Accretive Computation of Global Transformations of Graphs

Alexandre Fernandez, Luidnel Maignan, Antoine Spicher

Accretive Computation of Global Transformations of Graphs

Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher

Univ Paris Est Creteil, LACL, 94000, Creteil, France
firstname.lastname@u-pec.fr

Abstract. The framework of global transformations aims at describing synchronous rewriting systems on a given data structure. In this work we focus on the data structure of graphs. Global transformations of graphs are defined and a local criterion is given for a rule system to extend to a graph global transformation. Finally we present an algorithm, with its correction, which computes online the global transformation of a finite graph in an accretive manner.

1 Introduction

In [10], the model of *Global Transformations* is introduced with the purpose of capturing the essence of spatially extended dynamical systems which are simultaneously *local*, *synchronous* and *deterministic*. A paradigmatic example of such systems are cellular automata where a set of *local* rules drives the *synchronous* evolution of cells. We advocate that this kind of systems is not restricted to the unique case of cellular automata. It can be generalized to any kind of spatial organizations once the common and generic computation mechanism underlying such systems is identified. Global transformations provide a formal framework for describing this mechanism.

A classical approach in rewriting is to describe a local evolution rule as the removal of a sub-part of a given input followed by a gluing of a new part with the rest of the input, and the application of a rule can potentially prevent the application of another rule. In contrast, in the global transformation approach, the input is totally discarded. All applicable rules are applied together, their results being glued together without any reference to the input. The way these pieces of result are connected is driven by the way the matched sub-parts were originally connected within the input. Formally, this connectivity of the state space is captured by a category and the mechanism induces the local evolution rules to extend to a functor over that category.

The applicability of global transformations to capture *local*, *synchronous* and *deterministic* dynamical systems has been shown with examples like dynamic meshes operations, acting on abstract cellular complexes [10], and like deterministic Lindenmayer [5] systems and cellular automata [6], acting on formal words. In parallel of these works, a software development has been undertaken to demonstrate the practicability of the global transformation approach. The

tool aims at embedding several non trivial data structures and at providing a single generic algorithm for computing global transformations for all these cases at once.

The present work aims at introducing a part of this development. More particularly, we are interested in the description of the global transformation computation algorithm with a focus on the graph data structure. The corresponding underlying category is the category of graphs with subgraph relationship as morphisms. Formally, these morphisms are the monomorphisms of the usual category of graphs. The choice of working with monomorphisms only is a key element of our work. Indeed, the proposed algorithm computes the output of a global transformation application *accretively*: local evolutions contribute one after the other to the computation of the output by only adding fresh matter to the current result. This strategy saves memory during the computation since it avoids the generation of different instances of the same final element (vertex or edge) that must be merged once they have been identified. We give sufficient conditions for a collection of rules on graphs to only involves monomorphisms so that the accretive algorithm can be used.

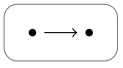
The article is organized as follows. Section 2 reminds general facts about graphs in category theory. After defining in Section 3 global transformations of graphs, Section 4 gathers the paper contributions: we start with the aforementioned sufficient conditions; then we present the proposed online accretive algorithm; we finish with its correction before conclusion.

2 Preliminaries

The reader is assumed to be familiar with the definitions of categories, functors, monomorphisms, comma categories, diagrams, cocones and colimits. Refer to [9] for details. These constructions are also pedagogically introduced in the context of global transformation in [5].

We work with unlabeled directed multigraphs that we refer to as *graphs* for simplicity. Let **Graph** be the usual category of graphs and **GraphI** be its subcategory consisting of all graphs but only monomorphisms of **Graph**. We write $\mathrm{U} : \mathbf{GraphI} \to \mathbf{Graph}$ for the obvious forgetful functor. For clarity, morphisms of **GraphI** and monomorphisms of **Graph** are written $g \hookrightarrow g'$.

Given a graph g, each of its elements (vertices and edges) corresponds bijectively to a morphism in **Graph** from an atomic graph, the single-vertex graph or the single-edge graph. In order to work in **GraphI**, we make a distinction between proper edges and self-loops and consider $\mathrm{A} = \{\mathrm{d}, \mathrm{e}, \mathrm{s}\}$ as set of atomic graphs where

$\mathrm{d} =$  , $\mathrm{e} =$  and $\mathrm{s} =$  .

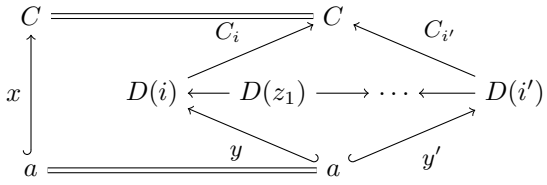
Monomorphisms in **Graph** are precisely the morphisms such that their vertex and edge components are injective. This characterization can be expressed directly within **Graph** by means of atomic graphs.

Proposition 1. *Given two graphs g and g' and a morphism $m : g \to g'$, m is a monomorphism if and only if m is an* atomic-monomorphism*, i.e. for all $a \in \mathrm{A}$ and all pairs of monomorphisms $p_1, p_2 : a \hookrightarrow g$ such that $m \circ p_1 = m \circ p_2$, we have $p_1 = p_2$.*

The category **Graph** is cocomplete. Given any diagram $D : \mathbf{I} \to \mathbf{Graph}$, we write $C = \mathrm{Colim}(D)$ for the colimit of D; C also abusively designates the apex and $C_i : D(i) \to C$ the associated cocone components for any $i \in \mathbf{I}$.

The usual description of colimits based on equivalence classes of vertices and edges can be rephrased in terms of *zig-zag* as follows. A *zig-zag* z in a category $\mathbf{C}$ is given by some natural number $|z|$, a sequence $\langle z_i \rangle_{0 \leq i \leq |z|}$ of $|z|+1$ objects of $\mathbf{C}$ and a sequence $\langle \overline{z}_i \rangle_{0 \leq i \leq |z|-1}$ of morphisms in $\mathbf{C}$ of the form $z_0 \to z_1 \leftarrow z_2 \to z_3 \cdots z_{|z|}$ or $z_0 \leftarrow z_1 \to z_2 \leftarrow z_3 \cdots z_{|z|}$. Given a functor $F : \mathbf{C} \to \mathbf{D}$ and two morphisms $g : c \to F(z_0)$ and $g' : c \to F(z_{|z|})$, we write $F(z)$ for the zig-zag defined with $|F(z)| = |z|$, $F(z)_i = F(z_i)$, and $\overline{F(z)}_i = F(\overline{z}_i)$. Given two morphisms $f_0 : c \to z_0$ and $f_{|z|} : c \to z_{|z|}$, z is said to *link* f_0 *and* $f_{|z|}$ if there is a sequence $\langle f_i : c \to z_i \rangle_{1 \leq i \leq |z|-1}$ of morphisms such that, for any $i \in \{0, \ldots, |z|-1\}$, $f_i = \overline{z}_i \circ f_{i+1}$ or $\overline{z}_i \circ f_i = f_{i+1}$ depending on the direction of $\overline{z}_i$. We say that z *links* g *and* g' *through* F if the zig-zag $F(z)$ links g and g'.

