Combining Parallel Graph Rewriting and Quotient Graphs

Thierry Boy de La Tour, Rachid Echahed

Combining Parallel Graph Rewriting and Quotient Graphs

Thierry Boy de la Tour and Rachid Echahed

CNRS and University Grenoble Alpes, LIG Lab. Grenoble, France
thierry.boy-de-la-tour@imag.fr rachid.echahed@imag.fr

Abstract. We define two graph transformations, one by parallelizing graph rewrite rules, the other by taking quotients of graphs. The former consists in the exhaustive application of local transformations defined by graph rewrite rules expressed in a set-theoretic framework. Compared with other approaches to parallel rewriting, we allow a substantial amount of overlapping only restricted by a condition called the *effective deletion property*. This transformation can be reduced by factoring out possibly many equivalent matchings by the automorphism groups of the rules. The second transformation is based on the use of equivalence relations over graph items and offers a new way of performing simultaneous merging operations. The relevance of combining the two transformations is illustrated on a running example.

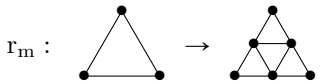
1 Introduction

Graph structures play an important role in the modeling and construction of complex systems in various disciplines including computer science, biology, chemistry or physics, as they provide natural and concise representation of many data structures. Computing with graphs as first-class citizens requires the use of advanced graph-based computational models. In contrast to term rewriting [2], there are different ways, in the literature, to define graphs (e.g., simple or multiple graphs, hyper-graphs, attributed graphs, etc.) as well as their transformations, see, e.g., [13, 15, 4].

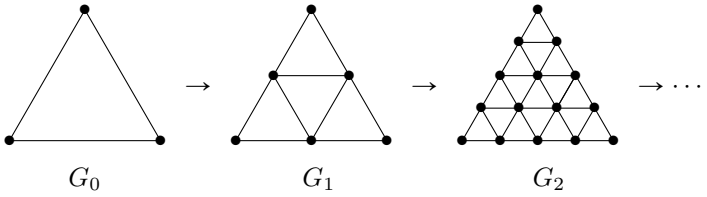
In this paper, we are interested in parallel transformations of graphs which yield deterministic computations. There are many situations where simultaneous graph transformations occur in practice such as social networks dynamics, cell-phones connections, cellular automata or biological processes such as plant growth. The need of using parallel graph transformations has been pointed out since the 70's, see e.g., [14, 17]. The main novelty of our proposal is twofold: first, overlapping transformations can be handled in parallel under a suitable condition called the *effective deletion property*. This new condition makes it possible to fire simultaneously two (or more) rules with a mild form of disagreement in the sense that one rule can delete an item (i.e., node, arrow or attribute) while this is not required by another rule. The second novelty of the paper consists in proposing graph equivalences to formally describe parallel merging of graph

items or attributes' expression evaluations. Furthermore, we introduce the notion of automorphism groups associated to graph rewrite rules which induce a substantial improvement of simultaneous graph transformations.

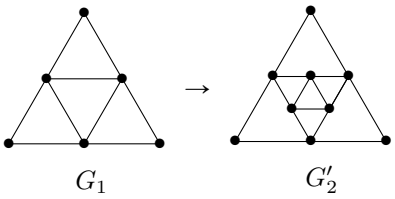
To motivate our purpose and to illustrate the introduced notions, we consider a running example borrowed from the rules defining mesh refinements [1]. In the following rule $\mathrm{r_m}$, a triangle is refined into four smaller triangles. Notice that this example does not show all the expressiveness of the rules we consider but illustrates sufficiently the investigated concepts.



This geometric rule specifies a sequence of mesh refinements as depicted below :



In this paper, we wish to view mesh refinement as a transformation of graphs, and to propose methods that could achieve this purpose. One obvious point is that $\mathrm{r_m}$ has to be applied simultaneously to every triangle subgraph of G_i to obtain G_{i+1}. However, if rule $\mathrm{r_m}$ is applied sequentially, say, at the center triangle subgraph of G_1, we get :



then it is no longer possible to obtain G_2 from G'_2 since the side subgraph triangles of G_1 have been modified and cannot be matched anymore by the left-hand side of the rule $\mathrm{r_m}$. In other words, the matchings of $\mathrm{r_m}$ in G_1 are not parallel independent. We therefore need to define a general parallel transformation that may yield a result unreachable by sequential rewriting. This will be achieved in Sections 5 and 6.

Before that, we start by introducing the considered definition of attributed graphs in Section 2, together with convenient notations. A set-theoretic framework is developed starting from Section 3 that will enable us to define graph transformations by an algebraic expression (Definition 6). Section 4 is dedicated to defining the notion of rewrite rules, together with their matchings. In Section 5, the general parallel graph transformations are defined and a central notion

of *effective deletion property* of sets of matchings is exposed, which guarantees that graph objects are consistently deleted during the transformation. Section 6 is dedicated to a particular parallel rewrite relation where parallel matchings are considered up to automorphisms, based on a notion of automorphism groups of the considered rules. This section uses notions borrowed from group theory. In Section 7, we introduce the notion of graph transformation as quotient graphs. This is not rule-based but allows one to write fancy definitions of merge actions over graphs that cannot be expressed with the rules above. Finally, related work and concluding remarks are given in Section 8. These transformations are all illustrated on the example of rule $\mathrm{r_m}$ in quite some detail. The missing proofs can be found in [5].

2 Preliminaries

We assume a many-sorted signature Σ and a set $\mathscr{V}$ of *variables*, disjoint from Σ, such that every variable has a Σ-sort. For any finite $X \subseteq \mathscr{V}$, $\mathscr{T}(\Sigma, X)$ denotes the algebra of Σ-terms over X.

An *attributed graph* (or *graph* for short) G is a tuple $(\dot{G}, \vec{G}, \acute{G}, \grave{G}, \mathscr{A}_G, \mathring{G})$ where $\dot{G}, \vec{G}$ are sets, $\acute{G}, \grave{G}$ are the *source* and *target* functions from $\vec{G}$ to $\dot{G}$, $\mathscr{A}_G$ is a Σ-algebra and $\mathring{G}$ is an *attribution of* G, i.e., a function from $\dot{G} \cup \vec{G}$ to $\mathscr{P}(\lfloor \mathscr{A}_G \rfloor)$ (the carrier set $\lfloor \mathscr{A}_G \rfloor$ of $\mathscr{A}_G$ is the disjoint union of the carrier sets of the sorts in $\mathscr{A}_G$). We assume that $\dot{G}$, $\vec{G}$ and $\lfloor \mathscr{A}_G \rfloor$ are pairwise disjoint; their elements are respectively called *vertices*, *arrows* and *attributes*. G is *unlabelled* if $\mathring{G}(x) = \varnothing$ for all $x \in \dot{G} \cup \vec{G}$, it is *finite* if the sets $\dot{G}$, $\vec{G}$ and $\mathring{G}(x)$ are finite. The *carrier* of G is the set $\lfloor G \rfloor \stackrel{\text{def}}{=} \dot{G} \cup \vec{G} \cup \lfloor \mathscr{A}_G \rfloor$.

A graph H is a *subgraph* of G, written $H \lhd G$, if the *underlying graph* $(\dot{H}, \vec{H}, \acute{H}, \grave{H})$ of H is a subgraph of G's underlying graph (in the usual sense), $\mathscr{A}_H = \mathscr{A}_G$ and $\forall x \in \dot{H} \cup \vec{H}, \mathring{H}(x) \subseteq \mathring{G}(x)$.

