Topological Consistency Preservation with Graph Transformation Schemes

Romain Pascual[a,*], Pascale Le Gall[a], Agnès Arnould[b], Hakim Belhaouari[b]

[a]*Laboratory Mathematics in Interaction with Computer Science (MICS), CentraleSupélec, Université Paris Saclay, France*

[b]*Laboratory XLIM UMR CNRS 7252, University of Poitiers, France*

Abstract

Topology-based geometric modeling tackles the issue of representing objects with data structures that encode the topological subdivision of modeled objects in vertices, edges, faces, and volumes. Such subdivisions can be represented with graphs labeled by dimensions on arcs, while modeling operations used to edit the objects can be formalized as graph transformations.

Among the existing topological models, we consider generalized and oriented maps, defined as constrained labeled graphs, to ensure the well-formedness of the represented objects. Since a modeling operation should provide a correct object when applied to a correct object, graph transformations are provided with conditions to ensure the model consistency.

Our approach exploits the firmly established framework of DPO graph transformations to implement modeling operations. We enrich standard DPO graph transformations with a product construction to ease the operation design, enabling generic modeling operations as rule schemes. We lift conditions from DPO rules to this enriched framework, ensuring the preservation of the topological consistency via static analysis of syntactic conditions on rule schemes.

Keywords: DPO graph transformation, Rule schemes, Combinatorial maps, Consistency preservation, Topology-based geometric modeling.

---

1. Introduction

Graphs are widely used for modeling issues, for instance, as data structures for geometric modeling. Geometric modeling allows for the creation and manipulation of n-dimensional objects. When geometric objects are represented with graphs, their modifications can be formalized with graph transformations. This paper focuses on the double-pushout approach to graph transformations [1] to specify topology-based geometric modeling operations.

Geometric modeling encompasses computer-aided design and manufacturing with applications across mechanical engineering, animated movies, video games, geology, and architecture. Numerous methods for the representation of geometric objects use a subdivided representation of objects. Objects can be divided according to application-domain properties, e.g., a segmentation based on a field value, or topological properties, i.e., a subdivision of a polyhedron into faces, edges, and vertices. Optimized data structures can be conceived depending on the subdivision structural properties (e.g., all faces are triangles) and the domain-related computations. For instance, triangle soup is a commonly used model for rendering virtual views because graphic processors are optimized to render triangular primitives [2, Chapter 12]. In animation, articulated objects are represented using a hierarchical structure of the object parts (called skeleton or rig) and a surface (typically a mesh) associated with each bone of the skeleton. To animate an articulated object, one can animate the skeleton and render the surface mesh on a set of poses [3].

[*]Corresponding author (romain.pascual@centralesupelec.fr)

In topology-based geometric modeling, information is added to a polygonal representation of objects to encode the topological relations between object sub-parts. These topological relations describe adjacency and incidence between the topological cells (vertices, edges, faces, and volumes). Constraints on the topological relations ensure the soundness of the object's geometry, e.g., angles are correctly formed, vertices are incident to edges that are, in turn, incident to faces. Since modifications of a well-formed object should produce an equally well-formed object, graph transformations should preserve these consistency properties. In graph transformations, property preservation is a commonly studied problem; here, we study it for graphs related to a specific application domain, geometric modeling.

Polygon soups and meshes of animated rigs emphasize the external surface to ease rendering. In topological-based geometric modeling, we are interested in both the outer and the complete internal structure of objects. Indeed, inner parts play a key role in applications where volumetric properties are essential: geology [4], architectural buildings [5], or physical simulation [6, 7]. Edge-based data structure [8] is undoubtedly the most widely used model in topology-based geometric modeling, exploited in the open-source modeler Blender [9]. These models are equivalent to the more formal *n*-dimensional maps [10]. In the literature, *n*-dimensional maps are called combinatorial maps. In this article, we follow the terminology given in [11] and call them oriented maps. We will use the expression 'combinatorial maps' as a generic term that encapsulates several models (such as oriented maps, generalized maps, hypermaps, chains of maps).

Combinatorial maps are often considered, from an algebraic perspective, as a collection of permutations on a set of atomic elements. The study of combinatorial maps comes from the field of combinatorics to represent cell decompositions of a surface [12]. These maps are also used to draw graphs on a given surface, exploiting the combinatorial properties of the underlying graph, e.g., Euler's formula, and topological maps have been defined as graphs embedded in a surface [13]. Among combinatorial map models, generalized maps and oriented maps are the most popular [14].

In a graph-based fashion, both generalized maps and oriented maps are encoded as graphs whose arcs are labeled on an alphabet of dimensions that describes topological relations between the object cells. Previous works [15] have already represented generalized maps as graphs and formalized geometric operations as DPO rewriting. DPO rules were extended to meta-rules [16] in a generator of geometric modelers called Jerboa [17]. Traditionally, geometric modeling operations are defined generically to modify cells regardless of their size (e.g., number of faces constituting a volume). Meta-rules abstract topological cells thanks to variables. These variables can be instantiated with a particular cell to give rise to classical DPO graph transformations. In [16], sufficient conditions for consistency preservation of generalized maps were given for DPO transformations and meta-rules. Thus, the instantiation of a meta-rule with a given cell provides a graph transformation preserving the object consistency.

These pioneering works have demonstrated that graph transformations can be used advantageously to design topology-based geometric modelers in the context of generalized maps. This well-established model is highly regular, allowing for reasoning on the manipulation of subdivided objects. For instance, formal proofs with the Coq system have been studied in [18] with the help of generalized maps. Compared to generalized maps, oriented maps present the main advantage of a compact representation of objects to the detriment of regularity in dimension, making reasoning more difficult. Oriented maps are supported by efficient implementations such as the CGoGN library [19] and are usually favored for industrial applications.

In this paper, we explore both generalized maps and oriented maps to promote graph transformations in the geometric modeling community. Compared with our previous work [15, 16], we no longer use variables and fully formulate rule schemes in terms of graph theory. We also study the constraint on the topological relations in a broader way so that the rules are no longer tuned for the generalized map model. To offer a unified framework, we have identified three topological constraints which, combined in different ways, define both generalized maps and oriented maps. As a consequence, the graphs under study are graphs subject to simple constraints.

Since DPO rewriting describes local modifications, we consider local definitions of the topological constraints, namely at the scale of nodes and arcs. Moreover, we split the constraints to study their preservation independently. Nonetheless, as a generic expression of topological operations is still required, we enrich the standard DPO approach with a pullback construction. More precisely, rule schemes depict transformations duplicated for each node of a graph, called pattern graph. Rule schemes and their instantiation with pattern graphs via the categorical product provide a framework to design generic operations. From a graph transformation perspective, we turn local DPO-based transformations into global product-based transformations.

The main contribution of this paper is the first framework based on graph transformations to express classical topological operations on both generalized and oriented maps. This framework results from the following realizations:

- Generalized maps and oriented maps are defined via three topological constraints: the incident arcs constraint, the non-orientation constraint, and the cycle constraint.
- Preservation of topological constraints in the context of DPO graph transformations is ensured by necessary and sufficient conditions.
- DPO graph transformations are generalized with rule schemes. Modifications expressed by rule schemes are essentially substitutions of successive arcs selected by their labeling. Rule scheme instantiation is defined using a product with a pattern graph extracted from the graph to be modified.
- Instantiation of rule schemes is guaranteed to result in DPO rules satisfying conditions for preserving topological constraints. This guarantee is obtained by sufficient conditions.

The article is organized as follows. The theoretical background of labeled graphs and their transformations is presented in Sections 2. We recall the DPO approach to graph transformations as a mechanism for local operations and show how the categorical product can express global operations. We provide contextualization elements in Section 3, introducing formal issues arising from the use of graph transformations in geometric modeling. In particular, we give graph-based definitions of generalized and oriented maps. We develop, in Section 4, the condition for preserving the incident arcs constraint in the DPO framework. The discussion is broadened to encompass the other two constraints (non-orientation and cycles) in Section 5. The results presented in Sections 4 and 5 ensure consistent rewriting of G-maps and O-maps with DPO transformations. In Section 6, we introduce rule schemes to facilitate the design of generic topological operations, as well as the instantiation mechanism using a product with a pattern graph extracted from the combinatorial graph under modification. The preservation of each of the topological constraints is successively studied in the three following sections (Sections 7, 8, and 9) by referring to the conditions of preservation on the DPO graph transformations resulting from their instantiation on a combinatorial graph. Application to topological-based geometric modeling is the subject of Section 10 with illustration on generalized and oriented maps. We also provide algorithms for the static verification of O-map and G-map consistency. In Section 11, we compare our approach to other constructions that preserve properties in graph transformations. Finally, concluding remarks are given in Section 12.

2. Graph transformations in the category of labeled graphs

In this section, we recall the main constructions of the category of edge-labeled graphs and the semantics of DPO rewriting. We show how the categorical product can generalize DPO rewriting.

2.1. Category of labeled graphs

We assume that the main notions from the category **Graph** are known. We advise the reading of [20] for an introduction to graph transformations and [1] for the complete construction of this category, with the corresponding constructions of pullbacks, pushouts, and pushout complements.

A graph is a collection of nodes and arcs that link two nodes, and information is added using labels. Such graphs are called *labeled graphs*. For our needs, we only consider labels on arcs and use totally labeled graphs.

Definition 1 (Labeled Graphs and Labeled Graph Morphisms)**.** *Let Σ be a finite set of labels. A graph labeled on Σ, or* Σ-graph*, $G = (V_G, E_G, s_G, t_G, l_G)$ is defined by two finite sets V_G of* nodes *and E_G of* arcs*,* source *and* target *functions $s_G, t_G : E_G \to V_G$ and a* label *function $l_G : E_G \to \Sigma$.*

A Σ-graph morphism *$m : G \to H$ consists of two functions $m_V : V_G \to V_H$ and $m_E : E_G \to E_H$ that preserve sources, targets and labels, i.e., such that $s_H \circ m_E = m_v \circ s_G$, $t_H \circ m_E = m_v \circ t_G$, and $l_H \circ m_E = l_G$.*

A Σ-graph morphism is a mono *if both m_V and m_E are injective and it is an* inclusion *if $m_V(x) = x$ for all $x \in V_G$, and $m_E(x) = x$ for all $x \in E_G$. If m is a mono, it is written $m : G \hookrightarrow H$. The category having Σ-labeled graphs as objects and Σ-graph morphisms as arrows is written Σ-**Graph**.*

We will use the following notation simplifications throughout the paper. When there is no ambiguity on the graph, the index G is omitted and G is denoted by $G = (V, E, s, t, l)$. We identify arcs in the graph using their labels: an arc e labeled by i in Σ is an *i-arc*. We identify a morphism $m = (m_V, m_E) : G \to H$ with its component-wise functions and omit the subscripts V and E: we write $m(x)$ regardless of whether x is a node or an arc of G.

The category Σ-**Graph** is an elementary topos [21] meaning it behaves essentially like the category of sets and functions. Weaker properties, such as adhesivity [22] and the existence of initial and terminal graphs are enough for our needs.

The terminal element $\mathbf{1}_\Sigma$ of the category Σ-**Graph** is the graph with a single node and an i-loop for every possible label i in Σ. The universal property of terminal elements ensures that given a Σ-graph P the morphism $!_P : P \to \mathbf{1}_\Sigma$ is uniquely defined. The morphism sends all nodes of P onto the single node of $\mathbf{1}_\Sigma$. Similarly, it sends all arcs of P onto the loop of $\mathbf{1}_\Sigma$ with the same label.

Because Σ-**Graph** is an adhesive category, it admits pullbacks and pushouts along monomorphisms. In particular, pullbacks on the terminal element are uniquely defined (by the universal property of terminal elements) and yield products. For any two Σ-graphs P and Π, the pullback of the cospan $P \xrightarrow{!_P} 1_\Sigma \xleftarrow{!_\Pi} \Pi$ is the product of P and Π in the category Σ-**Graph**:

$$\begin{tikzcd} P \times_\Sigma \Pi \arrow[r] \arrow[d] & \Pi \arrow[d, "!_\Pi"] \\ P \arrow[r, "!_P"] & \mathbf{1}_\Sigma \end{tikzcd}$$

In the following, Σ will be viewed as an alphabet, i.e., as a set of atomic elements called *letters*. A *word* on Σ is a finite sequence of letters from Σ. The *empty* word ε is the word with no letter. The set of words on Σ is denoted by Σ^*. A word w on an alphabet Σ is of length n in $\mathbb{N}$, denoted by $|w| = n$, if it is a sequence of n letters. The k-th ($0 < k \leq |w|$) letter of w is denoted by $w_{(k)}$. The concatenation of two words u and v from Σ^* is the word uv of length $|u| + |v|$ such that $(uv)_{(k)} = u_{(k)}$ for all $0 < k \leq |u|$ and $(uv)_{(|u|+k)} = v_{(k)}$ for all $0 < k \leq |v|$.

Let $G = (V_G, E_G, s_G, t_G, l_G)$ be a Σ-graph. A *path* is a sequence $e_1 \dots e_n$ of arcs such that $t_G(e_k) = s_G(e_{k+1})$ for all $1 \leq k < n$. If $s_G(e_1) = t_G(e_n)$ then the path is a *cycle*. A path $e_1 \dots e_n$ is a *w-path* if e_k is a $w_{(k)}$-arc for every $1 \leq k \leq n$. If the path is a cycle, then it is a w-cycle. A w-path $p = e_1 \dots e_n$ can be denoted by $s(e_1) \overset{w}{\rightsquigarrow} t(e_n)$ when we are only interested in the source, target and label of the path. An arc $e \in E$ is said to be *incident* to the nodes $s(e)$ and $t(e)$. The arc e is *non-oriented* if it has a reverse arc $e' \in E$, i.e., there exists an arc e' such that $s(e') = t(e)$, $t(e') = s(e)$ and $l(e) = l(e')$. When the reverse arc is unique, we write $e' = e^{-1}$.

2.2. Graph transformations using double pushouts

In an adhesive category, DPO rewriting exploits spans of monomorphisms as rules to ensure determinism and reversibility of the transformations [22].

Definition 2 (Rule)**.** *A rule r is a span $L \hookleftarrow K \hookrightarrow R$ of monomorphisms.*

L, R, and K are respectively called the left-hand side, right-hand side, and interface (or kernel) of the rule. The application of a rule r to a graph G consists of three steps: match L within G, delete the matched elements that do not belong to K, and add elements of R not in K. Matching the left-hand side of a rule within a graph is subject to the gluing condition [22] that states the existence of a pushout complement.

Definition 3 (Match)**.** *Let $r = L \hookleftarrow K \hookrightarrow R$ be a rule. A* match *for r is a morphism $m : L \to G$. The match $m : L \to G$ satisfies the gluing condition for r if there exists a pushout complement of $K \to L \to G$.*

Monic matches do not lessen the expressiveness of the rewriting system [23] but allow for explicit counting when deleting nodes or edges. Such matches also reduce the gluing condition to the dangling condition [24]: no node of $m(L) \setminus m(K)$ is source or target of an arc in $G \setminus m(L)$. In the sequel, we will assume that the following assumption holds.

Assumption 1. Matches are monomorphisms.

Matching the left-hand side L of a rule in a graph G is the first step towards the transformation of G. Still, we need to explain how to disconnect and reconnect graphs.

Definition 4 (Direct Derivation)**.** *Let G and H be two graphs, $r = L \hookleftarrow K \hookrightarrow R$ and $m : L \hookrightarrow G$ a match for r. The rule r transforms G into H, if there is a diagram*

$$\begin{tikzcd}
L \arrow[d, hook, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, hook'] \arrow[r, hook] \arrow[d, hook] \arrow[dr, phantom, "(PO)"] & R \arrow[d, hook, "m'"] \\
G & D \arrow[l, hook'] \arrow[r, hook] & H
\end{tikzcd}$$

where both squares are pushouts. The direct derivation $G \Rightarrow^{r,m} H$ *exists and is unique (up to isomorphism) if m satisfies the gluing condition. The morphism $m' : R \hookrightarrow H$ is called the comatch of the derivation.*

Example 1 (Direct Derivation). Relabeling an i-arc into a j-arc can be achieved by the rule depicted in the top span of Figure 1. The rule has two nodes in its left-hand side, right-hand side, and interface, an arc with the label to be modified in L, the relabeled arc in R, and no arc in K. The rule application is obtained via the match sending the node a to the node v, the node b to the node x and the i-arc between a and b to the i-arc between v and x. The i-arc is relabeled by j.

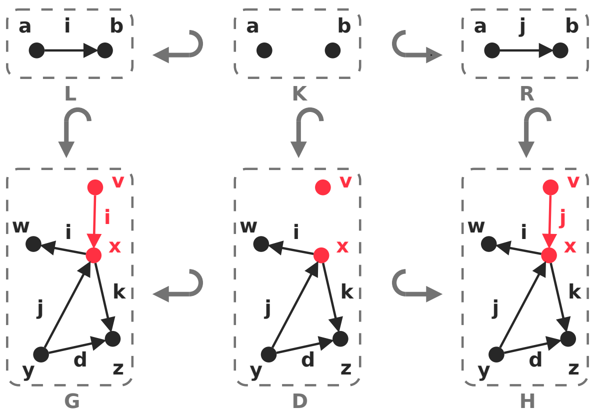


Figure 1: DPO transformation for relabeling.

We will study the preservation of constraints related to the graphs of geometric modeling. To simplify notations in the verification process, we use rules where the monomorphisms are inclusions. The interface K

contains all elements common to L and R. For such rules, we only specify L and R because K can be retrieved as the (component-wise set) intersection of L and R. From now on, we will consider rules satisfying the following hypothesis.

Assumption 2. The interface K of the rule is the (component-wise set) intersection $L \cap R$. Such rules are written $L \twoheadrightarrow R$, and their two monomorphisms are inclusions.

Graph transformations using DPO rewriting are helpful when trying to apply local transformations. Such transformations require isolating a part of the graph in order to modify it. This isolation forbids the application of the same modification uniformly to a subgraph. For instance, DPO rewriting does not provide a way to relabel all arcs of a given dimension for a cell without specifying its topology (e.g., the size of the cell). We aim at generalizing transformations to encode modeling operations regardless of the underlying topology. For this purpose, we need special operations such as relabeling or deleting all arcs sharing the same label. These operations can be designed using products.

2.3. Graph modifications using products

DPO rewriting models transformations based on deletion and creation of structure, allowing for a straightforward definition of graph modifications in a preservative framework. However, DPO rewriting is ill-suited should the initial graph elements be cloned or specifications on their incident arcs be made. On the other hand, Node-Labeled-Controlled grammars [25] inherently allow copying, merging, or deleting parts of the graph. Bauderon provided a categorical definition of this approach [26] to simultaneously apply various modifications at different spots in a graph with a single mathematical operation, defined by a pullback.

This paper will use the DPO rewriting to transform geometric objects (represented as graphs) and a pullback-based generalization to express cloning and transformations on a global scale. This construction offers a simple yet sufficient construction for our invariant-preserving framework.

Example 2 (Product Construction). Consider the transformation depicted in Figure 2 where $\Sigma = \{1,2\}$. In this example, Π has two nodes, x and y and three arcs. The 2-loop on x in Π and the 2-loop on a in P yield a 2-loop on (x,a) in $P \times_\Sigma \Pi$. Similarly, the 1-arc from y to x in Π and the 1-arc from b to c in P produce a 1-arc from (y,b) to (x,c) in $P \times_\Sigma \Pi$. Note that the 2-arc in P can not be matched with an arc of source y in Π. Thus, there is no 2-arc of source (y,a), (y,b) or (y,c) in $P \times_\Sigma \Pi$.

Arc deletion. We will consider the nodes of Π as different copies of the graph P. Thus, loops on the nodes of Π define the operation to be carried out on the arcs of P. We can use a product construction similar to Example 2 to delete all the i-arcs for a label $i \in \Sigma$. This deletion is achieved as the product of P with $\Pi = \Delta_i$, the graph having only one node and $|\Sigma| - 1$ loops labeled d for every d in $\Sigma \setminus \{i\}$. Such an operation is illustrated in Figure 3 where all the 2-arcs of P are deleted.

Arc relabeling. In the case of relabeling, P is the graph in which we want to relabel arcs, whereas Π stores the relabeling. Let us consider $\Sigma = \{i,\ i',\ j,\ j',\ k,\ d\}$ and assume we want to systematically relabel i into j, i' into j', keep k unchanged and delete d, meaning carry out these operations on all arcs of the corresponding labels. This global operation can be encoded with a relabeling function defined by the set of pairs $\{(i,j),(i',j'),(k,k)\}$. More generally, a relabeling operation is defined by a set of pairs $\mathfrak{R} = \{(i,j) \mid i \in \Sigma, j \in \Sigma\}$ such that for any i in Σ, there is at most one j in Σ satisfying $(i,j) \in \mathfrak{R}$. Such a set will be called a *relabeling set*. A letter i in Σ such that (i,i) is in $\mathfrak{R}$ is a label left unchanged by the relabeling. Conversely, a letter i in Σ such that for all j in Σ, (i,j) is not in $\mathfrak{R}$ is a label deleted by the relabeling. As $\mathfrak{R}$ is a subset of Σ^2, the product is expressed in the category Σ^2-**Graph**. The Σ-graph P to be renamed must be converted into a labeled graph with labels in Σ^2. We will consider all the possible renamings, i.e., for each arc i, we will replace it with as many arcs as possible labeled with pairs of the form (i,j) with j in Σ. Moving back and forth between the two categories is achieved with

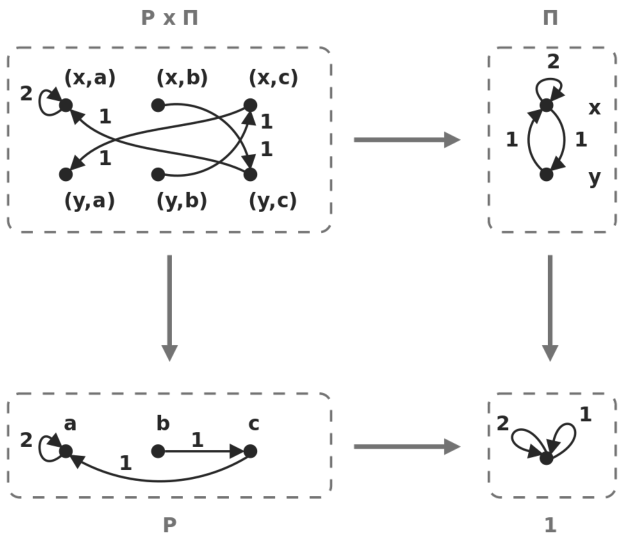


Figure 2: Pullback using a terminal graph.

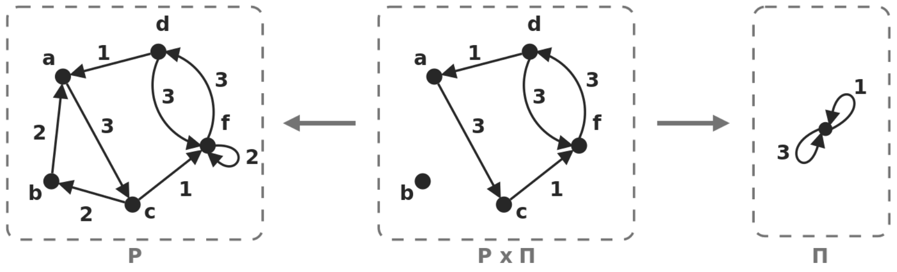


Figure 3: Deletion of 2-arcs using a product in the category $[\![1,3]\!]$-**Graph**.

functors. The embedding functor $\mathbb{E}_\Sigma : \Sigma$-**Graph** $\to (\Sigma^2)$-**Graph** transforms an i-arc ($i \in \Sigma$) into $|\Sigma|$ arcs labeled (i,j) for each $j \in \Sigma$, making each relabeling possible. The projecting functor $\pi_\Sigma : (\Sigma^2)$-**Graph** $\to \Sigma$-**Graph** keeps the relabeled part of arc labels. Formally, the embedding and projecting functors are defined as follows:

Definition 5 (Embedding and Projecting Functors)**.** *Let us consider a finite alphabet* Σ.

The embedding functor $\mathbb{E}_\Sigma : \Sigma$-**Graph** $\to (\Sigma^2)$-**Graph** *transforms:*

- *A graph $G = (V, E_G, s_G, t_G, l_G)$ into a graph $G' = (V, E_{G'}, s_{G'}, t_{G'}, l_{G'})$ such that for every i-arc e of E_G with i in Σ, for all $j \in \Sigma$, there exists an (i,j)-arc in $E_{G'}$ with the same source and target, written $e^{(j)}$.*
- *A morphism $m = (m_V, m_E) : F \to G$ into a morphism $m' = (m_V, m'_E) : F' \to G'$ such that for every e_F in E_F, for every j in Σ, $m'_E(e_F^{(j)})$ is the arc $m_E(e_F)^{(j)}$.*

The projecting functor $\pi_\Sigma : (\Sigma^2)$-**Graph** $\to \Sigma$-**Graph** *transforms:*

- *A graph $G = (V, E, s, t, l_G)$ into a graph $G' = (V, E, s, t, l_{G'})$ such that $l_{G'}$ only keeps the second part of the label from l_G.*
- *A morphism $m = (m_V, m_E) : F \to G$ into a morphism $m' = (m_V, m'_E) : F' \to G'$ such that if an (i,j)-arc e_F is sent to e_G by m_E then the j-arc $e_{F'}$ built on e_F is sent to the j-arc $e_{G'}$ built on e_G.*

The projecting functor π_Σ is the projection inherited from π_2 on the product **Set** $\times$ **Set**. Let us consider the relabeling defined by relabeling set $\mathfrak{R}$. The construction works as follows :

1. Embed P in the category Σ^2-**Graph** to get $\mathbb{E}_\Sigma(P)$.
2. Construct the product $\mathbb{E}_\Sigma(P) \times_{\Sigma^2} \Pi_{\mathfrak{R}}$, where $\Pi_{\mathfrak{R}}$ is the relabeling graph having only one node, and a loop labeled (i,j) for each pair (i,j) in the relabeling set $\mathfrak{R}$.
3. Apply the projecting functor π_Σ to $\mathbb{E}_\Sigma(P) \times_{\Sigma^2} \Pi_{\mathfrak{R}}$.

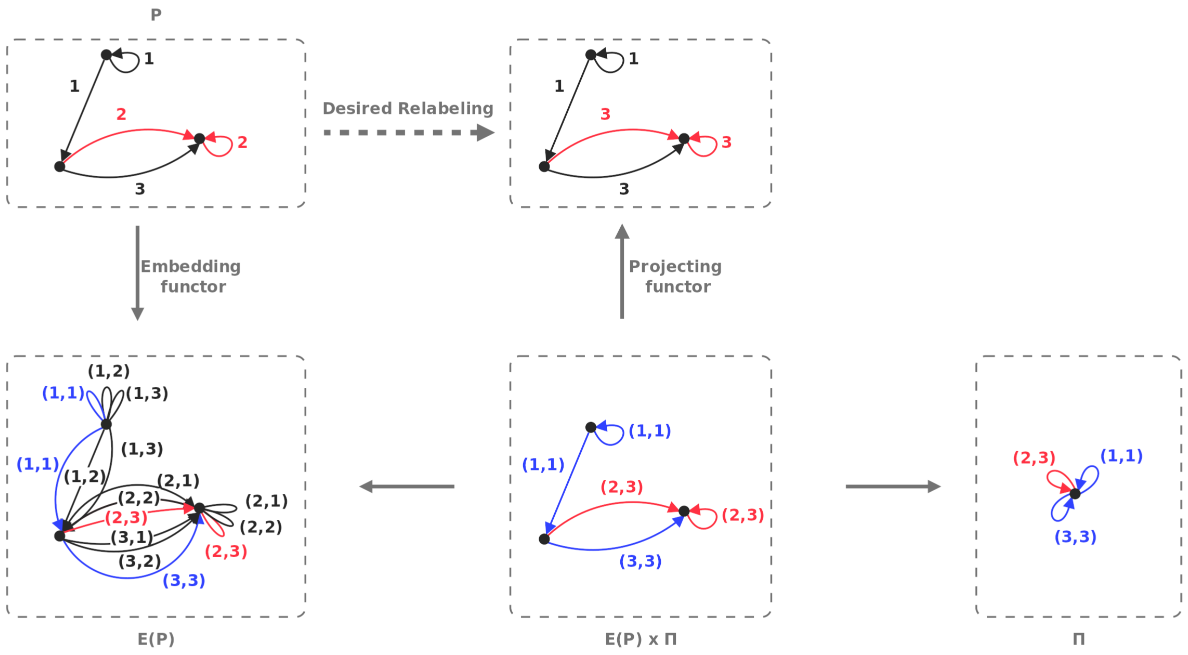


Figure 4: Global relabeling with a product.