Proposition 2. *For any diagram $D : \mathbf{I} \to \mathbf{Graph}$ of small domain $\mathbf{I}$ with $C = \mathrm{Colim}(D)$, any $a \in \mathrm{A}$ and any $x : a \hookrightarrow C$, there is at least one pair of $\langle i \in \mathbf{I}, y : a \hookrightarrow D(i) \rangle$ such that $x = C_i \circ y$, and any two such pairs $\langle i, y \rangle$ and $\langle i', y' \rangle$ have a zig-zag z in $\mathbf{I}$ that links y and y' through D.*



3 Global Transformations

In the section, we adapt the definitions of global transformations given in [5,10] to fit with the context of graphs.

Specification of Global Transformations. As a rewriting system, the specification of a global transformation is based on a set of rules. Each rule $\gamma = l \Rightarrow r$ expresses that any occurrence of the left hand side (l.h.s.) l in the input object produces the corresponding right hand side (r.h.s.) r in the output. The main feature of global transformations is the promotion of this set of rules to a *category* whose morphisms describe inclusion of rules. Such a *rule inclusion* $i : \gamma_1 \to \gamma_2$ from a sub-rule $\gamma_1 = l_1 \Rightarrow r_1$ to a super-rule $\gamma_2 = l_2 \Rightarrow r_2$ expresses how an occurrence of l_1 in l_2 is locally transformed into an occurrence of r_1 in r_2. So a rule inclusion is a pair $i = \langle i_l : l_1 \to l_2, i_r : r_1 \to r_2 \rangle$. Formally, such a presentation is captured by a category and two functors.

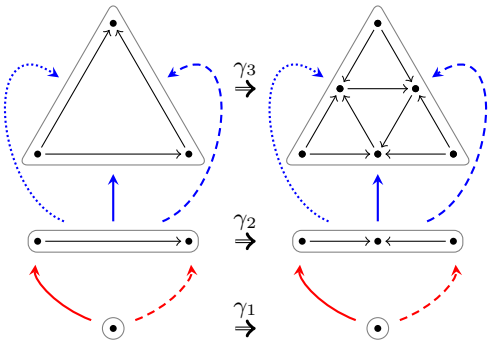


Fig. 1: Sierpinsky Rule System.

Definition 1. *A* rule system $T = \langle \mathbf{\Gamma}_T, \mathrm{L}_T, \mathrm{R}_T \rangle$ *is defined by:*

- *a category* $\mathbf{\Gamma}_T$ *whose objects are called* rules*,*
- *a full embedding functor* $\mathrm{L}_T : \mathbf{\Gamma}_T \to \mathbf{GraphI}$*, called the* l.h.s. functor*, and*
- *a functor* $\mathrm{R}_T : \mathbf{\Gamma}_T \to \mathbf{GraphI}$ *called the* r.h.s. functor*.*

The subscript T *is often omitted since this does not lead to any confusion.*

Figure 1 illustrates a global transformation specification for generating a Sierpinski gasket. The rule system is composed of 3 rules transforming locally nodes (γ_1), edges (γ_2) and none cyclic triangles (γ_3). These rules are related by 5 rule inclusions: in red for the 2 occurrences of γ_1 in γ_2 and in blue for the 3 occurrences of γ_2 in γ_3. Compounds, identities and symetries are not depicted. Each inclusion rule is represented as two graph inclusions, the one between l.h.s. and the other between r.h.s.; pairings are depicted using line styles.

Computing with Global Transformations. Given a rule system T, its application on an arbitrary graph g is a three-step process. An illustration is given Fig. 2 based on the rules of Fig. 1.

1. *Pattern matching* which consists in decomposing the input graph by mean of the rule l.h.s. It results a collection of l.h.s. instances, also called matches, structured by rule inclusions. This step is achieved by considering the comma category L_T/g. See the top row of Fig. 2 for an illustration. Formally, the figure is a representation of $\mathrm{L}_T \circ \mathrm{Proj}[\mathrm{L}_T/g]$ where Proj designates the first projection of the comma category mapping each instance $\langle \gamma \in \mathbf{\Gamma}_T, i : \mathrm{L}_T(\gamma) \hookrightarrow g \rangle$ to the used rule γ. Notice the role of the rule inclusion arrows (in red and blue) which are reminiscent of the input structure.
2. *Local application of rules* which consists in locally transforming each found l.h.s. into its corresponding r.h.s., the structure being conserved thanks to rule inclusions. This step is achieved by applying the r.h.s. functor R_T on each rule occurrence: $\mathrm{R}_T \circ \mathrm{Proj}[\mathrm{L}_T/g]$. See the right column of Fig. 2 for an illustration.

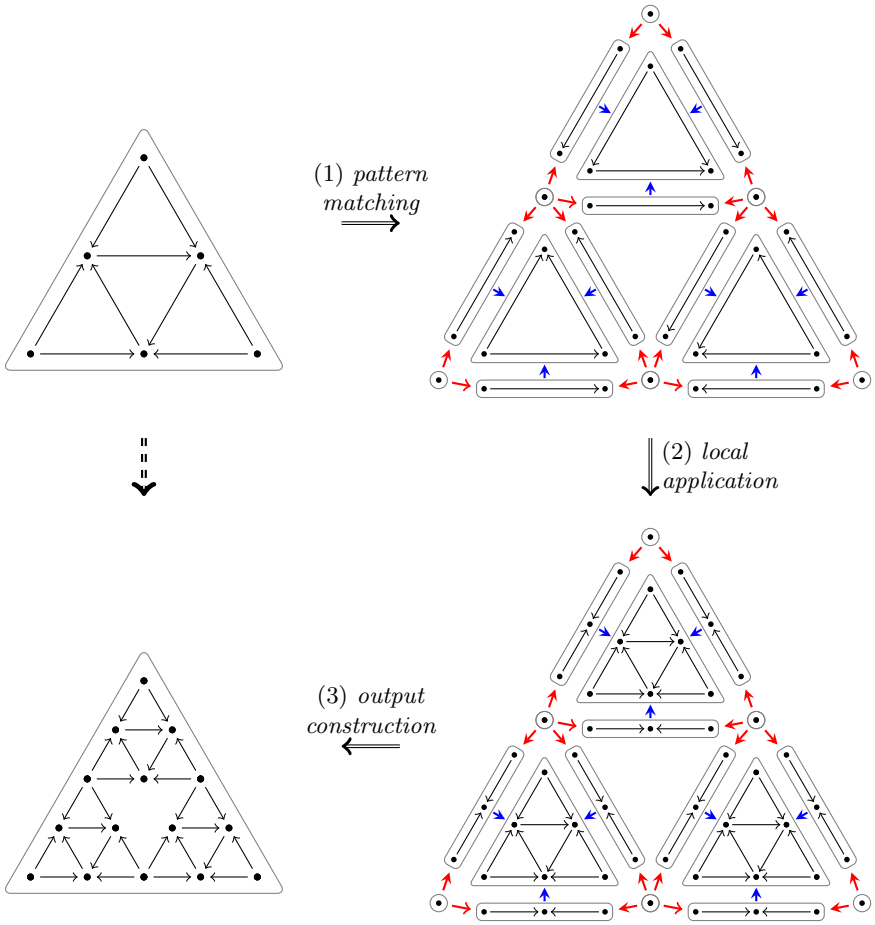


Fig. 2: Step of computation of the Sierpinsky gasket using the rules of Fig. 1.