A morphism α from graph H to graph G is a function from $\lfloor H \rfloor$ to $\lfloor G \rfloor$ such that the restriction of α to $\dot{H} \cup \vec{H}$ is a morphism from H's to G's underlying graphs (that is, $\acute{G} \circ \alpha = \alpha \circ \acute{H}$ and $\grave{G} \circ \alpha = \alpha \circ \grave{H}$, this restriction of α is called the *underlying graph morphism of* α), the restriction of α to $\lfloor \mathscr{A}_H \rfloor$ is a Σ-homomorphism from $\mathscr{A}_H$ to $\mathscr{A}_G$, denoted $\mathring{\alpha}$, and $\forall x \in \dot{H} \cup \vec{H}, \mathring{\alpha} \circ \mathring{H}(x) \subseteq \mathring{G} \circ \alpha(x)$. This means that α is an isomorphism if and only if α is a bijective morphism *and* α^{-1} is a morphism, hence if and only if the underlying graph morphism of α is an isomorphism, $\mathring{\alpha}$ is a Σ-isomorphism *and* $\mathring{\alpha} \circ \mathring{H} = \mathring{G} \circ \alpha$. We write $H \simeq G$ if there exists an isomorphism from H to G. For all $F \lhd H$, the *image* $\alpha(F)$ is the smallest subgraph of G w.r.t. the order $\lhd$ such that $\alpha|_{\lfloor F \rfloor}$ is a morphism from F to $\alpha(F)$.

If the underlying graph morphism of α is injective then α is called a *matching*. Note that the Σ-homomorphism $\mathring{\alpha}$ need not be injective.

Given two attributions l and l' of G we define $l \backslash l'$ (resp. $l \cap l'$, $l \cup l'$) as the attribution of G that maps any x to $l(x) \backslash l'(x)$ (resp. $l(x) \cap l'(x)$, $l(x) \cup l'(x)$). If l is an attribution of a subgraph $H \lhd G$, we extend it implicitely to the attribution of G that is identical to l on $\dot{H} \cup \vec{H}$ and maps any other x to $\varnothing$.

3 Joinable Graphs

In order to define parallel rewrite relations on graphs, it is convenient to join possibly many different graphs that have a common part, i.e., that are *joinable*. As a matter of fact, this notion also allows a simple definition of graph rewrite rules, and is crucial in defining the automorphism groups of these rules. We start with a simpler notion of joinable functions.

Definition 1 (joinable functions). *Two functions $f : D \to C$ and $g : D' \to C'$ are* joinable *if $\forall x \in D \cap D'$, $f(x) = g(x)$. Then, the* meet *of f and g is the function $f \curlywedge g : D \cap D' \to C \cap C'$ that is the restriction of f (or g) to $D \cap D'$. The* join *$f \curlyvee g$ is the unique function from $D \cup D'$ to $C \cup C'$ such that $f = (f \curlyvee g)|_D$ and $g = (f \curlyvee g)|_{D'}$.*

For any set I and any I-indexed family $(f_i : D_i \to C_i)_{i \in I}$ of pairwise joinable functions, let $\bigvee_{i \in I} f_i$ be the only function from $\bigcup_{i \in I} D_i$ to $\bigcup_{i \in I} C_i$ such that $f_i = \left(\bigvee_{i \in I} f_i\right)|_{D_i}$ for all $i \in I$.

If S and T are sets of functions, let $S \curlyvee T \stackrel{\text{def}}{=} \{f \curlyvee g \mid f \in S, g \in T\}$ and $S \circ T \stackrel{\text{def}}{=} \{f \circ g \mid f \in S, g \in T\}$, provided these operations can be applied. If f is a function, let $f \circ T \stackrel{\text{def}}{=} \{f\} \circ T$.

In particular, functions with disjoint domains are joinable (e.g. $\dot{\alpha}$ and $\vec{\alpha}$), and every function is joinable with itself: $f \curlyvee f = f \curlywedge f = f$. More generally, any two restrictions $f|_A$ and $f|_B$ of the same function f are joinable, $f|_A \curlywedge f|_B = f|_{A \cap B}$ and $f|_A \curlyvee f|_B = f|_{A \cup B}$. Conversely, if f and g are joinable then each is a restriction of $f \curlyvee g$.

It is obvious that these operations are commutative. On triples of pairwise joinable functions, they are also associative and distributive over each other.

Definition 2 (joinable graphs). *Two graphs H and G are* joinable *if $\mathscr{A}_H = \mathscr{A}_G$, $\dot{H} \cap \vec{G} = \vec{H} \cap \dot{G} = \varnothing$, and the functions $\acute{H}$ and $\acute{G}$ (and similarly $\grave{H}$ and $\grave{G}$) are joinable. We can then define the graphs*

$$H \sqcap G \stackrel{\text{def}}{=} (\ \dot{H} \cap \dot{G},\ \vec{H} \cap \vec{G},\ \acute{H} \curlywedge \acute{G},\ \grave{H} \curlywedge \grave{G},\ \mathscr{A}_H,\ \mathring{H} \cap \mathring{G}\ ),$$
$$H \sqcup G \stackrel{\text{def}}{=} (\ \dot{H} \cup \dot{G},\ \vec{H} \cup \vec{G},\ \acute{H} \curlyvee \acute{G},\ \grave{H} \curlyvee \grave{G},\ \mathscr{A}_H,\ \mathring{H} \cup \mathring{G}\ ).$$

Similarly, for any set I, any I-indexed family of graphs $(G_i)_{i \in I}$ that are pairwise joinable, and any Σ-algebra $\mathcal{A}$ such that $\mathcal{A} = \mathscr{A}_{G_i}$ for all $i \in I$, let

$$\bigsqcup_{i \in I} G_i \stackrel{\text{def}}{=} \left(\bigcup_{i \in I} \dot{G}_i,\ \bigcup_{i \in I} \vec{G}_i,\ \bigvee_{i \in I} \acute{G}_i,\ \bigvee_{i \in I} \grave{G}_i,\ \mathcal{A},\ \bigcup_{i \in I} \mathring{G}_i\right).$$

It is easy to see that these structures are graphs: the sets of vertices and arrows are disjoint and the adjacency functions have the correct domains and codomains. If $I = \varnothing$ the chosen algebra $\mathcal{A}$ is generally obvious from the context. Note that if H and G are joinable then $H \sqcap G = G \sqcap H \lhd H \lhd H \sqcup G = G \sqcup H$. Similarly, if the G_i's are pairwise joinable then $\forall j \in I$, $G_j \lhd \bigsqcup_{i \in I} G_i$. We also see that any two subgraphs of G are joinable, and that $H \lhd G$ iff $H \sqcap G = H$ iff $H \sqcup G = G$. As above, on triples of pairwise joinable $\mathcal{A}$-graphs, these operations are associative and distributive over each other.

Definition 3. *For any graph G, sets V, A and attribution l, we say that G* is disjoint from V, A, l *if $\dot{G} \cap V = \varnothing$, $\vec{G} \cap A = \varnothing$ and $\mathring{G}(x) \cap l(x) = \varnothing$ for all $x \in \dot{G} \cup \vec{G}$.*

We write $G\backslash[V, A, l]$ for the largest subgraph of G (w.r.t. $\lhd$) that is disjoint from V, A, l.

This provides a natural way of removing objects from an attributed graph. It is easy to see that $G\backslash[V, A, l]$ always exists (it is the union of all subgraphs of G disjoint from V, A, l), hence rewriting steps will not be restricted by a *gluing condition* as in the Double-Pushout approach (see [13]).

4 Rules

We consider rules with three joinable graphs L, K and R as depicted below.

The semantics of such rules is defined in Section 5. Informally, L shall be matched in the input graph G, the region $L\backslash K$ (the items matched by L but not by K) shall be removed from G, and the region $R\backslash L$ shall be added in order to obtain an image of R in the output graph.

Definition 4 (rules, matchings). *For any finite $X \subseteq \mathcal{V}$, we call (Σ, X)-*graph *a finite graph G such that $\mathcal{A}_G = \mathcal{T}(\Sigma, X)$. We define the set of variables occurring in a (Σ, X)-graph G as*

$$\mathrm{Var}(G) \stackrel{\text{def}}{=} \bigcup_{x \in \dot{G} \cup \vec{G}} \left(\bigcup_{t \in \mathring{G}(x)} \mathrm{Var}(t) \right),$$

where $\mathrm{Var}(t)$ is the set of variables occurring in t.

