Parallelism in AGREE Transformations

Andrea Corradini[1(✉)], Dominique Duval[2(✉)], Frederic Prost[3(✉)], and Leila Ribeiro[4(✉)]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
andrea@di.unipi.it
[2] LJK - Université Grenoble Alpes and CNRS, Grenoble, France
Dominique.Duval@imag.fr
[3] LIG - Université Grenoble Alpes and CNRS, Grenoble, France
Frederic.Prost@imag.fr
[4] INF - Universidade Federal Do Rio Grande Do Sul, Porto Alegre, Brazil
leila@inf.ufrgs.br

Abstract. The AGREE approach to graph transformation allows to specify rules that clone items of the host graph, controlling in a fine-grained way how to deal with the edges that are incident, but not matched, to the rewritten part of the graph. Here, we investigate in which ways cloning (with controlled embedding) may affect the dependencies between two rules applied to the same graph. We extend to AGREE the classical notion of parallel independence between the matches of two rules to the same graph, identifying sufficient conditions that guarantee that two rules can be applied in any order leading to the same result.

1 Introduction

Graph Transformations (GT) are very much used to specify systems where concurrency and non-determinism are present. For instance GT has been used to model the evolution of biological systems [6], chemical reactions [15] and also concurrent models of computations [8]. From this perspective a major concern is to investigate how the application of different rules may affect each other. There are two classical questions:

1. **(parallel independence)** Given two rules with matches in the same graph G, are they independent? That is, can they be applied in any order (or even in parallel) with the same result?
2. **(sequential independence)** Given a sequence of two rewrite steps, is the second step independent of the first? That is, could the second rule be applied first, followed by the application of the first rule, leading to the same result?

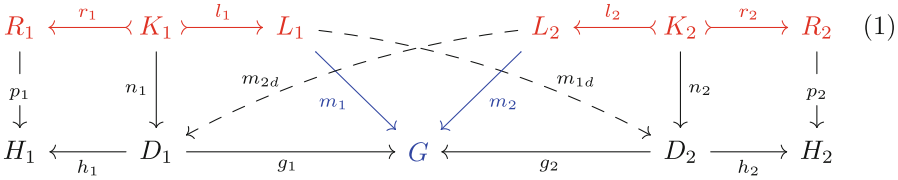
In this paper we shall consider parallel independence only. In the classical setting, where typically rules are injective, two rewrite steps are parallel independent if their matches overlap only on items that are preserved by both. In other words, they *are not* parallel independent if there is a conflict of the following types:

This work has been partly funded by projects CLIMT (ANR/(ANR-11-BS02-016), VeriTeS (CNPq 485048/2012-4 and 309981/2014-0), PEPS égalité (CNRS).

R. Echahed and M. Minas (Eds.): ICGT 2016, LNCS 9761, pp. 37–53, 2016.
DOI: 10.1007/978-3-319-40530-8_3

delete-delete: two rules try to delete the same item. In this case the conflict is symmetric and it means that the rules are mutually exclusive.
preserve-delete: a rule deletes an item that is preserved by the other. In this situation the conflict is asymmetric because the application of the rule that deletes the item prevents the other to occur, but not the other way around.

Parallel independence is usually formalized, in the algebraic approaches to GT, making reference to the following diagram. The rewrite step using rule 2 at match m_2 is said to be (parallel) independent from the rewrite step using rule 1 at match m_1 if there exists a morphism m_{2d} such that $m_2 = g_1 \circ m_{2d}$ (and symmetrically for rule 1). That is, it is still possible to apply rule 2 after rule 1 has been applied, using the "same" match, and in this case the *Local Church-Rosser Theorem* shows that the resulting graph is the same.



Those problems have been studied in many GT approaches: double pushout (DPO) [5], single pushout [12], sesqui-pushout (SqPO) [4], reversible sesqui-pushout [7], with negative application conditions [13], borrowed contexts [1] and nested application conditions [11]. To our knowledge, in all these approaches (but for [7]) rules are required to be *linear*, i.e. both the left- and the right-hand side have to be monomorphisms. In this paper we address the problem of parallel independence for the AGREE approach [3]. The main feature of AGREE rewriting is the ability to clone matched items, like in the SqPO approach that it extends, but with the possibility of specifying how edges incident to the image of the match can be handled. Because of this feature the analysis of parallel independence becomes quite more complex than in the other approaches, since new kinds of conflicts between rewrite steps arise.

The paper is organized as follows. We start with an informal introduction to AGREE in Sect. 2, showing how, from a programmer point of view, AGREE rewrite rules can be specified by exploiting the ability both to clone items, and to control the embedding of the preserved or cloned items in the context. In Sect. 3 we recall from [3] the formal definition of AGREE rewriting. Then we present in Sect. 4, through several canonical counter-examples, how new types of conflicts may arise due to cloning. Those counter-examples will motivate the assumptions needed for the main result that is stated and proved in Sect. 5. Finally we conclude and sketch future developments in Sect. 6.

2 Controlling the Embedding in AGREE

AGREE is a GT approach: states are represented by graphs and transitions are specified by rules. When specifying a transition between states using an AGREE

rule (see the left diagram of (2)), the designer describes with the *left-hand side* L the items that must be present to trigger the application of the rule. The morphism l from the *gluing graph* K to L describes which items of L will be preserved, cloned or deleted. More precisely an item of L is *deleted* if it is not in the image of l, it is *preserved* if it is the image of exactly one item of K along l, and it is *cloned* if it is the image of more than one item along l. The morphism r to the *right-hand side* R, that we assume to be mono in this paper, defines the items that will be created, i.e. those not in the image of r. Finally *the embedding component* T_K, which is typical of AGREE, is used to describe how the preserved or cloned items are embedded in the rest of the state graph.

To apply a rule to a graph G (see the right diagram of (2)), first an image of L in G has to be found (a *match*).[1] Then, basically, all items from G are removed, preserved or cloned according to the rule (using L, K and T_K), and new items are added according to r. In the following, we explain intuitively how to specify the embedding component of a rule, and how rule application is performed in the case of typed graphs. The formal definitions will be given in Sect. 3.

$$L \xleftarrow{l} K \xrightarrow{r} R \qquad\qquad \begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & \textit{deletion+cloning} & \downarrow{\scriptstyle n} & \textit{creation} & \downarrow{\scriptstyle p} \\ G & \xleftarrow{g} & D & \xrightarrow{h} & H \end{array} \qquad (2)$$

$$\begin{array}{c} K \\ \downarrow{\scriptstyle t} \\ T_K \end{array}$$

The embedding component describes how to handle the context, i.e. the part of G that is not in the image of the match. To specify this component, we first build a graph containing the gluing graph K and all possible ways in which it is connected to the rest. This is done by a construction called *partial map classifier* for K, denoted by $T(K)$ [2]. This graph contains the following classes of items:

(i) **preserved items:** K (the gluing graph);
(ii) **independent context items:** a copy of the type graph (to describe the part of the state graph that is not touched by the rule application);
(iii) **gluing context edges:** one instance of each type of edge (from the type graph) for each pair of nodes of K; and
(iv) **embedding context edges:** one instance of each type of edge (from the type graph) for each pair made of a node in class (i) and a node in class (ii).

