Electronic Communications of the EASST
Volume 6 (2007)

Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)

Simulating Multigraph Transformations Using Simple Graphs

Iovka Boneva, Frank Hermann, Harmen Kastenberg, and Arend Rensink

14 pages

Guest Editors: Karsten Ehrig, Holger Giese
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Simulating Multigraph Transformations Using Simple Graphs

Iovka Boneva[1], Frank Hermann[2], Harmen Kastenberg[1], and Arend Rensink[1]

[1] bonevai, h.kastenberg, rensink [at] cs.utwente.nl
Department of Computer Science, University of Twente
P.O. Box 217, NL-7500 AE Enschede, The Netherlands

[2]frank [at] cs.tu-berlin.de
Department of Electrical Engineering and Computer Science
Technical University of Berlin, D-10587 Berlin, Germany

Abstract: Application of graph transformations for software verification and model transformation is an emergent field of research. In particular, graph transformation approaches provide a natural way of modelling object oriented systems and semantics of object-oriented languages.

There exist a number of tools for graph transformations that are often specialised in a particular kind of graphs and/or graph transformation approaches, depending on the desired application domain. The main drawback of this diversity is the lack of interoperability.

In this paper we show how (typed) multigraph production systems can be translated into (typed) simple-graph production systems. The presented construction enables the use of multigraphs with DPO transformation approach in tools that only support simple graphs with SPO transformation approach, e.g. the GROOVE tool.

Keywords: graph transformations, graph transformation tools, tool interoperability, multigraphs, simple graphs

1 Introduction

Application of graph transformations for software verification and model transformation is an emergent field of research. In particular, graph transformation approaches provide a natural way of modelling object oriented systems and semantics of object-oriented languages [KKR06] or graphical modelling languages such as the UML [OMG05], see for instance [Hau06].

For performing the actual graph transformations, different approaches are around ranging from hyperedge replacement approach (see e.g. [DKH97]), logic based approach (see e.g. [Cou97]) to different algebraic approaches such as Single Pushout (SPO) [EHK+97] and Double Pushout (DPO) [CMR+97] approach. These different approaches all have specific application areas in which their features are used in an optimal fashion.

Another difference is the use of either multigraphs or simple graphs for modelling the application domain. Whereas the former is more general, the latter suites better when using graphs for representing relations between objects in order to reason about these objects using (first-order) logical formulae [Ren04b]. While SPO can be applied for both multigraphs and simple graphs, DPO is not defined for simple graphs in general.

For most tools performing graph transformations, the graph representation formalism and the transformation approach are determined by the targeted application domain. For instance, the GROOVE tool [Ren04a] is designed for modelling dynamic systems and verifying properties about their behaviour by generating all possible system configurations. GROOVE uses simple graphs and performs SPO based graph transformations. Another example is the AGG tool [TER99] which handles multigraphs with SPO and is used e.g. for independence and termination analysis on graph grammars.

The main drawback of this diversity in tools is their poor interoperability. One attempt to bridge this gap is the introduction of a common language used for exchanging models among tools, called the Graph eXchange Language (or GXL for short) [SSHW]. In order to extend this work for also exchanging the transformation specifications, GTXL [Tae01] has been proposed. However, since every implementation of a specific approach is not aware of details of other approaches, it is very difficult to include all the features in one common standard and thereby enable tools to perform semantically equivalent transformations.

In a previous work [HKM06] we have proposed translations of graph production systems between GROOVE and AGG, but these translations were too specific and are not applicable in a more general context. Moreover, these translations were not invertible.

In the current paper, we generalise these translations to a context that is tool independent. We show how one can encode typed multigraph production systems into simple-graph production systems, and simulate DPO transformations of multigraphs with SPO transformations on simple graphs. Then we shortly discuss how DPO transformations for multigraphs can be handled by a tool supporting only SPO on simple graphs. These results should allow, for instance, to use the GROOVE tool (or any other tool using simple graphs) with multigraphs. As a further extension, we believe that it would be possible to apply the theory of Subobject Transformation Systems [CHS06] in GROOVE.

Running Example. Throughout this paper we will clarify our ideas and results using a simple example. In the example we model the dynamic behaviour of Lists and Objects that can be elements of some specific Lists. One Object may occur in a List several times. We assume that Objects can be created instantly by the environment (which we do not model in this example). Once Objects are around, different actions can be performed on Lists and Objects, like adding Objects to Lists and moving, removing or copying Objects.

Fig. 1 depicts a possible configuration with two Lists: one containing a single Object and another having two entries referring to the same Object. In each configuration we assume that all List- and Object-instances have their own identity, although we do not show these identities.

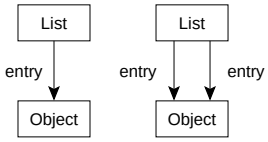


Figure 1: Example configuration of Lists and Objects.

Organisation of the Paper. The remaining of the paper is structured as follows. In Section 2 we provide a formal basis for the rest of the paper. In Section 3 we define our translation of multigraphs to simple graphs and prove the equivalence of DPO transformations on multigraphs on the one hand, and SPO transformations on (special) simple graphs on the other hand. In Section 4 we describe how this equivalence can be extended to typed/labelled graphs. Then, in Section 5 we describe how DPO transformations on multigraphs can be handled by tools implementing the SPO transformation approach, such as the GROOVE tool. Finally, in Section 6 concludes and gives some hints on the way we would like to use the results of this work for improving state space exploration in GROOVE.

