Multi-Amalgamated Triple Graph Grammars

Erhan Leblebici[1], Anthony Anjorin[1], Andy Schürr[1], and Gabriele Taentzer[2]

[1]Technische Universität Darmstadt, Germany
{erhan.leblebici,anthony.anjorin,andy.schuerr}@es.tu-darmstadt.de

[2]Philipps-Universität Marburg, Germany
taentzer@mathematik.uni-marburg.de

Abstract. *Triple Graph Grammars* (TGGs) are a well-known technique for rule-based specification of bidirectional model transformation. TGG rules build up consistent models simultaneously and are operationalized automatically to forward and backward rules describing single transformation steps in the respective direction. These operational rules, however, are of fixed size and cannot describe transformation steps whose size can only be determined at transformation time for concrete models. In particular, transforming an element to arbitrary many elements depending on the transformation context is not supported. To overcome this limitation, we propose the integration of the *multi-amalgamation* concept from classical graph transformation into TGGs. Multi-Amalgamation formalizes the combination of multiple transformations sharing a common subpart to a single transformation. For TGGs, this enables repeating certain parts of a forward or backward transformation step in a *for each* loop-like manner depending on concrete models at transformation time.

Keywords: triple graph grammars, amalgamation, model transformation

1 Introduction and Motivation

Model-Driven Engineering (MDE) has established itself as a viable means for dealing with the increasing complexity of modern software systems. Models in MDE provide suitable abstractions of a system, serve as both design and implementation artifacts, and facilitate the communication between domain experts. In most cases, several models co-exist and contain related information to describe a system from different perspectives, tools, or domains. Important challenges in this context are to create a related model from a given model, and to ensure consistency between related models during their life-cycles. Bidirectional model transformation automates these tasks and, therefore, plays a crucial role in MDE.

Triple Graph Grammars (TGGs) [17] are a declarative, rule-based technique for specifying bidirectional model transformation. Bidirectionality in this context means that *forward* (source to target) and *backward* (target to source) transformations are derived from the same TGG specification. Formalizing models as graphs, a TGG specification comprises *triple rules* that describe how consistent source and target graphs connected by a correspondence graph evolve simultaneously, and is thus a *grammar* over *triple graphs*.

For practical applications, TGGs are typically *operationalized* to deduce forward and backward transformations. The main idea of the operationalization, e.g., in the forward direction, is to decompose each triple rule into a source part, parsing the elements of a given source model, and a forward part, creating necessary correspondence and target model elements to perform the specified transformation step. The same applies analogously to the backward direction.

A crucial limitation when tackling complex transformation tasks is that triple rules are graph patterns of fixed size and cannot describe transformation steps whose size depends on concrete models. In particular, transforming an element to arbitrarily many elements in one step depending on the transformation context is not possible as the context of an unknown size cannot be specified via fixed patterns. To overcome this, we propose an extension to TGGs leveraging a formal concept from classical graph transformation, namely *amalgamation*.

Amalgamation [1] combines the applications of two rules (called *multi-rules*) over a shared application of a common subrule (called *kernel rule*). The concept is generalized in [19] to combining n multi-rule applications, which is formalized in [7] as *multi-amalgamation* within the algebraic framework for adhesive categories. Single transformation steps are specified via *interaction schemes* that contain a kernel rule and multi-rules that embed this kernel rule. Depending on a concrete model at transformation time, the multi-rules are combined over the kernel rule to a *multi-amalgamated rule*. Intuitively, this provides a means for repeating certain parts of a transformation step after a common kernel part in a for each loop-like manner. The main challenge when incorporating multi-amalgamation into TGGs is to revise their operationalization, i.e., to derive forward (backward) transformations compatible with the combination process.

After discussing the shortcomings of TGGs without multi-amalgamation via a compact but non-trivial example in Sect. 2, our contribution is to:

1. Extend the basic formalization of TGGs by multi-amalgamation in Sect. 3.
2. Operationalize multi-amalgamated TGGs in Sect. 4, yielding our main result, namely multi-amalgamation with source-forward derivations (Thm.1).
3. Define model transformation with multi-amalgamated TGGs and its formal properties in Sect. 5, based on our operationalization results.

Section 6 gives an overview of related work and Sect. 7 concludes the paper. While this paper focuses on formal results, we refer to [12] for our tool support.

2 Running Example and Preliminaries

Our running example is a compact but nontrivial excerpt of a transformation between class diagrams and a corresponding HTML-like documentation (e.g., Javadoc). In particular, we focus on transforming inheritance links in the class diagrams to hyperlinks in the documents (and vice versa). Direct hyperlinks are to be created for all transitive super classes. While allowing multiple inheritance, we consider class diagrams without repeated inheritance for simplicity, i.e., a transitive inheritance is not induced over multiple ways. An exemplary class diagram and its consistent documentation is depicted in Fig. 1 in concrete syntax.

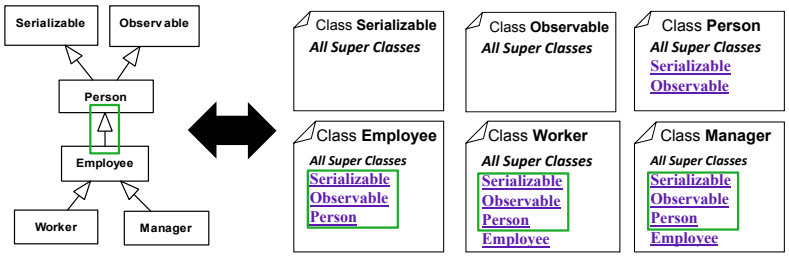


Fig. 1. A class diagram and its corresponding documentation

An inheritance link corresponds to multiple hyperlinks whose exact number can only be determined at transformation time for a concrete class diagram. Consider the transformation of the inheritance link between the Employee and Person classes in Fig.1 and assume all other inheritance links are already documented. Besides creating a hyperlink from the Employee document to the Person document, three additional steps are necessary: (i) the subclass document must get hyperlinks to the documents for new transitive super classes (in this case from Employee to Serializable and Observable), (ii) documents for all transitive subclasses must get a hyperlink to the super class document (from Worker and Manager to Person), and (iii) documents for transitive subclasses must get hyperlinks to the documents for transitive super classes (from Worker and Manager to Serializable and Observable in all four possible combinations). The transformation of one inheritance link in our concrete case creates 1+(2+2+4)=9 hyperlinks in the documentation. The portion in brackets ranges between 0 and arbitrarily many depending on concrete models.

2.1 Consistency Specification with Triple Graph Grammars

In this section, we briefly review the existing TGG formalization and look closer at our identified challenges with the running example. In line with the algebraic formalization in [4], we formalize *models* and *metamodels* as *typed graphs* and *type graphs*, respectively. For presentation purposes, we provide our formalization on the level of typed graphs. The formalization can, however, be extended compatibly to *attributed* typed graphs with *type inheritance* [4].

Definition 1 (Typed Graph and Typed Graph Morphism).
A graph $G = (V, E, s, t)$ *is defined by a set* V *of vertices, a set* E *of edges, and two functions* $s, t : E \to V$ *assigning to each edge a source and target vertex, respectively. A* graph morphism $f : G \to G'$, *with* $G' = (V', E', s', t')$, *is defined as a pair of functions* $f := (f_V, f_E)$ *such that* $f_V : V \to V'$, $f_E : E \to E'$ *and* $f_V \circ s = s' \circ f_E \wedge f_V \circ t = t' \circ f_E$.

