske, S.: More about control conditions for transformation units. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: TAGT. Volume 1764 of Lecture Notes in Computer Science., Springer (1998) 323–337

[18] 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. Software and Systems Modeling **13**(1) (2014) 273–299

[19] Golas, U., Lambers, L., Ehrig, H., Giese, H.: Bridging the Gap between Formal Foundations and Current Practice for Triple Graph Grammars. In: Proceedings of ICGT 2012. LNCS, Springer (2012)