2 Background

2.1 Graphs and Graph Morphisms

Graphs are a very powerful means of modelling systems and their behaviour. As will become clear in this paper, in some cases it is very important which notion of graphs is used, since the theory applied may depend on this choice quite heavily.

The *graph* concept is differently interpreted by people working in different domains or even in the same domain. Graphs can e.g. be *deterministic*, *directed* or *labelled*. In this paper we will explicitly distinguish between what we call *multigraphs* and *simple graphs*.

Definition 1 (multigraph, multigraph morphism) A *multigraph* is a tuple $G = \langle V_G, E_G, \mathsf{src}_G, \mathsf{tgt}_G \rangle$ where:

- V_G is a set of *nodes* (or vertexes);
- E_G is a set of *edges*;
- $\mathsf{src}_G, \mathsf{tgt}_G : E_G \to V_G$ are *source* and *target* functions.

A *multigraph morphism* $f : G \to H$ is a pair $\langle f_V, f_E \rangle$, where $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ are functions compatible with src and tgt functions, i.e.

- $f_V \circ \mathsf{src}_G = \mathsf{src}_H \circ f_E$;
- $f_V \circ \mathsf{tgt}_G = \mathsf{tgt}_H \circ f_E$. ■

Definition 2 (simple graph, simple graph morphism) Let Lab be a finite set of labels. A *simple graph* labelled over Lab is a tuple $G = \langle V_G, E_G \rangle$ where

- V_G is a set of *nodes* (or vertexes);
- $E_G \subseteq V_G \times \mathsf{Lab} \times V_G$ is a set of *edges*.

The source and target functions $\mathsf{src}_G, \mathsf{tgt}_G : E_G \to V_G$ are defined for any edge $e = (v, l, v') \in E_G$ by $\mathsf{src}_G(e) = v$ and $\mathsf{tgt}_G(e) = v'$.

A *simple graph morphism* $f : G \to H$ is a pair $\langle f_V, f_E \rangle$, where $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ are functions compatible with src and tgt functions and with labelling, i.e. for any edge $(v, l, v') \in E_G$, $f_E((v, l, v')) = (f_V(v), l, f_V(v'))$. ■

In the sequel we will call a graph morphism $f\colon G \to H$ *total* if its components f_V and f_E are total functions, and *partial* if its components are total functions from G' to H, where G' is some subgraph of G. An *injective* morphism is a morphism induced by injective functions. We will denote the set of multigraphs as $\mathscr{MG}$ and the set of simple graphs over Lab as $\mathscr{SG}(\mathsf{Lab})$. Hereafter, we will use the term *graph* to designate either a multigraph or a simple graph.

In our formal definitions we use unlabelled multigraphs and labelled simple graphs. We start with unlabelled multigraphs in order to keep proofs simple. However, all results of the paper can be extended to labelled graphs, as it will be discussed in Section 4. Therefore, our examples will already freely use labels on both nodes and edges.

2.2 Graph Transformations

When modelling system states as graphs, the dynamics of the system can be specified by graph transformations. The changes of states are then described by *graph productions*, also called *graph transformation rules*.

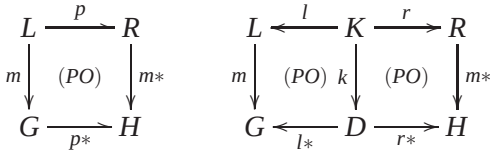
Definition 3 (graph production) A *graph production* p consists of two graphs L and R, being its *left-hand-side* and *right-hand-side*, respectively, together with a partial graph morphism from L to R, called the *rule morphism*.

We often denote a graph production p as $p\colon L \to R$, also using p when referring to the rule morphism. When combining a graph G with a set $\mathscr{P}$ of graph productions, we get a *graph production system* $GPS = \langle G, \mathscr{P} \rangle$. In a graph production system, G is called the *start graph*. By *applying* graph productions to G we can *derive* other graphs. The applications of graph productions are defined on categories in which the objects are multigraphs or simple graphs and the arrows are the corresponding graph morphisms. For an introduction to category theory, see e.g. [BW95]. Whether a rule is applicable and to what resulting graph a derivation leads depends on the particular graph transformation approach being applied. In this paper we distinguish between the Single Pushout (SPO) [EHK+97] and the Double Pushout (DPO) [CMR+97] approach. For applying a production in the SPO approach, we only need an occurrence of the left-hand-side of the graph production. When the application of a graph production would delete a node but not all of its adjacent edges, those *dangling edges* will also be removed. Furthermore, if the application prescribes one node (or edge) to be both deleted and preserved, this conflict is solved in favour of deletion. These conflicts are resolved in the DPO approach by forbidding such applications of productions, i.e. the DPO approach requires additional conditions on the applications which are called the *dangling edge condition* and the *identification condition* (together referred to as the *gluing condition*).