We call the items in classes (ii) to (iv) $\star$-items. They are used to represent the context: given any graph X with a map to $T(K)$, we can *classify* X's items into items that represent K's items (whose images are in (i)) and items that are context (whose images are in (ii)–(iv)). Now, to obtain the embedding component of a rule, one can specify how the preserved or cloned items are embedded in the rest of the graph by removing from $T(K)$ all items that should not be maintained when the rule is applied, or adding some items to obtain more copies of specific elements of the state graph.

For example, consider the graphs in Fig. 1. TG is a type graph having two nodes and two different edges. The items to be preserved/cloned by a rule are

[1] In the AGREE approach matches have to be monic.

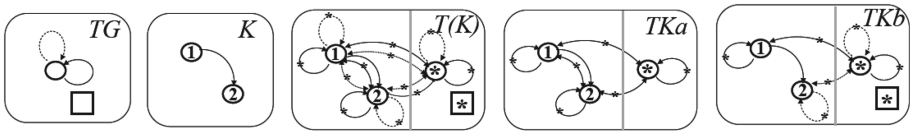


Fig. 1. Embedding Component Examples

shown in graph K (here the left- and right-hand sides are not important since we only want to illustrate the embedding component). To obtain $T(K)$ we follow the steps previously described. As a graphical notation, we use arrows with tips at both sides to depict two edges, one in each direction, all $\star$-items are marked with $\star$ and there is a vertical bar dividing $T(K)$ as follows: we put to the right of the bar the items of (ii) (the copy of the type graph), to the left the items of (i) (graph K) and (iii) (all possible kinds of edges among K's nodes), and connect the left and right sides with items of (iv) (all possible types of edges between items of K and of the copy of the type graph). Choosing for a rule any embedding component TK that does not include all the items of (ii) (or adds some elements to (ii)) would result in a rule with *non-local effects*. For example, if TKa is the embedding component of a rule with gluing graph K, the application of this rule would remove from a graph all square nodes and dashed edges, even if not in the image of the match, because TKa specifies that the context should not have those items. Instead the embedding component TKb has a *local effect* since the part to the right side of the bar is a copy of TG. The application of this rule would remove dashed edges between node ① and the rest of the graph, and solid edges between node ② and the rest of the graph (only one edge between nodes ① and ② would remain, since this edge is in K).

In the rest of this section we consider *local rules* only, that is, the embedding component must include a copy of the type graph.[2] For simplicity we also assume that the embedding component is included in $T(K)$ (even if the formal development does not require it), thus it is obtained from $T(K)$ by deleting only items belonging to (iii) and (iv). For a simpler graphical representation of the embedding component, to the right of the vertical bar we draw only the nodes of the type graph (considering the edges implicitly there), since only the nodes of (ii) are needed to specify how the gluing graph is connected to the context.

To illustrate the AGREE approach and the effect of the embedding component, we will model the generation of Sierpinski triangles. A Sierpinski triangle is a well-known fractal in which an equilateral triangle is divided into smaller equilateral triangles in a controlled way, given by a rule like the one depicted in Fig. 2(a). Applying this rule repeatedly and fairly a convenient number of times leads to shapes like the ones shown in Fig. 2(b).

In [16] the generation of Sierpinski triangles was used as a case study to compare different graph transformation tools. There, a triangle was modeled as a graph with three nodes and three edges, and each step of the generation deleted

[2] We refer the interested reader to [3] for a formal definition of locality.

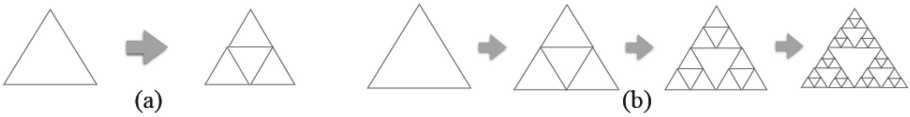


Fig. 2. (a) Sierpinski generation rule (b) Sierpinski triangles generation

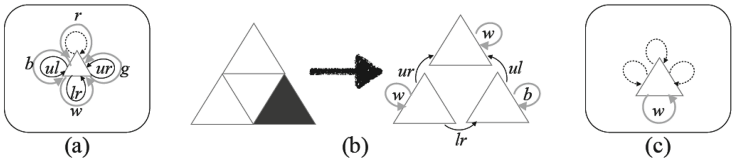


Fig. 3. (a) Type Graph (b) Graph Representation (c) Start Graph

edges and created new nodes and edges. Here, instead, we consider a triangle as a single node, and each step will split (or clone) the triangle into three other ones and create suitable connections (edges) among them. To control how many times the splitting process should occur, we use a special kind of edge: the number of dashed loops on a node indicates how many times the splitting process can be applied. To make the example more interesting, we will color the triangles: a gray loop on a triangle indicates its color (b for black, g for green, r for red and w for white). Moreover, there will be three different edges that are used to connect triangles, called *ur* (up-right), *ul* (up-left) and *lr* (left-right). The corresponding type graph is shown in Fig. 3(a). Figure 3(b) depicts a Sierpinski triangle and its corresponding graph representation, and Fig. 3(c) presents a possible start state for the generation of Sierpinki triangles of order 3.

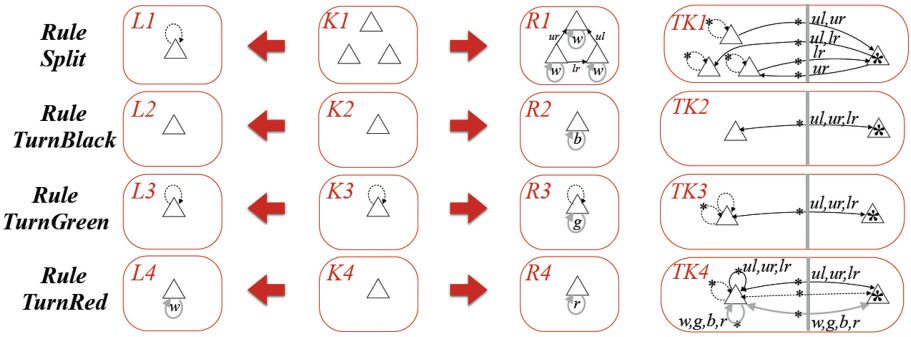


Fig. 4. Rules for generation and coloring of Sierpinski triangles.