Example 3 (Relabeling with Products). Assume we want to relabel all the 2-arcs of a graph into 3-arcs, as shown in Figure 4. First, we apply the embedding functor from $\{1,2,3\}$-**Graph** to $\{1,2,3\}^2$-**Graph** : from each arc, three arcs are built, having 1, 2 and 3 as second label. Then, we construct the product with the relabeling graph (written Π in the figure). Arcs in red will give rise to relabeled arcs at the end of the transformation. Arcs in blue are the arcs left unchanged. Arcs in black are unused in the product. The relabeling set is $\mathfrak{R}(1,3) = \{(1,1),\ (2,3),\ (3,3)\}$ and yields the graph that consists of a single node and three arcs, each specifying the relabeling of one letter: 2 is relabeled 3 whereas 1 and 3 are left unchanged. The product construction only keeps the arcs that match the relabeling. Lastly, the projecting functor erases the first label on the arcs, and we obtain the desired relabeling.

Before providing details on the graph formalism used in geometric modeling, we want to point out that DPO- and product-based transformations are intended to serve different purposes. We will exploit DPO rewriting as a generic tool to carry out modifications on the underlying object. We will utilize the product mechanism as a general construction to achieve modifications as higher-level rules. Graph transformations using products provide a framework for the parallel generalization of rewriting rules without postponing the parallelization to the application time. In particular, we will use the product construction to generalize relabeling. By definition, the product construction is symmetrical in P and Π. However, we will distinguish between P and Π in the transformation. P will serve as a pattern to embed the graph structure, whereas Π will sketch the desired modifications. The generalization of transformations with pullbacks (see Section 6) will appear as the cornerstone of our approach, satisfying our need for static verification at rule-design time.

3. Geometric modeling

In this section, we will introduce G-maps and O-maps as edge-labeled graphs with constraints and use these denominations in the remaining part of the paper. We first introduce geometrical modeling and its formal issues. In this context, we use the terminology given in [11] and call combinatorial maps the family that encompasses oriented and generalized maps, among other models. In the literature, the term 'combinatorial maps' has mostly been used to refer to the specific model of oriented maps.

In solid modeling and computer-aided design, many modelers rely on a boundary representation of objects, or B-rep. This method represents a solid as a collection of connected surface elements. These elements define the boundary between the interior of the volume and its exterior. B-rep modelers use different data structures to encode the topological relations of the surface elements, such as winged-edges, half-edges, or quad-edges. In [10], the author shows that such data structures can be formalized as oriented maps. In [27], the authors provide a complete conversion from half-edges to 2D oriented maps.

Combinatorial maps present two main advantages. The formalism allows defining objects in any dimension (1D, 2D, 3D, etc.). The definition of a generalized (resp. oriented) map is only parametrized by the maximal dimension. The aforementioned topological data structures are dedicated to specific dimensions and target application domains. The formal definition of combinatorial maps allows us to consider objects from any application domain thanks to its cell decomposition. The atomic elements of the decomposition are called darts and encoded by the nodes in a graph-based approach. Nodes are linked by arcs that describe neighboring relations between two cells.

The second main asset comes from the explicit expression of consistency constraints. These constraints state how the darts can be linked so that the modeled object is a quasi-manifold. Compared to the edges data structures, these constraints are defined within the model rather than hard-coded in the data structure and associated operations. Therefore, we can reason on these constraints. For this exact justification, combinatorial maps have proven fruitful to structure knowledge in [28, 29, 7, 30]. Hypermaps, a generalization of oriented maps with hypergraphs, were used to derive a Coq proof of the discrete form of the Jordan Curve theorem in [31].

The algebraic definition of oriented maps is not homogeneous in dimension. A new model, called generalized maps, was defined handling the 0 dimension to bypass this limitation [14]. As a result, orientation is lost, and the number of darts is doubled. From the algebraic definitions given in [11], G-maps correspond to open generalized maps, whereas O-maps correspond to closed oriented maps. Combinatorial maps are usually defined as topological relations on a set of darts. Considering the darts as graph nodes and the topological relations as graph arcs, we derive graph-based definitions of generalized maps and oriented maps. Consistency properties of the topological relations yield constraints on the graph that have been studied for the model of generalized maps [16]. We aim at extending this study to the more popular model of oriented maps. Once oriented and generalized maps are defined from a graph perspective, their transformations can be designed using graph transformations. In DPO rewriting, the rule interface contains the parts matched and preserved by the rule, which encodes some neighborhood of the modified part of the graph. Conditions on rules for consistency preservation were provided in [16] for the restricted case of generalized maps. These conditions led to the definition of a generic topological-based geometric modeler in [32] based on G-maps. Although the modeler can be used to apply modification on an object, its foremost interest is the rule editor. Geometric modeling operations can be conceived and written as a simple rule. Orbit variables were introduced to generalize rules to an orbit. Intuitively, an orbit is a subgraph induced by a subset of arc labels and allowed to retrieve topological cells in a generalized map. The orbit variables sanctioned the use in geometric modeling as the operation became independent of the underlying topology. At that point, the possibility to deal with domain-related information (called embeddings) was the main bottleneck of the approach. The introduction of I-labeled graph in [33] solved this issue and defined DPO rewriting for graphs where nodes are labeled by a family of labels. Jerboa [17] offers a graphical rule editor along with a syntax checker that guarantees the consistency of G-map rules (both for the topological aspect and the omission of geometric aspect). In [34] we solved geometrical consistency by statically detecting embedding computation equivalence on nodes of the same orbit.

We recall the topological constraints in Section 3.1 and provide a graph-based definition of generalized maps and oriented maps in Section 3.2. We discuss the formalization of modeling operations as graph transformation rules in Section 3.3 and the need for consistency preservation in Section 3.4.

3.1. Topological constraints

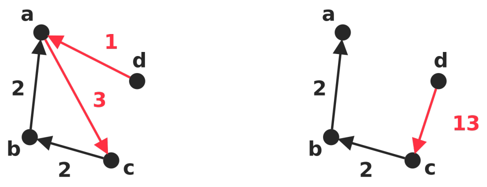
For our concerns, we will consider three kinds of label sets: dimensions, words, and pairs, which we will respectively use to model neighboring relations, paths in topological models, and modifications to be applied to graphs. We give names to each kind of graph to convey their respective usage.

We will use the category $\mathbb{D}$-**Graph** where $\mathbb{D}$ is a finite set of integers, e.g, $\mathbb{D} = [\![1,4]\!]$, the set of integers between 1 and 4 (included). As integers describe the neighboring relations in topological models, we call topological graph any graph labeled with integers from $\mathbb{D}$.

Definition 6 (Topological Graph)**.** *Let $\mathbb{D}$ be a finite set of integers, called* dimension set*. A* $\mathbb{D}$-topological graph *is a graph from the category $\mathbb{D}$-**Graph**.*

The set $\mathbb{D}$ will also be used as an alphabet for more structured sets. We will consider categories $\mathbb{W}$-**Graph** where $\mathbb{W}$ is a finite subset of $\mathbb{D}^*$. Intuitively, arc labels in such graphs describe paths in an underlying graph. For instance, in Figure 5b, the red arc labeled 13 can be interpreted as the 13-path in the graph of Figure 5a. We will call pattern graph any graph labeled with words from $\mathbb{W}$.

Definition 7 (Pattern Graph)**.** *Let $\mathbb{D}$ be a finite set of integers, and let $\mathbb{W}$ be a finite subset of $\mathbb{D}^*$. A* $\mathbb{W}$-pattern graph *is a graph from the category $\mathbb{W}$-**Graph**.*



(a) A $[\![1,3]\!]$-graph. (b) A $\{13,2\}$-graph.

Figure 5: Our main consideration for the label sets.

Since a function f from a set E to a set F can be defined by a set of pairs (x,y) in $E \times F$ with $f(x) = y$, we will use graphs labeled with pairs in $E \times F$ to encode graph transformations involving the use of such functions. More precisely, we will use functions from path labels to arc labels, that is, functions from $\mathbb{W}$ to $\mathbb{D}$. We call graph scheme any graph labeled with pairs from ($\mathbb{W}$, $\mathbb{D}$).

Definition 8 (Graph Scheme)**.** *Let $\mathbb{D}$ be a finite set of integers, and let $\mathbb{W}$ be a finite subset of $\mathbb{D}^*$. A* $(\mathbb{W},\mathbb{D})$-graph scheme *is a graph from the category ($\mathbb{W} \times \mathbb{D}$)-**Graph**.*

We consider three properties, also called topological constraints, defined locally on labeled graphs:

Definition 9 (Topological Constraints)**.** *Let $G = (V_G, E_G, s_G, t_G, l_G)$ be a Σ-labeled graph, v be a node of V_G, and i and j be two labels of Σ. The node v satisfies*

- *The* incident arcs constraint $\mathscr{I}_G(i)$ *if v is the source of a unique i-arc and the target of a unique i-arc.*
- *The* non-orientation constraint $\mathscr{O}_G(i)$ *if the i-arcs incident to v are non-oriented.*
- *The* cycle constraint $\mathscr{C}_G(i,j)$ *if v is the source of an $ijij$-cycle.*

These constraints can be extended to a subset of labels $I \subseteq \Sigma$ (or pairs of labels $J \subseteq \Sigma^2$ for the cycle constraint) and a subset of nodes $V' \subseteq V$. We say G satisfies the constraint $\mathscr{I}_{V',G}(I)$, $\mathscr{O}_{V',G}(I)$, or $\mathscr{C}_{V',G}(J)$ whenever each node in V' satisfies the corresponding constraint. If $V' = V$, we simply say G satisfies $\mathscr{I}_G(I)$, $\mathscr{O}_G(I)$, or $\mathscr{C}_G(J)$.

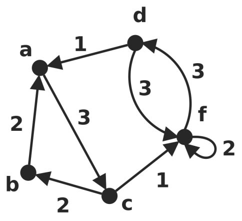


Figure 6: A $[\![1,3]\!]$-graph.

When dealing with the cycle constraint, we call the set of *exchangeable dimensions* the set $J \subseteq \Sigma^2$ of pairs of labels.

Example 4 (Topological Constraints). Let G be the graph represented in Figure 6. G is a $[\![1,3]\!]$-graph. Node b is the source of a single 2-arc and the target of a single 2-arc, thus node b satisfies the incident arcs constraint $\mathscr{I}_G(2)$. On the contrary, node c is not the source of a 3-arc and does satisfy $\mathscr{I}_G(3)$. Since there is a 3-arc of source d and target f and a 3-arc of source f and target d, node f satisfies the non-orientation constraint $\mathscr{O}_G(3)$. Similarly, since the only 2-arc incident to node f is a loop, it also satisfies $\mathscr{O}_G(2)$ and therefore satisfies $\mathscr{O}_G(\{2,3\})$. Conversely, node a does not satisfy $\mathscr{O}_G(3)$ because the 3-arc between nodes a and c does not admit a reverse arc. Note that no 2-arc is incident to node d, thus node d also satisfies $\mathscr{O}_G(2)$. Finally, as node c is the source of a 1313-cycle, node c satisfies the constraint $\mathscr{C}_G(1,3)$, whereas node a is the source of a 3131-cycle and satisfies $\mathscr{C}_G(3,1)$.

Note that the incident arcs constraint guarantees the existence and uniqueness of a path given a sequence of labels, which will massively simplify the study of consistency for the cycle constraint. Indeed, if a graph satisfies the incident arcs constraint for two dimensions, then the order of the dimensions does not matter when considering the cycle constraint at the whole graph scale.

Lemma 1. *Let G be a $\mathbb{D}$-topological graph and i, j two dimensions in $\mathbb{D}$. Suppose that G satisfies $\mathscr{I}_G(\{i,j\})$, then G satisfies $\mathscr{C}_G(i,j)$ if and only if G satisfies $\mathscr{C}_G(j,i)$.*

Proof. The lemma holds trivially from the incident arcs constraint. $\square$

3.2. *Graph-based definitions of combinatorial maps*

The incident arcs constraint allows us to unambiguously consider arcs based on their label and source (or target). This feature will prove helpful to generalize graph transformations. Therefore, we call a combinatorial graph any topological graph that satisfies the incident arcs constraint on every dimension. We define O-maps and G-maps based on combinatorial graphs.

Definition 10 (Combinatorial Graphs, Generalized Maps, and Oriented Maps)**.** *Let $\mathbb{D}$ be a finite subset of $\mathbb{N}$ and let $\mathbb{D}_{+2}$ denotes the set of pairs $(i,j) \in \mathbb{D}^2$ such that $i+2 \leq j$.*

A $\mathbb{D}$-topological graph is a $\mathbb{D}$-combinatorial graph if it satisfies the incident arcs condition $\mathscr{I}_G(\mathbb{D})$.

*An n-*dimensional generalized map*, or n-G-map is a $[\![0,n]\!]$-combinatorial graph satisfying $\mathscr{O}_G([\![0,n]\!])$ and $\mathscr{C}_G([\![0,n]\!]_{+2})$.*

*An n-*dimensional oriented map*, or n-O-map is a $[\![1,n]\!]$-combinatorial graph satisfying $\mathscr{O}_G([\![2,n]\!])$, and $\mathscr{C}_G([\![1,n]\!]_{+2})$.*

It is possible to define other models of [11] by twisting the constraints. For instance, a closed generalized map can be defined as a G-map where no arc is a loop. An open oriented map can be defined by relaxing the incident arcs condition on the maximal dimension n to “every node v is the source of, at most, one n-arc and the target of, at most, one n-arc”. Similarly, the dual model of oriented maps (open or closed) can be obtained by enforcing the non-orientation condition on the minimal dimension 1 but removing it on the maximal dimension n.

The study done in [16] covered G-maps, which is here extended to cover O-maps, and one could extend it to the other models. We chose to keep the study simple by narrowing the scope to these two models as they are the more regular ones, which simplifies the automatization of the transformation mechanism.

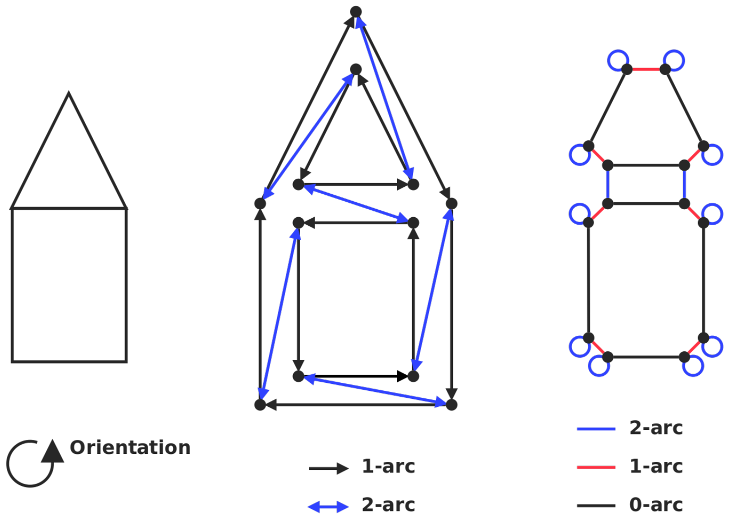
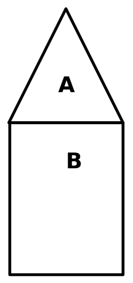


Figure 7: An object and its representation as a 2-O-map (middle) and a 2-G-map (right).

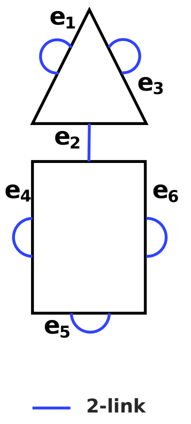
Example 5 (G-map and O-map). Let us illustrate both models with the example of Figure 7. In this figure, as it will be in the whole paper, non-oriented arcs are either represented as a line or as a double-oriented arrow. The modeled object (on the left-hand side) consists of a triangle on top of a square in 2D. The corresponding 2-O-map consists of three cycles of 1-arcs for the two faces plus the outer one. The 1-arcs are oriented according to the orientation provided, orienting each face. The 2-arcs link faces along edges. The 2-G-map associated with the same model is given on the right-hand side. In a G-map, a face consists of an alternation of 0-arcs and 1-arcs. Note that the outer face is not represented in a G-map, and outer nodes are the source of 2-loops.

We can intuitively reconstruct G-maps and O-maps from the object decomposition into cells.

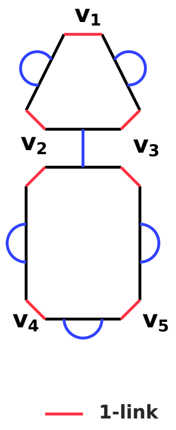
Example 6 (Object Decomposition). The representations of the object of Figure 7 as 2-G-map and 2-O-map result from the decomposition into decreasing dimensions. In the case of G-maps, we consider the object open, i.e., as the two faces A and B of Figure 8a. First, the object is split into faces sharing an edge with 2-links, in blue on Figure 8b. Likewise, the faces are decomposed into edges via 1-links, in red on Figure 8c. Finally, edges are dissociated with 0-links, in black on Figure 8d. The resulting nodes are the nodes of the Gmap, and the dimensional relations are the arcs of the G-map. The decomposition is similar for the model of O-maps, as illustrated in Figure 9. O-maps represent closed-oriented objects. Therefore, in Figure 9a, the faces A and B are oriented, and the outer face C is added. All faces have the same orientation. Thus, an edge shared between two faces is oriented in opposite directions in each face. For instance, the edge between faces A and B is oriented from left to right in the face A and from right to left in the face B. In the decomposition process, we first split the faces along their joint edges, as illustrated in Figure 9b. Then, edges are separated via 1-links. The decomposition stops at this step and provides the graph of Figure 9c. The nodes of this graph correspond to darts that intuitively represent parts of the edges from the initial object.



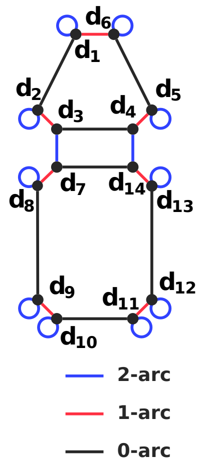
(a) Two faces sharing an edge.



(b) Decomposition along dimension 2.

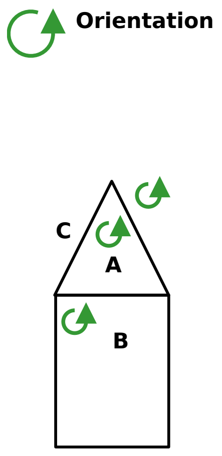


(c) Decomposition along dimension 1.

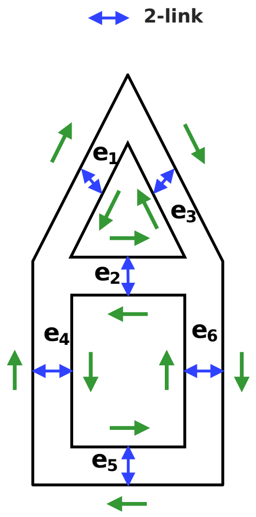


(d) Decomposition along dimension 0 and resulting 2-G-map.

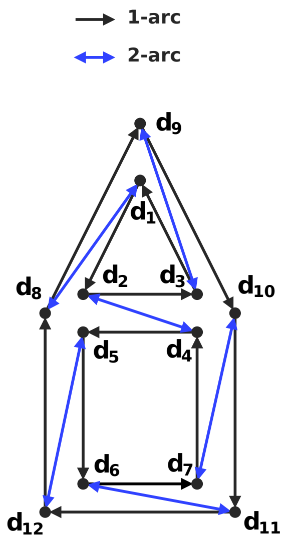
Figure 8: Decomposition of an object and construction of the underlying G-map.



(a) Closed representation of the same object as three faces.



(b) Decomposition along dimension 2.



(c) Decomposition along dimension 1.

Figure 9: Decomposition of an object and construction of the underlying O-map.

3.3. Modeling operations as graph transformations

Given that G-maps and O-maps are defined as graphs, the modeling operations can be formalized as graph transformations. For instance, the operation depicted in Figure 10 subdivides the face on the left to derive the four faces on the right. This example is a specific instance of quad subdivision dedicated to refining the topological structure of meshes [35]. This operation adds new vertices at the center of the face and the middle of each edge. An edge links the face center and the midpoint of the original edges.

The recursive subdivision into decreasing cell dimension yields the operation depicted by the G-maps of Figure 11a, for the object of Figure 10. The initial square face is represented by 8 nodes that are preserved by the transformation. The four 1-arcs and 2-arcs are also preserved while the 0-arcs are removed. The midpoints, the face center, and the linking edges are added. Based on the status of the element (added, preserved, and deleted),

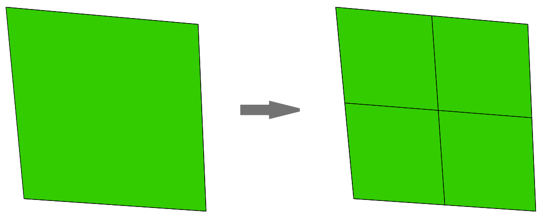
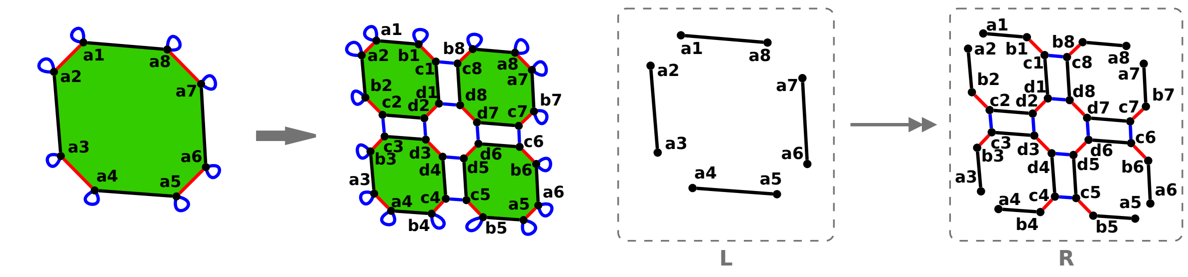


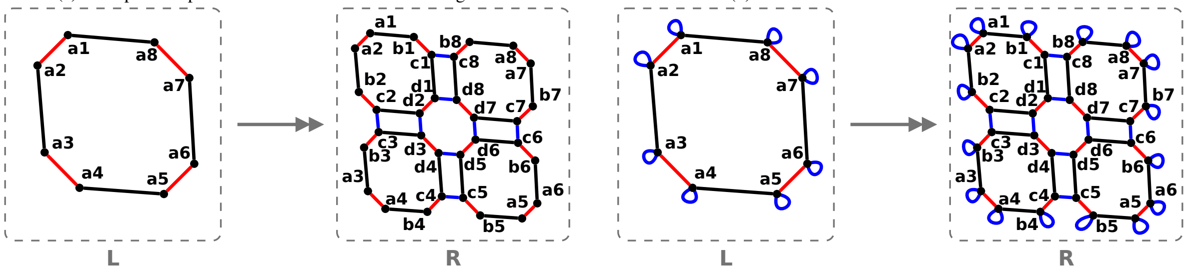
Figure 10: Quad subdivision of a face.

we determine possible rules for the operation. The rule in Figure 11b appears inconvenient because it does not specify that the matched element should be included in the same face. The rule in Figure 11c might seem like the most general, but it does not match the whole edges of the initial face. If the rule were applied to a non-isolated face, the half-edges in the surrounding faces would not be split. Finally, the rule in Figure 11d will never break the model consistency but is only applicable to isolated faces. From a purely algorithmic perspective, the quad subdivision operation can be defined without restriction to (non-)isolated faces. Besides, the rules presented here only apply to faces with four vertices, while the geometric modeling operation is applicable on any face regardless of the number of edges.



(a) G-map decomposition of the face subdivided in Figure 10.

(b) Minimal rule to obtain the transformation.



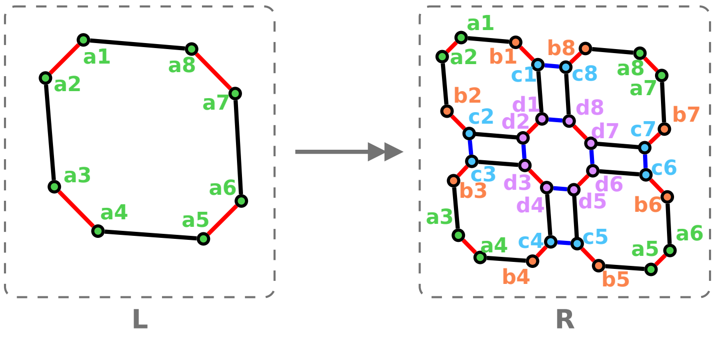
(c) Intermediate rule to obtain the transformation.

(d) Maximal rule to obtain the transformation.

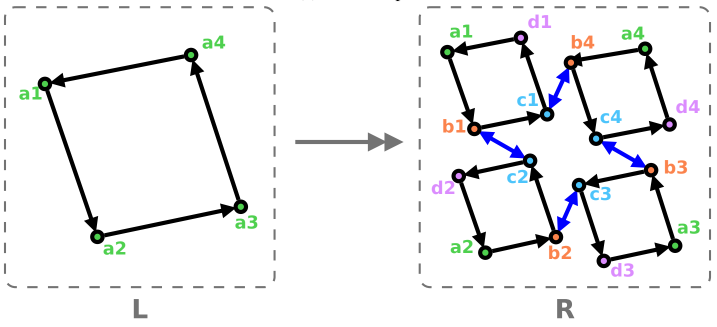
Figure 11: Quad subdivision of a face as a graph transformation.

In previous works, we used topological variables to perform this generalization. The topological variables allowed matching a subgraph induced by a set of dimensions. This subgraph could be copied, and the arcs could be relabeled. Arcs could be added between copies, meaning that each node of one copy of the subgraph would be linked to its alter ego in another copy. Such subgraphs are called orbits [11], and the definition varies between oriented and generalized maps. From an algebraic perspective, orbits in oriented maps require composing the underlying permutations on nodes. From a graph perspective, orbits in O-maps require considering paths in the underlying graph. From this single modification, our approach based on dimension relabeling did not provide a convenient solution. As illustrated in Figure 12a, the quad subdivision of a face in a G-map setting can be decomposed into various copies of the initial face where the initial arcs are relabeled. The corresponding transformation is given in Figure 12b in an O-map setting. In Figure 12, nodes are colored and named based on the copy of the initial subgraph (from the left-hand side). For instance, in the right-hand side of the rule of

Figure 12a, the nodes $c1$, $c2$, $\ldots c8$ belong the same copy of the initial subgraph corresponding to the nodes $a1$, $a2$, $\ldots a8$. Similarly, nodes bi and di ($1 \leq i \leq 8$) describe two other copies of the initial subgraph. On the contrary, nodes $b8$, $c8$ and $d8$ belong to different copies. The number indicates the initial node of a copy node. As such, nodes $b8$, $c8$ and $d8$ are copies of the node $a8$. Rules with topological variables allowed arcs that either link nodes between a single copy, such as nodes $c1$ and $c8$ or nodes $a5$ and $a6$, or link different copies of the same node, such as nodes $c8$ and $d8$ or nodes $a4$ and $b4$. These variables are not expressive enough to generalize the rule for O-map depicted in Figure 12b because node $b1$ is linked to node $c2$. The regularity of the model still provides ways to describe the operation homogeneously. For example, one can notice that each node bi is linked to the node $c(i+1 \mod 4)$. We will use path labels to encode these links. The extraction of paths to create new arcs proved challenging and resulted in one of the main novelties presented in this paper.



(a) For G-maps.



(b) For O-maps.

Figure 12: Generalization of the quad subdivision using copies and relabeling.

As a final note, we want to highlight the size of graphs used in practice. In Figure 13, the character consists of 12587 vertices, 12585 faces and a single connected component. The 3D character is represented by its boundary surface, i.e., a 2D manifold. The G-map decomposition produces 100680 nodes and 302040 arcs in 2D (Figure 13), while the O-map decomposition yields 50340 nodes and 100680 arcs (not drawn). We will see in Section 10 that the rule scheme for each model has one node in the left-hand side and four nodes in the right-hand side, regardless of the object size. The possibility to express operations as rules independent from the object size is an essential asset in geometric modeling as the objects are usually huge. The quad subdivision depicted in Figure 13 has been realized with Jerboa [17] with G-maps. We can see two ranges of faces on the detailed views that correspond to the front and the rear parts of the character since it is cylindrical.