A type graph *is a distinguished graph* $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG})$. *A* typed graph *is a pair* $(G, type)$ *of a graph* G *and a graph morphism* type*:* $G \to TG$. *Given* $(G, type)$ *and* $(G', type')$, $f : G \to G'$ *is a* typed graph morphism *iff* $type = type' \circ f$. $\mathcal{L}(TG)$ *denotes the set of all typed graphs of type* TG

We now introduce *triples* of graphs as we shall be dealing with *source* and *target* models connected via a *correspondence* model. Normal letters denote triple graphs while single graphs have subscripts S, C, or T indicating their domains.

Definition 2 (Typed Triple Graph, Typed Triple Graph Morphism). *A* triple graph $G := G_S \overset{\gamma_S}{\leftarrow} G_C \overset{\gamma_T}{\rightarrow} G_T$ *consists of typed graphs* $G_X \in \mathcal{L}(TG_X)$, $X \in \{S, C, T\}$, *and morphisms* $\gamma_S : G_C \rightarrow G_S$ *and* $\gamma_T : G_C \rightarrow G_T$.

A triple morphism $f : G \rightarrow G'$ *with* $G' = G'_S \overset{\gamma'_S}{\leftarrow} G'_C \overset{\gamma'_T}{\rightarrow} G'_T$, *is a triple* $f : (f_S, f_C, f_T)$ *of typed morphisms where* $f_X : G_X \rightarrow G'_X$ *and* $X \in \{S, C, T\}$, $f_S \circ \gamma_S = \gamma'_S \circ f_C$ *and* $f_T \circ \gamma_T = \gamma'_T \circ f_C$. *A* type triple graph *is a distinguished triple graph* $TG = TG_S \overset{\Gamma_S}{\leftarrow} TG_C \overset{\Gamma_T}{\rightarrow} TG_T$. *A* typed triple graph *is a pair* $(G, type)$ *of a triple graph* G *and triple morphism* $type : G \rightarrow TG$. *Given* $(G, type)$ *and* $(G', type')$, $f : G \rightarrow G'$ *is a* typed triple graph morphism *iff* $type = type' \circ f$. $\mathcal{L}(TG)$ *denotes the set of all triple graphs of type* TG.

Example 1. Fig. 2 depicts a type triple graph on the left, and a typed triple graph on the right. We choose class diagrams as the source domain and documents as the target domain. Hexagon-shaped vertices in the middle form the correspondence domain. In our type triple graph, class diagrams consist of Classes that might have super Classes. Accordingly, hyperlinked documents consist of Docs (representing documents) that might reference each other via a ref edge (representing hyperlinks). The correspondence type C2D relates Classes to Docs.
The exemplary typed triple graph to the right conforms to our type triple graph and represents the same pair of class diagram and documentation model depicted in Fig. 1, now with explicit correspondences in the middle. The structural difference between the documentation and its class diagram is the presence of explicit ref edges to the documents of all transitive super classes. Note that the super edge between the vertices 3 and 4 represents the inheritance between Employee and Person in Fig. 1 and will be used in further examples.

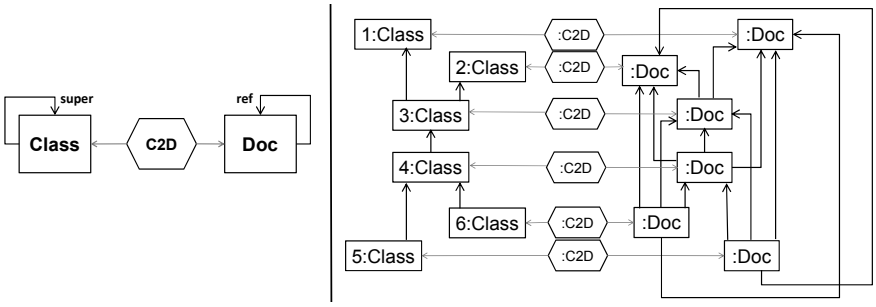
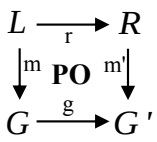


Fig. 2. a) A type triple graph and b) A typed triple graph for the running example

Consistent source, correspondence, and target graphs are created simultaneously with *triple rules*. TGGs only support *monotonic* (i.e., non-deleting) rules as they are a constructive description for a triple of graph languages. We simplify

the algebraic formalization [4] accordingly for monotonic rules. In this paper, we consider rules without *negative application conditions* (NACs) [4] and leave the lifting of all concepts to TGGs with NACs to future work.

Definition 3 (Monotonic Triple Rules and Derivations). *Given a type triple graph TG, a monotonic* triple rule $r : L \to R$ *is a typed triple monomorphism with* $L, R \in \mathcal{L}(TG)$*. A* direct derivation*, denoted as* $G \overset{r@m}{\Longrightarrow} G'$*, is constructed by building a pushout as depicted in the diagram to the right, i.e., by applying* r *to a typed triple graph* $G \in \mathcal{L}(TG)$ *via a typed triple morphism* $m : L \to G$*. The typed triple morphisms* m *and* m' *are referred to as* match *and* comatch*, respectively. We call* g *a* direct derivation morphism*.*



A sequence $d : G \overset{r_0@m_0}{\Longrightarrow} G_1 \overset{r_1@m_1}{\Longrightarrow} \ldots \overset{r_n@m_n}{\Longrightarrow} G'$ *with respective direct derivation morphisms* $\{g_0, \ldots, g_n\}$ *is a* derivation *where* $g = g_n \circ \ldots \circ g_0$ *denotes the* derivation morphism*. The decomposition of* d *into derivations* $d_1 : G \overset{r_0@m_0}{\Longrightarrow} G_1 \ldots \overset{r_i@m_i}{\Longrightarrow} G_i$ *and* $d_2 : G_i \overset{r_{i+1}@m_{i+1}}{\Longrightarrow} G_{i+1} \ldots \overset{r_n@m_n}{\Longrightarrow} G'$ *is denoted as* (d_1, d_2)*.*

Example 2. Figure 3 depicts two triple rules in an attempt to specify a TGG for our running example. As triple rules are monotonic, we use a compact syntax embedding L (context elements, i.e., the precondition of the rule) into R and depicting created elements ($R \backslash L$) in green with a ++ mark-up.

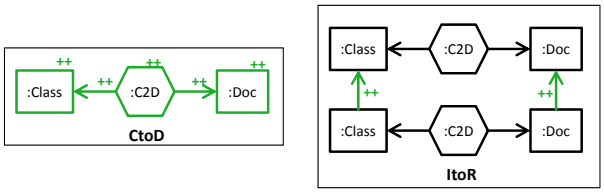
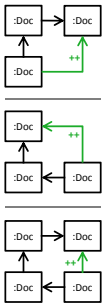


Fig. 3. Triple rules for the running example

The first rule CtoD (Class to Document) does not require any elements as context and creates a class and a document with a correspondence between them. The second rule ItoR (Inheritance to Reference) requires two classes and their corresponding documents. It creates an inheritance link (super) from one class to the other, and a hyperlink (ref) between the documents in the same direction.

