Attributed Typed Triple Graph Transformation with Inheritance in the Double Pushout Approach

Esther Guerra
Departamento de Informática
Universidad Carlos III de Madrid
E-mail:eguerra@inf.uc3m.es

Juan de Lara
Escuela Politécnica Superior, Ing. Informática
Universidad Autónoma de Madrid
E-mail:jdelara@uam.es

18th October 2006

Resumen

Las gramáticas de grafos triples fueron propuestas por Andy Schürr con el objetivo de reescribir grafos triples [Schürr, 1994]. Son útiles para mantener relaciones entre dos modelos que se están manipulando. Este documento presenta una formalización de las gramáticas de grafos triples basada en el enfoque algebraico Double Pushout. Comenzaremos definiendo un grafo triple en base al concepto de $E - grafo$ propuesto en [Ehrig *et al.*, 2004b], el cual nos permite tener atributos en nodos y relaciones. A continuación usaremos los resultados obtenidos en [Ehrig *et al.*, 2004a], donde las categorías y los sistemas de reemplazo de alto nivel (HLR) adhesivos se utilizan como marco para la transformación de grafos. De este modo, la mayoría de los resultados obtenidos para la categoría de los grafos pueden ser extrapolados y aplicados a cualquier categoría HLR adhesiva. Por esta razón nuestra formalización demuestra que grafos triples y morfismos de grafo triples son una categoría HLR adhesiva, demostrando para ello que grafos triples y morfismos de grafo triples son isomorfos a una categoría coma. Además, extendemos la noción de regla de gramática de grafos triple con condiciones de aplicación, y con un concepto de herencia similar al propuesto en [Bardohl *et al.*, 2004], pero permitiendo también herencia en relaciones.

Abstract

Andy Schürr proposed triple graph grammars with the purpose of rewriting triple graphs [Schürr, 1994]. They are useful as a means to maintain the relations between two models that are being manipulated. This document presents a formalization of triple graph grammars based on

the Double Pushout algebraic approach. We first start by defining triple graphs using the concept of $E - graph$ proposed in [Ehrig *et al.*, 2004b], which allows attributes in edges and nodes. Then, we use the results in [Ehrig *et al.*, 2004a], which introduce adhesive high-level replacement (HLR) categories and systems as a framework for graph transformation. In this way, most results from the category of graphs are lifted to adhesive HLR categories. Therefore, our formalization uses the fact that triple graphs and triple morphisms can be shown to be an adhesive HLR category, by demonstrating that triple graphs and morphisms are isomorphic to a comma category. In addition, we provide triple graph grammar rules with application conditions, and with an inheritance concept similar to the one proposed in [Bardohl *et al.*, 2004], but allowing also inheritance of edges.

1 Introduction

Graph transformation [Ehrig *et al.*, 1999][Ehrig *et al.*, 2006] is a formal, high level, graphical and declarative means to manipulate graphs. Graph transformation systems are made of a set of graph transformation rules, each having graphs in their left and right hand sides (LHS and RHS). In order to apply a rule to a graph (called host graph), an occurrence of the LHS should be found in it. If some is found, then the rule is applied by replacing such occurrence by the RHS.

The theory of graph transformation has been developing in the last 30 years [Ehrig *et al.*, 1999]. The available results allow checking if two rules can be applied in any order yielding the same result (parallel and sequential independence), to check confluence (if a graph transformation system is deterministic), to amalgamate rules together through a common part (concurrency theorem), to check termination (under certain circumstances [Ehrig *et al.*, 2005b]), etc. See [Ehrig *et al.*, 2006] for a comprehensive review of the main results.

Andy Schürr proposed triple graph grammars (TGGs) with the purpose of rewriting triple graphs [Schürr, 1994]. These are structures made of three graphs called source, target and correspondence respectively. The latter is used to maintain relationships between the elements of the other two graphs. Triple graph grammars have been used for model transformation [Taentzer *et al.*, 2005] (translation of models from a source to a target formalism) and have the potential to be incremental. We have also used triple graphs to model multi-view visual languages [Guerra *et al.*, 2005] and to relate the concrete and abstract syntax of visual languages [Guerra and de Lara, 2004]. TGGs are becoming ever more popular as model-driven software development (MDSDM) techniques [Stahl and Völter, 2006] gain more attention. In MDSD, model to model and model to platform (i.e. code) transformations play a central role. The OMG promotes the model-driven architecture (MDA) [MDA] as a particular implementation of MDSD using some of the OMG's standards: UML, MOF and QVT [QVT]. The latter is a specification of a language for queries, views and transformations. The language for transformations combines both imper-

ative and declarative constructs, but it is not formally defined. It is our belief that the theory of TGGs has the potential to provide such formalization.

The aim of this report is to extend and formalize the concept of triple graph grammar presented in [Schürr, 1994]. In this way, in the new extension, we consider attributes in nodes and edges of triple graphs, and allow nodes in the correspondence graph to have morphisms either to nodes or to edges of the other graphs, as well as to remain undefined. We consider typing of triple graphs by a triple type graph (similar to the concept of type graph [Corradini *et al.*, 1996] or meta-model [Atkinson and Kühne, 2002]). Type graphs are provided with node inheritance relations, in a similar way as done in [Bardohl *et al.*, 2004]. Moreover, we also provide edge type inheritance. We formalize triple graph transformation using the double pushout approach [Ehrig *et al.*, 2006]. This is done by proving that triple graphs are indeed an adhesive HLR category. Moreover, as we have inheritance in type graphs, we allow rules to have abstract elements in their LHS. In this way, these elements can be matched to an instance of their subtypes. For the theoretic discussions, we have followed and extended the works in [Ehrig *et al.*, 2006] and [Bardohl *et al.*, 2004]. Triple graph grammars techniques were proposed in [Schürr, 1994] a means to derive lower-level, operational rules from creation grammars (i.e. grammars that model the synchronized creation of elements in the source and target graphs). The derived lower-level operational rules perform forward or backwards translations, incremental updates or so called consitency observing analyzers. In the present work, we provide a richer graph concept and a formalization of triple graph transformation in the DPO approach. However, the algorithms for derivation of operational rules for this richer graphs we propose are up to future work.

The rest of the document is organized as follows. Section 2 starts by presenting attributed typed graphs, which will be used later for our formalization of triple graphs. Section 3 formalizes triple graphs, with typing and attributes in nodes and edges. Section 4 shows how to build pushouts and pullbacks in this structure. These categorical constructions are needed in order to define graph transformation systems. Section 5 proves that the category of triple graphs and morphisms are indeed an adhesive HLR category. This means that we can use most of the theory of graph transformation systems, which has been lifted from graphs to HLR categories [Ehrig *et al.*, 2004a]. Section 6 presents the basic concepts of graph transformation explicitly for triple graphs. Although this is not necessary, since one could follow the theory of HLR, we show it for illustrative purposes. Section 7 adds the inheritance concept to triple graphs. Finally, section 8 presents the conclusions.

2 Attributed Typed Graphs

In this section, we define node and edge attributed typed graphs, following the notion of $E - graph$ developed in [Ehrig *et al.*, 2004b]. These definitions are included here, as we will use them later in section 3 as a basis for triple graphs. E-graphs are extended graphs that allow attributes in both nodes and edges.

Attribute values are stored in set V_D, and two additional kind of edges model attribution: the node and edge attribution edges. The first ones allow nodes to have attributes, while the second ones model edge attributes.

Definition 1 *(E-graph, taken from [Ehrig* et al.*, 2006])*
An E-graph is a tuple $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j\in\{G,NA,EA\}})$, *where:*

- V_G *is a set of graph nodes.*
- V_D *is a set of data nodes.*
- E_G *is a set of graph edges.*
- E_{NA} *is a set of "node attribution" edges.*
- E_{EA} *is a set of "edge attribution" edges.*
- $source_G \colon E_G \to V_G$, $target_G \colon E_G \to V_G$.
- $source_{NA} \colon E_{NA} \to V_G$, $target_{NA} \colon E_{NA} \to V_D$.
- $source_{EA} \colon E_{EA} \to E_G$, $target_{EA} \colon E_{EA} \to V_D$.

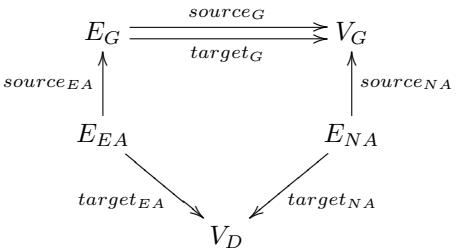


Figure 1: An E-graph.

Figure 2 shows an example of E-graph (using a graphical representation) depicting a sequence diagram. The diagram consist of two objects with one activation box each. The activation boxes are connected by message "msg1". Node *ActivationBox1* receives the start message (i.e. the entry point of the diagram).

In addition to E-graphs, we also define mappings between two E-graphs. An E-graph morphism is a tuple of set morphisms, one for each set in the E-graph (V_G, V_D, E_G, E_{NA}, E_{EA}). In addition, the structure of the E-graph should be preserved, that is, the $source_i$ and $target_i$ functions should commute with the morphisms.

Definition 2 *(E-graph morphism, taken from [Ehrig* et al.*, 2006])*
Given two E-graphs $G^i = (V_G^i, V_D^i, E_G^i, E_{NA}^i, E_{EA}^i, (source_j^i, target_j^i)_{j\in\{G,NA,EA\}})$, *with* $i = 1, 2$, *an E-graph morphism* $f \colon G^1 \to G^2$ *is a tuple* $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$

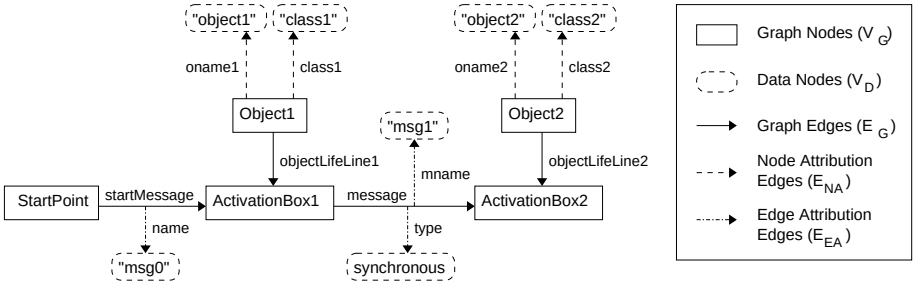


Figure 2: An E-graph representing a Sequence Diagram.

with $f_{V_i} : V_i^1 \rightarrow V_i^2$ *and* $f_{E_j} : E_j^1 \rightarrow E_j^2$ *with* $i \in \{G, D\}$, $j \in \{G, NA, EA\}$, *where* f *commutes for all source and target:*

- $f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}$
- $f_{V_G} \circ target_G^1 = target_G^2 \circ f_{E_G}$
- $f_{E_G} \circ source_{EA}^1 = source_{EA}^2 \circ f_{E_{EA}}$
- $f_{V_D} \circ target_{EA}^1 = target_{EA}^2 \circ f_{E_{EA}}$
- $f_{V_G} \circ source_{NA}^1 = source_{NA}^2 \circ f_{E_{NA}}$
- $f_{V_D} \circ target_{NA}^1 = target_{NA}^2 \circ f_{E_{NA}}$

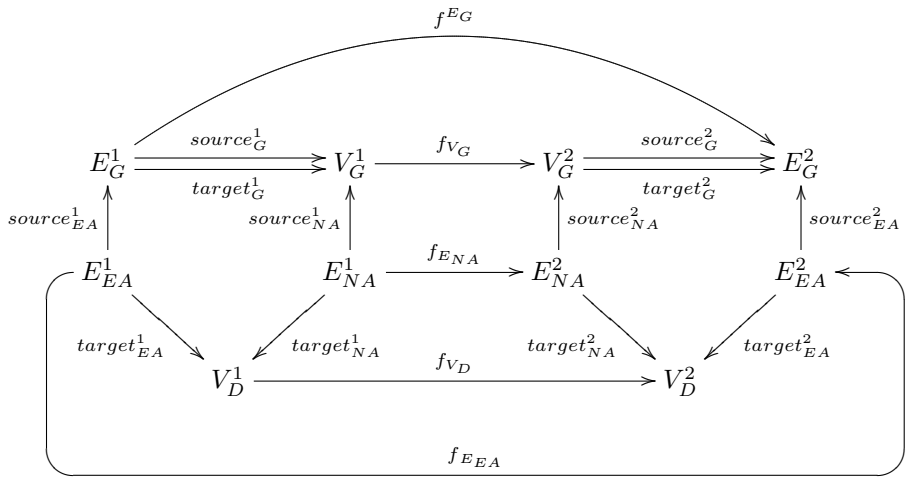


Figure 3: An E-graph morphism.

Figure 4 shows an example of E-graph morphism. Nodes and edges in the E-graphs have been provided with a numeric label for matching purposes. Note

that this is a non-injective morphism since graph nodes 2 and 3 have the same image.

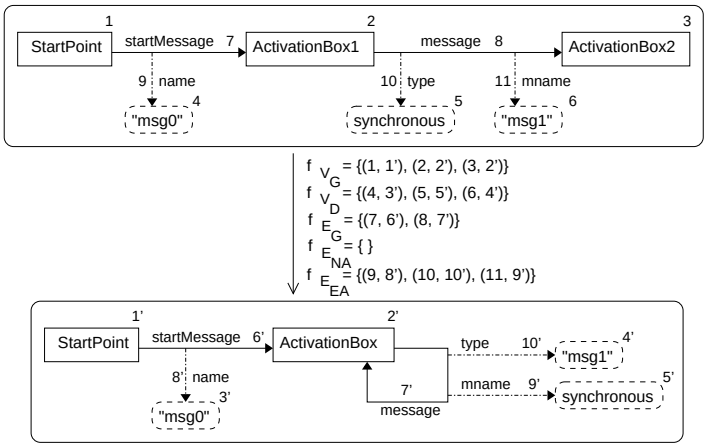


Figure 4: An E-graph morphism.

E-graphs and E-graph morphisms form a category, where the former are the objects and the latter the arrows. It is indeed a category, as the identity arrow is the identity morphism, and the composition of morphisms is associative.

Definition 3 *(Category* **EGraph***, taken from [Ehrig* et al.*, 2006])*
E-graphs together with E-graph morphisms form the category **EGraph***.*

Although E-graphs allow attributes in nodes and edges, we combine E-graphs together with an algebra over an appropriate signature in order to structure the attribute values (the elements of V_D). Having an algebra allows us to distinguish types (sorts like integer, string, and so forth) for attribution, as well as operations. Therefore, we assume that an E-graph has an associated data signature $DSIG$, which contains the appropriate declaration of sorts and operations. Some of the declared sorts will be used for attribution, and the carrier sets of the attribution sorts must be exactly the elements of V_D. Please note that it may be possible to have an infinite number of elements in V_D.

Definition 4 *(Attributed Graph, taken from [Ehrig* et al.*, 2006])*
Given a data signature $DSIG = (S_D, OP_D)$ which contains sorts for attribution $S'_D \subseteq S_D$, an attributed graph $AG = (G, D)$ consists of an E-graph G and a $DSIG-algebra$ D with $\biguplus_{s \in S'_D} D_s = V_D$.

Figure 5 shows an example of attributed graph $AG = (G, D)$ representing a sequence diagram. It uses a data signature $DSIG$, which is defined as:

```
DSIG = Char+String+
sorts :  MessageType
```

```
opns :
synchronous:  → MessageType
asynchronous:  → MessageType
destroy:  → MessageType
```

That is, sort $MessageType$ declares three constants *synchronous*, *asynchronous* and *destroy*. The data sorts used for attribution are $S'_D = \{String, MessageType\}$; $Char$ is an auxiliary type.

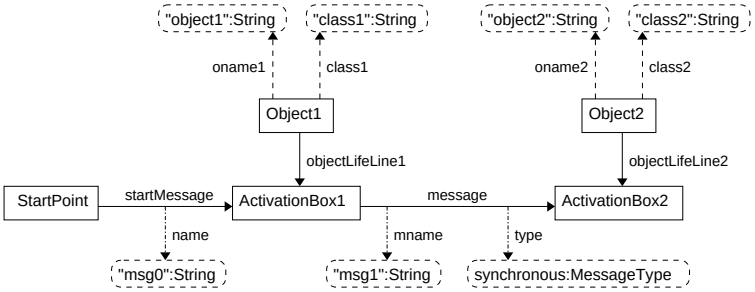


Figure 5: An attributed graph.

We define attributed graph morphisms between two attributed graphs as a tuple of two mappings. The first one is an E-graph morphism, the second one is an algebra homomorphism.

Definition 5 *(Attributed Graph morphism, taken from [Ehrig* et al.*, 2006])*
Given two attributed graphs $AG^i = (G^i, D^i)$ with $i = 1, 2$, an attributed graph morphism $f\colon AG^1 \to AG^2$ is a pair $f = (f_G, f_D)$ where $f_G\colon G^1 \to G^2$ is an E-graph morphism and $f_D\colon D^1 \to D^2$ is an algebra homomorphism, such that the diagram in Figure 6 commutes for all $s \in S'_D$.

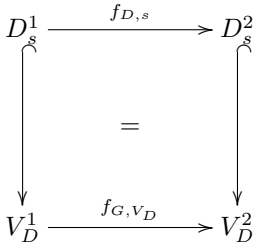


Figure 6: Condition for attributed graph morphisms.

Note how, in Figure 6, the inclusion of D^i_s in V^i_D is given by the definition of attributed graph. Attributed graphs and attributed morphisms form a category, where the former are the objects and the latter the arrows. As before, it is indeed a category as the identity arrow is the identity attributed morphism, and the composition of attributed morphisms is associative.

Definition 6 *(Category* **AGraph***, taken from [Ehrig* et al.*, 2006])*
Given a data signature DSIG as above, attributed graphs together with attributed graph morphisms form the category **AGraph***.*

For the typing of attributed graphs we use the concept of type graph [Corradini *et al.*, 1996]. This can be modelled as a distinguished attributed graph ATG, which is attributed over the final $DSIG-$algebra Z. That is, the carrier sets of each sort in Z have a unique element, which is the type name. Note how, the concept of type graph is similar to the one of meta-model [Atkinson and Kühne, 2002], but the latter includes inheritance, multiplicities and other constraints, possibly using a constraint language. In section 7 we extend (triple) type graphs with inheritance of nodes and edges.

Definition 7 *(Attributed Type Graph, taken from [Ehrig* et al.*, 2006])*
An attributed type graph is an attributed graph $ATG = (TG, Z)$, where Z is the final $DSIG-$algebra with carrier sets $Z_s = \{s\}\ \forall s \in S_D$.

Figure 7 shows an example of attributed type graph $ATG = (TG, Z)$ for the definition of UML sequence diagrams, where Z is the final algebra for the signature $DSIG = Char + String + MessageType$ used in the example corresponding to Figure 5. This type graph models the concrete syntax of sequence diagrams, that is, it is not equal to the UML 1.5 meta-model since this latter models the abstract syntax, but not the concrete one. Basically, the type graph declares objects that can be linked to activation boxes. Activation boxes can be linked through life lines and through messages. They can also be linked to objects through create messages. Finally, the sequence diagram should have a start point with a start message.

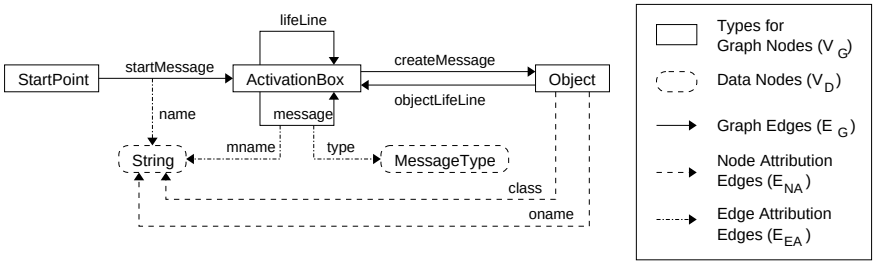


Figure 7: Attributed type graph for Sequence Diagrams.

Now, we define graphs conformant to a type graph. This notion is similar to the relation between a model and its meta-model. We say that a graph is conformant to a type graph (or is an instance of it) if there is an attributed morphism from the graph to the type graph. In fact, from now on, we will work with tuples where the first element is the graph itself and the second one the typing morphism. This indeed can be formalized as a slice category.

Definition 8 *(Attributed Typed Graph, taken from [Ehrig* et al.*, 2006])*

An attributed typed graph over ATG is an object $TAG = (AG, t)$ *in the slice category* **AGraph**/**ATG**, *where* $AG = (G, D)$ *is an attributed graph and* $t\colon AG \to ATG$ *is an attributed graph morphism called the typing of* AG.

Now, we define morphisms between attributed typed graphs. These are attributed morphisms with the restriction that the typing should be preserved from the source to the target graph.

Definition 9 *(Attributed Typed Graph morphism, taken from [Ehrig* et al.*, 2006])*

Given two attributed typed graphs $TAG^i = (AG^i, t^i)$ *over* ATG, *an attributed typed graph morphism* $f\colon (AG^1, t^1) \to (AG^2, t^2)$ *is an attributed graph morphism* $f\colon AG^1 \to AG^2$ *such that* $t^2 \circ f = t^1$ *as Figure 8 shows.*

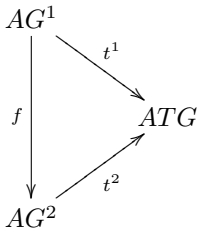


Figure 8: Condition for attributed typed graph morphisms.

Figure 9 shows an attributed typed graph (AG, t). Nodes and edges are labelled with its type (in the usual UML notation for instances). As stated before, attributed instance graphs can be infinite, since data set *String* for attribution is infinite and all the values have to be part of the graph. Otherwise, attribute computation is not possible. In the figure, we only show those data elements used for attribution.

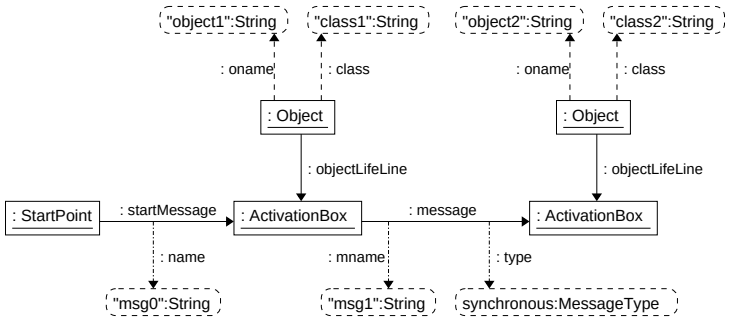


Figure 9: Attributed typed graph, with respect to the attributed type graph in Figure 7.

Attributed typed graphs and attributed typed morphisms form a category, where the former are the objects and the latter the arrows. As before, it is indeed

a category as the identity arrow is the identity attributed typed morphism, and the composition of attributed typed morphisms is associative. Moreover, it is built using a slice category construction.

Definition 10 *(Category* $\mathbf{AGraph_{ATG}}$*, taken from [Ehrig* et al.*, 2006])*
Attributed typed graphs over an attributed type graph ATG, together with attributed typed graph morphisms, form the category $\mathbf{AGraph_{ATG}}$*.*

3 Attributed Typed Triple Graphs

In this section, we use the previous concepts in order to formalize triple graphs. We start by defining the notion of TriE-graph, which is made of three E-graphs and two correspondence functions c_1 and c_2. One of the three graphs is called correspondence graph. The correspondence functions are defined from the nodes in the correspondence graph to the nodes and edges in the other two graphs. In addition the functions can be undefined, what is modelled with a special element in the codomain (named "$\cdot$"). Therefore, we have extended the actual notion of triple graphs [Schürr, 1994] in several ways. On the one hand, we use a definition that contemplates attributes in nodes and edges. On the other hand, our correspondence functions are more flexible as the codomain includes nodes, edges, and the special element for modelling that the function is undefined.

Being able to relate links with both nodes and edges is crucial in our approach. Suppose we have two attributed edges between the same source and target nodes in the source graph, that we want to relate with other edges (also with attributes) in the target graph. Relating only the source and target nodes is not enough, as then we do not know which edge in the source graph is related to which one in the target graph. Therefore it is necessary to be able to directly map edges. This situation is depicted in Figure 10. To the left it is shown how mapping the source and target nodes is not enough, as we don't know whether the edge with attribute *ValueA* is mapped to the edge with value 1 or 2. The situation is solved in the triple graph to the right by mapping the edges.

On the other hand, as in [Guerra and de Lara, 2004], the source graph may represent the concrete syntax graph, and the target graph the abstract syntax. In this casen the user interacts with the concrete graph, and he may delete elements which are already related to elements in the abstract graph. When such operation is performed, the mapping of the correspondence node becomes undefined. Keeping the correspondence node with just one mapping is useful as we may later want to delete the element in the target graph, and probably some others related to it. Moreover, being able to know that a mapping is undefined is also a very useful negative test in triple graph grammar rules.