3. *Output construction* which consists in assembling the output graph from the structured collection of r.h.s. The inclusions take here their full meaning as they are used to align the r.h.s. and drive the merge. See the bottom row of Fig. 2 for an illustration. The resulting graph is formally the apex of a cocone from the diagram defined in the previous step which we used to obtain by colimit [10,5]. Since colimits are only guaranteed in **Graph**, we consider the following functor $\overline{T} : \mathbf{GraphI} \to \mathbf{Graph}$:

$$\overline{T}(-) = \mathrm{Colim}(\mathrm{D}_T(-)) \qquad \text{with } \mathrm{D}_T(-) = \mathrm{U} \circ \mathrm{R}_T \circ \mathrm{Proj}[\mathrm{L}_T/-] \qquad (1)$$

using the forgetful functor U, $\overline{T}(g)$ being the result of the application.

Remark 1. Notice that $\overline{T}$ is a complete functor also acting on morphisms. Consider a monomorphism $m : g \hookrightarrow g'$. By definition of colimits, $\overline{T}(g)$ is the uni-

versal cocone with components $\overline{T}(g)_{\langle\gamma,i\rangle} : \mathrm{U}(\mathrm{R}_T(\gamma)) \to \overline{T}(g)$ for each instance $\langle\gamma, i : \mathrm{L}_T(\gamma) \hookrightarrow g\rangle \in \mathrm{L}_T/g$. We have a similar construction for $\overline{T}(g')$ which gives rise to a cocone C as the restriction of $\overline{T}(g')$ on the diagram of $\overline{T}(g)$. Formally, C is defined with apex $C = \overline{T}(g')$ and components $C_{\langle\gamma,i\rangle} = \overline{T}(g')_{\langle\gamma,m\circ i\rangle}$. The image $\overline{T}(m)$ is then the mediating morphism from the colimit $\overline{T}(g)$ to C.

We focus on those rule systems where the results stay inside **GraphI**, *i.e.* such that all previous mediating morphisms are monomorphisms. This leads to the following definition of global transformation for **GraphI**.

Definition 2. *A* global transformation T *is a rule system such that* $\overline{T}$ *factors through the forgetful functor* $\mathrm{U} : \mathbf{GraphI} \to \mathbf{Graph}$. *In this case, we denote* $T : \mathbf{GraphI} \to \mathbf{GraphI}$ *the functor such that* $\mathrm{U} \circ T = \overline{T}$.

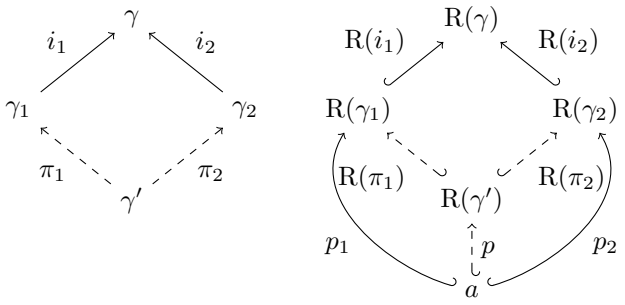
4 Incremental Global Transformations and Accretion

Now that the framework is set up, let us begin the algorithmic description of the application of a global transformation by an efficient *accretive* mechanism. Efficiency is obtained by producing each r.h.s. as soon as each l.h.s. is found, in an online fashion, and by arranging the discovery of l.h.s. so as to avoid useless duplication, fusion and associated bookkeeping of the same final element. Metaphorically, "matter is only added", each piece of matter attracting other closely related pieces of matter. This mechanism relies strongly on the fact that global transformations are not just rule systems, but actually manipulate only monomorphisms. We begin in Section 4.1 by providing a criterion ensuring that a rule system is indeed a global transformation. We then explain how the categorical concepts of Section 3 are represented computationally (Section 4.2). This is followed by a high-level (Section 4.3) then detailed description (Section 4.4) of the algorithm with the main elements of its proof of correction (Section 4.5).

4.1 Incremental Rule Systems and Global Transformations

We are interested in giving sufficient conditions for rule systems to be global transformations. The following conditions prevent a super-rule to merge by itself the r.h.s. of its sub-rules. In other words, the rule *only adds fresh matter* to the r.h.s. of its sub-rules in an incremental way. A positive expression of this constraint is as follows: if the r.h.s. of two rules overlap in the r.h.s. of a common super-rule, this overlap must have been required by some common sub-rules.

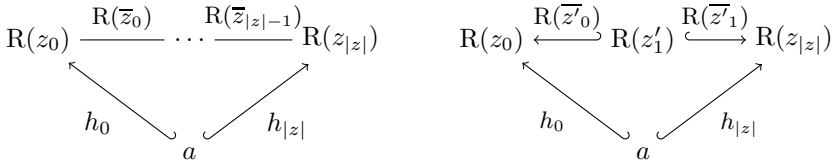
Definition 3. *Given a rule system* T*, we say that a rule* $\gamma \in \mathbf{\Gamma}_T$ *is incremental if for any two sub-rules* $\gamma_1 \xrightarrow{i_1} \gamma \xleftarrow{i_2} \gamma_2$ *in* $\mathbf{\Gamma}_T$*, any atomic graph* $a \in \mathrm{A}$*, and any* $\mathrm{R}(\gamma_1) \xleftarrow{p_1} a \xrightarrow{p_2} \mathrm{R}(\gamma_2)$ *such that* $\mathrm{R}(i_1) \circ p_1 = \mathrm{R}(i_2) \circ p_2$*, there are some* $\gamma_1 \xleftarrow{\pi_1} \gamma' \xrightarrow{\pi_2} \gamma_2$ *and* $p : a \hookrightarrow \mathrm{R}(\gamma')$ *such that the following diagrams commute.*



A rule system T is said incremental *if every $\gamma \in \mathbf{\Gamma}_T$ is incremental.*

We now establish that any incremental rule system is a global transformation. For this purpose, consider the following lemma which allows to reduce to a span any a zig-zag involved in the computation of the colimit of D_T.