A triple rule creates in general a *fixed* number c_S and c_t of source and target elements, respectively. The rule ItoR, for example, creates an inheritance link ($c_s = 1$) and a hyperlink ($c_t = 1$). As we have discussed at the beginning of this section, however, after creating a hyperlink we need to repeat three additional steps to *complement* all corresponding hyperlinks. In total, the desired consistency requires the creation of $c_t = 1 + c_{t_1} + c_{t_2} + c_{t_3}$ target elements, where c_{t_1}, c_{t_2}, and c_{t_3} are the numbers of hyperlinks to be created for these three cases. They can only be determined at transformation time for a concrete model triple.

Note that specifying three further separate rules as depicted to the right to create the missing hyperlinks in retrospect is not a solution, as the application of these rules cannot be enforced exactly once for c_{t_1}, c_{t_2}, and c_{t_3} cases. The resulting grammar would allow missing as well as superfluous transitive hyperlinks, leading to transformations that exhibit undesired behaviour. To express consistency in situations where the number of elements that are to be related in one step is unknown at design time of a TGG, we propose the integration of multi-amalgamation as established in classical graph transformation, adhering to the rule-based nature of TGGs.

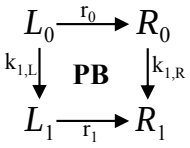


3 Multi-Amalgamated Triple Graph Grammars

The Amalgamation Theorem [1] combines applications of two rules (called *multi-rules*) over an embedded subrule (*kernel rule*) application. This concept is generalized to combining an arbitrary number of direct derivations via multi-rules [19], formalized as *multi-amalgamation* in [7] within a categorical setting. Single transformation steps in multi-amalgamation are specified as *interaction schemes* that consist of a kernel rule and multi-rules that embed this kernel rule. When applying an interaction scheme to a concrete model, the multi-rules are glued over the kernel rule depending on the collected multi-rule matches, i.e., the size of the gluing is first determined at transformation time. With regard to TGGs, therefore, interaction schemes can be regarded as a generalization of triple rules with which consistency of an unknown number of involved elements can be expressed.

Our goal in this section is to integrate multi-amalgamation into the basic formalization of TGGs. We use the general framework of multi-amalgamation as introduced in [7] but simplify the algebraic formalization by exploiting the monotonicity of triple rules. As from now on, we refer to a family of morphisms or direct derivations that start from the same typed triple graph as a *bundle*.

Definition 4 (Kernel Rule, Multi-Rule, Interaction Scheme). *Given triple rules r_0 and r_1, a* kernel morphism $k_1 : r_0 \to r_1$ *consists of two typed triple monomorphisms $k_{1,L} : L_0 \to L_1$ and $k_{1,R} : R_0 \to R_1$ such that the square to the right is a pullback, i.e., r_1 at least includes r_0 and might have a remainder. In this case, r_0 is called the* kernel rule *and r_1 the* multi-rule. *A kernel rule r_0 and a set of multi-rules $\{r_1, \ldots, r_n\}$ with the respective kernel morphisms $\{k_1, \ldots, k_n\}$ form an* interaction scheme.



Example 3. Fig. 4 depicts the interaction scheme ItoR consisting of a kernel rule ItoR_0 and three multi-rules ItoR_1, ItoR_2, and ItoR_3 embedding the kernel rule via the kernel morphisms k_1, k_2, and k_3, respectively. In a multi-rule, the vertices originating from the kernel rule are highlighted via a gray shading. Consequently, the white vertices and their incident edges form the remainder after the kernel. The kernel rule ItoR_0 is our original rule from Fig. 3 and creates an inheritance link between two classes and a hyperlink between the respective super class

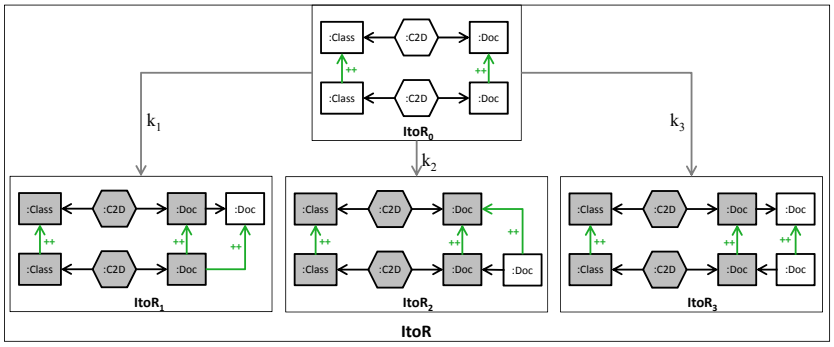


Fig. 4. Interaction scheme ItoR

and subclass documents. The multi-rule ItoR_1 includes the kernel rule and additionally creates a hyperlink from the subclass document to a transitive super class document. Analogously, ItoR_2 creates a hyperlink from a transitive subclass document to the super class document. Finally, ItoR_3 creates a hyperlink from a transitive subclass document to a transitive super class document. The remainders of these multi-rules create hyperlinks between two documents as soon as they are indirectly connected by the kernel part. Hyperlinks for transitive inheritance relations of an arbitrary depth can therefore be created.

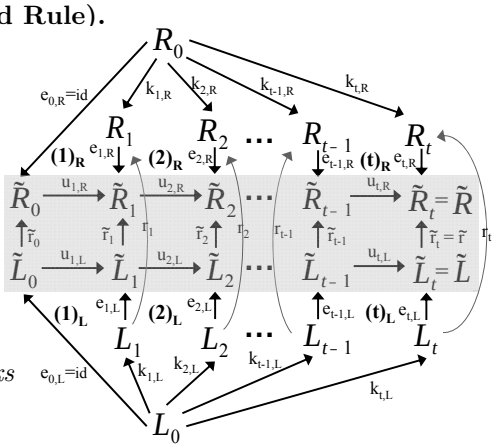
Next, we consider a bundle of direct derivations consisting of a kernel rule application and multi-rule applications that embed this kernel rule application. Moreover, we require maximal and unique multi-rule matches, which is essential to achieve transformations behaving in line with a *for each* loop.

Definition 5 (Maximally Amalgamable).
Given an interaction scheme s and a typed triple graph G, let $D : \{G \overset{r_j@m_j}{\Longrightarrow} G_j\}_{j=0,\ldots,t}$ be a bundle of direct derivations, where r_0 is the kernel rule of s and $\{r_1, \ldots, r_t\}$ are multi-rules of s with the respective kernel morphisms $\{k_1, \ldots, k_t\}$. D is amalgamable *for s if, $\forall p, q \in \{1, \ldots, t\}$ all multi-rule matches are (1) unique, i.e., $p \neq q \Rightarrow m_p \neq m_q$, and (2) agree on the kernel match m_0, i.e., $m_p \circ k_p = m_q \circ k_q = m_0$. We say D is* maximally amalgamable *for s if $\nexists d_z : G \overset{r_z@m_z}{\Longrightarrow} G_z$ such that $(D \cup \{d_z\})$ is amalgamable for s.*

Example 4. We now consider the interaction scheme ItoR from Fig. 4, applied to create the inheritance link between the classes Person and Employee (the vertices 3 and 4 in Fig. 2) with the corresponding hyperlinks. We assume that all other inheritance links were already created with all respective hyperlinks. The maximally amalgamable bundle applies ItoR_0 once in order to create the inheritance link with the corresponding direct hyperlink, ItoR_1 twice (by matching the Serializable and Observable documents as the white vertex), ItoR_2 twice (by matching the Worker and Manager documents as the white vertex), and ItoR_3 four times (by matching the Serializable and Observable documents as the upper white vertex and the Worker and Manager documents as the lower white vertex).

