Electronic Communications of the EASST
Volume 47 (2012)

Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2012)

Graph Passing in Graph Transformation

Amir Hossein Ghamarian and Arend Rensink

14 pages

Guest Editors: Andrew Fish, Leen Lambers
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Graph Passing in Graph Transformation

Amir Hossein Ghamarian and Arend Rensink

Department of Computer Science, Universiteit Twente
{a.h.ghamarian, rensink}@cs.utwente.nl

Abstract: Graph transformation works under the whole world assumption. Therefore, in realistic systems, both the individual graphs and the set of all such graphs can grow very large. In reactive formalisms such as process algebra, on the other hand, each system is split into smaller components which continually interact; the interactions pass information such as names or locations between components. The state spaces for the separate components are typically much smaller, and much efficiency can be gained by analysing system behaviour on this level.

In this paper we present a framework for compositional graph transformation inspired by name-passing calculi, in which (knowledge about) subgraphs can be passed between components. Essentially, we define graph-passing (reactive) component rules and their composition into traditional (reductive) whole-world rules. This extends previous work in which a simpler form of composition was proposed. The main result is a soundness and completeness result for the composition, showing that the transformations induced by the component rules and their whole-world counterparts are equivalent.

Keywords: Graph transformation, Compositionality, Soundness and completeness

1 Introduction

Graph transformation has shown to be an expressive and powerful formalism for modelling many kinds of systems, ranging from physical systems to network protocols. The direct correspondence between the formalism and its visualization makes it appealing and intuitive. However, one major drawback of graph transformation systems is that they require the whole system to be modelled by a single component. This "whole-world" view is the consequence of graph transformation's *reductive* semantics, which involves finding rule matches in the current model (host graph) and making local changes without any interaction with the external world. In contrast, *reactive* formalisms such as process algebra enjoy compositionality: submodels can be analyzed individually and produce partial results, which can then be composed to produce the final result.

1.1 Problem statement.

Fig. 1 is a typical process algebraic view: it depicts schematically how a global system is composed from a number of local components (C_1 and C_2), which communicate with each other over shared channels (A_s and A_h), and also communicate with the outside world, either in multi-party communication by exposing their shared channel A_s, or via private channels A_1 and A_2 unknown to the other component.

Note that there are actually two separate notions involved: firstly, components communicate, or synchronise, over a set of shared entities (channels in the case of process algebra, subgraphs in the case of graph transformation); secondly, entities have a limited scope and can be hidden from other components or from the environment. In process algebra, the first is covered by *parallel composition* and the second by *hiding*.

In the setting of graph transformation, components are specified by rule systems in combination with a start graph. In particular, we would have a rule system describing the global system of Fig. 1; the transformations and ensuing graphs describe the dynamic global behaviour of the system. This behaviour is captured by recursively applying enabled rules on the host graph, which leads to a *labelled transitions system* (LTS) with transformation rules as its labels. Consequently, this LTS lends itself to all the classic analysis techniques enabled on the labelled transition systems [MPW92].

We now ask ourselves the following question:

Given a decomposition of the global start graph G into start graphs G_i ($i = 1,2$), can we construct rule systems for C_1, C_2 whose behaviour when applied to the graphs G_i is the same as the global behaviour?

The decomposed local graph transformation systems have smaller LTS's resulting in cheaper analysis cost. Analysis can be carried out on these smaller state spaces and then lifted to the global level, thus partially avoiding the state space explosion common to monolithic models of realistic systems.

Furthermore, the decomposition should allow flexible communication between the components; in particular, we want to pass information in the form of subgraphs between components.

In a previous paper [Ren10], we proposed an initial approach for rule and graph composition, based on common subrules. In this approach, common nodes can only be created and deleted from components simultaneously; in other words, graph passing is not supported. Moreover, we investigated individual rules only and did not yet consider the rule system level.

In this paper, we extend the previous results in two ways. Firstly, we add a *synchronisation interface* that explicitly declares the degree of sharing between local rules, but does not prescribe whether nodes are created, deleted or passed from one component to the other. Secondly, we define behaviour-preserving (de)composition on the level of rule systems.

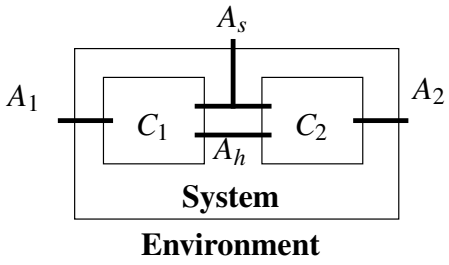


Figure 1: Composition schema: the A_i are shared between C_i and the environment, A_s is shared by all three, and A_h is shared between C_1 and C_2 only.

1.2 Approach

At the core of the approach, we compare the transitions of individual components, $G_1 \xrightarrow{p_1} H_1$ and $G_2 \xrightarrow{p_2} H_2$ where G_i and H_i are graphs and p_i, p transformation rules, to global transitions $G_1 \cup G_2 \xrightarrow{p} H$ and show the conditions under which the following properties hold.

Soundness. Compatible local transitions always give rise to global transitions: $G_1 \xrightarrow{p_1} H_1$ and $G_2 \xrightarrow{p_2} H_2$ imply $G_1 \cup G_2 \xrightarrow{p_1 \cup p_2} H_1 \cup H_2$.