To model the splitting of the triangle we use rule ***Split*** of Fig. 4: whenever there is a triangle that may be split (has a dashed loop) $L1$, it is split and new

connections between the copies are created. Also note that the new triangles are colored with white in $R1$. The embedding component is graph $TK1$. Remind that we are using local rules, thus only the nodes of the type graph of Fig. 3(a) are drawn to the right of the bar (the edges are implicitly there). Since $TK1$ does not contain gray loops on the left node, any possible color of the matched node is deleted by the application of the rule. The same happens in all other rules, but for ***TurnRed***. The last three rules change the color of the triangle to black, green or red in slightly different ways. Rule ***TurnBlack*** adds a black color to the matched triangle after removing any colored or dashed loop, thus preventing further splitting. Rule ***TurnGreen*** changes the color of a triangle (of any color) to green and requires the presence of at least one dashed loop, and preserves all dashed loops. Finally rule ***TurnRed*** changes the color of a triangle to red, if the triangle was white, and keeps all existing connections (the embedding component of this rule is the partial maps classifier of $K4$).

When a match $m : L \to G$ from an AGREE rule to a graph G is found, this match induces a partition of G's items into the following classes:

(i) **preserved/cloned items:** items in the image of K,
(ii) **independent context items:** the items that are neither in the image of L nor are connections to items in the image of L,
(iii) **gluing context edges:** edges not in the image of L that connect nodes in the image of K,
(iv) **embedding context edges:** edges not in the image of L that connect nodes in the image of K to other nodes in G (not in the image of L),
(v) **deleted items:** items in the image of L and not in the image of K,
(vi) **dangling edges:** edges that connect nodes marked for deletion (in class (v)) to other nodes (not in (v)).

The embedding component TK of a local rule is a subgraph of $T(K)$ that includes K, it specifies that items in classes (i) and (ii) remain untouched. The control of the embedding is performed on items of classes (iii) and (iv): edges of types that are in $T(K)$ and not in TK must be removed/can not be cloned. An AGREE rule application can be constructed by the following steps:

Deletion: delete from G all items that are in classes (v) and (vi); delete all items that are in classes (iii) and (iv) of G and whose type is not in TK;
Cloning/Preserving: clone or preserve all items of (i) according to $l : K \to L$, and all edges from (iii) and (iv) according to TK (for every node that is cloned, clone all edges connected to this node whose type is in TK);
Creation: add new items according to $r : K \rightarrowtail R$.

Two examples of derivations using AGREE rules are shown in Fig. 5. Starting with graph $G1$ rule ***Split*** is applied (to the right), deleting one of the dashed loops of the triangle and splitting the triangle in three (cloning also the rest of the dashed loops and creating a white-loop in each of the resulting triangles). To the left, rule ***TurnBlack*** is applied, removing all the dashed arrows from the triangle. These extra deletion effects are specified in the corresponding embedding components (see $TK2$ in Fig. 4).

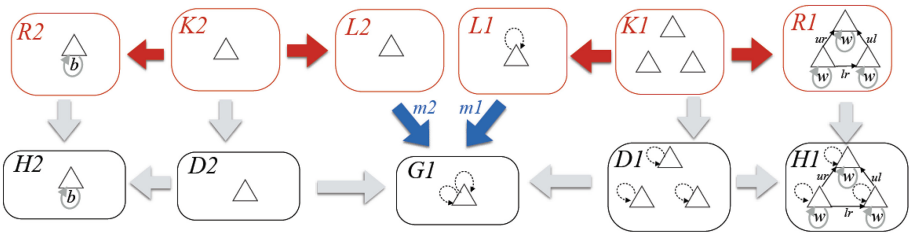


Fig. 5. AGREE rewrite steps using rules ***TurnBlack*** (left) and ***Split*** (right)

3 The AGREE approach to graph transformation

In this section we recall basic definitions of the AGREE approach to rewriting [3]. We assume the reader to be familiar with categorical notions used in the algebraic approaches to GT (including pushouts, pullbacks and their properties).The following definition will be useful in the technical development in Sect. 5.

Definition 1 (reflection). *Given arrows* $A \rightarrowtail C \leftarrow B$*, we say that* (the image of) A is reflected identically by f (to B) *if the square below to the right is a pullback for some mono* $A \rightarrowtail B$ *or equivalently if the pullback object of* f *and* m *is isomorphic to* A*.*

$$\begin{array}{ccc} A & \xrightarrow{id} & A \\ \downarrow & \lrcorner & \downarrow m \\ B & \xrightarrow{f} & C \end{array}$$

Intuitively, this means that f is an iso when restricted to the image of A. If objects are concrete structures like graphs, then every item of the image of A in C has exactly one inverse image along f in B.

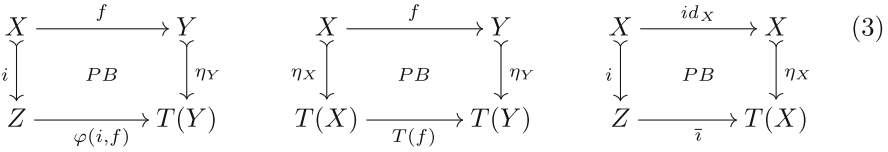
We assume that the category in which GT is performed has *partial maps classifiers* [2] (needed for the definition of AGREE rewriting [3]) and is *adhesive*, the latter assumption being standard for the results about parallelism [10,14].