3.4. Consistency preservation in modeling operations

Since G-map and O-map definitions are based on topological constraints, transformations on the underlying graph need to preserve these constraints. These topological constraints are elementary (incident arcs, non-orientation, and cycle); thus, we can study the conditions for their preservation sequentially and separately for DPO-based transformations and product-based transformations. Superimposing the corresponding conditions

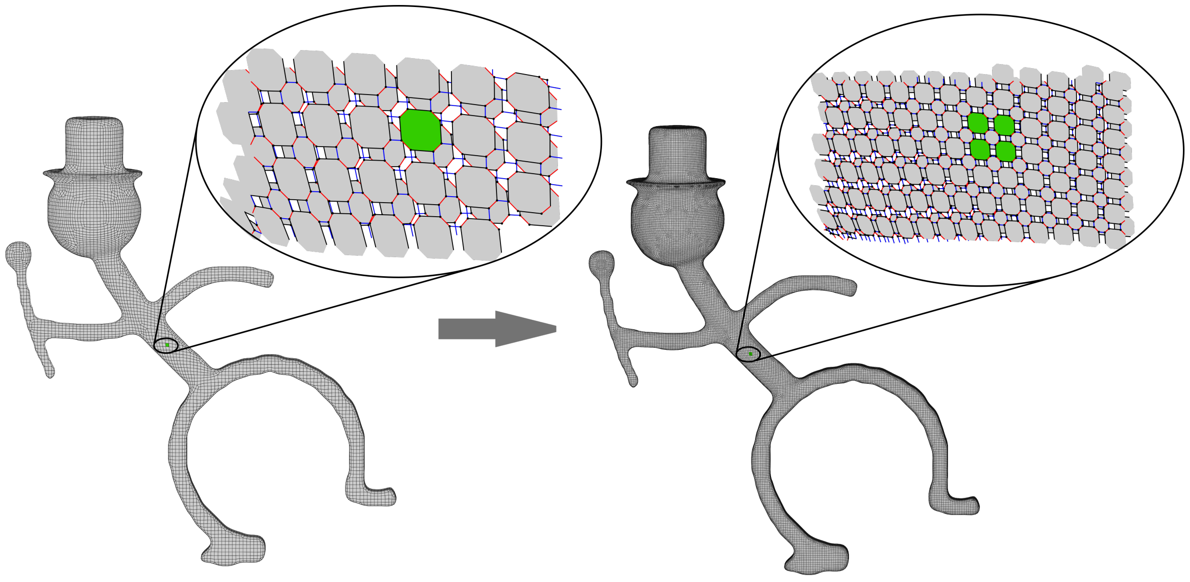


Figure 13: Quad subdivision of a surface.

will ensure the preservation of the model consistency. In the remaining part of the paper, we will use the denomination consistency preservation to refer to either constraint, depending on context.

We will give necessary and sufficient conditions to preserve the topological constraints on a local scale in topological graphs. Note that the non-orientation property was not a concern for generalized maps. Imposing that all graphs be non-oriented was sufficient to preserve the non-orientation constraint [16]. With the introduction of combinatorial maps, the study needs to be more general. This study leads us to provide a weaker yet sufficient condition for preserving the non-orientation constraint. For the incident arcs condition, the proof of the consistency preservation remains an extrapolation of the one given in [16].

The incident arcs property is the cornerstone of the model. This property provides a framework that is rich enough to define interesting abstractions of graph transformations via rule schemes using products. Since the operations are performed using direct transformations from DPO rewriting, conditions on rules need to be lifted to rule schemes. Lifting the conditions from DPO rules to rule schemes is relatively straightforward for the conditions of incident arcs (Section 7) and non-orientation (Section 8). However, the extension of the cycle condition for rules comes with additional constructions to deal with possible orientation for dimensions (Section 9).

4. Necessary and sufficient condition for incident arcs preservation in DPO rewriting

In Section 3.1, we presented the three topological constraints (incident arcs, non-orientation and cycles). We now study conditions on rules to explain how a graph can be modified while preserving its consistency. We want to ensure consistency at the rule level to provide feedback to the rule designer. In particular, our only assumption on the match morphism concerns its injectivity. We strive to establish conditions that can be statically checked on the rules without hindering their applicability.

Consistency preservation has been studied in previous work [16, 34], but only sufficient conditions were given. As a result, the syntax analyzer [32, 17] would raise false negatives and consider valid rules as inconsistent. In this section, we provide necessary and sufficient conditions for the preservation of the incident arcs constraint. The approach developed in this section exploits a set-theory formalism and can be used as a reference for the discussion of Section 11 regarding consistency preservation in other frameworks. Combinatorial graphs play a key role in our framework and will prove essential for the study conducted in Section 5 on the remaining two constraints.

The weak incident arcs condition states that preserved nodes should retain their incidence to a dimension, and added nodes should satisfy the incident arcs constraint.

Definition 11 (Weak Incident Arcs Condition)**.** *A rule $r = L \twoheadrightarrow R$ satisfies the* weak incident arcs condition *$\mathscr{I}_r(i)$-weak if it satisfies the two following sub-conditions :*

1. *Any preserved node of $L \cap R$ is the source (resp. target) of an i-arc in L if and only if it is the source (resp. target) of an i-arc in R.*
2. *R satisfies $\mathscr{I}_{(V_R \setminus V_L),R}(i)$, i.e., any added node of $V_R \setminus V_L$ is the source of a unique i-arc and the target of a unique i-arc in R.*

When this condition is fulfilled, the derivation yields a graph satisfying the incident arcs constraint.

Theorem 1 (Incident Arcs Preservation)**.** *Let $r = L \twoheadrightarrow R$ be a rule in the category of $\mathbb{D}$-graphs.*

The rule r satisfies the weak incident arcs condition $\mathscr{I}_r(i)$-weak if and only if for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-graph satisfying $\mathscr{I}_G(i)$, the result graph H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the incident arcs constraint $\mathscr{I}_H(i)$.

Proof. The first part of this proof ($\Rightarrow$) has been shown in [16]. Intuitively, regardless of whether preserved nodes are the source of an arc in L (i.e., whether the arc is matched), sub-condition 1 ensures the same status in R, yielding an arc in H. Similarly, sub-condition 2 guarantees the property for added nodes.

($\Leftarrow$). Suppose the result graph H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the incident arcs constraint $\mathscr{I}_H(i)$, for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-graph satisfying $\mathscr{I}_G(i)$. Consider such a graph H and m' the mono $R \hookrightarrow H$.

Sub-condition 1 of the weak incident arcs condition.

Let v be a preserved node of $L \cap R$ that is the source of an i-arc e in L. Then $m(v)$ is the source of $m(e)$ in G. Suppose this arc is deleted by the application of the rule (otherwise e is in R). However, H satisfies $\mathscr{I}_H(i)$ so there exists an i-arc e_H in H of source $m'(v)$. This arc is not in G because $m(v)$ is the source of only one i-arc in G. Thus, an arc e' exists in R such that $m'(e') = e_H$. By construction the source of e' is v. Therefore v is the source of an i-arc in R. The construction holds when taking v as the target of e or inverting L and R. Hence, any preserved node of $L \cap R$ is the source (resp. target) of an i-arc in L if and only if it is the source (resp. target) of an i-arc in R.

Sub-condition 2 of the weak incident arcs condition.

Let v be an added node in $V_R \setminus V_L$. Then $m'(v)$ is an added node in H. Since H satisfies $\mathscr{I}_H(i)$, then an i-arc e_H of source $m'(v)$ exists in H. Since v is not in $L \cap R$, any incident arc of v comes from R, i.e., there exists e in R such that $m'(e) = e_H$ and $s_R(e) = v$. Because e_H is unique, so is e. Thus, v is the source of a unique i-arc in R. The proof holds when replacing source by target and R satisfies $\mathscr{I}_{(V_R \setminus V_L),R}(i)$.

Thereafter r satisfies the weak incident arcs condition $\mathscr{I}_r(i)$-weak. □

Example 7 (Incident Arcs Preservation). Consider the rule in Figure 14, where the black elements are both in L and R, thus in the interface. The rule satisfies the weak incident arcs condition for the dimension 2. Indeed, interface nodes b and d are the source of a unique 2-arc in both sides, whereas nodes a and c are the target of a unique 2-arc in both sides. The added node e satisfies the incident arcs constraint for the dimension 2 as it is the source (resp. target) of a unique 2-arc. An example of DPO transformation depicted for this rule is given in Figure 15. Node names have been removed to simplify the figure. The matched parts are drawn in red, while non-oriented arcs are marked with two arrows. The graph G in the bottom left corner satisfies the incident arcs

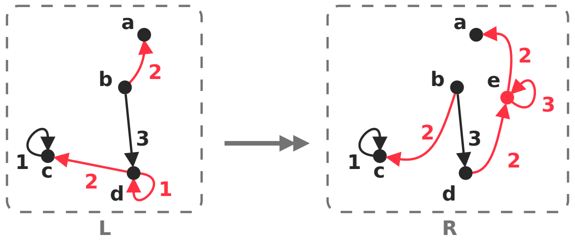


Figure 14: A rule satisfying the incident arcs condition for the dimension 2.

constraint for the dimension 2. As we can see, the nodes in G, not matched by the rule, keep their incident arcs while matched nodes are held with the same incidences in R. Thus, the direct derivation yields a graph satisfying the incident arcs constraint for the dimension 2.

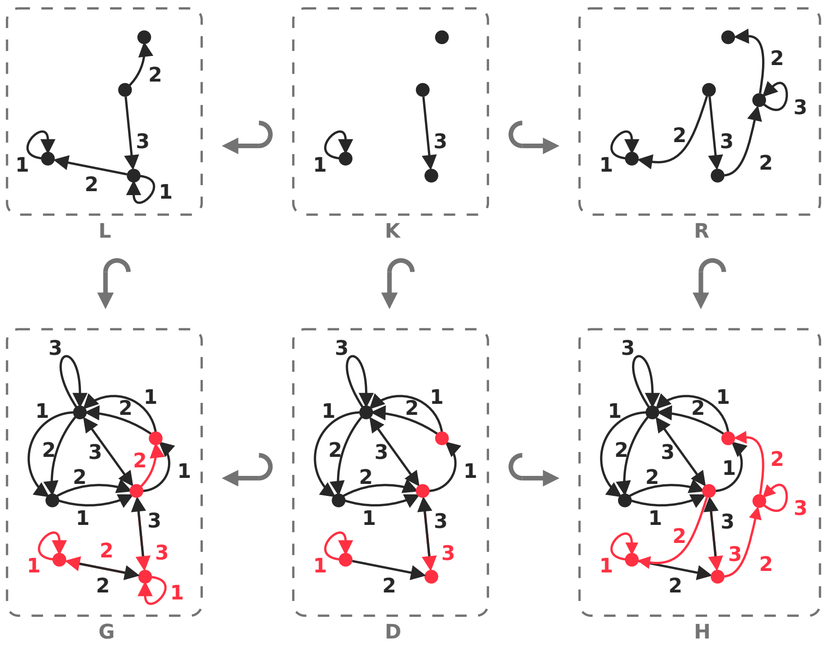


Figure 15: Incident arcs preservation.

The dangling condition is usually a critical concern [24, 1] in the DPO approach to graph transformations. As shown in [16], the gluing condition can be statically checked on the rule when applied to combinatorial graphs: a match $m : L \hookrightarrow G$ for a rule $r = L \twoheadrightarrow R$, where G is a $\mathbb{D}$-combinatorial graph, satisfies the dangling condition if and only if L satisfies $\mathscr{I}_{(V_L \setminus V_R),L}(\mathbb{D})$. We extend the weak incident arcs condition to incorporate the dangling condition aggregating in the incident arcs condition.

Fact 1. For a $\mathbb{D}$-rule applied to a $\mathbb{D}$-combinatorial graph, the dangling condition on the match is equivalent to the incident arcs constraint on the left-hand side of the rule, restricted to the deleted elements (proved in [16]).

Definition 12 (Combinatorial Rule)**.** *A rule $r = L \twoheadrightarrow R$ satisfies the* incident arcs condition $\mathscr{I}_r(\mathbb{D})$ *if it satisfies the weak incident arcs condition $\mathscr{I}_r(\mathbb{D})$-weak and L satisfies $\mathscr{I}_{(V_L \setminus V_R),L}(\mathbb{D})$. Then, r is called a* $\mathbb{D}$-combinatorial rule.

Combinatorial graphs and rules play a crucial role in our framework as the incident arcs constraint guarantees the existence and uniqueness of arcs given a label (and a source node). The remaining part of this article relies heavily on this constraint, and we will consider (unless stated otherwise) that the following assumptions hold.

Assumption 3. Graphs under modification belong to the category of combinatorial graphs. This category is the full subcategory of $\mathbb{D}$-**Graph** whose objects are the $\mathbb{D}$-combinatorial graphs. Rules are combinatorial rules, transforming a combinatorial graph into a combinatorial graph.

5. Consistency preservation in DPO rewriting for combinatorial graphs

With the benefit of the incident arcs constraint and condition, we now present necessary and sufficient conditions to preserve both the non-orientation constraint and the cycle constraint. A watchful reader will notice that we could impose a less restrictive constraint. Indeed the incident arcs constraint and condition for a dimension i are enough to deal with the non-orientation property on this same dimension. Similarly, the incident arcs constraint for the dimensions i and j are sufficient for the cycle property on these same dimensions.

5.1. Non-orientation preservation

The non-orientation preservation boils down to only having oriented arcs in the interface $L \cap R$.

Definition 13 (Non-Orientation Condition). *A $\mathbb{D}$-combinatorial rule $r = L \twoheadrightarrow R$ satisfies the* non-orientation condition *$\mathscr{O}_r(i)$ if any deleted (resp. added) i-arc of $E_L \setminus E_R$ (resp. $E_R \setminus E_L$) has a unique reverse i-arc that is also deleted (resp. added), i.e., in $E_L \setminus E_R$ (resp. $E_R \setminus E_L$).*

We now give the corresponding theorem.

Theorem 2 (Non-Orientation Preservation). *Let $r = L \twoheadrightarrow R$ be a $\mathbb{D}$-combinatorial rule.*

The rule r satisfies the non-orientation condition $\mathscr{O}_r(i)$ if and only if for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{O}_G(i)$, the result graph H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the non-orientation constraint $\mathscr{O}_H(i)$.

Proof. Let $r = L \twoheadrightarrow R$ be a $\mathbb{D}$-combinatorial rule.

($\Rightarrow$). Suppose r satisfies the non-orientation condition $\mathscr{O}_r(i)$.

Let $m : L \hookrightarrow G$ be a match on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{O}_G(i)$, H be the result of the direct derivation $G \Rightarrow^{r,m} H$, m' be the comatch $R \hookrightarrow H$, and v_H be a node in V_H.

Suppose e_H is an i-arc incident to v_H. Let us show that e_H has a unique reverse i-arc in H.

1. If $e_H \in H \setminus m'(R)$ then $e_H \in G \setminus m(L)$. Thus, there exists $e_G \in G$ such that $e_G = e_H^{-1}$ in G.
   (a) If $e_G \in G \setminus m(L)$, both arcs are left unchanged by r, i.e., e_H and $e_G = e_H^{-1}$ are in H.
   (b) Otherwise, there exists e' in L such that $m(e') = e_G$. Because G is a combinatorial graph and e_H is in $G \setminus m(L)$, e' is oriented in L. Since r satisfies $\mathscr{O}_r(i)$, e' is in $L \cap R$. Therefore, $m'(e') = e_G = e_H^{-1}$ in H, by injectivity of m'.
2. Otherwise, $e_H \in m'(R)$ and, by injectivity of m', there exists a unique $e \in R$ such that $m'(e) = e_H$.
   (a) If e admits a reverse arc e' in R, then $m'(e')$ is the reverse arc of e_H in H.
   (b) Otherwise, similarly to case 1b, e is an oriented arc of $L \cap R$. The arc e cannot have a reverse arc in L, as this arc would be in $E_L \setminus E_R$ and would break the non-orientation condition on r. Since e is in L, $m(e) = e_H$. The non-orientation constraint in G yields an arc e_G reverse of e_H in G. This arc has no antecedent in L and belongs to $G \setminus m(L)$. Therefore e_G is in H where it is the reverse arc of e_H.

Thereafter H satisfies the non-orientation constraint $\mathscr{O}_H(i)$.

($\Leftarrow$). Suppose for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{O}_G(i)$, the graph H, result of the direct derivation $G \Rightarrow^{r,m} H$, satisfies the non-orientation constraint $\mathscr{O}_H(i)$.

Let m' be the morphism $R \hookrightarrow H$, which is a mono by construction of DPO rewriting and consider an added arc e of $E_R \setminus E_L$. Then, there exists e_H in H such that $m'(e) = e_H$.

Because H satisfies $\mathscr{O}_H(i)$, e_H has a reverse arc e'_H in H. Since e is not in L, e_H is not in G. Because G satisfies $\mathscr{O}_G(i)$, e'_H is not in G (otherwise e_H would also be in G). Thus, e'_H is an added arc in H. By injectivity of m' there exists a unique i-arcs e' in R such that $m'(e') = e'_H$. Then e and e' are reversed arcs of $E_R \setminus E_L$ in R. The proof holds by reversing the roles of H and G, as well as R and L.

Thereafter r satisfies the non-orientation condition $\mathscr{O}_r(i)$. □

Example 8 (Non-Orientation Preservation). The rule from Figure 14 also satisfies the non-orientation condition for the dimension 3: there is no deletion (in $L \setminus R$) of a 3-arc without its reverse, the 3-arc added (in $R \setminus L$) is a loop. As we can see, the only oriented 3-arc in the rule belongs to the interface $L \cap R$. The graph G in the bottom left corner of Figure 15 satisfies the non-orientation constraint for the dimension 3. Thus, the direct derivation yields a graph satisfying the non-orientation constraint for the dimension 3.

5.2. Dealing with the cycle constraint

Finally, the cycle constraint needs a comparable condition on transformation rules to acknowledge the rules that guarantee consistency. The main issue when dealing with the cycle constraint is the description of neighboring elements. The preservation of the first two topological constraints considered arcs always incident to matched nodes. Since rules may only partially match cycles, we discuss constructions and properties that will help us define a necessary and sufficient condition to preserve the cycle constraint.

Recall from Lemma 1 that the order of the dimensions for the cycle constraints does not matter in a combinatorial graph when considering the constraint on the whole graph.

5.2.1. Rule completion

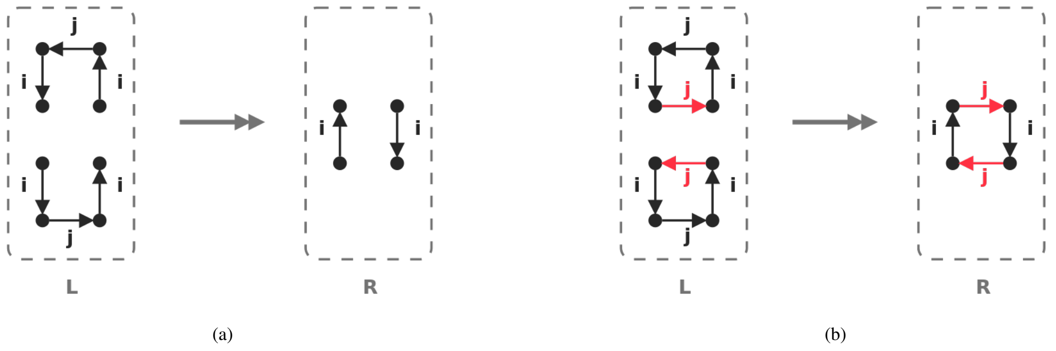


Figure 16: ij-completion of a rule: implicit arcs in an $ijij$-cycle are added.

First, since the cycle constraint characterizes cycles of length 4, if there is an iji-path or a jij-path in L, we can add the missing arc to make a cycle. In other words, if there is an iji-path p in the left-hand side L of a rule r and we apply r to a graph G that satisfies $\mathscr{C}_G(i,j)$ via the match morphism $m : L \hookrightarrow G$ then there is a j-arc of source $m(t(p))$ and target $m(s(p))$ in G. If not already in the rule interface, this j-arc can be added to $L \cap R$ without modifying the result of the direct transformation $G \Rightarrow^{r,m} H$. For instance, we can add the missing j-arcs to the rule of Figure 16a and get the rule of Figure 16b. The application of either rule results in the same graph. However, it is easier to use the latter construction because it clarifies the cycles in the rule. Consequentially, when considering the cycle constraint in a rule, we first impose the completion of every iji-path and jij-path (in L and R). We call this operation the ij-completion of the rule and assume that every rule is ij-completed.

Assumption 4. Every rule is ij-completed.

5.2.2. Alternating paths

Given a rule r and a match $m : L \hookrightarrow G$, an $ijij$-cycle in G is split into (possibly empty) elements of $G \setminus m(L)$, $m(L \cap R)$, and $m(L \setminus R)$. The elements of $G \setminus m(L)$ are not matched by the rule and are left unchanged by the rule. The elements of $m(L \cap R)$ belong to the rule interface and are not modified. Therefore, the only part that raises concern consists of the elements in $m(L \setminus R)$. If we guarantee that for each path in $m(L \setminus R)$ coming from the cycle, a similar path exists in $m'(R \setminus L)$, by concatenation with elements of $m'(R \cap L)$ and elements of $H \setminus m'(R)$, we reconstruct a cycle in H.

Example 9 (Intuition for Cycle Preservation). Figure 17, where all arcs are considered non-orientated, illustrates a rule that deletes a 3-arc between nodes c and e, and essentially exchanges the roles of c and e between the top 1212-cycle and the bottom cycle. In the right-hand side of the rule, nodes a, b, d and e belongs to a cycle. The question is whether the images of c, f, and h also belong to a cycle in H. As we can see, there is a 12-path from f to h in both L and R. Thus, the image of the path in G belongs to a cycle in G. Therefore, the unmatched part can be concatenated with the image of the path from R. As a result, we get back the cycle in H. This example provides intuition to preserve cycles with rules, explained more thoroughly in this section.

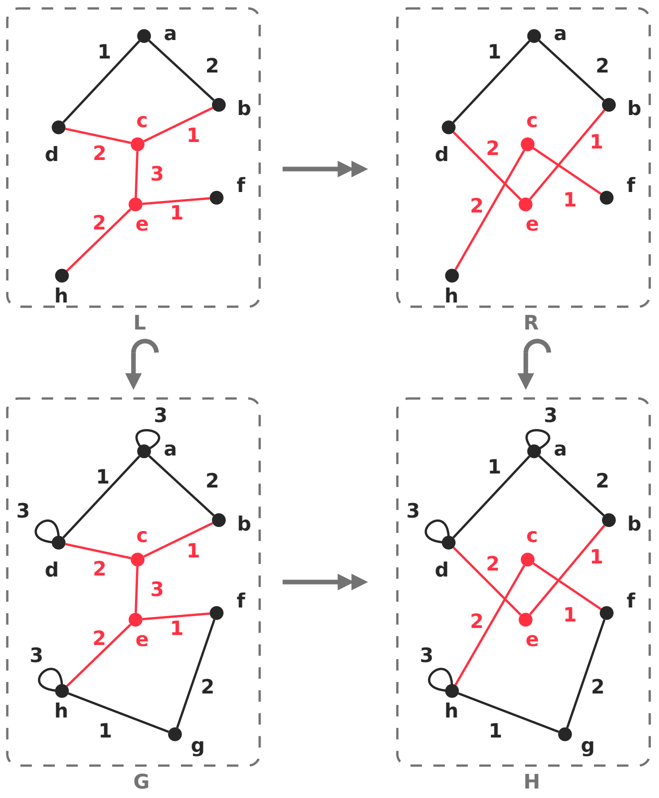


Figure 17: A rule preserving the cycle constraint for the dimensions 1 and 2.

Formally, elements in L that can be matched to an $ijij$-cycle in G are paths labeled by a word on the alphabet $\{i, j\}$ such that two consecutive letters are distinct. We call (i, j)-*alternating* such paths. We want to ensure that if L contains an (i, j)-alternating path with all arcs in $E_L \setminus E_R$, R also has an (i, j)-alternating path with the same source, target, and label. What we want to consider are only the longest paths. Since L may contain cycles with all arcs in $E_L \setminus E_R$, we need to enforce non-overlapping conditions to consider maximal elements. We say a path *overlaps* if it contains an arc twice (note that it may include a node several times). We will try to preserve paths to guarantee the preservation of the cycle constraint on combinatorial graphs. We call optimal paths such paths.

Definition 14 (Optimal Path)**.** *Let $r = L \twoheadrightarrow R$ be a $\mathbb{D}$-combinatorial rule. We say the path $p = v_s \rightsquigarrow v_t$ is (i,j)-optimal in L (resp. in R) if it satisfies the following properties:*

- *The path p is an (i,j)-alternating path (i.e., path labeled by a word on the alphabet $\{i,j\}$ such that two consecutive letters are distinct).*
- *All arcs of p are deleted arcs of $E_L \setminus E_R$ (resp. added arcs of $E_R \setminus E_L$).*
- *The path p does not overlap.*
- *The path p is maximal (with repect to the inclusion of paths) between paths that satisfy the first* 3 *properties.*
- *The path p does not belong to an $ijij$-cycle in L (resp. in R).*

Example 10 (Optimal Path). Consider the left-hand side of the rule from Figure 17. The 1212-cycle *adcb* forms a $(1,2)$-alternating path. This path is non-overlapping and maximal. However, it is not optimal as it contains interface arcs (the arc between *a* and *d* and the one between *b* and *a*). Restricted to the 21-path *dcb*, it is still not optimal as it is part of an $ijij$-cycle. The 132-path *bceh* is labeled with three distinct dimensions. Therefore it is not an alternating path, and a fortiori not an optimal path. Conversely, the 131-path *fecb* is a maximal non-overlapping $(1,3)$-alternating path. As it contains only deleted arcs and is not part of a 1313-cycle, the path is $(1,3)$-optimal.

Note that any optimal path satisfies the following straightforward property: if $p = v_s \rightsquigarrow v_t$ is an (i,j)-optimal path in a $\mathbb{D}$-combinatorial rule $r = L \twoheadrightarrow R$, then v_s and v_t are distinct interface nodes of $L \cap R$.

Consider again the 1212-cycle of source f in the graph G from Figure 17. The cycle can be divided into an optimal path from f to h (the red 12-path going through e) and a path from h to f in $G \setminus m(L)$ (the black 12-path going through g). For any optimal path that is not a cycle in L, we impose a path with the same source, target, and label in R and call coherent such paths.

Definition 15 (Coherence)**.** *Let $r = L \twoheadrightarrow R$ be a $\mathbb{D}$-combinatorial rule. An optimal path in L (resp. in R) is* coherent *with an optimal path in R (resp. in L) if they share the same source, target, and label. An optimal path in r is* coherent *if it admits a unique, coherent optimal path in the other side of the rule.*

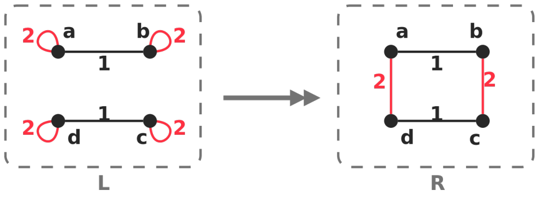
To guarantee that a combinatorial rule preserves the cycle constraint, we impose that every optimal path is coherent.

Definition 16 (Cycle Condition)**.** *A $\mathbb{D}$-combinatorial rule $r = L \twoheadrightarrow R$ satisfies the* cycle condition $\mathscr{C}_r(i,j)$ *if it satisfies the following sub-conditions:*

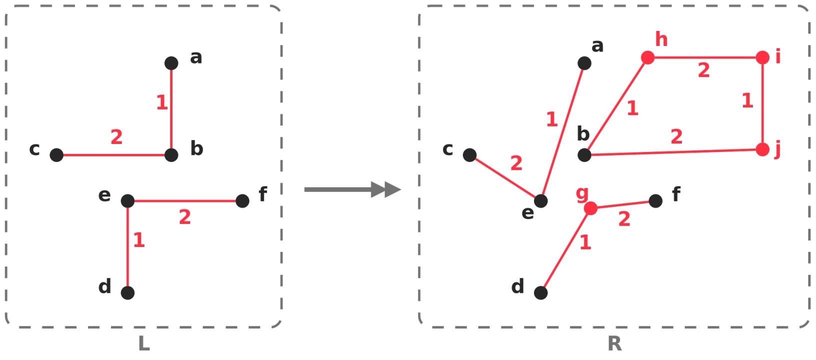
1. *Any (i,j)-optimal path in r is coherent.*
2. *Any preserved node of $L \cap R$ that is the source of an i-arc (resp. j-arc) in $E_L \setminus E_R$ is the source of an i-arc (resp. j-arc) in R that either belongs to an $ijij$-cycle or to an (i,j)-optimal path.*
3. *Any added node of $V_R \setminus V_L$ is the source of an i-arc (resp. j-arc) in R that either belongs to an $ijij$-cycle or to an (i,j)-optimal path.*

Note that, in a $\mathbb{D}$-combinatorial rule r that satisfies the cycle condition $\mathscr{C}_r(i,j)$, there can be no (i,j)-optimal path that consists of a single arc. Indeed, such an arc would have to be a coherent optimal path and therefore in $L \cap R$, which contradicts the optimality of the path.

Example 11 (Cycle Condition). One can easily verify that the rule from Figure 17 satisfies the cycle condition of Definition 16. We provide more examples in Figure 18, where arcs are still considered non-oriented, interface



(a)



(b)

Figure 18: Rules that satisfies the cycle condition $\mathscr{C}_r(1,2)$.