In the DPO approach, a graph production $p\colon L \to R$ is depicted as a span $L \xleftarrow{l} K \xrightarrow{r} R$ of total graph morphisms, such that $K = L \cap R$, $l\colon dom(p) \to L$, and $r\colon dom(p) \to R$. To be deterministic, it is necessary that either rule morphisms or matchings are injective. We will now define applications of graph productions and the corresponding derivations for both SPO and DPO.

Definition 4 (derivation) Given a graph production $p\colon L \to R$ and a graph G, a total graph morphism $m\colon L \to G$ is called *matching*. The *direct derivation* from a graph G to a graph H through

the production p via matching m, denoted $G \overset{p,m}{\Longrightarrow} H$, is constructed:
(SPO) as the pushout of p and m in the category of graphs and partial graph morphisms (see Fig. 2(a));
(DPO) by taking, in the category of graphs and total graph morphisms, first the pushout complement D (with $k\colon K \to D$ and $l*\colon D \to G$) of l and m, if it exists (ensured by the gluing condition), and then the pushout of r and k (see Fig. 2(b)). ■



(a) SPO (b) DPO

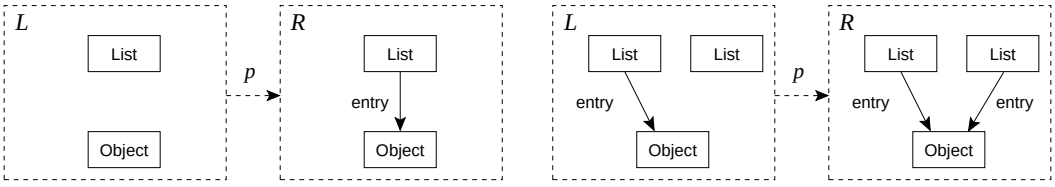
Figure 2: Graph H as the result of an SPO and a DPO derivation.

Intuitively, applying a graph production p to a graph G can be seen as a sequence of two actions: *find* an occurrence (matching) of L in G and then *replace* that occurrence by R. This then results in the graph H. An example direct derivation is shown in Fig. 4.

An important difference between SPO and DPO is the fact that DPO does not work on simple graphs with arbitrary matchings, because in some cases the required pushout construction is not unique or does not exist. In this paper we do apply DPO on simple graphs, but then ensure that we restrict to a special class of matchings and/or morphisms. This issue will be discussed in Section 3.

2.3 Back to the Example

Now that we have introduced the notion of graphs and the graph transformation technique, we can recall the example and give a formal description of the actions. In Fig. 3 we specify some of the actions from the example as graph transformation rules by showing their left-hand-side and right-hand-side graph. The rule morphisms in Fig. 3 are defined by the placing of the elements.



(a) add (b) copy

Figure 3: Graph transformation rules for some of the actions in the example.

In Fig. 4 we show a single (SPO) rule application in which we apply the copy-rule (Fig. 3(b)) on a graph G consisting of two Lists each containing one Object, also showing the resulting graph H.

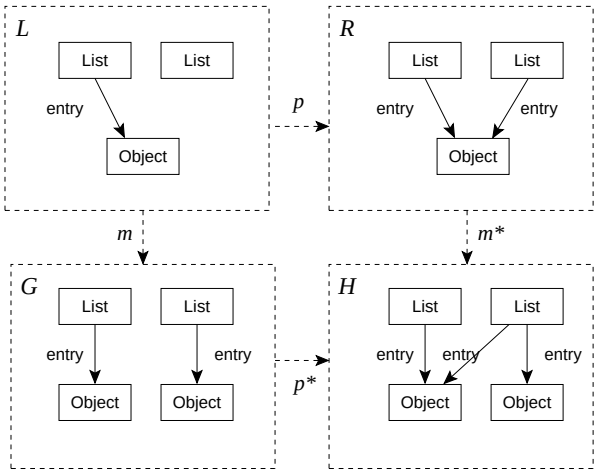


Figure 4: An example direct derivation.

3 From Multigraphs to Simple Graphs and back again

In this section we describe our translation between multigraphs and simple graphs. At a categorical level we will show that these translations are functors which are isomorphisms, moreover being each others inverse.

3.1 From Multigraphs to Simple Graphs

Consider the set of labels $L_{MG} = \{\mathtt{s},\mathtt{t}\}$. The function **Sim** maps multigraphs from $\mathscr{MG}$ into simple graphs in $\mathscr{SG}(L_{MG})$ as follows: every edge e in the multigraph with source node v_s and target node v_t becomes a special node (this we call a *proxy* node) with two outgoing edges $(e,\mathtt{s},v_\mathrm{s})$ and $(e,\mathtt{t},v_\mathrm{t})$. Thus, we will use e as a variable ranging over edges of multigraphs and proxy nodes in simple graphs. Fig. 5 shows an example applying the **Sim** function.

Formally, let $G = \langle V_G, E_G, \mathtt{src}_G, \mathtt{tgt}_G\rangle$ be a multigraph. Then $\mathbf{Sim}(G)$ is the graph $H = \langle V_H, E_H\rangle$ with

- $V_H = V_G \cup E_G$, that is, edges of G are nodes in H;
- $E_H = \bigcup_{e\in E_G}\{(e,\mathtt{s},\mathtt{src}_G(e)),(e,\mathtt{t},\mathtt{tgt}_G(e))\}$.