Definition 11 *(TriE-graph)*
A TriE-graph $TriG = (G_1, G_2, G_C, c_1, c_2)$ *is made of three E-graphs* $G_i = (V_{G_i}, V_{D_i}, E_{G_i}, E_{NA_i}, E_{EA_i}, (source_{j_i}, target_{j_i})_{j \in \{G,NA,EA\}})$ *for* $i \in \{1, 2, C\}$*, with* $V_{D_1} = V_{D_2} = V_{D_C}$*, and two functions* $c_j \colon V_{G_C} \to V_{G_j} \cup E_{G_j} \cup \{\cdot\}$ *(for* $j = 1, 2$*).*

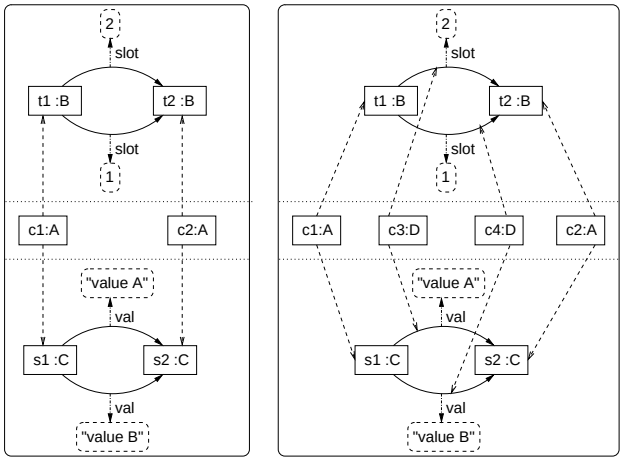


Figure 10: Insufficient Mapping of Nodes(left). Mapping of Edges (right)

Graph G_1 is called source, G_2 is called target and G_C is called correspondence. Functions c_1 and c_2 are called source and target correspondence functions respectively. We use the auxiliary sets $edges_i = \{x \in V_{G_C} | c_i(x) \in E_{G_i}\}$, $nodes_i = \{x \in V_{G_C} | c_i(x) \in V_{G_i}\}$ and $undef_i = \{x \in V_{G_C} | c_i(x) = \cdot\}$ for $i = 1, 2$. The latter set is used to denote that the correspondence function c_i for an element x is undefined. The previous two sets are used to denote that the codomain of the correspondence function c_i for an element x are edges or nodes, respectively.

Morphisms c_1 and c_2 represent m-to-n relationships between nodes and edges in G_1 and G_2 via G_C in the following way: $x \in V_{G_1} \cup E_{G_1}$ is related to $y \in V_{G_2} \cup E_{G_2} \iff \exists z \in V_{G_C} \,|\, x = c_1(z)\, and\, y = c_2(z)$. Note also that all the data sets V_{D_i} are assumed to be equal. We could have used just one set for each one of the three graphs, but having independent E-graphs makes it possible to reuse some of the concepts developed in [Ehrig *et al.*, 2004b] for E-graphs.

Figure 12 shows a TriE-graph for the definition of the abstract and concrete syntax of UML sequence diagrams. The target graph G_2 in the upper part corresponds to the abstract syntax, the source graph G_1 in the lower part corresponds to the concrete syntax, and finally, the correspondence graph G_C in the middle contains elements relating G_1 and G_2 by means of the correspondence functions. Note that, although $V_{D_1} = V_{D_2} = V_{D_C}$, for clarity, we have represented the elements of these sets in each one of the E-graphs. However, the elements of the three sets are the same. Therefore, node "$class1''$ in V_{D_2} is the same as node "$class1''$ in V_{D_1}.

Now, we define mappings between two TriE-graphs. These are made of three E-graph morphisms plus additional constraints regarding the preservation of the correspondence functions.

Definition 12 *(TriE-graph morphism)*

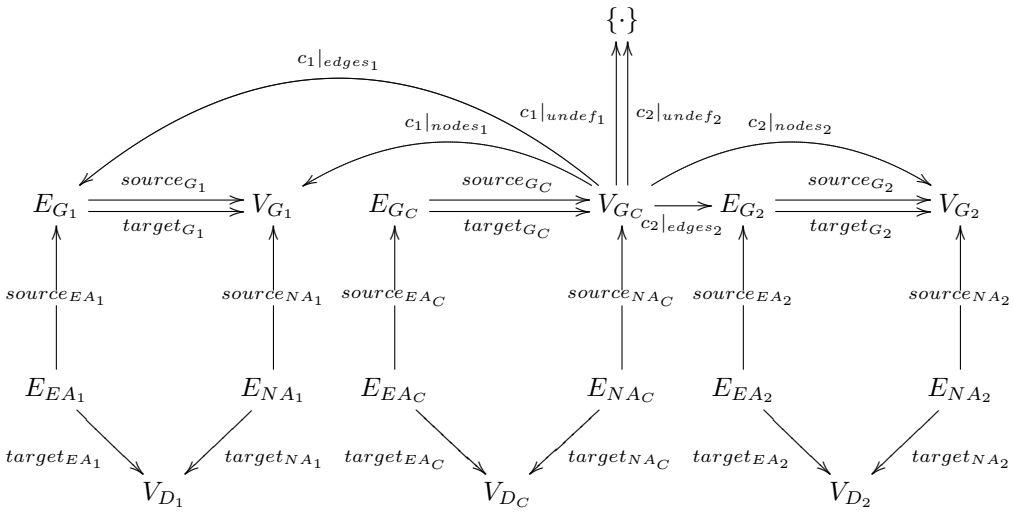


Figure 11: A TriE-graph.

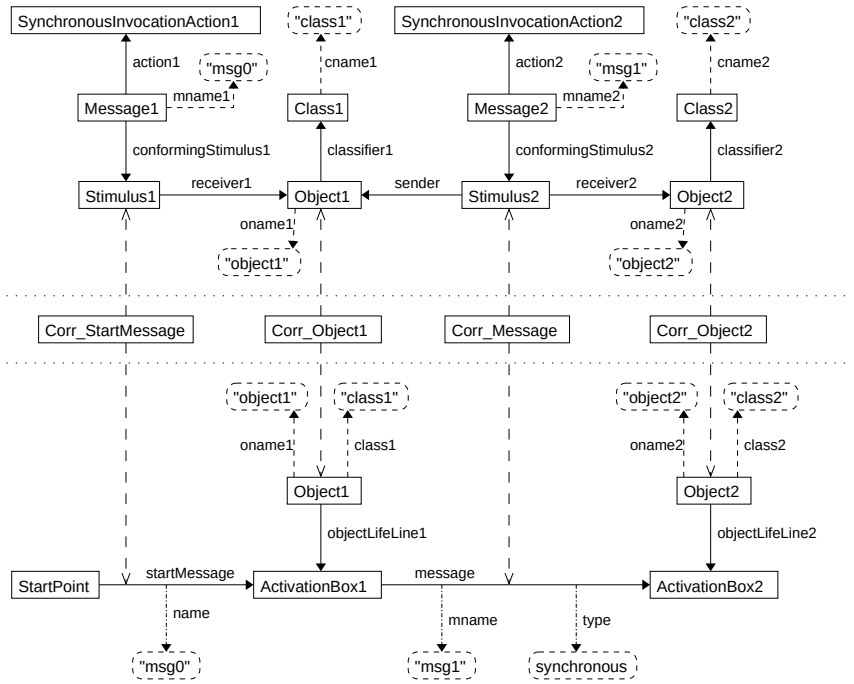


Figure 12: TriE-graph for the concrete and abstract syntax of Sequence Diagrams.

Given two TriE-graphs $TriG^i = (G_1^i, G_2^i, G_C^i, c_1^i, c_2^i)$ with $i = 1, 2$, a TriE-graph morphism $f\colon TriG^1 \to TriG^2$ is a tuple $f = (f^1, f^2, f^c)$ made of three E-graph morphisms $f^i\colon G_i^1 \to G_i^2$ ($i \in \{1, 2, C\}$) such that:

- $f^i_{V_{G_i}} \circ c_i^1|_{nodes_i^1} = c_i^2 \circ f^C_{V_{G_C}}|_{nodes_i^1}$ *for* $i = 1, 2$.
- $f^i_{E_{G_i}} \circ c_i^1|_{edges_i^1} = c_i^2 \circ f^C_{V_{G_C}}|_{edges_i^1}$ *for* $i = 1, 2$.
- $c_i^1|_{undef_i^1} = c_i^2 \circ f^C_{V_{G_C}}|_{undef_i^1}$ *for* $i = 1, 2$.

as shown in Figure 13.

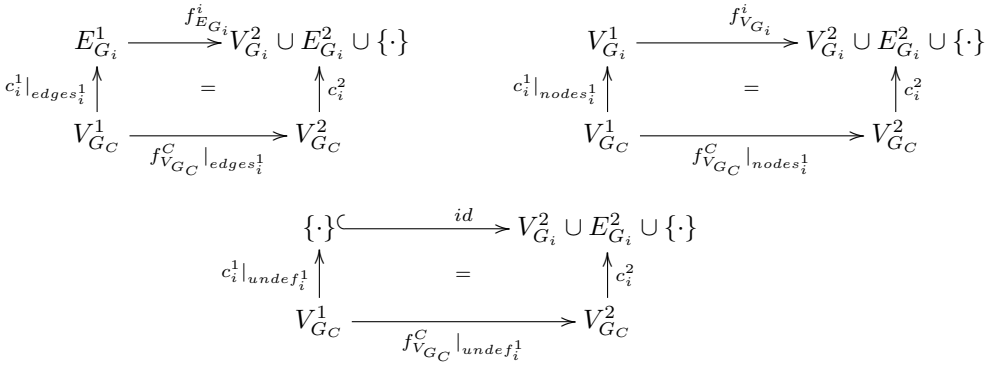


Figure 13: Conditions for TriE-graph morphisms.

Remark: note that f^1, f^2 and f^C are E-graph morphisms and thus put additional constraints (not explicitly shown in Figure 13) regarding the preservation of the three E-graph structures making each TriE-graph.

TriE-graphs and TriE-graph morphisms form a category, where the former are the objects and the latter the arrows. As before, it is indeed a category as the identity arrow is the identity TriE-graph morphism, and the composition of TriE-graph morphisms is associative.

Definition 13 *(Category* **TriEGraph***)*
TriE-graphs together with TriE-graph morphisms form the category **TriE-Graph**.

Proof
Category **TriEGraph** is made of the class of all TriE-graphs together with the class $\biguplus_{(A,B)\in TriEGraph\times TriEGraph}[A, B]_{TriEGraph}$ of all TriE-graph morphisms, where $[A, B]_{TriEGraph}$ is the set of all TriE-graph morphisms from A to B. In order to demonstrate that **TriEGraph** is a category, we have to check: (i) that TriE-graph morphisms can be composed, (ii) that they are associative and (iii) the existence of identity morphisms.

(i) Composition of **TriEGraph** morphisms. Given two TriE-graph morphisms $f\colon TriG^1 \to TriG^2$ and $g\colon TriG^2 \to TriG^3$, the composition $g \circ f = (g^1 \circ f^1, g^2 \circ f^2, g^c \circ f^c)\colon TriG^1 \to TriG^3$ can be defined by composing the three E-graph morphisms in f and g. As the following three formulae show, the resulting morphism h fulfills the three conditions of definition 12:

- $(g^i_{V_{G_i}} \circ f^i_{V_{G_i}}) \circ (c^1_i|_{nodes^1_i}) = c^3_i \circ g^C_{V_{G_C}}|_{nodes^2_i} \circ f^C_{V_{G_C}}|_{nodes^1_i} = c^3_i \circ ((g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{nodes^1_i})$ for $i = 1, 2$. See Figure 14. Note that equality $g^C_{V_{G_C}}|_{nodes^2_i} \circ f^C_{V_{G_C}}|_{nodes^1_i} = (g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{nodes^1_i}$ holds because TriE-Graph morphisms require that $f^C_{V_{G_C}}|_{nodes^1_i}(V^1_{GC})$ be a subset of $nodes^2_i$ (see condition 1 of definition 12). The commutativity of the outer square follows from the commutativity of the two inner squares.

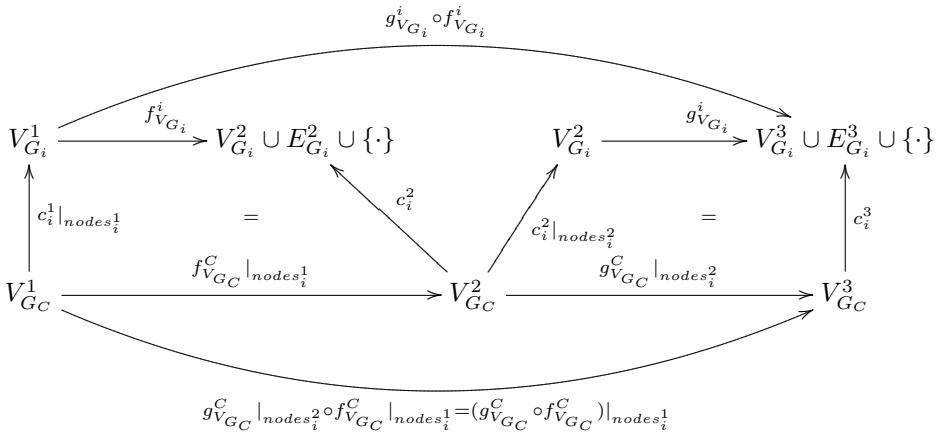


Figure 14: Composition of TriE-graph morphisms: (a) condition for correspondence nodes mapped to nodes

- $(g^i_{E_{G_i}} \circ f^i_{E_{G_i}}) \circ (c^1_i|_{edges^1_i}) = c^3_i \circ g^C_{V_{G_C}}|_{edges^2_i} \circ f^C_{V_{G_C}}|_{edges^1_i} = c^3_i \circ ((g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{edges^1_i})$ for $i = 1, 2$. See Figure 15. Note that equality $g^C_{V_{G_C}}|_{edges^2_i} \circ f^C_{V_{G_C}}|_{edges^1_i} = (g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{edges^1_i}$ holds because TriE-Graph morphisms require that $f^C_{V_{G_C}}|_{edges^1_i}(V^1_{GC})$ be a subset of $edges^2_i$ (see condition 2 of definition 12). The commutativity of the outer square follows from the commutativity of the two inner squares.

- $c^1_i|_{undef^1_i} = c^3_i \circ (g^C_{V_{G_C}}|_{undef^2_i} \circ f^C_{V_{G_C}}|_{undef^1_i}) = c^3_i \circ ((g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{undef^1_i})$ for $i = 1, 2$. See Figure 16. Note that equality $g^C_{V_{G_C}}|_{undef^2_i} \circ f^C_{V_{G_C}}|_{undef^1_i} = (g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{undef^1_i}$ holds because TriE-Graph morphisms require that $f^C_{V_{G_C}}|_{undef^1_i}(V^1_{GC})$ be a subset of $undef^2_i$ (see condition 3 of definition 12). The commutativity of the outer square follows from the commutativity of the two inner squares.

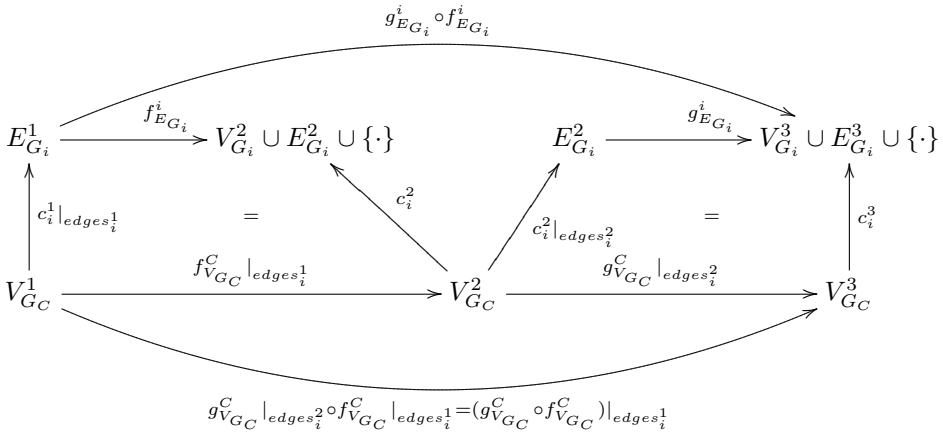


Figure 15: Composition of TriE-graph morphisms: (b) condition for correspondence nodes mapped to edges

(ii) Associativity of **TriEGraph** morphisms. Given three TriE-graph morphisms $f = (f^1, f^2, f^C)\colon G^1 \to G^2$, $g = (g^1, g^2, g^C)\colon G^2 \to G^3$ and $h = (h^1, h^2, h^C)\colon G^3 \to G^4$, we have to demonstrate that $(h \circ g) \circ f = h \circ (g \circ f)$. This easily follows from the associativity of the three E-graph morphisms that make each TriE-graph morphism. Figure 17 shows the case for correspondence nodes mapped to nodes. It can be seen how the associativity of the TriE-graph morphisms reduces to the associativity of the f^i and f^C E-graph morphisms, which are associative because E-graphs form a category. The case for correspondence nodes mapped to edges and with undefined morphisms is similar to the one of Figure 17. Formally we have that:

- $((h^i_{V_{G_i}} \circ g^i_{V_{G_i}}) \circ f^i_{V_{G_i}}) \circ c^1_i|_{nodes^1_i} = (h^i_{V_{G_i}} \circ (g^i_{V_{G_i}} \circ f^i_{V_{G_i}})) \circ c^1_i|_{nodes^1_i} = c^4_i \circ ((h^C_{V_{G_C}} \circ (g^C_{V_{G_C}} \circ f^C_{V_{G_C}}))|_{nodes^1_i}) = c^4_i \circ (((h^C_{V_{G_C}} \circ g^C_{V_{G_C}}) \circ f^C_{V_{G_C}})|_{nodes^1_i})$, because each single morphism is an E-graph morphism and they are associative.
- $((h^i_{E_{G_i}} \circ g^i_{E_{G_i}}) \circ f^i_{E_{G_i}}) \circ c^1_i|_{edges^1_i} = (h^i_{E_{G_i}} \circ (g^i_{E_{G_i}} \circ f^i_{E_{G_i}})) \circ c^1_i|_{edges^1_i} = c^4_i \circ ((h^C_{V_{G_C}} \circ (g^C_{V_{G_C}} \circ f^C_{V_{G_C}}))|_{edges^1_i}) = c^4_i \circ (((h^C_{V_{G_C}} \circ g^C_{V_{G_C}}) \circ f^C_{V_{G_C}})|_{edges^1_i})$, because each single morphism is an E-graph morphism and they are associative.
- $c^1_i|_{undef^1_i} = c^4_i \circ (((h^C_{V_{G_C}} \circ g^C_{V_{G_C}}) \circ f^C_{V_{G_C}})|_{undef^1_i}) = c^4_i \circ ((h^C_{V_{G_C}} \circ (g^C_{V_{G_C}} \circ f^C_{V_{G_C}})|_{undef^1_i})$, because each single morphism is an E-graph morphism and they are associative.

Therefore we have that $(h \circ g) \circ f = h \circ (g \circ f)$.

(iii) Identities in **TriEGraph**. We have to show that for each TriE-graph G, there exists the identity morphism $id_G\colon G \to G$, such that for all TriE-graphs

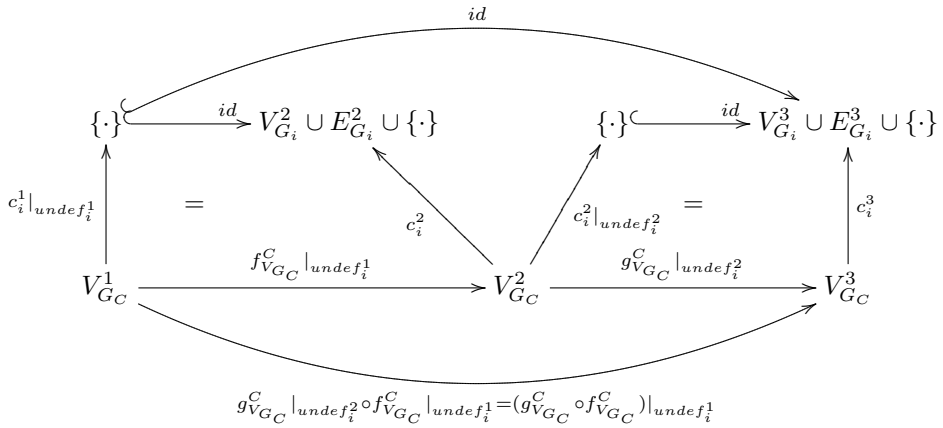


Figure 16: Composition of TriE-graph morphisms. (iii) condition for correspondence nodes with undefined mapping

G^1, G^2 and morphisms $f\colon G^1 \to G^2$, it holds that $f \circ id_{G^1} = f$ and $id_{G^2} \circ f = f$.

Given a TriE-graph G, the identity morphism $id_G = (id^1_G, id^2_G, id^C_G)$ is built by taking the identity morphisms between each graph making the triple graph. Given an arbitrary TriE-graph morphism $f = (f^1, f^2, f^C)\colon G \to H$, we have that:

- Using E-graph identities, we have that $f^i \circ id^i_G = f^i$, for $i \in \{1, 2, C\}$. Therefore $(f^1, f^2, f^C) \circ (id^1_G, id^2_G, id^C_G) = (f^1 \circ id^1_G, f^2 \circ id^2_G, f^C \circ id^C_G) = (f^1, f^2, f^C)$ and thus $f \circ id_G = f$.
- Using E-graph identities again, we have that $id^i_H \circ f^i = f^i$, for $i \in \{1, 2, C\}$. Therefore $(id^1_H, id^2_H, id^C_H) \circ (f^1, f^2, f^C) = (id^1_H \circ f^1, id^2_H \circ f^2, id^C_H \circ f^C) = (f^1, f^2, f^C)$ and thus $id_H \circ f = f$.

□

As in the case of E-graphs, we provide TriE-graphs with an algebra, in order to structure the attribution set and to provide appropriate operations for attribute computation. We assign one algebra for the whole TriE-graph, in such a way that the attribute sets V_{D_i} contain the disjoint union of the algebra carrier sets.

Definition 14 *(Attributed Triple Graph)*
Given a data signature $DSIG = (S_D, OP_D)$ with attribute value sorts $S'_D \subseteq S_D$, an attributed triple graph $TriAG = (TriG, D)$ consists of a TriE-graph $TriG = (G_1, G_2, G_C, c_1, c_2)$ and one algebra D of the given $DSIG$ signature with $\biguplus_{s \in S'_D} D_s = V_{D_i}$ for $i \in \{1, 2, C\}$.

Note that $V_{D_1} = V_{D_2} = V_{D_C}$ and that $AG_i = (G_i, D)$ for $i \in \{1, 2, C\}$ are three attributed graphs. We could have used three different algebras for

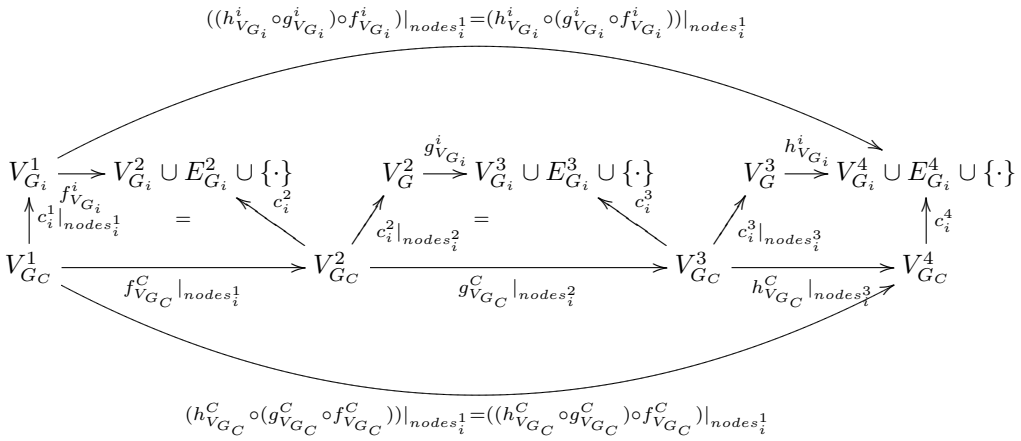


Figure 17: Associativity of TriE-graph morphisms: (a) condition for correspondence nodes mapped to nodes

each E-graphs in the TriE-graph, however, using a unique algebra simplifies the theory.

Now, we define mappings between two attributed triple graphs. These are made of a TriE-graph morphism and one algebra homomorphism.