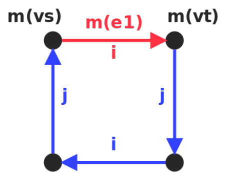
elements are drawn in black, and added or deleted elements are drawn in red. For instance, the rule of Figure 18a transforms two 1212-cycles, where the 2-arcs are loops, into a single 1212-cycle on four nodes. In L, each node is the source of an $ijij$-cycle, L admits no $(1,2)$-optimal path. Note that R does not add any $(1,2)$-optimal path. All nodes have their incident 2-arcs in L deleted but are the source of a 2-arc that belongs to a cycle in R. In the rule of Figure 18b, L consists of two $(1,2)$-optimal paths. The path abc is coherent in L as there is also a $(1,2)$-optimal path between a and c in R. Similarly, the path def is coherent in L. All nodes of L are interface nodes with deleted 1-arcs and 2-arcs. Nodes a, c, d e, and f are the source of arcs that belong to an optimal path. Node b is the source of a 1-arc and a 2-arc that belong to a 1212-cycle in R. Likewise, the added nodes h, i, and j are the sources of a 1-arc and a 2-arc that belong to a 1212-cycle in R. Finally, an added node can also be the source of an arc that belongs to optimal paths, such as node g.

5.3. *Preserving the cycle constraint*

Before detailing the theorem for cycle preservation, we will provide a preliminary result on the size of optimal paths in a coherent rule.

Lemma 2. *Let $r = L \twoheadrightarrow R$ be an ij-completed rule such that for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{C}_G(i,j)$, the resulting graph H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the cycle constraint $\mathscr{C}_H(i,j)$. Then any (i,j)-optimal path in L is of size 2.*

Proof. Suppose an (i,j)-optimal path exists in L with elements in $L \setminus R$ of size 1. Without loss of generality, we can suppose there is an i-arc e_1 of source v_s and target v_t in L such that v_s is the target of no j-arc and v_t is the source of no j-arc. When matched to G, the $ijij$-cycle contains $m(e_1)$ and an (i,j)-alternating path $p = m(v_t) \overset{jij}{\rightsquigarrow} m(v_s)$ with all arcs in $G \setminus m(L)$. Below, the arc in $m(L \setminus R)$ is displayed in red whereas arcs in $G \setminus m(L)$ are drawn in blue.



Therefore p is also in H. Since H satisfies the cycle constraint $\mathscr{C}_H(i,j)$, p can be extended to an $ijij$-cycle with an i-arc of source $m'(v_s)$ and target $m'(v_t)$. As r is a $\mathbb{D}$-combinatorial rule and H is a $\mathbb{D}$-combinatorial graph, this i-arc is in $m'(R)$, which contradicts that e_1 is in $L \setminus R$.

Furthermore, an (i,j)-optimal path can not be of size greater than 4 as there exists no (i,j)-alternating path of size greater than 4 without duplicate arcs in a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{C}_G(i,j)$. The ij-completion of the rule prevents it from being of size 3. Finally, the definition of optimality prevents it from being of size 4.

Therefore, any optimal path is of size 2. □

Exploiting this characterization, we show that the cycle condition on a combinatorial rule is necessary and sufficient to guarantee that its application on a combinatorial graph satisfying the cycle constraint results in a graph that also satisfies the cycle constraint.

Theorem 3 (Cycle Preservation)**.** *Let $r = L \twoheadrightarrow R$ be a $\mathbb{D}$-combinatorial rule.*

The rule r satisfies the cycle condition $\mathscr{C}_r(i,j)$ if and only if for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{C}_G(i,j)$, the graph H, result of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the cycle constraint $\mathscr{C}_H(i,j)$.

Proof. Given a match $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{C}_G(i,j)$, let m' denote the morphism $R \hookrightarrow H$, which is a mono by construction.

($\Rightarrow$). Suppose r satisfies the cycle condition $\mathscr{C}_r(i,j)$. Consider $m : L \hookrightarrow G$ a match on a $\mathbb{D}$-combinatorial graph G satisfying $\mathscr{C}_G(i,j)$, and H the result of the direct derivation $G \Rightarrow^{r,m} H$. Let v_H be a node of H.

1. Suppose that $v_H \in H \setminus m'(R)$, then v_H is a node of G. Thus, v_H is the source of an $ijij$-cycle in G since G satisfies $\mathscr{C}_G(i,j)$.
   (a) If no arc of this cycle is in $m(L \setminus R)$, then this cycle is left unchanged by the rule and v_H is the source of the same $ijij$-cycle in H.
   (b) Otherwise, certain parts of the cycle are in $m(L \setminus R)$. Because v_H is in $G \setminus m(L)$, two nodes $m(v_s)$ and $m(v_t)$ satisfies :
      - Nodes $m(v_s)$ and $m(v_t)$ belong to the cycle.
      - Nodes $m(v_s)$ and $m(v_t)$ are in $m(L \cap R)$.
      - All arcs from the cycle in $p = m(v_s) \overset{w}{\rightsquigarrow} m(v_t)$ are in $G \setminus m(L)$.
      - Node v_H belongs to p.

      Since v_H is in $G \setminus m(L)$, its incident arcs cannot be in $m(L)$ and p is a path, the size of which is at least 2. If w' completes w to get a cycle, the remaining part of the cycle is a path $m(v_t) \overset{w'}{\rightsquigarrow} m(v_s)$, which is the image of an (i,j)-alternating path in L. This (i,j)-alternating path consists of interface arcs and optimal paths. Since any optimal path is coherent according to sub-condition 1, there is a similar path in $m'(R)$. Either way, their concatenation with $m'(v_s) \overset{w}{\rightsquigarrow} m'(v_t)$ is an $ijij$-cycle of source v_H in H.
2. Otherwise, v_H is in $m'(R)$. Denote v the node in R such that $m'(v) = v_H$.
   (a) Suppose v is a node of $L \cap R$ and v is not the source of an i-arc in L. Since v belongs to $L \cap R$, v_H is a node of G. As G satisfies $\mathscr{C}_G(i,j)$, v_H is the source of an $ijij$-cycle in G. From that point, the proof is similar to cases 1a and 1b depending on whether the cycle is partially matched.

(b) Otherwise, assume v is a node of $L \cap R$, but v is the source of an i-arc in L. Since r is a combinatorial rule, v is the source of an i-arc in R. Sub-condition 2 guarantees that this i-arc either belongs to an $ijij$-cycle or to an (i,j)-optimal path in R. In the former case, the image by m' of the cycle yields an $ijij$-cycle of source v_H in H. In the latter, let $p = v_s \overset{w}{\rightsquigarrow} v_t$ be the (i,j)-optimal path in R. By sub-condition 1, p is coherent and there exists an (i,j)-optimal path $p' = v_s \overset{w}{\rightsquigarrow} v_t$ in L. Without loss of generality, assume that the first arc of p' is labeled by i. Since G satisfies $\mathscr{C}_G(i,j)$, $m(v_s)$ is the source of an $ijij$-cycle in G and the first part of this cycle is $m(p')$. The remaining arcs of the cycle can be subdivided into elements of $G \setminus m(L)$ left unchanged by the rule, interface arcs of $m(L \cap R)$ preserved by the rule, and (i,j)-optimal paths. Any such optimal path is coherent from sub-condition 1 and yields a similar optimal path in R. Their concatenation with $m'(p)$ is a cycle. When considering it with v_H instead of $m(v_s)$, it is an $ijij$-cycle of source v_H in H.

(c) If v is a node of $R \setminus L$, the proof is similar to case 2b, exploiting sub-condition 3 instead of sub-condition 2.

Thereafter H satisfies the cycle constraint $\mathscr{C}_H(i,j)$.

($\Leftarrow$). Suppose for all matches $m : L \hookrightarrow G$ on a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{C}_G(i,j)$, the result H of the direct derivation $G \Rightarrow^{r,m} H$ satisfies the cycle constraint $\mathscr{C}_H(i,j)$.

Sub-condition 1 of the cycle condition

Suppose there is an (i,j)-optimal path $v_s \overset{w}{\rightsquigarrow} v_t$ in L. By optimality, $v_s \neq v_t$. According to Lemma 2, the path is of size 2. The path $m(v_s \overset{w}{\rightsquigarrow} v_t)$ belongs to an $ijij$-cycle in G because G satisfies $\mathscr{C}_G(i,j)$. The two arcs that define the path $m(v_t) \overset{w}{\rightsquigarrow} m(v_s)$ are in $G \setminus m(L \setminus R)$. Thus, they are in H, where they belong to an $ijij$-cycle. The incident arcs condition guarantees that the path $m'(v_t) \overset{w}{\rightsquigarrow} m'(v_s)$ comes entirely from $R \setminus L$ and is a maximal non-overlapping path. Since the path contains two arcs and have distinct endpoints, it cannot overlap. Therefore it is the (i,j)-optimal path in R coherent with $v_s \overset{w}{\rightsquigarrow} v_t$.

Suppose there is an (i,j)-optimal path $p = v_s \overset{w}{\rightsquigarrow} v_t$ in R and consider the cycle completion of the path in H. It yields elements in G that belong to a cycle with arcs or vertices from $m(L)$. Elements on this alternating path are in $m(L \setminus R)$ by hypothesis on p. According to Lemma 2, the path is of size 2. The incident arcs constraint on G ensures that it is an optimal path coherent with p.

Sub-condition 2 of the cycle condition

Consider a preserved node v of $L \cap R$ that is the source of an i-arc in $L \setminus R$. Because r is a combinatorial rule, sub-condition 1 of the weak incident arcs condition $\mathscr{I}_G(i)$ guarantees that v is also the source i-arc in $R \setminus L$ (this arc can not be in $L \cap R$ because v is already the source of an i-arc in $L \setminus R$).

Let v_H be the image of v by m', i.e., $v_H = m'(v)$. Since H satisfies $\mathscr{C}_H(i,j)$, v_H is the source of an $ijij$-cycle in H. Depending on whether the whole cycle is present in R, v is the source of an i-arc in R that belongs either to an $ijij$-cycle or to an (i,j)-optimal path. The proof holds for a j-arc.

Sub-condition 3 of the cycle condition

Consider an added node v of $R \setminus L$. Because r is a combinatorial rule, sub-condition 2 of the weak incident arcs condition $\mathscr{I}_G(i)$ guarantees that v is the source i-arc in R. Similar to the proof of sub-condition 2, this arc belongs either to an $ijij$-cycle or to an (i,j)-optimal path. The proof holds for a j-arc.

Thereafter r satisfies the cycle condition $\mathscr{C}_r(i,j)$. □

Theorem 3 is stronger than the theorems presented in previous work for the preservation of the cycle constraint. We also want to point out that Theorems 1, 2, and 3 express consistency preservation as simple syntactic conditions on the rule. Intuitively, such conditions are possible because of our specific framework, namely $\mathbb{D}$-combinatorial graphs, which is highly regular.

6. Rule schemes using product

As we have seen in Section 2.3, products offer a simple way to apply transformations, such as arc deletion or arc relabeling, on a global scale. Taking benefit from our specific graphs, we will use the product as a mechanism to generalize transformations of $\mathbb{D}$-combinatorial graphs in the context of the DPO approach.

Our product-based generalization of DPO rewriting will allow us to define a large family of operations on combinatorial graphs taking advantage of their regularity with respect to the dimensions of $\mathbb{D}$. In Section 6.1, we provide more details about pattern graphs introduced in Section 3.1. Pattern graphs carry words of dimensions as labels on their arcs and enable us to select subgraphs to be modified in the combinatorial graphs. In Section 6.2, we define *graph scheme* as a way to encompass a transformation to be applied globally on a graph. We call *instantiation of a graph scheme* the operation (relying on a graph product) that modifies a pattern graph based on a graph scheme. We also define *rule schemes* as spans in the category of graph schemes. Similarly, we can instantiate a rule scheme by instantiating each of its graph schemes with the same pattern graph. Since we can instantiate a rule scheme with any pattern graph (assuming the right set of labels), rule schemes are rule abstractions, encoding infinitely many possible rewriting. Finally, in Section 6.3, we will discuss the mechanisms involved in the application of rule schemes.

6.1. Global extraction of paths

In a $\mathbb{D}$-combinatorial graph, every node is the source of a unique arc for each dimension in $\mathbb{D}$. Therefore, given a word w on $\mathbb{D}$, every node is the source of a unique w-path. In the relabeling operation defined in Section 2.3, we used pairs of the form (i, j) to rename a label i into j. Here, we will consider, more generally, pairs of the form (w, i) to create an i-labeled arc between the source and the target of a w-path.

The uniqueness of source and target per dimension makes it possible to build paths by following arcs forward or backward. Thus, for a dimension i, we introduce the notation $\overline{i}$ to indicate that, from a given node v, the next node in the path is the unique node v', source of the i-arc whose target is v. We accordingly extend the labeling alphabet of the graphs:

Definition 17 (Conjugate Dimensions)**.** *For any $d \in \mathbb{D}$, we define the* conjugate dimension $\overline{d}$ *which yields the* conjugate alphabet $\overline{\mathbb{D}} = \mathbb{D} \cup \{\overline{d} \mid d \in \mathbb{D}\}$.

The conjugation is extended to words on $\overline{\mathbb{D}}^*$ such that the conjugate of $w_{(1)} \dots w_{(n)}$ is $\overline{w_{(n)}} \dots \overline{w_{(1)}}$.

Conjugation allows us to broaden the definition of a path in the case of $\mathbb{D}$-combinatorial graphs. For a $\mathbb{D}$-combinatorial graph $G = (V, E, s, t, l)$, we introduce the conjugate $\overline{\mathbb{D}}$-combinatorial graph $\overline{G} = (V, E \cup \overline{E}, \overline{s}, \overline{t}, \overline{l})$, with $\overline{E} = \{\overline{e} \mid e \in E\}$. The functions $\overline{s}$, $\overline{t}$ and $\overline{l}$ on arcs in E are the corresponding functions of G, i.e., for $e \in E$, $\overline{s}(e) = s(e)$, $\overline{t}(e) = t(e)$ and $\overline{l}(e) = l(e)$. On arcs in $\overline{E}$, the functions $\overline{s}$, $\overline{t}$ and $\overline{l}$ rely on the inversion of the underlying arcs, i.e., for $e \in E$, $\overline{s}(\overline{e}) = t(e)$, $\overline{t}(\overline{e}) = s(e)$ and $\overline{l}(\overline{e}) = \overline{l(e)}$. From now on, for w in $\overline{\mathbb{D}}^*$, w-paths of a $\mathbb{D}$-combinatorial graph will implicitly refer to the w-paths defined on its conjugate version. In short, such w-paths are well-defined, and their target always exists because a node in G is the source and target of a unique i-arc for each dimension i in $\mathbb{D}$:

Fact 2. The incident arcs constraint guarantees the existence and uniqueness (up to the starting node) of paths labeled by a sequence of conjugate dimensions.

In order to build rule schemes as an abstraction of rules, labels on the conjugate alphabet will be used to encode the reconnection between nodes in the graph that supports the instantiation. We extend the definition of pattern graph (from Section 3.1 to graphs labeled over subsets of $\overline{\mathbb{D}}^*$. Figure 19 provides an example of a $\{\varepsilon, 21, \overline{21}\}$-pattern graph where nodes a, b, c and d are the sources of a singe ε-arc, a single 21-arc and a single $\overline{21}$-arc.

Intuitively, labeling arcs with paths offers greater freedom to characterize modified parts of the graphs and create arcs from such complex paths. In general, any pattern graph can be used to instantiate a rule scheme. In

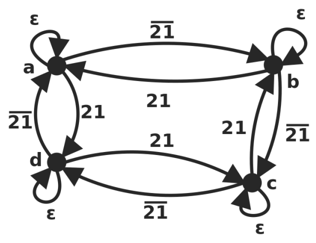


Figure 19: A pattern graph.

practice, pattern graphs will be extracted from the graph modified by the application of the rule scheme. We will discuss in Section 6.3 how to derive pattern graphs from a combinatorial graph. In the sequel, $\mathbb{W}$ is a finite set of words of $\overline{\mathbb{D}}^*$.

6.2. *Rule schemes*

Following the idea given in Section 2.3 of representing a graph modification with a product, we use a graph to represent the modification to be carried out on the pattern graph. Since pattern graphs belong to the category $\mathbb{W}$-**Graph** and we are trying to build objects from the category $\mathbb{D}$-**Graph**, the modification will rely on a function from $\mathbb{W}$ to $\mathbb{D}$. Therefore, the product is expressed in the category of graph scheme. The construction is similar to the relabeling operation, with the functors $\mathbb{E}_\Sigma$ and π_Σ replaced by:

- An embedding functor $\mathbb{E}_\mathbb{D} : \mathbb{W}$-**Graph** $\to (\mathbb{W}\times\mathbb{D})$-**Graph** that transforms the arcs labeled by a word w in $\mathbb{W}$ into a set of $|\mathbb{D}|$ arcs labeled (w,d), for all $d \in \mathbb{D}$.
- A projecting functor $\pi_\mathbb{D} : (\mathbb{W}\times\mathbb{D})$-**Graph** $\to \mathbb{D}$-**Graph** that only keeps the second part of the arc labels.

To ease certain proofs and constructions, we will also consider the projecting functor $\pi_\mathbb{W} : (\mathbb{W}\times\mathbb{D})$-**Graph** $\to \mathbb{W}$-**Graph** that only keeps the first part of the arc labels.

The instantiation $\iota(\Pi,P)$ of a $(\mathbb{W},\mathbb{D})$-graph scheme Π along a $\mathbb{W}$-pattern graph P is similar to the relabeling operation given in Section 2.3. The instantiation is achieved as follows :

1. Embed P in the category of $(\mathbb{W}\times\mathbb{D})$-graphs to get $\mathbb{E}_\mathbb{D}(P)$.
2. Construct the product $\mathbb{E}_\mathbb{D}(P)\times_{(\mathbb{W}\times\mathbb{D})}\Pi$.
3. Apply the projecting functor $\pi_\mathbb{D}$ to $\mathbb{E}_\mathbb{D}(P)\times_{(\mathbb{W}\times\mathbb{D})}\Pi$ and get $\iota(\Pi,P)$, the instantiation of Π along P :

$$\iota(\Pi,P) = \pi_\mathbb{D}(\mathbb{E}_\mathbb{D}(P)\times_{(\mathbb{W}\times\mathbb{D})}\Pi).$$

This construction is summarized by the following commutative diagram:

$$\begin{tikzcd}
\iota(\Pi,P) & \mathbb{E}_\mathbb{D}(P)\times\Pi \arrow[l, Leftarrow, "\pi_\mathbb{D}"'] \arrow[r] \arrow[d] & \Pi \arrow[d, "!_\Pi"] \\
P \arrow[r, Rightarrow, "\mathbb{E}_\mathbb{D}"] & \mathbb{E}_\mathbb{D}(P) \arrow[r, "!_{\mathbb{E}_\mathbb{D}(P)}"] & 1_{\mathbb{W}\times\mathbb{D}}
\end{tikzcd}$$

where the square is a pullback, and double arrows represent functors.

Example 12 (Instanciation of a Graph Scheme). Let us take back the pattern graph from Figure 19, which is displayed at the bottom left corner in Figure 20 (graph P). As depicted in Figure 20, the pattern graph is first embedded in the category $(\{21,\overline{21},\varepsilon\},\{1,2\})$-**Graph** to give the graph $\mathbb{E}_\mathbb{D}(P)$, placed just above. The graph scheme Π is built on the graph from Figure 2 and depicted on the right. Arc labels are extended as follows: the 2-loop is turned into a $(21,2)$-loop on x whereas the 1-arcs are replaced by $(\varepsilon,1)$-arcs. The product $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} \Pi$ of $\mathbb{E}_{\mathbb{D}}(P)$ and Π is shown in the center of the figure. Applying the projecting functor erases the first part of the labels and yields $\iota(\Pi,P) = \pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} \Pi)$, the top graph of Figure 20.

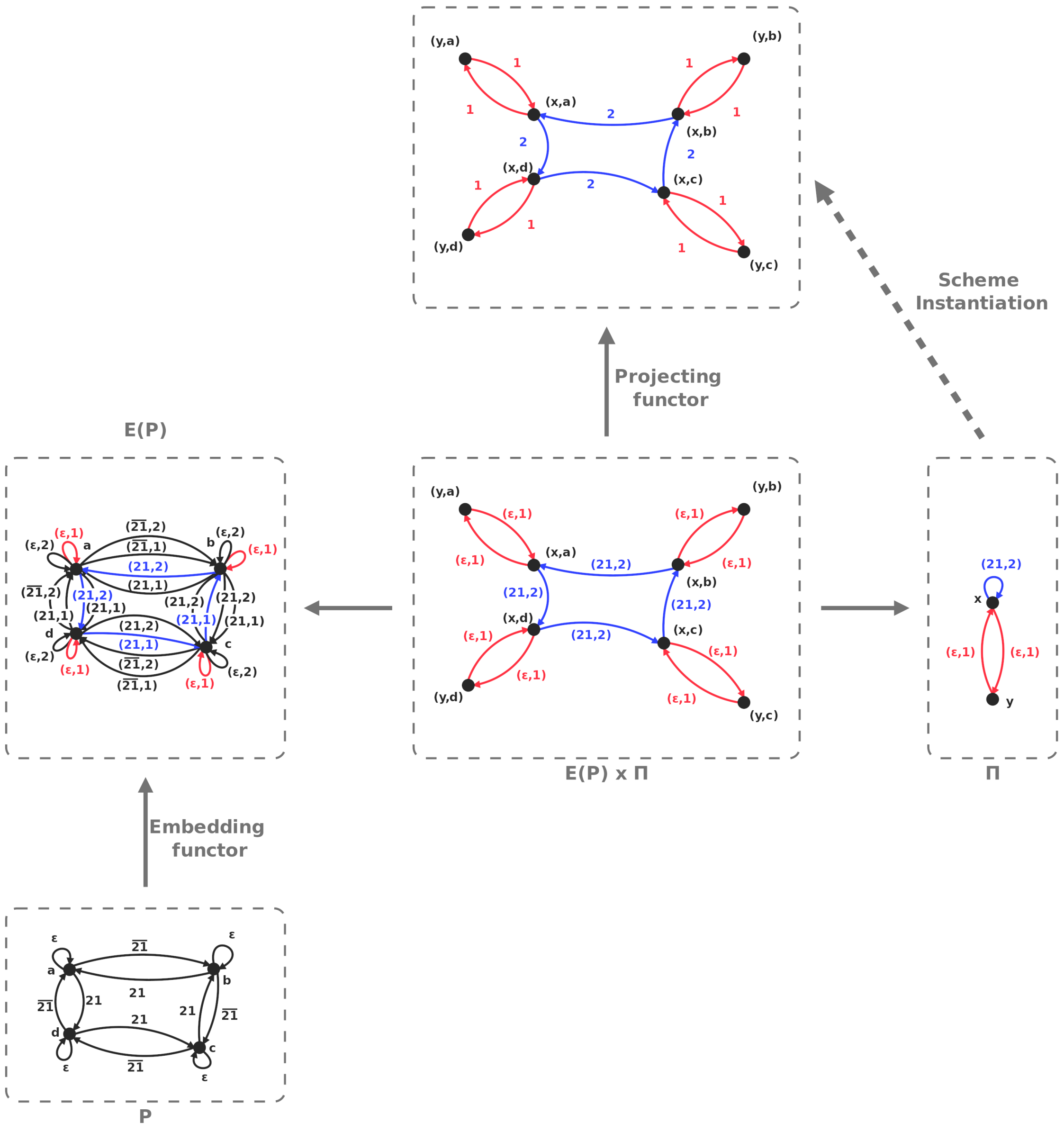
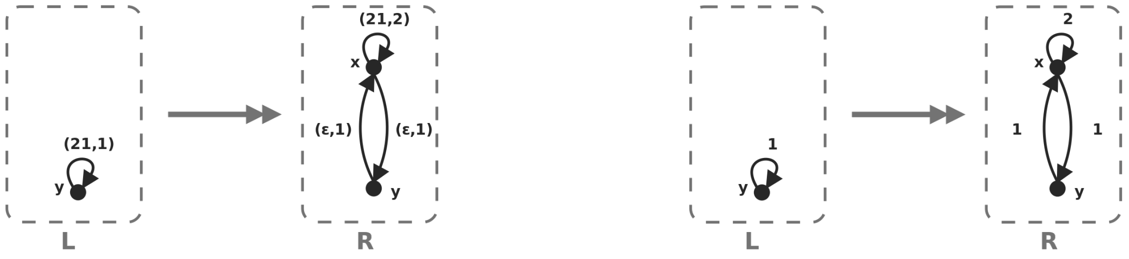


Figure 20: Instantiation of a graph scheme.

Definition 18 (Rule Scheme)**.** *A* $(\mathbb{W},\mathbb{D})$*-rule scheme* $\mathscr{S} = L \twoheadrightarrow R$*, is a rule in the category of* $(\mathbb{W},\mathbb{D})$*-graph schemes. The* core *of the rule scheme* $\mathscr{S}$ *is the projection of* $\mathscr{S}$ *in the category* $\mathbb{D}$*-**Graph**:*

$$\pi_{\mathbb{D}}(\mathscr{S}) = \pi_{\mathbb{D}}(L) \twoheadrightarrow \pi_{\mathbb{D}}(R).$$

Note that the core of a rule scheme is the rule built by deleting the labels part on $\mathbb{W}$. For instance, the rule of Figure 21b is the core of the rule scheme from Figure 21a. To apply a rule scheme to a graph, we instantiate each graph of the rule thanks to a product with the same pattern graph.



(a)

(b)

Figure 21: A $(\{21,\varepsilon\},\{1,2\})$-rule and its core.

Definition 19 (Rule Scheme Instantiation)**.** *Let $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$-rule scheme and P a $\mathbb{W}$-pattern graph. The* scheme instantiation $\iota(\mathscr{S},P)$ *of $\mathscr{S}$ on P is the rule $\iota(L,P) \twoheadrightarrow \iota(R,P)$.*

The definition of rule scheme instantiation is well-founded.

Proof. Since the pattern graph P and the rule schemes are defined on the same set of words $\mathbb{W}$, the products are well-defined. □

Example 13 (Instantiation of a Rule Scheme). Figure 22 shows the instantiation of the rule scheme from Figure 21a with the pattern graph of Figure 19. Note that the instantiation of the graph scheme in Figure 20 is the instantiation of the right-hand side of the rule and that we do not detail the instantiation of the left-hand side.

We now explain how the instantiation of a rule scheme can be used to obtain a rule that is applicable to a combinatorial graph.

6.3. Application of rule schemes

The application of a rule scheme $\mathscr{S}$ to a $\mathbb{D}$-combinatorial graph G is done in a series of steps. First, a pattern graph $P = (V_P, E_P, s_P, t_P, l_P)$ for $\mathscr{S}$ is built from G. This construction essentially relies on the replacement of G by its $\mathbb{W}$-transitive closure. The idea is to replace any path $v \rightsquigarrow v'$ in G labeled by a word w of $\mathbb{W}$ by an arc $e \in E_P$ such that $s_P(e) = v$, $t_P(e) = v'$ and $l_P(e) = w$. Formally, the construction is achieved using a functor.