The consolidation of a maximally amalgamable bundle results in a direct derivation with a *multi-amalgamated rule*.

Definition 6 (Multi-Amalgamated Rule). *Given an interaction scheme s, and a bundle $D : \{G \xRightarrow{r_j@m_j} G_j\}_{j=0,\dots,t}$ of direct derivations that is maximally amalgamable for s, let $\tilde{K} : \{k_i = r_0 \to r_i\}_{i=1,\dots,t}$ be the bundle of respective kernel morphisms for D. The* multi-amalgamated rule $\tilde{r} : \tilde{L} \to \tilde{R}$ *is constructed by gluing multi-rules over the kernel rule via iterated pushouts with the kernel morphisms in $\tilde{K}$ as depicted to the right, where the gray region marks the results after each iteration: The construction starts with $\tilde{r}_0 = r_0$, i.e., the kernel rule, and ends with $\tilde{r} = \tilde{r}_t$. After each iteration $i \in \{1, \dots, t\}$, the pushouts $(i)_L$ and $(i)_R$ construct $\tilde{L}_i$ and $\tilde{R}_i$, respectively. The rule morphism $\tilde{r}_i : \tilde{L}_i \to \tilde{R}_i$ is induced via the universal property of the pushout $(i)_L$, i.e., $\tilde{r}_i \circ u_{i,L} = u_{i,R} \circ \tilde{r}_{i-1}$ and $\tilde{r}_i \circ e_{i,L} = e_{i,R} \circ r_i$. We call $G \xRightarrow{\tilde{r}@\tilde{m}} G'$ a* multi-amalgamated direct derivation *where $\tilde{m}$ is determined by the multi-rule matches in D, i.e., $\tilde{m} \circ e_{t,L} = m_t$ and $\tilde{m} \circ (u_{t,L} \circ \dots \circ u_{q+1,L}) \circ e_{q,L} = m_q$, $\forall q \in \{0, \dots, t-1\}$.*



Example 5. The multi-amalgamated rule for the maximally amalgamable bundle from Ex. 4 is constructed by gluing ItoR_1 twice, ItoR_2 twice, and ItoR_3 four times over ItoR_0. Figure 5 depicts this multi-amalgamated rule with its match $\tilde{m}$ where vertices matching the same Doc are merged to one vertex.

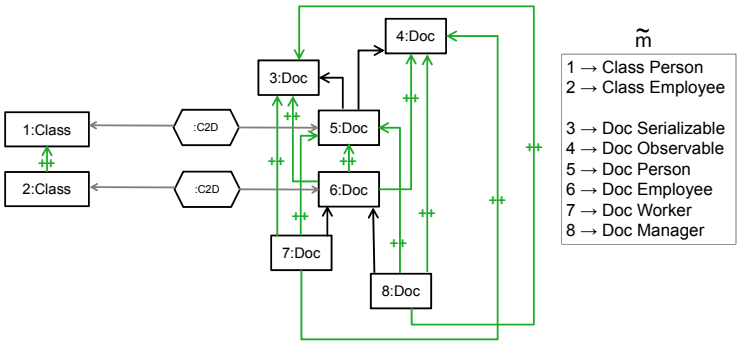


Fig. 5. A multi-amalgamated rule an its match for our example

Note that such a multi-amalgamated rule is not specified explicitly by the transformation designer but induced given a triple graph at transformation time. The multi-amalgamated direct derivation $\tilde{d} : G \xRightarrow{\tilde{r}@\tilde{m}} G'$ in this case creates one inheritance link in the source graph and nine hyperlinks in the target graph.

Having introduced interaction schemes of triple rules and multi-amalgamated direct derivations, we finally define *multi-amalgamated TGGs*.

Definition 7 (Multi-Amalgamated Triple Graph Grammar).
A multi-amalgamated triple graph grammar $TGG = (TG, \mathcal{S})$ *consists of a type triple graph* TG *and a set* S *of interaction schemes. The generated language* $\mathcal{L}(TGG) \subseteq \mathcal{L}(TG)$ *is defined as follows:*
$\mathcal{L}(TGG) := \{G_\emptyset\} \cup \{G \mid \exists \tilde{d} : G_\emptyset \overset{\tilde{r}_1 @ \tilde{m}_1}{\Longrightarrow} G_1 \overset{\tilde{r}_2 @ \tilde{m}_2}{\Longrightarrow} \dots \overset{\tilde{r}_n @ \tilde{m}_n}{\Longrightarrow} G\}$*, where* $G_\emptyset$ *is the empty triple graph, each* $\tilde{r}_i$ *with* $i \in \{1, \dots, n\}$ *is a multi-amalgamated rule derived from an interaction scheme* $s_i \in \mathcal{S}$ *and* G_{i-1}*.* $\mathcal{L}_S(TGG)$ *denotes all source graphs in* $\mathcal{L}(TGG)$*,* $\mathcal{L}_T(TGG)$ *analogously all target graphs.*

Example 6. For a uniform handling, we consider CtoD from Fig. 3 also as an interaction scheme with an empty set of multi-rules. The interaction schemes CtoD and ItoR (Fig.4) together with the type triple graph in Fig. 2 constitute a multi-amalgamated TGG, which is indeed able to generate class diagrams with multiple inheritance and corresponding documents with all necessary hyperlinks.

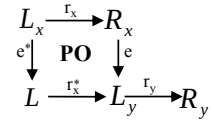
4 Operationalizing Multi-Amalgamated TGGs

In this section, our goal is to operationalize interaction schemes in order to deduce forward and backward transformation steps from a multi-amalgamated TGG. From an interaction scheme s, we derive source and forward rules to achieve forward transformation steps that are equivalent to a multi-amalgamated direct derivation via s. All concepts apply analogously to the backward direction.

We apply two decompositions to interaction schemes, making use of the Concurrency Theorem [5], which states that two sequential direct derivations can be composed to (or decomposed from) a direct derivation with a so-called *E-concurrent rule*. The following definition of an E-concurrent rule is a special case of Def. 5.21 in [4]. We only consider the E-concurrent rule of two monotonic rules $r_x : L_x \to R_x$ and $r_y : L_y \to R_y$, where R_x can be embedded in L_y.

Definition 8 (E-Concurrent Rule).

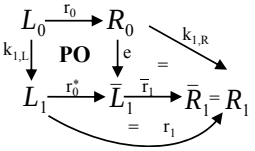
Given triple rules $r_x : L_x \to R_x$ *and* $r_y : L_y \to R_y$*, a triple morphism* $e : R_x \to L_y$*, as depicted to the right, is an* E-dependency relation *over* r_x *and* r_y *if the pushout complement (i.e.,* $e^* : L_x \to L$ *and* $r_x^* : L \to L_y$*) exists. The corresponding* E-concurrent rule $r_x *_E r_y$ *is defined as* $r_y \circ r_x^* : L \to R_y$



First, we derive so-called *complement rules* from the multi-rules. A complement rule accomplishes the remainder of a multi-rule after its kernel [1], i.e., a multi-rule is an E-concurrent rule of its kernel and complement rule.