Lemma 1. *Given an incremental rule system $T = \langle \mathbf{\Gamma}, \mathrm{L}, \mathrm{R} \rangle$, a zig-zag z in $\mathbf{\Gamma}$, an atomic graph $a \in \mathrm{A}$ and two monomorphisms $p_0 : a \hookrightarrow \mathrm{R}(z_0)$, $p_{|z|} : a \hookrightarrow \mathrm{R}(z_{|z|})$ such that z links h_0 and $p_{|z|}$, there is a zig-zag z' in $\mathbf{\Gamma}$ of the form $z_0 = z'_0 \leftarrow z'_1 \rightarrow z'_2 = z_{|z|}$ that also links p_0 and $p_{|z|}$.*



Proof. This is proved by induction on the length of z. We show the base case with $|z| = 0$ by taking $z' = \langle id_{z_0}, id_{z_0} \rangle$. For the induction case we assume that the proposition is true for any zig-zag of size k and take z of size $k+1$. Then we have two cases that depends on the direction of the first morphism of z:

- If $\overline{z}_0 : z_1 \rightarrow z_0$ we can apply the induction hypothesis on the zig-zag $y = \langle \overline{z}_1, \ldots \overline{z}_{k+1} \rangle$ to get a zig-zag y' of the form $z_1 = y'_0 \leftarrow y'_1 \hookrightarrow y'_2 = z_{k+1}$ that links h_1 and h_{k+1}. Then we take $z' = \langle \overline{z}_0 \circ \overline{y'}_0, \overline{y'}_1 \rangle$ to conclude this case.
- If $\overline{z}_0 : z_0 \rightarrow z_1$ we first apply the induction hypothesis on $y = \langle \overline{z}_1, \ldots \overline{z}_{k+1} \rangle$ to get a zig-zag y' of the form $z_1 = y'_0 \leftarrow y'_1 \hookrightarrow y'_2 = z_{k+1}$ that links h_1 and h_{k+1}. Now observe that we have the commutative square $\mathrm{R}(\overline{z}_0) \circ h_0 = \mathrm{R}(\overline{y'}_0) \circ h'$. Applying Def. 3 on this square, we get a quadruplet $\langle \gamma' \in \mathbf{\Gamma}, \pi_1 : \gamma' \rightarrow z_0, \pi_2 : \gamma' \rightarrow y'_1, h : a \rightarrow \mathrm{R}(\gamma') \rangle$ such that $\mathrm{R}(\overline{z}_0) \circ \mathrm{R}(\pi_1) = \mathrm{R}(\overline{y'}_0) \circ \mathrm{R}(\pi_2)$, $h_0 = \mathrm{R}(\pi_1) \circ h$, and $h' = \mathrm{R}(\pi_2) \circ h$. Here the wanted z' is $\langle \pi_1, \overline{y'}_1 \circ \pi_2 \rangle$. □

Theorem 1. *Any incremental rule system is a global transformation.*

Proof. Consider the setting of Remark 1. We need to show that $\overline{T}(m)$ is a monomorphism for $m : g \hookrightarrow g'$. Using Proposition 1, it is enough to show that

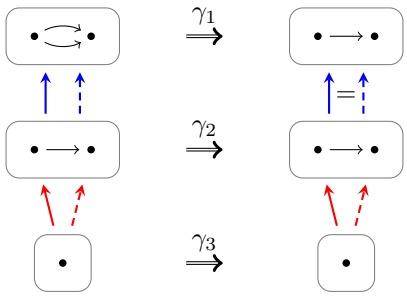
$\overline{T}(m)$ is an atomic-monomorphism. Take any two morphisms $p_1, p_2 : a \hookrightarrow \overline{T}(g)$ for $a \in \mathrm{A}$ such that $\overline{T}(m) \circ p_1 = \overline{T}(m) \circ p_2$. We are left to show that $p_1 = p_2$.

Take $k \in \{1, 2\}$. By Proposition 2, p_k factors through some cocone component of $\overline{T}(g)$. Say $p_k = \overline{T}(g)_{\langle \gamma_k, m_k \rangle} \circ q_k$ where $\langle \gamma_k, m_k \rangle$ is a object of L/g. So $\overline{T}(m) \circ p_k = \overline{T}(m) \circ \overline{T}(g)_{\langle \gamma_k, m_k \rangle} \circ q_k = \overline{T}(g')_{\langle \gamma_k, m \circ m_k \rangle} \circ q_k$. Since $\overline{T}(m) \circ p_1 = \overline{T}(m) \circ p_2$ then $\overline{T}(g')_{\langle \gamma_1, m \circ m_1 \rangle} \circ q_1 = \overline{T}(g')_{\langle \gamma_2, m \circ m_2 \rangle} \circ q_2$. Using Proposition 2 on the latter equality into the colimit $\overline{T}(g')$, there is a zig-zag z in L/g' from $\langle \gamma_1, m \circ m_1 \rangle$ to $\langle \gamma_2, m \circ m_2 \rangle$ that links q_1 and q_2 through $\mathrm{D}(g')$. The zig-zag $\mathrm{Proj}[\mathrm{L}/g'](z)$ in Γ obviously links q_1 and q_2 through $\mathrm{U} \circ \mathrm{R}$. Since the rule system T is incremental, we apply Lemma 1 to obtain a zig-zag z' in Γ of the form $\gamma_1 = z'_0 \hookleftarrow z'_1 \hookrightarrow z'_2 = \gamma_2$ that links q_1 and q_2 through $\mathrm{U} \circ \mathrm{R}$. Let us define the zig-zag z'' in L/g to be $\langle \gamma_1, m_1 \rangle \hookleftarrow \langle z'_1, h \rangle \hookrightarrow \langle \gamma_2, m_2 \rangle$ where $h = m_1 \circ \mathrm{L}(\overline{z'}_1) = m_2 \circ \mathrm{L}(\overline{z'}_2)$ and $\overline{z''}_j = \langle \overline{z'}_k, m_k \rangle$ for $k \in \{1, 2\}$. Clearly, $\mathrm{Proj}[\mathrm{L}/g](z'') = z'$, so z'' links q_1 and q_2 through $\mathrm{D}(g)$. By commutation properties of cocones over $\mathrm{D}(g)$, we have that $\overline{T}(g)_{\langle \gamma_1, m_1 \rangle} \circ q_1 = \overline{T}(g)_{\langle \gamma_2, m_2 \rangle} \circ q_2$, which implies $p_1 = p_2$ as wanted. □

The suffficient conditions of Definition 3 are definitively not necessary.