The **Sim** function can be extended on graph morphisms. That is, if G and H are multigraphs and $m\colon G\to H$ is a morphism, then $\mathbf{Sim}(m)\colon \mathbf{Sim}(G)\to\mathbf{Sim}(H)$ is the morphism defined by [1]:

- for all v in $V_{\mathbf{Sim}(G)}$ (i.e. $v\in V_G\cup E_G$), $(\mathbf{Sim}(m))(v) = m(v)$;
- for all (e,l,v) in $E_{\mathbf{Sim}(G)}$, $(\mathbf{Sim}(m))((e,l,v)) = (m(e),l,m(v))$.

Note that the definition of $\mathbf{Sim}(m)$ on edges of $\mathbf{Sim}(G)$ ensures that $\mathbf{Sim}(m)$ is indeed a simple graph morphism.

[1] In this definition m is supposed to be a total morphism. This is not a restriction as a partial morphism is a total morphism on a subgraph.

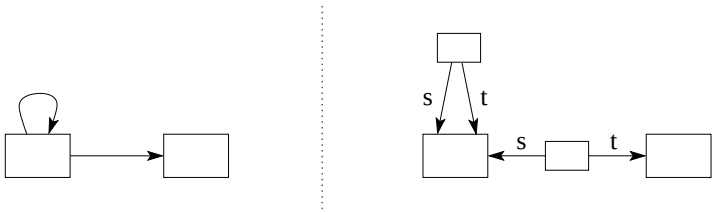


Figure 5: Encoding of a multigraph (on the left) into simple graphs with proxy nodes (on the right) by the **Sim** function.

3.2 From Simple Graphs to Multigraphs

Let $\mathscr{SG}_{\mathscr{MG}}$ be the set of bipartite simple graphs over L_{MG} satisfying the following conditions: $G = (V,E) \in \mathscr{SG}_{\mathscr{MG}}$ if

1. $V = V_\mathrm{n} \cup V_\mathrm{e}$ where V_n and V_e are two disjoint sets;
2. $E = E_\mathrm{s} \cup E_\mathrm{t}$ where E_s and E_t are disjoint sets and $E_\mathrm{s} \subseteq V_\mathrm{e} \times \{\mathsf{s}\} \times V_\mathrm{n}$, and $E_\mathrm{t} \subseteq V_\mathrm{e} \times \{\mathsf{t}\} \times V_\mathrm{n}$;
3. any node e in V_e has exactly two adjacent edges $(e,\mathsf{s},v'_\mathrm{n}) \in E_\mathrm{s}$ and $(e,\mathsf{t},v''_\mathrm{n})$ for some $v'_\mathrm{n}, v''_\mathrm{n} \in V_\mathrm{n}$.

We now define the function $\mathbf{Sim}^{-1}\colon \mathscr{SG}_{\mathscr{MG}} \to \mathscr{MG}$ as follows: if $G = \langle V_\mathrm{n} \cup V_\mathrm{e}, E_G \rangle$ where V_n and V_e are as in the description of $\mathscr{SG}_{\mathscr{MG}}$ stated above, then $H = \mathbf{Sim}^{-1}(G)$ is the graph $\langle V,E,\mathsf{src},\mathsf{tgt}\rangle$ such that $V = V_\mathrm{n}$, $E = V_\mathrm{e}$, and for any $e \in E$, $\mathsf{src}(e) = v_\mathrm{s}$ and $\mathsf{tgt}(e) = v_\mathrm{t}$, where $v_\mathrm{s}, v_\mathrm{t} \in V_\mathrm{n}$ are the nodes such that $(e,\mathsf{s},v_\mathrm{s}), (e,\mathsf{t},v_\mathrm{t}) \in E_G$. We know by condition 3 of the definition of the set of graphs $\mathscr{SG}_{\mathscr{MG}}$ that the nodes v_s and v_t exist and are unique.

The $\mathbf{Sim}^{-1}$ function can also be extended on graph morphisms. If $m\colon G \to H$ is a simple graph morphism, then $\mathbf{Sim}^{-1}(m)\colon \mathbf{Sim}^{-1}(G) \to \mathbf{Sim}^{-1}(H)$ is the multigraph morphism such that for any x in V_G, $(\mathbf{Sim}^{-1}(m))(x) = m(x)$. We now show that $\mathbf{Sim}^{-1}(m)$ defined this way is indeed a multigraph morphism.

Let $G' = \mathbf{Sim}^{-1}(G)$, $H' = \mathbf{Sim}^{-1}(H)$ and $m' = \mathbf{Sim}^{-1}(m)$. Then for any edge $e \in E_{G'}$, $(m' \circ \mathsf{src}_{G'})(e) = m(v_\mathrm{s})$ where v_s is the unique node in G such that $(e,\mathsf{s},v_\mathrm{s})$ is an edge of G. As m is a simple graph morphism, $(m(e),\mathsf{s},m(v_\mathrm{s}))$ is an edge in H. On the other hand, $(\mathsf{src}_{H'} \circ m')(e) = \mathsf{src}_{H'}(m(e))$ is the unique node v'_s in H such that $(m(e),\mathsf{s},v'_\mathrm{s})$ is a edge in H. We deduce then that both $(m(e),\mathsf{s},v'_\mathrm{s})$ and $(m(e),\mathsf{s},m(v_\mathrm{s}))$ are edges in H. By uniqueness of v'_s, necessarily $v'_\mathrm{s} = m(v_\mathrm{s})$, so $m' \circ \mathsf{src}_{G'} = \mathsf{src}_{H'} \circ m'$. We can see in a similar way that $m' \circ \mathsf{tgt}_{G'} = \mathsf{tgt}_{H'} \circ m'$.