Completeness. All global transitions can be obtained by composing compatible local transitions: $G_1 \cup G_2 \xrightarrow{p} H$ implies that there are rules p_1 and p_2 such that $G_1 \xrightarrow{p_1} H_1$ and $G_2 \xrightarrow{p_2} H_2$ and $H = H_1 \cup H_2$.

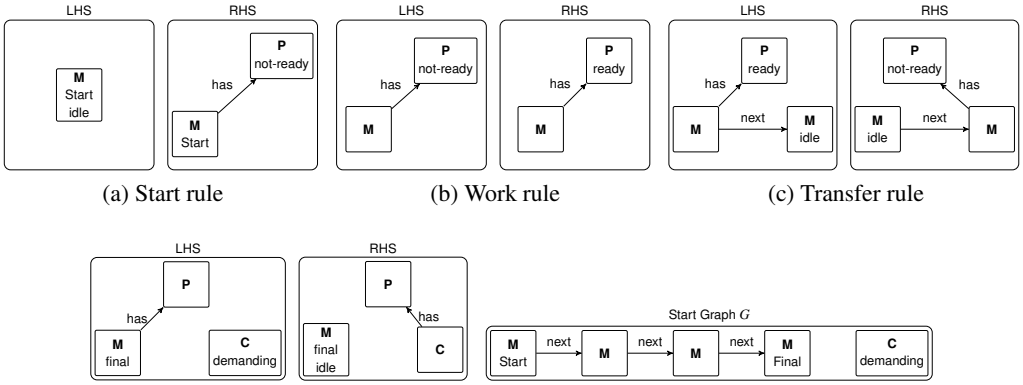
When decomposing a rule system, the split of the host graph is not precisely known, as this may develop during previous rule applications. For that reason, the decomposition must take all possible splits into account. Unfortunately, this gives rise to a very large number of decomposed rules, threatening to make the method infeasible. For that reason, we also study a special scenario where components are specified by *partial graphs* [SE76]. In this scenario, every node can be *owned* by at most one component; other components can only *know* the node. This severely limits the number of possible splits.

It should be noted that, although we rely on the algebraic style of transformation, the results of this paper are not put into a categorical framework: instead, we use a concrete graph representation and rely on node identities to identify the common parts of graphs.

1.3 Motivating example

We illustrate the results on a simple example of a production chain. The whole-world, global rule system is given in Fig. 2. As always, the intuitive meaning of the rules is that a match is found for the left hand side (LHS), which is then replaced by the right hand side (RHS). The production chain consists of connected machines denoted by M —next→ M. The start machine creates some preliminary product denoted by P (start rule, Fig. 2a), and transfers it to the next machine (transfer rule, Fig. 2c). The next machine similarly works on the product (work rule, Fig. 2b) and passes it on to the next machine. Finally the last machine hands the final product to the client (denoted by C), if there is still a demanding client (deal rule, Fig. 2d). A sample start graph is given in Fig. 2e. Note that the same rule system potentially applies to different production chain systems with different configurations. For example, a system can have different number of machines in the production lane or even multiple parallel production lanes.

In Fig. 3a, we split our start graph into four subgraphs where each graph models the behaviour of a single machine. We also add some context information to be able to glue the subgraphs together (the right machine in each subgraph). Consequently, the effect of the rules given in Fig. 2 has to be divided over the split graphs. For example, the transfer rule should transfer the product from one subgraph to another. Therefore, we need to adapt our rule systems as well. For example, one way to decompose the transfer rule is shown in Fig. 3b: the top and bottom show split rules, and the middle shows the additional synchronisation interface as a subset of the union of the left and right hand side graphs.



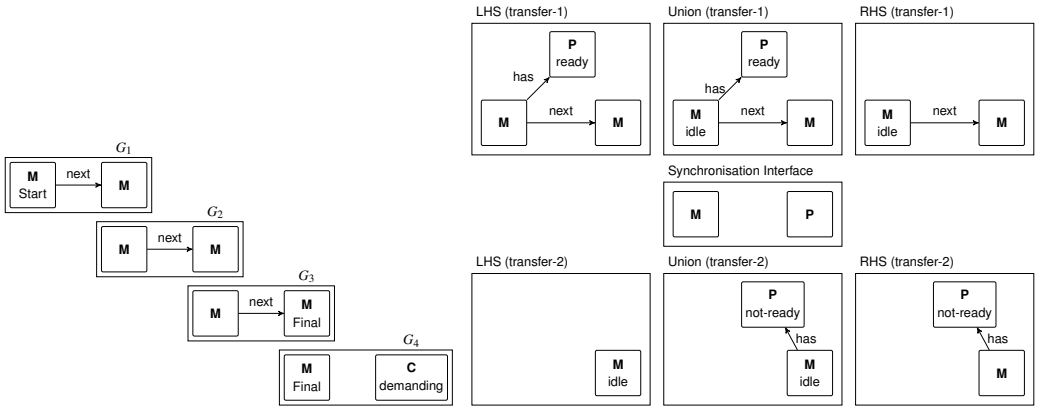
(a) Start rule (b) Work rule (c) Transfer rule

(d) Deal rule (e) Start graph

Figure 2: Running example: A simple production chain

The decomposition of the entire rule system of Fig. 2 will result in new rule systems for each of the components (defined by the split graphs). For example, each machine may have dedicated rules to reflect the different ways in which each of them perform their task. Moreover, only the rule system of the first component should contain the start rule, and similarly, the deal rule only needs to be included in the grammar of the last component. The resulting local rule systems can then be analyzed separately and their results can be composed.