A rule *r is a triple (L, K, R) of (Σ, X)-graphs such that L and R are joinable, $L \sqcap R \lhd K \lhd L$ and $\mathrm{Var}(L) = X$ (see Remark 1 below). The rule r is* standard *if $L \sqcap R = K$.*

A matching *μ of r in a graph G is a matching from L to G such that*

$$\mathring{\mu}(\mathring{L}(x) \backslash \mathring{K}(x)) \cap \mathring{\mu}(\mathring{K}(x)) = \varnothing \tag{1}$$

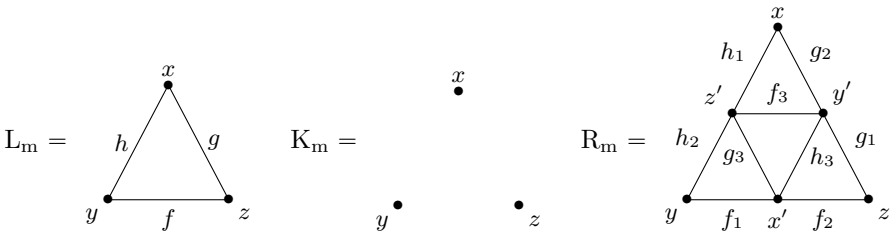
(or equivalently $\mathring{\mu}(\mathring{L}(x) \backslash \mathring{K}(x)) = \mathring{\mu}(\mathring{L}(x)) \backslash \mathring{\mu}(\mathring{K}(x))$) for all $x \in \dot{K} \cup \vec{K}$. We denote $\mathscr{M}(r, G)$ the set of all matchings of r in G (they all have domain $\lfloor L \rfloor$).

We consider finite sets $\mathcal{R}$ of rules such that $\forall r, r' \in \mathcal{R}$, if $(L, K, R) = r \neq r' = (L', K', R')$ then $\dot{L} \cup \vec{L} \neq \dot{L}' \cup \vec{L}'$, so that $\lfloor L \rfloor \neq \lfloor L' \rfloor$ hence $\mathscr{M}(r, G) \cap \mathscr{M}(r', G) = \varnothing$ for any graph G; we then write $\mathscr{M}(\mathcal{R}, G)$ for $\biguplus_{r \in \mathcal{R}} \mathscr{M}(r, G)$. For any $\mu \in \mathscr{M}(\mathcal{R}, G)$ there is a unique rule $\mathrm{r}_\mu \in \mathcal{R}$ such that $\mu \in \mathscr{M}(\mathrm{r}_\mu, G)$, and its components are denoted $\mathrm{r}_\mu = (\mathrm{L}_\mu, \mathrm{K}_\mu, \mathrm{R}_\mu)$.

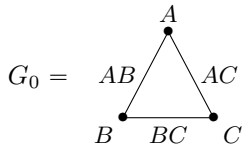
Remark 1. If X were allowed to contain a variable v not occurring in L, then v would freely match any element of $\mathscr{A}_G$ and the set $\mathscr{M}(r, G)$ would contain as many matchings with essentially the same effect. Note that matchings are only injective on graph items (vertices and arrows), so that distinct variables may match the same attribute, as is the case in term rewriting. However, condition (1) restricts the possible matchings in order to distinguish the attributes intended for deletion (those matched by $\dot{L}(x)\backslash\mathring{K}(x)$) from those that are not (matched by $\mathring{K}(x)$).

Also note that $\mathrm{Var}(R) \subseteq \mathrm{Var}(L)$, R and K are joinable and $R \sqcap K = L \sqcap R$. The fact that K is not required to be a subgraph of R allows the possible deletion by other rules of data matched by K but not by R, see Section 5.

Example 1. Let us consider the rule given in the introduction. It could be defined as the standard rule $\mathrm{r_m} = (\mathrm{L_m}, \mathrm{K_m}, \mathrm{R_m})$ with



We assume, for the rule above, that all attributes are empty, hence $\mathrm{L_m}$, $\mathrm{K_m}$ and $\mathrm{R_m}$ are $(\Sigma, \varnothing)$-graphs. Each edge f, g, h represents a pair of opposite arrows; for sake of simplicity they will be treated as single objects. Note that $\mathrm{r_m}$ is a standard rule. A matching μ of $\mathrm{r_m}$ in the graph



is given by $\mu = \{(x, A), (y, B), (z, C), (f, BC), (g, AC), (h, AB)\}$. The Σ-algebra $\mathscr{A}_{G_0}$ and the Σ-morphism $\mathring{\mu}$ are not relevant here, we can choose $\mathscr{T}(\Sigma, \varnothing)$ and its identity morphism (though other algebras and attributes will be adopted later).

A rewrite step may involve the creation of new vertices in a graph, corresponding to the vertices of a rule that have no match in the input graph, i.e., those in $\dot{R}\backslash\dot{L}$ (or similarly may create new arrows). These vertices should really be new, not only different from the vertices of the original graph but also different from the vertices created by other transformations (corresponding to other matchings in the graph). This is computationally easy to do but not that easy to formalize in an abstract way. We simply reuse the vertices x from $\dot{R}\backslash\dot{L}$ by *indexing* them with any relevant matching μ, each time yielding a new vertex (x, μ) which is obviously different from any new vertex (x, ν) for any other matching $\nu \neq \mu$, and also from any vertex of G since μ depends on G.

Definition 5 (graph $G\uparrow_\mu$ and matching $\mu\uparrow$). *For any rule $r = (L, K, R)$, graph G and $\mu \in \mathscr{M}(r, G)$ we define a graph $G\uparrow_\mu$ together with a matching $\mu\uparrow$ of R in $G\uparrow_\mu$. We first define the sets*

$$\dot{G}\uparrow_\mu \stackrel{\text{def}}{=} \mu(\dot{R} \cap \dot{K}) \uplus ((\dot{R} \backslash \dot{K}) \times \{\mu\}) \text{ and } \vec{G}\uparrow_\mu \stackrel{\text{def}}{=} \mu(\vec{R} \cap \vec{K}) \uplus ((\vec{R} \backslash \vec{K}) \times \{\mu\}).$$

Next we define $\mu\uparrow$ by: $\mathring{\mu}\uparrow \stackrel{\text{def}}{=} \mathring{\mu}$ and for all $x \in \dot{R} \cup \vec{R}$, if $x \in \dot{K} \cup \vec{K}$ then $\mu\uparrow(x) \stackrel{\text{def}}{=} \mu(x)$ else $\mu\uparrow(x) \stackrel{\text{def}}{=} (x, \mu)$. Since the restriction of $\mu\uparrow$ to $\dot{R} \cup \vec{R}$ is bijective, then $\mu\uparrow$ is a matching from R to the graph

$$G\uparrow_\mu \stackrel{\text{def}}{=} (\dot{G}\uparrow_\mu, \vec{G}\uparrow_\mu, \mu\uparrow \circ \acute{R} \circ \mu\uparrow^{-1}, \mu\uparrow \circ \grave{R} \circ \mu\uparrow^{-1}, \mathscr{A}_G, \mathring{\mu}\uparrow \circ \mathring{R} \circ \mu\uparrow^{-1}).$$

Example 2. Following Example 1, we get $\dot{G}_0\uparrow_\mu = \{A, B, C, (x', \mu), (y', \mu), (z', \mu)\}$, $\vec{G}_0\uparrow_\mu = \{(f_1, \mu), \ldots, (h_3, \mu)\}$, $\mu\uparrow = \{(x, A), (y, B), (z, C), (x', (x', \mu)), (y', (y', \mu)), (z', (z', \mu)), (f_1, (f_1, \mu)), \ldots, (h_3, (h_3, \mu))\}$. The graph $G_0\uparrow_\mu$ is obtained as $\mu\uparrow(\mathrm{R_m})$.

By construction μ and $\mu\uparrow$ are joinable and $\mu \curlywedge \mu\uparrow$ is a matching from $R \sqcap K$ to $\mu(R \sqcap K)$. It is easy to see that the graph G and the graphs $G\uparrow_\mu$ are pairwise joinable.

5 Parallel Rewriting

For any set $M \subseteq \mathscr{M}(\mathcal{R}, G)$ of matchings in a graph G we define below how to rewrite G by applying simultaneously the rules associated with matches in M.