It is not very hard to see that $\mathscr{SG}_{\mathscr{MG}}$ is exactly the set of simple graphs that are images of multigraphs by the **Sim** function, and that the function $\mathbf{Sim}^{-1}$ is the inverse of the function **Sim**. This will be formally stated in the following section.

3.3 Categories for Multigraphs and Simple Graphs

In this section we define the categories **MG** and $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$ on which DPO transformation is defined for multigraphs and for simple graphs that are encodings of multigraphs. We show also that the functions **Sim** and $\mathbf{Sim}^{-1}$ define free functors from **MG** to $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$ and from

$\mathbf{SG_{MG}}(L_{MG})$ to **MG** respectively. This will guarantee that performing DPO transformations on multigraphs can be simulated by DPO transformations on simple graphs that belong to $\mathscr{SG}_{\mathscr{MG}}$, as stated in Theorem 1. The reader who is not familiar with category theory will probably only be interested in the result of this theorem.

Definition 5 (categories **MG**, $\mathbf{SG}(L)$, and $\mathbf{SG_{MG}}(L_{MG})$) **MG** is the category whose objects are elements of $\mathscr{MG}$ and whose arrows are multigraph morphisms. $\mathbf{SG}(L)$ is the category whose objects are simple graphs over the set of labels L and whose arrows are simple graph morphisms. Finally, $\mathbf{SG_{MG}}(L_{MG})$ is the category whose objects are elements of $\mathscr{SG}_{\mathscr{MG}}$ and whose arrows are simple graph morphisms.

Note that $\mathbf{SG_{MG}}(L_{MG})$ can be equivalently defined as the full subcategory of $\mathbf{SG}(L_{MG})$ induced by $\mathscr{SG}_{\mathscr{MG}}$.

Recall that a functor $f = \langle f_{\mathrm{o}}, f_{\mathrm{m}} \rangle$ from a category **C** to a category **D** is a function with f_{o} (resp. f_{m}) associating objects (resp. morphisms) of **D** with objects (resp. morphisms) of **C** and such that f preserves morphisms, identities and composition.

The following lemma easily follows from the definitions.

Lemma 1 *It holds that*

1. **Sim** *is a functor from* **MG** *to* $\mathbf{SG_{MG}}(L_{MG})$ *and*
2. $\mathbf{Sim}^{-1}$ *is a functor from* $\mathbf{SG_{MG}}(L_{MG})$ *to* **MG***;*
3. *the functors* **Sim** *and* $\mathbf{Sim}^{-1}$ *are isomorphisms:*

$$\mathbf{Sim} \circ \mathbf{Sim}^{-1} = ID_{\mathbf{SG_{MG}}(L_{MG})} \quad \text{and} \quad \mathbf{Sim}^{-1} \circ \mathbf{Sim} = ID_{\mathbf{MG}}.$$

Graph morphisms are called edge reflecting if edges are reflected along their boundary, i.e. whenever there is an edge between two nodes in the image of the morphism, there should be an edge between the pre-images of these nodes in the domain of the morphism (see next lemma).

Lemma 2 *All morphisms* $f : G \to H$ *in* $\mathbf{SG_{MG}}(L_{MG})$ *are edge reflecting, i.e.*

$$\textbf{if} \quad (f(x), l, f(y)) \in E_H \quad \textbf{then} \quad (x, l, y) \in E_G.$$

Proof. It is enough to show that **Sim** translates to edge reflecting morphisms, because the categories are isomorphic. By definition, **Sim** translates edges to special nodes with two outgoing edges to other nodes. Nodes in **MG** are connected via structured edges in $\mathbf{SG_{MG}}(L_{MG})$, thus edges connect an original node with a proxy node. Let f be a graph morphism in **MG**. If $\mathbf{Sim}(f)$ reaches a proxy node, f has to map to the original edge. Therefore, also the adjacent edges are reached by $\mathbf{Sim}(f)$ and thus, $\mathbf{Sim}(f)$ is edge reflecting. □

3.4 Multigraph versus Simple Graph transformations

In the sequel we combine the graph categories **MG**, $\mathbf{SG_{MG}}(L_{MG})$ and $\mathbf{SG}(L_{MG})$ with the transformation approaches SPO and DPO. We will denote such combinations with **MG**+DPO etc. The

aim of this paper is to translate **MG**+DPO into **SG**(L_{MG})+SPO. This is achieved in two steps:

$$\textbf{MG}+\text{DPO} \;\rightarrow\; \textbf{SG}_{\textbf{MG}}(L_{MG})+\text{DPO} \;\rightarrow\; \textbf{SG}(L_{MG})+\text{SPO}$$

The first step consists in encoding multigraphs and production rules using the **Sim** function, thus obtaining simple graphs in $\mathscr{SG}_{\mathscr{MG}}$ and simple graph morphisms. The second step consists in encoding the DPO rules into SPO rules. In [HHT96] (Proposition 3.5) it has been shown that it is possible to translate the application conditions of a DPO derivation (i.e. dangling edge and identification condition) in **MG** to equivalent negative application conditions (NACs) for performing SPO derivations in **MG**. In Theorem 1 we show that the initial DPO transformations in **MG** can be simulated by the translated SPO transformation in **SG**(L_{MG}).