Definition 2 (partial map classifier). *Let* $\mathbf{C}$ *be a category with pullbacks along monos. A* partial map *over* $\mathbf{C}$*, denoted* $(i, f) : Z \rightharpoonup Y$*, is a span* $(i : X \rightarrowtail Z, f : X \rightarrow Y)$ *in* $\mathbf{C}$ *with* i *mono, up to the equivalence relation* $(i', f') \sim (i, f)$ *when there is an isomorphism* h *with* $i' \circ h = i$ *and* $f' \circ h = f$*. Category* $\mathbf{C}$ *has a* partial map classifier (T, η) *if* T *is a functor* $T : \mathbf{C} \rightarrow \mathbf{C}$ *and* η *is a natural transformation* $\eta : Id_{\mathbf{C}} \dot{\rightarrow} T$*, such that for each object* Y *and each partial map* $(i, f) : Z \rightharpoonup Y$ *there is a unique arrow* $\varphi(i, f) : Z \rightarrow T(Y)$ *such that* (i, f) *is a pullback of* $(\varphi(i, f), \eta_Y)$ *(see the left diagram of (3)).*[3]

Then η_Y is mono for each Y, T preserves pullbacks, and η is *cartesian*, which means that for each $f : X \rightarrow Y$ the span (η_X, f) is a pullback of $(T(f), \eta_Y)$. For

[3] Intuitively, a partial map classifier provides a bijective correspondence between partial maps over $\mathbf{C}$ from object Z to Y and arrows of $\mathbf{C}$ from Z to $T(Y)$, given by $[(i, f)] \iff \phi(i, f)$, as described by the left diagram of (3).

each mono $i : X \rightarrowtail Z$ let $\bar{\imath} = \varphi(i, id_X)$; then $\bar{\imath}$ is characterized by the fact that (i, id_X) is a pullback of $(\bar{\imath}, \eta_X)$ (right diagram of (3)).



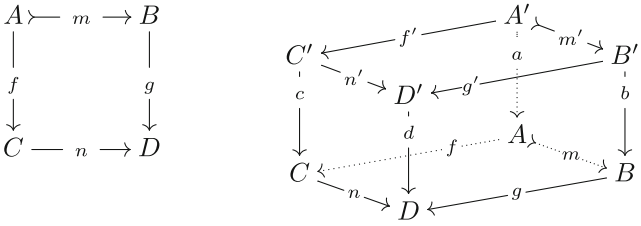
(3)

By composing the right and middle squares we get the left one, which proves that for each partial map $(i, f) : Z \rightharpoonup Y$:

$$\varphi(i, f) = T(f) \circ \bar{\imath} \tag{4}$$

For the definition of adhesivity, we stick to the seminal work [14]. Since then adhesivity has been generalized in several variants and sometimes in subtly different ways: for a recollection of such notions the reader is referred to [10].

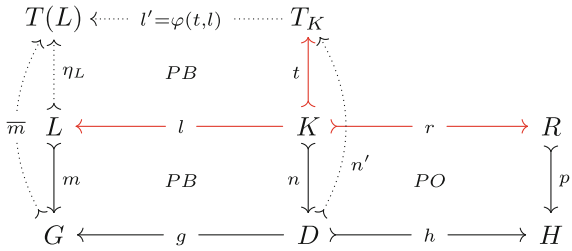
Definition 3 (adhesive category). *A category* $\mathbf{C}$ *is* adhesive *if it has all pullbacks, pushouts along monos, and if each pushout along a mono, like the square to the left below, is a* Van Kampen square*, i.e. if for any commutative cube as below to the right, where the pushout is the bottom face and the back faces are pullbacks, we have: the top face is a pushout if and only if the front faces are pullbacks.*



(5)

We recall that in an adhesive category pushouts preserve monos, and pushouts along monos are also pullbacks; pullbacks preserve monos in any category.

Definition 4 (AGREE rewriting). *Let* $\mathbf{C}$ *be an adhesive category with a partial map classifier* (T, η)*, An* AGREE rule *is a triple of arrows with the same source* $\rho = (K \xrightarrow{l} L, K \xrightarrowtail{r} R, K \xrightarrowtail{t} T_K)$*, with* r *and* t *mono. Arrows* l *and* r *are the* left- *and* right-hand side*, respectively, and* t *is the* embedding*. A* match *of rule* ρ *is a mono* $L \xrightarrowtail{m} G$*. An* AGREE rewrite step $G \Rightarrow_{\rho,m} H$ *is constructed as follows (see diagram (6)). First* $G \xleftarrow{g} D \xrightarrow{n'} T_K$ *is the pullback of* $G \xrightarrow{\overline{m}} T(L) \xleftarrow{l'} T_K$*. It follows that there is a unique* $n : K \rightarrow D$ *such that* $n' \circ n = t$*,* $g \circ n = m \circ l$ *and* (l, n) *is a pullback of* (m, g)*, and that* n *is mono. Then* $R \xrightarrowtail{p} H \xleftarrow{h} D$ *is the pushout of* $D \xleftarrow{n} K \xrightarrowtail{r} R$*.*



(6)

The assumptions of Definition 4 are satisfied by the categories of graphs, of *typed graphs* (defined as a slice category), and by toposes in general.

Notice that, differently from [3], we stick to rules with monic right-hand side, thus rules which possibly model the cloning of items, but not their merging. This choice is supported by the observation that matches must be monic in AGREE, and thus even if a monic morphism, say, $m_{1d} : L_1 \rightarrowtail D_2$ can be found (see diagram (1)), its composition with a non-monic $h_2 : D_2 \to H_2$ would not necessarily result in a legal (i.e., monic) match of L_1 in H_2. The analysis of such more complex situations is left as future work.

Finally it is worth recalling that as proved in [3], AGREE rewriting coincides with SqPO rewriting [4] for rules where $T_K = T(K)$.

4 Analysis of Independence of Rewrite Steps

As stated in the Introduction, *parallel independence* is a condition on two rewrite steps from the same graph that ensures that they can be applied sequentially in both orders, leading to the same result. We formalize this last property with the following notion of *commutativity*, also known as *diamond property*.

Definition 5 (Commutativity of rewrite steps). *Let ρ_1 and ρ_2 be two rules and for $i \in \{1, 2\}$ let m_i be a match for ρ_i in G. We say that the rewrite steps $G \Rightarrow_{\rho_1, m_1} H_1$ and $G \Rightarrow_{\rho_2, m_2} H_2$* commute *if there exist an object H and matches m_{12} of ρ_1 in H_2 and m_{21} of ρ_2 in H_1 such that $H_1 \Rightarrow_{\rho_2, m_{21}} H$ and $H_2 \Rightarrow_{\rho_1, m_{12}} H$.*

We discussed two possible kinds of conflicts that could prevent commutativity in classical approaches to GT: ***preserve-delete*** (one of the rules deletes some item that is preserved by the other) and ***delete-delete*** (two rules delete the same item). In AGREE we still have these kinds of conflicts. But we have to investigate what is the impact of using the embedding component TK of the rules, and of allowing the cloning of items.