Definition 6 (graph $G\|_M$). *For any graph G, set $M \subseteq \mathscr{M}(\mathcal{R}, G)$ and matching $\mu \in \mathscr{M}(\mathcal{R}, G)$, let*

$$G\|_M \stackrel{\text{def}}{=} G\backslash[\mathrm{V}_M, \mathrm{A}_M, \ell_M] \sqcup \bigsqcup_{\mu \in M} G\uparrow_\mu \text{ where}$$

$$\mathrm{V}_M \stackrel{\text{def}}{=} \bigcup_{\mu \in M} \mu(\dot{\mathrm{L}}_\mu \backslash \dot{\mathrm{K}}_\mu),\ \mathrm{A}_M \stackrel{\text{def}}{=} \bigcup_{\mu \in M} \mu(\vec{\mathrm{L}}_\mu \backslash \vec{\mathrm{K}}_\mu) \text{ and } \ell_M \stackrel{\text{def}}{=} \bigcup_{\mu \in M} \mathring{\mu} \circ (\mathring{\mathrm{L}}_\mu \backslash \mathring{\mathrm{K}}_\mu) \circ \mu^{-1}.$$

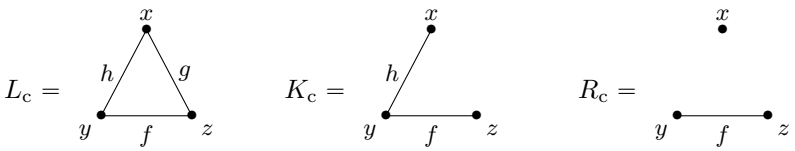
If M is a singleton $\{\mu\}$ we write $G\|_\mu$ for $G\|_M$, V_μ for V_M, etc.

Note that ℓ_M is only defined on $\bigsqcup_{\mu \in M} \mu(\mathrm{K}_\mu)$; so ℓ_M is implicitly extended to the suitable domain by mapping other vertices and arrows to $\varnothing$. $G\|_M$ is guaranteed to be a graph since the $\sqcup$ operation is only applied on joinable graphs.

The definition of $G\|_M$ bears some similarity with the double pushout diagram (see [13]), where $G\backslash[\mathrm{V}_M, \mathrm{A}_M, \ell_M]$ replaces the pushout complement of G and $\bigsqcup_{\mu \in M} G\uparrow_\mu$ its pushout with the right hand side of the rule. But we are not restricted by the gluing condition, and since we use a set of matchings the pushout is actually a colimit. The case where M is a singleton defines the classical semantics of one sequential rewrite step.

It is obvious that $G\|_M$ contains images of the right hand sides of all the rules involved (through the elements M). However it may be the case that some elements of V_M, A_M or ℓ_M occur in $\bigsqcup_{\mu\in M} G\!\uparrow_\mu$, hence also in the result $G\|_M$, since any two matchings may conflict as one retains what another removes as illustrated in the following example.

Example 3. Let us consider the (non standard) rule $r_\mathrm{c} = (L_\mathrm{c}, K_\mathrm{c}, R_\mathrm{c})$ with



This rule removes g since g is in L_c but not in K_c , retains f (and all vertices) since f is in $K_\mathrm{c} \sqcap R_\mathrm{c}$, and does not care about h since h is in K_c but not in R_c. Nothing is added since $K_\mathrm{c} \sqcap R_\mathrm{c} = R_\mathrm{c}$. We consider the same graph G_0 and matching μ as in Example 1. Let $\nu = \mu \circ (x\ y)(f\ g)$ (in cyclic notation), this is obviously a matching of r_c in G_0. By μ we must remove AC and retain BC, while ν asks exactly the opposite which means there is a conflict between the application of the two matches. We easily see that the graph $G_0\!\uparrow_\mu$ contains BC and that $G_0\!\uparrow_\nu$ contains AC, so that $G_0\|_{\{\mu,\nu\}} = G_0$, hence the instructions of removing AC and BC have not been fulfilled. The reader may check that no such conflict occurs between μ and $\mu \circ (y\ z)(g\ h)$; they remove AC and AB.

Since the semantics of individual rules have to be preserved under parallelization, we must avoid such conflicts by stating that any item deleted by any rule should not occur in the result. We can however allow attributes to be removed and yet restored: this is a situation similar to an assignment $a := 1$, where the former value of a is deleted unless it is 1.

Definition 7 (effective deletion property). *For any graph G, a set $M \subseteq \mathscr{M}(\mathcal{R}, G)$ is said to satisfy the* effective deletion property *if $G\|_M$ is disjoint from $\mathrm{V}_M, \mathrm{A}_M, \ell_M \backslash \ell_M^{\uparrow}$, where*

$$\ell_M^{\uparrow} \stackrel{\text{def}}{=} \bigcup_{\mu\in M} \mathring{\mu} \circ (\mathring{\mathrm{R}}_\mu \backslash \mathring{\mathrm{K}}_\mu) \circ \mu^{-1}.$$

We thus see in Example 3 that $\{\mu, \nu\}$ does not have the effective deletion property, since $G_0\|_{\{\mu,\nu\}}$ (i.e., G_0) is not disjoint from $\mathrm{A}_{\{\mu,\nu\}} = \{AC, BC\}$. The following example illustrates the special treatment of attributes.

Example 4. Consider the assignment $a \coloneqq b$ where a and b are identifiers of type `nat`. This expression can be represented as a graph transformation rule in the following way. We assume a signature Σ with two sorts `idtf` and `nat`, and two constants a, b of sort `idtf`. Let $X = \{u, v\}$ where u, v are two variables of sort `nat`. A placeholder is represented as a vertex attributed by a set containing both its identifier and its value. Thus the environment where, say, a has value 1 and b

has value 2 can be represented by the graph $G = (\{x, y\}, \varnothing, \varnothing, \varnothing, \mathscr{A}_G, \mathring{G})$ where $\mathscr{A}_G$ interprets the sort `idtf` by $\{a, b\}$ (the terms of sort `idtf`) and the sort `nat` by $\mathbb{N}$, and where $\mathring{G}(x) = \{a, 1\}$, $\mathring{G}(y) = \{b, 2\}$. For the sake of conciseness this graph can be represented by

$$G = x, \{a, 1\}\ y, \{b, 2\}.$$

With these conventions the assignment $a \coloneqq b$ can be represented by the rule $r_1 = (L_1, K_1, R_1)$ with

$$L_1 = x_1, \{a, u\}\ y_1, \{b, v\} \qquad K_1 = x_1, \{a\}\ y_1, \{b, v\} \qquad R_1 = x_1, \{a, v\}$$

that removes the value u of a before replacing it by the value v of b. Similarly the assignment $b \coloneqq a$ is represented by the rule $r_2 = (L_2, K_2, R_2)$ with

$$L_2 = x_2, \{a, u\}\ y_2, \{b, v\} \qquad K_2 = x_2, \{a, u\}\ y_2, \{b\} \qquad R_2 = y_2, \{b, u\}.$$

Note that these rules are non standard. There is exactly one matching μ_1 of r_1 in G, and one matching μ_2 of r_2 in G, as given below.

	x_1	y_1	a	b	u	v
μ_1	x	y	a	b	1	2

	x_2	y_2	a	b	u	v
μ_2	x	y	a	b	1	2

These two matchings perfectly overlap since $\mu_1(L_1) = \mu_2(L_2) = G$. Let $M = \{\mu_1, \mu_2\}$, hence $\mathrm{V}_M = \mathrm{A}_M = \varnothing$ (no vertex and no arrow is deleted), $\ell_M(x) = \{1\}$ and $\ell_M(y) = \{2\}$ (the values 1 and 2 are removed from $\mathring{G}(x)$ and $\mathring{G}(y)$ respectively), hence $G\backslash[\mathrm{V}_M, \mathrm{A}_M, \ell_M] = x, \{a\}\ y, \{b\}$. We also have $G{\uparrow_{\mu_1}} = \mu_1(R_1) = x, \{a, 2\}$ and $G{\uparrow_{\mu_2}} = \mu_2(R_2) = y, \{b, 1\}$, so that

$$G\|_M = (x, \{a\}\ y, \{b\}) \sqcup (x, \{a, 2\}) \sqcup (y, \{b, 1\}) = x, \{a, 2\}\ y, \{b, 1\}.$$

Hence the parallel transformation of G by r_1 and r_2 yields the same result as the simultaneous assignment $a, b \coloneqq b, a$ in Python: it swaps the values of a and b. Besides, this transformation preserves the semantics of r_1 and r_2 since the initial values of a and b have effectively been deleted from $\mathring{G}(x)$ and $\mathring{G}(y)$ respectively. Indeed, $\ell_M(x)\backslash\ell_M^{\uparrow}(x) = \{1\}\backslash\{2\} = \{1\}$ and $\ell_M(x)\backslash\ell_M^{\uparrow}(y) = \{2\}\backslash\{1\} = \{2\}$, hence $G\|_M$ is disjoint from $\mathrm{V}_M, \mathrm{A}_M, \ell_M\backslash\ell_M^{\uparrow}$, and M therefore has the effective deletion property.