Roadmap. The structure of the paper is as follows: Sect. 2 introduces the basic definitions; subsequently, Sect. 3 defines the composition of graphs, rules and transformations. Furthermore, it shows the soundness and completeness properties of the composition of transformations. In Sect. 4, we study partial graphs and show that they improve the size of the decomposed rule systems. Sect. 5 concludes the paper with an overview of related work and open questions. Due to space limitations proofs are omitted and can be found in [GR12].



(a) Split start graph (b) Split transfer rule

Figure 3: split start graph and split transfer rule

2 Basic definitions

Throughout this paper we assume global (countable) disjoint universes N of nodes, E of edges, and L of labels, with (also globally defined) functions $src, tgt\colon \mathsf{E} \to \mathsf{N}$ and $lab\colon \mathsf{E} \to \mathsf{L}$. In particular, for every combination of $v, w \in \mathsf{N}$ and $a \in \mathsf{L}$, there are assumed to be countably many $e \in \mathsf{E}$ such that $src(e) = v$, $tgt(e) = w$ and $lab(e) = a$.

Furthermore, we will use *structure-preserving functions* over $\mathsf{N} \cup \mathsf{E}$, which are functions $f = f_V \cup f_E$ with $f_V\colon V \to \mathsf{N}$ for some $V \subseteq \mathsf{N}$ and $f_E\colon E \to \mathsf{E}$ for some $E \subseteq \mathsf{E}$ such that $src(E) \cup tgt(E) \subseteq V$ and the following equations are satisfied:

$$src \circ f_E = f_V \circ src \restriction E \qquad tgt \circ f_E = f_V \circ tgt \restriction E \qquad lab \circ f_E = lab \restriction E$$

Definition 1 (graph) A graph is a finite set $G \subseteq \mathsf{N} \cup \mathsf{E}$, such that $src(G \cap \mathsf{E}) \cup tgt(G \cap \mathsf{E}) \subseteq G$. We often write V_G for $G \cap \mathsf{N}$ and E_G for $G \cap \mathsf{E}$, or just V and E if the index G is clear from the context. Given graphs G, H, a *graph morphism* $f\colon G \to H$ is a structure-preserving function such that $f(G) \subseteq H$. If f is bijective we also call it an *isomorphism* and G and H *isomorphic*.

As usual, rules are essentially defined by a left hand side (LHS) and a right hand side (RHS) graph. For the purpose of rule composition we use named rules; The names are taken from some global set which we do not further specify. As discussed in the introduction, we also add a an extra component, the *synchronisation interface*, which essentially declares which part of the rule is exposed to the environment.

Definition 2 (rule) A graph transformation rule is a tuple $p = \langle a, L, R, S \rangle$, consisting of a name a, a left hand side L, a right hand side R, and the synchronisation interface $S \subseteq L \cup R$.

We often denote the intersection $L \cap R$ by I. A transformation rule $p = \langle a, L, R, S \rangle$ is *applicable* to a graph G (often called the *host graph*) if there exists a *match* $m\colon L \to G$, which is a graph morphism satisfying the following conditions:

No dangling edges: For all $e \in E_G$, $src(e) \in m(L \setminus I)$ or $tgt(e) \in m(L \setminus I)$ implies $e \in m(L \setminus I)$;

No delete conflicts: $m(L \setminus I) \cap m(I) = \emptyset$.

The intuition is that the elements of G that are in $m(L)$ but not in $m(I)$ are scheduled to be deleted by the production. If a node is deleted, then so must its incident edges, or the result would not be a graph. Note that, due to the absence of delete conflicts, $m(L \setminus R) = m(L \setminus I) = m(L) \setminus m(I)$.

Given such a match m, the *application* of p to G is defined by extending m to a morphism $m'\colon L \cup R \to H'$, where $H' \supseteq G$ and all elements of $R \setminus L$ have distinct, fresh images under m', and defining $H = (G \setminus m(L \setminus I)) \cup m'(R \setminus I)$. We call H the *target* of the production; we write $G \xrightarrow{p,m'} H$ to denote that $m \subseteq m'$ is a valid match on G, giving rise to target graph H. Note that m' is not uniquely defined for a given p and m, due to the freedom in choosing the fresh images for $R \setminus I$; however, it is well-defined modulo isomorphism of H. In the remainder of this paper, we will use this freedom to make *a posteriori* assumptions about the actual choice of identities, for instance that they are fresh with respect to a set of nodes elsewhere in the system.

3 Compositionality

In the following we define graph and rule composition. Furthermore, we will show the soundness and completeness properties for the compositionality of transformations. We also lift the notion of decomposition to the rule system.

Definition 3 (graph and rule composition) Composition of two graphs G and H is defined as $G \cup H$. We refer to G and H as *local graphs* and $G \cup H$ as the *global graph*.

Two rules $p_1 = \langle a_1, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a_2, L_2, R_2, S_2 \rangle$ are *compatible* iff the have the same name ($a_1 = a_2$), and $S_1 \cap S_2 = (L_1 \cup R_1) \cap (L_2 \cup R_2)$ and their composition, denoted by $p_1 \cup p_2$, is defined as the composition of their components, i.e., $p_1 \cup p_2 = \langle a_1, L_1 \cup L_2, R_1 \cup R_2, S_1 \cup S_2 \rangle$. We denote the composition of two rules by $p_1 \cup p_2$. We refer to the rules p_1 and p_2 as *local rules* and to their composition, $p_1 \cup p_2$, as a *global rule*.