Let us now consider some examples illustrating different kinds of situations that may occur in AGREE derivations. These examples are meant to show that, although cloning is a kind of preservation, the application of a rule that clones may hinder the application of a rule that uses the cloned items, since the

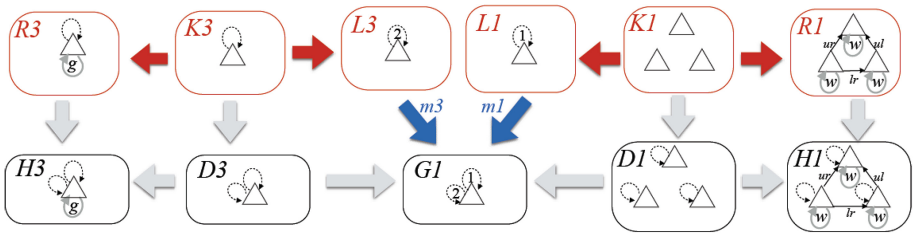


Fig. 6. Rewrite steps with rules ***TurnGreen*** and ***Split***: Clone-Use Confict

match may become non-deterministic (leading to different results) and items that belong to the context of one rule application may be changed in a way that prevents the other rule from being applied. Moreover, even if no cloning is used, rules may get into conflict due to the treatment of context items specified in the TK component of one of the rules.

Cloning vs Use. Consider the derivations shown in Fig. 6, where rules ***TurnGreen*** (left) and ***Split*** (right) are applied to graph $G1$ (indices indicate the match). The application of rule ***TurnGreen*** just changes the color of the triangle and, after applying this rule, it would still be possible to apply ***Split*** to the *same* match (that is, it is possible to extend $m1$ to $H3$) and the result would be a graph with 3 white triangles (and corresponding edges). However, if ***Split*** is applied first, we would have three possibilities to match rule ***TurnGreen*** that would be extensions of $m1$. By choosing any of them, the result would be a graph with 3 triangles, two white and one green, i.e. the results would not be the same.

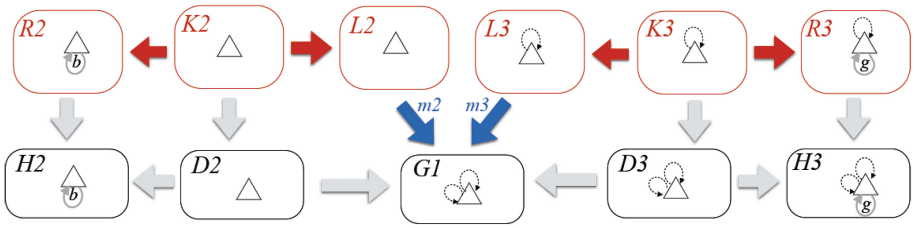


Fig. 7. Applying ***TurnBlack*** and ***TurnGreen***: Context Deletion–Preservation Confict

Context Deletion vs Preservation. Now consider the derivations shown in Fig. 7, where rules ***TurnBlack*** (left) and ***TurnGreen*** (right) are applied to graph $G1$. The application of rule ***TurnGreen*** just changes the color of the triangle and, after applying this rule, it would still be possible to apply ***TurnBlack*** to the *same* match (that is, it is possible to extend $m2$ to $H3$) and the result would be a graph with only one black triangle. But if ***TurnBlack*** is applied first, all dashed loops are removed (as specified by $TK2$) preventing ***TurnGreen***

from being applied. None of these rules clones items, thus this kind of conflict depends only on the embedding component of the rules: if the embedding components were $TK2 = T(K2)$ and $TK3 = T(K3)$, no conflict would arise because all context items would be preserved by both rules.

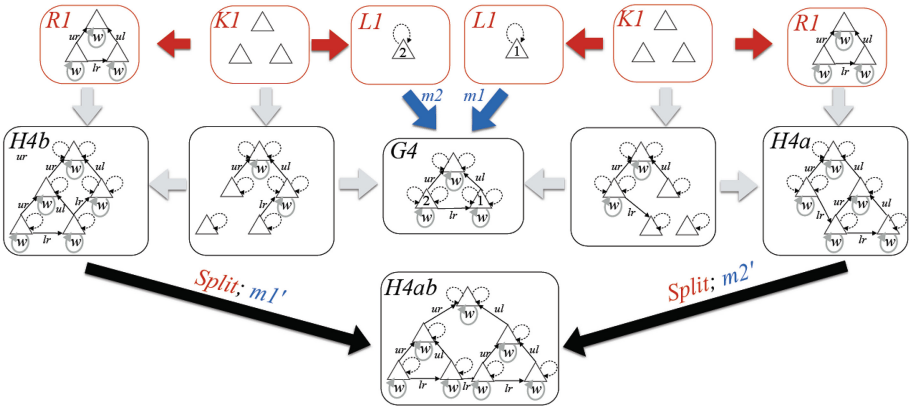


Fig. 8. AGREE Derivations using rule ***Split***

Figure 5 shows a situation where both of the above cases occur: rule ***TurnBlack*** removes the dashed loops, preventing ***Split*** from being applied, and ***Split*** clones the triangle, creating three different matches for ***TurnBlack***. Finally, Fig. 8 shows an example involving cloning and using a non-trivial embedding component ($TK1 \neq T(K1)$), where the two steps commute. In fact the two matches do not overlap, and thus are trivially parallel independent.

Summarizing, using AGREE rules, we have three new kinds of conflict:

clone-use (where **use** could be **delete** or **preserve** or **clone**)**:** an item that is preserved/deleted/cloned by one rule is cloned by the other.

ctxdel-use (context deletion-use): an item used in one rule is specified for context-deletion by the embedding component TK of the other rule.

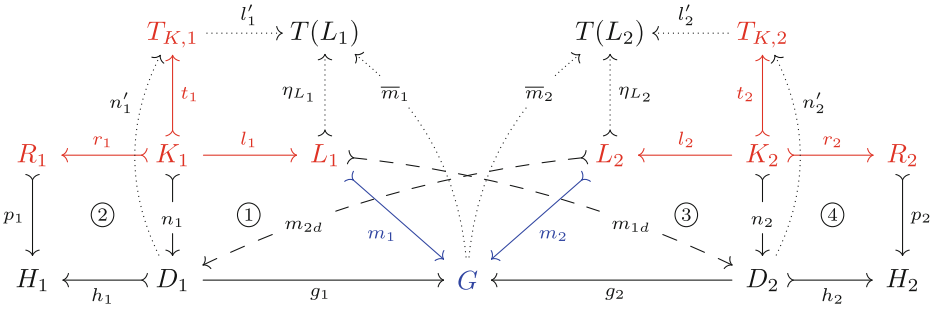
ctxclone-use (context clone-use): an item used in one rule is specified for context-cloning by the embedding component TK of the other rule[4].