Definition 20 (Pattern Functor)**.** *Let $\mathbb{D}$ be a finite set of dimensions and $\mathbb{W}$ a finite set of words in $\overline{\mathbb{D}}^*$.*

*The $\mathbb{W}$-pattern functor is the functor $\mathbb{P} : \mathbb{D}$-***Graph** *$\to \mathbb{W}$-***Graph** *from $\mathbb{D}$-combinatorial graphs to $\mathbb{W}$-pattern graphs defined as follows:*

- *For a $\mathbb{D}$-combinatorial graph G, the $\mathbb{W}$-pattern graph $\mathbb{P}(G)$ has the same nodes as G and, for $w \in \mathbb{W}$, a w-arc of source v_1 and target v_2 whenever there is a w-path $v_1 \overset{w}{\rightsquigarrow} v_2$ in G.*
- *For a $\mathbb{D}$-combinatorial graph morphism $m : G \to F$, the $\mathbb{W}$-graph morphism $\mathbb{P}(m) : \mathbb{P}(G) \to \mathbb{P}(F)$ has the same node morphism as m and an edge morphism that sends the unique w-arc of source v in $\mathbb{P}(G)$ to the unique w-arc of source v in $\mathbb{P}(F)$.*

Proof. Let G be a $\mathbb{D}$-combinatorial graph, then for all v in v_G and all $w \in \mathbb{W}$, there exists a unique path p of source v and labeled w in G. Therefore, the functor is well-defined on objects. Furthermore, the uniqueness of w-path in G yields uniqueness of w-arcs in $\mathbb{P}(G)$, for each node of G. Thus, the functor is well-defined on morphisms. □

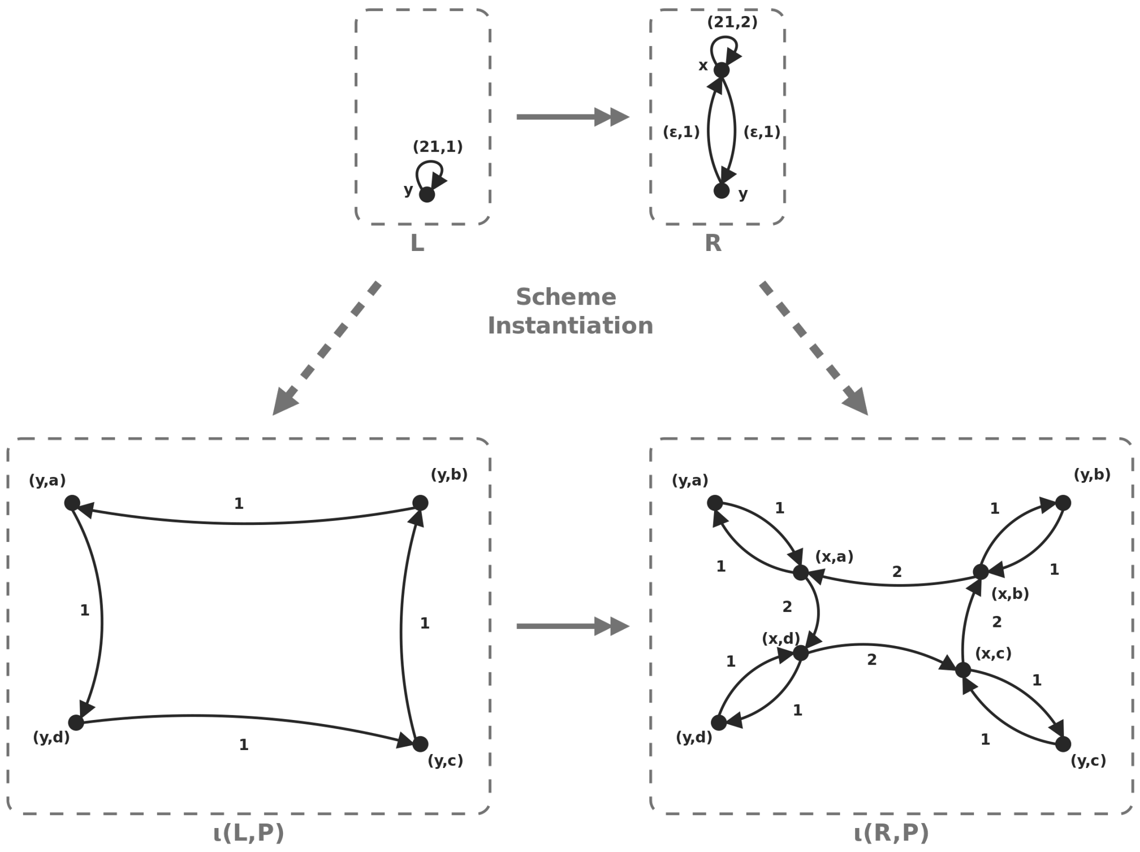


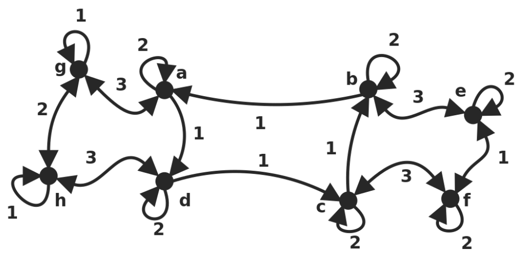
Figure 22: Instantiation of a rule scheme.

To be able to apply a rule scheme on a graph $\mathscr{G}$, we still need to ensure that the instantiation of the left-hand side of the rule scheme produces a graph that can be matched in $\mathscr{G}$. Therefore we will now make precise the two following points:

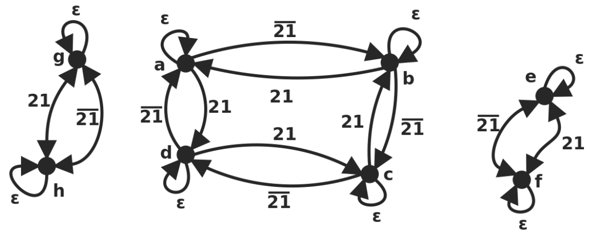
- A selection mechanism should specify what part of the $\mathbb{P}(\mathscr{G})$ will be used as the pattern graph.
- A method should specify how to construct the match from the instantiated rule.

Let us first deal with the selection mechanism and assume we want to apply the transformation to a topological cell in an object modeled by a graph $\mathscr{G}$, i.e., a maximal subgraph built using a subset of the labeling alphabet. We may not know all the nodes in the graph affected by the transformation, but we know for sure that the nodes corresponding to the cell are concerned. If we specify one node that must be in the transformed part of the graph, then the transformed part is the subgraph of $\mathbb{P}(\mathscr{G})$ containing the node and all nodes reachable to and from this node. Formally, let us define P_v (for a node v in $\mathscr{G}$) as the maximal subgraph of $\mathbb{P}(\mathscr{G})$ such that v is in P_v and, for all arcs e such that $s(e)$ or $t(e)$ is in P_v, then e, $s(e)$, and $t(e)$ are in P_v. For all nodes v in $\mathscr{G}$, there is a unique graph P_v and a unique inclusion $p_v : P_v \hookrightarrow \mathbb{P}(\mathscr{G})$.

Example 14 (Pattern Functor). Consider the combinatorial graph depicted in Figure 23a. The application of the $\mathbb{W}$-pattern functor for $\mathbb{W} = \{\varepsilon, 21, \overline{21}\}$ yields the $\mathbb{W}$-graph of Figure 23b. The 3-arcs are discarded. Each node is the source of an ε-loop, the 21 and $\overline{21}$-paths are turned into arcs. The application of the pattern functor results in three strongly connected components. The left one corresponds to P_g and P_h, the middle one to P_a, P_b, P_c and P_d, and the right one to P_e and P_f. Note that the middle one corresponds to the pattern graph of Figure 19.



(a)



(b)

Figure 23: A $\{1,2,3\}$-combinatorial graph (a) and its image by the $\{\varepsilon,21,\overline{21}\}$-pattern functor (b).

For each node v from $\mathscr{G}$, the pattern graph P_v is unique. Given a graph scheme L from a rule scheme $\mathscr{S}$, the product of the embedded pattern graph $\mathbb{E}_{\mathbb{D}}(P_v)$ and L is also unique. Therefore, given a node v from $\mathscr{G}$, the instantiation $\iota(L,P_v)$ is unique. However, node v from P_v may have been duplicated in the product construction. Indeed, node v yields as many nodes in the product as nodes in L. Nevertheless, if we identify a node v_L among the nodes of L that we call a *hook*, the product yields a unique node (v,v_L).

Let us assume that the maximal subgraph of $\iota(L,P_v)$ containing the node (v,v_L) is actually $\iota(L,P_v)$, i.e., $\iota(L,P_v)$ is connected. If there is a mono $\iota(L,P_v) \hookrightarrow \mathscr{G}$ mapping (v,v_L) to v, then the morphism is unique. This uniqueness comes from the incident arcs condition verified by $\mathscr{G}$. Indeed, each arc incident to (v,v_L) can only be sent to the unique arc incident to v with the same label. The mapping of the arcs incident to (v,v_L) provides a mapping of the nodes adjacent to (v,v_L). There is a unique way to propagate this mapping and we get the uniqueness on $m : \iota(L,P_v) \hookrightarrow \mathscr{G}$ such that $m((v,v_L)) = v$.

Example 15 (Construction of the Match). If we identify y as a hook in the left hand side of the rule scheme of Figure 22, then the node (y,a) is uniquely defined in the instantiated rule of the same figure. As we can see, there is a unique mono $\iota(L,P_a) \hookrightarrow \mathscr{G}$ (where $\mathscr{G}$ is the combinatorial graph of Figure 23a) that maps (y,a) to a. It is the mono that maps (y,a) to a, (y,b) to b, (y,c) to c, (y,d) to d and the arcs accordingly. The morphism is depicted by the grey dashed arrows in Figure 24.

Note that if $\iota(L,P_v)$ is not connected, there might not be uniqueness of the mono $\iota(L,P_v) \hookrightarrow \mathscr{G}$. For instance, if two distinct components of $\iota(L,P_v)$ are isomorphic, we can exchange their image and get another mono. This limit can be bypassed whenever the partition arises from L. In this case, if we identify one hook pair (v,v_L) per component of L such that all the corresponding pattern graph P_v's are isomorphic, we restore the uniqueness of the mono. Note that the construction of the pattern graph guarantees that it is connected. The embedding functor construction ensures every arc incident to each node of L is associated with a unique arc in $\mathbb{E}_{\mathbb{D}}(P_v)$. Therefore, the specification of hook pairs is sufficient to guarantee the uniqueness of match, should it exist. In this case, the local propagation exploiting the incident arcs property provides an algorithm to build the match.

As the instantiation mechanism is rather general, it can produce rules that may not be applicable in some cases. We could impose conditions on the labels used in the rule scheme, but we may unnecessarily restrict our framework expressiveness. Therefore, we believe it is up to the user to check on examples that any written rule corresponds to the desired operation.

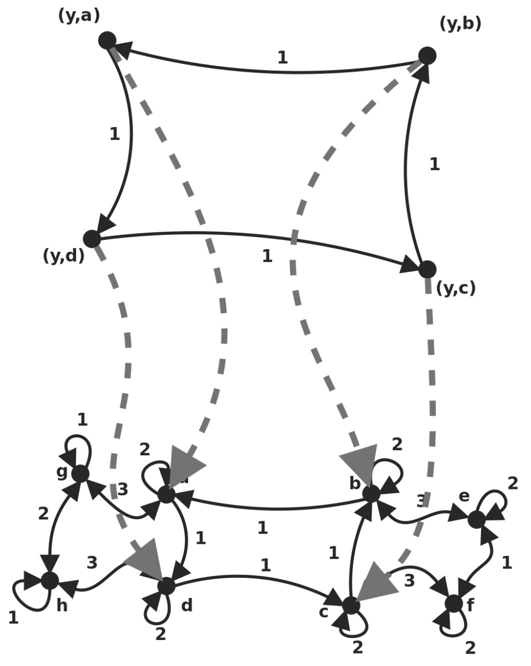


Figure 24: Mono from the instantiated left-hand side of the rule to the combinatorial graph.

We have discussed how we can apply a rule scheme to a combinatorial graph; let us now wrap up the whole process. To apply a $(\mathbb{W},\mathbb{D})$-rule scheme $\mathscr{S}$ to a $\mathbb{D}$-combinatorial graph $\mathscr{G}$, we first extracted a pattern graph P from $\mathscr{G}$. This pattern graph P is built in two steps: the application of the $\mathbb{W}$-pattern functor and the choice of inclusions $p_v : P_v \hookrightarrow \mathbb{P}(\mathscr{G})$ (i.e., the choice of a node v in $V_{\mathscr{G}}$ for each hook in L) to specify where the operation should be done. With this pattern graph, we instantiate the $(\mathbb{W},\mathbb{D})$-rule scheme to get a $\mathbb{D}$-rule. Thanks to the hook nodes in L, we construct the unique mono $m : \iota(\mathscr{S},P) \hookrightarrow \mathscr{G}$ and finally realize the direct derivation $\mathscr{G} \Rightarrow^{\iota(\mathscr{S},P),m} \mathscr{H}$. An example of the complete pipeline is given in Figure 25 where both the hook node in the rule scheme and the chosen node in the combinatorial graph are filled in green. The transformation in the graph is put forward by the red coloring.

The complete construction can be summarized by the following commutative diagram:

$$\begin{tikzcd}
1_{\mathbb{W}\times\mathbb{D}} & & & \\
E_{\mathbb{D}}(P_v) \arrow[u, "!_{E_{\mathbb{D}}(P_v)}"] & L^\bullet \arrow[ul, "!_{L^\bullet}"'] & K \arrow[l, hook'] \arrow[r, hook] \arrow[ull, "!_K"'] & R \arrow[ulll, "!_R"'] \\
P_v \arrow[u, Rightarrow, "E_{\mathbb{D}}"] \arrow[d, hook, "p_v"'] & E_{\mathbb{D}}(P_v)\times L^\bullet \arrow[u] \arrow[ul] \arrow[d, Rightarrow, "\pi_{\mathbb{D}}"] & E_{\mathbb{D}}(P_v)\times K \arrow[l, hook'] \arrow[r, hook] \arrow[u] \arrow[ull] \arrow[d, Rightarrow, "\pi_{\mathbb{D}}"] & E_{\mathbb{D}}(P_v)\times R \arrow[u] \arrow[ulll] \arrow[d, Rightarrow, "\pi_{\mathbb{D}}"] \\
\mathbb{P}(\mathscr{G}) & \iota(L^\bullet,P_v) \arrow[d, hook, "m"] & \iota(K,P_v) \arrow[l, hook'] \arrow[r, hook] \arrow[d, hook] & \iota(R,P_v) \arrow[d, hook] \\
& \mathscr{G} \arrow[ul, Rightarrow, "\mathbb{P}"] & \mathscr{D} \arrow[l, hook'] \arrow[r, hook] & \mathscr{H}
\end{tikzcd}$$

where $L^\bullet$ means that L has a hook per connected component. The top span $L^\bullet \hookleftarrow K \hookrightarrow R$ is the rule scheme. The second span $P_v \times L^\bullet \hookleftarrow P_v \times K \hookrightarrow P_v \times R$ is obtained by the product with P_v (a subgraph of the embedding

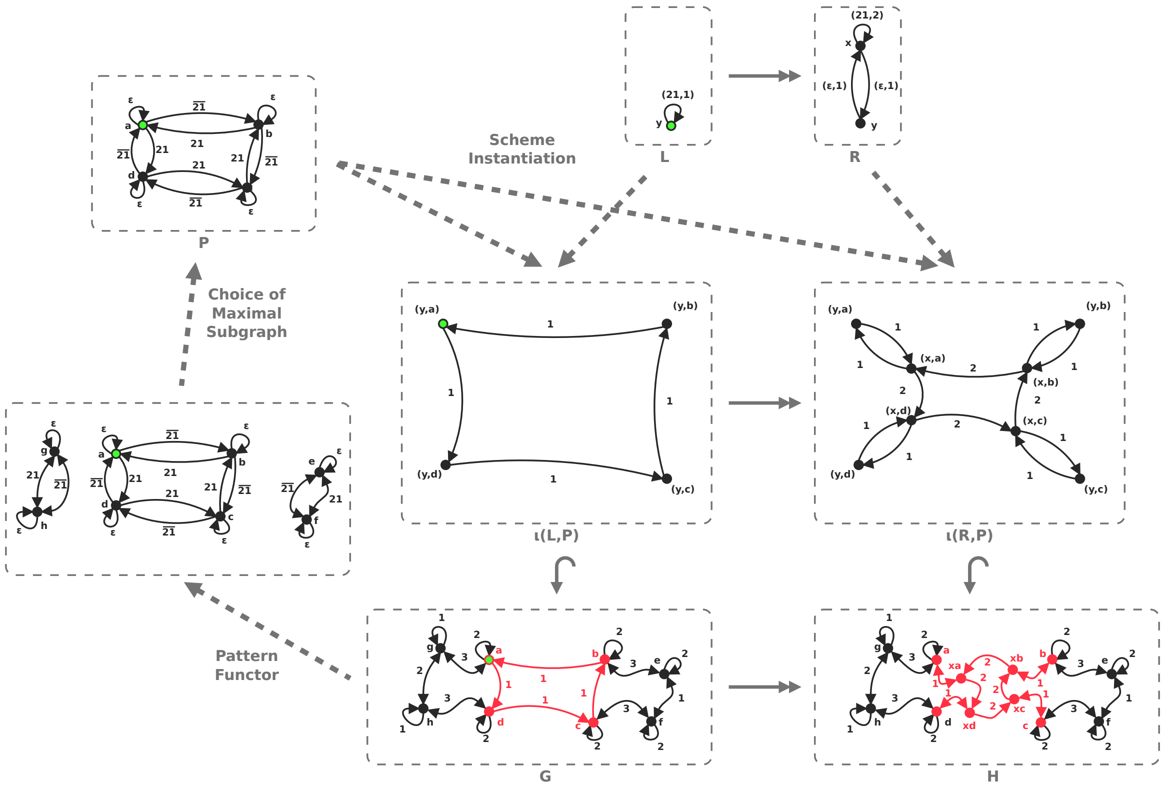


Figure 25: Complete process to transform a combinatorial graph with a rule scheme.

of $\mathscr{G}$). The third span is obtained with the projecting functor and yields the last span by standard DPO-rewriting.

As a final remark, let us point out that the set of words $\mathbb{W}$ has been given a priori for the rule scheme and the pattern graph. As of now, a $\mathbb{W}$ is just a superset of all words used to label a given $\mathbb{W}$-pattern graph or the $(\mathbb{W},\mathbb{D})$-graph schemes of a rule scheme. In the following three sections, we will impose conditions on $\mathbb{W}$ to lift the condition for the preservation of our three elementary constraints. Nonetheless, the choice of $\mathbb{W}$ inherently comes from the transformation that we want to describe through the rule scheme.

7. Incident arcs consistency in rule schemes

We now lift the incident arcs condition from $\mathbb{D}$-combinatorial rules to rule schemes. Since rule instantiations are essentially defined through a product, it is natural that each of the two members of the product should be subject to conditions to ensure that the resulting instantiation fulfills the intended incident arcs condition. Therefore, we will start by defining a first condition for the pattern graphs and a second one for the rule schemes. Afterward, we will show that, under such hypotheses, the instantiation of a consistent rule scheme via a consistent pattern graph yields combinatorial rules.

Definition 21 (Incident Arcs Constraint for Pattern Graphs)**.** *A $\mathbb{W}$-pattern graph $P = (V_P, E_P, s_P, t_P, l_P)$ satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$ if every node v in V_P is the source of a unique w-arc and the target of a unique w-arc for each word in $\mathbb{W}$.*

Note that when the pattern graph P is defined as a maximal subgraph of a $\mathbb{D}$-combinatorial graph G generated from a given node and the words of $\mathbb{W}$, P trivially satisfies the incident arcs constraint for pattern graphs $\mathscr{I}_P(\mathbb{W})$. For instance, the pattern graph from Section 6 (see Figure 19) satisfies the incident arcs constraint for pattern graph on the set of words $\{\varepsilon, 21, \overline{21}\}$. Let us point out that the incident arcs constraint on pattern graphs guarantees that when building a product using such a pattern graph, all arcs of a graph scheme have a corresponding arc for each node in the embedded pattern graph. Let us now introduce a counterpart condition for rule schemes:

Definition 22 (Incident Arcs Condition for Rule Schemes). *A $(\mathbb{W},\mathbb{D})$-rule scheme $\mathscr{S} = L \twoheadrightarrow R$ satisfies the incident arcs condition $\mathscr{I}_{\mathscr{S}}(i)$ for a dimension i in $\mathbb{D}$ if the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the incident arcs condition $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$. If the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ is a $\mathbb{D}$-combinatorial rule, $\mathscr{S}$ is said to be a $(\mathbb{W},\mathbb{D})$-combinatorial rule scheme.*

In other words, a $(\mathbb{W},\mathbb{D})$-rule scheme $\mathscr{S}$ satisfies $\mathscr{I}_{\mathscr{S}}(i)$ for a dimension i in $\mathbb{D}$ if :

1. Any preserved node of $\pi_{\mathbb{D}}(L\cap R)$ is the source (resp. target) of an i-arc in $\pi_{\mathbb{D}}(L)$ if and only if it is the source (resp. target) of an i-arc in $\pi_{\mathbb{D}}(R)$.
2. $\pi_{\mathbb{D}}(L)$ satisfies $\mathscr{I}_{(V_L\setminus V_R),\pi_{\mathbb{D}}(L)}(i)$.
3. $\pi_{\mathbb{D}}(R)$ satisfies $\mathscr{I}_{(V_R\setminus V_L),\pi_{\mathbb{D}}(R)}(i)$.

Let us recall that the first two sub-conditions concern the weak-incident arcs conditions, while the last one is equivalent to the dangling arc condition (also called gluing condition).

Example 16 (Incident Arcs Condition for Rule Schemes). The rule scheme from the previous section (see Figure 21a) satisfies $\mathscr{I}_{\mathscr{S}}(\{1,2\})$. The core of the rule scheme is given in Figure 21b. Node y belongs to the interface. It is the source of a 1-loop in L. It is also the source and target of an i-arc in R. Thus, the first sub-condition holds. There is no deleted nod. The added node x is the source of a 2-loop and the source and target of a 1-arc. Thus the second and the third sud-conditions also hold. In the end, the rule scheme is a $\{\varepsilon,21,\overline{21}\},\{1,2\})$-combinatorial rule scheme.

Theorem 4 (Lifting the Incident Arcs Condition to Rule Schemes). *Let i be a dimension in $\mathbb{D}$ and $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$–rule scheme.*

The rule scheme $\mathscr{S}$ satisfies incident arcs condition $\mathscr{I}_{\mathscr{S}}(i)$ if and only if for all $\mathbb{W}$-pattern graph P that satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S},P)$ satisfies the incident arcs condition $\mathscr{I}_{\iota(\mathscr{S},P)}(i)$.

Proof. Let $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$-rule scheme and $i \in \mathbb{D}$ be a dimension.

($\Rightarrow$). Assume that the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the incident arcs condition $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$. Consider a $\mathbb{W}$-pattern graph P that satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$.

Let $K = L\cap R$ be the interface of the rule scheme, p_k be the morphism $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} K \to \mathbb{E}_{\mathbb{D}}(P)$, p_r be the morphism $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} R \to \mathbb{E}_{\mathbb{D}}(P)$, k_p be the morphism $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} K \to K$, r_p be the morphism $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} R \to R$, such that the morphisms come from the following products:

$$\begin{tikzcd}
\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} K \arrow[d, "k_p"] \arrow[dr, "p_k"] & & \mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} R \arrow[dl, "p_r"'] \arrow[d, "r_p"] \\
K & \mathbb{E}_{\mathbb{D}}(P) & R
\end{tikzcd}$$

Sub-condition 1 of the weak incident arcs condition.

Let v be a preserved node of $\iota(K,P)$. Thus, v is a preserved node of $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} K$ and there are two nodes a in $\mathbb{E}_{\mathbb{D}}(P)$ and u in K such that $p_k(v) = a$ and $k_p(v) = u$. If v is the source of an i-arc in $\iota(L,P)$ then there is a word w in $\mathbb{W}$ such that v is the source of an arc labeled (w,i) in $\mathbb{E}_{\mathbb{D}}(P)\times_{(\mathbb{W}\times\mathbb{D})} L$. By construction of the product, there is an arc of source a in $\mathbb{E}_{\mathbb{D}}(P)$ and an arc of source u in L both labeled (w,i). Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, there is a word w' in $\mathbb{W}$ such that u is the source of an arc labeled (w',i) in

R. Since P satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$, the embedding functor $E_{\mathbb{W}}$ ensures the existence of an arc labeled (w',i) in $\mathbb{E}_{\mathbb{D}}(P)$ of source a. Therefore, v is the source of an arc labeled (w',i) in $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R$, transformed into an i-arc in $\iota(R,P)$ by the projecting functor.

The proof holds for the target of an i-arc in $\iota(L,P)$ and for the source or target of an i-arc in $\iota(R,P)$. Thereafter, any preserved node of $\iota(K,P)$ is the source (resp. target) of an i-arc in $\iota(L,P)$ if and only if it is the source (resp. target) of an i-arc in $\iota(R,P)$.

Sub-condition 2 of the weak incident arcs condition.

Let v be an added node of $\iota(R,P) \setminus \iota(K,P)$. Thus, v is a node of $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R \setminus \mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} K$ and there are two nodes a in $\mathbb{E}_{\mathbb{D}}(P)$ and u in $R \setminus K$ such that $p_r(v) = a$ and $r_p(v) = u$. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, u is the source of a unique i-arc in $\pi_{\mathbb{D}}(R)$. This i-arc yields an arc labeled (w,i), for a word in w in $\mathbb{W}$, that is the only arc in R of source u having i as the second part of its label. Since P satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$, the embedding functor $E_{\mathbb{W}}$ ensures the existence and uniqueness of a (w,i)-arc of source a. Therefore, there is an arc labeled (w,i) of source v in $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R$ and it is the only arc of source v having i as the second part of its label. This arc is transformed into an i-arc in $\iota(R,P)$ by the projecting functor. Thus, there is a unique i-arc of source v in $\iota(R,P)$.

The proof holds for an i-arc of target v. Thereafter, any added node of $\iota(R,P) \setminus \iota(K,P)$ is the source (resp. target) of a unique i-arc and $\iota(\mathscr{S},P)$ satisfies sub-condition 2 of the weak incident arcs condition.

Dangling condition.

The proof for a node deleted from $\iota(L,P) \setminus \iota(K,P)$ is the same as the proof for a node added in $\iota(R,P) \setminus \iota(K,P)$.

Thereafter, $\iota(\mathscr{S},P)$ satisfies the incident arcs condition $\mathscr{I}_{\iota(\mathscr{S},P)}(\mathbb{D})$.

($\Leftarrow$). Assume that for all $\mathbb{W}$-pattern graph P that satisfies the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S},P)$ satisfies the incident arcs condition $\mathscr{I}_{\iota(\mathscr{S},P)}(\mathbb{D})$.

In particular, consider the terminal graph $\mathbf{1}_{\mathbb{W}}$. Then $\iota(\mathscr{S},\mathbf{1}_{\mathbb{W}})$ is the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$. Thus $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies the incident arcs condition $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$.

Thereafter, the rule scheme $\mathscr{S}$ satisfies incident arcs condition $\mathscr{I}_{\mathscr{S}}(i)$. □

Provided that pattern graphs and rule schemes satisfy certain conditions, which can be verified by direct static analysis, then Theorem 4 states that all rules obtained by instantiation verify the incident arcs conditions.

Example 17 (Lifting the Incident Arcs Condition to Rule Schemes). The rule scheme from Figure 21 is a $\{\varepsilon, 21, \overline{21}\}, \{1,2\})$-combinatorial rule scheme. Its instantiation with the pattern graph of Figure 19 yields the rule of Figure 22, which satisfies the incident arcs condition for the dimensions 1 and 2.

Likewise, we will study constraints on pattern graphs and conditions on rule schemes for the non-orientation and cycle properties.

8. Non-orientation consistency in rule schemes

By analogy with the extension of the incident arcs condition, we study the condition of non-orientation. First, we give a constraint on pattern graphs and a condition on the rule schemes such that scheme instantiation produces a rule that satisfies the non-orientation condition.

Definition 23 (Non-Orientation Constraint for Pattern Graphs)**.** *A $\mathbb{W}$-pattern graph P satisfies the non-orientation constraint $\mathscr{O}_P(\mathbb{W})$ if for every w in $\mathbb{W}$, $\overline{w}$ is also in $\mathbb{W}$ and every w-arc in P admits a reverse $\overline{w}$-arc.*

If the pattern graph P is built on a $\mathbb{D}$-combinatorial graph G that satisfies the non-orientation constraint $\mathscr{O}_G(\mathbb{D})$, then P trivially satisfies the non-orientation constraint for pattern graphs $\mathscr{O}_P(\mathbb{W})$. For instance, the pattern graph from Section 6 (see Figure 19) satisfies the non-orientation constraint for pattern graph on the set of words $\{\varepsilon, 21, \overline{21}\}$. The non-orientation constraint on pattern graphs ensures that, for all words w in $\mathbb{W}$, arcs labeled w and $\overline{w}$ in the graph scheme yields arcs and reverse arcs in the product construction.

Definition 24 (Non-Orientation Condition for Rule Schemes)**.** *A $(\mathbb{W}, \mathbb{D})$-rule scheme $\mathscr{S} = L \twoheadrightarrow R$ satisfies the non-orientation condition $\mathscr{O}_{\mathscr{S}}(i)$ for a dimension i in $\mathbb{D}$ if :*

- *The core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the non-orientation condition $\mathscr{O}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$.*
- *For any arcs e and e' in $\mathscr{S}$ such that*[1] *$l_{\pi_{\mathbb{D}}(\mathscr{S})}(\pi_{\mathbb{D}}(e)) = l_{\pi_{\mathbb{D}}(\mathscr{S})}(\pi_{\mathbb{D}}(e')) = i$ and e is the reverse arc of e' in $\pi_{\mathbb{D}}(\mathscr{S})$, then $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e))$ is the conjugate of $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e'))$.*

Example 18 (Non-Orientation Condition for Rule Schemes). The rule scheme given in Section 6 in Figure 21a satisfies the non-orientation condition $\mathscr{O}_{\mathscr{S}}(1)$. Indeed, its core rule (see Figure 21b) contains a 1-loop in L and two reverse i-arcs in R. The 1-arcs in the right hand side of the core rule are $(1, \varepsilon)$-arcs in the rule scheme, thus the parts of the label on $\mathbb{W}$ are conjugate words.