Definition 4 (rule dominance) Let $p_1 = \langle a_1, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a_2, L_2, R_2, S_2 \rangle$ be two rules. Then p_1 *dominates* p_2 iff $a_1 = a_2$, $L_1 = L_2$, $R_1 = R_2$ and $S_1 \supseteq S_2$. We denote this dominance relation by $p_1 \succeq p_2$.

Rule compatibility only imposes restrictions on the intersection of the synchronisation interfaces of two rules. Consequently, the following proposition is a direct implication of the rule compatibility and rule dominance definitions.

Proposition 1 *Let p_1, p_2 and p_2' be rules such that $p_2' \succeq p_2$. If p_1 and p_2 are compatible, then p_1 and p_2' are also compatible.*

Two compatible local rules are applied to two local graphs, and each results in a target graph. Now we show under which conditions the composition of two target graphs is identical to the target graph resulted from applying the transformation of the global rule.

Definition 5 (transformation compatibility) Given two compatible rules $p_1 = \langle a, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a, L_2, R_2, S_2 \rangle$, we call two transformations $G_1 \xrightarrow{p_1, m_1'} H_1$ and $G_2 \xrightarrow{p_2, m_2'} H_2$ compatible when the following conditions hold:

Synchronisation compatibility: $m_1'(S_1 \cap S_2) = m_2'(S_1 \cap S_2)$

Delete compatibility: $L_1 \setminus R_1 \cap m_1'^{-1}(G_2) \subseteq L_2$ and $L_2 \setminus R_2 \cap m_2'^{-1}(G_1) \subseteq L_1$.

Transfer compatibility: $m'(L \setminus R) \cap m'(L_1 \cap R_2) = \emptyset$ and $m'(L \setminus R) \cap m'(L_2 \cap R_1) = \emptyset$.

The first condition says that the image of the synchronisation interfaces must be identical, i.e., nodes with the same identity in the local rules must have the same image in the global graph. For example, in cases where a node is passed from one graph to another (by getting deleted from one local graph and getting created in the other), this condition makes sure that the created node shares the identity of the deleted one. Similarly, when the right hand sides of local rules share a node, they both create nodes with the same identities. The second condition states that if an element is deleted from the intersection of the two graphs by one rule, the same element must either be deleted or explicitly matched and preserved by the other rule. Finally, the transfer

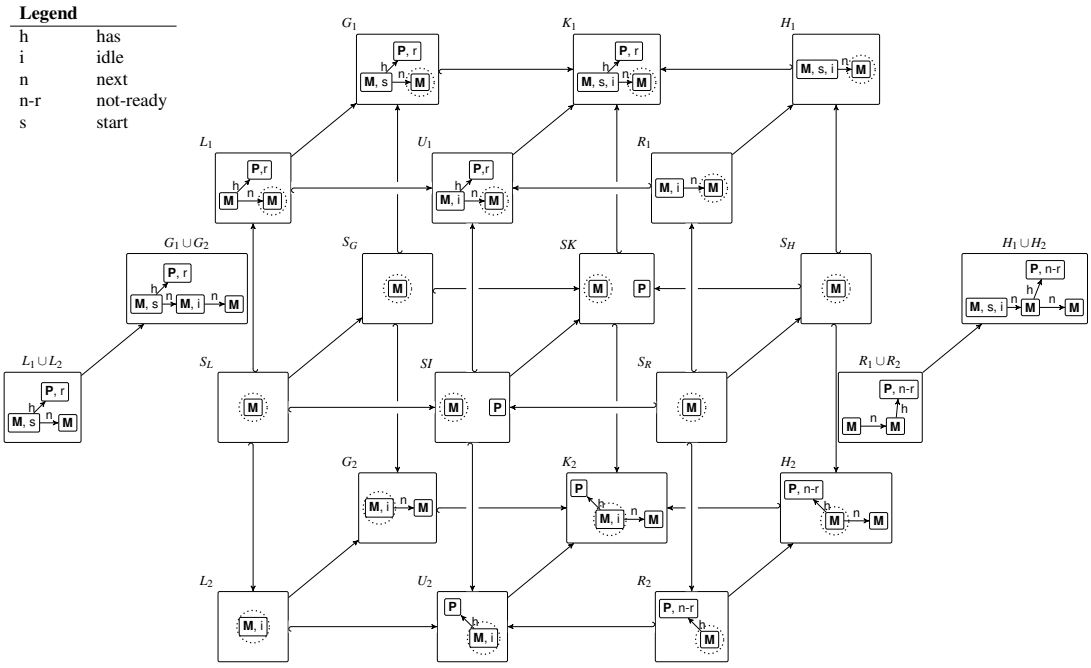


Figure 4: Running example: composition of the transfer rule

compatibility states that the nodes that are to be deleted by the global rule must not have images in common with the transferred nodes from one subgraph to another by applying the local rules. Note that the last condition is automatically satisfied for injective matches.

Synchronisation interface compatibility, and Prop. 1 imply the following proposition.