As discussed in Sect. 3, multi-amalgamated rules are *dynamically* constructed and are used to define the semantics of an interaction scheme for a particular triple graph at transformation time. Complement rules, by contrast, are statically constructed to realise multi-amalgamated direct derivations via repeated application, representing a practical means of implementing multi-amalgamation [12].

Definition 9 (Complement Rule).
Given a kernel morphism $k_1 : r_0 \to r_1$, the respective complement rule $\overline{r}_1 : \overline{L}_1 \to \overline{R}_1$ *is constructed, as depicted to the right, such that $\overline{L}_1$ is a pushout over r_0 and $k_{1,L}$ and $\overline{R}_1 = R_1$. The rule morphism $\overline{r}_1$ is induced uniquely via the universal property of the pushout.*



Example 7. Figure 6 depicts the complement rules of the interaction scheme ItoR from Fig. 4. The complement rules $\overline{\mathsf{ItoR}}_1$, $\overline{\mathsf{ItoR}}_2$, and $\overline{\mathsf{ItoR}}_3$ correspond to the multi-rules ItoR_1, ItoR_2, and ItoR_3, respectively, and only create the transitive hyperlinks for an existing pair of an inheritance link and a direct hyperlink.

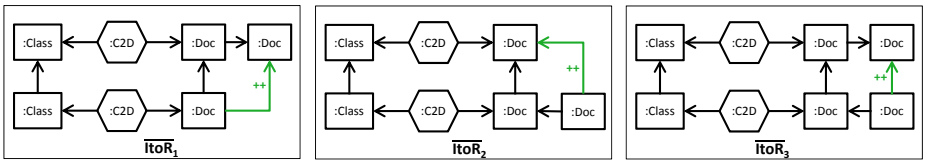


Fig. 6. Complement rules for ItoR

Complement rules allow us to decompose multi-amalgamated direct derivations into a kernel direct derivation and a sequence of complement direct derivations. Having required maximal multi-matches in Def. 6, we now define the analogous characterization for decomposed derivations with complement rule matches.

Definition 10 (Maximally Complemented Bundle).
Given an interaction scheme s with the respective set $\overline{CR}$ of complement rules and kernel rule r_0, and a typed triple graph G, let $d_0 : G \overset{r_0@m_0}{\Longrightarrow} G_0$ be a direct derivation via r_0 with comatch m'_0 and $\overline{D} : \{G_0 \overset{\overline{r}_i@\overline{m}_i}{\Longrightarrow} H_i\}_{i=1,\dots,t}$ a bundle of direct derivations where $(\overline{r}_i : \overline{L}_i \to \overline{R}_i) \in \overline{CR}$ with $e_i : R_0 \to \overline{L}_i$.
$\overline{D}$ is complemented *for d_0, if, $\forall p, q \in \{1, \dots, t\}$, all complement matches are (1) unique, i.e., $p \neq q \Rightarrow \overline{m}_p \neq \overline{m}_q$, and (2) agree on the kernel comatch m'_0, i.e., $\overline{m}_p \circ e_p = \overline{m}_q \circ e_q = m'_0$. $\overline{D}$ is* maximally complemented *for d_0 if $\nexists \overline{d}_z : G_0 \overset{\overline{r}_z@\overline{m}_z}{\Longrightarrow} H_z$ such that $(\overline{D} \cup \{\overline{d}_z\})$ is complemented for d_0.*

The following lemma states the equivalence of a multi-amalgamated direct derivation (Def. 6) to a derivation with a kernel and subsequent complement rule applications. The complement rule applications form a maximal bundle (Def.10). This yields our first decomposition to operationalize a multi-amalgamated TGG.

Lemma 1 ((De-)composition of Multi-Amalg. Direct Derivations).
Given an interaction scheme s and a typed triple graph G, $\exists(\tilde{d} : G \overset{\tilde{r}@\tilde{m}}{\Longrightarrow} G') \Leftrightarrow \exists(\overline{d} : G \overset{r_0@m_0}{\Longrightarrow} G_0 \overset{\overline{r}_1@\overline{m}_1}{\Longrightarrow} G_1 \dots \overset{\overline{r}_t@\overline{m}_t}{\Longrightarrow} G_t = G')$, where $\tilde{d}$ is a multi-amalgamated direct derivation with s and the bundle $\overline{D} : \{G_0 \overset{\overline{r}_i@\overline{m}_i}{\Longrightarrow} H_i\}_{i=1,\dots,t}$ is maximally complemented for $d_0 : G \overset{r_0@m_0}{\Longrightarrow} G_0$.

Proof. Using the Multi-Amalgamation Theorem [7], as depicted to the right, $\tilde{d}$ can be decomposed into (or composed from) a direct derivation $d_0 : G \overset{r_0@m_0}{\Longrightarrow} G_0$ via the kernel rule r_0, and a subsequent direct derivation q that accomplishes the remainder of $\tilde{d}$ via the complement rules of s. As $\tilde{d}$ is constructed via a maximally amalgamable bundle $D : \{G \overset{r_j@m_j}{\Longrightarrow} H_j\}_{j=0,\ldots,t}$ (Def. 5), the remainder q corresponds to the maximally complemented bundle $\overline{D} : \{G_0 \overset{\overline{r}_i@\overline{m}_i}{\Longrightarrow} H_i\}_{i=1,\ldots,t}$ applied in one step. Moreover, all direct derivations in $\overline{D}$ are pairwise parallel independent as they only require d_0. The Parallelism Theorem [6] leads to the equivalence of q with the sequence $G_0 \overset{\overline{r}_1@\overline{m}_1}{\Longrightarrow} G_1 \ldots \overset{\overline{r}_t@\overline{m}_t}{\Longrightarrow} G_t$. That is, $\overline{d}$ is equivalent to (d_0, q), and thus to $\tilde{d}$. □

Definition 11 (Maximally Complemented Derivation).
Given a multi-amalgamated direct derivation $\tilde{d} : G \overset{\tilde{r}@\tilde{m}}{\Longrightarrow} G'$ via an interaction scheme s, we refer to $\overline{d} : G \overset{r_0@m_0}{\Longrightarrow} G_0 \overset{\overline{r}_1@\overline{m}_1}{\Longrightarrow} G_1 \ldots \overset{\overline{r}_t@\overline{m}_t}{\Longrightarrow} G_t = G'$, the derivation induced according to Lemma 1, as maximally complemented *for s.*

Example 8. The multi-amalgamated direct derivation $\tilde{d}$ presented in Ex. 5 can be decomposed into (or composed from) a derivation $\overline{d}$ that is maximally complemented for ItoR as follows:
$\overline{d} : G \overset{\mathsf{ItoR}_0@m_0}{\Longrightarrow} G_0 \overset{\overline{\mathsf{ItoR}}_1@\overline{m}_1}{\Longrightarrow} G_1 \overset{\overline{\mathsf{ItoR}}_1@\overline{m}_2}{\Longrightarrow} G_2 \overset{\overline{\mathsf{ItoR}}_2@\overline{m}_3}{\Longrightarrow} G_3 \overset{\overline{\mathsf{ItoR}}_2@\overline{m}_4}{\Longrightarrow} G_4 \overset{\overline{\mathsf{ItoR}}_3@\overline{m}_5}{\Longrightarrow}$
$G_5 \overset{\overline{\mathsf{ItoR}}_3@\overline{m}_6}{\Longrightarrow} G_6 \overset{\overline{\mathsf{ItoR}}_3@\overline{m}_7}{\Longrightarrow} G_7 \overset{\overline{\mathsf{ItoR}}_3@\overline{m}_8}{\Longrightarrow} G_8 = G'$
This corresponds to the creation of an inheritance link and a direct hyperlink with the kernel rule, and eight transitive hyperlinks with complement rules.