Remark 1 *(Uniqueness of derivations)* To be deterministic for given graph production and matching, DPO derivations need the uniqueness of pushout complements. In adhesive categories this is the case if the rule morphisms are, or the match is, monomorphic (see Lemma 15 in [LS04]), meaning injective in the category **Graph**. In our setting, the category **MG** is adhesive and therefore also **SG**$_{\textbf{MG}}(L_{MG})$ is, because it is isomorphic. The monomorphisms in the latter one are also equalisers by their property of being edge reflecting and thus, they are regular monomorphisms.

Given a DPO rule $p = L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$, we use **Sim**$(p)$ to denote **Sim**$(L) \stackrel{\textbf{Sim}(l)}{\leftarrow}$ **Sim**$(K) \stackrel{\textbf{Sim}(r)}{\rightarrow}$ **Sim**(R), and we denote by **Sim**$^*(p)$ the translated rule equipped with additional NACs, as described in [HHT96]. For the following lemma we interpret graphs of **SG**$_{\textbf{MG}}(L_{MG})$ as graphs in **MG** by forgetting all labels. This allows us to show that pushouts are not only translated to those in a different category, but also remain pushouts in the original category of multigraphs, after applying **Sim**. An extension of **MG** with labels is direct and only adds information, which does not interfere with the pushout construction.

Lemma 3

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & (PO) & \downarrow \\ C & \longrightarrow & D \end{array} \quad \textit{in } \textbf{MG} \textit{ implies} \quad \begin{array}{ccc} \textbf{Sim}(A) & \longrightarrow & \textbf{Sim}(B) \\ \downarrow & (PO) & \downarrow \\ \textbf{Sim}(C) & \longrightarrow & \textbf{Sim}(D) \end{array} \quad \textit{in } \textbf{MG} \textit{ up to label information.}$$

Proof. (sketch) Pushouts in **MG** are constructed component-wise for the sets of edges and nodes by building the disjoint union and factorising along the equivalence generated by the span of morphisms. The definition of **Sim** is compatible with the standard pushout construction, i.e. **Sim**$(D) =$ **Sim**$(B +_A C) \cong$ **Sim**$(B) +_{\textbf{Sim}(A)}$ **Sim**(C). □

Theorem 1 (simulation) *Given a rule $p = L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ and a match $m : L \rightarrow G$ in* **MG***, where l is injective, the following three are equivalent:*

1. $G \stackrel{p,m}{\Longrightarrow}_{\text{DPO}} G'$ *in* **MG***;*
2. **Sim**$(G) \stackrel{\textbf{Sim}(p),\textbf{Sim}(m)}{\Longrightarrow}_{\text{DPO}}$ **Sim**(G') *in* **SG**$_{\textbf{MG}}(L_{MG})$*;*

3. $\mathbf{Sim}(G) \xRightarrow{\mathbf{Sim}^*(p),\mathbf{Sim}(m)}_{\text{SPO}} \mathbf{Sim}(G')$ *in* $\mathbf{SG}(L_{MG})$.

Furthermore, if a rule in 2 or 3 is applicable, then the result is always a graph in $\mathbf{SG}(L_{MG})$.

Proof. $1 \Leftrightarrow 2$ **Sim** and $\mathbf{Sim}^{-1}$ are isomorphisms by Lemma 1 and hence, they preserve all Limits and Colimits. Since l is injective the DPO-derivations are unique up to isomorphism.

$2 \Rightarrow 3$ The derivation in 2 can be considered as a derivation in **MG** up to labels, according to Lemma 3. Then using [HHT96], it is equivalent to an SPO derivation with NACs in **MG** with result $\mathbf{Sim}(G')$, that is, $\mathbf{Sim}(G')$ is the pushout of p and m in **MG**. But, as $\mathbf{Sim}(G')$ is a simple graph, it is also the pushout of p and m in $\mathbf{SG}(L_{MG})$, up to labels. Because of the strict relation between the labels in graphs in $\mathscr{SG}_{\mathscr{MG}}$ and their structure, it is not difficult to see that $\mathbf{Sim}(G')$ is also the pushout of p and m in $\mathbf{SG}(L_{MG})$ without ignoring the labels.