Theorem 5 (Lifting the Non-orientation Condition to Rule Schemes)**.** *Let i be a dimension in $\mathbb{D}$ and $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W}, \mathbb{D})$-combinatorial rule scheme.*

The rule scheme $\mathscr{S}$ satisfies the non-orientation condition $\mathscr{O}_{\mathscr{S}}(i)$ if and only if for all $\mathbb{W}$-pattern graph P that satisfies both the incident arcs constraint $\mathscr{I}_P(\mathbb{W})$ and the non-orientation constraint $\mathscr{O}_P(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S}, P)$ satisfies the non-orientation condition $\mathscr{O}_{\iota(\mathscr{S},P)}(i)$.

Proof. Let i be a dimension $\mathbb{D}$ and $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W}, \mathbb{D})$-combinatorial rule scheme.

($\Rightarrow$). Assume that the rule scheme $\mathscr{S}$ satisfies the two sub-conditions and consider a $\mathbb{W}$-pattern graph P that satisfies $\mathscr{I}_P(\mathbb{W})$ and $\mathscr{O}_P(\mathbb{W})$.

Without loss of generality, let us consider the case of $\iota(L, P) \setminus \iota(R, P)$. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies $\mathscr{O}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$ and $\mathscr{I}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbb{D})$, $L \setminus R$ always contains an arc and its reverse. Theses arcs have conjugate words on the first part of their label. Because P satisfies $\mathscr{O}_P(\mathbb{W})$, the product pairs these arcs with reverse arcs of $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(P))$. Thus, the product creates reverse arcs in $\iota(L, P)$.

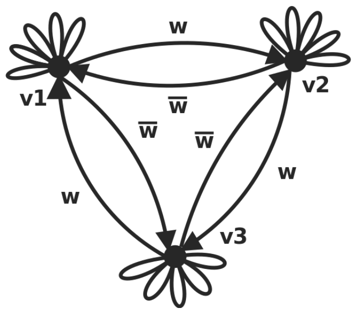
Thereafter, $\iota(\mathscr{S}, P)$ satisfies $\mathscr{O}_{\iota(\mathscr{S},P)}(i)$.

($\Leftarrow$). Assume that for all $\mathbb{W}$-pattern graph P that satisfies $\mathscr{I}_P(\mathbb{W})$ and $\mathscr{O}_P(\mathbb{W})$, the scheme instantiation $\iota(\mathscr{S}, P)$ satisfies the non-orientation condition $\mathscr{O}_{\iota(\mathscr{S},P)}(i)$. Similar to the proof of theorem 4, the terminal graph $\mathbf{1}_{\mathbb{W}}$ gives us the condition on the core of $\mathscr{S}$.

For the second condition, consider an i-arc e in $\mathscr{S}$ such that $\pi_{\mathbb{D}}(e)$ is an arc constrained by $\mathscr{O}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$. Note that since $\mathscr{S}$ is a $(\mathbb{W}, \mathbb{D})$-combinatorial rule scheme, the reverse arc of $\pi_{\mathbb{D}}(e)$ is unique (theorem 4). Let us assume that $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e)) = w$. Again, we build a scheme instantiation for $\mathscr{S}$ using a particular graph.

Intuitively, G_w is composed of 3 occurrences of $S_{\mathbb{W} \setminus \{w, \overline{w}\}}$ glued together with a www-cycle, like this :

[1]Where $l_{\pi_{\mathbb{D}}(\mathscr{S})}$ stands for $l_{\pi_{\mathbb{D}}(L)}$, $l_{\pi_{\mathbb{D}}(K)}$ or $l_{\pi_{\mathbb{D}}(R)}$ (resp. $l_{\pi_{\mathbb{W}}(\mathscr{S})}$ stands for $l_{\pi_{\mathbb{W}}(L)}$, $l_{\pi_{\mathbb{W}}(K)}$ or $l_{\pi_{\mathbb{W}}(R)}$) depending on whether $e \in L$, $e \in K$ or $e \in R$.



Formally[1], let $G_w = (V_w, E_w, s_w, t_w, l_w)$ be the graph such that :

- $V_w = \{v_1, v_2, v_3\}$.
- $E_w = \{e_{1,w}, e_{1,\overline{w}}, e_{2,w}, e_{2,\overline{w}}, e_{3,w}, e_{3,\overline{w}}\} \cup E_1 \cup E_2 \cup E_3$ where $E_k = \{e_{k,w'} \mid w' \in \mathbb{W} \setminus \{w, \overline{w}\}\}$ for $k = 1$, 2 or 3.
- $s_w(e_{k,w}) = v_k$, $s_w(e_{k,\overline{w}}) = v_{k+1}$, and $s_w(e \in E_k) = v_k$ for $k = 1$, 2 or 3.
- $t_w(e_{k,w}) = v_{k+1}$, $t_w(e_{k,\overline{w}}) = v_k$, and $t_w(e \in E_k) = v_k$ for $k = 1$, 2 or 3.
- For any $1 \le k \le 3$, for any $w' \in \mathbb{W}$, $l_w(e_{k,w'}) = w'$.

G_w satisfies $\mathscr{I}_P(\mathbb{W})$ and $\mathscr{O}_P(\mathbb{W})$. In $\iota(\mathscr{S}, G_w)$, e yields 3 arcs e_1, e_2, and e_3 labeled (w,i) and such that, for $k = 1$, 2 or 3:

$$s_{\iota(\mathscr{S},G_w)}(e_k) = (v_k, s_{\mathscr{S}}(e)),\ t_{\iota(\mathscr{S},G_w)}(e_k) = (v_{k+1}, t_{\mathscr{S}}(e)).$$

Because the scheme instantiation $\iota(\mathscr{S}, G_w)$ satisfies the non-orientation condition $\mathscr{O}_{\iota(\mathscr{S},G_w)}(i)$, the arcs e_1, e_2, and e_3 admit reverse arcs e_1^{-1}, e_2^{-1}, and e_3^{-1}. By construction of the product, they correspond to a $(\overline{w}, i)$-arc in $\mathscr{S}$. Call it a. Since $\pi_{\mathbb{D}}(\mathscr{S})$ satisfies $\mathscr{O}_{\pi_{\mathbb{D}}(\mathscr{S})}(i)$, $\pi_{\mathbb{D}}(a)$ is a reverse arc of $\pi_{\mathbb{D}}(e)$.

By unicity, $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(e))$ is the conjugate of $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(a))$ and $a = e^{-1}$.

Thereafter, the rule scheme $\mathscr{S}$ satisfies the non-orientation condition $\mathscr{O}_{\mathscr{S}}(i)$. □

Example 19 (Lifting the Non-orientation Condition to Rule Schemes). The rule scheme of Figure 21 satisfies the $\mathscr{O}_{\mathscr{S}}(1)$ while the pattern graph of Figure 19 satisfies $\mathscr{O}_P(\{\varepsilon, 21, \overline{21}\})$. Thus, the instantiated rule of Figure 22 satifies the non-orientation condition for $\mathbb{D}$-rules $\mathscr{O}_r(1)$.

We are left with the study of the cycle property for rule schemes.

9. Cycles consistency in rule schemes

Lifting the incident arcs condition and the non-orientation condition from rules to rule schemes was pretty straightforward. However, extrapolating the cycle condition relies on verifying whether a cycle in the scheme will be instantiated into a cycle. The three sub-conditions of the cycle condition for an instantiated rule relate to the arcs in $\iota(\mathscr{S}, P)$. The issue is thus to be able to verify if a cycle in L or R yields a cycle in $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} L$ or $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R$ without any prior knowledge on P. Intuitively, a pattern graph and a graph scheme can then be seen as two orthogonal spaces: a path containing only moves in the graph scheme stays on the same node from the pattern graph point of view and reciprocally. Thus, we understand that a path in the product of two graphs is a cycle if and only if it is a cycle in both graphs.

[1]In the definition of G_w, all the arithmetic operations are described modulo 3

9.1. Global constraints on combinatorial graphs

The possibility of oriented arcs complicates the identification of cycles at the level of the rule scheme. To bypass this difficulty, we broaden the local study to a global one and fully state the conditions on all the dimensions.

First, we split the dimension set $\mathbb{D}$ between the oriented dimensions $\mathbf{O}$ and the non-oriented dimension $\mathbf{N}$ so that[1] $\mathbf{O} \sqcup \mathbf{N} = \mathbb{D}$ (meaning $\overline{\mathbb{D}} = \mathbf{N} \sqcup \mathbf{O} \sqcup \overline{\mathbf{O}}$). Furthermore, we denote $\|d\|$ the integer value of d, ie $\|d\| = d$ if d is in $\mathbb{D}$ and $\|d\| = \overline{d}$ if d is in $\overline{\mathbf{O}}$. Therefore, conjugation is a bijection on $\overline{\mathbb{D}}^*$ such that $\overline{\overline{d}} = d$ for d in $\mathbf{O}$ and $\overline{d} = d$ for d in $\mathbf{N}$.

In the sequel, we will consider $\mathbb{D} \subseteq \mathbb{N}$, split into $\mathbb{D} = \mathbf{O} \sqcup \mathbf{N}$, and $\mathbf{E} \subseteq \{(i,j) \in \mathbb{D}^2 \mid i < j\}$ as the set of exchangeable dimensions.

9.2. Path equivalence in combinatorial graphs

In this section, we will introduce a rewriting system on $\overline{\mathbb{D}}^*$. For a complete study of string rewriting systems, we advise the reading of the first two chapters of [36]. Let us recall that a set A together with a binary relation $\to$ on A define an abstract reduction system $(A, \to)$. $\xrightarrow{+}$ is the transitive closure of $\to$ and $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$. An abstract reduction system is confluent whenever two elements obtained from the same ancestor have a common descendant. The system is noetherian if it contains no infinite reduction chain. A string rewriting system on an alphabet Σ is a binary relation on Σ^* and defines a reduction system via sub-string rewriting.

The *conjugate rewriting system for* $\mathbf{N}$ *and* $\mathbf{E}$ is:

$$R_{\mathbf{N},\mathbf{E}} = \{(i\overline{i}, \varepsilon) \mid i \in \overline{\mathbb{D}}\} \cup \{(ji, \overline{i}\,\overline{j}) \mid (i,j) \in (\mathbf{N} \cup \mathbf{O})^2 \cup (\mathbf{N} \cup \overline{\mathbf{O}})^2, (\|i\|, \|j\|) \in \mathbf{E}\}.$$

Let us point out that $R_{\mathbf{N},\mathbf{E}}$ is noetherian, but, for $\|\mathbf{N}\| > 1$, $R_{\mathbf{N},\mathbf{E}}$ is not confluent.

The conjugate rewriting system encapsulates path reduction only based on their label. A word can be reduced to another if both words label paths with the same endpoints. The first part of the set definition of $R_{\mathbf{N},\mathbf{E}}$ states that a possible simplification is the removal of two consecutive conjugate labels. When the label is in $\mathbf{N}$, the simplification corresponds to the traversal of an arc and its reverse. When the label is in $\mathbf{O}$, an arc is traveled back and forth. The second part of the set definition classifies which two consecutive dimensions can be inverted, exploiting the cycle constraint on the underlying graph.

Lemma 3. *For two words w and w' in $\overline{\mathbb{D}}^*$, we denote $\mathbf{P}(w,w')$ the following property: For any combinatorial graph G satisfying $\mathscr{O}_G(\mathbf{N})$ and $\mathscr{C}_G(\mathbf{E})$, if there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ then there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$.*

Let w and w' be two words of $\overline{\mathbb{D}}^*$. Then we have :

$$w \xrightarrow{*}_{R_{\mathbf{N},\mathbf{E}}} w' \Rightarrow \mathbf{P}(w,w')$$

Proof. Let G be a $\mathbb{D}$-combinatorial graph satisfying $\mathscr{O}_G(\mathbf{N})$ and $\mathscr{C}_G(\mathbf{E})$, and let v be a node of G. Let us show the result of the lemma by induction on the number of steps in the reduction.

If there are no steps, $w = w'$, and the result is trivial. Otherwise there exists a sequence of reduction:

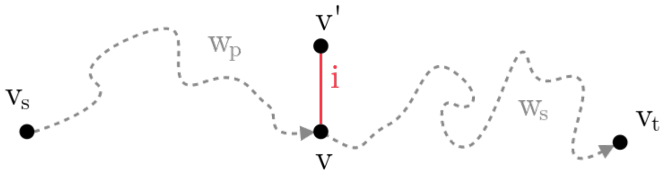
$$w = w_0 \to_{R_{\mathbf{N},\mathbf{E}}} w_1 \to_{R_{\mathbf{N},\mathbf{E}}} w_2 \to_{R_{\mathbf{N},\mathbf{E}}} \dots \to_{R_{\mathbf{N},\mathbf{E}}} w_k = w'$$

for some $k \geq 1$. Suppose there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ in G.

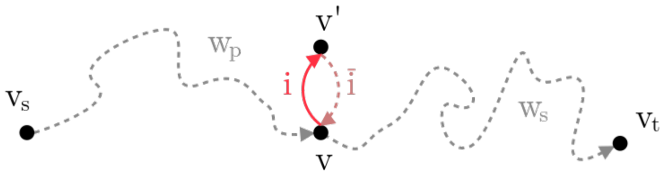
[1]Where $A \sqcup B$ is the disjoint union of the sets A and B.

- If the reduction from w to w_1 is of the form $(i\bar{i}, \varepsilon)$ (with i in $\mathbb{D}$), there exists w_p and w_s in $\overline{\mathbb{D}}^*$ such that $w = w_p i \bar{i} w_s$ and $w_1 = w_p w_s$. Let v be the target of the w_p-path starting at v_s. Since v satisfies $\mathscr{I}_v(i)$, there exists a node v' and an i-arc e such that $s(e) = v$ and $t(e) = v'$. Besides G satisfies $\mathscr{O}_G(\mathbf{N})$. If i is in $\mathbf{N}$ then e admits a reverse i-arc e', otherwise $\bar{i}$ corresponds to the reverse traversal of e. Either way, $i\bar{i}$ is a cycle and the target of the $i\bar{i}$ path of source v is v. Therefore the target of the w_p path of source v is v_s. We can remove the $i\bar{i}$-cycle and the target of the $w_p w_s$-path of source v_s is v_t.

  The case where i belongs to $\mathbf{N}$ and is a non oriented dimension is illustrated in the following figure:

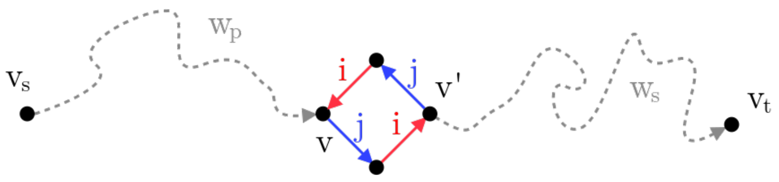


  The case where i belongs to $\mathbf{O}$ and is an oriented dimension is illustrated in the following figure:



- If the reduction from w to w_1 is of the form $(ji, \overline{ji})$ (with (i, j) in $\mathbf{E}$), there exists w_p and w_s in $\overline{\mathbb{D}}^*$ such that $w = w_p j i w_s$ and $w_1 = w_p \bar{i} \bar{j} w_s$. Let v be the target of the w_p-path starting at v_s, and v' be the target of the ji-path starting at v. Because the w-path starting at v_s has v_t for target, the target of the w_s-path from v' is v_t. Since G satisfies $\mathscr{C}_G(\mathbf{E})$, v is the source of a $jiji$-cycle. By unicity of the i and j arcs, the cycle contains v', yielding a ji-path from v' to v. The reverse traversal of this path is an $\overline{ji}$-path from v to v'. Therefore, the target of the $\bar{i}\bar{j}$-path starting at v is v' and we can conclude that the target of the path labeled $w = w_p \bar{i} \bar{j} w_s$ starting at v_s is v_t.

  The most generic case with both i and j being oriented is illustrated in the following figure:



By induction, since $w_1 \xrightarrow{*}_{R_{\mathbf{N},\mathbf{E}}} w_k = w'$, there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$ in G.

□

Note that the essential part of this lemma is about the possibility of switching two dimensions, which is a lot simpler without oriented dimensions. From this lemma, we can derive straightforward corollaries:

1. If $w \overset{*}{\longleftrightarrow}_{R_{\mathbf{N},\mathbf{E}}} w'$ then, in any combinatorial graph G satisfying $\mathscr{O}_G(\mathbf{N})$ and $\mathscr{C}_G(\mathbf{E})$, there is a path $v_s \overset{w}{\rightsquigarrow} v_t$ if and only if there is a path $v_s \overset{w'}{\rightsquigarrow} v_t$.
2. If $w \xrightarrow{*}_{R_{\mathbf{N},\mathbf{E}}} \varepsilon$, then, in any combinatorial graph G satisfying $\mathscr{O}_G(\mathbf{N})$ and $\mathscr{C}_G(\mathbf{E})$, a w-path is a cycle.

3. These results extends[1] to words $\hat{w}$ and $\hat{w}'$ in $\mathbb{W}^*$, we consider the flatten word in $\overline{\mathbb{D}}^*$ to use the reduction system.

9.3. Preservation of consistency in rule schemes

The result of Lemma 3 can also be extended to $\mathbb{W}$-pattern graphs obtained via the $\mathbb{W}$-pattern functor on $\mathbb{D}$-combinatorial graphs. A word $\hat{w}$ in $\mathbb{W}^*$ labels a path $v_s \rightsquigarrow v_t$ in a $\mathbb{W}$-pattern graph P if and only if the flatten word labels a path $v_s \rightsquigarrow v_t$ in the underlying $\mathbb{D}$-combinatorial graph G. Indeed the $\mathbb{W}$-pattern functor extracts a subset of the node set of G and turns w-paths (for w in $\mathbb{W}$) into w-arcs. Therefore, we can reduce flatten words of $\mathbb{W}^*$ using $R_{\mathbf{N},\mathbf{E}}$ and consider paths in P.

Definition 25 (Cycle Constraint for Pattern Graphs)**.** *A $\mathbb{W}$-pattern graph P satisfies the cycle constraint $\mathscr{C}_P(\mathbf{N},\mathbf{E})$ if it is coherent with $R_{\mathbf{N},\mathbf{E}}$: for all words w, w' in $\mathbb{W}^*$, w can be rewritten as w' by $R_{\mathbf{N},\mathbf{E}}$ implies that for any w-path $v_s \overset{w}{\rightsquigarrow} v_t$ in P, there is a w'-path $v_s \overset{w'}{\rightsquigarrow} v_t$ in P.*

The cycle constraint for pattern graphs is quite restrictive. Yet, if the pattern graph P is built on a $\mathbb{D}$-combinatorial graph that satisfies $\mathscr{O}_G(\mathbf{N})$ and $\mathscr{C}_G(\mathbf{E})$, then P satisfies the cycle constraint for pattern graphs $\mathscr{C}_P(\mathbf{N},\mathbf{E})$ as a consequence of Lemma 3.

For instance, the pattern graph given in Figure 19 (see Section 6) satisfies the non-orientation constraint for the set of exchangeable dimensions $\{(1,2)\}$ and $\{1\}$ as the set of non-oriented dimensions.

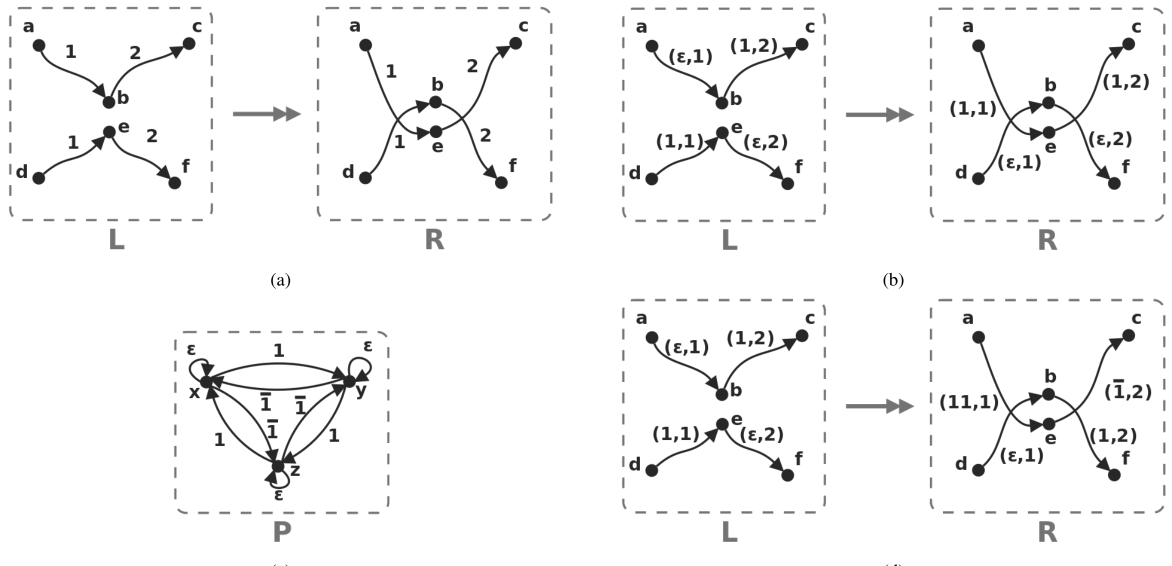
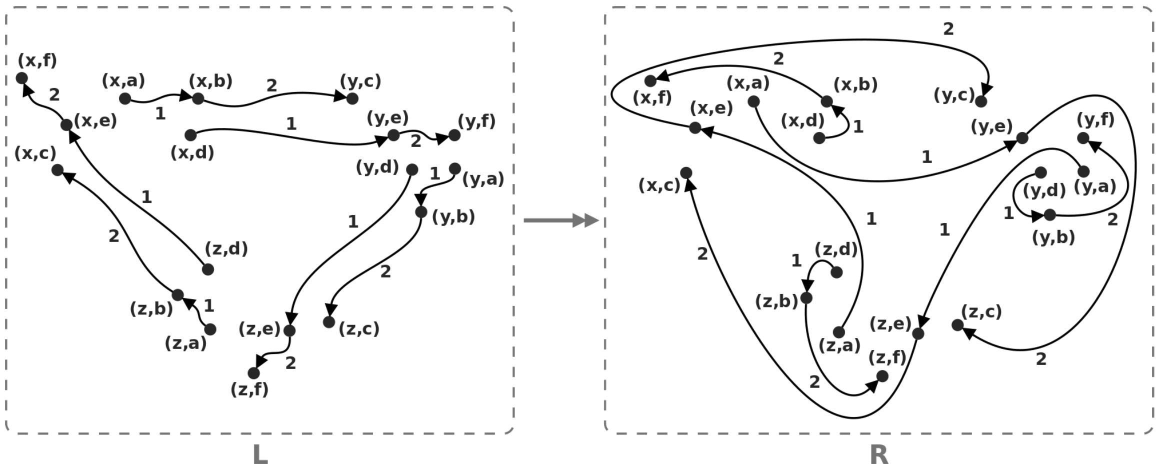


Figure 26: Two rule schemes with the same core and a pattern graph for instantiation.

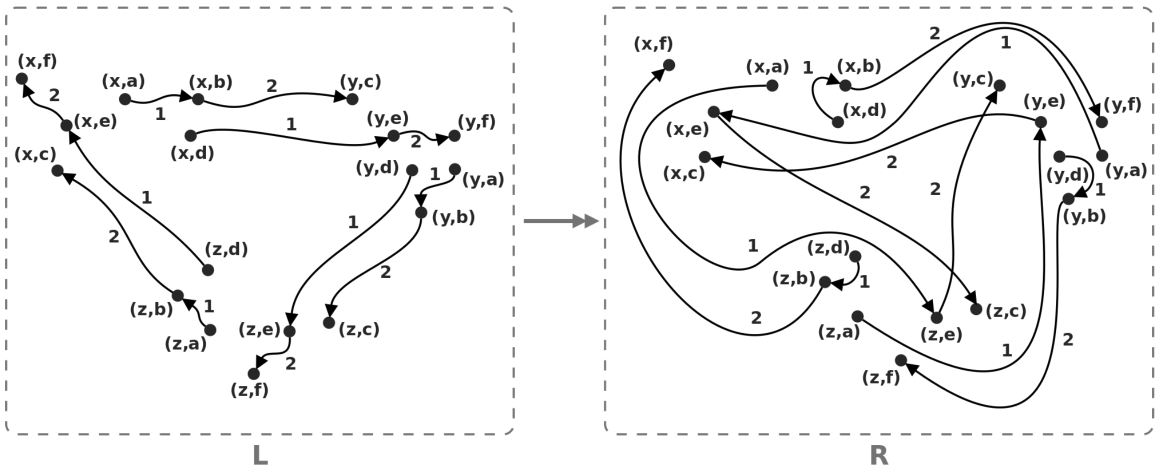
Example 20 (Intuition for Cycle Preservation in Rule Schemes). Similar to the incident arcs and the non-orientation properties, we force the core of the rule to satisfy the cycle condition for $\mathbb{D}$-rules. Consider the rule given in Figure 26a. It satisfies $\mathscr{C}_r(1,2)$ since the two 12-paths are coherent, and each node is the source of an optimal path. The two rules from Figure 26b and 26d have the previous $\mathbb{D}$-rule as core rule. We consider their instantiation with the pattern graph given in Figure 26c. The instantiation of the first rule scheme (see Figure 26b)

[1] A word $\hat{w}$ in $\mathbb{W}^*$ is understood as the concatenation of words from $\mathbb{W}$. Thus, for such a word $\hat{w}$, there exists $k \geq 0$ and $w_1,\ w_2,\ \dots w_k$ in $\mathbb{W}$ such that $\hat{w} = w_1 w_2 \dots w_k$. Each w_i (for $1 \leq i \leq k$) is a word of $\mathbb{W}$, i.e., a word with letters from $\overline{\mathbb{D}}$, such that $w_i = (w_i)_{(1)} \dots (w_i)_{(l_i)}$. When flatten, $\hat{w}$ is therefore equal to $(w_1)_{(1)} \dots (w_1)_{(l_1)} (w_2)_{(1)} \dots (w_2)_{(l_2)} \dots (w_k)_{(1)} \dots (w_k)_{(l_k)}$.

is given in Figure 27a. The coherent optimal paths in the core of the rule do not yield coherent paths in the instantiated rule. During the instantiation process, the product of the graph scheme with the embedded pattern graph creates an arc between nodes in the Cartesian product of the node sets whenever there is an arc with the same label in both graphs. Indeed, arcs in the pattern graph stand for paths in the underlying combinatorial graph. Thus, the $\mathbb{W}$-part of the label in the graphs of the rule scheme indirectly represents the path in the combinatorial graph where we will apply the rule scheme. In the instantiated rule's left-hand side, we can find a 12-optimal path from (x,a) to (y,c), but we cannot find a coherent path in the right-hand side. Essentially the $\mathbb{W}$-parts of the label misfit and provide different paths. Indeed the path *abc* in *L* (of Figure 26b) has 1 for $\mathbb{W}$-label whereas the path *aec* in *R* has 11 for $\mathbb{W}$-label. From Lemma 3, we know that we can prevent this from happening if we force coherent paths to have their $\mathbb{W}$-part of label congruent for $\overset{*}{\leftrightarrow}_{R_{\mathbf{N},\mathbf{E}}}$.



(a)



(b)

Figure 27: Instantiation of the rule schemes from Figure 26 with the pattern graph of Figure 26c.

To ensure the coherence of optimal paths, we extend the definition to rule schemes.

Definition 26 (Coherence of Optimal Path in Rule Schemes)**.** *Let $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$-combinatorial rule scheme. Two optimal paths p and p' of $\pi_{\mathbb{D}}(\mathscr{S})$ are coherent in $\mathscr{S}$ if p and p' are coherent in $\pi_{\mathbb{D}}(\mathscr{S})$ and*

$$l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p)) \overset{*}{\leftrightarrow}_{R_{\mathbf{N},\mathbf{E}}} l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p')).$$

Example 21 (Coherence of Optimal Path in Rule Schemes). The rule scheme of Figure 26d has coherent optimal paths. For instance, the 12-path *abc* has 1 for $\mathbb{W}$-label in L whereas the 12-path *aec* has $11\overline{1} \xrightarrow{*}_{R_{\emptyset,\{(1,2)\}}} 1$ for $\mathbb{W}$-label in R. Similarly, the paths def (in L) and dbf (in R) have 1 for $\mathbb{W}$-label. The instantiated rule of Figure 27b preserves the coherence of optimal paths. For example, the optimal path $(x,a)(x,b)(y,c)$ in L is coherent with the optimal path $(x,a)(z,e)(y,c)$ in R. Likewise, the optimal path $(y,d)(z,e)(z,f)$ in L is coherent with the optimal path $(y,d)(y,b)(z,f)$ in R.

The cycle constraint on pattern graphs ensures that cycles from graph schemes will be associated with cycles in the embedded pattern graph, yielding cycles in the instantiated rule.