Definition 15 *(Attributed Triple Graph Morphism)*

Given two attributed triple graphs $TriAG^i = (TriG^i, D^i)$ with $i = 1, 2$, an attributed triple graph morphism $f\colon TriAG^1 \to TriAG^2$ (short ATT-morphism) is a tuple $f = (f_{TriG}, f_D)$ where $f_{TriG}\colon TriG^1 \to TriG^2$ is a TriE-graph morphism and $f_D\colon D^1 \to D^2$ is an algebra-homomorphism such that the diagram in Figure 18 commutes for all $s \in S'_D$.

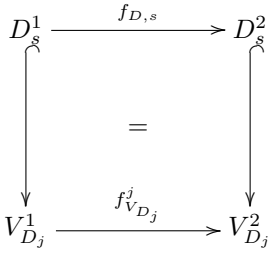


Figure 18: Condition for attributed triple graph morphisms.

Attributed triple graphs and attributed triple graph morphisms form a category, where the former are the objects and the latter the arrows. As before, it is indeed a category as the identity arrow is the identity attributed triple graph morphism, and the composition of this kind of morphisms is associative.

Definition 16 *(Category* **TriAGraph***)*
Attributed triple graphs together with attributed triple graph morphisms form the category **TriAGraph**.

Proof
Category **TriAGraph** is made of the class of all TriAGraphs together with the class $\biguplus_{(A,B)\in TriAGraph\times TriAGraph}[A,B]_{TriAGraph}$ of all TriAGraph morphisms, where $[A,B]_{TriAGraph}$ is the set of all ATT-morphisms from A to B. In order to demonstrate that *TriAGraph* is a category, we have to check: (i) that ATT-morphisms can be composed, (ii) that they are associative and (iii) the existence of identity morphisms.

(i) Composition of **TriAGraph** morphisms. Given two ATT-morphisms $f\colon TriAG^1 \to TriAG^2$ and $g\colon TriAG^2 \to TriAG^3$, the composition $g \circ f = (g_{TriG} \circ f_{TriG} = (g^1 \circ f^1, g^2 \circ f^2, g^c \circ f^c), g_D \circ f_D)\colon TriAG^1 \to TriAG^3$ can be defined by composing the TriE-graph morphisms and the algebra homomorphisms in f and g. That is, $g_{TriG} \circ f_{TriG}$ is a TriE-graph morphism, as this kind of morphisms are closed under composition. On the other hand $g_D \circ f_D\colon D^1 \to D^3$ is again an algebra homomorphism, which fullfils the diagram shown in Figure 19, where the commutativity of both squares ensures the commutativity of the outer square. Therefore $g \circ f$ is an ATT-morphism.

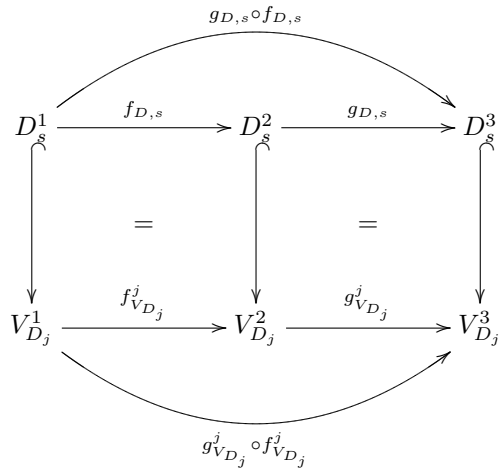


Figure 19: Composition of TriAGraph morphisms. Condition for the algebra homomorphisms.

(ii) Associativity of **TriAGraph** morphisms. Given ATT-morphisms $f = (f_{TriG} = (f^1, f^2, f^C), f_D)\colon G^1 \to G^2$, $g = (g_{TriG} = (g^1, g^2, g^C), g_D)\colon G^2 \to G^3$ and $h = (h_{TriG} = (h^1, h^2, h^C), h_D)\colon G^3 \to G^4$, we have to demonstrate that $(h \circ g) \circ f = h \circ (g \circ f)$. This easily follows from the associativity of the TriE-graph morphisms and the algebra homomorphisms that make each TriAGraph morphism. Formally we have that:

$(h \circ g) \circ f = ((h_{TriG}, h_D) \circ (g_{TriG}, g_D)) \circ (f_{TriG}, f_D) =$
$(h_{TriG} \circ g_{TriG}, h_D \circ g_D) \circ (f_{TriG}, f_D) =$
$(h_{TriG} \circ g_{TriG} \circ f_{TriG}, h_D \circ g_D \circ f_D) =$
$(h_{TriG} \circ (g_{TriG} \circ f_{TriG}), h_D \circ (g_D \circ f_D)) =$
$(h_{TriG}, h_D) \circ (g_{TriG} \circ f_{TriG}, g_D \circ f_D) =$
$(h_{TriG}, h_D) \circ ((g_{TriG}, g_D) \circ (f_{TriG}, f_D)) =$
$h \circ (g \circ f)$.

Therefore we have that $(h \circ g) \circ f = h \circ (g \circ f)$.

(iii) Identities in **TriAGraph** morphisms. We have to show that for each TriAGraph G, there exists the identity morphism $id_G \colon G \to G$, such that for all TriAGraphs G^1, G^2 and morphisms $f \colon G^1 \to G^2$, it holds that $f \circ id_{G^1} = f$ and $id_{G_2} \circ f = f$.

Given a TriAGraph G, the identity morphism $id_G = (id_G^{TriG} = (id_G^1, id_G^2, id_G^C), id_D)$ is built by taking the TriE-graph identity morphism and the algebra identity homomorphism. Given an arbitrary TriAGraph morphism $f = (f_{TriG} = (f^1, f^2, f^3), f_D) \colon G \to H$, we have that:

- $id_G \circ f = (id_G^{TriG}, id_D) \circ (f_{TriG}, f_D) = (id_G^{TriG} \circ f_{TriG}, id_D \circ f_D) = (f_{TriG}, f_D) = f$.
- $f \circ id_H = (f_{TriG}, f_D) \circ (id_H^{TriG}, id_H) = (f_{TriG} \circ id_H^{TriG}, f_D \circ id_H) = (f_{TriG}, f_D) = f$.

□

Now, we provide attributed triple graphs a typing. This is modelled by a distinguished attributed triple graph called attributed type triple graph. This graph is attributed on the final algebra of the signature.

Definition 17 *(Attributed Type Triple Graph)*
An attributed type triple graph is an attributed triple graph $TriATG = (TriTG, Z)$, where Z is the final algebra of the $DSIG$ signature with carrier sets $Z_s = \{s\}$ $\forall s \in S_D$.

Figure 20 shows an attributed type triple graph $TriATG = (TriTG, Z)$ for the definition of both abstract and concrete syntax of UML sequence diagrams. The data signature is $DSIG = Char + String + MessageType$ (see example corresponding to Figure 7 for a detailed description of the data signature $MessageType$). The target graph in the upper part of the triple graph corresponds to the abstract syntax. The source graph in the lower part corresponds to the concrete syntax, while the correspondence graph in the middle relates concepts of both sides.

Now, we define the relationship between attributed triple graphs and attributed type triple graphs. This is done in a similar way as in the previous section. That is, we consider tuples where the first element is the attributed triple graph, while the second contains the typing morphism from the attributed triple graph to the attributed type triple graph. Again, it can be formalized as a slice category.

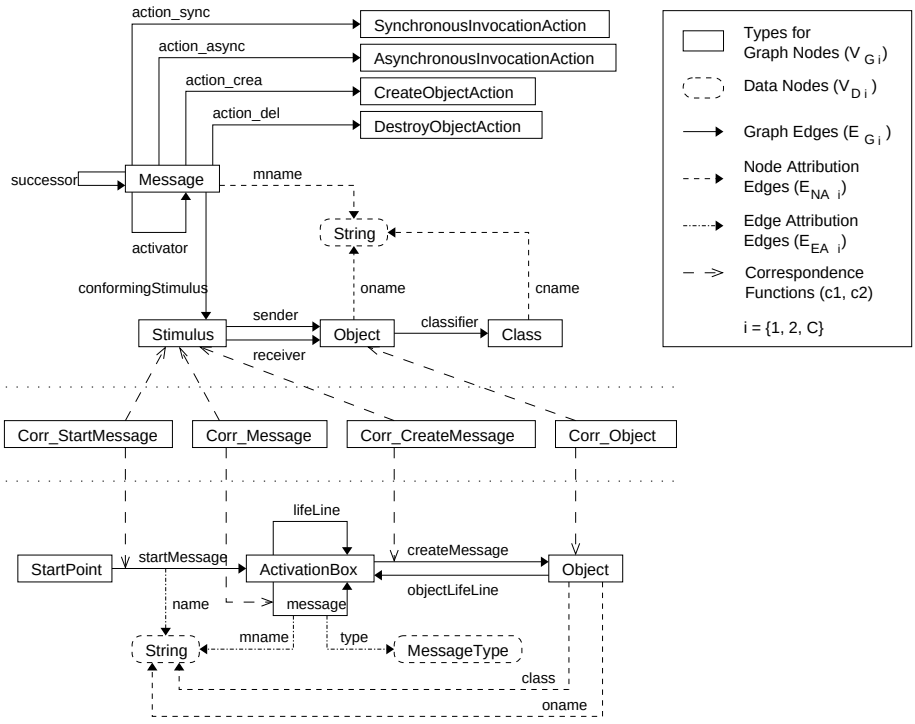


Figure 20: Attributed type triple graph for the concrete and abstract syntax of Sequence Diagrams.

Definition 18 *(Attributed Typed Triple Graph)*
An attributed typed triple graph (short ATT-graph) over $TriATG$ is an object $TriTAG = (TriAG, t)$ in the slice category **TriAGraph/TriATG***, where $TriAG = (TriG, D)$ is an attributed triple graph and $t\colon TriAG \to TriATG$ is an attributed triple graph morphism called the typing of $TriAG$.*

Mappings between attributed typed triple graphs are like mappings between attributed triple graphs, but in addition, the typing has to be preserved.

Definition 19 *(Attributed Typed Triple Graph morphism)*
Given two ATT-graphs $TriTAG^i = (TriAG^i, t^i)$ over an attributed type triple graph $TriATG$, an attributed typed triple graph morphism (short ATT-morphism) $f\colon (TriAG^1, t^1) \to (TriAG^2, t^2)$ is an attributed triple graph morphism $f\colon TriAG^1 \to TriAG^2$ such that $t^2 \circ f = t^1$, as Figure 21 shows.

Figure 22 shows an ATT-graph over the attributed type graph in Figure 20. Nodes and edges are labelled with its type (in the usual UML notation for instances).

ATT-graphs and ATT-morphisms form a category, where the former are the objects and the latter the arrows. As before, it is indeed a category as

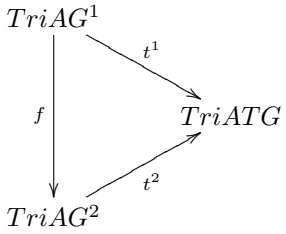


Figure 21: Condition for attributed typed triple graph morphisms.

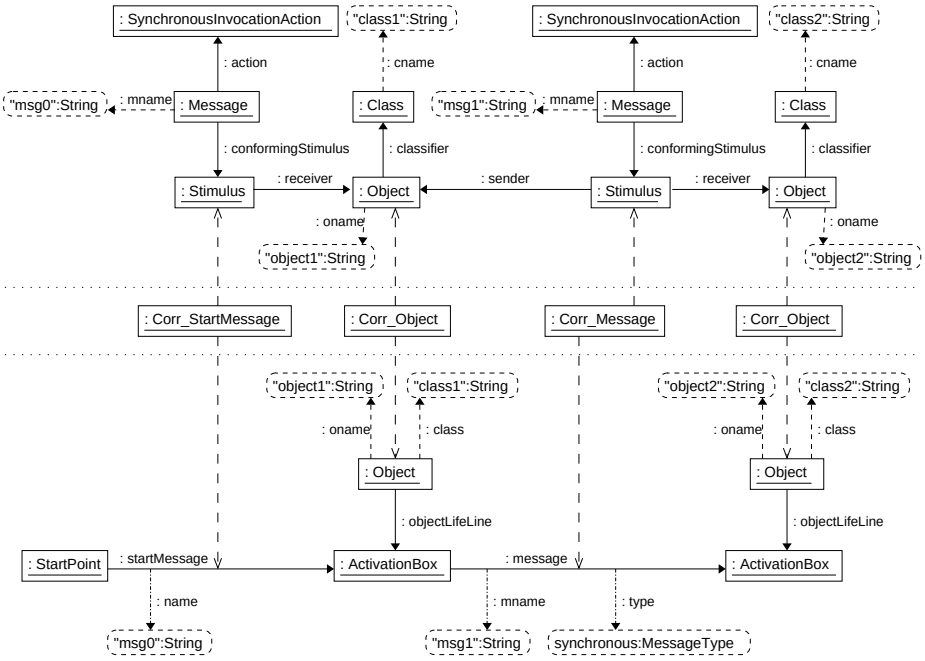


Figure 22: Attributed typed triple graph, with respect to the attributed type triple graph in Figure 20.

the identity arrow is the identity attributed typed triple morphism, and the composition of this kind of morphisms is associative.

Definition 20 *(Category $TriAGraph_{TriATG}$)*
Attributed typed triple graphs over an attributed type triple graph $TriATG$, together with attributed typed triple graph morphisms, form the category $\mathbf{TriAGraph_{TriATG}}$.

Proof
Follows from the fact that $\mathbf{TriAGraph_{TriATG}}$ is a slice category.

□

4 Pushouts and Pullbacks for Attributed Typed Triple Graphs

In this section, we show how pushouts and pullbacks in category $\mathbf{TriAGraph_{TriATG}}$ are constructed (we closely follow [Ehrig *et al.*, 2006]). Pushouts model the gluing of two objects through some common elements, and are needed for the construction of typed attributed graph transformations. In fact, we only need pushouts along a special class $\mathcal{M}$ of monomorphisms, which are injective in the graph part and isomorphisms in the data part. The reason for this, is that pushouts are constructed for modelling a rule application, and rules are represented in the DPO approach as: $p = (L \xleftarrow{l} K \xrightarrow{r} R)$, with l and r injective (that is, belonging to class $\mathcal{M}$ of monomorphisms). Therefore, we only need pushouts along $\mathcal{M}$ morphisms. Of course, the match $m\colon L \to G$ can be non-injective. Note how, as TriE-graphs are indeed made of several sets, pushouts can be constructed componentwise by building the pushout of each set.

Pullbacks are the dual construction of pushouts and model the intersection of two objects (triple graphs in our case) in a larger context. Pullbacks in $\mathbf{TriAGraph_{TriATG}}$ are needed in the next section in order to show that this is an adhesive HLR category. In a similar way as pushouts, pullbacks can also be constructed componentwise.

We first start defining the class $\mathcal{M}$ of special morphisms (indeed monomorphisms) in categories **TriAGraph** and $\mathbf{TriAGraph_{TriATG}}$.

Definition 21 *(Class $\mathcal{M}$ of monomorphisms in* **TriAGraph***)*
A attributed triple morphism $f\colon TriAG^1 \to TriAG^2$ with $f = (f_{TriG} = (f^1, f^2, f^C), f_D)$ belongs to class $\mathcal{M}$ if f_{TriG} is an injective TriE-graph morphism (that is, injective in each component for each of the three graphs), and f_D is a $DSIG-$algebra isomorphism. This implies that $f^i_{V_D}$ is also bijective.

Definition 22 *(Class $\mathcal{M}$ of monomorphisms in* $\mathbf{TriAGraph_{TriATG}}$*)*
An ATT-morphism $f\colon (TriAG^1, t^1) \to (TriAG^2, t^2)$ belongs to class $\mathcal{M}$ if $f\colon TriAG^1 \to TriAG^2$ belongs to $\mathcal{M}$.

Classes $\mathcal{M}$ of monomorphisms in **TriAGraph** and $\mathbf{TriAGraph_{TriATG}}$ are closed under composition. Next, we show how to build pushouts in which one of the morphisms belongs to $\mathcal{M}$. This is basically done by building pushouts in category **Set** (see for example [Ehrig *et al.*, 2006]).

Figure 23 shows an example of pushout in category **Set**. Sets $A = \{a1, a2, a3\}$, $B = \{b1, b2\}$ and $C = \{c1, c2, c3, c4\}$ with morphisms $f : A \to B$ and $g : A \to C$ are given. The pushout PO is constructed as the quotient $B \dot\cup C|_{\equiv}$, where $\equiv$ is the smallest equivalence relation with $(f(a), g(a)) \in \equiv$ for all $a \in A$.

Fact 23 *(Pushouts along $\mathcal{M}$-morphisms in* **TriAGraph***)*
Given the attributed triple morphisms $f\colon TriAG^0 \to TriAG^1 \in \mathcal{M}$ and $g\colon TriAG^0 \to TriAG^2$, a pushout $(TriAG^3, f', g')$ in **TriAGraph***, with $TriAG^k = (TriG^k, D^k)$, $TriG^k = (G^k_1, G^k_2, G^k_C, c^k_1, c^k_2)$, and $G^k_i = (V^k_{G_i}, V^k_{D_i}, E^k_{G_i}, E^k_{NA_i}, E^k_{EA_i},$*

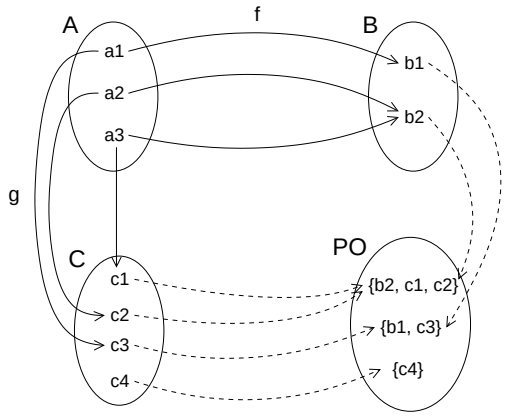


Figure 23: Example of pushout in category **Set**.

$(source^k_{j_i}, target^k_{j_i})_{j\in\{G,NA,EA\}})$ for $k \in \{0,1,2,3\}$, $i \in \{1,2,C\}$ can be constructed as follows (see Figure 24), where $f' \in \mathcal{M}$ and any other pushout $TriAG'$ is isomorphic to $TriAG^3$:

$$\begin{array}{ccc} TriAG^0 = (TriG^0, D^0) & \xrightarrow{f=((f^1,f^2,f^C),f_D)\in\mathcal{M}} & TriAG^1 = (TriG^1, D^1) \\ \Big\downarrow{\scriptstyle g=((g^1,g^2,g^C),g_D)} & = & \Big\downarrow{\scriptstyle g'=((g'^1,g'^2,g'^C),g'_D)} \\ TriAG^2 = (TriG^2, D^2) & \xrightarrow{f'=((f'^1,f'^2,f'^C),f'_D)\in\mathcal{M}} & TriAG^3 = (TriG^3, D^3) \end{array}$$

Figure 24: Construction of pushouts in **TriAGraph**.

1. $(V^3_{G_i}, f'^i_{V^3_{G_i}}, g'^i_{V^3_{G_i}})$ *is pushout of* $(V^0_{G_i}, f^i_{V^0_{G_i}}, g^i_{V^0_{G_i}})$ *in* **Sets** *for* $i \in \{1,2,C\}$.
2. $(E^3_{G_i}, f'^i_{E^3_{G_i}}, g'^i_{E^3_{G_i}})$ *is pushout of* $(E^0_{G_i}, f^i_{E^0_{G_i}}, g^i_{E^0_{G_i}})$ *in* **Sets** *for* $i \in \{1,2,C\}$.
3. $(E^3_{NA_i}, f'^i_{E^3_{NA_i}}, g'^i_{E^3_{NA_i}})$ *is pushout of* $(E^0_{NA_i}, f^i_{E^0_{NA_i}}, g^i_{E^0_{NA_i}})$ *in* **Sets** *for* $i \in \{1,2,C\}$.
4. $(E^3_{EA_i}, f'^i_{E^3_{EA_i}}, g'^i_{E^3_{EA_i}})$ *is pushout of* $(E^0_{EA_i}, f^i_{E^0_{EA_i}}, g^i_{E^0_{EA_i}})$ *in* **Sets** *for* $i \in \{1,2,C\}$.
5. $(V^3_{D_i}, f'^i_{V^3_{D_i}}, g'^i_{V^3_{D_i}}) = (V^2_{D_i}, id, g'^i_{V^2_{D_i}})$ *with* $g'^i_{V^2_{D_i}} = g^i_{V^2_{D_i}} \circ f^{-1,i}_{V^0_{D_i}} : V^1_{D_i} \to V^3_{D_i}$ *for* $i \in \{1,2,C\}$.
6. $(D^3, f'_D, g'_D) = (D^2, id, g'_D)$ *with* $g'_D = g_D \circ f_D^{-1} : D^1 \to D^3$.

7. *The* $source^3_{j_i}$ *and* $target^3_{j_i}$ *operations (for* $j \in \{G, NA, EA\}$*,* $i \in \{1, 2, C\}$*) are uniquely determined by the pushouts in (1)-(5).*

8. *The* c^3_1 *and* c^3_2 *functions are also uniquely determined by the pushouts in (1)-(4).*

Proof

The componentwise construction leads to a well-defined attributed triple graph $TriAG^3$ and attributed triple morphisms f' and g' with $f' \in \mathcal{M}$. The universal pushout property follows from the universal property of pushouts in **Sets**. Note also that $(V^3_{D_i}, f'^i_{V^3_{D_i}}, g'^i_{V^3_{D_i}})$ and (D^3, f'_D, g'_D) are pushouts in **Sets** resp. $\mathbf{DSIG} - \mathbf{Alg}$.

□

Figure 25 shows a simple example of a pushout in category **TriAGraph**. The example contains only nodes in V_G and E_G for simplicity. Note how $g \notin \mathcal{M}$ as both nodes 5 and 6 are mapped to node 9, and therefore nodes 11 and 12 are mapped into node 15. Consequently, function g' does not belong to $\mathcal{M}$.

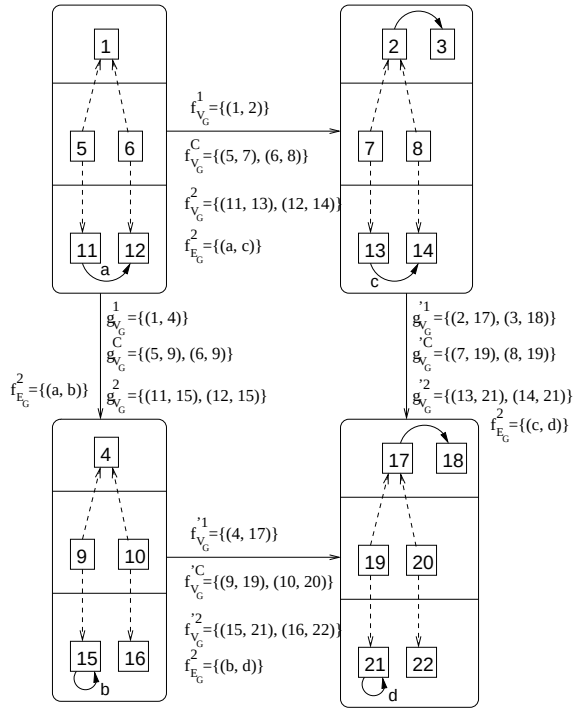


Figure 25: Example of pushout in category **TriAGraph**.

Fact 24 *(Pushouts along* $\mathcal{M}$*-morphisms in* $\mathbf{TriAGraph_{TriATG}}$*)*

Given ATT-morphisms $f\colon (TriAG^0, t^0) \to (TriAG^1, t^1) \in \mathcal{M}$ *and* $g\colon (TriAG^0, t^0) \to (TriAG^2, t^2)$*, the pushout* $((TriAG^3, t^3), f', g')$ *in* $\mathbf{TriAGraph_{TriATG}}$ *can be*

constructed as the pushout $(TriAG^3, f', g')$ *of* $f\colon TriAG^0 \to TriAG^1 \in \mathcal{M}$ *and* $g\colon TriAG^0 \to TriAG^2$ *in* **TriAGraph**, *where* $t^3\colon TriAG^3 \to TriATG$ *is uniquely determined by the pushout properties of* $(TriAG^3, f', g')$ *in* **TriAGraph**.

Proof
Follows from the construction of pushouts in slice categories.

□

Next, we consider the construction of pullbacks in which one of the morphisms belongs to $\mathcal{M}$.