Next, we apply basic operationalization results [3,17] for TGGs to kernel and complement rules in order to decompose them further into their *source* and *forward rules*. A source rule creates only source elements while the respective forward rule creates the correspondence and target elements. Thus, each kernel and complement rule is an E-concurrent rule of its source and forward rule. We apply this decomposition to kernel and complement rules, yielding a static construction of operationalized rules that together can achieve a multi-amalgamated direct derivation.

Definition 12 (Source and Forward Rules).

Given a triple rule $r : L \to R$ with $L = L_S \leftarrow L_C \to L_T$ and $R = R_S \leftarrow R_C \to R_T$, a source rule *$sr : SL \to SR$ is constructed such that $SL = L_S \leftarrow \emptyset \to \emptyset$ and $SR = R_S \leftarrow \emptyset \to \emptyset$, and a* forward rule *$fr : FL \to FR$ is constructed such that $FL = R_S \leftarrow L_C \to L_T$ and $FR = R_S \leftarrow R_C \to R_T$. The rule morphisms sr and fr are induced, as depicted in the diagram, such that r is an E-concurrent rule (Def. 8) of sr and fr. We call the E-dependency relation $e : SR \to FL$* source rule embedding. *Given a kernel morphism $k_1 : r_0 \to r_1$, we call sr_0 (fr_0) the kernel source (forward) rule and $\overline{sr}_1$ ($\overline{fr}_1$) the complement source (forward) rule.*

Example 9. Fig. 7 depicts the source and forward rules derived from the kernel and complement rules of the interaction scheme ltoR. The kernel source rule sltoR_0 creates an inheritance link between two classes while the kernel forward rule fltoR_0 requires such an inheritance link and creates a hyperlink between the corresponding documents. The complement source rules $\overline{\mathsf{sltoR}}_1$, $\overline{\mathsf{sltoR}}_2$, and $\overline{\mathsf{sltoR}}_3$ are identical (as $\overline{\mathsf{ltoR}}_1$, $\overline{\mathsf{ltoR}}_2$, and $\overline{\mathsf{ltoR}}_3$ are identical in their source parts), and require an inheritance link without creating any elements. The complement forward rules $\overline{\mathsf{fltoR}}_1$, $\overline{\mathsf{fltoR}}_2$, and $\overline{\mathsf{fltoR}}_3$ create a transitive hyperlink in accordance with $\overline{\mathsf{ltoR}}_1$, $\overline{\mathsf{ltoR}}_2$, and $\overline{\mathsf{ltoR}}_3$, respectively.

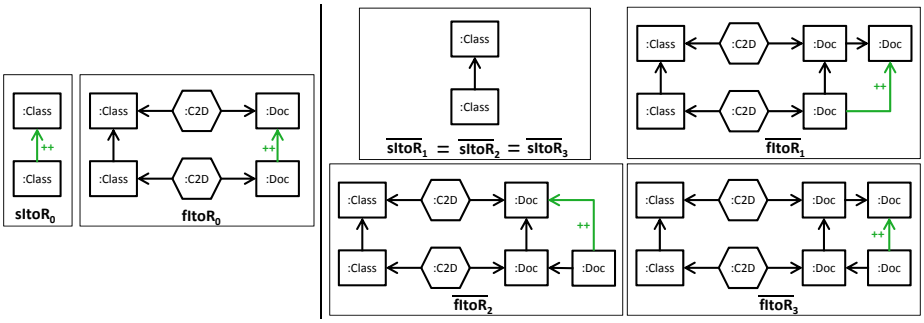


Fig. 7. Source and forward rules for ltoR

Source and forward rules enable us to decompose a derivation with triple rules via the Concurrency Theorem [5] into a source and forward derivation. The former creates the elements of a source graph via source rules while the latter extends it to a triple via forward rules. Inversely, a derivation with triple rules can be composed from such a source and forward derivation.

Fact 1 ((De-)composition of Derivations with Triple Rules)
Given triple rules $\{r_0, \dots, r_t\}$ *with their respective source rules* $\{sr_0, \dots, sr_t\}$, *forward rules* $\{fr_0, \dots, fr_t\}$, *and source rule embeddings* $\{e_0, \dots, e_t\}$,
$\exists(d : G \overset{r_0@m_0}{\Longrightarrow} G_0 \dots \overset{r_t@m_t}{\Longrightarrow} G_t) \Leftrightarrow \exists(sfd : G \overset{sr_0@sm_0}{\Longrightarrow} G_{s_0} \dots \overset{sr_t@sm_t}{\Longrightarrow} G_{s_t} \overset{fr_0@fm_0}{\Longrightarrow} G_{f_0} \dots \overset{fr_t@fm_t}{\Longrightarrow} G_{f_t} = G_t)$ *where each forward rule match* fm_i *is determined by* e_i *and the source rule comatch* sm'_i, *i.e,* $g_i \circ sm'_i = fm_i \circ e_i$ *while* g_i *is the derivation morphism* $G_{s_0} \to G_{s_t}$ *for* $i = 0$ *and* $G_{s_i} \to G_{f_{i-1}}$ *for* $i > 0$.

Proof. For the proof we refer the interested reader to Thm. 1 in [3]. □

Fact 1 is a general (de-)composition result for derivations with triple rules. Having kernel and complement rules as triple rules in case of a multi-amalgamated TGG, we apply Fact 1 to a maximally complemented derivation (Def. 11) in order to achieve an equivalent derivation with source and forward rules derived from the kernel and complement rules of an interaction scheme.

Definition 13 (Maximally Complemented Source-Forward Derivation). *Given a derivation* $\overline{d} : G \overset{r_0@m_0}{\Longrightarrow} G_0 \overset{\overline{r}_1@\overline{m}_1}{\Longrightarrow} G_1 \ldots \overset{\overline{r}_t@\overline{m}_t}{\Longrightarrow} G_t$ *that is maximally complemented for an interaction scheme* s*, we refer to the derivation* $\overline{sfd} : G \overset{sr_0@sm_0}{\Longrightarrow} G_{s_0} \overset{\overline{sr}_1@\overline{sm}_1}{\Longrightarrow} G_{s_1} \ldots \overset{\overline{sr}_t@\overline{sm}_t}{\Longrightarrow} G_{s_t} \overset{fr_0@fm_0}{\Longrightarrow} G_{f_0} \overset{\overline{fr}_1@\overline{fm}_1}{\Longrightarrow} G_{f_1} \ldots \overset{\overline{fr}_t@\overline{sm}_t}{\Longrightarrow} G_{f_t} = G_t$*, induced according to Fact 1, as a* maximally complemented source-forward derivation *for* s*.*

Finally, both (de-)compositions from Lemma 1 and Fact 1 lead to the equivalence of a multi-amalgamated direct derivation (Def. 6) and a maximally complemented source-forward derivation (Def. 13). The former describes a canonical and multi-amalgamated step for building up consistent triples while the latter is a conforming transformation step with source and forward rules.