Definition 27 (Cycle Condition for Rule Schemes)**.** *A $(\mathbb{W},\mathbb{D})$-combinatorial rule scheme $\mathscr{S} = L \twoheadrightarrow R$ satisfies the cycle condition $\mathscr{C}_{\mathscr{S}}(\mathbf{E})$ if :*

- *The core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathscr{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$.*
- *$\mathscr{S}$ is coherent with $R_{\mathbf{N},\mathbf{E}}$, i.e.:*
  - *For any (i,j) in $\mathbf{E}$, any coherent (i,j)-optimal path of $\pi_{\mathbb{D}}(\mathscr{S})$ is coherent in $\mathscr{S}$.*
  - *For any (i,j) in $\mathbf{E}$, any concatenated word w labeling a cycle in $\pi_{\mathbb{W}}(\mathscr{S})$ corresponding to an $ijij$-cycle in $\pi_{\mathbb{D}}(\mathscr{S})$ can be reduced to ε using $R_{\mathbf{N},\mathbf{E}}$.*

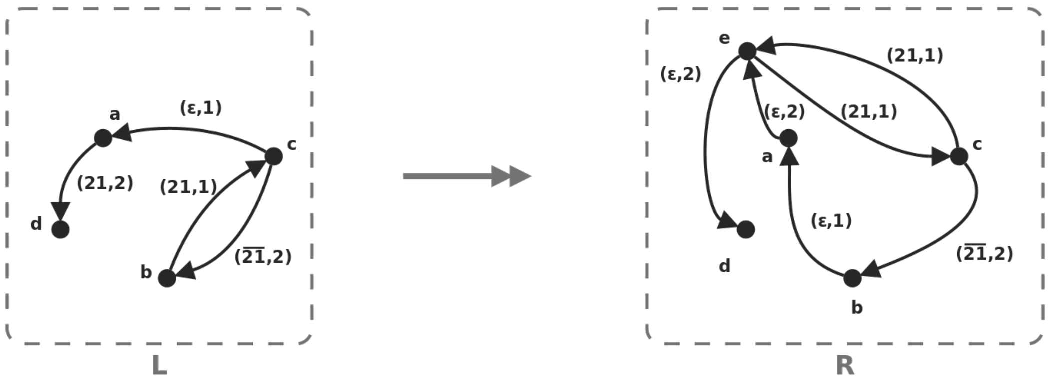


Figure 28: A rule scheme satifying $\mathscr{C}_{\mathscr{S}}(1,2)$.

Example 22 (Cycle Condition for Rule Schemes). The rule scheme illustrated previously (see Figure 26d) satisfies the cycle condition $\mathscr{C}_{\mathscr{S}}(\{(1,2)\})$ as it contains only coherent optimal paths. Let us provide another rule scheme satisfying a cycle condition with the example of Figure 28. The 12-optimal path *cad* is coherent with the 12-optimal path *ced* (with 21 as the $\mathbb{W}$-label). Node *b* is the source of a 1212-cycle in L and R. The cycle is labeled $21\,\overline{21}\,21\,\overline{21} \xrightarrow{*}_{R_{\emptyset,\{(1,2)\}}} \varepsilon$ in L and $21\,\overline{21} \xrightarrow{*}_{R_{\emptyset,\{(1,2)\}}} \varepsilon$ in R. Similarly, node *c* is the source of a 2121-cycle in L and R with $\mathbb{W}$-label congruent to ε in both sides. The added node *e* is source of a 2-arc in an coherent optimal path and the source of a 1-arc in a cycle.

Theorem 6 (Lifting the Cycle Condition to Rule Schemes)**.** *Let $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$-combinatorial rule scheme satisfying the non-orientation condition $\mathscr{O}_{\mathscr{S}}(\mathbf{N})$.*

If the rule scheme $\mathscr{S}$ satisfies the cycle condition $\mathscr{C}_{\mathscr{S}}(\mathbf{E})$, then for all $\mathbb{W}$-pattern graph P that satisfies $\mathscr{I}_P(\mathbb{W})$, and $\mathscr{C}_P(\mathbf{N},\mathbf{E})$, the scheme instantiation $\iota(\mathscr{S},P)$ satisfies the cycle condition $\mathscr{C}_{\iota(\mathscr{S},P)}(\mathbf{E})$.

Proof. Let $\mathscr{S} = L \twoheadrightarrow R$ be a $(\mathbb{W},\mathbb{D})$-combinatorial rule scheme satisfying the non-orientation condition $\mathscr{O}_{\mathscr{S}}(\mathbf{N})$.

Let $(i,j) \in \mathbf{E}$ be a pair of dimension, p_l be the morphism $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} L \to \mathbb{E}_{\mathbb{D}}(P)$, p_r be the morphism $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R \to \mathbb{E}_{\mathbb{D}}(P)$, l_p be the morphism $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} L \to L$, r_p be the morphism $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R \to R$, such that the morphisms come from the following products:

$$\begin{tikzcd}
\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} L \arrow[d, "l_p"] \arrow[dr, "p_l"] & & \mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R \arrow[dl, "p_r"'] \arrow[d, "r_p"] \\
L & \mathbb{E}_{\mathbb{D}}(P) & R
\end{tikzcd}$$

Sub-condition 1 of the cycle condition.

Let $p = v_s \rightsquigarrow v_t$ be (i,j)-optimal path in $\pi_{\mathbb{D}}(\iota(L,P))$. And denote $p_{\iota(L,P)}$ the corresponding path in $\iota(L,P)$. Thus, v_s and v_t are nodes of $\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} (L \cap R)$ and there exist nodes a_s, a_t in $\mathbb{E}_{\mathbb{D}}(P)$ and u_s, u_t in $L \cap R$ such that $p_r(v_s) = a_s$, $r_p(v_s) = u_s$, $p_r(v_t) = a_t$ and $r_p(v_t) = u_t$.

By construction of the product, the (i,j)-optimal path p yields two specific paths. One is an (i,j)-alternating path p_P in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(P))$, from a_s to a_t with the same label. The other is (i,j)-optimal path $p_{\pi_{\mathbb{D}}(L)}$ in $\pi_{\mathbb{D}}(L)$, from u_s to u_t with the same label.

Because the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathscr{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$, $p_{\pi_{\mathbb{D}}(L)}$ is coherent and there is an (i,j)-optimal path $p_{\pi_{\mathbb{D}}(R)}$ from u_s to u_t with the same label in $\pi_{\mathbb{D}}(R)$. The paths in $\mathbb{E}_{\mathbb{D}}(P)$ and R create an (i,j)-alternating path p' in $\pi_{\mathbb{D}}(\iota(R,P))$. By hypothesis on $\mathscr{S}$, $p_{\iota(L,P)}$ is coherent in $\mathscr{S}$. Thus, $l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p)) \overset{*}{\leftrightarrow}_{R_{\mathbf{N},\mathbf{E}}} l_{\pi_{\mathbb{W}}(\mathscr{S})}(\pi_{\mathbb{W}}(p'))$. Because P satisfies $\mathscr{C}_P(\mathbf{N},\mathbf{E})$ p' is a path from v_s to v_t.

Since $p_{\pi_{\mathbb{D}}(L)}$ is optimal, the path p' only contains added arcs and does not overlap. Besides, this path is maximal (if not it would be included in a path obtained from R, meaning $p_{\pi_{\mathbb{D}}(L)}$ was not coherent). Similarly, it cannot belong to a cycle, as such a cycle would come from R. Therefore p' is optimal in $\pi_{\mathbb{D}}(\iota(R,P))$ and coherent with p.

The reverse proof holds by symmetry and the scheme instantiation $\iota(\mathscr{S},P)$ satisfies sub-condition 1 of Definition 16.

Sub-condition 2 of the cycle condition.

Let v be a preserved node of $(\iota(L,P)) \cap (\iota(R,P))$ that is the source of an i-arc in $\pi_{\mathbb{D}}(\iota(L,P)) \setminus \pi_{\mathbb{D}}(\iota(R,P))$. Therefore, there exist a node a in $\mathbb{E}_{\mathbb{D}}(P)$ and a node u in K such that $p_r(v) = a$ and $r_p(v) = u$. By construction of the product, the existence of an i-arc of source v in $\pi_{\mathbb{D}}(\iota(L,P)) \setminus \pi_{\mathbb{D}}(\iota(R,P))$ means there is an i-arc in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(P))$ of source a and an i-arc in $\pi_{\mathbb{D}}(L)$ of source u. Since the core $\pi_{\mathbb{D}}(\mathscr{S})$ of $\mathscr{S}$ satisfies the cycle condition $\mathscr{C}_{\pi_{\mathbb{D}}(\mathscr{S})}(\mathbf{E})$, the i-arc in R either belongs to an $ijij$-cycle or to an (i,j)-optimal path.

- Assume that the i-arc belongs to an (i,j)-optimal path. Because P satisfies $\mathscr{I}_P(\mathbb{W})$, the (i,j)-optimal path in $\pi_{\mathbb{D}}(R)$ can be associated with an (i,j)-alternating path of source a in $\mathbb{E}_{\mathbb{D}}(P)$. Similar to the proof for sub-condition 1, the paths in $\mathbb{E}_{\mathbb{D}}(P)$ and R yield an (i,j)-optimal path in $\iota(R,P)$.
- Otherwise, the i-arc extends to an $ijij$-cycle. The concatenated word w in $\pi_{\mathbb{W}}(R)$, corresponding to the cycle, can be reduced to ε using $R_{\mathbf{N},\mathbf{E}}$. Because P satisfies $\mathscr{C}_P(\mathbf{N},\mathbf{E})$, w labels a cycle in P and thus in $\pi_{\mathbb{W}}(\mathbb{E}_{\mathbb{D}}(P))$. The product construction keeps this cycle and $\pi_{\mathbb{D}}(v)$ is the source of an $ijij$-cycle in $\pi_{\mathbb{D}}(\mathbb{E}_{\mathbb{D}}(P) \times_{(\mathbb{W}\times\mathbb{D})} R)$.

Thus, the scheme instantiation $\iota(\mathscr{S},P)$ satisfy sub-condition 2 of Definition 16.

Sub-condition 3 of the cycle condition.

The proof is similar to sub-condition 2.

Thereafter, $\iota(\mathscr{S},P)$ satisfies $\mathscr{C}_{\iota(\mathscr{S},P)}(E)$. □

Note that Lemma 3 provides a formal characterization to describe which rule schemes always instantiate into consistent rules. Although the rewriting system is a valuable characterization, it is not confluent, and we would need a reduction strategy to use it in practice.

10. Combinatorial maps

10.1. Preservation of consistency for combinatorial maps

Recall that an n-G-map is a totally labeled $[\![0,n]\!]$-topological graph satisfying $\mathscr{O}_G([\![0,n]\!])$, $\mathscr{I}_G([\![0,n]\!])$, and $\mathscr{C}_G([\![0,n]\!]_{+2})$ where $[\![0,n]\!]_{+2}$ denotes the dimensions $i, j \in [\![0,n]\!]$ such that $i+2 \leq j$. From Theorems 1, 2, and 3, if a rule r in the category of $[\![0,n]\!]$-**Graphs** satisfies $\mathscr{O}_r([\![0,n]\!])$, $\mathscr{I}_r([\![0,n]\!])$ and $\mathscr{C}_r([\![0,n]\!]_{+2})$, then the result graph H of the direct derivation $G \Rightarrow^{r,m} H$ is an n-G-map. Similarly, if a rule r satisfies $\mathscr{O}_r([\![2,n]\!])$, $\mathscr{I}_r([\![1,n]\!])$ and $\mathscr{C}_r([\![1,n]\!]_{+2})$ then the graph H result of the direct derivation $G \Rightarrow^{r,m} H$ where G is an n-O-map is also an n-O-map.

Graph transformations built on production rules satisfying the appropriate conditions yield derivations preserving the topological constraints of the considered model. When extending rules to rule schemes, the verifications are lifted to the rule scheme level (Theorems 4, 5, and 6). In particular, the result from Theorem 4 ensures the dangling condition can be checked independently of the combinatorial map on which the rule scheme is applied.

G-maps and O-maps are defined by slightly different constraints, namely the orientation of the 1-arcs for the O-maps. These constraints yield different conditions for the preservation of the topological consistency when applying rule-based graph transformations. When trying to transpose the conditions to rule schemes, we need to cope with dimension conjugation. In the model of G-maps, the whole set $\mathbb{D}$ of dimensions is non-oriented, such that $\mathbf{N} = \mathbb{D}$ and $\mathbf{O} = \emptyset$. With $\mathbf{N}$ equals to $\mathbb{D}$, the construction of rule schemes is drastically simplified, and we can do away with dimension conjugation. In the case of O-maps, $\mathbb{D}$ is split into $\mathbf{N} = \mathbb{D} \setminus \{1\}$ and $\mathbf{O} = \{1\}$ meaning that we actually need to take care of the conjugation condition from Theorem 6.

When applying a rule scheme to an n-G-map or n-O-map, the pattern graph built during the application process inherits properties from the map. For instance, consider an n-G-map G. Thus any pattern graph P built on G with a set of words $\mathbb{W}$ satisfies the consistency constraint for pattern graphs $\mathscr{I}_P(\mathbb{W})$, $\mathscr{O}_P(\mathbb{W})$, and $\mathscr{C}_P([\![0,n]\!], [\![0,n]\!]_{+2})$. Therefore the instantiation of any rule scheme $\mathscr{S}$ that satisfies $\mathscr{I}_{\mathscr{S}}([\![0,n]\!])$, $\mathscr{O}_{\mathscr{S}}([\![0,n]\!])$, $\mathscr{C}_{\mathscr{S}}([\![0,n]\!]_{+2})$ yields a rule $\iota(\mathscr{S},P)$ that satisfies the conditions $\mathscr{O}_{\iota(\mathscr{S},P)}([\![0,n]\!])$, $\mathscr{I}_{\iota(\mathscr{S},P)}([\![0,n]\!])$ and $\mathscr{C}_{\iota(\mathscr{S},P)}([\![0,n]\!]_{+2})$. Therefore any direct transformation on any instantiation of the rule scheme results in an n-G-map. Similarly, rule schemes satisfying the appropriate conditions yield instantiations that transform n-O-maps into n-O-maps.

As a consequence, we can certify that a rule preserves the model's constraints. We provide algorithms to check this preservation similarly to Jerboa's syntactic analyzer. Jerboa's rule editor assumes a set-based representation of graphs: a graph is described as a set of nodes and a set of arcs. We can access nodes or arcs using their names or looping over the corresponding set and assume that the set of dimensions $\mathbb{D}$ is fixed.

The function providing all cycles of a graph can be obtained with a union-find strategy. Start with paths that consist of arcs labeled i and j and unify them recursively whenever one is the source of the other and the dimensions alternate. When the construction stops, the set of paths is pruned to keep only the cycles.

A rule $L \twoheadrightarrow R$ consists of two graphs $L = (V_L, E_L)$ and $R = (V_R, E_R)$ where the preserved elements have the same nodes in both sides. Thus we can check if a node or an arc is preserved. Besides, we compute optimal paths for one side of a rule scheme. The function to compute optimal paths is similar to the construction of cycles, only considering non-preserved arcs. The obtained set of paths should also be pruned, in this case, to eliminate paths that are sub-paths of cycles.

Since the rewriting system $R_{\mathbf{N},\mathbf{E}}$ is noetherian, the set of normal forms for a word is finite and computable. However, because $R_{\mathbf{N},\mathbf{E}}$ is not necessarily confluent, we may need to compute the complete collection of normal forms to check whether a word reduces to ε, or whether two words are congruent. In practice, the words on arcs are usually of length smaller than 2 or 3, and the concatenation along the path nearly always yields words of size at most 6 to 10. For simplicity, we denote $\mathscr{N}\mathscr{F}_{R_{\mathbf{N},\mathbf{E}}}(w)$ the set of normal forms for a word w from $\overline{\mathbb{D}}^*$ for the rewriting system $R_{\mathbf{N},\mathbf{E}}$.

The verification should first run Algorithm 1, then Algorithm 2 and Algorithm 3 as the last two algorithms assume that the input rule scheme satisfies the incident arcs condition. The algorithms only check for one dimension (resp. pair of dimension for the cycle condition) and, therefore, should be run on all dimensions relevant to the model.

Algorithm 1: Check incident arcs consistency for a dimension i.

```
Input: A rule scheme $\mathscr{S} = L \twoheadrightarrow R$, and a dimension $i$ of $\mathbb{D}$.
Output: $True$ if the rule satisfies $\mathscr{I}_{\mathscr{S}}(i)$, $False$ otherwise.
1  Function check_incident_arcs(L,R,i):
2      $consistent \leftarrow True$
3      foreach $v \in V_L$ do
4          $S_L \leftarrow \{e \in E_L \mid s_L(e) = v$ and $l_{\pi_{\mathbb{D}}(L)}(e) = i\}$
5          $T_L \leftarrow \{e \in E_L \mid t_L(e) = v$ and $l_{\pi_{\mathbb{D}}(L)}(e) = i\}$
6          if $v \in V_R$ then                                                    // Preserved node
7              $consistent \leftarrow consistent$ and $|S_L| \leq 1$ and $|T_L| \leq 1$          // At most one incident arc
8              $S_R \leftarrow \{e \in E_R \mid s_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i\}$
9              $T_R \leftarrow \{e \in E_R \mid t_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i\}$
10             $consistent \leftarrow consistent$ and $|S_L| = |S_R|$ and $|T_L| = |T_R|$     // Same in both sides
11         else                                                                  // Deleted node
12             $consistent \leftarrow consistent$ and $|S_L| = 1$ and $|T_L| = 1$       // Look for a unique deleted arc
13     foreach $v \in V_R$ do
14         if $v \notin V_L$ then                                                  // Added node
15             $S_R \leftarrow \{e \in E_R \mid s_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i\}$
16             $T_R \leftarrow \{e \in E_R \mid t_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i\}$
17             $consistent \leftarrow consistent$ and $|S_R| = 1$ and $|T_R| = 1$       // Look for a unique added arc
18     return $consistent$
```

In Algorithm 1, we check the condition $\mathscr{I}_{\mathscr{S}}(i)$ for a rule scheme $\mathscr{S}$ and a dimension i from Definition 22. Lines 6 to 10, ensures that a preserved node is the source (resp. target) of an i-arc in the left-hand side if and only if it is the source (resp. target) of an i-arc in the right-hand side. Lines 13 to 17 ascertain that each added node is the source (resp. target) of an i-arc, i.e., $\pi_{\mathbb{D}}(R)$ satisfies $\mathscr{I}_{(V_R \setminus V_L),\pi_{\mathbb{D}}(R)}(i)$. Similarly, line 12 certifies that each deleted node is the source (resp. target) of an i-arc, i.e., $\pi_{\mathbb{D}}(L)$ satisfies $\mathscr{I}_{(V_L \setminus V_R),\pi_{\mathbb{D}}(L)}(i)$. This last sub-condition yields the dangling condition on the instantiated rule, which guarantees applicability (provided that an instantiation is possible).

In Algorithm 2, we check the condition $\mathscr{O}_{\mathscr{S}}(i)$ for a rule scheme $\mathscr{S}$ and a dimension i from Definition 24. This condition boils down to only having oriented arcs in the interface, where the definition of reverse arc is extended to encompass the conjugation of the $\mathbb{W}$ part of the label. In line 4, we construct the set of possible reverse arcs. The first three boolean conditions $s_L(e') = t_L(e)$, $t_L(e') = s_L(e)$, and $l_{\pi_{\mathbb{D}}(L)}(e') = i$ ensure that the arcs are reverse arcs in the core $\pi_{\mathbb{D}}(\mathscr{S})$. These boolean conditions tackle the first sub-condition of Definition 24. The last boolean condition states that the reverse arcs in the core have conjugate labels, according to the second sub-condition of Definition 24. When we find one, we make sure that both have the same status (deleted or preserves) at lines 5 to 10. Otherwise, i.e., when the arc is oriented, we check whether it is preserved at line 12.

Algorithm 2: Check non orientation consistency for a dimension i.

```
    Input: A combinatorial rule scheme $\mathscr{S} = L \twoheadrightarrow R$, and a dimension $i$ of $\mathbb{D}$.
    Output: $True$ if the rule satisfies $\mathscr{O}_{\mathscr{S}}(i)$, $False$ otherwise.
1   Function check_non_orientation(L,R,i):
2       $consistent \leftarrow True$
3       foreach $e \in \{e' \in E_L \mid l_{\pi_{\mathbb{D}}(L)}(e) = i\}$ do
4           $I_L \leftarrow \{e' \in E_L \mid s_L(e') = t_L(e)$ and $t_L(e') = s_L(e)$ and $l_{\pi_{\mathbb{D}}(L)}(e') = i$ and $l_{\pi_{\mathbb{W}(L)}}(e') = \overline{l_{\pi_{\mathbb{W}(L)}}(e)}\}$
5           if $|I_L| = 1$ then                                         // Found a reverse arc
6               $r \leftarrow e' \in I_L$
7               if $e \in E_R$ then                                     // Preserved arc
8                   $consistent \leftarrow consistent$ and $r \in E_R$    // The reverse must be preserved too
9               else                                                    // Deleted arc
10                  $consistent \leftarrow consistent$ and $r \notin E_R$ // The reverse must be deleted too
11          else                                                        // No reverse arc
12              $consistent \leftarrow consistent$ and $e \in E_R$        // The arc must be preserved
13      $(\ldots)$ idem (lines 3 to 12) for arcs in $E_R$ by considering functions $s_R$, $t_R$, $l_{\pi_{\mathbb{D}}(R)}$, and $l_{\pi_{\mathbb{W}(R)}}$, as well as switching the roles of $E_L$
          and $E_R$.
14      return $consistent$
```

Algorithm 3: Check cycles consistency for a pair of dimensions (i, j).

```
   Input: A combinatorial rule scheme $\mathscr{S} = L \twoheadrightarrow R$, and two dimensions $i$ and $j$ of $\mathbb{D}$.
   Output: $True$ if the rule satisfies $\mathscr{C}_{\mathscr{S}}(\{(i,j)\})$, $False$ otherwise.
1  Function check_cycles(L,R,i,j):
2      $consistent \leftarrow True$
3      $PathsL,\ PathsR \leftarrow \{p = v_s \rightsquigarrow v_t \in L \mid \text{p is } (i,j)\text{-optimal in } L\},\ \{p = v_s \rightsquigarrow v_t \in R \mid \text{p is } (i,j)\text{-optimal in } R\}$ // Optimal paths
4      $CyclesL,\ CyclesR \leftarrow \{p = v_s \rightsquigarrow v_t \in L \mid \text{p is an} (i,j)\text{-cycle in } L\},\ \{p = v_s \rightsquigarrow v_t \in R \mid \text{p is an} (i,j)\text{-cycle in } R\}$ // Cycles
       // Check paths coherence
5      foreach $p \in PathsL$ do
6          $C \leftarrow \{p' \in PathsR \mid s_L(p) = s_R(p')$ and $t_L(p) = t_R(p')$ and $l_{\pi_{\mathbb{D}}(L)}(p) = l_{\pi_{\mathbb{D}}(R)}(p')\}$
7          if $|C| = 1$ then
8              $c \leftarrow p' \in C$
9              $Np \leftarrow \mathscr{NF}_{R_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(L)}(p))$    // The set of normal forms of the label of p
10             $Nc \leftarrow \mathscr{NF}_{R_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(L)}(c))$
11             $consistent \leftarrow consistent$ and $Np \cap Nc \neq \emptyset$    // Labels are congruent
12         else
13             $consistent \leftarrow False$
14     $(\ldots)$ idem (lines 5 to 12) for paths in $R$
       // Check cycles
15     foreach $p \in CyclesL$ do
16         $consistent \leftarrow consistent$ and $[|p| = 2$ or $|p| = 4]$    // Check cycle size
17         $N \leftarrow \mathscr{NF}_{R_{\mathbf{N},\mathbf{E}}}(l_{\pi_{\mathbb{W}}(L)}(p))$
18         $consistent \leftarrow consistent$ and $\varepsilon \in N$    // Reduction to the empty word
19     $(\ldots)$ idem (lines 14 to 17) for cycles in $R$
       // Check extension to path or cycle
20     foreach $v \in V_R$ do
21         $i\_arc \leftarrow null$
22         if $v \in V_L$ then    // Preserved node
23             $S_L \leftarrow \{e \in E_L \mid s_L(e) = v$ and $l_{\pi_{\mathbb{D}}(L)}(e) = i\}$
24             if $|S_L| = 1$ then
25                 $i\_arc \leftarrow e \in E_R$ such that $s_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i$    // Uniquely exists from Alg. 1
26         else    // Added node
27             $i\_arc \leftarrow e \in E_R$ such that $s_R(e) = v$ and $l_{\pi_{\mathbb{D}}(R)}(e) = i$    // Uniquely exists from Alg. 1
28         $P \leftarrow \{p \in CyclesR \cup PathsR \mid i\_arc \in p\}$    // Empty set if i_arc = null
29         $consistent \leftarrow consistent$ and $|P| \neq 0$
30         $(\ldots)$ idem (lines 20 to 28) for dimension $j$
31     return $consistent$
```

In Algorithm 3, we check the condition $\mathscr{C}_{\mathscr{S}}(i, j)$ for a rule scheme $\mathscr{S}$ and a pair of dimensions (i, j) from Definition 27. The condition is obtained from the condition for instantiated rules on the core of the scheme and an extension with the rewriting system. The sub-condition on the core (see Definition 16) is verified by lines 8 (path coherence), lines 24 to 27 and 30 to 32 (preserved nodes), lines 29 to 32 (added nodes). The extension with the rewriting system to coherence with $R_{\mathbf{N},\mathbf{E}}$ is covered by lines 10 to 15 (path coherence) and lines 17 to 20 (cycles reduction to ε).

In Section 3, we talked about the quad subdivision operation dedicated to mesh refinement. The rule scheme of Figure 29 describes the subdivision operation for G-maps, while the one of Figure 30 illustrates it for O-maps.

Example 23 (Consistency verification of the quad subdivision operation for a 2-G-map (Figure 29)). The rule scheme $\mathscr{S}_G$ is a $(\{\varepsilon, 0, 1, 2\}, \{0, 1, 2\})$-rule schemes. It satisfies $\mathscr{I}_{\mathscr{S}_G}(\{0, 1, 2\})$ since the preserved node a has incident non-oriented 0-, 1-, and 2-arcs in both L and R while added nodes b, c, and d have incident arcs for each dimension in $\{0, 1, 2\}$. All arcs are non-oriented and the only arcs with $\mathbb{W}$ label different from ε are loops. Thus, $\mathscr{S}_G$ also satisfies $\mathscr{O}_{\mathscr{S}_G}(\{0, 1, 2\})$. Finally, $\mathscr{S}_G$ also satisfies $\mathscr{C}_{\mathscr{S}_G}(\{(0, 2)\})$ because all nodes are sources of $ijij$-cycles whose $\mathbb{W}$ label reduces to ε via $R_{\{0,1,2\},\{(0,2)\}}$. For instance, the cycle $(\varepsilon, 0)(2, 2)(\varepsilon, 0)(2, 2)$ of source a corresponds to a 0202 cycle. The $\{\varepsilon, 0, 1, 2\}$-part of the label is $\varepsilon 2 \varepsilon 2 = 22$. Since 2 is non oriented, $(22, \varepsilon)$ is a rewriting rule of $R_{\{0,1,2\},\{(0,2)\}}$ and the label reduces to ε. The execution of Algorithms 1, 2, and 3

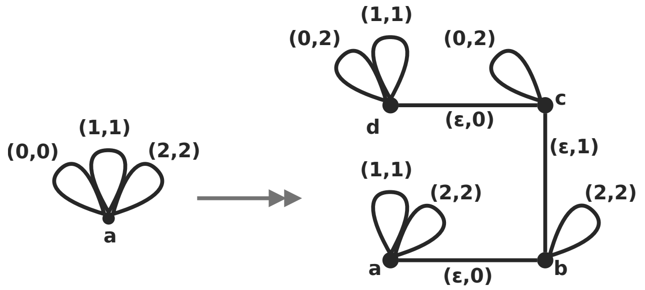


Figure 29: Quad subdivision operation: rule scheme $\mathscr{S}_G$ for a 2-G-map.

will return $True$ for all relevant dimensions. From these three conditions, we deduce that the rule scheme always transforms a 2-G-map into a 2-G-map.

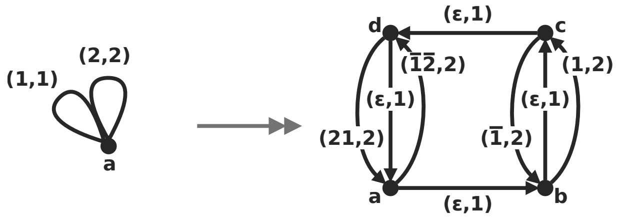


Figure 30: Quad subdivision operation: rule scheme $\mathscr{S}_O$ for a 2-O-map.

Example 24 (Consistency verification of the quad subdivision operation for a 2-O-map (Figure 30)). The rule scheme $\mathscr{S}_O$ is a $(\{\varepsilon,1,\bar{1},2,21,\bar{1}\bar{2}\},\{1,2\})$-rule schemes. The execution of the Algorithms 1 and 2 respectively for dimension sets $\{1,2\}$ and 2 will return $True$ as $\mathscr{S}_O$ satisfies $\mathscr{I}_{\mathscr{S}_O}(\{1,2\})$ and $\mathscr{O}_{\mathscr{S}_G}(\{2\})$. Note that, as 1 is an oriented dimension, the conjugation part of the condition of non-orientation needs to be taken into account. The cycle condition does not need any verification because a 2-O-map is not subject to any cycle constraint.