Proposition 3. *There exist non-incremental global transformations.*

Proof. Consider the following rule system as a counter-example:



Let $e_1 : \gamma_2 \to \gamma_3$ the plain arrow into γ_3 and $e_2 : \gamma_2 \to \gamma_3$ the dashed arrow into γ_3. Observe that for the cospan $\gamma_2 \hookrightarrow \gamma_3 \hookleftarrow \gamma_2$ we have $h_1 : e \hookrightarrow \mathrm{R}(\gamma_2)$ and $h_2 : e \hookrightarrow \mathrm{R}(\gamma_2)$ such that $\mathrm{R}(e_1) \circ h_1 = \mathrm{R}(e_2) \circ h_2$ but there is no rule γ' to ensure the incremental condition. However, the functor $\overline{T}$ associated to this rule system maps any graph to a thin version of it where parallel edges are replaced by a single edge. Given any $i : g \hookrightarrow g'$, $\overline{T}(i) : \overline{T}(g) \to \overline{T}(g')$ is a monomorphism. □

4.2 Categorical Constructions Computationally

Up to now, we exposed everything formally using categorical concepts. Let us describe their computational counterparts. For this, we consider *finite* global transformations and *finite* graphs.

First of all, the category **GraphI** is the formal abstraction of any library for manipulating graphs. Such a library has to provide data structures for representing (finite) graphs and monomorphisms between them, respectively the

objects and morphisms of **GraphI**. It also needs to come with a function taking as input two finite graphs g and g' and returning the set of monomorphisms $\mathrm{Hom}_{\mathbf{GraphI}}(g, g')$. This function corresponds in fact to a pattern matching algorithm, as usual in categorical accounts of graph rewriting. A function $- \circ -$ also needs to be provided to compute composition of monomorphisms, together with a function $- = -$ testing equality of monomorphisms. Finally, an extension function can be optionally provided. Such a function takes two monomorphisms $m : g' \to g$ and $e : g' \to g''$ and returns the set of all extensions of m along e, *i.e.* all monomorphisms $m' \in \mathrm{Hom}_{\mathbf{GraphI}}(g'', g)$ such that $m' \circ e = m$. A default implementation is readily possible, but a more efficient one is typically available.

Second, a finite rule system is described as a finite graph whose vertices are pairs $\langle l, r\rangle \in \mathbf{GraphI} \times \mathbf{GraphI}$ and edges are pairs of monomorphisms $\langle m : l \to l', i : r \to r'\rangle$. The l.h.s. and r.h.s. functors L and R return the first and second components of these pairs respectively. At the semantic level, $\mathbf{\Gamma}$ is the category generated from this graph. For example, Fig. 1 actually describes the graph that is used to generate the complete category corresponding to Definition 1. Although our implementation uses this graph directly without actually generating the category $\mathbf{\Gamma}$, we simplify the following discussion by referring to the generated category only.

The algorithm being *online*, the comma category of step (1) is never entirely represented in memory, neither is the cocone associated to the resulting colimit of step (3). Their descriptions are part of the algorithm.

4.3 A High-level Description of the Algorithm

Let us consider once and for all a finite global transformation $T = \langle \mathbf{\Gamma}, \mathrm{L}, \mathrm{R}\rangle$ and a finite graph g. We write D for $\mathrm{D}_T(g)$ for simplicity. As already mentioned, the comma category L/g represents the collection of all instances of any l.h.s. in g and their relations. It is discovered in a specific way for efficiency, by moving from neighbors to neighbors and by exploiting two important asymmetries.

The first one is that, whenever we have a morphism $\langle e, f'\rangle : \langle \gamma, f\rangle \to \langle \gamma', f'\rangle \in \mathrm{L}/g$ between two instances, the r.h.s. of γ' contains the r.h.s. of γ through $e : \gamma \to \gamma'$. This means that $\langle \gamma, f\rangle$ does not contribute more data to the output and it is enough to amalgamate only the r.h.s. of those $\langle \gamma', f'\rangle$ that are "maximal" in L/g. A non-maximal $\langle \gamma, f\rangle$ has a relevant role only when there is a second morphism $\langle \gamma, f\rangle \to \langle \gamma'', f''\rangle$ to another maximal instance $\langle \gamma'', f''\rangle$. In this case, it specifies how the r.h.s. of γ'' should be aligned with the r.h.s. of γ' in the resulting graph.

The second asymmetry concerns the neighbors of a given instance $\langle \gamma' \in \mathit{\Gamma}, f' : \mathrm{L}(\gamma') \to g\rangle$. Indeed, it is really efficient to compute all "incoming neighbors" or sub-instances $\bigcup_n \mathrm{Hom}_{\mathrm{L}/g}(n, \langle \gamma', f'\rangle)$ specified as

$$\{\langle e, f'\rangle : \langle \gamma, f' \circ \mathrm{L}(e)\rangle \to \langle \gamma', f'\rangle \mid e : \gamma \to \gamma'\}.$$

This corresponds simply to the composition in **GraphI** discussed earlier. On the contrary, "outgoing neighbors" or super-instances $\bigcup_m \mathrm{Hom}_{\mathrm{L}/g}(\langle \gamma', f'\rangle, m)$

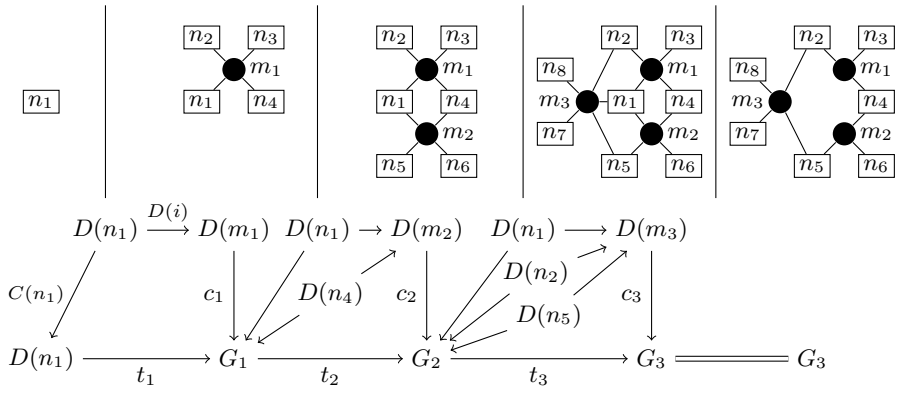


Fig. 3: Evolution of the data during the four firsts steps of the algorithm. From left to right: we start with a non-maximal instance, process its associated maximal instances successively, and finally drop the non-maximal. At each stage, the output is updated by generalized pushout.

correspond to extensions (so pattern matching) and are more expensive:

$$\{\langle e', f''\rangle : \langle \gamma', f'\rangle \to \langle \gamma'', f''\rangle \mid \\ e' : \gamma' \to \gamma'', f'' \in \mathrm{Hom}_{\mathbf{GraphI}}(\mathrm{L}(\gamma''), g) \text{ s.t. } f' = f'' \circ \mathrm{L}(e')\}.$$

Notice that these two specifications are obtained by simply unfolding the definition of the morphisms of the comma category. Also, the set of incoming morphisms $e : \gamma \to \gamma'$ and outgoing morphisms $e' : \gamma' \to \gamma''$ in $\mathbf{\Gamma}$ are directly available in the description of T as an abstract graph. All in all, L/g is thought as an abstract undirected bipartite graph that we call *the network*.