Theorem 1 (Multi-Amalgamation with Source-Forward Derivations). *Given an interaction scheme* s *and a typed triple graph* G*,* $\exists(\tilde{d} : G \overset{\tilde{r}@\tilde{m}}{\Longrightarrow} G') \Leftrightarrow \exists(\overline{sfd} : G \overset{sr_0@sm_0}{\Longrightarrow} G_{s_0} \overset{\overline{sr}_1@\overline{sm}_1}{\Longrightarrow} G_{s_1} \ldots \overset{\overline{sr}_t@\overline{sm}_t}{\Longrightarrow} G_{s_t} \overset{fr_0@fm_0}{\Longrightarrow} G_{f_0} \overset{\overline{fr}_1@\overline{fm}_1}{\Longrightarrow} G_{f_1} \ldots \overset{\overline{fr}_t@\overline{sm}_t}{\Longrightarrow} G_{f_t} = G')$*, where* $\tilde{d}$ *is a multi-amalgamated direct derivation with* s *and* $\overline{sfd}$ *is a maximally complemented source-forward derivation for* s*.*

Proof. $\exists\tilde{d} \Leftrightarrow \exists\overline{d}$ (Lemma 1) and $\exists\overline{d} \Leftrightarrow \exists\overline{sfd}$ (Fact 1) where the intermediate derivation $\overline{d}$ is a maximally complemented derivation for s (Def. 10). □

Example 10. $\overline{d}$ from Ex.8, whose equivalence to $\tilde{d}$ from Ex. 5 is shown by applying Lemma 1, can be further decomposed into (or composed from) the following derivation $\overline{sfd}$ by applying Fact 1:

$\overline{sfd} : G \overset{\mathsf{sltoR}_0@sm_0}{\Longrightarrow} G_{s_0} \overset{\overline{\mathsf{sltoR}}_1@\overline{sm}_1}{\Longrightarrow} G_{s_1} \overset{\overline{\mathsf{sltoR}}_1@\overline{sm}_2}{\Longrightarrow} G_{s_2} \overset{\overline{\mathsf{sltoR}}_2@\overline{sm}_3}{\Longrightarrow} G_{s_3} \overset{\overline{\mathsf{sltoR}}_2@\overline{sm}_4}{\Longrightarrow} G_{s_4} \overset{\overline{\mathsf{sltoR}}_3@\overline{sm}_5}{\Longrightarrow} G_{s_5} \overset{\overline{\mathsf{sltoR}}_3@\overline{sm}_6}{\Longrightarrow} G_{s_6} \overset{\overline{\mathsf{sltoR}}_3@\overline{sm}_7}{\Longrightarrow} G_{s_7} \overset{\overline{\mathsf{sltoR}}_3@\overline{sm}_8}{\Longrightarrow} G_{s_8} \overset{\mathsf{fltoR}_0@fm_0}{\Longrightarrow} G_{f_0} \overset{\overline{\mathsf{fltoR}}_1@\overline{fm}_1}{\Longrightarrow} G_{f_1} \overset{\overline{\mathsf{fltoR}}_1@\overline{fm}_2}{\Longrightarrow} G_{f_2} \overset{\overline{\mathsf{fltoR}}_2@\overline{fm}_3}{\Longrightarrow} G_{f_3} \overset{\overline{\mathsf{fltoR}}_2@\overline{fm}_4}{\Longrightarrow} G_{f_4} \overset{\overline{\mathsf{fltoR}}_3@\overline{fm}_5}{\Longrightarrow} G_{f_5} \overset{\overline{\mathsf{fltoR}}_3@\overline{fm}_6}{\Longrightarrow} G_{f_6} \overset{\overline{\mathsf{fltoR}}_3@\overline{fm}_7}{\Longrightarrow} G_{f_7} \overset{\overline{\mathsf{fltoR}}_3@\overline{fm}_8}{\Longrightarrow} G_{f_8} = G'.$

The overall decomposition in this case corresponds to the transformation of an inheritance link to nine hyperlinks using source and forward rules derived from kernel and complement rules.

5 Model Transformation with Multi-Amalgamated TGGs

Having discussed the (de-)composition of multi-amalgamated direct derivations, we now apply our results to entire multi-amalgamated derivations, which generate the language of a TGG. This yields a notion of *source-forward transformation*, stating how a source graph G_S can be created via kernel and complement source rules and extended to a triple $G_S \leftarrow G_C \rightarrow G_T$ via kernel and complement forward rules. All results are analogously applicable in the backward direction.

Definition 14 (Source-Forward Transformation).
Given a multi-amalgamated TGG with a set $\mathcal{S}$ of interaction schemes and a source graph $G_S \in \mathcal{L}_S(TGG)$, a source-forward transformation *for G_S is a derivation $SFT_{G_S} : (\overline{sfd}_1, \ldots, \overline{sfd}_n)$ which (1) starts from the empty graph $G_\emptyset$, (2) creates a typed triple graph $G : G_S \leftarrow G_C \rightarrow G_T$, i.e., G_S is the source graph of G, and (3) consists of maximally complemented source-forward derivations $\overline{sfd}_i$, $i \in \{1, \ldots, n\}$, for an interaction scheme $s_i \in \mathcal{S}$.*

Remark: In a source-forward transformation, each maximally complemented source-forward derivation $\overline{sfd}_i$ is sorted in itself such that a source sequence is followed by a forward sequence, yielding together one multi-amalgamated transformation step (cf. Thm. 1). For readability, we do not undertake this sorting across different steps. Our proofs in the following, nonetheless, remain straightforwardly applicable to completely sorted source-forward transformations as a consequence of Fact 1, which holds orthogonally to multi-amalgamation.

Theorem 1 leads to the fact that a source-forward transformation can be composed to a sequence of multi-amalgamated direct derivations. A source-forward transformation thus produces a typed triple graph that is in the language of a multi-amalgamated TGG (Def. 7), referred to as *correctness*.

Theorem 2 (Correctness of SFT with Multi-Amalgamated TGGs).
Given a multi-amalgamated TGG, each source-forward transformation SFT_{G_S} is correct*, i.e., produces a typed triple graph $G \in \mathcal{L}(TGG)$.*

Proof. Let $SFT_{G_S} : (\overline{sfd}_1, \ldots, \overline{sfd}_n)$ where each $\overline{sfd}_i$ with $i \in \{1, \ldots, n\}$ is a maximally complemented source-forward derivation for an interaction scheme s_i. Applying Thm. 1 to each $\overline{sfd}_i$, we get a derivation $\tilde{d} : (\tilde{d}_1, \ldots, \tilde{d}_n)$ such that each $\tilde{d}_i$ is a multi-amalgamated direct derivation via s_i and $\tilde{d}$ is equivalent to SFT_{G_S}. That is, SFT_{G_S} produces a $G \in \mathcal{L}(TGG)$ according to Def. 7. □

Furthermore, Thm. 1 shows the decomposability of each multi-amalgamated direct derivation, and thus guarantees a forward transformation for each source graph $G_S \in \mathcal{L}_S(TGG)$, referred to as *completeness*.