Fact 25 *(Pullbacks along* $\mathcal{M}$*-morphisms in* **TriAGraph***)*
Given the attributed triple morphisms $f'\colon TriAG^2 \to TriAG^3 \in \mathcal{M}$ *and* $g'\colon TriAG^1 \to TriAG^3$, *a pullback* $(TriAG^0, f, g)$ *in* **TriAGraph**, *with* $TriAG^k = (TriG^k, D^k)$, $TriG^k = (G_1^k, G_2^k, G_C^k, c_1^k, c_2^k)$, *and* $G_i^k = (V_{G_i}^k, V_{D_i}^k, E_{G_i}^k, E_{NA_i}^k, E_{EA_i}^k, (source_{j_i}^k, target_{j_i}^k)_{j\in\{G,NA,EA\}})$ *for* $k \in \{0,1,2,3\}$, $i \in \{1,2,C\}$ *can be constructed as follows (see Figure 24), where* $f' \in \mathcal{M}$ *and any other pullback* $TriAG'$ *is isomorphic to* $TriAG^0$*:*

1. $(V_{G_i}^0, f_{V_{G_i}^0}^i, g_{V_{G_i}^0}^i)$ *is pullback of* $(V_{G_i}^3, f_{V_{G_i}^3}'^i, g_{V_{G_i}^3}'^i)$ *in* **Sets** *for* $i \in \{1,2,C\}$.
2. $(E_{G_i}^0, f_{E_{G_i}^0}^i, g_{E_{G_i}^0}^i)$ *is pullback of* $(E_{G_i}^3, f_{E_{G_i}^3}'^i, g_{E_{G_i}^3}'^i)$ *in* **Sets** *for* $i \in \{1,2,C\}$.
3. $(E_{NA_i}^0, f_{E_{NA_i}^0}^i, g_{E_{NA_i}^0}^i)$ *is pullback of* $(E_{NA_i}^3, f_{E_{NA_i}^3}'^i, g_{E_{NA_i}^3}'^i)$ *in* **Sets** *for* $i \in \{1,2,C\}$.
4. $(E_{EA_i}^0, f_{E_{EA_i}^0}^i, g_{E_{EA_i}^0}^i)$ *is pullback of* $(E_{EA_i}^3, f_{E_{EA_i}^3}'^i, g_{E_{EA_i}^3}'^i)$ *in* **Sets** *for* $i \in \{1,2,C\}$.
5. $(V_{D_i}^0, f_{V_{D_i}^0}^i, g_{V_{D_i}^0}^i) = (V_{D_i}^1, id, g_{V_{D_i}^0}^i)$ *with* $g_{V_{D_i}^0}^i = f_{V_{D_i}^2}'^{-1,i} \circ g_{V_{D_i}^1}'^i : V_{D_i}^0 \to V_{D_i}^2$ *for* $i \in \{1,2,C\}$.
6. $(D^0, f_D, g_D) = (D^1, id, g_D)$ *with* $g_D = g_D' \circ f_D^{'-1}\colon D^0 \to D^2$.
7. *The* $source_{j_i}^0$ *and* $target_{j_i}^0$ *operations (for* $j \in \{G, NA, EA\}$, $i \in \{1,2,C\}$*) are uniquely determined by the pullbacks in (1)-(4).*
8. *The* c_1^0 *and* c_2^0 *functions are also uniquely determined by the pullbacks in (1)-(4).*

Proof
The componentwise construction leads to a well-defined attributed triple graph $TriAG^0$ and attributed triple morphisms f and g with $f \in \mathcal{M}$. The universal pullback property follows from the universal property of pullbacks in **Sets**. Note also that $(V_{D_i}^0, f_{V_{D_i}^0}^i, g_{V_{D_i}^0}^i)$ and (D_i^0, f_D, g_D) are pullbacks in **Sets** resp. $\mathbf{DSIG} - \mathbf{Alg}$.

□

Fact 26 *(Pullbacks along $\mathcal{M}$-morphisms in* **TriAGraph$_{\mathbf{TriATG}}$**$)$
Given the ATT-morphisms $f'\colon (TriAG^2, t^2) \rightarrow (TriAG^3, t^3) \in \mathcal{M}$ and $g'\colon (TriAG^1, t^1) \rightarrow (TriAG^3, t^3)$, the pullback $((TriAG^0, t^0), f, g)$ in **TriAGraph$_{\mathbf{TriATG}}$** *can be constructed as the pullback $(TriAG^0, f, g)$ of $f'\colon TriAG^2 \rightarrow TriAG^3 \in \mathcal{M}$ and $g'\colon TriAG^1 \rightarrow TriAG^3$ in* **TriAGraph**, *where $t^0\colon TriAG^0 \rightarrow TriATG$ is uniquely determined by $t^0 = t^1 \circ f = t^2 \circ g$*

Proof
Follows from the construction of pullbacks in slice categories.

□

Fact 27 *(Pushouts along $\mathcal{M}$-morphisms are pullbacks)*
Pushouts along $\mathcal{M}$-morphisms in **TriAGraph** *and* **TriAGraph$_{\mathbf{TriATG}}$** *are pullbacks.*

Proof
Due to the fact that pushouts along injective functions in **Sets** are also pullbacks.

□

5 Attributed Typed Triple Graphs as Adhesive HLR Category

In this section, we give an alternative formalization of attributed triple graphs in terms of a comma category construction. This will facilitate proving that both **TriAGraph** and **TriAGraph$_{\mathbf{TriATG}}$** are adhesive HLR categories. This means that we can use the main results of graph transformation theory, as these results have been lifted from graphs to adhesive HLR categories [Ehrig *et al.*, 2006].

Category **TriAGraph** is isomorphic to a comma category **ComCat**$(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id})$, where V_1 and V_2 are forgetful functors. The first one goes from category **TriEGraph** to category **Set** and "forgets" the triple graph structure, taking the set of data values of one of the graphs (as all the V_{D_i} sets are equal). The second functor V_2 goes from category **DSIG** $-$ **Alg** to **Set**, placing together in a set the elements of the carrier sets for attribution (disjointly). The resulting comma category has objects $(TG, D, op\colon V_1(TG) \rightarrow V_2(D))$ which satisfy $V_1(TG) = V_2(D)$ and $op = id$. This category, and therefore **TriAGraph$_{\mathbf{TriATG}}$**, can be proved to be an adhesive HLR category. This is formalized in the following facts, which are adapted from [Ehrig *et al.*, 2006].

Fact 28 *(Comma Category Construction for* **TriAGraph**$)$
Category **TriAGraph** *is isomorphic to a subcategory* **ComCat**$(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id})$ *of the comma category* **ComCat**$(\mathbf{V_1}, \mathbf{V_2}; \mathbf{I})$ *with $I = \{1\}$.*

Construction:
Let V_1 and V_2 be the forgetful functors:

- V_1: **TriEGraph** $\rightarrow$ **Set**, with $V_1(TG) = V_{D_1}$, and $V_1(f = (f^1, f^2, f^C)) = f^1_{V_{D_1}}$. Note that, the choice of V_{D_1} instead of V_{D_2} or V_{D_C} is irrelevant, as by construction, $V_{D_1} = V_{D_2} = V_{D_C}$.
- V_2: **DSIG** $-$ **Alg** $\rightarrow$ **Set**, with $V_2(D) = \uplus_{s \in S'_D} D_s$ and $V_2(f_D) = \uplus_{s \in S'_D} f_{D_s}$.

$\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id})$ is the subcategory of $\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{I})$ with $I = \{1\}$ where the objects $(TG, D, op\colon V_1(TG) \rightarrow V_2(D))$ satisfy $V_1(TG) = V_2(D)$ and $op = id$.

Proof:
For an attributed triple graph $TriAG = (TriG = (G_1, G_2, G_C, c_1, c_2), D)$ with
$G_i = (V_{G_i}, V_{D_i}, E_{G_i}, E_{NA_i}, E_{EA_i}, (source_{j_i}, target_{j_i})_{j \in \{G, NA, EA\}}))$, we have by Definition 14 that $\uplus_{s \in S'_D} D_s = V_{D_i}$, for $i \in \{1, 2, C\}$. For morphisms $f\colon TriAG^1 \rightarrow TriAG^2$ with $f = ((f^1, f^2, f^C), f_D)$, we have the commutativity of (1), (2) and (3) in Figure 26. Taking each sort in each signature and the data part of G_1, the commutativity of (3) in Figure 27 follows.

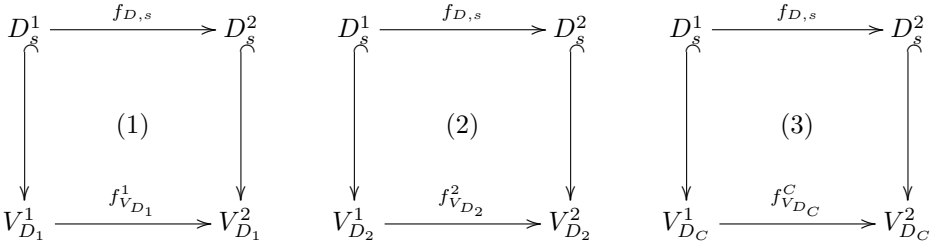


Figure 26: Condition for attributed triple graph morphisms.

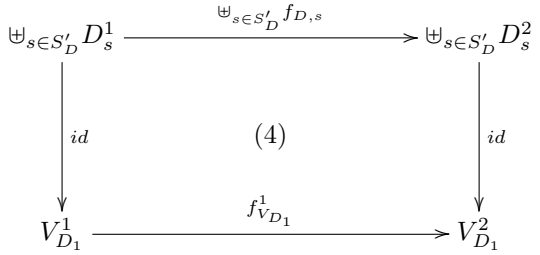


Figure 27: Condition for attributed triple graph morphisms, source graph G_1. This is the compatibility condition of f and f_D in the comma category construction.

Using the condition $V_1(TG) = V_2(D)$ of the comma category construction, and the definition of functors V_1 and V_2, we have that $V^1_{D_1} = \uplus_{s \in S'_D} D_s$, which is exactly the condition for an attributed triple graphs (an object in

category **TriAGraph** see Definition 14). Moreover, taking the definition of the arrows in the comma category, we have that $f^1_{V_{D_1}} = \uplus_{s \in S'_D} f_{D_s}$. This is the condition for attributed triple graph morphisms shown in Figure 27, because $V_1(f = (f^1, f^2, f^C)) = f^1_{V_D}$ and $V_2(f_D) = \uplus_{s \in S'_D} f_{D_s}$. This implies that $\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id}) \cong \mathbf{TriAGraph}$.

□

For the following fact, we use two results from [Ehrig *et al.*, 2006] (Theorems 4.13.3 and 4.13.4). The first one states that if $(\mathbf{C}, \mathcal{M})$ is a (weak) adhesive HLR category, then for each category $\mathbf{X}$ the functor category $([\mathbf{X}, \mathbf{C}]$, $\mathcal{M}$-functor transformation) is a (weak) adhesive HLR category. An $\mathcal{M}$-functor transformation is a natural transformation $t\colon F \to G$ where all morphisms $t_X\colon F(X) \to G(X)$ are in $\mathcal{M}$.

The second theorem states that the comma category $(\mathbf{ComCat}(\mathbf{F}, \mathbf{G}; \mathcal{I}), \mathcal{M})$ with $\mathcal{M} = (\mathcal{M}_1 \times \mathcal{M}_2) \cap Mor_{ComCat(V_1,V_2;I)}$ is an adhesive HLR category if F preserves pushouts along $\mathcal{M}_1$-morphisms and G preserves general pullbacks. In case of weak adhesive HLR categories, it is enough that F preserves pushouts along $\mathcal{M}_1 - morphisms$ and G preserves pullbacks along $\mathcal{M}_2$-morphisms.

Fact 29 *(***TriAGraph** *is an Adhesive HLR category)*

Let $\mathcal{M}_1$ be the class of injective **TriE-graph** *morphisms and $\mathcal{M}_2$ the class of $DSIG-$isomorphisms. Then $(\mathbf{TriEGraph}, \mathcal{M}_1)$ is a functor category over $(\mathbf{Set}, \mathcal{M}_1)$ and hence adhesive HLR category. In a similar way, $(\mathbf{DSIG - Alg}, \mathcal{M}_\mathbf{2})$ is also an adhesive HLR category. This implies that $\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{I})$ with $I = 1$ and $\mathcal{M} = (\mathcal{M}_1 \times \mathcal{M}_2) \cap Mor_{ComCat(V_1,V_2;I)}$ is an adhesive HLR category, provided that V_1 preserves pushouts along $\mathcal{M}_1$ morphisms and V_2 preserves pullbacks. Pushouts in category* **TriEGraph** *are constructed componentwise in* **Set***; therefore, V_1 preserves pushouts. As shown in [Ehrig* et al.*, 2006], functor $F\colon$ $\mathbf{DSIG - Alg} \to \mathbf{Set}$ preserves pullbacks; therefore, V_2 preserves pullbacks. Finally, it can also be shown that a special choice of pushouts and pullbacks in* **TriEGraph** *and* **DSIG-Alg** *leads also to pushouts and pullbacks in the subcategory $\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id})$, therefore $(\mathbf{ComCat}(\mathbf{V_1}, \mathbf{V_2}; \mathbf{Id}), \mathcal{M})$ and $(\mathbf{TriAGraph}, \mathcal{M})$ are adhesive HLR categories.*

For the following fact, we use the result of Theorem 4.13.2 in [Ehrig *et al.*, 2006], which states that the slice category $(\mathbf{C/X}, \mathcal{M} \cap \mathbf{C/X})$ is an adhesive HLR category, where $\mathcal{M} \cap C/X$ are monomorphisms in C/X (monomorphisms in C are also monomorphisms in C/X but the converse is not necessarily true).

Fact 30 *(***TriAGraph**$_{\mathbf{TriATG}}$ *is an Adhesive HLR category)*

$(\mathbf{TriAGraph}_{\mathbf{TriATG}}, \mathcal{M})$ *is a slice category of $(\mathbf{TriAGraph}, \mathcal{M})$, (with $\mathcal{M}$ the class of morphisms $f = ((f^1, f^2, f^C), f_D)$ in which f^i are injective and f_D is a $DSIG-$Algebra homomorphism), and therefore an adhesive HLR category.*

6 Attributed Typed Triple Graph Transformation

Once proved that ($\mathbf{TriAGraph_{TriATG}}, \mathcal{M}$) is an adhesive HLR category, we can instantiate the general graph transformation theory for HLR systems [Ehrig *et al.*, 2006]. For illustrative purpose, we present only the basic concepts of graph transformation: production, derivation and grammar. Later, we extend productions with application conditions. Further results for HLR systems can be found in [Ehrig *et al.*, 2006].

The main idea in the DPO approach is that rules are modelled using three components: L, K and R. The L component (the rule's left hand side, LHS) represents the necessary elements to be found in the structure where the rule is applied (a graph, a triple graph, a Petri net, etc.). The *kernel* K contains the elements that are preserved by the rule application. Finally, R (the rule's right hand side, RHS) contains the elements that should replace the identified part in the structure that is being rewritten. Note that $L - K$ are the elements that should be deleted by the rule application, while $R - K$ are the elements that should be added. In our case, L, K and R are ATT-graphs.

Definition 31 *(Triple Rule)*

Given an attributed type triple graph $TriATG$ with data signature $DSIG$ an attributed typed triple graph rule (short triple rule) $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of ATT-graphs L, K and R with common $DSIG$-algebra $T_{DSIG}(X)$ (the $DSIG$-termalgebra with variables X) and injective ATT-morphisms $l\colon K \to L$, and $r\colon K \to R$.

Remark: as l and r are injective ATT-morphisms, the $DSIG$-part of l and r is the identity on $T_{DSIG}(X)$.

In order to apply a triple rule p to an ATT-graph G (called *host* ATT-graph), an occurrence of the LHS should be found in the graph. That is, an ATT-morphism $m\colon L \to G$ needs to be found. Once the morphism is found, the rule is applied in two steps. In the first one, the elements in $m(L - l(K))$ are deleted from G, yielding graph D. In the second step, the elements from $R - r(K)$ are added to D, resulting in graph H. Notice that these two steps are modelled by two pushouts. As shown in the previous section, in **TriAGraph** and $\mathbf{TriAGraph_{TriATG}}$ pushouts are built componentwise, by calculating the pushout of each set in each one of the three E-graphs.

Definition 32 *(Direct Triple Graph Derivation)*

Given a triple rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$, an ATT-graph G and an ATT-morphism $m : L \to G$ (called match), a direct triple graph derivation (short direct derivation) $G \overset{p,m}{\Longrightarrow} H$ from G is given by the double pushout (DPO) diagram in category $\mathbf{TriAGraph_{TriATG}}$ shown in Figure 28, where (1) *and* (2) *are pushouts.*

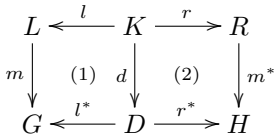


Figure 28: Direct derivation as DPO construction.

Figure 29 shows an example of direct derivation. The rule simply connects an object with its corresponding class, creating an edge (labelled "7" in R and H). Triple morphisms l, r, m, d, m^*, l^* and r^* have been depicted with numbers.

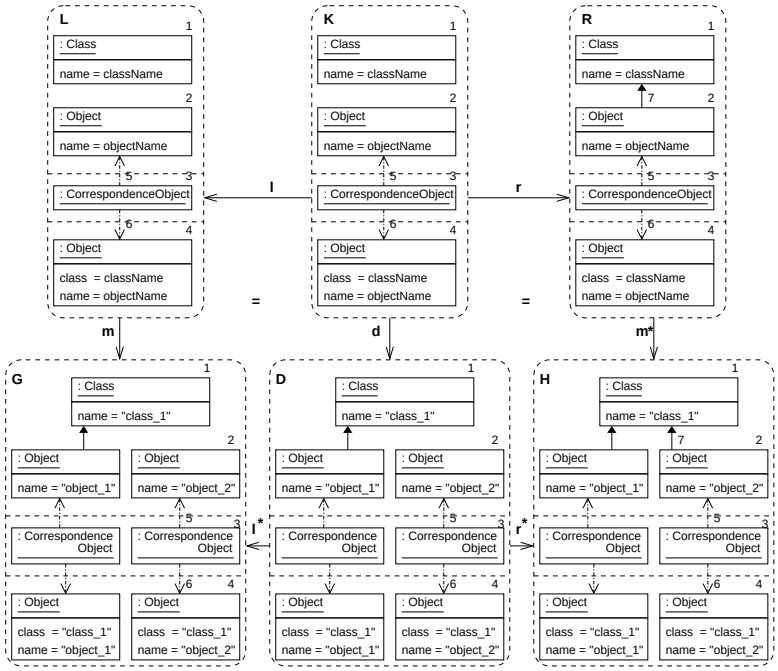


Figure 29: A direct derivation example.

Note however, that in the first pushout (1), what we are really calculating is graph D, that is, a pushout complement. In order for the pushout complement to exist, besides the well-known dangling edge and identification conditions [Ehrig *et al.*, 1999] (also known as gluing condition [Ehrig *et al.*, 2006]) for each E-graph in the triple graph, an additional condition is needed concerning the correspondence functions. We first present the gluing condition for **AGraph** and $\mathbf{AGraph_{ATG}}$, and next the condition for **TriAGraph** and $\mathbf{TriAGraph_{ATG}}$.

Definition 33 *(Gluing Condition for* **AGraph** *and* $\mathbf{AGraph_{ATG}}$*, taken from [Ehrig* et al.*, 2006])*

1. *Given an attributed graph production* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$, *an attributed graph* G *and a match* $m : L \to G$ *in* **AGraph** *with* $X = (V_G^X, V_D^X, E_G^X, E_{NA}^X, E_{EA}^X, (source_j^X, target_j^X)_{j\in\{G,NA,EA\}}, D^X)$ *for all* $X \in \{L, K, R, G\}$.

   - *The gluing points* GP *are those graph items in* L *that are not deleted by* p, *i.e.* $GP = l_{V_G}(V_G^K) \cup l_{E_G}(E_G^K) \cup l_{E_{NA}}(E_{NA}^K) \cup l_{E_{EA}}(E_{EA}^K)$.
   - *The identification points* IP *are those graph items in* L *that are identified by* m, *i.e.* $IP = IP_{V_G} \cup IP_{E_G} \cup IP_{E_{NA}} \cup IP_{E_{EA}}$ *with:*
     $IP_{V_G} = \{a \in V_G^L | \exists a' \in V_G^L, a \neq a', m_{V_G}(a) = m_{V_G}(a')\}$
     $IP_{E_j} = \{a \in E_j^L | \exists a' \in E_j^L, a \neq a', m_{E_j}(a) = m_{E_j}(a')\}$, *for all* $j \in \{G, NA, EA\}$
   - *The dangling points* DP *are those graph items in* L, *whose images are the source or target of an item that does not belong to* $m(L)$, *i.e.*
     $DP = DP_{V_G} \cup DP_{E_G}$
     $DP_{V_G} = \{a \in V_G^L | (\exists a' \in E_{NA}^G - m_{E_{NA}}(E_{NA}^L),\ m_{E_{NA}}(a) = source_{NA}^G(a')) \vee (\exists a' \in E_G^G - m_{E_G}(E_G^L),\ m_{E_G}(a) = source_G^G(a')\ or\ m_{E_G}(a) = target_G^G(a'))\}$
     $DP_{E_G} = \{a \in E_G^L | (\exists a' \in E_{EA}^G - m_{E_{EA}}(E_{EA}^L),\ m_{E_{EA}}(a) = source_{EA}^G(a')\}$
     p *and* m *satisfy the gluing condition in* **AGraph** *if all identification and all dangling points are also gluing points, i.e.* $IP \cup DP \subseteq GP$

2. *Given* p *and* m *in* $\mathbf{AGraph_{ATG}}$, *they satisfy the gluing condition in* $\mathbf{AGraph_{ATG}}$ *if* p *and* m *considered in* **AGraph** *satisfy the gluing condition in* **AGraph**.

Definition 34 *(Gluing Condition in* **TriAGraph** *and* $\mathbf{TriAGraph_{ATG}}$*)*

1. *Given a triple rule* $p = (L \xleftarrow{l} K \xrightarrow{r} R)$, *an attributed triple graph* $TriAG = (TriG = (G_1, G_2, G_C, c_1^G, c_2^G), D)$ *and a match* $m = (m_{TriG} = (m^1, m^2, m^C), m_D) : L \to G$ *in* **TriAGraph** *with* $X = (V_G^X, V_D^X, E_G^X, E_{NA}^X, E_{EA}^X, (source_j^X, target_j^X)_{j\in\{G,NA,EA\}})$ *for all* $X \in \{L^i, K^i, R^i, G_i\}$, *for* $i \in \{1, 2, C\}$.

   - *The correspondence gluing points* CGP *are the graph nodes and edges in source or target graphs of* L, *that are not deleted by* p, *i.e.* $CGP = l_{V_G}^1(V_G^{K^1}) \cup l_{E_G}^1(E_G^{K^1}) \cup l_{V_G}^2(V_G^{K^2}) \cup l_{E_G}^2(E_G^{K^2})$.
   - *The correspondence dangling points* CDP *are those graph nodes or edges in the source or target graphs of* L, *whose images are the target of a correspondence function* c_i *from an element which does not belong to* $m_{V_G}^C(L_{V_G}^C)$, *i.e.* $CDP = CDP_1 \bigcup CDP_2$, *with:*
     $CDP_i = \{a \in L_{V_G}^i \cup L_{E_G}^i | \exists x \in G_{V_G}^C - m_{V_G}^C(L_{V_G}^C)\ with\ c_i^G(x) = (m_{V_G}^i \uplus m_{E_G}^i)(a)\}$, *for* $i \in \{1, 2\}$

   p *and* m *satisfy the gluing condition in* **TriAGraph**, *if :*

   - *Matches* $m_i' = (m^i, m_D) : (L^i, D) \to (G^i, D)$ *(for* $i \in \{1, 2, C\}$*) in* **AGraph** *satisfy the gluing condition for* **AGraph**.

- *All correspondence dangling points are also correspondence gluing points, i.e.* $CDP \subseteq CGP$

2. *Given* p *and* m *in* $\mathbf{AGraph_{ATG}}$*, they satisfy the gluing condition in* $\mathbf{AGraph_{ATG}}$ *if* p *and* m *considered in* **AGraph** *satisfy the gluing condition in* **AGraph**.

The correspondence gluing condition states that an element in the source or target graph cannot be deleted, if it is the target of a correspondence function from an element in the correspondence graph that is not deleted.

Note that, in the second item of Definition 34.1, there is no need to consider the nodes in the correspondence graph that are source of a correspondence function. This is because if such a node is erased, the definition of the correspondence function for that node is also erased. The situation is quite similar to when an edge is deleted in a regular graph, then the definition for the source and target functions are deleted for that edge. However, care should be taken when deleting the target of source or target functions (targets of c_1 and c_2 functions), as the functions would not be well defined.

Note also that there is no need for an "identification condition" for the correspondence function. If two nodes in the correspondence graph of L are identified into a single node of G, and one is deleted and the other not, then the rule cannot be applied, due to the identification condition in the correspondence graph. An example of this is shown in Figure 30, where nodes B and C are identified into node BC in the correspondence graph of triple graph G. The rule cannot be applied, as one is deleted and the other one is preserved.