Proposition 2 *Let p_1, p_2, and p_2' be rules, and $G_1 \xrightarrow{p_1,m_1'} H_1$ and $G_2 \xrightarrow{p_2,m_2'} H_2$ compatible transformations. If $p_2' \succeq p_2$, then $G_1 \xrightarrow{p_1,m_1'} H_1$ and $G_2 \xrightarrow{p_2',m_2'} H_2$ are also compatible.*

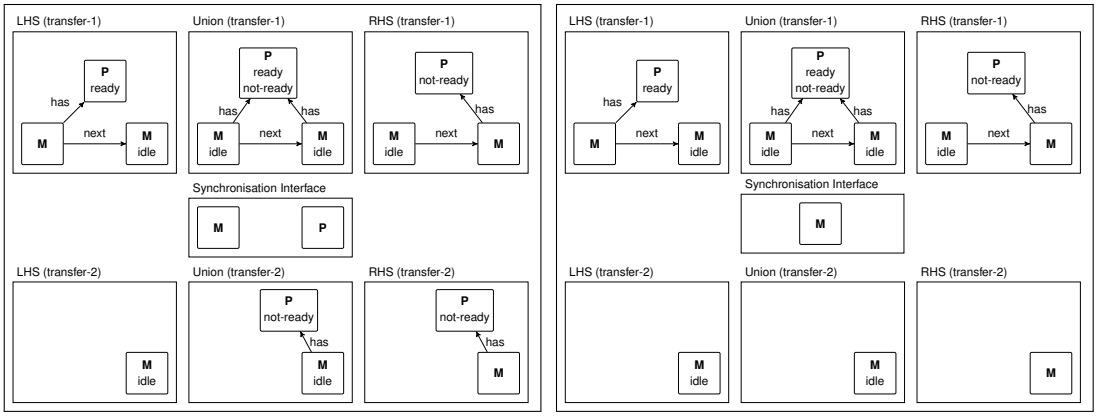
We can also deduce the following proposition directly from synchronisation compatibility.

Proposition 3 *Given two compatible transformations $G_1 \xrightarrow{p_1,m_1'} H_1$ and $G_2 \xrightarrow{p_2,m_2'} H_2$, $m_1' \cup m_2'$ is a function from $G_1 \cup G_2$ to $H_1 \cup H_2$.*

From now on, in cases where two transformations are synchronisation compatible, we use $m' = m_1' \cup m_2'$ for the combination of m_1' and m_2'. In the remainder of this section, we show that the composition of transformations is sound and complete.

Theorem 1 (Soundness) *Let $p_1 = \langle a, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a, L_2, R_2, S_2 \rangle$ be two compatible rules. If $G_1 \xrightarrow{p_1,m_1'} H_1$ and $G_2 \xrightarrow{p_2,m_2'} H_2$ are two compatible transformations, then there is a global transformation $G_1 \cup G_2 \xrightarrow{p_1 \cup p_2, m_1' \cup m_2'} H_1 \cup H_2$.*

The application of the split transfer rule (Fig. 3b) on G_1 and G_2 (shown in Fig. 3a) is illustrated in Fig. 4. The larger rectangles are graphs and the arrows between them are morphisms. Arrows of the form $\hookrightarrow$ denote subgraph relations (monomorphisms), and arrows with the shape $\rightarrow$ represent general morphisms. Due to space limitation, some of the labels are abbreviated (refer to



(a) split transfer rule 1 (b) split transfer rule 2

Figure 5: Running example: decomposition of the transfer rule

the legend for further clarification).

The top face of the cube shows the application of transfer-1 on G_1 and the bottom face represents the application of transfer-2 on G_2. In the middle of the cube, SI and SK are the synchronisation interface and its image, respectively. S_L and S_R are $L_1 \cap L_2$ and $R_1 \cap R_2$ respectively and S_G and S_H are their images. In all the morphisms in the picture, P s are always mapped to each other. In cases where two M s exist in a graph, nodes with dotted circles around them are mapped to each other. The application of the global transfer rule on $G_1 \cup G_2$ also results in the union of H_1 and H_2 as illustrated in the picture.

In the following, we also show that the decomposition of a graph transformation is complete. First we show a mechanism for decomposing rules.

Definition 6 (rule decomposition) Given a rule $p = \langle a, L, R, S \rangle$, we call two compatible rules p_1 and p_2 a *decomposition* of p if $p_1 \cup p_2 \succeq p$ and $V_{I_j} \subseteq V_I \cap V_{L_j}$, for $j = 1, 2$.

The conditions on the node set of the interfaces of the rules state that local rules cannot delete a node unless it is deleted by the global rule. In other words, if a node is preserved by the global rule and appears on the left hand side of the local rule, it must also be preserved by the local rule. The following proposition shows that the decomposition of a rule results in compatible transformations.

Proposition 4 *Let $G_1 \cup G_2 = G \xrightarrow{p,m'} H$ be a transformation and p_1 and p_2 a decomposition of p such that $L_1 = m'^{-1}(G_1)$ and $L_2 = m'^{-1}(G_2)$. Then $G_1 \xrightarrow{p_1,m'} H_1$ and $G_2 \xrightarrow{p_1,m'} H_2$ are compatible transformations.*

The completeness theorem directly follows from Prop. 4 and Th. 1.