$3 \Rightarrow 2$ Let H' be the result of the derivation (a) $\mathbf{Sim}(G) \xRightarrow{\mathbf{Sim}^*(p),\mathbf{Sim}(m)}_{\text{SPO}} H'$ in **MG**. By [HHT96] we know that then (b) $\mathbf{Sim}(G) \xRightarrow{\mathbf{Sim}(p),\mathbf{Sim}(m)}_{\text{DPO}} H'$ is a derivation in **MG**. Since $\mathbf{Sim}(p)$, $\mathbf{Sim}(m)$ are morphisms in $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$, by Lemma 2 we know that they are edge reflecting, and this allows to deduce that the graph H' is a simple graph, that is, an object of $\mathbf{SG}(L_{MG})$. Now, as $\mathbf{SG}(L_{MG})$ is a full subcategory of **MG** and by (a), we have that H' is the pushout of $\mathbf{Sim}^*(p)$ and $\mathbf{Sim}(m)$ in $\mathbf{SG}(L_{MG})$. By uniqueness of this pushout and the derivation in point 3 we deduce that $H' = H$, thus (b) is a derivation in $\mathbf{SG}(L_{MG})$. Finally, one can see that H' and the context graph in (b) are also objects of $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$ because the translated rule will only produce and delete complete structured edges by definition of **Sim**. Hence, no garbage (i.e. proxy nodes with either an outgoing s-edge or a t-edge, but not both) will occur. Thus, (b) is also a derivation in $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$.

Result $H \cong \mathbf{Sim}(G')$ is a direct consequence of the last part of the proof for the previous item. □

4 Extensions

Theorem 1 immediately extends to rules with negative application conditions, because they contain just additional graphs and morphisms of the same kind. Thus, we will not describe this aspect in more detail.

We are also confident that the results from this paper can be extended in a straightforward manner to hypergraphs [Kön02], which differ from multigraphs in not having source and target functions, but rather a single function $\mathsf{ends}\colon E_G \to V_G^*$ that associates with every edge a *string* of nodes. Hypergraphs can be translated to simple graphs using precisely the same technique of encoding edges as proxy nodes, with in this case as many auxiliary edges (to nodes) as there are elements in $\mathsf{ends}(e)$.

Up to now we have only considered unlabelled and untyped multigraphs, but all the results that we have shown can be easily extended to typed multigraphs, and hence to labelled ones, since labelling can be insured by typing; see, e.g., [EEPT06]. Fig. 6 shows how one of our example labelled multigraphs would be encoded into a simple graph.

Figure 6: Encoding of a labelled multigraph.

A typed graph $\langle G,m\rangle$ is a graph G together with a morphism $m : G \to TG$ to some graph TG called the type graph. A typed graph morphism $f : \langle G,m\rangle \to \langle G',m'\rangle$ is a morphism for which $m = m' \circ f$. Transformations of typed graphs should involve only typed graph morphisms. It is equivalent to consider transformations in a *slice category*. That is, typed transformations in $\mathbf{C}$ w.r.t. the type graph TG are equivalent to transformations in the slice category $\mathbf{C} \downarrow TG$, where $\mathbf{C}$ is either $\mathbf{MG}$ or $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$ and TG is a multigraph or simple graph, respectively. Now, as $\mathbf{MG}$ and $\mathbf{SG}_{\mathbf{MG}}(L_{MG})$ are isomorphic with $\mathbf{Sim}$ as isomorphism functor, it is trivial to see that the slice categories are also isomorphic. Thus, there is a pushout in $\mathbf{MG} \downarrow TG$ if, and only if, there is a pushout in $\mathbf{SG}_{\mathbf{MG}}(L_{MG}) \downarrow \mathbf{Sim}(TG)$. Then the simulation result stated in Theorem 1 also holds for a typed transformation.

However, in this case, an additional translation step is still required to translate to untyped simple graphs. Then we have to extend the labels to encode the typing; hence, the translation is from $[\mathbf{SG}_{\mathbf{MG}}(L_{MG}) \downarrow \mathbf{Sim}(TG)]+\text{SPO}$ to $[\mathbf{SG}_{\mathbf{MG}}(L_{MG} \times (V_{TG} \cup E_{TG}))]+\text{SPO}$. We are convinced that this translation is straightforward, but we have not given the proof.

5 Simulation in SPO Tools

Tools performing graph transformations often implement SPO since this requires only one pushout construction whether for DPO an additional pushout complement construction is needed. Problems arise when performing rule applications using SPO that do not satisfy the gluing condition. In the running example such a situation would occur when applying the delete rule on an Object that is contained in more than one List.

In order still to be able to perform DPO transformation, there are basically two alternatives:

1. restrict rule applications by checking the gluing condition after searching for matchings;
2. encode the gluing condition using additional negative application conditions in the transformation rules.

Choosing the first alternative requires that the tool performs an additional gluing check on the found matches. This gluing check means that for all identifications in the matching and for all node deletions we need to ensure that there is no preserve-delete conflict (identification condition) and that the node-deletions do not cause dangling edges (dangling condition), respectively. The AGG tool's kernel implements SPO and uses a similar mechanism for handling DPO transformations.

The second alternative is based on Theorem 1, in which we show that it is possible to simulate DPO on our special simple graphs by adding additional negative application conditions as described in [HHT96].