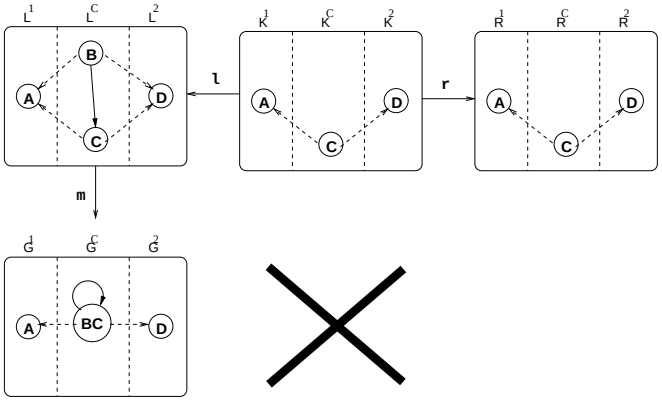


Figure 30: Forbidden rule application due to identification condition in the correspondence graph.

Fact 35 *(Existence and Uniqueness of Attributed (Typed) Triple Context Graphs)*
For an attributed (typed) graph triple rule p*, an attributed (typed) graph* G *and a match* $m : L \to G$*, the attributed (typed) triple context graph* D *with*

PO(1) *exists in* **TriAGraph** *(***TriAGraph$_{\mathbf{ATG}}$***), iff the gluing condition is satisfied in* **TriAGraph** *(***TriAGraph$_{\mathbf{ATG}}$***). If D exists, it is unique up to isomorphism*

$$\begin{array}{ccc} L & \xleftarrow{l \in \mathcal{M}} & K \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle k} \\ G & \xleftarrow[f \in \mathcal{M}]{} & D \end{array}$$

Proof:
'$\Rightarrow$' Given the $PO(1)$ then the properties of the gluing condition follow from the properties of pushouts along $\mathcal{M}$-morphism in **TriAGraph** and **TriAGraph$_{\mathbf{ATG}}$**:

- As pushouts are built componentwise in **TriAGraph** (**TriAGraph$_{\mathbf{ATG}}$**), then the gluing condition is satisfied separately for each graph in the triple graph (see [Ehrig *et al.*, 2006]).
- Consider $v \in CDP$ and assume $v \in L^i_{V_G}$ (that is, v is a node), with $x \in G^C_{V_G} - m^C_{V_G}(L^C_{V_G})$ and $c^G_i(x) = m^i_{V_G}(v) = b$. As G is pushout, then $f \in \mathcal{M}$ and m are jointly surjective. Therefore, both b and x have preimages in D. Be $v' \in D^i_{V_G}$ and $x' \in D^C_{V_G}$ with $f^i_{V_G}(v') = b$ and $f^C_{V_G}(x') = x$. As G is a pushout and l is injective ($l \in \mathcal{M}$), then $\exists_1 v'' \in K^i_{V_G}$, such that $l^i_{V_G}(v'') = v$ and $k^i_{V_G}(v'') = v'$. Thus, v is not deleted. A similar reasoning holds for the case of v being an edge. Therefore $CDP \subseteq CGP$ and all correspondence dangling points are also correspondence gluing points.

'$\Leftarrow$' If the gluing condition is satisfied we can construct $D = (TriG^D = (G^D_1, G^D_2, G^D_C, c^D_1, c^D_2), D^D)$, with $G^D_i = (V^D_{G_i}, V^D_{D_i}, E^D_{G_i}, E^D_{NA_i}, E^D_{EA_i}, (source^D_{j,i}, target^D_{j,i})_{j \in \{G,NA,EA\}})$ $(i = \{1, 2, C\})$, with k, f, and $type_D : D \to TriATG$ as follows:

- $V^D_{G_i} = (V^D_{G_i} - m^i_{V_G}(L^i_{V_G})) \cup m^i_{V_G} \circ l^i_{V_G}(K^i_{V_G})$.
- $V^D_{D_i} = V^G_{D_i}$.
- $D^i_{E_j} = (G_{E_j} - m^i_{E_j}(L_{E_j})) \cup m \circ l(K^i_{E_j})$, for $j \in \{G, NA, EA\}$.
- $source^D_{G,i} = source^G_{G,i}|_{V^D_{G_i}}$, $target^D_{G,i} = target^G_{G,i}|_{V^D_{G_i}}$.
- $source^D_{j,i} = source^G_{j,i}|_{E^D_{j_i}}$, $target^D_{j,i} = target^G_{j,i}|_{E^D_{j_i}}$ for $j \in \{NA, EA\}$.
- $D^D = D^G$
- $c^D_i(x) = c^G_i(x)$, $\forall x \in D^C_{V_G}$, for $i = \{1, 2\}$.
- $k(x) = m(l(x))$ for all items x in K
- f inclusion

- $type_D = type_G|_D$

□

A triple derivation is a sequence of zero or more direct triple derivations and is depicted as $G_0 \Rightarrow^* G_n$. A triple graph grammar is made of a set of triple rules and an initial ATT-graph, together with the data signature and the type graph. The language of the triple grammar consists of all the ATT-graphs that can be obtained by means of derivations, starting from the initial ATT-graph.

Definition 36 *(Triple Graph Grammar and Language)*
A triple graph grammar $TGG = (DSIG, TriATG, P, TriAS)$ is made of a data signature $DSIG$, an attributed type triple graph $TriATG$, a set P of triple rules, and an initial ATT-graph $TriAS$ typed over $TriATG$. The language generated by TGG is given by $L(TGG) = \{TriAG | TriAS \Rightarrow^* TriAG\}$.

In addition, we provide triple rules with application conditions, in the style of [Heckel and Wagner, 1995]. We first define conditional constraints on ATT-graphs. An application condition is then a conditional constraint on the L component of the triple rule.

Definition 37 *(Triple Conditional Constraint)*
A triple conditional constraint $cc = (x\colon L \to X, A)$ over an ATT-graph L consists of an ATT-morphism x and a set $A = \{y_j\colon X \to Y_j\}$ of ATT-morphisms. An ATT-morphism $m\colon L \to G$ satisfies a constraint cc over L, written $m \models_L cc$, iff $\forall n\colon X \to G$ with $n \circ x = m$ $\exists o\colon Y_j \to G$ (where $y_j\colon X \to Y_j \in A$) such that $o \circ y_j = n$ (see Figure 31).

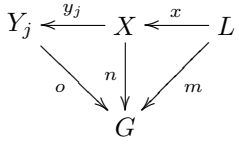


Figure 31: A triple conditional constraint satisfied by m.

Roughly, the constraint is satisfied by the morphism m if no occurrence of X is found in G, but if some is found, then an occurrence of some Y_j should also be found. If the set A is empty, then we have a negative application condition (NAC), where the existence of an ATT-morphism n implies $m \not\models_L cc$. Morphisms x and y_j are total, but we use a shortcut notation. In this way, the subgraph of L (resp. X) that do not have an image in X (resp. Y_j) is isomorphically copied into X (resp. Y_j) and appropriately linked with their elements.

We assign triple rules a set AC of triple conditional constraints (called application condition). For a rule to be applicable at a match m, it must satisfy all the application conditions in the set. Figure 32 shows an example of two triple rules with NACs (the set A in the application condition is empty). Following the mentioned shortcut notation, in the NAC only the additional elements to LHS

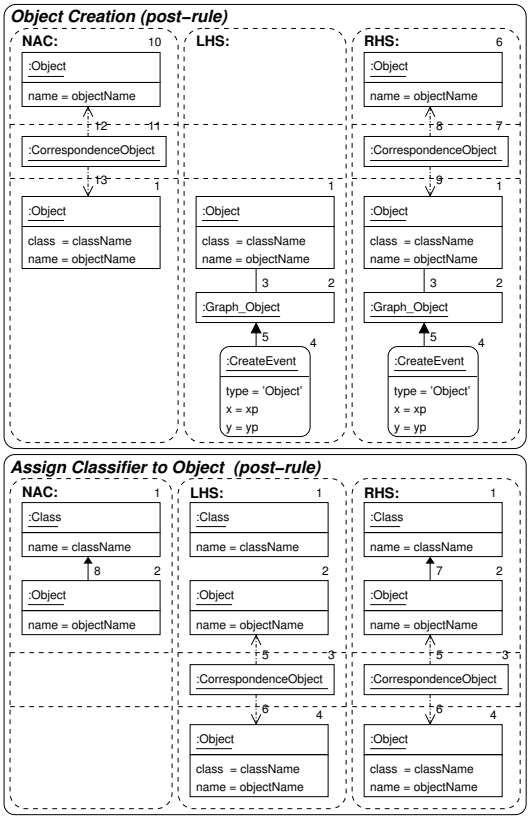


Figure 32: An example with two triple rules.

and their context have been depicted. The kernel triple graph K of both rules is not explicitly shown. Their elements are those having the same numbers in L and R (which are labelled LHS and RHS). We use this notation throughout the paper. The first rule creates an object in the target graph (in the upper side part of the rule), if an object has been created in the source graph. The second rule connects an object with its classifier in the target graph.

7 Attributed Typed Triple Graph Transformation with Inheritance

In this section, we present an inheritance concept, similar to the one presented in [Bardohl *et al.*, 2004] and [Ehrig *et al.*, 2005a], but adapted to ATT-graphs. In addition, we consider edge inheritance and rules have more complex application conditions (not only NACs). Node inheritance is an extension mechanism because it allows children nodes to inherit edges and attributes of parent nodes.

Edge inheritance is also an extension mechanism, by allowing children edges to inherit attributes of parent edges, but it is also used as a restriction technique in node inheritance hierarchies to forbid certain connections between children nodes.

The main idea is to add the concept of inheritance to the type triple graph, in a similar way as in UML class diagrams. Then, we can put elements in the LHS of rules whose (node or edge) type has a number of (node or edge) subtypes. In this way, any node having the exact type or any of its subtypes can match that element. That is, these inheritance-extended rules are equivalent to a number of rules with concrete typing (called concrete rules), resulting from the valid substitutions of each graph node or edge by elements of their subtypes. In this way, rules become more compact.

Therefore, the typing of triple graphs by triple type graphs has to be extended. This is shown in subsection 7.1. In this section, we also show how to "flatten" such type triple graphs with inheritance to obtain a regular type triple graph. In 7.2, we extend triple rules with the inheritance concept and obtain *Inheritance-Extended Triple Rules*. If one of such rules is equivalent to more than one concrete rule, then the former rule is called meta-rule. Finally, in section 7.3, we show that a derivation made of meta-rule applications is equivalent to a derivation of concrete rules, and the other way around.

7.1 Attributed Type Triple Graphs with Inheritance

We first start defining an attributed type triple graph with inheritance (or meta-model triple, short MMT) in a similar way as in [Ehrig *et al.*, 2005a], but, in addition to the inheritance of nodes, we also allow inheritance of edges. The extended type graph with inheritance is defined like a normal type triple graph, plus two additional graphs for the node and edge inheritance hierarchies, and two sets of abstract nodes and edges. Moreover, for technical reasons concerning the correspondence functions, multiple inheritance for nodes is forbidden in the correspondence graph. As in [Taentzer and Rensink, 2005], we only allow an edge to inherit from another one, if the source and target nodes of the child edge belong to the children nodes of the source and target nodes of parent edge. For this purpose, we use the notion of *clan* (see definition 39), which is a function that applied to a node or edge returns the set of all its children nodes or edges, including itself.

Definition 38 *(Attributed Type Triple Graph with Inheritance)*
An attributed type triple graph with inheritance (short meta-model triple, MMT) $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$*, consists of:*

- *An attributed type triple graph* $TriATG = (TriTG, Z)$*, where* $TriTG = (TG_1, TG_2, TG_C, tc_1, tc_2)$ *is a TriE-graph.*
- *Three node inheritance graphs* $VI_i = (VI^i_V, VI^i_E, vs^i\colon VI^i_E \to VI^i_V, vt^i\colon VI^i_E \to VI^i_V)$ *(for* $i = \{1, 2, C\}$*) with* $VI^i_V = TG^i_{V_G}$*.* VI^i_V *is the set of nodes, and* VI^i_E *is the set of edges, and* vs *and* vt *are the source*

and target functions for the edges. Multiple inheritance is forbidden in the correspondence graph. That is, $\forall n \in VI_V^C$, $|\{e \in VI_E^C | vs^C(e) = n\}| \leq 1$.

- *Three edge inheritance graphs* $EI_i = (EI_V^i, EI_E^i, es^i\colon EI_E^i \to EI_V^i,$ $et^i\colon EI_E^i \to EI_V^i)$ *(for* $i = \{1, 2, C\}$*) with* $EI_V^i = TG_{E_G}^i$. *Moreover* $\forall e, e' \in EI_V^i, x \in EI_E^i$ *such that* $es^i(x) = e'$ *and* $et^i(x) = e$ *(i.e.* e' *inherits from* e*), we have that* $source_{G_i}(e') \in clan_{VI^i}(source_{G_i}(e))$ *and* $target_{G_i}(e') \in clan_{VI^i}(target_{G_i}(e))$.
- *Three sets* $AV_i \subseteq VI_V^i$, *for* $i = \{1, 2, C\}$, *called abstract nodes.*
- *Three sets* $AE_i \subseteq EI_V^i$, *for* $i = \{1, 2, C\}$, *called abstract edges.*

Figure 33 shows an example meta-model triple, which is an extension of the attributed type triple graph in Figure 20. We have collapsed each graph TG_i, node inheritance graph VI_i and edge inheritance graph EI_i in a unique graph. The edges of the inheritance graphs are shown with hollow edges (following the usual UML notation) and the elements in AV^i and AE^i are shown in italics. We treat "composition" edges (the ones with a black diamond) as any other edge in E_G^i.

The upper part of the meta-model triple depicts a slight variation of the UML 1.5 standard meta-model proposed by OMG (see [UML]) for sequence diagrams (abstract syntax). The lowest meta-model in the figure declares the concrete appearance concepts and their relations. The elements in this meta-model are in direct relationship with the graphical forms that will be used for graphical representation, which is quite different from the abstract syntax representation. Abstract class *ConcreteElement* has two abstract edges *AbsMessage* and *AbsLifeLine*. *ConcreteElement* has three concrete children: *StartPoint*, *ActivationBox* and *Object*. Abstract edge *AbsMessage* is refined by *Message*, *StartMessage* and *createMessage*. They restrict the kind of *ConcreteElement* types that can be connected through a message: *StartPoint* and *ActivationBox*, *ActivationBox* with itself and *ActivationBox* and *Object*. A similar situation happens for *AbsLifeLine*.

The correspondence meta-model specifies which elements in the concrete and abstract meta-models can be related by means of two nodes of types *CorrespondenceMessage* and *CorrespondenceObject*. The correspondence functions for the former node go to *Stimulus* and *AbsMessage*. The latter is an abstract edge, which means that nodes with type $CorrespondenceMessage$ can have correspondence functions to each one of the *AbsMessage*'s concrete children edges (that is, $StartMessage$, $Message$ and $createMessage$). Note how, including the *AbsMessage* edge reduces the number of nodes in the correspondence graph with respect to the type graph in Figure 20.

Now we formalize the sets of children nodes and edges for a given one, which we call (node and edge) inheritance clans.

Definition 39 *(Node and Edge Inheritance clan)*

Given a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i \in \{1,2,C\}})$, *the node inheritance clan for each node* $n \in VI_V^i$, *is defined as* $clan_{VI^i}(n) =$

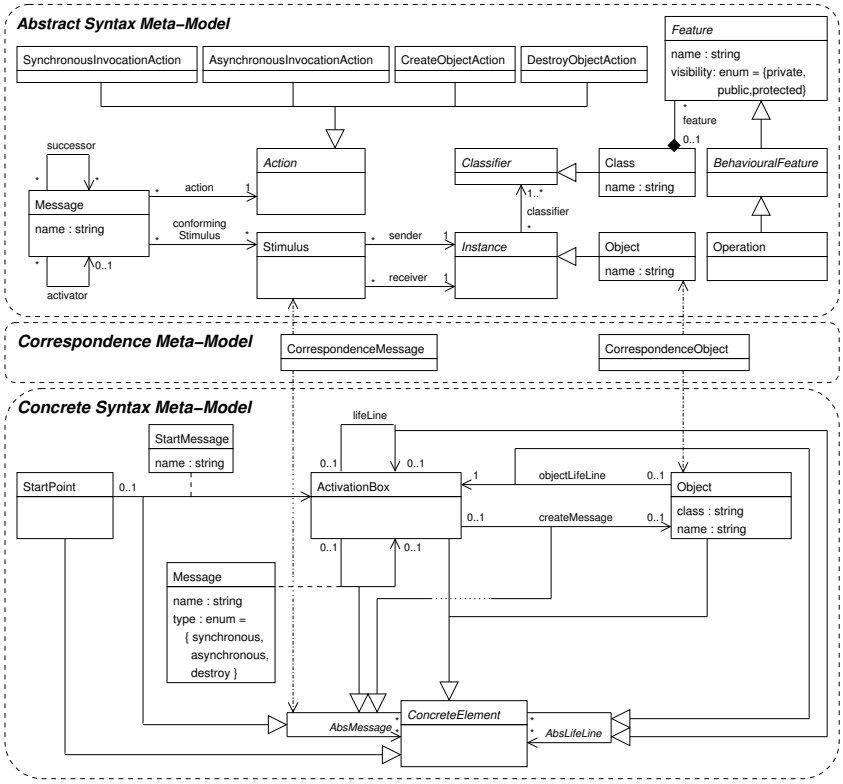


Figure 33: Meta-model triple example.

$\{n' \in VI^i_V \mid \exists\ path\ n' \xrightarrow{*} n\ in\ VI^i\} \subseteq VI^i_V\ with\ n \in clan_{VI^i}(n)$. *In a similar way, for each edge* $e \in EI^i_V$, *the edge inheritance clan is defined as* $clan_{EI^i}(e) = \{e' \in EI^i_V \mid \exists\ path\ e' \xrightarrow{*} e\ in\ EI^i\} \subseteq EI^i_V\ with\ e \in clan_{EI^i}(e)$

For example, in Figure 33, the node inheritance clan of node *Action* in the abstract graph is the set $clan_{VI^2}(Action) = \{Action, SynchronousInvocationAction, AsynchronousInvocationAction, CreateObjectAction, DestroyObjectAction\}$. The edge inheritance clan of edge *AbsMessage* in the concrete graph is the set $clan_{EI^1}(AbsMessage) = \{AbsMessage, Message, StartMessage, createMessage\}$. Finally, the node inheritance clan of *Object* in the concrete graph is the set $clan_{VI^1}(Object) = \{Object\}$. That is, the node (resp. edge) inheritance clan of a node (rep. edge) without children nodes (resp. edge) contains just one element which is the node (resp. edge) itself.

In order to benefit from the theory of graph transformation, we flatten meta-model triples to ordinary type triple graphs. The flattening makes explicit the inherited attributes and edges, and allows defining instances of attributed type graphs with inheritance. Moreover, in the correspondence graph, we allow overriding of the tc_1 and tc_2 functions.

Figure 34 shows an example meta-model triple in the upper part and its closure below. Both source and target graphs are flattened using the same procedure, while the flattening of the correspondence graph is different. For the source and target graphs, we explicitly copy edges and attributes to all nodes in the inheritance graph of a given one. In this way, for example, attribute $at1$ is copied from node E to F and H. In a similar way, relation rel is added between all the combinations of nodes in the inheritance clan of E and G. However, it is not copied between nodes F and J and between H and I, because they have two relations ($relFJ$ and $relHI$) that refine relation rel through an inheritance relation. Also, attribute $at2$ from abstract edge rel is copied to relations $relFJ$ and $relHI$, and the two additional relations created between nodes H and J and F and I.

Flattening the correspondence graph is similar; however, in addition, we have to take care of the correspondence functions. First, children nodes can override the function. In this way, if a node has a correspondence function undefined, it takes its value from the nearest ancestor in the node inheritance path for which the function is defined. In the example, node B takes the definition of tc_1 from its parent A. Moreover, the target of a correspondence function may be a node or an edge which has a number of subnodes or subedges. In this case, the flattening should reflect the fact that the correspondence function can lead to any of the subtypes. This is done by creating additional nodes in the correspondence graph with all the combinations of elements in the inheritance clans of source and target graphs. In the example, the target correspondence function of node B leads to node E, which has subnodes F and G. Therefore, we have created three nodes in the correspondence graph, for each combination of source (only C is possible) and target (E, F and H) are possible correspondence functions. The newly created nodes receive the edges and attributes of the original node (B in the example). Again, we have represented sets V_{D_i} in each one of the three E-graphs. Therefore, nodes Int in V_{D_C} and in V_{D_2} are the same.

Definition 40 *(Closure (Flattening) of Meta-Model Triple)*
Given a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$ *with* $TriATG = (TriTG, Z)$, $TriTG = (TG_1, TG_2, TG_C, tc_1, tc_2)$ *and* $TG_i = (V_G^i, V_D^i, E_G^i, E_{NA}^i, E_{EA}^i, (source_j^i, target_j^i)_{j\in\{G,NA,EA\}})$ *for* $i \in \{1, 2, C\}$, *the abstract closure of* $TriATGI$ *is the attributed type triple graph* $\overline{TriATG} = (\overline{TriTG}, Z)$ *with* $\overline{TriTG} = (\overline{TG_1}, \overline{TG_2}, \overline{TG_C}, \overline{tc_1}, \overline{tc_2})$ *and* $\overline{TG_i} = (\overline{V_G^i}, V_D^i, \overline{E_G^i}, \overline{E_{NA}^i}, \overline{E_{EA}^i}, (\overline{source_j^i}, \overline{target_j^i})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1, 2, C\}$. *First, the closure of* TG^i *for* $i = \{1, 2\}$ *is calculated as follows:*

- $\overline{V_G^i} = V_G^i$.
- $\overline{E_G^i} = \{(n_1, e, n_2) \mid e \in E_G^i \wedge n_1 \in clan_{VI^i}(source_G^i(e)) \wedge$
  $n_2 \in clan_{VI^i}(target_G^i(e)) \wedge$
  $\nexists e' \in clan_{EI^i}(e) - \{e\}$ *with* $n_1 \in clan_{VI^i}(source_G^i(e')) \wedge$
  $n_2 \in clan_{VI^i}(target_G^i(e'))\}$.

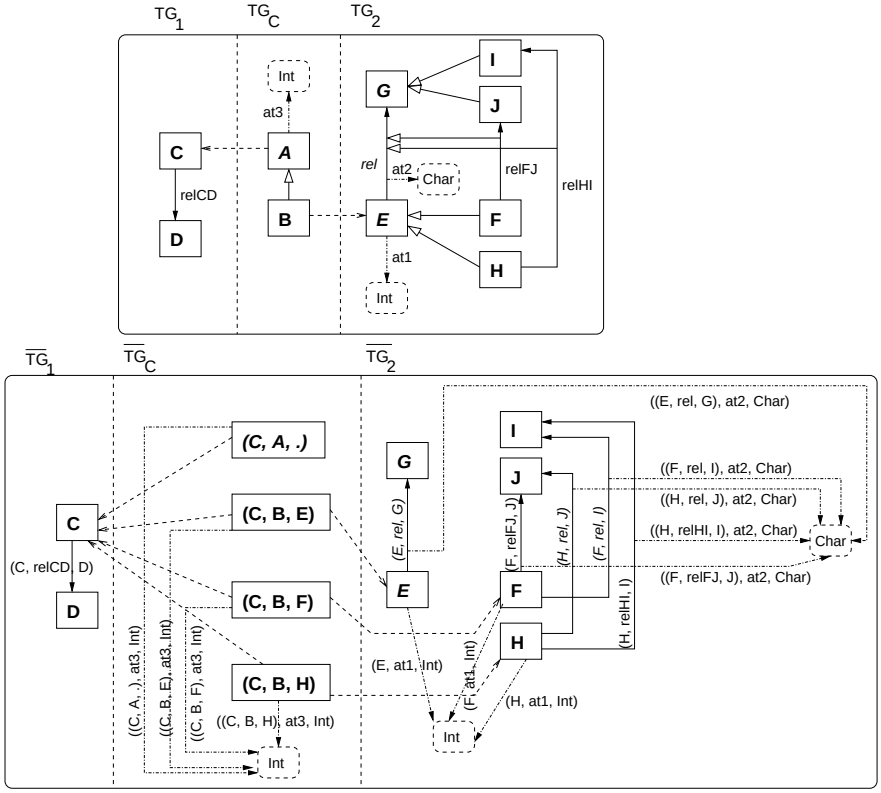


Figure 34: Abstract closure example.