Theorem 2 (completeness) *$G = G_1 \cup G_2 \xrightarrow{p,m'} H$ can be decomposed into two compatible transformations $G_1 \xrightarrow{p_1,m'_1} H_1$ and $G_2 \xrightarrow{p_2,m'_2} H_2$ such that $p_1 \cup p_2 \succeq p$, and $m' = m'_1 \cup m'_2$ and $H = H_1 \cup H_2$.*

Remember that we are ultimately interested in the behaviour of the global system, therefore we need to mimic the global behaviour by combining the behaviour of local systems. That means that whenever there is a global rule which is applicable on the global graph, we need to have two local rules whose combined effect is equivalent to the global rule. Moreover, we do not know the local host graphs in advance as they develop by local rule applications. This means that the local host graphs are not completely known and therefore local rule systems must contain all different possible ways of splitting the global rules. Therefore, for every rule we need to consider all different possible ways that its left hand side can be decomposed. The following definition specifies such a set of decomposed rules.

Definition 7 (rule split set) Let $p = \langle a, L, R, S \rangle$ be a rule. We call $\mathbb{U}_p$ the *split set* of p, if for L_1, L_2 such that $L_1 \cup L_2 = L$ there exist $p_1 = \langle a, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a, L_2, R_2, S_2 \rangle$ in $\mathbb{U}_p$ such that p_1 and p_2 are a decomposition of p.

The decomposition of a rule system comes down to finding the split set of all the global rules. However, the number of all different splits is an exponential function of the number of nodes and edges, as each element of the global rule can appear in the first, the second or both local rules.

Definition 8 (rule system decomposition) The decomposition of a rule system $\mathfrak{R}$ is a set containing a $\mathbb{U}_p$ for every rule $p \in \mathfrak{R}$.

Note that the decomposed rule systems constitutes a new rule system which on its own can be subjected to further decompositions. This can result in decompositions with more than two local components.

Back to our running example, two possible ways of splitting the transfer rule (Fig. 2c) are shown in Fig. 5. Both rules have the same left hand sides and the combined effect of both of them on $G_1 \cup G_2$ is equivalent to $H_1 \cup H_2$. Therefore, the split set of transfer rule only needs to have one of them.

Besides, observe that there is redundancy between transfer-1 and transfer-2 of both splits. This redundancy originates from the fact that both splits have a node in common, which is one of the machines. This redundancy is inherent in the structure of the graph. For instance, in our running example naturally we like to decompose our start graph in such a way that each local graph only contains one machine. However, we needed to add the second machine to the first local graph (Fig. 3a) because otherwise the edges between the machines would not belong to any of the local graphs. That means that some machines needed to appear in two graphs. In the next section we relax the definition of graphs so that, firstly, we can avoid the redundancy caused by the inherent structure of graphs, and secondly, the decomposition of the global rules lead to considerably fewer number of local rules.

4 Partial graphs

In this section, inspired by [SE76], we propose partial graphs as opposed to graphs for the local components.

Definition 9 (partial graph) A *partial graph* is a finite set $G \subseteq \mathsf{N} \cup \mathsf{E}$, such that $src(G \cap \mathsf{E}) \subseteq G$. The *completion* of a partial graph G, denoted by $\overline{G}$, is $G \cup tgt(G \cap E)$. Given partial graphs G, H, a *partial graph morphism* $f\colon G \to H$ is a structure-preserving function such that $f(G) \subseteq H$.

Similar to graphs, we often write V_G for $G \cap \mathsf{N}$ and E_G for $G \cap \mathsf{E}$, or just V and E if the index G is clear from the context. Note that a partial graph can contain an edge which is dangling from its target side. Therefore, any graph is also a partial graph. We refer to the missing target nodes of the dangling edges of a partial graph as its *virtual* nodes. The virtual nodes of a partial graph are included in its completion, which implies that the completion of a partial graph is a graph. Note that the partial graph morphism is defined exactly similar to graph morphisms.

In line with the intuition behind proposing partial graphs to avoid redundancy between the local components we define the following definition.

Definition 10 (disjoint partial graphs) Two partial graphs G_1 and G_2 are disjoint if they have no node in common, i.e., $V_{G_1} \cap V_{G_2} = \emptyset$.

Note that as edges cannot be dangling from their source sides, therefore two disjoint graphs do not have any edge in common.

In the following we give the definition of a partial rule, and partial rule applicability. Furthermore, we also discuss the applicability conditions of partial rules.

Definition 11 (partial rule) A partial rule is a tuple $p = \langle a, L, R, S \rangle$ where a is its name and L and R are partial graphs, and $S \subseteq (\overline{L} \cup \overline{R})$ is a graph.

Let $p = \langle a, L, R, S \rangle$ be a partial rule and G a partial graph. Then p is *applicable* to G if there exists a match $m\colon L \to G$ satisfying the following conditions

No dangling edges: For all $e \in E_G$, if $src(e) \in m(L \setminus I)$ then $e \in m(L \setminus I)$; and if $tgt(e) \in m(L \setminus (I \cup S))$ then $e \in m(L \setminus I)$;

No delete conflicts: $m(L \setminus R) \cap m(I) = \emptyset$.

Similar to the application of a transformation rule to a graph, the elements of G that are in $m(L)$ but not in $m(I)$ are scheduled to be deleted by the production. Moreover, if a node is deleted, then so must its outgoing edges. In contrast to the application of a rule to a graph, the deletion of a node leads to the deletion of its incoming edges only if the node is not in the image of the synchronisation interface.