Fig. 3 illustrates the first steps of the algorithm representing maximal instances as black dots, and non-maximal instances as white squares. The initialization step is to find a first instance. For efficiency, we therefore try each minimal pattern, and start with the first founded minimal instance $n_1 \in \mathbf{\Gamma}$. At this point, the first result G_0 is simply the r.h.s. $D(n_1)$ and we memorize the (identity) relationship between $D(n_1)$ and G_0 and call it $C(n_1) : D(n_1) \to G_0$. Then we consider all maximal super-instances of n_1. In Fig. 3, we consider that there are three such super-instances m_1, m_2 and m_3 with associated morphisms $n_1 \to m_1$, $n_1 \to m_2$ and $n_1 \to m_3$, and they are processed one after the after.

The first iteration is the processing of m_1 which consists in taking all its sub-instances $n_1, \ldots, n_4$, noticing that n_1 was already computed, and therefore serves as a link with the current result G_0. The pushout of G_0 and the r.h.s. $D(m_1)$ of m_1 through the morphisms $C(n_1) : D(n_1) \to G_0$ and $D(i) : D(n_1) \to D(m_1)$ is therefore computed and gives the new result G_1, where i is the morphism from n_1 to m_1. Since G_1 now includes the r.h.s. of all non-maximal $N = \{n_1, \ldots, n_4\}$ discovered, we memorize as $C(n) : D(n) \to G_1$ for $n \in N$ the location of these r.h.s. in G_1.

The second iteration processes m_2 similarly and all its sub-instances n_1, n_4, n_5, and n_6 are computed. This times, this is n_1 and n_4 that were already computed and that serves as links with the current result G_1. So a generalized pushout is computed to amalgamate $D(m_2)$ within G_1. We say *generalized* because there are two spans $\langle n_1, C(n_1) : D(n_1) \to G_1, D(j) : D(n_1) \to D(m_2)\rangle$ and $\langle n_4, C(n_4) : D(n_4) \to G_1, D(k) : D(n_4) \to D(m_2)\rangle$, one for each for each already discovered non-maximal n_1 and n_4. We thus obtain G_2 and update the set N of discovered non-maximals and the locations C of their r.h.s. in G_2. Note that it is important to take the two spans into account simultaneously. Otherwise, some non-monomorphism would enter the game, preventing some optimization explained at the end.

The processing of m_3 is similar and shows no novelty. At this point non-maximal n_1 does not have any further role to play and is dropped together with all data associated to it, as shown in the the last step of Fig.3. Indeed, it has linked all its associated maximal together in the result.

During these processings, other non-maximal instances see some of their associated maximals being processed. We have to keep track of this to avoid double processing of maximals which would cause infinite loops. Also, non-maximals are processed in the order of first discovery, so the next one is n_2 in the example. With these precautions, the algorithm proceeds by treating the maximals of n_1, which are at distance 1 from n_1 in the network. Then each new non-maximals, at distance 2 from n_1, launches the processing of their new associated maximals, at distance 3, and so on, until the complete connected component of the network is processed. In memory, there are never stored more than four "radius" of instances d, $d+1$, $d+2$ and $d+3$ from n_1.

4.4 The Global Transformation Algorithm

Algorithm 1 gives a complete description of the previous procedure. The algorithm manages four variables G, N, E and C. Variable G contains the output graph. The part of the comma category that is kept in memory is represented by variables N and E: N is a queue containing, in order of discovery, the non-maximal instances that might still have a role to play and E associates each instance in N to the set of their maximal super-instances that have already been processed. For simplicity, E is not represented as a function from N to a sets but as a relation. The r.h.s. $D(n)$ of each instance $n \in N$ is already in the current result G through the morphism kept as $C(n)$.

Lines 1 to 5 corresponds to the initialization step, where a first instance n is found, its associated r.h.s. $D(n)$ is taken as initial result G, and E and C are initialized accordingly. While there are some non-maximal instances to treat (line 6), take the first of them called n without removing it from N (line 7), and compute (lazily) all its super-instances (line 8). Now, for each super-instance m of n that is maximal and not already processed (line 9), compute all its sub-instances (line 10) and the spans (line 11) allowing to amalgamate its associated r.h.s. $D(m)$ with the current result G (line 12). This produces a new result G' and two morphisms $t : G \to G'$ and $r : D(m) \to G'$ exhibiting G' as the

Algorithm 1: Global Transformation Application

Data: $G : \mathbf{GraphI}$
Data: $N : (\mathrm{L}/g)^*$
Data: $E \subseteq \coprod_{n \in N} \coprod_m \mathrm{Hom}_{\mathrm{L}/g}(n, m)$
Data: $C : \prod_{n \in N} \mathrm{Hom}_{\mathbf{GraphI}}(D(n), G)$
1 **let** $n = \texttt{findAnyMinimal}(T, g)$ any minimal element in L/g
2 **let** $E = \emptyset$
3 **let** $C = \{n \mapsto id_{D(n)}\}$
4 **let** $N = n$
5 **let** $G = D(n)$
6 **while** $N \neq \epsilon$ **do**
7 **let** $n \cdot \_ = N$
8 **let** $M' = \coprod_m \mathrm{Hom}(n, m)$
9 **for** $(m, e) \in M'$ *s.t.* $(n, m, e) \notin E$ *and* t *is maximal* **do**
10 **let** $E' = \coprod_{n'} \mathrm{Hom}(n', m)$
11 **let** $S = \{\langle n', C(n'), D(e')\rangle \mid (n', e') \in E', n' \in N\}$
12 **let** $(G', t, c) = \texttt{generalizedPushout}(G, D(m), S)$
13 $E := E \cup \{(n', m, e') \mid (n', e') \in E'\}$
14 $C := \{n' \mapsto t \circ C(n') \mid n' \in N\}$
15 $C := C \cup \{n' \mapsto c \circ D(e') \mid (n', e') \in E', n' \notin N\}$
16 $N := N \cdot \langle n' \mid (n', e') \in E', n' \notin N\rangle$
17 $G := G'$
18 $E := \{(n', m, e') \in E \mid n' \neq n\}$
19 $C := \{n' \mapsto C(n') \in C \mid n' \neq n\}$
20 $\_ \cdot N := N$
21 **return** G

generalized pushout, *i.e.* the colimit of all the spans in S simultaneously. At line 13, we keep track of which already discovered non-maximal the maximal m is associated to, to prevent double-processing and infinite-loops as explained. At line 14, we update for each previous non-maximal instance $n' \in N$ its morphisms $C(n') : D(n') \to G$ to G into a morphism $t \circ C(n')$ to G'. At line 15, we bookkeep for each new non-maximal $n' \notin N$ the morphism $r \circ D(e)$ indicating where its associated r.h.s. $D(n')$ is to be found in G. Finally, we add all new non-maximal instances to the queue N and switch to the new result at line 16 and 17. At this point, the invariant is restored and we loop to the next maximal instance. Once all maximal instances of n processed, we drop references to it at line 18, 19 and 20 and loop to the next non-maximal.