Assume now that this transformation is applied to an environment where a and b have the same value, say 1. This is represented by $G' = x', \{a, 1\}\ y', \{b, 1\}$, and there are two obvious matchings μ_1', μ_2' of r_1, r_2 respectively in G' (the variables u and v are both matched to 1). Let $M' = \{\mu_1', \mu_2'\}$,, we see that

$$G'\|_{M'} = (x', \{a\}\ y', \{b\}) \sqcup (x', \{a, 1\}) \sqcup (y', \{b, 1\}) = x', \{a, 1\}\ y', \{b, 1\} = G'$$

as expected. But the initial values of a and b have not been deleted from $\mathring{G}(x)$ and $\mathring{G}(y)$ respectively. Yet we have

$$\ell_{M'}(x')\backslash\ell_{M'}^{\uparrow}(x') = \ell_{M'}(y')\backslash\ell_{M'}^{\uparrow}(y') = \{1\}\backslash\{1\} = \varnothing,$$

hence $G\|_M$ is trivially disjoint from $\mathrm{V}_{M'}, \mathrm{A}_{M'}, \ell_{M'} \backslash \ell^{\uparrow}_{M'}$, and M' therefore has the effective deletion property. Even in this case we agree that the semantics of r_1 and r_2 has been respected by the parallel transformation, since 1 has been deleted before it was restored.

We may ask wether it is possible to ensure from particular properties of $\mathcal{R}$ that effective deletion holds for all G and M. It is however easy to see that, given a rule that removes an object, say a vertex, and another (or the same) rule that retains some vertex, there always exists a graph G and two matchings in G that conflict on the same vertex, hence that do not have the effective deletion property. The effective deletion property should therefore be checked for every input G and M.

This naturally leads to the following definition.

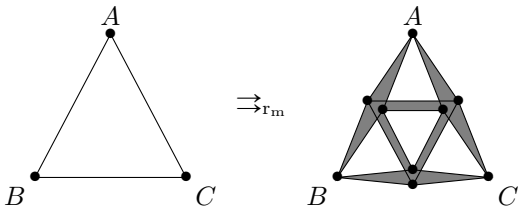
Definition 8 (full parallel rewriting). *For any finite set of rules $\mathcal{R}$, we define the relation $\rightrightarrows_{\mathcal{R}}$ of* full parallel rewriting *between graphs by stating that, for all G such that $\mathscr{M}(\mathcal{R}, G)$ has the effective deletion property, $G \rightrightarrows_{\mathcal{R}} G\|_{\mathscr{M}(\mathcal{R},G)}$.*

It can be shown that $\rightrightarrows_{\mathcal{R}}$ is *deterministic up to isomorphism*, that is, if $G \rightrightarrows_{\mathcal{R}} H$, $G' \rightrightarrows_{\mathcal{R}} H'$ and $G \simeq G'$ then $H \simeq H'$.

6 Parallel Rewriting modulo Automorphisms

Using the full set of matchings exceeds the needs of mesh refinement, see below.

Example 5. Following Example 1, we see that there are 6 matchings of $\mathrm{r_m}$ in G_0, hence the relation $\rightrightarrows_{\mathrm{r_m}}$ does not create 3 but 18 new vertices, not 9 but 54 new edges, which is illustrated as gray areas below.



We therefore wish to select a subset M of $\mathscr{M}(\mathcal{R}, G)$ for defining a rewrite relation that yields more natural and concise graphs.

In Example 5, the similarities between the 6 matchings clearly come from the symmetries of rule $\mathrm{r_m}$. These depend on the automorphisms of the graphs $\mathrm{L_m}$, $\mathrm{K_m}$ and $\mathrm{R_m}$, i.e., on the groups commonly denoted $\mathrm{Aut}(\mathrm{L_m})$, $\mathrm{Aut}(\mathrm{K_m})$ and $\mathrm{Aut}(\mathrm{R_m})$. We need to build a notion of rule automorphisms that properly accounts for the interactions between these 3 groups. We first extend the notion of automorphism groups of graphs to their subgraphs.

Definition 9 (groups $\mathrm{Aut}_G(H_1,\dots,H_n)$ and $\mathcal{S}|_H$). *For all $n \geqslant 1$, graph G and subgraphs $H, H_1, \dots, H_n \lhd G$, let*

$$\mathrm{Aut}_G(H) \stackrel{\text{def}}{=} \{\alpha \in \mathrm{Sym}(\dot{G}) \curlyvee \mathrm{Sym}(\vec{G}) \curlyvee \mathrm{Aut}(\mathscr{A}_G) \mid \alpha(H) = H\},$$

$$\mathrm{Aut}_G(H_1,\dots,H_n) \stackrel{\text{def}}{=} \bigcap_{i=1}^{n} \mathrm{Aut}_G(H_i).$$

For any $\alpha \in \mathrm{Aut}_G(H)$, we write $\alpha|_H$ for $\alpha|_{\lfloor H \rfloor}$, and for any subgroup $\mathcal{S}$ of $\mathrm{Aut}_G(H)$, let $\mathcal{S}|_H = \{\alpha|_H \mid \alpha \in \mathcal{S}\}$; this is a subgroup of $\mathrm{Aut}(H)$.

It is obvious that $\mathrm{Aut}_G(G) = \mathrm{Aut}(G)$. We see that $\mathrm{Aut}_G(H)$ is a permutation group on $\lfloor G \rfloor$, but only the graph structure of H is involved in the constraint $\alpha(H) = H$, not the structure of G.

Example 6. Take for instance

$$H \;=\; x \underset{g}{\overset{f}{\rightrightarrows}} y \quad \text{and} \quad G \;=\; x \underset{g}{\overset{f}{\rightrightarrows}} y \underset{k}{\overset{h}{\rightleftarrows}} z$$

with empty attributes. We have

$$\mathrm{Aut}(H) = \{1_H,\, (x)(y)(f\ g)\} \text{ and } \mathrm{Aut}(G) = \{1_G,\, (x)(y)(z)(f\ g)(h)(k)\}$$

We write non permuted points such as (x) in order to make the domains explicit. However, in $\mathrm{Aut}_G(H)$ the permutations of objects that do not belong to H are free, hence

$$\begin{aligned}\mathrm{Aut}_G(H) &= \{1_G,\, (x)(y)(z)(f\ g)(h)(k),\, (x)(y)(z)(f)(g)(h\ k),\\ &\qquad\qquad (x)(y)(z)(f\ g)(h\ k)\}\\ &= \mathrm{Aut}(H) \curlyvee \{(z)(h)(k),\, (z)(h\ k)\}\\ &= \mathrm{Aut}(H) \curlyvee \{(z)\} \curlyvee \{(h)(k),\ (h\ k)\}\\ &= \mathrm{Aut}(H) \curlyvee \mathrm{Sym}\{z\} \curlyvee \mathrm{Sym}\{h,\, k\}.\end{aligned}$$

Since $\mathscr{A}_H = \mathscr{A}_G$, it is easy to see that $\mathrm{Aut}_G(H) = \mathrm{Aut}(H) \curlyvee \mathrm{Sym}(\dot{G}\backslash\dot{H}) \curlyvee \mathrm{Sym}(\vec{G}\backslash\vec{H})$ always holds and thus $\mathrm{Aut}_G(H)|_H = \mathrm{Aut}(H)$. This means that, compared to the elements of $\mathrm{Aut}(H)$ which are only permutations of $\lfloor H \rfloor$, the elements of $\mathrm{Aut}_G(H)$ are *all* possible extensions of the elements of $\mathrm{Aut}(H)$ to permutations of $\lfloor G \rfloor$. This allows us to conveniently intersect the automorphism groups of joinable graphs.