Given such a match m, the *application* of p to G is defined by extending m to a morphism $m'\colon L \cup R \to H'$, where $H' \supseteq G$ and all elements of $R \setminus L$ have distinct, fresh images under m', and defining $H = (G \setminus m(L \setminus I)) \cup m'(R \setminus I)$. Before we define the composition of partial graphs and rules, we define the condition under which two partial rules are compatible.

Definition 12 (partial rule compatibility and composition) Two partial rules $p_1 = \langle a_1, L_1, R_1, S_1 \rangle$ and $p_2 = \langle a_2, L_2, R_2, S_2 \rangle$ are compatible if they have the same name, i.e., $a_1 = a_2$, and both L_1 and L_2, and R_1 and R_2 are disjoint, also $S_1 \cap S_2$ must be equal to $(\overline{L_1} \cup \overline{R_1}) \cap (\overline{L_2} \cup \overline{R_2})$. Then their composition is $\langle L_1 \cup L_2, R_1 \cup R_2, S_1 \cup S_2 \rangle$.

Compared with the rule compatibility condition of graph rules, the partial rule compatibility enforces two extra conditions, i.e., the left and right hand sides of the partial rules to be disjoint. The intuition behind these extra constraints is that if two compatible partial rules are applied to disjoint partial graphs the results will also be disjoint. This is of course desirable as it guarantees the lack of redundancy between all related pairs of local states. This property is shown in the following proposition.

Proposition 5 *Let G_1 and G_2 be compatible partial graphs and p_1 and p_2 compatible partial rules. If $G_1 \xrightarrow{p_1,m'_1} H_1$ and $G_2 \xrightarrow{p_2,m'_2} H_2$ are two transformations which satisfy synchronisation compatibility, then H_1 and H_2 are also compatible.*

The compatibility of two partial graph transformations is defined exactly the same way as it is defined for graph transformations, namely, two partial graph transformations are compatible if they satisfy delete compatibility, synchronisation compatibility, and transfer compatibility. Through an analogous line of reasoning to the one for graphs, the soundness and completeness properties for partial graphs hold as well. The split set of a partial graph rule can also be defined similarly to the split set of graph rules given in Def. 7. However, the size of the split set of a partial rule is considerably smaller than that of a graph rule. This is because we can always decompose a partial graph into two disjoint partial graphs. In fact, decomposing a global partial graph to two disjoint local partial graph comes down to just choosing which local graph a node belongs to. Then each edge must reside in the local partial graph where its source node has gone. However, in case of graphs, each edge and each node can belong to either of the local graphs or even both.

5 Conclusion

We have defined a notion of composition for graphs and graph transformation rules which allows passing subgraphs between components. This was done by equipping every graph transformation rule with a synchronisation interface, which declares the part of the rule that is exposed to the environment. Rules and transformations can be composed when they have compatible synchronisation interfaces.

Moreover, we defined the behaviour-preserving (de)composition on the level of rule systems. We also showed that the decomposition of a rule system can give rise to a very large number of decomposed rules. To tackle this problem, we also studied a scenario in which components are specified by partial graphs and partial graph rules.

5.1 Related work

The concepts of graph and rule composition, with the appropriate notions of soundness and completeness, were introduced in [Ren10] and later generalised in [Hei10]. With respect to those papers, the variation studied here offers a more powerful notion of composition, in which nodes and edges can be deleted in one component and simultaneously created in the other. On the other hand, we have only presented the approach for a single concrete category of (multi-sorted) graphs. The partial graph variant in Sect. 4 is inspired by [SE76].

In addition, there are a number of other approaches to introduce aspects of compositionality into graph transformation.

Synchronised Hyperedge Replacement. This is a paradigm in which graph transformation rules (more specifically, hyperedge replacement rules) can be synchronised based one the adjacency of their occurrences within a graph; see [HM01, FHL+06]. The synchronised rules are not themselves understood as graph transformation rules, and consequently the work does not address the type of compositionality issues that we have studied here. Still, it is interesting to see whether SHR synchronisation can be understood as a special type of composition in our sense.

History-Dependent Automata. This is a behavioural model in which states are enriched with a set of *names* (see [MP05] for an overview). Transitions expose names to the environment, and can also record the deletion, creation and permutation of names. HD-automata can be composed while synchronising their transitions: this provides a model for name passing. Transition systems induced by graph transformation rules can be understood as a variant of HD-automata where the states are enriched with graphs rather than just sets, and the information on the transitions is extended accordingly.

Rule amalgamation and distributed graph transformation. Studied in [BFH87] and later, more extensively, in [Tae97], the principle of rule amalgamation provides a general mechanism for rule (de)composition. This is a sub-problem of the one we have addressed here, as we study composition of the graphs as well as the rules. Our notion of rule composition is actually a generalisation of rule amalgamation, as local rules do not have to synchronise on deletions and creations.

Borrowed contexts. Like our paper, the work on borrowed contexts [EK06, BEK06] uses a setting where only part of a graph is available, and studies the application of rules to such sub-graphs in a way that is compatible with the original, reductive semantics. In contrast to our approach, however, they do not decompose rules: instead, when a rule is applied to a graph in which some of the required structure ("context") for the match is missing, this is imported ("borrowed") as part of the transformation. As a result, in this paradigm the subgraphs grow while being transformed, incorporating ever more context information. This is quite different from the basic intuitions behind our approach.