Theorem 3 (Completeness of SFT with Multi-Amalgamated TGGs).
Given a multi-amalgamated TGG, there exists a source-forward transformation SFT_{G_S} for each $G_S \in \mathcal{L}_S(TGG)$.

Proof. Having $G_S \in \mathcal{L}_S(TGG)$, there is a derivation $\tilde{d} : (\tilde{d}_1, \ldots, \tilde{d}_n)$ such that $\tilde{d}$ creates a typed graph triple $G : G_S \leftarrow G_C \rightarrow G_T$ and every $\tilde{d}_i$ with $i \in \{1, \ldots, n\}$ is a multi-amalgamated direct derivation for an interaction scheme s_i (Def. 7). Applying Thm. 1 to each $\tilde{d}_i$, we get a derivation $SFT_{G_S} : (\overline{sfd}_1, \ldots, \overline{sfd}_n)$ such that each $\overline{sfd}_i$ is a maximally complemented source-forward derivation for s_i. □

6 Related Work

In the following, we consider two groups of related work: (1) alternative approaches to multi-amalgamation, which could also have been used to extend TGGs, and (2) other bidirectional languages and their support for "*for each*".

Alternatives to multi-amalgamation: Although different extensions to graph transformation exist for transforming arbitrarily many occurrences of certain patterns, to the best of our knowledge none of them have been integrated into TGGs. PROGRES [18] features *set nodes* that are to be matched optionally once (or at least once) and arbitrarily often. Multi-amalgamation is more expressive than set nodes as it handles multiple occurrences of graph patterns rather than single nodes. Extensions such as *collection operators* [8], *cloning* [10], or *rule quantification* [15] indicate that certain parts of a rule can be repeated. It is, however, challenging to determine how these extensions interact with splitting up triple rules into source and forward rules. Multi-amalgamation is the most natural way for TGGs to describe repetitions, as repeated parts are formalized via morphisms between plain rules. Basic source and forward rule construction results [3,17] remain directly applicable to kernel and complement rules. Nevertheless, rule quantification in [15] allows for hierarchical nesting of multi-rules, demonstrated in [16] on examples beyond the capabilities of multi-amalgamation. However, (de-)composition results such as complement rule construction and the Multi-Amalgamation Theorem [7], which enable a viable integration into TGGs as we discuss, are yet to be adapted for hierarchical multi-rules.

Bidirectional languages: *GRoundTram* [9], a *bidirectional programming* approach, features queries that are bidirectionally interpreted and inherently not bounded to a constant number of elements. Similarly, the QVT (*Query, View, Transformation*) standard [14], in particular QVT-R (QVT-Relations), features language constructs (e.g., $forall$ or *closure*) or recursive rule invocations to address the consistency of an unbounded number of involved elements. Adopting the QVT-R syntax, bidirectional approaches such as *Echo* [13], *JTL* [2], and *medini QVT* [11] allow for a *constraint-based* specification of consistency, and find consistent models by checking and enforcing constraint satisfaction. Although such approaches are more expressive than TGGs, TGGs have, nonetheless, gained acceptance due to efficient, scalable implementations and their constructive formal foundation based on graph transformation. Increasing the expressiveness of TGGs, however, is essential to ensure their competitiveness in an MDE context. Our contribution takes a step towards this goal.

7 Conclusion and Future Work

In this paper, we integrated the multi-amalgamation concept into TGGs. This enables us to derive forward and backward transformation steps that transform and create an unbounded number of elements where the number is determined via concrete models at transformation time. The achieved extension increase the capabilities of TGGs while adhering to their rule-based and declarative nature.

Further tasks for future work include support for (i) *incremental model synchronization* with multi-amalgamated TGGs, (ii) *critical pair analysis* [4] to ensure efficient model synchronization, (iii) *consistency checks* between existing models, (iv) *hierarchical multi-rules* comparable to nested for each loops and (v) NACs in interaction schemes to further increase expressiveness.

References

1. Boehm, P., Fonio, H.R., Habel, A.: Amalgamation of graph transformations: A synchronization mechanism. JCSS 34(2-3), 377–408 (1987)
2. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: JTL : a bidirectional and change propagating transformation language. In: Malloy, B.A., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 183–202. Springer (2010)
3. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information Preserving Bidirectional Model Transformations. In: Dwyer, M., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 72–86. Springer (2007)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer (2006)
5. Ehrig, H., Habel, A., Kreowski, H.J., Parisi-Presicce, F.: Parallelism and concurrency in high-level replacement systems. MSCS 1(03), 361–404 (1991)
6. Ehrig, H., Kreowski, H.J.: Parallelism of Manipulations in Multidimensional Information Structures. In: Mazurkiewicz, A. (ed.) MFCS 76. LNCS, vol. 45, pp. 285–293. Springer (1976)
7. Golas, U., Ehrig, H., Habel, A.: Multi-Amalgamation in Adhesive Categories. In: Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A. (eds.) ICGT 2010. LNCS, vol. 6372, pp. 346–361. Springer (2010)
8. Grønmo, R., Krogdahl, S., Møller-Pedersen, B.: A Collection Operator for Graph Transformation. In: Paige, R. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 67–82. Springer (2009)
9. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Nakano, K.: GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations. In: Alexander, P., Pasarenau, C.S., Hosking, J.G. (eds.) ASE 2011. pp. 480–483 (2011)
10. Hoffmann, B., Janssens, D., Van Eetvelde, N.: Cloning and Expanding Graph Transformation Rules for Refactoring. ENTCS 152, 53–67 (2006)
11. Ikv++: Medini QVT, `http://projects.ikv.de/qvt`
12. Leblebici, E., Anjorin, A., Schürr, A.: Tool Support for Multi-Amalgamated Triple Graph Grammars. In: this volume (2015)
13. Macedo, N., Cunha, A.: Implementing QVT-R Bidirectional Model Transformations using Alloy. In: Cortelessa, V., Varro, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 297–311. Springer (2013)
14. OMG: QVT Specification, V1.1 (2011), `http://www.omg.org/spec/QVT/1.1/`
15. Rensink, A.: Nested Quantification in Graph Transformation Rules. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 1–13. Springer (2006)
16. Rensink, A., Kuperus, J.H.: Repotting the Geraniums : On Nested Graph Transformation Rules. In: Boronat, A., Heckel, R. (eds.) GT-VMT 2009. ECEASST, vol. 18. EASST (2009)
17. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Tinhofer, G., Schmidt, G., Ernst, W.M. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer (1994)
18. Schürr, A.: Programmed Graph Replacement Systems. In: Rozenberg, G. (ed.) Handbook on Graph Grammars: Foundations, pp. 479–546. World Scientific (1997)
19. Taentzer, G.: Parallel and Distributed Graph Transformation : Formal Description and Application to Communication-Based Systems. Ph.D. thesis (1996)