- $\overline{source^i_G}((n_1, e, n_2)) = n_1 \in V^i_G$.
- $\overline{target^i_G}((n_1, e, n_2)) = n_2 \in V^i_G$.
- $\overline{E^i_{NA}} = \{(n_1, e, n_2) \mid e \in E^i_{NA} \wedge n_1 \in clan_{VI^i}(source^i_{NA}(e)) \wedge n_2 = target^i_{NA}(e)\}$.
- $\overline{source^i_{NA}}((n_1, e, n_2)) = n_1 \in V^i_G$.
- $\overline{target^i_{NA}}((n_1, e, n_2)) = n_2 \in V^i_D$.
- $\overline{E^i_{EA}} = \{((n_{11}, e_1, n_{12}), e, n_2) \mid e \in E^i_{EA} \wedge e_1 \in clan_{EI^i}(source^i_{EA}(e)) \wedge ((n_{11}, e_1, n_{12}), e, n_2) \in \overline{E^i_G} \wedge [\nexists e' \in clan_{EI^i}(source^i_{EA}(e)) - \{source^i_{EA}(e)\} \; with \; n_{11} \in clan_{VI^i}(source^i_G(e')) \wedge n_{12} \in clan_{VI^i}(target^i_G(e'))] \wedge n_2 = target^i_{EA}(e) \in V^i_D\}$.
- $\overline{source_{EA}}((n_{11}, e_1, n_{12}), e, n_2) = (n_{11}, e_1, n_{12}) \in \overline{E^i_G}$.

- $\overline{target_{EA}}((n_{11}, e_1, n_{12}), e, n_2) = n_2 \in \overline{V_D^i}$.

Then, the correspondence graph is flattened as follows:

- *We first define two auxiliary functions* $tc_i^\flat \colon V_G^C \rightarrow V_G^i \cup E_G^i \cup \{\cdot\}$ *for* $i = \{1, 2\}$ *that will be used in order to define how the overriding of the correspondence function is done. For this, we define define functions* $nearest\_anc_i \colon V_G^C \rightarrow 2^{V_G^C}$ *as follows*[1]*:*

  $nearest\_anc_i(x) = \{y \in V_G^C - \{x\} | x \in clan_{VI^C}(y) \wedge tc_i(y) \neq \cdot \wedge$
  $\nexists y_1 \in clan_{VI^C}(y) - \{y\} | x \in clan_{VI^C}(y_1) \wedge tc_i(y_1) \neq \cdot\}$ *(for* $i = 1, 2$*).*

  Then, we can define functions $tc_i^\flat$ *(for* $i = 1, 2$*) as follows:*

$$tc_i^\flat(x) = \begin{cases} tc_i(x) \; if \; tc_i(x) \neq \cdot \vee nearest\_anc_i(x) = \{\} \\ tc_i(y) \; if \; tc_i(x) = \cdot \wedge y \in nearest\_anc_i(x) \end{cases}$$

- $\overline{V_G^C} = \{(n_1, n, n_2) | n \in V_G^C \wedge tc_i^\flat(n) \in V_G^i \wedge n_i \in clan_{VI^i}(tc_i^\flat(n))$
  $(for \; i = \{1, 2\})\} \bigcup$
  $\{((source_G^1(e_1), e_1, target_G^1(e_1)), n, n_3) | n \in V_G^C \wedge tc_1^\flat(n) \in E_G^1 \wedge$
  $tc_2^\flat(n) \in V_G^2 \wedge n_3 \in clan_{VI^2}(tc_2^\flat(n)) \wedge e_1 \in clan_{EI^1}(tc_1^\flat(n))\} \bigcup$
  $\{(n_3, n, (source_G^2(e_1), e_1, target_G^2(e_1))) | n \in V_G^C \wedge tc_1^\flat(n) \in V_G^1 \wedge$
  $tc_2^\flat(n) \in E_G^2 \wedge n_3 \in clan_{VI^1}(tc_1^\flat(n)) \wedge e_1 \in clan_{EI^2}(tc_2^\flat(n))\} \bigcup$
  $\{(source_G^1(e_1), e_1, target_G^1(e_1)), n, (source_G^2(e_2), e_2, target_G^2(e_2)))|$
  $n \in V_G^C \wedge tc_i^\flat(n) \in E_G^i \wedge e_i \in clan_{EI^i}(tc_i^\flat(n))(for \; i = \{1, 2\})\} \bigcup$
  $\{(n_1, n, \cdot) | n \in V_G^C \wedge n_1 \in clan_{VI^1}(tc_1^\flat(n)) \wedge tc_2^\flat(n) = \cdot\} \bigcup$
  $\{(\cdot, n, n_1) | n \in V_G^C \wedge n_1 \in clan_{VI^2}(tc_2^\flat(n)) \wedge tc_1^\flat(n) = \cdot\} \bigcup$
  $\{(\cdot, n, \cdot) | n \in V_G^C \wedge tc_i^\flat(n) = \cdot \; (for \; i = \{1, 2\})\} \bigcup$
  $\{((source_G^1(e_1), e_1, target_G^1(e_1)), n, \cdot) | n \in V_G^C \wedge tc_1^\flat(n) \in E_G^1 \wedge$
  $tc_2^\flat(n) = \cdot \wedge e_1 \in clan_{EI^1}(tc_1^\flat(n))\} \bigcup$
  $\{(\cdot, n, (source_G^2(e_1), e_1, target_G^2(e_1))) | n \in V_G^C \wedge tc_2^\flat(n) \in E_G^2 \wedge$
  $tc_1^\flat(n) = \cdot \wedge e_1 \in clan_{EI^2}(tc_2^\flat(n))\}$.

- $\overline{E_G^C} = \{((x, n_1, y), e, (x', n_2, y')) | e \in E_G^C \wedge$
  $n_1 \in clan_{VI^C}(source_G^C(e)) \wedge$
  $n_2 \in clan_{VI^C}(target_G^C(e)) \wedge$
  $(x, n_1, y) \in \overline{V_G^C} \wedge$
  $(x', n_2, y') \in \overline{V_G^C} \wedge$
  $[\nexists e' \in clan_{EI^C}(e) - \{e\} \; with \; n_1 \in clan_{VI^C}(source_G^C(e')) \wedge$
  $n_2 \in clan_{VI^C}(target_G^i(e'))]\}$.

- $\overline{source_G^C}((x, e, y)) = x \in \overline{V_G^C}$.

[1] with 2^X being the powerset of X. However, note that as we do not have multiple inheritance in the correspondence graph, the result of *nearest_anc* is a set of at most one element.

- $\overline{target_G^C}((x,e,y)) = y \in \overline{V_G^C}$.
- $\overline{E_{NA}^C} = \{((x,n_1,y),e,n_2) | e \in E_{NA}^C \wedge n_1 \in clan_{VI^C}(source_{NA}^C(e)) \wedge (x,n_1,y) \in \overline{V_G^C} \wedge n_2 = target_{NA}^C(e))\}$.
- $\overline{source_{NA}^C}((n_1,e,n_2)) = n_1 \in \overline{V_G^C}$.
- $\overline{target_{NA}^C}((n_1,e,n_2)) = n_2 \in V_D^C$.
- $\overline{E_{EA}^C} = \{((n_1,e_1,n_2),e,n_3) | e \in E_{EA}^C \wedge e_1 \in clan_{EI^C}(source_{EA}^C(e)) \wedge ((n_1,e_1,n_2),e,n_3) \in \overline{E_G^C} \wedge [\nexists e' \in clan_{EI^C}(source_{EA}^C(e)) - \{source_{EA}^C(e)\} \text{ with } n_1 \in clan_{VI^C}(source_G^C(e')) \wedge n_2 \in clan_{VI^C}(target_G^C(e'))] \wedge n_3 = target_{EA}^C(e) \in V_D^C\}$.
- $\overline{source_{EA}^C}((x,e,y)) = x \in \overline{E_G^C}$.
- $\overline{target_{EA}^C}((x,e,y)) = y \in V_D^C$.
- $\overline{tc_1}((x,n,y)) = x$, *with* $(x,n,y) \in \overline{V_G^C}$.
- $\overline{tc_2}((x,n,y)) = y$, *with* $(x,n,y) \in \overline{V_G^C}$.

The meta-model triple $\widehat{TriATG} = (\widehat{TriTG}, Z)$ *with* $\widehat{TriTG} = (\widehat{TG_1}, \widehat{TG_2}, \widehat{TG_C}, \widehat{tc_1}, \widehat{tc_2})$, *where* $\widehat{TG_i} = \overline{TG_i}|_{\overline{V_G^i} - AV_i, \overline{E_G^i} - \{(x,e,y) | e \in AE_i\}} \subseteq \overline{TG^i}$ *(i=1,2) and* $\widehat{TG_C} = \overline{TG_C}|_{\overline{V_G^C} - \{(x_1,n,x_2) | n \in AV_C \vee x_j \in (AV_j \cup AE_j)_{j=1,2}\}, \overline{E_G^C} - \{(x,e,y) | e \in AE_C\}} \subseteq \overline{TG^C}$, *is called the concrete closure of* $TriATGI$ *because all abstract nodes and edges are removed:* $\widehat{TG} = \overline{TG_i}|_{A,B}$ *is the restriction of triple graph* $\overline{TG_i}$ *to sets* A *and* B *for graph nodes and edges.*

Figure 35 shows another example of the closure of the source graph of a meta-model triple, which illustrates how the edges are explicitly copied. In the example, edge *rel* is not copied between nodes A and C because there is another edge (rel') between these nodes, which refines *rel* through an inheritance relation. Moreover, edge *rel* is not copied between nodes A and D, because relation rel' is inherited first.

Figure 36 shows the concrete closure of the meta-model triple depicted in the upper part of Figure 34. In the source and target graphs we have eliminated the abstract nodes and edges. In the correspondence graph we have eliminated the abstract nodes and edges, as well as nodes having a correspondence function leading to an abstract node or edge.

Note that we have $TriTG \subseteq \overline{TriTG}$ with $TG_{V_j}^i \subseteq \overline{TG_{V_j}^i}$ for $j \in \{G, D\}$ and $TG_{E_j}^i \subseteq \overline{TG_{E_j}^i}$ if we identify $e \in TG_{E_j}^i$ with $(source_j^i(e), e, target_j^i(e)) \in \overline{TG_{E_j}^i}$

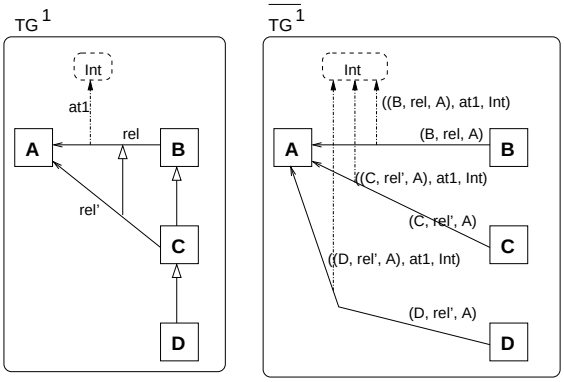


Figure 35: Closure example.

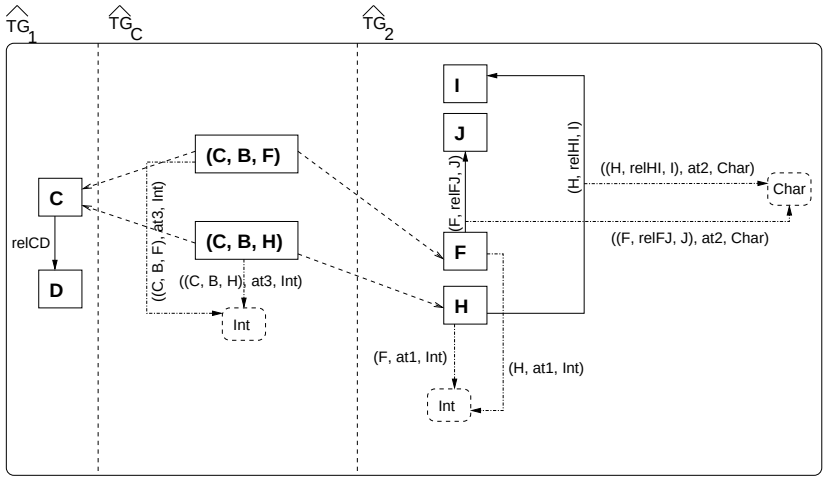


Figure 36: Concrete closure example.

for $j \in \{G, NA, EA\}$ and $n \in TG^C_{V_G}$ with $(tc^\flat_1(n), n, tc^\flat_2(n))$[2]. In Figure 34, in the correspondence graph, we identify node B in the original meta-model triple with node (C, B, E) in the flattened meta-model triple, as $tc^\flat_1(B) = C$ and $tc^\flat_2(B) = E$.

Next, we show that all graphs typed over $TriATG$ are also typed over $\overline{TriATG}$. This can be easily done for the source and target graphs, due the canonical inclusion $TG^i \subseteq \overline{TG^i}$ for $i = \{1, 2\}$. For the correspondence graphs,

[2]more precisely, we identify n with $(tc^\flat_1(n), n, tc^\flat_2(n))$ in case of $tc^\flat_i(n) \in TC^i_{V_G} \cup \{\cdot\}$; otherwise, with $(tc^\flat_1(n), n, (source^2_G(tc^\flat_2(n)), tc^\flat_2(n), target^2_G(tc^\flat_2(n))))$ or $((source^1_G(tc^\flat_1(n)), tc^\flat_1(n), target^1_G(tc^\flat_1(n))), n, tc^\flat_2(n))$ or $((source^1_G(tc^\flat_1(n)), tc^\flat_1(n), target^1_G(tc^\flat_1(n))), n, (source^2_G(tc^\flat_2(n)), tc^\flat_2(n), target^2_G(tc^\flat_2(n))))$ in case of $tc^\flat_2(n) \in TG^2_{E_G}$; $tc^\flat_1(n) \in TG^1_{E_G}$ or $tc^\flat_1(n) \in TG^1_{E_G}$ and $tc^\flat_2(n) \in TG^2_{E_G}$ respectively.

the appropriate node type should be selected taking into account source and target correspondence functions.

Fact 41 *(Typing with Respect to the Closure of Meta-Model Triple)*
Given a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$ *with* $TriATG = (TriTG, Z)$ *and* $TriTG = (TG_1, TG_2, TG_C, tc_1, tc_2)$ *as before; and the abstract closure of* $TriATGI$, $\overline{TriATG} = (\overline{TriTG}, Z)$ *with* $\overline{TriTG} = (\overline{TG_1}, \overline{TG_2}, \overline{TG_C}, \overline{tc_1}, \overline{tc_2})$ *with* $\overline{TG_i} = (\overline{V_G^{TG,i}}, V_D^{TG,i}, \overline{E_G^{TG,i}}, \overline{E_{NA}^{TG,i}}, \overline{E_{EA}^{TG,i}}, (\overline{source_j^{TG,i}}, \overline{target_j^{TG,i}})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1,2,C\}$, *built as shown before; and given a triple graph* $G = ((G_1, G_2, G_C, c_1, c_2), D)$ *with* $G_i = (V_G^{G,i}, V_D^{G,i}, E_G^{G,i}, E_{NA}^{G,i}, E_{EA}^{G,i}, source_j^{G,i}, target_j^{G,i})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1,2,C\}$, *and a typing* $type^G\colon G \to TriTG$, *it is possible to build a typing* $\overline{type}\colon G \to \overline{TriATG}$ *to the closure* $\overline{TriATG}$ *of the meta-model triple as follows:*

- $\overline{type_{V_D^{G,i}}} = type_{V_D^{G,i}}$, *for* $i = \{1,2,C\}$.
- $\overline{type_{V_G^{G,i}}} = type_{V_G^{G,i}}$, *for* $i = \{1,2\}$.
- $\overline{type_{E_j^{G,i}}}(e) = (source_j^{TG,i}(type_{E_j^{G,i}}^{G,i}(e)), type_{E_j^{G,i}}(e), target_j^{TG,i}(type_{E_j^{G,i}}(e)))$, *for* $i = \{1,2,C\}$, $j = \{G, NA, EA\}$.
- *The typing of the nodes in the correspondence graph is defined as follows:*

$$\overline{type_{V_G^{G,C}}}(n) = \begin{cases} (\overline{type_{X^{G,1}}}(c_1(n)), \overline{type_{V_G^{G,C}}}(n), \overline{type_{X^{G,2}}}(c_2(n))) \\ \qquad if\ c_i(n) \in X^{G,i}\ for X \in \{V_G, E_G\}, i \in \{1,2\} \\ (\cdot, \overline{type_{V_G^{G,C}}}(n), \overline{type_{X^{G,2}}}(c_2(n))) \\ \qquad if\ c_1(n) = \cdot \wedge c_2(n) \in X^{G,2}\ for X \in \{V_G, E_G\} \\ (\overline{type_{X^{G,1}}}(c_1(n)), \overline{type_{V_G^{G,C}}}(n), \cdot) \\ \qquad if\ c_2(n) = \cdot \wedge c_1(n) \in X^{G,1}\ for X \in \{V_G, E_G\} \\ (\cdot, \overline{type_{V_G^{G,C}}}(n), \cdot)\ if\ c_i(n) = \cdot\ for\ i = \{1,2\} \end{cases}$$

Figure 37 shows an example of typing of a triple graph by the concrete closure of the meta-model triple shown in Figure 36. Nodes are labelled with their type, except the ones of the correspondence graph. The upper one has type (C, B, H) and the lower one has type (C, B, F). Thus, in a first step we define instances of meta-model triples as triple graphs typed by its concrete closure. Therefore, the triple graph in Figure 37 is an instance of the meta-model triple shown in the upper part of Figure 34. We also say that a triple graph *is conformant to* a meta-model triple. We distinguish abstract and concrete instances of a meta-model triple; the latter cannot have nodes or edges abstractly typed.

Definition 42 *(Instance of Meta-Model Triple)*
An abstract instance $TriTAG^A$ *of* $TriATGI$ *is an attributed typed triple graph over* $\overline{TriATG}$, *i.e.* $TriTAG^A = (TriAG, type\colon TriAG \to \overline{TriATG})$. *Similarly, a concrete instance* $TriTAG^C$ *of* $TriATGI$ *is an attributed typed triple graph over* $\widehat{TriATG}$, *i.e.* $TriTAG^C = (TriAG, type\colon TriAG \to \widehat{TriATG})$.

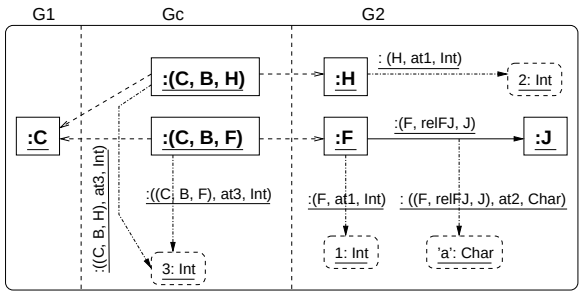


Figure 37: Example of typing.

Note that the previous typing relations were defined between triple graphs and flattened triple type graphs. As we want to avoid explicitly flattening the meta-model triple, our objective is to give the typing directly to the type graph with inheritance. This is interesting also for the implementation, as flattening a meta-model triple is undesirable for efficiency reasons. In order to define the typing directly from a triple graph to its meta-model triple, we introduce triple clan morphisms.

Definition 43 *(Triple Clan Morphism)*

Given a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$ *where* $TriATG = (TriTG, Z)$ *with* $TriTG = (TG_1, TG_2, TG_C, tc_1, tc_2)$ *and* $TG_i = (V_G^{TG,i}, V_D^{TG,i}, E_G^{TG,i}, E_{NA}^{TG,i}, E_{EA}^{TG,i}, (source_j^{TG,i}, target_j^{TG,i})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1,2,C\}$*, and given an attributed triple graph* $TriAG = (TriG, D)$ *with* $TriG = (G_1, G_2, G_C, c_1, c_2)$ *and* $G_i = (V_G^{G,i}, V_D^{G,i}, E_G^{G,i}, E_{NA}^{G,i}, E_{EA}^{G,i}, (source_j^{G,i}, target_j^{G,i})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1,2,C\}$*, the family of functions* $type^j : TriAG \to TriATGI$ *(for* $j = \{1,2,C\}$*) with* $type^j = (type_i^j, type_D)_{i\in\{V_G,V_D,E_G,E_{NA},E_{EA}\}}$ *and*

- $type_{V_i}^j : V_i^{G,j} \to V_i^{TG,j}$ *(for* $i \in \{G,D\}$*,* $j \in \{1,2,C\}$*).*
- $type_{E_i}^j : E_i^{G,j} \to E_i^{TG,j}$ *(for* $i \in \{G,NA,EA\}$*,* $j \in \{1,2,C\}$*).*
- $type_D : D \to Z$ *unique final DSIG-homomorphism.*

is called a triple clan morphism*, if:*

1. $\forall s \in S_D^{'}$ *the following diagram commutes,*

$$\begin{array}{ccc} D_{j,s} & \xrightarrow{type_{D_s}^j} & Z_{j,s} = \{s\} \\ \downarrow & = & \downarrow \\ V_D^{G,j} & \xrightarrow[type_{V_D}^j]{} & V_D^{TG,j} = S_D^{'} \end{array}$$

i.e. $type_{V_D}^j(d) = s$ *for* $d \in D_s$ *and* $s \in S_D^{'}$ *and* $j \in \{1,2,C\}$*.*

2. $type^j_{V_G} \circ source^{G,j}_G(e_1) \in clan_{VI^j}(source^{TG,j}_G \circ type^j_{E_G}(e_1)) \quad \forall e_1 \in V^{G,j}_{E_G}$.

3. $type^j_{V_G} \circ target^{G,j}_G(e_1) \in clan_{VI^j}(target^{TG,j}_G \circ type^j_{E_G}(e_1)) \quad \forall e_1 \in V^{G,j}_{E_G}$.

4. $type^j_{V_G} \circ source^{G,j}_{NA}(e_2) \in clan_{VI^j}(source^{TG,j}_{NA} \circ type^j_{E_{NA}}(e_2)) \quad \forall e_2 \in E^{G,j}_{NA}$.

5. $type^j_{V_D} \circ target^{G,j}_{NA}(e_2) = target^{TG,j}_{NA} \circ type^j_{E_{NA}}(e_2) \quad \forall e_2 \in E^{G,j}_{NA}$.

6. $type^j_{E_G} \circ source^{G,j}_{EA}(e_3) \in clan_{EI^j}(source^{TG,j}_{EA} \circ type^j_{E_{EA}}(e_3)) \quad \forall e_3 \in E^{G,j}_{EA}$.

7. $type^j_{V_D} \circ target^{G,j}_{EA}(e_3) = target^{TG,j}_{EA} \circ type^j_{E_{EA}}(e_3) \quad \forall e_3 \in E^{G,j}_{EA}$.

8. $type^i_{V_G} \circ c_i(x) \in clan_{VI^i}(tc^\flat_i \circ type^C_{V_G}(x))$, $\forall x \in nodes_i$ *for* $i = 1, 2$.

9. $type^i_{E_G} \circ c_i(x) \in clan_{EI^i}(tc^\flat_i \circ type^C_{V_G}(x))$, $\forall x \in edges_i$ *for* $i = 1, 2$.

10. $c_i(x) = \cdot = tc^\flat_i \circ type^C_{V_G}(x)$, $\forall x \in undef_i$ *for* $i = 1, 2$.

Where $tc^\flat_i$ *is defined as in definition 40:*

$$tc^\flat_i(x) = \begin{cases} tc_i(x) \; if \; tc_i(x) \neq \cdot \vee nearest\_anc_i(x) = \{\} \\ tc_i(y) \; if \; tc_i(x) = \cdot \wedge y \in nearest\_anc_i(x) \end{cases}$$

A triple clan morphism type: $TriAG \rightarrow TriATG$ *is called* concrete triple clan morphism *if* $type^j_{V_G}(n) \notin AV_j$ *for all* $n \in V^{G,j}_G$ *and* $type^j_{E_G}(e) \notin AE_j$ *for all* $e \in E^{G,j}_G$.

Conditions 2 and 3 in the previous definition mean that edges are inherited from a node to its node inheritance clan. Conditions 4 and 5 model the fact that attributes are also inherited from a node to its node inheritance clan. With conditions 6 and 7 we make attributes to be inherited from an edge to its edge inheritance clan. Conditions 8 and 9 establish that the target of a correspondence function can lead to any subtype (node or edge) of the target element in the type graph. In addition, these conditions take into account that the correspondence function can be "inherited" through the node inheritance clan in the correspondence graph. Finally, condition 10 specifies the fact that if a correspondence function is undefined, it should also be undefined in the type graph.