Summarising, where only rules are (de)composed in rule amalgamation, and only graphs in borrowed contexts, in our approach both rules and graphs are subject to (de)composition.

Compositional model transformation. [BHE09] studies a notion of compositionality in model transformation. Though on the face of it this sounds similar, in fact they study a different question altogether, namely whether a transformation affects the *semantics* of a model (given as a separate mapping to a semantic domain) in a predictable (compositional) manner. This is in sharp contrast with our work, which rather addresses the compositionality of the graph transformation framework itself.

Graph Transformation Units. The graph transformation units exemplified in [KBK01], also provide a notion of composition. However, this work takes the form of an explicit structuring mechanism of local graph transformation systems, called Units. The question of equivalence of a monolithic graph transformation system and a composition of local units is not addressed in this approach.

5.2 Future work

This paper continues our investigation into compositional graph transformation, and although it is a step forward with respect to [Ren10] in that the notion of composition is more general, it is simultaneously a step backward in that the technical development in this paper is within a single concrete category only. We plan to investigate whether our results carry over to adhesive categories [LS04] as a foundation for graph transformation.

Negative application conditions (NACs) as introduced in [HHT96] have shown to be very useful in practice. It will be interesting to extend our notion of compositionality to rules with NACs, in particular with respect to the soundness and completeness properties.

The left hand sides of the local rules are usually smaller than the one of the global rules. This can potentially lead to many rule applications on the local components. We intend to investigate this potential explosion of the local state spaces and devise some heuristics for the decompositions which are immune from this possible explosion. A major challenge is that, even for the variant with partial graphs ([SE76]), the number of decomposed rules can grow very large (exponential in the size of the LHS). We plan to study conditions under which the number of decomposed rules stays within bounds. This can be done for instance by associating a notion of *hierarchy* with the graphs, as in [DHP02]: the idea is that the graph nodes that are hierachically contained in another node must reside in the same component, and so there are fewer possible splits of a rule.

Bibliography

[BEK06] P. Baldan, H. Ehrig, B. König. Composition and Decomposition of DPO Transformations with Borrowed Context. In Corradini et al. (eds.), *Third International Conference on Graph Transformations, (ICGT)*. LNCS 4178, pp. 153–167. Springer, 2006.

[BFH87] P. Boehm, H.-R. Fonio, A. Habel. Amalgamation of Graph Transformations: A Synchronization Mechanism. *J. Comput. Syst. Sci.* 34(2/3):377–408, 1987.

[BHE09] D. Bisztray, R. Heckel, H. Ehrig. Compositionality of Model Transformations. In Aldini et al. (eds.), *3rd International Workshop on Views On Designing Complex Architectures (VODCA)*. ENTCS 236, pp. 5–19. 2009.

[DHP02] F. Drewes, B. Hoffmann, D. Plump. Hierarchical Graph Transformation. *J. Comput. Syst. Sci.* 64(2):249–283, 2002.

[EK06] H. Ehrig, B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *Math. Structures in Computer Science* 16(6):1133–1163, 2006.

[FHL+06] G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, E. Tuosto. Synchronised Hyperedge Replacement as a Model for Service Oriented Computing. In Boer et al. (eds.), *Formal Methods for Components and Objects (FMCO)*. LNCS 4111, pp. 22–43. Springer, 2006.

[GR12] A. H. Ghamarian, A. Rensink. Graph Passing in Graph Transformation. Technical report TR-CTIT-12-04, Centre for Telematics and Information Technology, University of Twente, 2012.

[Hei10] T. Heindel. Structural Decomposition of Reactions of Graph-Like Objects. In Aceto and Sobocinski (eds.), *SOS*. EPTCS 32, pp. 26–41. 2010.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundam. Inform.* 26(3/4):287–313, 1996.

[HM01] D. Hirsch, U. Montanari. Synchronized Hyperedge Replacement with Name Mobility. In Larsen and Nielsen (eds.), *Concurrency Theory (CONCUR)*. LNCS 2154, pp. 121–136. Springer, 2001.

[KBK01] H.-J. Kreowski, G. Busatto, S. Kuske. GRACE as a unifying approach to graph-transformation-based specification. *Electronic Notes in Theoretical Computer Science* 44(4):1 – 15, 2001.

[LS04] S. Lack, P. Sobocinski. Adhesive Categories. In Walukiewicz (ed.), *Foundations of Software Science and Computation Structures, (FoSSaCS)*. LNCS 2987, pp. 273–288. Springer, 2004.

[MP05] U. Montanari, M. Pistore. History-Dependent Automata: An Introduction. In Bernardo and Bogliolo (eds.), *Formal Methods for Mobile Computing*. LNCS 3465, pp. 1–28. Springer, 2005.

[MPW92] R. Milner, J. Parrow, D. Walker. A Calculus of Mobile Processes, I. *Inf. Comput.* 100(1):1–40, 1992.

[Ren10] A. Rensink. Compositionality in graph transformation. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming: Part II*. ICALP'10, pp. 309–320. Springer-Verlag, Berlin, Heidelberg, 2010.

[SE76] H. J. Schneider, H. Ehrig. Grammars on Partial Graphs. *Acta Inf.* 6:297–316, 1976.

[Tae97] G. Taentzer. Parallel High-Level Replacement Systems. *TCS* 186(1-2):43–81, 1997.