5 The Church-Rosser Property for AGREE

This section is devoted to the main result of the paper, that is the identification of sufficient conditions for two AGREE rewrite steps to commute, according to Definition 5. Such conditions are identified in the next definition.

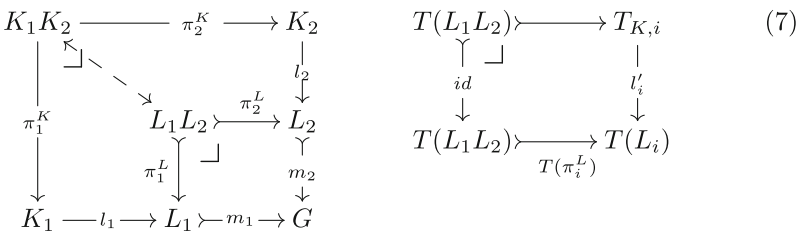
[4] We did not present examples of this kind of conflict, which can be avoided by requiring the embedding component to be included in $T(K)$.

Definition 6 (Parallel Independence in AGREE). *Let* **C** *be an adhesive category with a partial map classifier* (T, η)*. Let* $\rho_i = (K_i \overset{l_i}{\rightarrowtail} L_i, K_i \overset{r_i}{\rightarrowtail} R_i, K_i \overset{t_i}{\rightarrowtail} T_{K,i})$*, for* $i \in \{1,2\}$*, be two AGREE rules and let* $L_1 \overset{m_1}{\rightarrowtail} G$ *and* $L_2 \overset{m_2}{\rightarrowtail} G$ *be two matches for them to the same object* G*. Consider the corresponding* AGREE *rewriting steps* $G \Rightarrow_{\rho_1,m_1} H_1$ *and* $G \Rightarrow_{\rho_2,m_2} H_2$ *depicted in the following diagram.*[5]



Then $G \Rightarrow_{\rho_1,m_1} H_1$ *and* $G \Rightarrow_{\rho_2,m_2} H_2$ *are* parallel independent *if the following are satisfied:*

1. *In the left diagram of (7) where the inner and the outer squares are built as pullbacks, the mediating morphism* $K_1K_2 \to L_1L_2$ *is an isomorphism.*
2. *The right diagram of (7) is a pullback for* $i \in \{1,2\}$*, that is the image of* $T(L_1L_2)$ *is reflected identically by* l_i' *to* $T_{K,i}$



(7)

The main result is formulated as follows.

Theorem 1 (Local Church-Rosser). *If two* AGREE *rewrite steps are parallel independent, then they commute.*

As a first observation note that, unlike most related results for other algebraic approaches to GT, parallel independence does not require explicitly the existence

[5] For future reference we also depict the dashed arrows m_{1d} and m_{2d}, which are not mentioned in this definition.

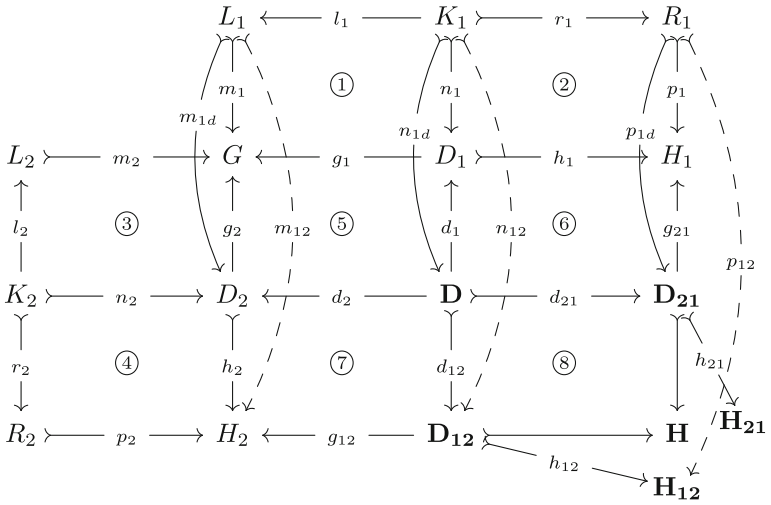
of arrows m_{1d} and m_{2d}, which will be inferred in the proof from the other conditions. Nevertheless, note that the first condition can be seen as a direct translation in categorical terms of the classical *set-theoretical* definition of parallel independence (see [9]) requiring $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$.

Since the conditions of Definition 6 are pretty technical, let us explain them by making reference to the specific case of graphs. The first condition guarantees that each item of G that is needed for the application of both rules (belongs to the intersection of the images of L_1 and L_2) is preserved by both rules (is in the image of both K_1 and K_2) and it is not cloned by any rule (it has only one inverse image in K_1K_2). This forbids all delete-use and clone-use conflicts. Equivalently, if a rule duplicates or deletes an item of G, that item cannot be accessed by the other rule not even in a read-only way. For example, the application of rules ***TurnGreen*** and ***Split*** shown in Fig. 6 does not satisfy this condition because the pullback of $L3 \to G1$ and $L1 \to G1$ contains a single node, while the pullback of $K3 \to L3 \to G1$ and $K1 \to L1 \to G1$ contains three nodes, and thus they are not isomorphic.

For the second condition, remember from Sect. 2 that for any graph X, the partial map classifier $T(X)$ is made of a copy of X plus the $\star$-elements which, given any graph Y with a partial morphism to X, classify in a unique way the items of the context, i.e. the items of Y on which the morphism is not defined. Thus the second condition expresses a strong requirement on the embeddings $T_{K,i}$ of the two rules: they cannot modify (i.e. delete or duplicate) any item in the context of L_1L_2. For example, this condition is not satisfied by the application of rules ***TurnBlack*** and ***TurnGreen*** to graph $G1$ in Fig. 7. In fact, in this case the pullback object of $L2 \to G1$ and $L3 \to G1$ is a single node (it is identical to $L2$), but $T(L2)$ is not reflected identically by $TK2 \to T(L2)$, because the embedding $TK2$ (see Fig. 4) does not contain the $\star$-loop on the left node.