Figure 38 shows a simple example of triple clan morphism. It illustrates how the correspondence functions are inherited. In the meta-model triple in the upper side, node B, defines tc_2 leaving tc_1 undefined. That means that tc_1 will be inherited from A, and this is modelled by $tc^\flat_1$. Moreover, as nodes F and H in the target graph of the meta-model triple inherit from node E, then nodes of type B can be connected to nodes of types E, F or H, and this is modelled by using the $clan_{VI^i}$ function in condition 8. A similar situation occurs if the target of a correspondence function is an edge. This is shown in the example with the instance of node A, whose tc_2 function is connected to a $relFJ$ association, which is a subtype of rel. Note how the example triple

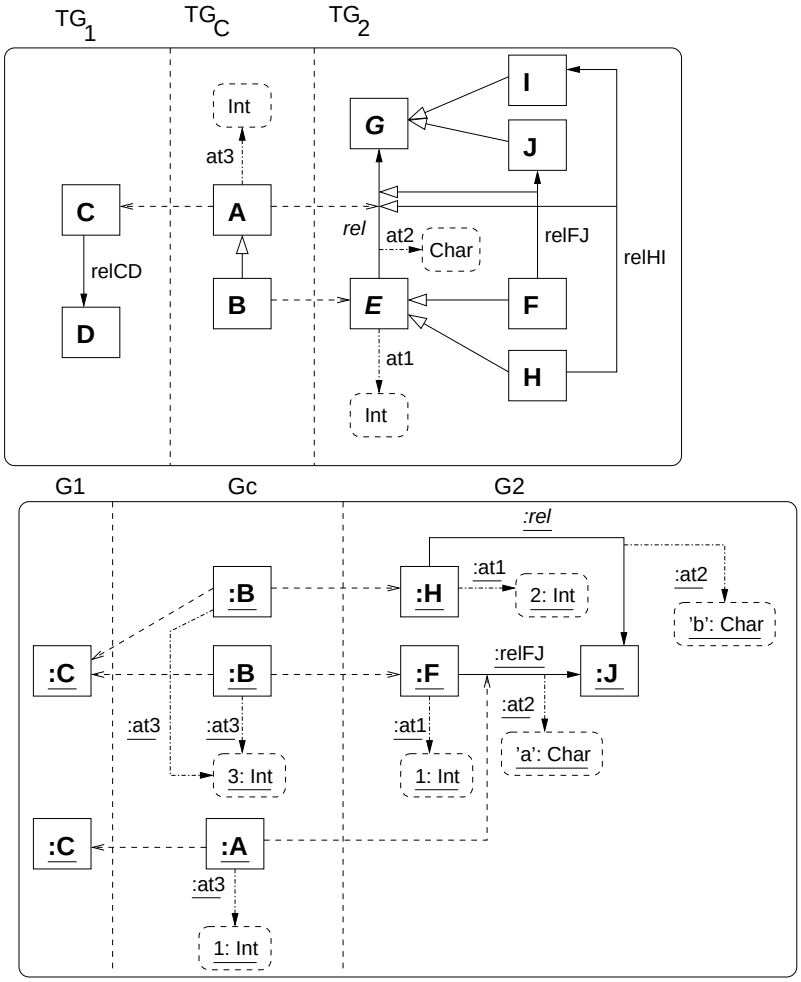


Figure 38: An example of triple clan morphism.

graph has an abstract typing, because there is an abstract edge with type *rel* between the instances of H and J.

The following technical properties of triple clan morphisms are needed to show the results in next subsection based on Double Pushout Transformation in the category **TriAGraph**. In order to show the bijective correspondence between triple clan morphisms and normal typing triple morphisms $\overline{type}: TriAG \rightarrow \overline{TriATG}$ we first define a universal triple clan morphism, which uniquely maps the flattened type graph and the meta-model triple.

Definition 44 *(Universal Triple Clan Morphism)*
Given a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$, *the universal triple clan morphism* $u_{TriATG}: \overline{TriATG} \rightarrow TriATGI$ *with* $\overline{TriATG} =$

$(\overline{TriTG}, Z)$ *with* $\overline{TriTG} = (\overline{TG_1}, \overline{TG_2}, \overline{TG_C}, \overline{tc_1}, \overline{tc_2})$ *and* $\overline{TG_i} = (\overline{V_G^{TG,i}}, \overline{V_D^{TG,i}} = V_D^{TG,i}, \overline{E_G^{TG,i}}, \overline{E_{NA}^{TG,i}}, \overline{E_{EA}^{TG,i}}, (\overline{source_j^{TG,i}}, \overline{target_j^{TG,i}})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1, 2, C\}$ *is defined by:*

- $u^i_{TriATG,V_G} = id^i_{V_G} : \overline{V_G^{TG,i}} \to V_G^{TG,i}$, *for* $i \in \{1, 2\}$.
- $u^C_{TriATG,V_G} : \overline{V_G^{TG,C}} \to V_G^{TG,C}, u^C_{TriATG,V_G}((x, n, y)) = n \in V_G^{TG,C}$.
- $u^i_{TriATG,V_D} = id^i_{V_D} : \overline{V_D^{TG,i}} \to V_D^{TG,i}$, *for* $i \in \{1, 2, C\}$.
- $u^i_{TriATG,E_G} : \overline{E_G^{TG,i}} \to E_G^{TG,i}$, $u^i_{TriATG,E_G}((x, e, y)) = e \in E_G^{TG,i}$, *for* $i \in \{1, 2, C\}$.
- $u^i_{TriATG,E_{NA}} : \overline{E_{NA}^{TG,i}} \to E_{NA}^{TG,i}$, $u^i_{TriATG,E_{NA}}((x, e, y)) = e \in E_{NA}^{TG,i}$, *for* $i \in \{1, 2, C\}$.
- $u^i_{TriATG,E_{EA}} : \overline{E_{EA}^{TG,i}} \to E_{EA}^{TG,i}$, $u^i_{TriATG,E_{EA}}((x, e, y)) = e \in E_{EA}^{TG,i}$, *for* $i \in \{1, 2, C\}$.
- $u_{TriATG,D} = id_Z : Z \to Z$.

As an example of the universal triple clan morphism for Figure 34, we have that $u^C_{TriATG,V_G}((C, B, E)) = B = u^C_{TriATG,V_G}((C, B, F)) = u^C_{TriATG,V_G}((C, B, H))$, and also that $u^C_{TriATG,V_G}((C, A, \cdot)) = A$. For the graph nodes in $\overline{TG_1}$ and $\overline{TG_2}$, the universal clan morphism is the identity. For the edges, the typing is obtained from the second component of the label. In this way, for example, $u^C_{TriATG,E_{NA}}((C, B, H), at3, Int)) = at3$. This universal morphism is indeed a clan morphism, as next lemma states.

Lemma 45 *The universal triple clan morphism* $u_{TriATG} \colon \overline{TriATG} \to TriATGI$ *is a triple clan morphism. Triple clan morphisms are closed under composition with attributed triple graph morphisms. This means that given an attributed triple morphism* $f \colon TriAG' \to TriAG$ *and a triple clan morphism* $f' \colon TriAG \to TriATGI$ *then* $f' \circ f \colon TriAG' \to TriATGI$ *is a triple clan morphism. If* f' *is concrete, so is* $f' \circ f$.

The following theorem relates triple clan morphisms and attributed triple morphisms, and is essential to show the main results in the next subsection.

Theorem 46 *(Universal Triple Clan Property)*
For each triple clan morphism $type \colon TriAG \to TriATGI$, *there is a unique attributed triple morphism* $\overline{type} \colon TriAG \to \overline{TriATG}$ *such that* $u_{TriATG} \circ \overline{type} = type$, *as Figure 39 shows.*

Construction.
Given the triple clan morphism $type \colon TriAG \to TriATGI$ with $TriAG = (TriG, D)$ and $TriG = (G_1, G_2, G_C, c_1, c_2)$, where $G_i = (V_G^{G,i}, V_D^{G,i}, E_G^{G,i}, E_{NA}^{G,i},$

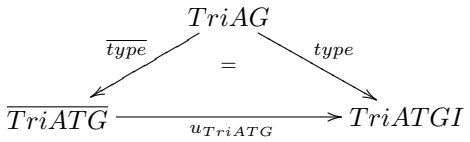


Figure 39: Uniqueness of attributed triple morphism.

$E_{EA}^{G,i}, (source_j^{G,i}, target_j^{G,i})_{j\in\{G,NA,EA\}})$ for $i \in \{1,2,C\}$, and $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$ where $TriATG = (TriTG, Z)$ with $TriTG = (TG_1, TG_2, TG_C, tc_1, tc_2)$ and $TG_i = (V_G^{TG,i}, V_D^{TG,i}, E_G^{TG,i}, E_{NA}^{TG,i}, E_{EA}^{TG,i}, (source_j^{TG,i}, target_j^{TG,i})_{j\in\{G,NA,EA\}})$ for $i \in \{1,2,C\}$, the attributed triple morphism $\overline{type}\colon TriAG \to \overline{TriATG}$ can be uniquely constructed as follows:

- $\overline{type_{V_G}^i} = type_{V_G}^i \colon V_G^{G,i} \to V_G^{TG,i} = \overline{V_G^{TG,i}}$, for $i \in \{1,2\}$.
- $\overline{type_{V_D}^i} = type_{V_D}^i \colon V_D^{G,i} \to V_D^{TG,i} = \overline{V_D^{TG,i}}$, for $i \in \{1,2,C\}$.
- $\overline{type_{E_G}^i} : E_G^{G,i} \to \overline{E_G^{TG,i}}$, $\overline{type_{E_G}^i}(e_1) = (n_1, e_1', n_2)$ with $e_1' = type_{E_G}^i(e_1) \in E_G^{TG,i}$, $n_1 = \overline{type_{V_G}^i}(source_G^{G,i}(e_1)) \in V_G^{TG,i}$, $n_2 = \overline{type_{V_G}^i}(target_G^{G,i}(e_1)) \in V_G^{TG,i}$, for $i \in \{1,2,C\}$.
- $\overline{type_{E_{NA}}^i} : E_{NA}^{G,i} \to \overline{E_{NA}^{TG,i}}$, $\overline{type_{E_{NA}}^i}(e_2) = (n_1, e_2', n_2)$ with $e_2' = type_{E_{NA}}^i(e_2) \in E_{NA}^{TG,i}$, $n_1 = \overline{type_{V_G}^i}(source_{NA}^{G,i}(e_2)) \in V_G^{TG,i}$, $n_2 = \overline{type_{V_D}^i}(target_{NA}^{G,i}(e_2)) \in V_D^{TG,i}$, for $i \in \{1,2,C\}$.
- $\overline{type_{E_{EA}}^i} : E_{EA}^{G,i} \to \overline{E_{EA}^{TG,i}}$, $\overline{type_{E_{EA}}^i}(e_3) = ((n_{11}, e_3'', n_{12}), e_3', n_2)$ with $e_3' = type_{E_{EA}}^i(e_3) \in E_{EA}^{TG,i}$, $(n_{11}, e_3'', n_{12}) = \overline{type_{E_G}^i}(source_{EA}^{G,i}(e_3)) \in \overline{E_G^{TG,i}}$, $n_2 = \overline{type_{V_D}^i}(target_{EA}^{G,i}(e_3)) \in V_D^{TG,i}$, for $i \in \{1,2,C\}$.
- $\overline{type_D} = type_D : D \to Z$.
- The typing of the nodes in the correspondence graph is defined as follows:

$$\overline{type_{V_G^{G,C}}}(n) = \begin{cases} (\overline{type_{X^{G,1}}}(c_1(n)), \overline{type_{V_G^{G,C}}}(n), \overline{type_{X^{G,2}}}(c_2(n))) \\ \qquad if\ c_i(n) \in X^{G,i} for X \in \{V_G, E_G\}, i \in \{1,2\} \\ (\cdot, \overline{type_{V_G^{G,C}}}(n), \overline{type_{X^{G,2}}}(c_2(n))) \\ \qquad if\ c_1(n) = \cdot \wedge c_2(n) \in X^{G,2}\ for X \in \{V_G, E_G\} \\ (\overline{type_{X^{G,1}}}(c_1(n)), \overline{type_{V_G^{G,C}}}(n), \cdot) \\ \qquad if\ c_2(n) = \cdot \wedge c_1(n) \in X^{G,1}\ for X \in \{V_G, E_G\} \\ (\cdot, \overline{type_{V_G^{G,C}}}(n), \cdot)\ if\ c_i(n) = \cdot\ for\ i = \{1,2\} \end{cases}$$

By lemma 45, we have that the composition $u_{TriATG} \circ \overline{type}$ is a triple clan morphism.

Lemma 47 *(Pushout Property of Triple Clan Morphisms)*

1. *A pushout in* **TriAGraph** *is also a pushout with respect to (concrete) triple clan morphisms (see Figure 40).*
   Given a pushout PO in **TriAGraph** *as shown in Figure 40 with attributed triple morphisms g_1, g_2, g'_1, g'_2 and triple clan morphisms f_1, f_2 with $f_1 \circ g_1 = f_2 \circ g_2$, then there is a unique triple clan morphism $f : G_3 \to TriATGI$ with $f \circ g'_1 = f_1$ and $f \circ g'_2 = f_2$.*

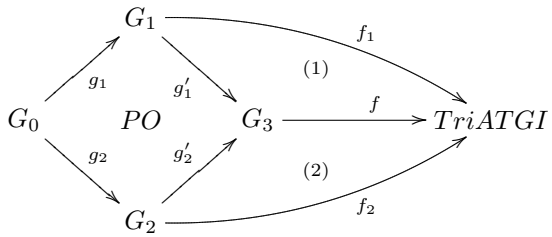


Figure 40: Pushout w.r.t. concrete triple clan morphisms.

2. *Double pushouts in* **TriAGraph** *can be extended to double pushouts for attributed triple graphs with typing by concrete triple clan morphisms with respect to the match morphism and the triple rule (see Figure 41).*
   Given pushouts (1) and (2) in **TriAGraph** *as shown in Figure 41 and concrete triple clan morphisms $type_L$, $type_K$, $type_R$, and $type_G$ for the triple rule and the host triple graph G such that (3), (4) and (5) commute, then there are also unique concrete triple clan morphisms $type_D$ and $type_H$ such that (6) and (7) commute.*

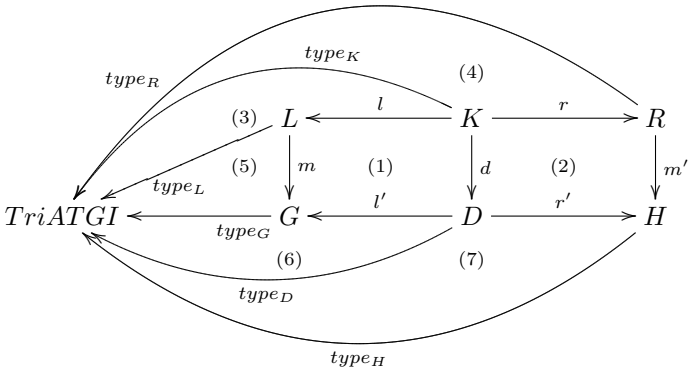


Figure 41: Double pushout for attributed triple graphs with typing by concrete triple clan morphism.

7.2 Attributed Typed Triple Graph Transformation with Inheritance

The goal of this subsection is to extend triple rules with the inheritance concept. We call these rules inheritance-extended triple rules, or IE-triple rules. In this way, nodes and edges in an IE-triple rule can be typed by node and edge types (also called classes and associations) in the meta-model triple, which may be refined by a number of sub-classes and sub-associations. An IE-triple rule typed in that way is equivalent to a set of *concrete* IE-triple rules, resulting by the valid substitutions of each node and edge in the IE-triple rule by all the concretely typed nodes and edges in their inheritance clans. If the set of equivalent rules of an IE-triple rule has a cardinality greater than one, the IE-triple rule is called *IE-triple meta-rule*. As these rules are equivalent to more than one concrete rule, they give as a result grammars that are more compact. Nodes and edges abstractly typed are thus allowed to appear in the LHS of an IE-triple rule. However, if an abstract node appears in the RHS, then it must also appear in the LHS. That is, we do not allow triple rules to create elements with an abstract typing. This could be done in principle, and then the meta-rule would be equivalent to a number of concrete rules resulting from the valid substitutions of the elements with an abstract type by elements with a concrete one. However, this could result in non-determinism when applying the meta-rule, which we want to avoid. We first define type refinement and then define IE-triple rules.

Definition 48 *(Type Refinement)*
Given an attributed triple graph $TriAG = (TriG, D)$ *with* $TriG = (G_1, G_2, G_C, c_1, c_2)$ *and* $G_i = (V_G^{G,i}, V_D^{G,i}, E_G^{G,i}, E_{NA}^{G,i}, E_{EA}^{G,i}, (source_j^{G,i}, target_j^{G,i})_{j\in\{G,NA,EA\}})$ *for* $i \in \{1, 2, C\}$*, and two clan morphisms* $type\colon TriAG \to TriATGI$ *and* $type'\colon TriAG \to TriATGI$*, type′ is called a type refinement of type, written* $type' \leq type$[3] *if:*

- $type'^{i}_{V_G}(n) \in clan_{VI^i}(type^i_{V_G}(n)), \quad \forall n \in V_G^{G,i}$*, for* $i \in \{1, 2, C\}$.
- $type'^{i}_{E_G}(n) \in clan_{EI^i}(type^i_{E_G}(n)), \quad \forall n \in E_G^{G,i}$*, for* $i \in \{1, 2, C\}$.
- $type'^{i}_{X} = type^i_X, \quad$ *for* $X \in \{V_D, E_{NA}, E_{EA}\}, i \in \{1, 2, C\}$.
- $type'_D = type_D$.

Given clan morphisms $type, type' : TriAG \to TriATGI$ with $type' \leq type$ and an attributed triple morphism $g : TriAG' \to TriAG$, then also $type' \circ g \leq type \circ g$. Figure 42 shows an example of type refinement. Triple graph G' has a typing $type'$ that is finer than the typing $type$ of G. The reason is that in the source graph, $type'^{1}_{V_G}(:K) = K \in clan_{VI^1}(type^1_{V_G}(:C) = C)$. Besides, in the target graph we have that $type'^{2}_{V_G}(:F) = F \in clan_{VI^2}(type^2_{V_G}(:E) = E)$ and that $type'^{2}_{E_G}(:relFJ) = relFJ \in clan_{EI^2}(type^2_{E_G}(:rel) = rel)$.

[3] we say that $type'$ is finer than $type$.

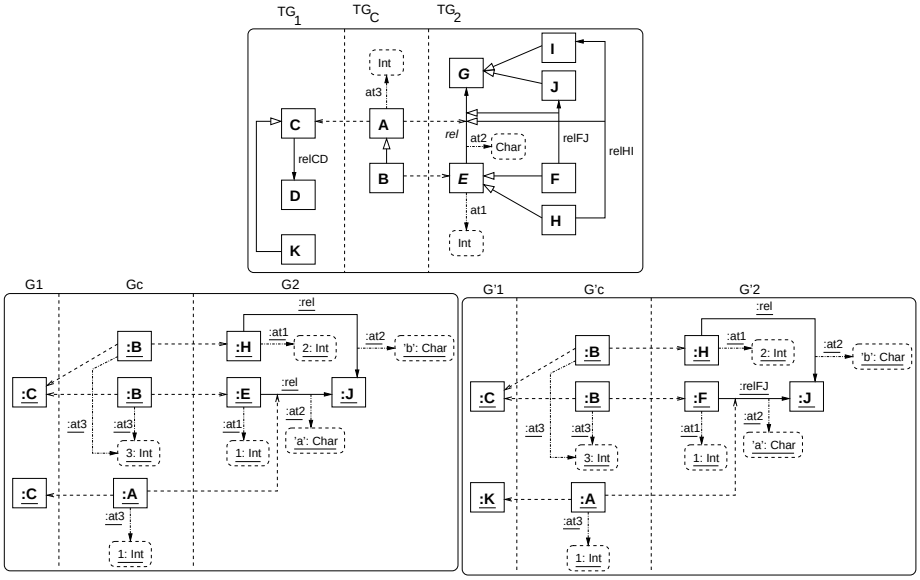


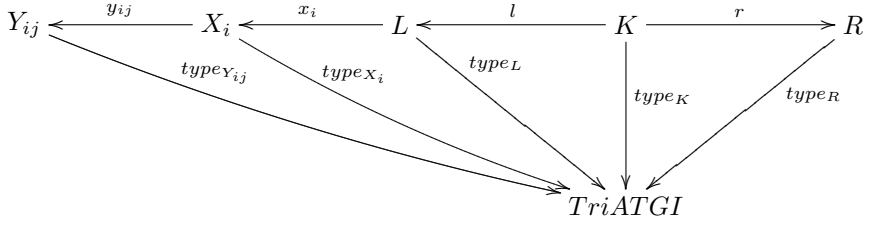
Figure 42: Type refinement example.

Definition 49 *(Meta- and Concrete IE-triple Rule)*

An inheritance-extended triple rule (short IE-triple rule) is a triple rule typed by a meta-model triple $TriATGI = (TriATG, (VI_i, EI_i, AV_i, AE_i)_{i\in\{1,2,C\}})$*, and is given by* $p = (L \xleftarrow{l} K \xrightarrow{r} R, type, AC)$*. The first element is an attributed triple graph rule (*l *and* r *are attributed triple morphisms); type =* $(type_i\colon i \to TriATGI)_{i\in\{L,K,R\}}$ *is a triple of typing triple clan morphisms, one for each part of the triple rule;* $AC = \{cc_i = (x_i\colon L \to X_i, type_{X_i}, A_i = \{(y_{ij}\colon X_i \to Y_{ij}, type_{Y_{ij}})\})\}$ *is a set of application conditions where* $type_{X_i}\colon X_i \to TriATGI$ *and* $type_{Y_{ij}}\colon Y_{ij} \to TriATGI$ *are triple clan morphisms, such that the following conditions hold:*

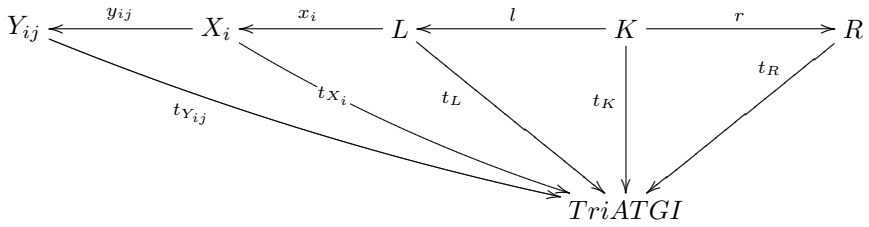
- $type_L \circ l = type_K = type_R \circ r$.
- $type^i_{R,V_G}(V_G'^{R,i}) \cap AV_i = \emptyset$, *where* $V_G'^{R,i} := V_G^{R,i} - r^i_{V_G}(V_G^{K,i})$, *for* $i \in \{1,2,C\}$.
- $type^i_{R,E_G}(E_G'^{R,i}) \cap AE_i = \emptyset$, *where* $E_G'^{R,i} := E_G^{R,i} - r^i_{E_G}(E_G^{K,i})$, *for* $i \in \{1,2,C\}$.
- $type_{Y_{ij}} \circ y_{ij} \leq type_{X_i}$ *and* $type_{X_i} \circ x_i \leq type_L$ *for all* $cc_i \in AC$.
- $type^i_{L,V_G} \circ c^L_i \circ l^C_{V_G^C}|_{nodes^K_i} = type^i_{K,V_G} \circ c^K_i|_{nodes^K_i} = type^i_{R,V_G} \circ c^R_i \circ r^C_{V_G^C}|_{nodes^K_i}$ *for* $i = 1,2$ *where* c^K_i, c^L_i *and* c^R_i *are the correspondence functions of* K, L *and* R.

- $type^i_{L,E_G} \circ c^L_i \circ l^C_{V^C_G}|_{edges^K_i} = type^i_{K,E_G} \circ c^K_i|_{edges^K_i} = type^i_{R,E_G} \circ c^R_i \circ r^C_{V^C_G}|_{edges^K_i}$ *for* $i = 1, 2$.
- $c^L_i \circ l^C_{V^C_G}|_{undef^K_i} = c^K_i|_{undef^K_i} = c^R_i \circ r^C_{V^C_G}|_{undef^K_i} = \cdot$ *for* $i = 1, 2$.
- *The datatype part of* L, K, R, X_i *and* Y_{ij} *is* $T_{DSIG}(X)$, *the term algebra of* $DSIG$ *with variables* X, *and* l, r, x_i *and* y_{ij} *are data preserving, i.e.* $l_{D_i}, r_{D_i}, x_{i_D}, y_{ij_D}$ *are identities*



A concrete IE-triple rule p_t *with respect to an IE-triple rule* p *is given by* $p_t = (L \xleftarrow{l} K \xrightarrow{r} R, t, \overline{AC})$, *where* t *is a triple of concrete typing clan morphisms* $t = (t_L : L \to TriATGI, t_K : K \to TriATGI, t_R : R \to TriATGI)$, *such that:*

- $t_L \circ l = t_K = t_R \circ r$
- $t_L \leq type_L$, $t_K \leq type_K$, $t_R \leq type_R$
- $t_{R,V_G}(x) = type_{R,V_G}(x) \quad \forall x \in V'^{R,i}_G$, *for* $i \in \{1, 2, C\}$.
- $t_{R,E_G}(e) = type_{R,E_G}(e) \quad \forall e \in E'^{R,i}_G$, *for* $i \in \{1, 2, C\}$.
- *For each* $(x_i \colon L \to X_i, type_{X_i}, A_i = \{(y_{ij} \colon X_i \to Y_{ij}, type_{Y_{ij}})\}) \in AC$, *we have all* $(x_i \colon L \to X_i, t_{X_i}, A_i = \{(y_{ij} \colon X_i \to Y_{ij}, t_{Y_{ij}})\}) \in \overline{AC}$ *with concrete triple clan morphisms* t_{X_i} *and* $t_{Y_{ij}}$ *satisfying* $t_{Y_{ij}} \circ y_{ij} = t_{X_i}$, $t_{X_i} \circ x_i = t_L$, $t_{Y_{ij}} \leq type_{Y_{ij}}$ *and* $t_{X_i} \leq type_{X_i}$.