Definition 10 (group $\mathrm{Aut}(r)$, relation $\approx$). *For any rule $r = (L, K, R)$, the* automorphism group *of r is $\mathrm{Aut}(r) \stackrel{\text{def}}{=} \mathrm{Aut}_{L \sqcup R}(L, K, R)|_L$. For any graph G, let $\approx$ be the equivalence relation on $\mathscr{M}(\mathcal{R}, G)$ defined by $\mu \approx \nu$ iff $\mu \circ \mathrm{Aut}(\mathrm{r}_\mu) = \nu \circ \mathrm{Aut}(\mathrm{r}_\nu)$. The equivalence class of μ is denoted $\bar{\mu}$. For any subset $M \subseteq \mathscr{M}(\mathcal{R}, G)$ we write $\bar{M}$ for the set $\bigcup_{\mu \in M} \bar{\mu}$.*

Lemma 1. $\forall \mu \in \mathscr{M}(\mathcal{R}, G)$, $\bar{\mu} = \mu \circ \mathrm{Aut}(\mathrm{r}_\mu)$.

Note that $|\bar{\mu}| \leqslant |\mathrm{Aut}(\mathrm{r}_\mu)|$ and that the equality holds if μ is injective. The more symmetric a rule is, the more matchings are likely to occur in the equivalence classes of matchings of this rule. The definition of the automorphism groups of rules has been crafted so that the isomorphism classes of the output graphs do not depend on the choice of elements in the equivalence classes of matchings.

Theorem 1. *For any graph G, any $\mathcal{M} \subseteq \mathscr{M}(\mathcal{R}, G)$ and any minimal sets M, N such that $\mathcal{M} = \bar{M} = \bar{N}$, the graphs $G\|_M$ and $G\|_N$ are isomorphic.*

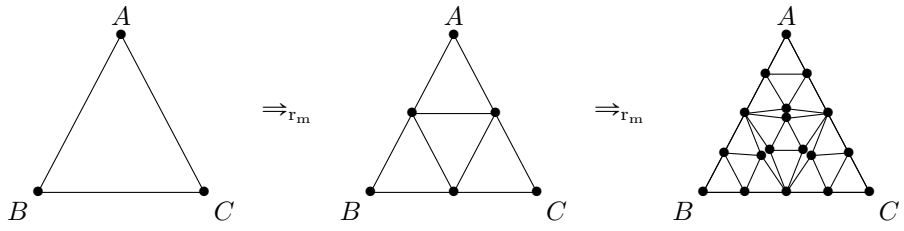
This means that the following graph rewrite relation is deterministic up to isomorphism.

Definition 11 (parallel rewriting modulo automorphisms). *For any finite set of rules $\mathcal{R}$, we define the relation $\Rightarrow_{\mathcal{R}}$ of* parallel rewriting modulo automorphisms *between graphs by stating that, for all G and minimal set M such that $\bar{M} = \mathscr{M}(\mathcal{R}, G)$ and M has the effective deletion property, $G \Rightarrow_{\mathcal{R}} G\|_M$.*

Example 7. Following Example 1, we see that the group $\mathrm{Aut}(\mathrm{K_m})$ is generated by $\{(x\ y), (x\ z)\}$ (this is $\mathrm{Sym}\{x, y, z\}$), the group $\mathrm{Aut}(\mathrm{L_m})$ is generated by $\{(x\ y)(f\ g), (x\ z)(f\ h)\}$, and $\mathrm{Aut}(\mathrm{R_m})$ is generated by $\{\rho_1, \rho_2\}$ where $\rho_1 = (x\ y)(x'\ y')(h_1\ h_2)(f_1\ g_2)(f_2\ g_1)(f_3\ g_3)$ and $\rho_2 = (x\ z)(x'\ z')(g_1\ g_2)(f_1\ h_2)(f_2\ h_1)(f_3\ h_3)$. We then use the facts that

$$\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{L_m}) = \mathrm{Aut}(\mathrm{L_m}) \curlyvee \mathrm{Sym}\{x', y', z'\} \curlyvee \mathrm{Sym}\{f_1, g_1, h_1, f_2, g_2, h_2, f_3, g_3, h_3\}$$
$$\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{K_m}) = \mathrm{Aut}(\mathrm{K_m}) \curlyvee \mathrm{Sym}\{x', y', z'\} \curlyvee \mathrm{Sym}\{f, g, h, f_1, \ldots, h_3\}$$
$$\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{R_m}) = \mathrm{Aut}(\mathrm{R_m}) \curlyvee \mathrm{Sym}\{f, g, h\}$$

to see that $\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{L_m})$ is a subgroup of $\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{K_m})$, and then we easily see that $\mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{L_m}, \mathrm{K_m}, \mathrm{R_m}) = \mathrm{Aut}_{\mathrm{L_m} \sqcup \mathrm{R_m}}(\mathrm{L_m}, \mathrm{R_m})$ is generated by $\{\rho_1 \curlyvee (f\ g), \rho_2 \curlyvee (f\ h)\}$. We thus obtain that $\mathrm{Aut}(\mathrm{r_m})$ is the group generated by $\{(\rho_1 \curlyvee (f\ g))|_{\mathrm{L_m}}, (\rho_2 \curlyvee (f\ h))|_{\mathrm{L_m}}\} = \{(x\ y)(f\ g), (x\ z)(f\ h)\}$, i.e., $\mathrm{Aut}(\mathrm{r_m}) = \mathrm{Aut}(\mathrm{L_m})$ and this group has 6 elements. The 6 matchings of $\mathrm{L_m}$ in G_0 are therefore all equivalent by $\approx$. Similarly, the 24 matchings of $\mathrm{L_m}$ in G_1 form 4 equivalence classes modulo $\approx$. This yields the following parallel rewrite steps modulo automorphisms.



7 Graph Transformations as Quotients

The last example shows that we still need to be able to merge graph items. In this section, we propose to use equivalence relations to merge graph vertices or edges as well as their possible attributes.

Definition 12 (Congruence on a Graph, quotient graph). *For any graph G, a* congruence $\mathcal{C}$ on G *is a tuple* $(\sim, \simeq, \cong)$ *where* $\cong$ *is a congruence on* $\mathscr{A}_G$ *(see [2, p. 45]) and* $\sim$, $\simeq$ *are equivalence relations on* $\dot{G}$, $\vec{G}$ *respectively, such that*

$$\forall f, g \in \vec{G},\ \text{if } f \simeq g \text{ then } \acute{G}(f) \sim \acute{G}(g) \text{ and } \grave{G}(f) \sim \grave{G}(g). \tag{2}$$

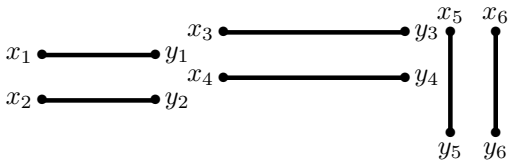
$\mathcal{C}$ is neutral *if $\cong$ is the identity relation on $\mathscr{A}_G$.*

The quotient of G by $\mathcal{C}$ *is the graph* $G/\mathcal{C} = (\dot{G}/\sim, \vec{G}/\simeq, s, t, \mathscr{A}_G/\cong, l)$ *where $\dot{G}/\sim$ and $\vec{G}/\simeq$ are the standard quotients (the sets of equivalence classes), $\mathscr{A}_G/\cong$ is the quotient algebra (see [2, p. 45]), and*

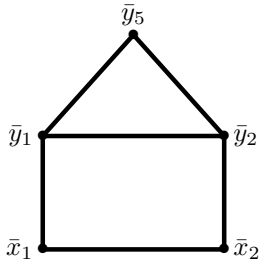
- *for any $F \in \vec{G}/\simeq$ and any $f \in F$, $s(F)$ (resp. $t(F)$) is the class of $\acute{G}(f)$ (resp. $\grave{G}(f)$) modulo $\sim$ (which by (2) depends only on the class F of f modulo $\simeq$),*
- *for any $C \in (\dot{G}/\sim) \cup (\vec{G}/\simeq)$, $l(C) \stackrel{\text{def}}{=} \bigcup_{x \in C} \{\bar{t} \mid t \in \mathring{G}(x)\}$, where $\bar{t}$ is the equivalence class of $t \in \lfloor \mathscr{A}_G \rfloor$ modulo $\cong$.*

Note that if $\mathcal{C}$ is neutral then $\mathscr{A}_{G/\mathcal{C}}$ is identified with $\mathscr{A}_G$.