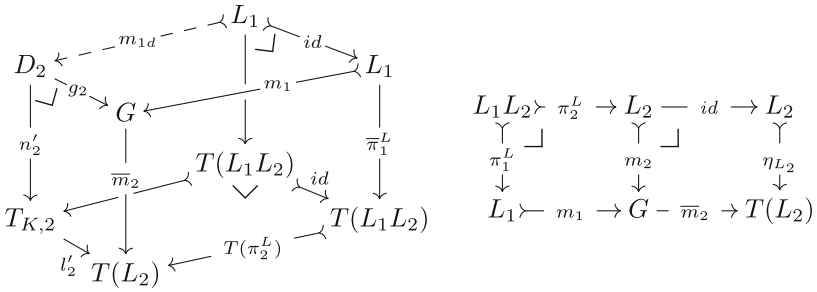
Proof (of Theorem 1). We present the overview of the proof, which is detailed in the rest of the section. We focus on the application of ρ_2 and ρ_1 in this order, since the reverse order is symmetric. Consider Diagram (8), where for readability reasons we do not depict the embeddings of the rules and the partial maps classifiers, even if they are necessary for the constructions. Objects in plain math style exist by hypotheses, others (in bold) are introduced during the proof.

By Lemma 1, L_1 is reflected identically by $D_2 \text{-}g_2\!\succ G$ providing the mono $L_1 \succ^{m_{1d}} \!\to H_2$, which composed with $D_2 \succ^{h_2} \!\to H_2$ becomes a match $L_1 \succ^{m_{12}} \!\to H_2$ .By Construction 1 the AGREE rewrite step $H_2 \Rightarrow_{\rho_1, m_{12}} H_{12}$ generates objects D_{12} and H_{12} in the bottom line. Symmetrically, the AGREE rewrite step $H_1 \Rightarrow_{\rho_2, m_{21}} H_{21}$ generates the objects D_{21} and H_{21} in the right column. By Lemma 2, defining D as the pullback of square ⑤, K_1 is reflected identically by $D \text{-}^{d_1} \!\to D_1$ and R_1 is reflected identically by $D_{21} \text{-}^{g_{21}} \!\to H_1$, providing monos $K_1 \succ^{n_{1d}} \!\to D$ and $R_1 \succ^{p_{1d}} \!\to D_{21}$. Lemma 3 shows that the only arrow $D \succ^{d_{21}} \!\to D_{21}$ that makes square ⑥ a pullback also makes the composed square ②+⑥ a pushout. It concludes by building H in ⑧ as the pushout object of $D_{21} \leftarrowtail D \rightarrowtail D_{12}$ (where the arrow $D \rightarrowtail D_{12}$ is built symmetrically to $D \rightarrowtail D_{21}$, making square ⑦ a pullback) and showing, by compositionality of pushouts, that H must be isomorphic to H_{12}. The result follows by symmetry.



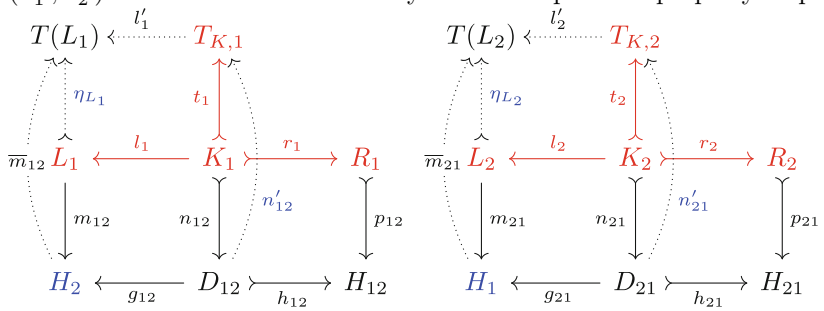
(8)

Lemma 1. *Consider the left diagram of (9). There is a unique (monic) arrow* $m_{1d} : L_1 \to D_2$ *making the top and the back-left faces commuting, and the top face a pullback. Thus* L_1 *is reflected identically by* g_2*.*



(9)

Proof. In the left cube, the front-left face is a pullback by construction of step $G \Rightarrow_{\rho_2,m_2} H_2$, the bottom face is a pullback by hypothesis (see (7)), and the back-right face is trivially a pullback. In addition the front-right face commutes: in fact on one hand we have $T(\pi_2^L) \circ \overline{\pi}_1^L = \varphi(\pi_1^L, \pi_2^L)$ by property (4) of partial maps classifiers, on the other hand the right diagram of (9) proves that $\overline{m}_2 \circ m_1 = \varphi(\pi_1^L, \pi_2^L)$. The statement follows by the decomposition property of pullbacks.



(10)

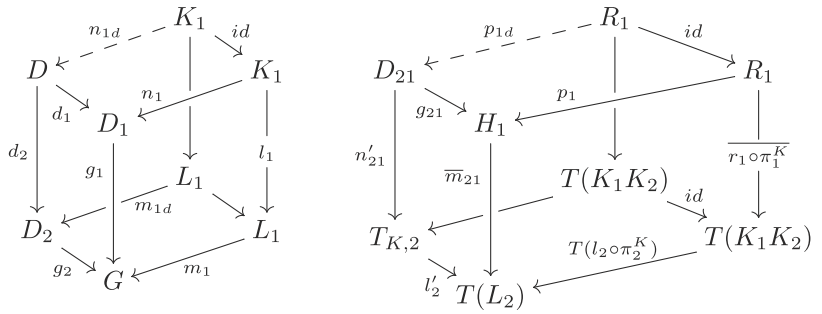
Construction 1. *Arrow $m_{1d} : L_1 \to D_2$ of Lemma 1 composed with $h_2 : D_2 \to H_2$ (see (8)) provides a match $m_{12} = h_2 \circ m_{1d} : L_1 \to H_2$: it is mono because both m_{1d} and h_2 are, the latter because pushouts preserve monos in adhesive categories. The left diagram of (10) represents the resulting* AGREE *rewrite step $H_2 \Rightarrow_{\rho_1, m_{12}} H_{12}$. The right diagram of (10) represents the symmetric rewrite step $H_1 \Rightarrow_{\rho_2, m_{21}} H_{21}$, where $m_{21} = h_1 \circ m_{2d}$.*

The proofs of the next two lemmas are omitted for space constraints, and will appear in the full version of the paper.