4.5 Correction of the Algorithm

This section is the proof of the correction of Algorithm 1 that we consider without any memory management. We do as if the lines 18 and 19 were removed and some alternative mechanism controlled the while loop.

Theorem 2. *Given any* finite *rule system T, the algorithm computes $\overline{T}(g)$ for all* finite *graphs g.*

First, let us show that the different components of the previous coarse reasoning indeed correspond to the formal specification given in the Equation 1. First of all, thinking in terms of maximal, non-maximal and minimal instances, sub- and super-instances actually makes sense for any comma category L/g.

Proposition 4. *For any category* $\mathbf{I}$*, any functor* $F : \mathbf{I} \to \mathbf{GraphI}$*, and any graph* $g \in \mathbf{GraphI}$*, the comma category* F/g *is* thin*, i.e. there is at most one morphism between any two objects of* F/g*.*

A thin category is isomorphic to a preordered set, but in our implementation, any time an element is generated, all of its isomorphic elements are taken care of at the same time. This corresponds informally to taking the poset of equivalence classes of the preordered set.

Second, note that the *network* considered in the algorithm does not contain the morphisms of L/g that are between non-maximal instances. The actual implementation does use them, but only to discard useless redundancies in S at line 11. The correction of this approach is given by the following proposition.

Proposition 5. *The subcategory of* L/g *given by all instances but only morphisms to maximal instances is final in* L/g*, in the sense of final functor.*

Proof. We need to show that for any instance $o \in \mathrm{L}/g$, and any two morphisms $e_0 : o \to o_0$ and $e_1 : o \to o_1$, there is a zig-zag z in the subcategory and a sequence of morphisms of L/g from o to each z_k that commutes with each $\overline{z}_k$. Take $i \in \{0, 1\}$. If o_i is maximal we set $\overline{z}_i = e_i$. If it is non-maximal, we set $\overline{z}_i = e'_i \circ e_i$ for any $e'_i : o_i \to o'_i$ to some maximal o'_i. We have just built a valid z of length 2 in the subcategory, the associated sequence of morphism being simply $\overline{z}_0, id_o, \overline{z}_1$, which trivially commutes as wanted. □

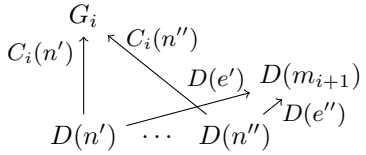
This means that any colimit computed on a functor from L/g is unchanged if the functor is restricted to the network subcategory.

To continue, let us describe the evolution of the data, and start by using E_0, C_0, n_0 and G_0 to denote the values taken by the variable E, C, n and G at lines 2, 3, 4 and 5 respectively. Let us also write $N_0 = \{n_0\}$, and $M_0 = \emptyset$. These data describe two things: the initial sub-network $\overline{E_0}$ with a single non-maximal $N_0 = \{n_0\}$, no maximal $M_0 = \emptyset$, and no morphisms $E_0 = \emptyset$; and the initial cocone K_0 of $D \restriction \overline{E_0}$ having G_0 for apex and components $K_{0,n_0} = C_0(n_0) = id_{D(n_0)}$.

This initial situation is then updated at the discovery of each new maximal. Let us use $\langle m_i, E'_i, S_i, G'_i, t_i, c_i, E_i, C_i, N_i, G_i \rangle$ starting with $i = 1$ to denote the successive values taken by the respective variables at lines 9 to 17. Let us add $M_i = M_{i-1} \cup \{m_i\}$ to denote the set of discovered maximals. These data describe the successive sub-networks $\overline{E_i}$ having for set of objects the non-maximals N_i and the maximals M_i, and for set of morphisms E_i. The described cocone K_i over $D \restriction \overline{E_i}$ have G_i for apex and components $K_{i,n} = C_i(n)$ for $n \in N_i$, $K_{i,m_j} = t_i \circ K_{i-1,m_j}$ for all $j < i$ and $K_{i,m_i} = c_i$. Lines 14 and 15 gives us that $K_{i,n} = t_i \circ C_{i-1}(n) = t_i \circ K_{i-1,n}$ for $n \in N_{i-1}$ and $K_{i,n} = c_i \circ D_{i-1}(e)$ for all $a \in N_i \backslash N_{i-1}$.

In this setting, what are left to be proved are, first, that the successive sub-network $\overline{E_i}$, as described by the evolution of N_i and M_i and E_i, grows until the complete network is covered, and, second, that the cocone K_i, as described by the evolution of G_i, C_i and t_i and c_i, maintain the property of being the universal cocone of $D \restriction \overline{E_i}$. For the growing of $\overline{E_i}$, a look at lines 1, 2, 4, 6, 7, 8, 9, 10, 13, and 16, 20, should be enough to convince the reader that we progress from each radius to the next as wanted, until the complete connected component is covered. In fact, $\overline{E_i}$ consists of all maximals $M_i = \{m_j \mid j \leq i\}$ and all morphisms to them, a new maximal being added at each iteration.

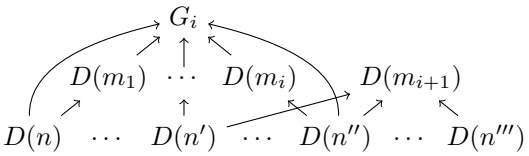
Now, we proceed inductively to show that K_i is indeed the universal cocone of $D \restriction \overline{E_i}$ for all i. This is clearly true for $i = 0$, so we jump to the induction step and suppose that some i. Looking at lines 10, 11, 12 and 17, we see that G_{i+1} is the generalized pushout of the spans collected in S_{i+1}, *i.e.* $G_{i+1} = \mathrm{Colim}(\overline{S_{i+1}})$, where $\overline{S_{i+1}}$ refers to the following diagram



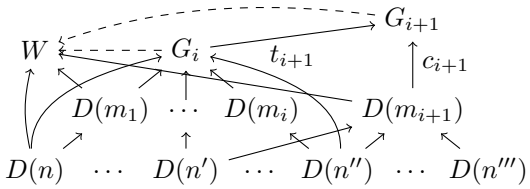
where $\{n', \ldots, n''\}$ are sub-instances of m_{i+1} that are also sub-instances of previous $m \in M_i$. This can also be written $\{n', \ldots, n''\} = N_i \cap N_{i+1}$.

Proposition 6. *For all $i \geq 1$, if $G_i = \mathrm{Colim}(D \restriction \overline{E_i})$ with K_i as universal cocone, then $G_{i+1} := \mathrm{Colim}(\overline{S_{i+1}})$ is such that $G_{i+1} = \mathrm{Colim}(D \restriction \overline{E_{i+1}})$ with K_{i+1} as universal cocone.*

Proof. Let us consider a bigger diagram containing the diagrams $D \restriction \overline{E_i}$, $\overline{S_{i+1}}$ and $D \restriction \overline{E_{i+1}}$. For this, we take $D \restriction \overline{E_{i+1}}$ and add the object G_i and all morphisms $K_i(o) : D(o) \to G_i$ for $o \in N_i \cup M_i$. But universality of G_i, there is a bijection between cocones on this big diagram and cocones on $D \restriction \overline{E_{i+1}}$.