Let us now briefly describe how one can use the GROOVE tool (or some other tool supporting simple graph transformations with SPO) for performing DPO transformations on multigraphs. Given a (multi-) graph production system (GPS) $T = \langle G, \mathscr{P} \rangle$, one first has to create the production system $\mathbf{Sim}(T)$ by encoding the graph G and all graphs and morphisms that are parts of the productions in $\mathscr{P}$ in the manner described in Section 3. Note that if some productions include negative application conditions, these conditions together with the morphisms that relate them to the corresponding production are encoded just as normal graphs and morphisms. Now, if the tool offers the possibility to check for the gluing condition (choice 1 above), then the GPS $\mathbf{Sim}(T)$ can be submitted to the tool, specifying that the check for the gluing condition has to be performed. Otherwise (choice 2 above), one has to construct the production system $\mathbf{Sim}^*(T)$ by augmenting $\mathbf{Sim}(T)$ with additional NACs for encoding the gluing condition in $\mathbf{Sim}(T)$. The GPS $\mathbf{Sim}^*(T)$ is then submitted to the tool as a normal (simple) graph production system. Any derivation results obtained by the tool (e.g. graphs that can be derived from the start graph or the actual rule applications) can be transformed back to multigraphs using the $\mathbf{Sim}^{-1}$ mapping. This forth and back translation can be used, for instance, for exchanging results between different graph transformation tools.

6 Conclusion and Future Work

We have proposed a method for performing DPO multigraph transformations using tools handling SPO simple graph transformations. Compared to previous work [HKM06], this method is generic, i.e. has been proved correct on categorical level and does not depend on the tools to be used.

Pushing theory to work in practise. Tool interoperability is one major motivating point to translate graph transformation systems using multigraphs and DPO to equivalent systems with simple graphs and SPO derivations. On the more fundamental level it is even more interesting to have the possibilities of applying a wide range of theoretical results and implementing them in the tool of favour. During the last three decades, a lot of theory was developed using DPO and multigraphs. One special new technique is the analysis of derivations using Subobject Transformation Systems (STS) presented in [CHS06]. Since the GROOVE tool performs graph derivations to verify systems, the translation presented in this paper could give the possibility of combining the power of both (which was not possible before, because STS are not defined for SPO). And indeed, this idea already has a concrete structure: basically one can exploit the possible results of dependencies using a translation to STSs and furthermore, the branching derivations of the state space can be folded into one summary object. Thus, only a small number of derivation steps will have to be performed to construct an abstraction of a much bigger state space. The idea is then to use the abstraction equipped with an STS to deliver only effective states and perform model checking on these states and their concrete successors.

Acknowledgements. The first and third authors are employed in the GROOVE project funded by the Dutch NWO (project number 612.000.314).

Bibliography

[BW95] M. Barr, C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1995.

[CHS06] A. Corradini, F. Hermann, P. Sobociński. Subobject Transformation Systems. *Applied Categorical Structures*, 2006. To appear.

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe. Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach. Pp. 163–246 in [Roz97].

[Cou97] B. Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. Pp. 313–400 in [Roz97].

[DKH97] F. Drewes, H.-J. Kreowski, A. Habel. Hyperedge Replacement Graph Grammars. Pp. 95–162 in [Roz97].

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer Verlag, 2006.

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, A. Corradini. Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach. Pp. 247–312 in [Roz97].

[Hau06] J. H. Hausmann. *Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Techniques*. PhD thesis, Universität Paderborn, 2006.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Special issue of Fundamenta Informaticae* 26(3,4):287–313, 1996.

[HKM06] F. Hermann, H. Kastenberg, T. Modica. Towards Translating Graph Transformation Approaches by Model Transformation. In *Proc. of the Int. Workshop on Graph and Model Transformation (GraMoT'06)*. 2006.

[KKR06] H. Kastenberg, A. Kleppe, A. Rensink. Defining Object-Oriented Execution Semantics Using Graph Transformations. In Gorrieri and Wehrheim (eds.), *Proc. of the 8th IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*. LNCS 4037, pp. 186–201. Springer Verlag, 2006.

[Kön02] B. König. Hypergraph Construction and its Application to the Static Analysis of Concurrent Systems. *Mathematical Structures in Computer Science* 12(2):149–175, 2002.

[LS04] S. Lack, P. Sobociński. Adhesive Categories. In Walukiewicz (ed.), *Proc. of the 7th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'04)*. LNCS 2987, pp. 273–288. Springer Verlag, 2004.

[OMG05] OMG. Unified Modeling Language Specification. 2005. http://www.omg.org/technology/documents/formal/uml.htm

[Ren04a] A. Rensink. The GROOVE Simulator: A Tool for State Space Generation. In Pfaltz et al. (eds.), *Applications of Graph Transformations with Industrial Relevance (AGTIVE'03)*. LNCS 3062, pp. 479–485. Springer Verlag, 2004.

[Ren04b] A. Rensink. Representing First-Order Logic using Graphs. In Ehrig et al. (eds.), *Proc. of the 2nd Int. Conf. on Graph Transformations (ICGT'04)*. LNCS 3256, pp. 319–335. Springer Verlag, 2004.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation*. Volume I: Foundations. World Scientific, 1997.

[SSHW] A. Schürr, S. E. Sim, R. Holt, A. Winter. The GXL Graph eXchange Language. http://www.gupro.de/GXL

[Tae01] G. Taentzer. Towards Common Exchange Formats for Graphs and Graph Transformation Systems. In Padberg (ed.), *Proc. of the Workshop on Uniform Approaches to Graphical Process Specification Techniques (UNIGRA'01)*. ENTCS 44. 2001.

[TER99] G. Taentzer, C. Ermel, M. Rudolf. The AGG Approach: Language and Tool Environment. In Ehrig et al. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformations*. Volume II: Applications, Languages and Tools, pp. 163–246. World Scientific, 1999.