Lemma 2. *Let $D_2 \prec_{d_2} D \succ_{d_1} D_1$ be the pullback of $D_2 \succ_{g_2} G \prec_{g_1} D_1$ (see square ⑤ of (8)), and consider the diagrams (11).*

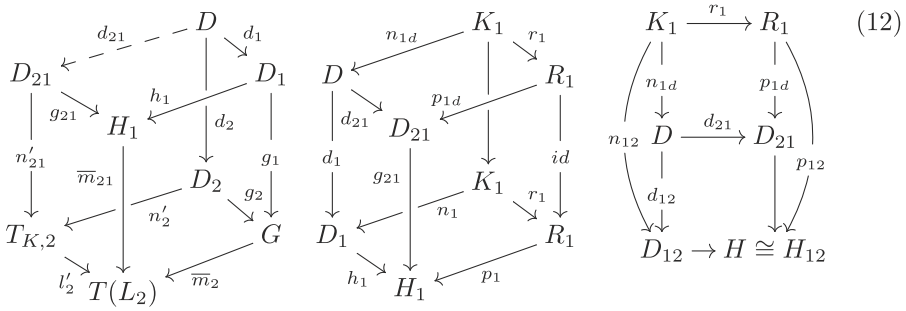
1. *In the left cube, there is a unique (monic) arrow $n_{1d} : K_1 \to D$ making the top and the back-left faces commuting, and the top face a pullback. Thus K_1 is reflected identically by d_1.*
2. *In the right cube, there is a unique (monic) arrow $p_{1d} : R_1 \to D_{21}$ making the top and the back-left faces commuting, and the top face a pullback. Thus R_1 is reflected identically by g_{21}.*

(11)



Lemma 3. *In the left diagram of (12) there is a unique arrow $d_{21} : D \to D_{21}$ making the top and the back-left faces commuting and the top face a pullback. Symmetrically, we get an arrow $d_{12} : D \to D_{12}$. Furthermore, the top face of the central diagram is a pushout. Now define $D_{21} \succ H \prec D_{12}$ as the pushout of $D_{21} \prec_{d_{21}} D \succ_{d_{12}} D_{12}$ (see square ⑧ of (8)). Then from the right diagram we infer that $H \cong H_{12}$.*

(12)



6 Conclusion and Related Works

In this paper we proposed sufficient conditions to ensure that two rewrite steps in the AGREE approach to GT commute. Unlike most of previous works on parallel independence [1,4,5,11–13], we consider an approach in which cloning is possible. Actually, general rules are considered also in the restricted version of the SqPO approach proposed in [7]: the exact relationship with those results is under investigation. The possibility of cloning makes the analysis of parallel independence more complex. Moreover, the fact that the embedding of cloned parts can be finely tuned in AGREE adds another layer of complexity: besides of conflicts that may arise from overlapping matches (as for classical approaches), new conflicts may arise from cloning or deletion of edges incident to the matched parts of the transformed graph.

The conditions for commutativity proposed in this paper are sufficient, but not necessary. It is easy to build a counterexample with two rules, that act as the identity transformation on a given graph G (the left- and right-hand sides are all identities on G), but differ in the embedding component in such a way that the second condition of Definiton 6 is not satisfied. For example, the first rule has the partial map classifier $T(G)$ as embedding, while the second has a larger embedding (e.g. duplicating some contextual arc). Since the first rule acts as the identity (both G and the context are preserved), the two rules clearly commute when applied to G, even if they are not "parallel independent" according to Definition 6. We are currently working on the identification of refined conditions which could enjoy completeness. A first analysis suggests that such conditions, if they exist, should also depend on the right-hand sides of the rules, differently from those identified in Sect. 5.

Following the classical outline of the theory of parallelism for the algebraic approaches to GT, other interesting topics worthy of study are the analysis of conditions for *sequential independence* for AGREE rewrite steps, and the definition of *parallel rules* allowing to model the simultaneous application of two rules to a state. Both topics look not obvious: the first one because AGREE rewrite steps are intrinsically non-symmetric (unlike, e.g., DPO or Reversible SqPO rewrite steps); the second because of the need of merging in some way the embedding components of the two rules.

References

1. Bonchi, F., Gadducci, F., Heindel, T.: Parallel and sequential independence for borrowed contexts. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 226–241. Springer, Heidelberg (2008)
2. Cockett, J., Lack, S.: Restriction categories II: partial map classification. Theor. Comput. Sci. **294**(1–2), 61–102 (2003)
3. Corradini, A., Duval, D., Echahed, R., Prost, F., Ribeiro, L.: AGREE – algebraic graph rewriting with controlled embedding. In: Parisi-Presicce, F., Westfechtel, B. (eds.) ICGT 2015. LNCS, vol. 9151, pp. 35–51. Springer, Heidelberg (2015)

4. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer, Heidelberg (2006)
5. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation - part I: basic concepts and double pushout approach. In: Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1, pp. 163–246. World Scientific, Singapore (1997)
6. Danos, V., Feret, J., Fontana, W., Harmer, R., Hayman, J., Krivine, J., Thompson-Walsh, C.D., Winskel, G.: Graphs, rewriting and pathway reconstruction for rule-based models. In: FSTTCS 2012. LIPIcs, vol. 18, pp. 276–288. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
7. Danos, V., Heindel, T., Honorato-Zimmer, R., Stucki, S.: Reversible sesqui-pushout rewriting. In: Giese, H., König, B. (eds.) ICGT 2014. LNCS, vol. 8571, pp. 161–176. Springer, Heidelberg (2014)
8. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G. (eds.): Handbook of Graph Grammars and Computing by Graph Transformations. Concurrency, Parallelism and Distribution, vol. 3. World Scientific, Singapore (1999)
9. Ehrig, H.: Introduction to the algebraic theory of graph grammars (a survey). In: Claus, V., Ehrig, H., Rozenberg, G. (eds.) Graph-Grammars and Their Application to Computer Science and Biology. LNCS, vol. 73, pp. 1–69. Springer, Heidelberg (1979)
10. Ehrig, H., Golas, U., Hermann, F.: Categorical frameworks for graph transformation and HLR systems based on the DPO approach. Bulletin of the EATCS **102**, 111–121 (2010)
11. Ehrig, H., Habel, A., Lambers, L.: Parallelism and concurrency theorems for rules with nested application conditions. Electronic Communications of the EASST **26**, 1–23 (2010)
12. Ehrig, H., Löwe, M.: Parallel and distributed derivations in the single-pushout approach. Theor. Comput. Sci. **109**(1&2), 123–143 (1993)
13. Hermann, F., Corradini, A., Ehrig, H.: Analysis of permutation equivalence M-adhesive transformation systems with negative application conditions. Math. Struct. Comput. Sci. **24**(4), 1–47 (2014)
14. Lack, S., Sobociński, P.: Adhesive categories. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004)
15. Rosselló, F., Valiente, G.: Chemical graphs, chemical reaction graphs, and chemical graph transformation. Electr. Notes Theor. Comput. Sci. **127**(1), 157–166 (2005)
16. Taentzer, G., et al.: Generation of Sierpinski triangles: a case study for graph transformation tools. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 514–539. Springer, Heidelberg (2008)