Example 8. Let us consider the following graph Z consisting of six sticks. These sticks could be thought, for example, as a furniture kit. Every stick can be considered as a pair of two opposite arrows attributed by its length (not depicted), with $\lfloor \mathscr{A}_Z \rfloor = \mathbb{R}$. Each end of a stick is a vertex whose attribute is $\varnothing$.



The instructions for assembling the kit can be given as the following equivalence relation on vertices: $x_1 \sim x_3$, $x_2 \sim y_3$, $y_1 \sim x_4 \sim x_5$, $y_2 \sim y_4 \sim x_6$ and $y_5 \sim y_6$. We now consider the neutral congruence $\mathcal{Z} = (\sim, =_{\vec{Z}}, =_{\mathscr{A}_G})$, where $=_{\vec{Z}}$ is the identity relation on $\vec{G}$ (so that condition (2) is obviously satisfied). The reader can easily check that the quotient graph $Z/\mathcal{Z}$ is the following one.



By definition, a congruence is specific to a particular graph, hence we need a way to define a congruence from a graph in order to define a universal transformation by taking quotients of graphs. There are many ways this could be done, and we only propose a solution relevant to mesh refinement.

Definition 13 (The localizing congruence). *For any graph G, the* localizing congruence $\mathscr{L}_G$ *on G is the tuple $(\sim, \simeq, =_{\mathscr{A}_G})$ where*

- *for all $x, y \in \dot{G}$, $x \sim y$ iff $\mathring{G}(x) = \mathring{G}(y)$,*
- *for all $f, g \in \vec{G}$, $f \simeq g$ iff $\acute{G}(f) \sim \acute{G}(g)$ and $\grave{G}(f) \sim \grave{G}(g)$,*
- *$=_{\mathscr{A}_G}$ is the identity relation on $\mathscr{A}_G$.*

We write $\gtrsim_{\mathscr{L}}$ for the binary relation on graphs defined by $G \gtrsim_{\mathscr{L}} (G/\mathscr{L}_G)$ for all graphs G.

In this transformation, the attributes of the vertices act as coordinates in the sense that there can be only one vertex (or point) at each coordinate, and only one arrow from one point to another. Note that, since $\mathscr{L}_G$ is neutral, then obviously $\mathscr{A}_G = \mathscr{A}_H$ whenever $G \gtrsim_{\mathscr{L}} H$.

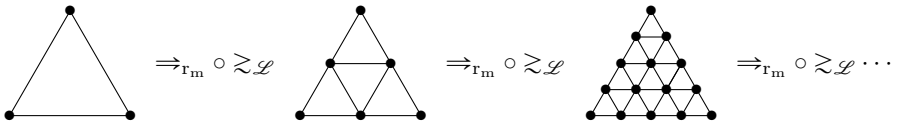
Example 9. In Example 1, it was not necessary to define precisely which algebra was considered since all attributes were empty. We now give more substance to the mesh graphs by assuming that every vertex is attributed with its coordinates in the affine plane, i.e., by a singleton containing an element of $\mathbb{R}^2$. The only operation we need is the function that returns the coordinates of the middle of two points, hence we take Σ with a function symbol $\texttt{mid}$ of arity 2, and we consider the Σ-algebra $\mathcal{P}$ with carrier set $\mathbb{R}^2$ where $\texttt{mid}$ is interpreted as the function that to any $(x, y) \in \mathbb{R}^2$ and $(x', y') \in \mathbb{R}^2$ maps $(\frac{x+x'}{2}, \frac{y+y'}{2}) \in \mathbb{R}^2$. We start with the graph G_0 as in Example 1, but with $\mathscr{A}_{G_0} = \mathcal{P}$, $\mathring{G}_0(A)$ is a singleton that contains the coordinates of A (an element of $\mathbb{R}^2$), and similarly for vertices B and C.

In order to match these coordinates we also need to use variables in the rule $\mathrm{r_m}$, hence we consider 3 distinct variables $u, v, w \in \mathscr{V}$ and let $X = \{u, v, w\}$. We consider the graphs $\mathrm{L_m}$, $\mathrm{K_m}$, $\mathrm{R_m}$ of Example 1 but with the algebra $\mathscr{T}(\Sigma, X)$ and with the following attributes on vertices:

- $\mathring{\mathrm{L}}_\mathrm{m}(x) = \mathring{\mathrm{K}}_\mathrm{m}(x) = \mathring{\mathrm{R}}_\mathrm{m}(x) = \{u\}$, $\mathring{\mathrm{L}}_\mathrm{m}(y) = \mathring{\mathrm{K}}_\mathrm{m}(y) = \mathring{\mathrm{R}}_\mathrm{m}(y) = \{v\}$ and $\mathring{\mathrm{L}}_\mathrm{m}(z) = \mathring{\mathrm{K}}_\mathrm{m}(z) = \mathring{\mathrm{R}}_\mathrm{m}(z) = \{w\}$. These attributes are therefore not modified by the rule.
- We must also compute the coordinates of the new vertices x', y', z' created by $\mathrm{R_m}$. One difficulty is that in the algebra of terms $\texttt{mid}(u, v)$ is different from $\texttt{mid}(v, u)$, and if we choose one then we necessarily loose some automorphisms of the rule. The solution is to take both, that is

$$\mathring{\mathrm{R}}_\mathrm{m}(x') = \{\texttt{mid}(v, w), \texttt{mid}(w, v)\},$$
$$\mathring{\mathrm{R}}_\mathrm{m}(y') = \{\texttt{mid}(u, w), \texttt{mid}(w, u)\},$$
$$\mathring{\mathrm{R}}_\mathrm{m}(z') = \{\texttt{mid}(v, u), \texttt{mid}(u, v)\}.$$

With these attributes it is easy to see that $\mathrm{Aut}(\mathrm{r_m})$ is generated by the two permutations $(x\ y)(f\ g)(u\ v)$ and $(x\ z)(f\ h)(u\ w)$, and has 6 elements. The 6 matchings of $\mathrm{L_m}$ in G_0 are all $\approx$-equivalent to the matching $\mu \in \mathscr{M}(\mathrm{r_m}, G_0)$ with the same images of vertices and arrows as in Example 1, and where $\mathring{\mu}(x)$ is the coordinate of A, and similarly for y and z. Hence we have $G_0 \Rightarrow_{\mathrm{r_m}} G_1$, where to every vertex is attributed the coordinate of the corresponding point of $\mathcal{P}$. Applying $\Rightarrow_{\mathrm{r_m}}$ still yields a graph with too many vertices and edges, though with correct coordinates, hence quotienting this graph with its localizing congruence yields the graph G_2. We thus see that



ad infinitum, as we expected (though these graphs contain attributes that are not depicted). Note that $\rightrightarrows_{\mathrm{r_m}} \circ \gtrsim_{\mathscr{L}}$ yields the same result as $\Rightarrow_{\mathrm{r_m}} \circ \gtrsim_{\mathscr{L}}$, though in a less efficient way since $\mathrm{Aut}(\mathrm{r_m})$ needs only be computed once.