Adding to the picture $G_{i+1} := \mathrm{Colim}(\overline{S_{i+1}})$ with the associated morphisms t_{i+1} and c_{i+1}, recall that $K_{i+1,o} = t_{i+1} \circ K_{i,o}$ for all $o \in N_i \cup M_i$, and $K_{i+1,o} = c_{i+1} \circ D(e)$ for all $e : o \to m_{i+1}$. This is indeed a cocone on $D \restriction \overline{E_{i+1}}$ by properties of t_{i+1} and c_{i+1}, and extending it with $K_{i+1,\mathtt{G_i}} = t_{i+1}$ gives us the unique corresponding cocone on the big diagram. To establish that it is universal, let us consider another arbitrary cocone W on $D \restriction \overline{E_{i+1}}$.



By diagram chasing, one can see that this cocone can be restricted to its components on $\overline{E_i}$, *i.e.* on $\{n, \ldots, n'', m_1, \ldots, m_i\}$. This gives us a unique extension of W on all the big diagram. By restricting again to its components on S, *i.e.* components at $n', \ldots, n'', G_i, m_{i+1}$, and by universality of G_{i+1}, we obtain a unique mediating from G_{i+1} to W, as wanted. □

As a last remark, note that the splitting of colimit as specified by the algorithm ensures that all t_i and c_i are monomorphisms.

Proposition 7. *Given that T is incremental, we have that for each $i \geq 1$, $t_i : G_{i-1} \to G_i$ and $c_i : D(m_i) \to G_i$ are monomorphisms.*

Proof. The proof is similar to the proof of Theorem 1. □

In the implementation of the algorithm, this fact is used, and all modifications are realized *in place*. In other words, everything is implemented to ensures that t and c always act as identity functions. This means that neither line 14 of the algorithm, nor the composition in line 15 are actually implemented.

5 Conclusion

In this paper, we have presented an algorithm for computing the application of global transformations on graphs. Our primary goal was to show the feasibility of implementing global transformations. Indeed parallel graphs rewriting usually rises the issue of rule overlaps and conflict management (for example with restricting conditions to get only conflict-free rule systems [2] or by merging alternatives [4,3]). This issue does not arise in our approach which formally enforces the rules to mutually agree in case of overlap [10]. The cost of this strategy is a multiplication of rules causing an over load of work during computation [5]. As a comparison, the Sierpinski rule system of Fig. 1 is the data of 8 rules where a unique one is enough for alternative approaches [11]. The focus on maximal instances in the proposed algorithm clearly shows that only maximal instances matters during computation, sub-instances being used to guide the exploration of the match space. Moreover, the extra cost of memory requirement for storing instances is strongly reduced thanks to the comma category discovery strategy.

The proposed algorithm has been designed to be generic. Indeed in Algorithm 1 no graph-specific operations are referred, so it can also be used with some underlying category $\mathbf{C}$ once a suitable $\texttt{generalizedPushout}_{\mathbf{C}}$ operation is provided. As an example, the algorithm is perfectly functional for the category of words given in [5]. We are currently investigating the necessary properties

on $\mathbf{C}$ and $\texttt{generalizedPushout}_{\mathbf{C}}$ for the algorithm to be correct. The study of the specificity of on the case $\mathbf{C} = \mathbf{GraphI}$ and comparison with other tools for graph rewriting (like GPaR, Two Tapes, GrGen.NET, XL, etc.) have been delayed to future work. A part of the effort is also devoted to the extensions of the algorithm to account for parametrized (*i.e.*, dealing with labels) and non-deterministic variants of global transformations [8].

By essence parallel rewriting systems implement the principle of *space homogeneity*: all rules apply everywhere. Rule inclusions obey to the same principle: the r.h.s. of a super-rule has to contain the r.h.s. of its sub-rules. Conversely, a sub-rule factorizes the common behavior of its super-rules. The incremental criterion goes further by stating the independence of rules: given a super-rule, its r.h.s. contains the r.h.s. of its sub-rules as if they were considered independently. This alternative expression has been proved equivalent to Definition 3 [7]. The criterion prevents from intuitively non-local behavior like collapsing non-empty graphs to a single vertex. From that point of view, incremental global transformations follow the research direction of causal graph dynamics [1]. In this work any produced matter (vertex or edge) in the output is attached to an element of the input graph and a particular attention is put on the fact that two rule instances cannot produced a common fresh piece of matter. Future work also concerns understanding the notion of locality and causality in global transformations and to relate these notions to a kind of topological continuity of the transformation.

References

1. Pablo Arrighi, Simon Martiel, and Vincent Nesme. Cellular automata over generalized cayley graphs. *Math. Struct. in Comp. Sc.*, 18:340–383, 2018.
2. Thierry Boy de la Tour and Rachid Echahed. A set-theoretic framework for parallel graph rewriting. *arXiv preprint arXiv:1808.03161*, 2018.
3. Stéphane Despréaux and Aude Maignan. Gpar: A parallel graph rewriting tool. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 53–60. IEEE, 2018.
4. Rachid Echahed and Aude Maignan. Parallel graph rewriting with overlapping rules. *arXiv preprint arXiv:1701.06790*, 2017.
5. Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Lindenmayer systems and global transformations. In *International Conference on Unconventional Computation and Natural Computation*, pages 65–78. Springer, 2019.
6. Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Cellular Automata and Kan Extensions. Research report, Univ Paris Est Creteil, LACL, 94000, Creteil, France, February 2021. URL: `https://hal.archives-ouvertes.fr/hal-03149398`.
7. Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. Incremental Global Transformation of Graphs. Research report (to be published), Univ Paris Est Creteil, LACL, 94000, Creteil, France, 2021.
8. Alexandre Fernandez, Luidnel Maignan, and Antoine Spicher. The Bicategory of Open Functors. Research report, Univ Paris Est Creteil, LACL, 94000, Creteil, France, February 2021. URL: `https://hal.archives-ouvertes.fr/hal-03139482`.

9. Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.
10. Luidnel Maignan and Antoine Spicher. Global graph transformations. In *GCM@ ICGT*, pages 34–49, 2015.
11. Gabriele Taentzer, Enrico Biermann, Dénes Bisztray, Bernd Bohnet, Iovka Boneva, Artur Boronat, Leif Geiger, Rubino Geiß, Ákos Horvath, Ole Kniemeyer, et al. Generation of sierpinski triangles: A case study for graph transformation tools. In *International Symposium on Applications of Graph Transformations with Industrial Relevance*, pages 514–539. Springer, 2007.