The set of all concrete productions p_t *with respect to an IE-triple rule* p *is denoted by* $\widehat{p}$. p *is called IE triple meta-rule if* $|\widehat{p}| > 1$.

The top row of Figure 43 shows a simple IE-triple meta-rule example. The rule is typed with respect to the meta-model triple shown in Figure 33. The rule identifies the activator of a message, creating an edge in the abstract graph (the rule is simplified, we do not include application conditions for clarity). Note how nodes 7, 8 and 9 and edges 10 and 11 of the concrete graph have an abstract typing. The meta-rule is equivalent to four concrete rules. Node 7 can take types *StartPoint* or *ActivationBox* in the concrete rule, node 8 has to be an *ActivationBox*, and node 9 can be an *Object* or an *ActivationBox*. Thus, only four combinations are possible. The choice of the node types determines the edge types.

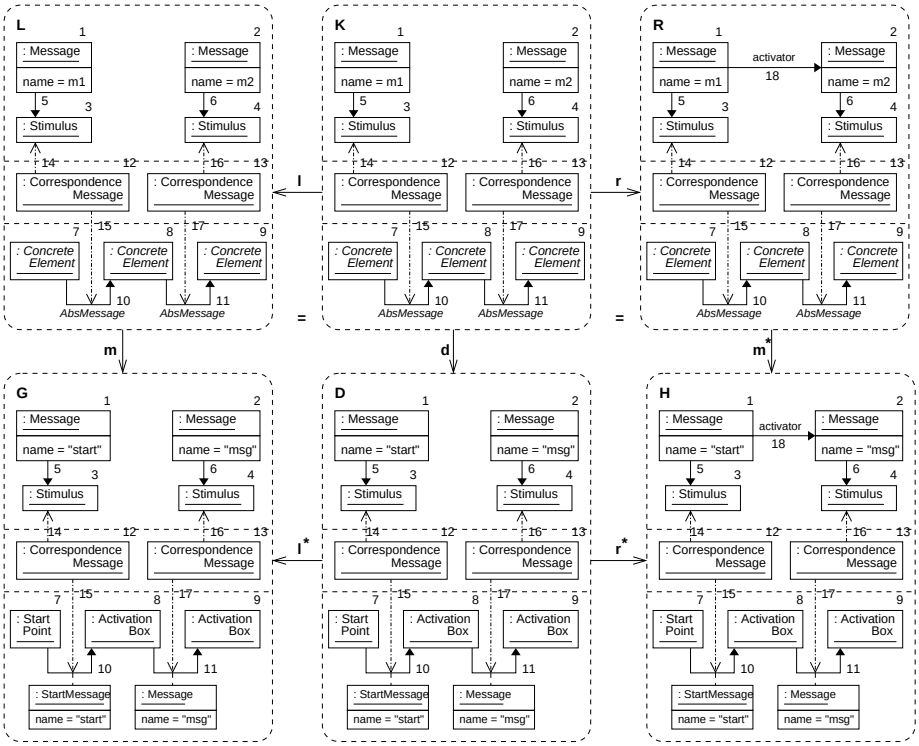


Figure 43: An example of IE-triple meta-rule and direct derivation.

In order to apply an IE-triple meta-rule to an ATT-graph, a structural match with respect to the untyped rule has to be found. The typing of the match should be concrete and finer than the type of the rule's LHS. Therefore, the typing of the target of the correspondence functions in the host graph should also be finer than in the rule's LHS. Finally, the match should satisfy the application conditions. The direct derivation can be built by first constructing the double pushout in **TriAGraph**, yielding the attributed triple graph H. Then, the typing is added. The preserved elements by the rule do not change their type. The new elements take their type from R, as the elements added by the rule

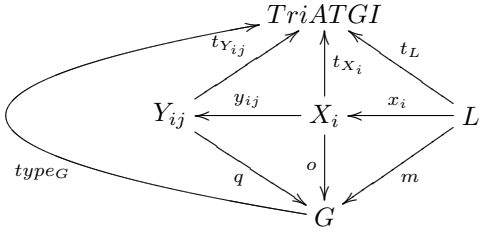
should have a concrete typing. Figure 43 shows a direct derivation example, where abstract elements 7, 8, 9, 10 and 11 in the rule take concrete types *StartPoint*, *ActivationBox*, *ActivationBox*, *StartMessage* and *Message*. Next, we first define how to apply concrete IE-triple rules to a triple graph and then show how to apply IE-triple meta-rules.

Definition 50 *(Direct Derivation by Concrete IE-Triple Rule)*

Let $p_t = (L \xleftarrow{l} K \xrightarrow{r} R, t, \overline{AC})$ *be a concrete IE-triple rule,* $(G, type_G)$ *an attributed typed triple graph with a concrete triple clan morphism* $type_G : G \to TriATGI$ *and* $m : L \to G$ *an attributed triple morphism. Morphism* m *is a* consistent match *with respect to* p_t *and* $(G, type_G)$*, if*

- m *satisfies the gluing condition (see definition 34) with respect to the untyped production* $L \xleftarrow{l} K \xrightarrow{r} R$ *and the attributed triple graph* G*,*
- $type_G \circ m = t_L$*, and*
- m *satisfies the application conditions* $\overline{AC}$*, i.e. for each* $(x_i \colon L \to X_i, t_{X_i}, A_i = \{(y_{ij} \colon X_i \to Y_{ij}, t_{Y_{ij}})\}) \in \overline{AC}$ *it holds, that :*
  - $\nexists o : X_i \to G$ *in* $\mathcal{M}$ *such that* $o \circ x_i = m$ *and* $type_G \circ o = t_{X_i}$ *or*
  - $\forall o : X_i \to G$ *in* $\mathcal{M}$ *such that* $o \circ x_i = m$ *and* $type_G \circ o = t_{X_i}$*;* $\exists q \colon Y_{ij} \to G$ *in* $\mathcal{M}$ *such that* $q \circ y_{ij} = o$ *and* $type_G \circ q = t_{Y_{ij}}$*.*

Given a consistent match m*, the concrete production can be applied to the typed attributed triple graph* $(G, type_G)$*, yielding a typed attributed triple graph* $(H, type_H)$ *by constructing the DPO of* l*,* r *and* m *and applying Lemma 47.2. We write* $(G, type_G) \overset{p_t, m}{\Longrightarrow} (H, type_H)$ *for such a* direct derivation*.*



In order to apply a meta-rule, one could calculate the set of equivalent concrete rules and then apply one of them. However, it is more efficient to define how to apply meta-rules directly. This is done in the following definition.

Definition 51 *(Direct Derivation by IE-Triple Meta-Rule)*

Let $p = (L \xleftarrow{l} K \xrightarrow{r} R, type, AC)$ *be a IE-triple meta-rule typed over an attributed type triple graph with inheritance* $TriATGI$*,* $(G, type_G)$ *an attributed typed triple graph with a concrete triple clan triple morphism* $type_G : G \to TriATGI$*, and* $m : L \to G$ *an attributed triple morphism. Morphism* m *is called consistent* match *with respect to* p *and* $(G, type_G)$*, if:*

- *m satisfies the gluing condition (see definition 34) with respect to the untyped production $L \xleftarrow{l} K \xrightarrow{r} R$ and the attributed triple graph G, i.e. pushout (1) in Figure 44 exists,*
- *$type_G \circ m \leq type_L$.*
- *$type^i_{G,V_G} \circ c^G_i \circ m_C|_{nodes^L_i} \leq type^i_{L,V_G} \circ c^L_i|_{nodes^L_i}$, for $i = 1, 2$.*
- *$type^i_{G,E_G} \circ c^G_i \circ m_C|_{edges^L_i} \leq type^i_{L,E_G} \circ c^L_i|_{edges^L_i}$, for $i = 1, 2$.*
- *$c^G_i \circ m_C|_{undef^L_i} = c^L_i|_{undef^L_i}$, for $i = 1, 2$.*
- *m satisfies the application conditions AC, i.e. for each $(x_i \colon L \to X_i, t_{X_i}, A_i = \{(y_{ij} \colon X_i \to Y_{ij}, t_{Y_{ij}})\}) \in AC$ it holds, that :*
  - *$\nexists o : X_i \to G$ in $\mathcal{M}$ such that $o \circ x_i = m$ and $type_G \circ o = t_{X_i}$ or*
  - *$\forall o : X_i \to G$ in $\mathcal{M}$ such that $o \circ x_i = m$ and $type_G \circ o = t_{X_i}$, and $\exists q \colon Y_{ij} \to G$ in $\mathcal{M}$ such that $q \circ y_{ij} = o$ and $type_G \circ q = t_{Y_{ij}}$.*

Given a consistent match m, the IE-triple meta-rule can be applied to $(G, type_G)$ yielding a direct meta-derivation *$(G, type_G) \overset{p,m}{\Longrightarrow} (H, type_H)$ with the concrete clan triple morphism $type_H$ as follows:*

1. *Construct the (untyped) DPO of l, r and m in* **TriAGraph** *given by pushouts (1) and (2) in Figure 44.*

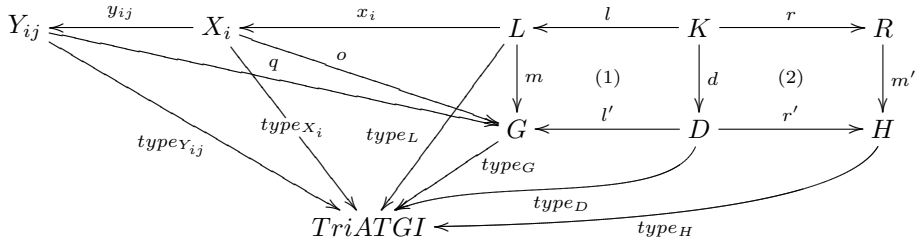


Figure 44: Direct derivation by IE-triple meta-rule.

2. *Construct $type_D$ and $type_H$ as follows*
   - *$type_D = type_G \circ l'$*
   - *$type^i_{H,X}(x) = \underline{if}\ \exists x' \in X^{D,i} \mid x = r'^i_X(x')\ \underline{then}\ type^i_{D,X}(x')\ \underline{else}\ type^i_{R,X}(x'')$, where $x'' \in X^{R,i}$ and $m'^i(x'') = x$ and $X \in \{V_G, V_D, E_G, E_{NA}, E_{EA}\}$, $i \in \{1, 2, C\}$.*
   - *$type^i_{H,D}(x) = \underline{if}\ \exists x' \in D^D \mid x = r'_D(x')\ \underline{then}\ type^{D,D}(x')\ \underline{else}\ type_{R,D}(x'')$, where $x'' \in D^R$ and $m'(x'') = x$.*

Next lemma states the relations between meta-derivations and concrete derivations. The idea is that for each application of a meta-rule, there is a unique concrete rule (in its equivalent set) that yields the same result. Moreover, starting from that concrete rule, one can reconstruct the typing of the meta-rule.

Lemma 52 *(Construction of Concrete and Meta-Derivations)*

Given an IE-triple meta-rule $p = (L \xleftarrow{l} K \xrightarrow{r} R, type, AC)$ *with* $AC = \{cc_i = (x_i\colon L \to X_i, type_{X_i}, A_i = \{(y_{ij}\colon X_i \to Y_{ij}, type_{Y_{ij}})_{j\in J}\})_{i\in I}\}$ *, a concrete attributed typed triple graph* $(G, type_G : G \to TriATGI)$ *and a consistent match morphism* $m : L \to G$ *with respect to* p *and* $(G, type_G)$*, we have:*

1. *There is a unique concrete IE-triple rule* $p_t \in \widehat{p}$ *with* $p_t = (L \xleftarrow{l} K \xrightarrow{r} R, t, \overline{AC})$ *and* $t_L = type_G \circ m$*. In this case,* t_K*,* t_R *and* $\overline{AC}$ *are defined by:*

   - $t_K = t_L \circ l$
   - $t^i_{R,V_G}(x) = \underline{if}\ \exists x' \in V^K_G\ x = r^i_{V_G}(x')\ \underline{then}\ t^i_{K,V_G}(x')\ \underline{else}\ type^i_{R,V_G}(x)$ *for* $x \in V^R_G$*,* $i = \{1, 2, C\}$*.*
   - $t^i_{R,E_G}(e) = \underline{if}\ \exists e' \in E^K_G\ e = r^i_{E_G}(e')\ \underline{then}\ t^i_{K,E_G}(e')\ \underline{else}\ type^i_{R,V_G}(e)$ *for* $e \in E^R_G$*,* $i = \{1, 2, C\}$*.*
   - $t^i_{R,X} = type^i_{R,X}$ *for* $X \in \{V_D, E_{NA}, E_{EA}, D\}$*,* $i = \{1, 2, C\}$*.*
   - $\overline{AC} = \bigcup_{i\in I}\{(x_i\colon L \to X_i, t_{X_i}, \bigcup_{j\in J}\{(y_{ij}\colon X_i \to Y_{ij}, t_{Y_{ij}})|t_{Y_{ij}}$ *is a concrete triple clan morphism with* $t_{Y_{ij}} \le type_{Y_{ij}}$ *and* $t_{Y_{ij}} \circ y_{ij} = t_{X_i}\})|t_{X_i}$ *is a concrete triple clan morphism with* $t_{X_i} \le type_{X_i}$ *and* $t_{X_i} \circ x_i = t_L\}$*.*

2. *There is a concrete direct derivation* $(G, type_G) \overset{p_t,m}{\Longrightarrow} (H, type_H)$ *with consistent match* m *w.r.t.* p_t*, and* $type_D = type_G \circ l'$ *and* $type_H$ *uniquely defined by* $type_D, t_R$ *and pushout properties of (2) (see Lemma 47), where* $type_H : H \to TriATGI$ *is a concrete triple clan morphism given by:*

   - $type^i_{H,X}(x) = \underline{if}\ \exists x' \in X^{D,i}$ *with* $x = r'^i_X(x')\ \underline{then}\ type^i_{D,X}(x')\ \underline{else}\ t^i_{R,X}(x'')$*, where* $x'' \in X^{R,i}$ *and* $m'^i(x'') = x$ *and* $X \in \{V_G, V_D, E_G, E_{NA}, E_{EA}\}$*,* $i \in \{1, 2, C\}$*.*
   - $type^i_{H,D}(x) = \underline{if}\ \exists x' \in D^D$ *with* $x = r'_D(x')\ \underline{then}\ type_{D,D}(x')\ \underline{else}\ t_{R,D}(x'')$*, where* $x'' \in D^R$ *and* $m'^i(x'') = x$*.*

3. *The concrete direct derivation becomes a direct meta-derivation (see Definition 51):*
   $(G, type_G) \overset{p,m}{\Longrightarrow} (H, type_H)$ *with* $type_D = type_H \circ r'$*,* $type_G \circ m \le type_L$*,* $type_D \circ d \le type_K$ *and* $type_H \circ m' \le type_R$*, where the typing* $t = (t_L, t_K, t_R)$ *of the concrete rule* p_t *is replaced by* $type = (type_L, type_K, type_R)$ *of the IE-triple meta-rule* p*.*

7.3 Equivalence of Concrete and Abstract Transformations

In this section we show that the transformations (a sequence of zero or more direct derivations) using meta-rules and the transformations using concrete rules are equivalent. This is stated in the following theorem. As the proofs of these theorems are similar to the ones presented in [de Lara *et al.*, 2005], we omit the proofs.

Theorem 53 *(Equivalence of Transformations)*

Given an abstract IE-triple rule $p = (L \xleftarrow{l} K \xrightarrow{r} R, type, AC)$ *over an attributed type triple graph with inheritance* $TriATGI$*, a concrete typed attributed triple graph* $(G, type_G)$ *and a match morphism* $m : L \to G$ *(which satisfies the gluing condition with respect to the untyped triple rule* $(L \longleftarrow K \longrightarrow R)$*. Then the following statements are equivalent, where* $(H, type_H)$ *is the same concrete typed graph in both cases:*

1. $m : L \to G$ *is a consistent match with respect to the abstract IE-triple rule* p *yielding an abstract direct derivation* $(G, type_G) \overset{p,m}{\Longrightarrow} (H, type_H)$.
2. $m : L \to G$ *is a consistent match with respect to the concrete IE-triple rule* $p_t = (L \leftarrow K \rightarrow R, t, \overline{AC})$ *with* $p_t \in \widehat{p}$ *and* $t_L = type_G \circ m$ *(where* t_K*,* t_R *and* $\overline{AC}$ *are uniquely defined by Lemma 52.1) yielding a concrete direct derivation* $(G, type_G) \overset{p_t,m}{\Longrightarrow} (H, type_H)$.

In a similar way as for regular graphs, we give a definition of graph grammar and language for IE-triple meta-rules.

Definition 54 *(Graph Grammar and Language for IE-Triple Meta-Rules)*

Given an attributed type triple graph with inheritance $TriATGI$ *and an attributed triple graph* G *typed over* $TriATGI$ *with a concrete triple clan morphism* $type_G$*, an* ATTGI grammar *is denoted by* $GG = (TriATGI,\ (G, type_G : G \to TriATGI),\ P)$*, where* P *is a set of IE-triple rules that are typed over* $TriATGI$.

The corresponding graph language *is defined by the set of all the concretely typed triple graphs that are generated by a meta-transformation (cf. definitions 50 and 51):* $L(GG) = \{(H,\ type_H : H \to TriATGI) \mid \exists\ meta - transformation\ (G, type_G) \overset{*}{\Rightarrow} (H, type_H)\}$.

Finally, the following theorem states that for each ATTGI grammar, there is an equivalent graph grammar without inheritance. Moreover, it is also possible to find an equivalent graph grammar, where all the rules are concrete.

Theorem 55 *(Equivalence of Attributed Triple Graph Grammars)*

For each ATTGI grammar $GG = (TriATGI, (G, type_G), P)$ *with IE-triple rules* P *there are:*

1. *An equivalent ATTGI grammar* $\widehat{GG} = (TriATGI, (G, type_G), \widehat{P})$ *with concrete IE-triple rules* $\widehat{P}$*, i.e.* $L(GG) = L(\widehat{GG})$.

2. *An equivalent attributed typed triple graph grammar without inheritance* $\overline{GG} = (\overline{TriATG}, (G, \overline{type_G}), \overline{P})$ *typed over* $\overline{TriATG}$ *where* $\overline{TriATG}$ *is the closure of* $TriATGI$*, and with productions* $\overline{P}$*, i.e.* $L(GG) \cong L(\overline{GG})$*, that means:* $(G, type_G) \in L(GG) \Leftrightarrow (G, \overline{type_G}) \in L(\overline{GG})$.

Construction.

1. The set $\widehat{P}$ is defined by $\widehat{P} = \cup_{p \in P} \widehat{p}$ with $\widehat{p}$ the set of all concrete IE-rules with respect to p.

2. $\overline{type_G} : G \to \overline{TriATG}$ is the triple graph morphism corresponding to the triple clan morphism $type_G$ (see Theorem 46). $\overline{P}$ is defined by $\overline{P} = \cup_{p \in P} \{\overline{p_t} \mid p_t \in \widehat{p}\}$, where for $p_t \in \widehat{p}$ with $p_t = (p, t, \overline{AC})$ we define $\overline{p_t} = (p, \overline{t}, \overline{AC'})$ with $u_{triATG} \circ \overline{t_X} = t_X$ for $X \in \{L, K, R\}$ and $\overline{AC'}$ is defined by AC as follows:

   For each $(x_i \colon L \to X_i, t_{X_i}, \{(y_{ij} \colon X_i \to Y_{ij}, t_{Y_{ij}})\})$, we have all $(x_i \colon L \to X_i, \overline{t_{X_i}}, \{(y_{ij} \colon X_i \to Y_{ij}, \overline{t_{Y_{ij}}})\})$ with $t_{X_i} = u_{TriATG} \circ \overline{t_{X_i}}$ and $t_{Y_{ij}} = u_{TriATG} \circ \overline{t_{Y_{ij}}}$.

8 Conclusions

In this report, we have presented a formalization of triple graph grammars based on the double pushout approach. We have extended the notion of triple graphs by allowing attributes in nodes and edges, typing to a triple type graph, and allowing the target of the correspondence functions to be a node, an edge, or a special element meaning that the function is undefined. Moreover, we have equipped the type graph with node and edge inheritance. Thus, rules may contain nodes and edges whose types in the type graph are refined by other ones. In this way, these elements can be matched with any instance of their subtypes. This technique makes the transformation rules much more compact and allows the integration of graph transformation and meta-modelling techniques.

The developed theory is highly relevant as triple graph grammars are increasingly used for model transformation [Taentzer *et al.*, 2005], to maintain relations between two models (for example, when relating abstract and concrete syntax in visual languages [Guerra and de Lara, 2004], or when relating different views of a system [Guerra *et al.*, 2005]) and for incremental transformations. We believe the theory can also be useful in order to formally express the semantics of transformation languages such as the QVT (Query/Views/Transformation) proposed by the OMG [QVT].

References

[Atkinson and Kühne, 2002] C. Atkinson, and T. Kühne. Rearchitecting the UML infrastructure. *ACM Transactions on Modeling and Computer Simulation*, Vol 12(4), pages 290–321. 2002.

[Bardohl *et al.*, 2004] R. Bardohl, H. Ehrig, J. de Lara, and G. Taentzer. Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation. *Proceedings of the ETAPS/FASE'04*, LNCS 2984, pages 214–228. 2004.

[Corradini *et al.*, 1996] A. Corradini, U. Montanari, and F. Rossi. Graph Processes. *Fundamenta Informaticae*, vol. 6(3-4):241–265. IOS Press. 1996.

[de Lara *et al.*, 2005] J. de Lara, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Attributed Graph Transformation with Node Type Inheritance: Long Version. *Technical Report of the Technical University of Berlin*, Vol 03-2005.

[Ehrig *et al.*, 2006] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* Editorial Springer (to appear in 2006).

[Ehrig *et al.*, 2005a] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Formal Integration of Inheritance with Typed Attributed Graph Transformation for Efficient VL Definition and Model Manipulation. *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 71–78. Dallas (USA), September 2005.

[Ehrig *et al.*, 2005b] H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, and S. Varró-Gyapay. Termination Criteria for Model Transformation. *Proceedings of the ETAPS/FASE'05*, LNCS 3442, pages 49–63. 2005. Springer.

[Ehrig *et al.*, 2004a] H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive High-Level Replacement Categories and Systems. *Proceedings of the ICGT'04*, LNCS 3256, pages 144–160. Rome (Italy), September 2004. Springer.

[Ehrig *et al.*, 2004b] H. Ehrig, U. Prange, and G. Taentzer. Fundamental Theory for Typed Attributed Graph Transformation. *Proceedings of the ICGT'04*, LNCS 3256, pages 161–177. Rome (Italy), September 2004. Springer.

[Ehrig *et al.*, 1999] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation.* (1) Editorial World Scientific. 1999.

[Guerra *et al.*, 2005] E. Guerra, P. Díaz, and J. de Lara. A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views. *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 284–286. Dallas (USA), September 2005.

[Guerra and de Lara, 2004] E. Guerra, and J. de Lara. Event-Driven Grammars: Towards the Integration of Meta-Modelling and Graph Transformation. *Proceedings of the ICGT'04*, LNCS 3256, pages 54–69. Rome (Italy), September 2004. Springer.

[Heckel and Wagner, 1995] R. Heckel, and A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. *Proceedings of the SEGRAGRA'95*, ENTCS Vol 2. 1995.

[MDA] MDA at the OMG's home page: `http://www.omg.org/mda/`.

[QVT] QVT specification at the OMG's home page: `http://www.omg.org/docs/ptc/05-11-01.pdf`.

[Schürr, 1994] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. *Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994)*, LNCS 903, pages 151–163. Herrsching (Germany), 1995. Springer.

[Stahl and Völter, 2006] Stahl. T., Völter, M. *Model-Driven Software Development.* Wiley. 2006.

[Taentzer *et al.*, 2005] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varró, and S. Varró-Gyapay. Model Transformation by Graph Transformation: A Comparative Study. *Proceedings of the International Workshop on Model Transformation in Practice (MTiP), satellite event of MoDELS'05.* 2005.

[Taentzer and Rensink, 2005] G. Taentzer and A. Rensink. Ensuring Structural Constraints in Graph-Based Models with Type Inheritance. *Proceedings of FASE'05*, LNCS 3442, pages 64-79. 2005.

[UML] UML specification at the OMG's home page: `http://www.omg.org/UML`.