8 Related Work and Concluding Remarks

Parallel graph rewriting has already been considered in the literature. In the mid-seventies, H. Ehrig and H.-J. Kreowski [14] tackled the problem of parallel graph transformations and proposed conditions under which parallel graph transformations could be sequentialized and how sequential independent graph transformations could be parallelized. This pioneering work has been considered for several algebraic graph transformation approaches, see, e.g., the most recent contributions [9, 22, 21] or Volume 3 of the Handbook of Graph Grammars and Computing by Graph Transformation [12]. However, this stream of work departs drastically from our goal where parallel graph transformations are not aimed to be sequentialized.

Non independent parallel graph transformations has been considered in the Double-Pushout setting, see e.g. [24] where rules can be amalgamated by agreeing on common deletions and preservations. However, the amalgamation technique does not allow the amount of overlaps achieved in the present framework. Indeed, the effective deletion property makes it possible for one rule to delete an item that is matched but not deleted by another (non standard) rule. This is an essential feature for instance in cellular automata where the state of a cell can be modified by one rule and only consulted by others (see [5]).

In [19], a framework based on the algebraic Single-Pushout approach has been proposed and where parallel transformations consider only matchings provided by a control flow mapping. The users can solve the possible conflicts between the rules by providing the right control flow. More recently, a parallel graph rewrite relation has been defined in [11] for a special kind of graphs called port-graphs. Unfortunately, such graphs are not closed under parallel graph transformation,

in the sense that a port-graph can be rewritten in a structure which is not a port-graph. In addition, conditions for avoiding conflicts in parallel transformations have been defined over the considered rewrite rules, which limits drastically the class of the considered systems. The present framework provides more abstract and more general conditions over matchings that ensure a correct definition of parallel graph transformations for a large class of systems.

In [23, chapter 14], parallel graph transformations have been studied in order to improve the operational semantics of the functional programming language CLEAN [16]. In that contribution, the authors do not deal with true parallelism but rather have an interleaving semantics. This particularly entails that their parallel rewrite steps can be simulated by sequential ones. This is also the case for other frameworks where massive parallel graph transformations is defined so that it can be simulated by sequential rewriting e.g., [10, 21, 20].

Graph equivalence has already been used to encode vertex merging as in [3] where the notion of e-graphs has been proposed. An e-graph is a pair $(G, \sim)$ of a hypergraph and an equivalence over vertices. Contrary to our framework, quotient graphs are not used per se as objects to be transformed. Furthermore, our notion of equivalence over graphs is more general since it can be defined either on vertices, arrows or even attributes.

Transforming a graph by using simultaneously several rules in parallel is not an easy task. As mentioned above, most of the proposals in the literature consider parallel transformations that can be sequentialized. In this paper, we have developed a new framework where true parallel graph transformations are defined following an algorithmic approach. We proposed deterministic parallel rewrite relations, particularly one based on the notion of automorphism groups of rules. Furthermore, we defined the notion of *effective deletion property* of matchings which ensures that these relations are well-behaved, even when the overlappings of matches forbids sequentialization, as illustrated by the mesh refinement rule $\mathrm{r_m}$. The proposed rewrite relations may be used in several contexts such as extensions of L-systems to dynamic graph structures (see, e.g., [25, 17, 18]). For the sake of simplicity we have not addressed here the problem of the finiteness of the graphs obtained by parallel rewriting, see [5] for a discussion and results on this subject.

The considered rewrite systems could be enriched by means of new features such as vertex and edge cloning as proposed in [7, 8]. This is possible in an algebraic framework, see [6]. Future work also includes implementation issues, particularly for the parallel rewrite relation up to automorphisms. The present framework has been designed so that the automorphism groups of rules are finite permutation groups, thus paving the way to efficient implementations through the methods of Algorithmic Group Theory.

References

1. Andrew, R.B., Sherman, A.H., Weiser, A.: Some refinement algorithms and data structures for regular local mesh refinement (1983)

2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
3. Baldan, P., Gadducci, F., Montanari, U.: Concurrent rewriting for graphs with equivalences. In: CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4137, pp. 279–294. Springer (2006)
4. Boronat, A., Heckel, R., Meseguer, J.: Rewriting logic semantics and verification of model transformations. In: Chechik, M., Wirsing, M. (eds.) Fundamental Approaches to Software Engineering. pp. 18–33. Springer (2009)
5. Boy de la Tour, T., Echahed, R.: A set-theoretic framework for parallel graph rewriting. CoRR (abs/1808.03161) (2018)
6. Boy de la Tour, T., Echahed, R.: True parallel graph transformations: an algebraic approach based on weak spans. CoRR (abs/1904.08850) (2019)
7. Brenas, J.H., Echahed, R., Strecker, M.: Verifying graph transformation systems with description logics. In: 11th ICGT. LNCS, vol. 10887, pp. 155–170. Springer (2018)
8. Corradini, A., Duval, D., Echahed, R., Prost, F., Ribeiro, L.: AGREE - algebraic graph rewriting with controlled embedding. In: 8th ICGT. LNCS, vol. 9151, pp. 35–51. Springer (2015)
9. Corradini, A., Duval, D., Löwe, M., Ribeiro, L., Machado, R., Costa, A., Azzi, G.G., Bezerra, J.S., Rodrigues, L.M.: On the essence of parallel independence for the double-pushout and sesqui-pushout approaches. In: Heckel, R., Taentzer, G. (eds.) Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig. LNCS, vol. 10800, pp. 1–18. Springer (2018)
10. Echahed, R., Janodet, J.: Parallel admissible graph rewriting. In: Recent Trends in Algebraic Development Techniques, 13th International Workshop WADT'98, Selected Papers. LNCS, vol. 1589, pp. 122–137. Springer (1999)
11. Echahed, R., Maignan, A.: Parallel graph rewriting with overlapping rules. CoRR (abs/1701.06790) (2017)
12. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G. (eds.): Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism and Distribution. World Scientific (1999)
13. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006)
14. Ehrig, H., Kreowski, H.: Parallelism of manipulations in multidimensional information structures. In: Mathematical Foundations of Computer Science. LNCS, vol. 45, pp. 284–293. Springer (1976)
15. Engelfriet, J., Rozenberg, G.: Node replacement graph grammars. In: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. pp. 1–94 (1997)
16. Group, S.T.R.: The Clean Home Page. Radboud University, Nijmegen
17. II, K.C., Lindenmayer, A.: Parallel graph generating and recurrence systems for multicellular development. International Journal of General Systems **3**(1), 53–66 (1976). https://doi.org/10.1080/03081077608934737
18. Janssens, D., Rozenberg, G., Verraedt, R.: On sequential and parallel node-rewriting graph grammars. Computer Graphics and Image Processing **18**(3), 279–304 (1982). https://doi.org/10.1016/0146-664X(82)90036-3, https://doi.org/10.1016/0146-664X(82)90036-3

19. Kniemeyer, O., Barczik, G., Hemmerling, R., Kurth, W.: Relational growth grammars - A parallel graph transformation approach with applications in biology and architecture. In: Third International Symposium AGTIVE, Revised Selected and Invited Papers. pp. 152–167 (2007)
20. Kreowski, H., Kuske, S.: Graph multiset transformation: a new framework for massively parallel computation inspired by DNA computing. Natural Computing **10**(2), 961–986 (2011)
21. Kreowski, H., Kuske, S., Lye, A.: A simple notion of parallel graph transformation and its perspectives. In: Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig. LNCS, vol. 10800, pp. 61–82. Springer (2018)
22. Löwe, M.: Characterisation of parallel independence in AGREE-rewriting. In: 11th ICGT. LNCS, vol. 10887, pp. 118–133. Springer (2018)
23. Plasmeijer, R., Eekelen, M.V.: Functional Programming and Parallel Graph Rewriting. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1993)
24. Taentzer, G.: Parallel high-level replacement systems. TCS: Theoretical Computer Science **186**, 43–81 (1997)
25. Wolfram, S.: A new kind of science. Wolfram-Media (2002)