The quad subdivision of the character presented at the beginning of the paper was realized with Jerboa. Jerboa [17] is a generator of geometric modelers based on the work from [15] and [16] for the framework of G-maps. This platform allows for the prototyping of dedicated modelers. In its current iteration, Jerboa supports the definition of modeler kernels using G-maps in any dimension. The rules are specified using variables [32] that can be simulated in our framework. Nodes of the rule with variables correspond to the nodes of the rule scheme. Any i-arc between nodes of the rules with variables is replaced by a (ε,i)-arc. A relabeling of a dimension i into a dimension j in a node variable yields a (i,j)-loop. Thus, we can write the rule scheme of Figure 29 in Jerboa (with a slightly different syntax). The currently implemented syntactic analyzer uses more restrictive conditions to preserve the model's consistency but will still consider the rule scheme consistent.

Example 25 (Rule scheme instantiation for the quad subdivision operation). The left-hand side of $\mathscr{S}_G$ (Figure 29) contains a single node and arcs labeled (i,i) for each dimension i in $\{0,1,2\}$. Therefore, the instantiation with any pattern graph obtained via the $\{\varepsilon,0,1,2\}$-pattern functor on a 2-G-map yields a graph equal (up to isomorphism) to the initial 2-G-map. The rule scheme instantiates into a rule that is always applicable. In practice, the selection mechanism of Section 6.3 makes the scheme rule applicable on a connected component. Thus, the instantiation left-hand-side of the rule scheme $\mathscr{S}_G$ that contains a single node and three arcs produces a DPO-rule whose left-hand side contains 100680 nodes and 302040 arcs. Since the right-hand side of the scheme rule contains four nodes and 12 arcs (counting the non-oriented arcs as two arcs), the right-hand side of the instantiated DPO-rule contains 402720 nodes and 1208160 arcs. In practice, the application of such a scheme rule can be parallelized to speed up the computation time [37]. Likewise, the left-hand side of $\mathscr{S}_O$ (Figure 29) contains a single node and arcs labeled (i,i) for each dimension i in $\{1,2\}$. For the same reasons, the instantiation on a pattern graph obtained by the associated functor will always yield a complete connected component in any 2-O-map. Therefore, the instantiated rule is always applicable.

The rule scheme $\mathscr{S}_O$ of Figure 30 has non-loop arcs labels with $(w,2)$ where w is different from ε. Such a rule scheme is currently not supported in Jerboa (neither in the rule editor nor in the generic viewer that allows application of operations to objects).

10.2. Application example in topology-based geometric modeling

As explained in Section 3, n-G-maps and n-O-maps represent n-dimensional, or nD, objects. Topology-based modeling structures object by their cell subdivision and the adjacency relations between these cells. This subdivision represents the topological structure of the object. Any other type of information (e.g., the geometric shape) is called an embedding and attached to an orbit. Therefore, any modeling operation is defined by a topological transformation (e.g., subdivide a face) and an embedding transformation (e.g., translate a point). In this paper, we have only considered topological transformations.

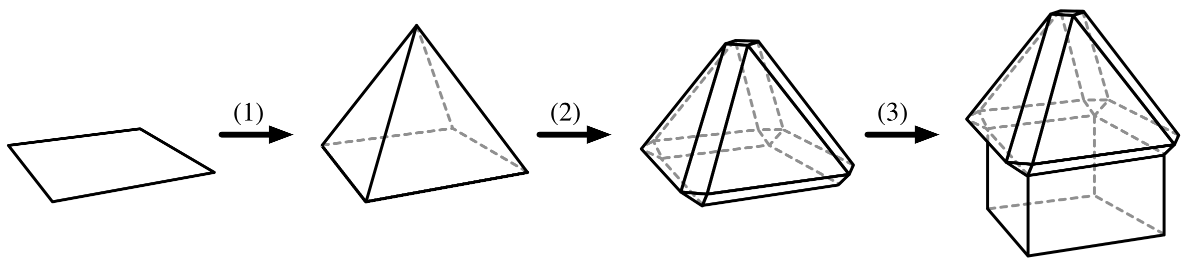


Figure 31: House modeling: (1) cone extrusion, (2) edge rounding, (3) face extrusion.

Although the work presented in this paper focuses on scheme rules dedicated to formalizing modeling operations, we also want to discuss the application of operations in the context of modeling an object. Modeled objects result from complex successive applications of several operations. Let us take the example of modeling a simplified house from Figure 31. From a square face, a pyramid is constructed by applying a cone extrusion operation. Topologically speaking, this operation creates as many faces as there are edges in the starting face (four edges in this case). All those faces are sewn along the starting face on a single point to create a cone. In a second step, all edges of the created volume are rounded, i.e., replaced by faces. Finally, the base of the house is constructed by applying an extrusion to the base square.

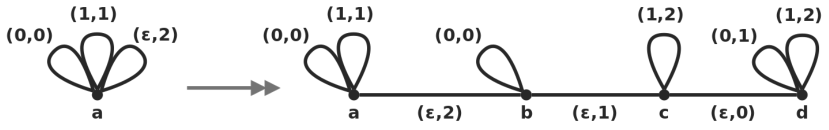
In geometric modeling, the most widespread practice consists of defining each operation algebraically before implementing it. The consistency preservation is hardly verified and comes with an expensive additional programming cost. Conversely, our approach takes care of operation genericity with the categorical product and the consistency preservation with the conditions on rules and rule schemes.

Let us use the whole pipeline of Figure 31 to illustrate operations. We will provide certain in 2D and others in 3D for both models. 2D operations only modify the outer surface on an object, making them easier to read and understand, whereas 3D operations also change the inner parts of objects.

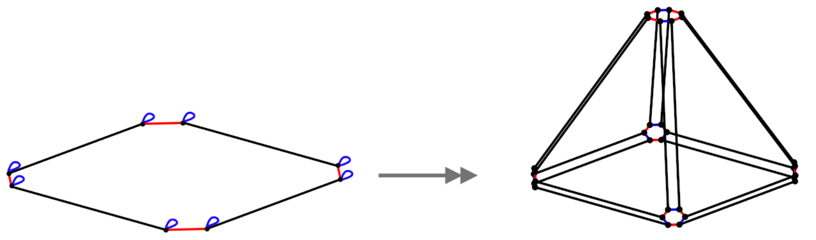
Cone extrusion. The cone extrusion adds a new topological vertex and a triangular face for each edge of the original face. In 2D, this operation is restrained to only be applicable to isolated faces, i.e., faces that belong to a non-volume. We build the new vertex as the dual of the original face. Each vertex of the original face is linked to this dual vertex, resulting in triangular faces.

In a 2-G-map, isolated faces are represented by orbits on the dimensions 0, 1, and 2, where 2-arcs are loops. The left-hand side of the rule scheme depicted in Figure 32a encodes such an orbit. The $(0,0)$-loops (resp.

$(1,1)$-loops) in the graph scheme represent 0-arcs (resp. 1-arcs) in the G-map whereas the $(\varepsilon,2)$-loops represent 2-loops. Let us provide a reading grid for the right-hand side of the rule. Node a corresponds to the initial face. Node d is the dual vertex where the initial 0-arcs in a have been replaced by 1-arcs and the 1-arcs by 2-arcs. The $(\varepsilon,0)$-arc between nodes c and d together with the $(1,2)$-loops on both nodes represent the edges between the face and the dual vertex. Node b and its $(0,0)$-loop stand for the edges of faces added to mirror the initial face and close the lateral triangular ones. Figure 32b is an instantiation of the rule scheme on a square face.



(a) Rule scheme for cone extrusion in a 2-G-map.



(b) Instantiated DPO rule on an hexagonal face.

Figure 32: Cone extrusion operation on a 2-G-map.

In a 2-O-map, an isolated face cannot be modeled as the model only supports volumes surrounded by closed surfaces. To bypass this limitation, we start with a flat volume made of two faces. This falt volume corresponds to an orbit on the dimensions 1 and 2, where the 1-arcs define two cycle graphs with reverse orientation. Such an orbit is illustrated by the left-hand side of the rule scheme in Figure 33a. Each of the reverse cycles is encoded by a node and its loop. The conjugated $\mathbb{W}$-labels on the loops of nodes a and b depict the reverse orientation of the cycles. In the right-hand side of the rule scheme, the $(\varepsilon,1)(\varepsilon,1)(\varepsilon,1)$-cycle acd correspond to the lateral faces while the $(1,2)$-arc from node c to node d represents the link between the lateral faces. Figure 33b gives the corresponding instantiated rule on a square face.

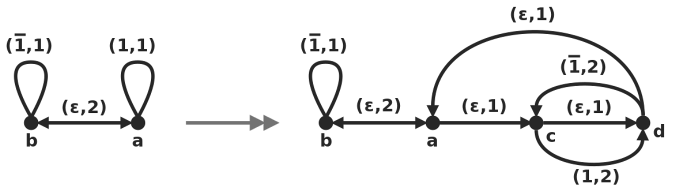
Edge rounding. The rule scheme of Figure 34a describes the edge rounding of a surface in 2D. We can use it with Jerboa's generic viewer to transform the pyramid of Figure 35a into the object of Figure 35c. The corresponding 2-G-maps are respectively given in Figures 35b and 35d.

The same operation is given for a 2-O-map with the rule scheme of Figure 34b where nodes b and c represent the face replacing the rounded edge, and node d corresponds to the face replacing the rounded vertex.

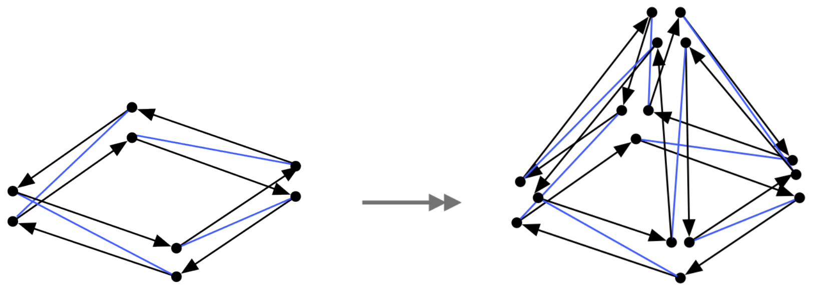
Face extrusion. Face extrusion is similar to cone extrusion but adds a dual face instead of a dual vertex. Each vertex from the initial face is linked to its copy in the dual face by an edge. Thereby, lateral faces are rectangles instead of triangles.

The rule scheme of Figure 36a describes face extrusion for a 3-G-map. Nodes b and g correspond to the face copies. The 3-arc between nodes a and b represents the link between the newly added volume and the original face. Nodes d and e along with their common $(\varepsilon,0)$-arc correspond to the edges that link the two bases of the prism. Nodes c and f represent the edges of faces from the bases that belong to the lateral faces. This rule scheme is applied on the square face of the pyramid from Figure 35c (here considered in 3D), yielding the 3-G-map of Figure 36b. The final object is illustrated in Figure 36c.

In a 3-O-map, face extrusion corresponds to the rule schemes of Figure 37. Nodes a, b and c have been respectively filed in red, blue, and green to be more easily retrievable in the right-hand side. On the left-hand side, node a matches the face to be extruded in the original volume, node b the same face as a, but in the external

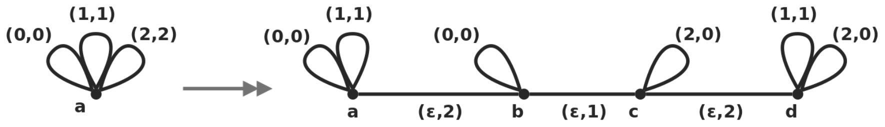


(a) Rule scheme for cone extrusion in a 2-O-map.

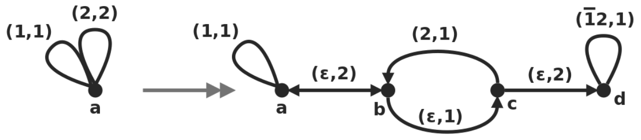


(b) Instantiated DPO rule on an hexagonal face.

Figure 33: Cone extrusion operation on a 2-O-map.

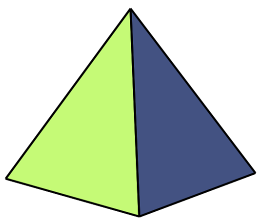


(a) For a 2-G-map.

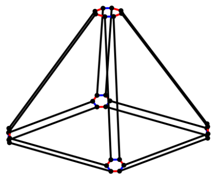


(b) For a 2-O-map.

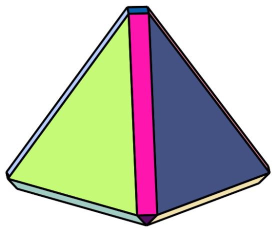
Figure 34: Rule schemes for edge rounding.



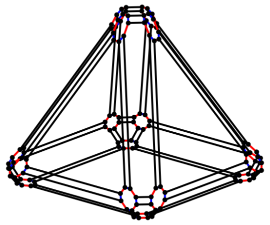
(a) Pyramid before edge rounding.



(b) Underlying 2-G-map.



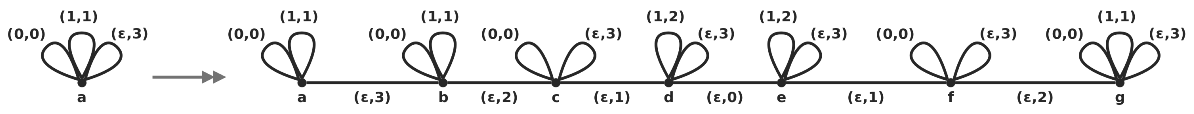
(c) Pyramid after edge rounding.



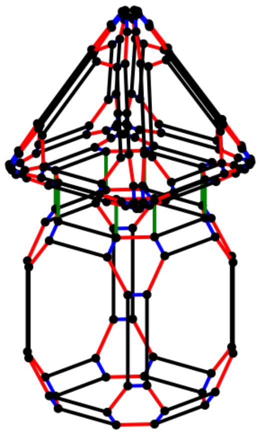
(d) Underlying 2-G-map.

Figure 35: Edge rounding of the pyramid.

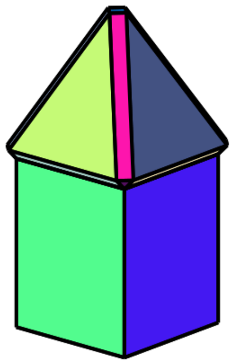
volume, node *c* the nodes in faces adjacent to the face corresponding to *b*. On the right-hand side, the extruded face *a* is now linked to the new face *d* of the extruded volume. The four nodes *e*, *f*, *g*, and *k* correspond to the side faces, and share 3-link with the four nodes *l*, *m*, *n* and *o* that represent the new side faces of the external volume. The side faces of the external volume are 2-linked to the green node *c* of the matched object, therefore reconnecting with the outer volume. Node *h* is the base face of the extruded volume, 3-linked to the matched blue face *b* of the external volume. Finally, both *h* and *b* are connected to the side faces with 2-arcs.



(a) Rule scheme for the operation of face extrusion in 3-G-map.



(b) 3-G-map obtained after extruding the face.



(c) Corresponding object.

Figure 36: Face extrusion.

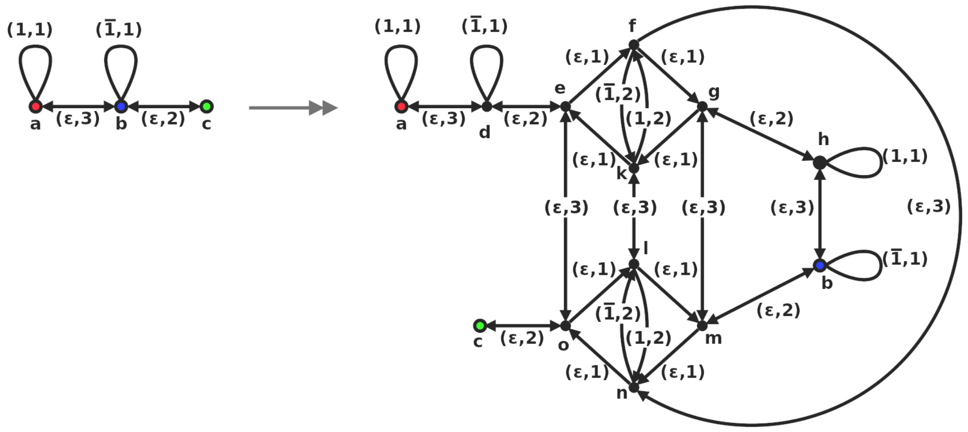


Figure 37: Rule scheme for the operation of face extrusion in a 3-O-map.

The rule scheme from Figure 37 is more complicated than its counterpart in the G-map model. Nonetheless, a formal description of the operation is precious to fulfill the need of the geometric modeling community, more likely to use the model of O-maps. Besides, consistency constraints on rule schemes help its writing. If one were to implement this topological operation in a programming language, it would result in a lot more development and debugging efforts.

We described a few operations in 2D and 3D for the models of G-maps and O-maps. Such topological operations, characteristics of geometric modelers, can easily be checked against the conditions provided previously in the article with the algorithms presented at the beginning of the Section.

11. Related works

Graph transformations were introduced to generalize Chomsky grammars to non-linear structures [38]. The algebraic approach to graph transformations uses production rules where the left-hand side and the right-hand side are graphs. The rules are applied using constructions from the theory of categories, namely single-pushout or double-pushout [24].

Formalizations based on graph transformations have been successfully achieved in many areas across computer science, such as database design [39], concurrent and distributed systems [40], software engineering [41], IT landscape modeling [42]. Essentially, graph transformations provide a safe framework to study the preservation of domain-related properties.

Sometimes, graph transformations do not allow for encapsulating a model or a theory completely. For instance, formal issues from a graph-based approach to geometric modeling cannot be solved directly with graph transformations. In such cases, the standard approach is to extend graph rewriting. To our knowledge, there are essentially two possibilities to conduct this extension: extend the category of graphs or extend the rewriting mechanism. Once the model is correctly defined within the adequate framework, the preservation of domain-related properties might still need to be answered. Application conditions [43] provide the most common approach for consistency preservation, as they provide conditions to prohibit a rule from being applicable if its application would violate a given constraint.

In this section, we review extensions of graph rewriting, their use for consistency preservation, and, finally, some applications of graph transformations to geometric modeling. Note that application conditions are solely discussed as a means for consistency preservation.

11.1. Rewriting in different categories

The first solution to extend graph transformations replaces the category of graphs with a more general one. Such a replacement allows for the rewriting of objects that are not directly expressed as graphs. The most straightforward addition is that of labels (on nodes, arcs, or both), resulting in the category of labeled graphs. Labeled graphs can be further structured with the help of a type graph that describes relations on arcs and nodes based on their label. Type graphs are used to slice the category of labeled graphs, leading to particular typed-graph categories [44]. Other kinds of high-level structures have also been used to replace graphs, such as hypergraphs or attributed graphs [45, 1]. In each of these generalizations, the rewriting process is expressed as a single-pushout or double-pushout in the appropriate categories. Instead of considering a new category in which rewriting is proven to be well-defined, another approach is to study minimal properties, ensuring that graph-like transformations are well-defined. This line of research led to high-level replacement (HLR) systems and grammars [46], adhesive categories [22], adhesive HLR categories [47]. Recently, a rewriting framework for abstract graphs (or objects from a topos) has been investigated in [48] to cover several previous works [49, 50, 51, 52]. A first requirement is that (DPO) rewriting is well defined. Furthermore, the rewriting needs to meet certain requirements to be of practical interest. These requirements include well-known properties of rewriting systems, such as the local Church-Rosser, parallelism, concurrency, and amalgamation theorems [53, 22, 47].

We believe that replacing the category of edge-labeled graphs with a category tailored to our models might not be practical. We could have considered the category of G-maps (or O-maps) or the less specific category of $\mathbb{D}$-combinatorial graphs. However, the left-hand and right-hand sides of rules would then belong to the category of $\mathbb{D}$-combinatorial graphs. Production rules would always need to modify the complete connected component because of the incident arc constraint. In [54], the authors explain how to achieve relabeling of totally labeled graphs with rules expressed using the category of partially labeled graphs. Their approach allowed for relabeling a node even if all its incident arcs were not matched. Similarly, we modify $\mathbb{D}$-combinatorial graphs with rules that are not expressed in the category of $\mathbb{D}$-combinatorial graphs. In [54], the authors add two conditions on rules to ensure that the result of the derivation is totally labeled. Likewise, we provided conditions on rules (in Section 4) to guarantee that we derive $\mathbb{D}$-combinatorial graphs. Essentially, despite the rule being expressed in the category of $\mathbb{D}$-graphs, we presented conditions certifying that the result of the direct derivation belongs to the category of $\mathbb{D}$-combinatorial graphs (more generally, the category of G-maps or O-maps).

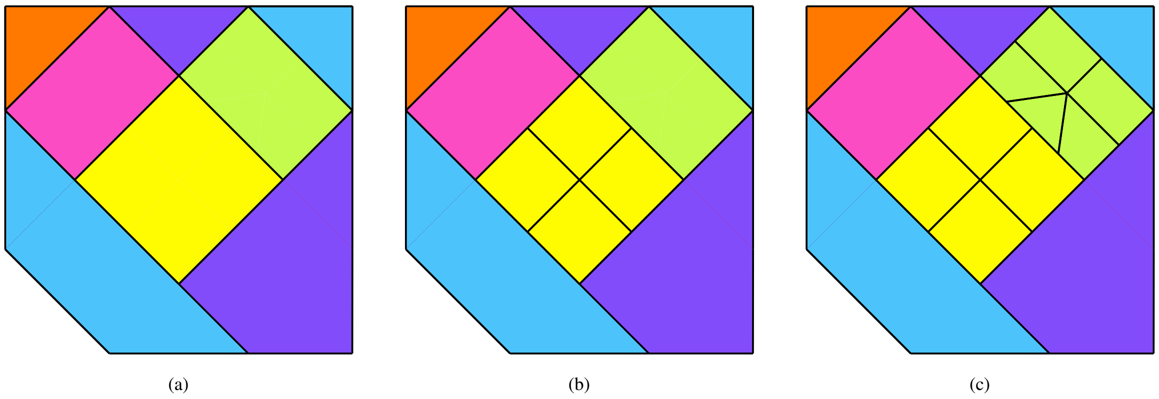


Figure 38: Sequential application of the face subdivision operation.

In Section 3, we discussed the possible generalization of the quad subdivision from a face to a surface, independent of the context and the underlying topology. This operation aims at subdividing surfaces for mesh refining. Each face of the mesh has to be subdivided. Since we are considering rewriting in the category of edge-labeled graphs, one could iteratively apply the operation on each face of the surface. Unfortunately, this approach would provide incorrect results as the subdivision of one face increased the number of edges on the surrounding faces, as explained in Example 26. Similarly, one could try to apply the operation on every face simultaneously. This approach is not conceivable because of the concurrent modification of edges.

In the sense of graph transformations, the transformations are neither sequentially nor parallel independent [38, 1, 55]. Parallel-dependent transformations can be amalgamated to a transformation thanks to the amalgamation rule [56, 55]. The computation of the amalgamation rule is done based on the transformations and principally used as a synchronization mechanism [57, 58]. Likewise, sequentially-dependent transformations can be reduced to a single transformation using the concurrency rule [59, 55]. One way to use these results would be to decompose a transformation into simpler transformations (and use the results to aggregate these simple transformations into more complex ones). For instance, one could have built a transformation for one node and then applied it to each node of the topological cell. Such a solution would fulfill the needs of a geometric modeler, i.e., specifying modeling operations that do not depend on the underlying topology of the object under modification. Nevertheless, this solution would be ill-suited as we need to link elements that belong to copies of different nodes from the topological cell (see Section 3.3). Besides, one considering oriented maps, we need to link different copies of different nodes (which led us to introduce path-related abstraction in the product construction). Indeed, the amalgamation and concurrency theorems focus on graph transformation systems, although we are interested in providing a rule editor with compact and straightforward operations. Were the subdivision rule to be computed as the amalgamated rule, it would have to be recomputed for each application. Instead, we want to generalize the transformations to abstract a part of the object's topology. The preservation of the model consistency can therefore be checked once, at design time, on the generalized rule. Nonetheless, such results could be a real asset for our framework at the level of rule schemes, i.e., to apply them systematically with a particular purpose. For example, these results would allow the application of several modeling operations simultaneously and, perhaps, even compose our scheme rules.

Example 26 (Invalid sequential application of the quad subdivision). A sequential application of the face subdivision application is shown in Figure 38. This sequence of operations is incorrect. Indeed, the edge between the yellow and green faces is split twice. First, when the yellow face is subdivided (Figure 38b), thus yielding two edges. Second, when these two new edges are split again, and the green face is subdivided. As a result, three vertices are added instead of one.

11.2. Different kind of rewriting

Our approach is closer to the second solution to extend graph transformations: twisting the rewriting mechanism. Several constructions with implicit or explicit pullbacks have been defined to enrich DPO rewriting with cloning. For instance, the Sesqui-Pushout approach [60] uses a final pullback complement as the left square in the rewriting step. The AGREE approach [61] relies on a morphism from the host graph to an enriched version of the left-hand side of the rule to specify how edges should be transformed. The Pullback-Pushout (PB-PO) approach [62] extends AGREE, capitalizing on two spans (intuitively two rules). The host graph is put in between the two spans of the left-hand side, yielding 4 commuting squares, one being a pullback and another being a pushout.

In AGREE [61], rules are local if they do not modify the strict complement of the left-hand side of the rule L in the host graph G. Therefore, even if cloning and merging are allowed, it can only occur in the image of L in G and the arcs incident to nodes in this image. The strict complement is defined as a pullback complement but can intuitively be understood as the biggest subgraph of G that contains no element from the image of L. Similarly, the PB-PO [62] approach can affect any element in the host graph G. This more general approach offers a more general definition of locality. This locality depends on a subgraph Γ of the left-hand side of the bottom span of the rule. Intuitively, a transformation is Γ-preserving if the transformation preserves this sub-object. However, the properties we want to preserve do not easily fit in either AGREE or PB-PO.

Our construction of product-based generalization of DPO via rule scheme can be seen as an attempt to combine enrichment with cloning and a categorical approach to relabeling. A categorical construction of relabeling in the DPO approach has already been realized in [63]. The authors control the ambiguity of label changes by epi-mono factorization and the use of least-upper-bound on the set of solutions. The incident arcs constraint of $\mathbb{D}$-combinatorial graphs removes that ambiguity. The relabeling only occurs on arcs in our specific framework while being entirely determined by the arc label. In that sense, arc relabeling coincides with a function on the label set. We showed that pair labels on arcs could encode such relabeling. The relabeling function is simulated by an embedding functor into the category where arcs are labeled with pairs, a product in this category, and a projecting functor back onto the category where arcs have a single label. Cloning and relabeling for G-maps have always been achieved in previous works [15, 16]. To deal with O-maps, we also need to consider paths. The navigational logic presented in [64, 65] allows reasoning about paths in a graph. However, this approach does not inherently support relabeling.

11.3. Consistency preservation in graph transformations

Twisting the rewriting mechanism allows for studying consistency preservation. For instance, application conditions hinder rule applicability, ensuring that the transformation of a well-formed object yields an equally well-formed object.

In general, such conditions are based on the existence (positive application condition) or non-existence (negative application condition, or NAC) of a given subgraph in the host graph. If this (non-)existence is fulfilled, the production rule can adequately be applied. As shown in [66], a colimit of the positive application conditions yields a rule applicable in the same context with only negative application conditions. Such conditions have proven helpful in many cases but too restrictive in others. They were extended to nested application conditions (or nested conditions) in [67] and shown to be expressively equivalent to first-order graph formulas, fulfilling the requirements presented in [68]. In [55], standard results were proven for rules with nested conditions. In particular, Local-Church Rosser, Parallelism, Concurrency, and Amalgamation theorems hold for rules with nested conditions in $\mathscr{M}$-adhesive categories (a weaker form of adhesivity restricted to a subclass of monomorphisms). Application conditions offer mechanisms to repair a rule (with or without condition) if its application to a graph satisfying a constraint does not preserve the constraint. One solution is to declare the rule inapplicable whenever its application would result in an ill-formed graph.

Thanks to static analysis of rules with nested application conditions, one can derive correct transformation systems.

A first solution is to demonstrate that the output satisfies a postcondition, provided the input satisfies a precondition, and construct a weakest precondition relative to the postcondition. The proof that the precondition implies the weakest precondition yields the correctness of the program [69]. The construction of weakest preconditions of a rule relative to a postcondition is discussed in [67] for nested application conditions; it uses shifts of conditions over rules and morphisms. Although each transformation and each condition are finite (i.e., are finite conjunction of conditions, each one being finitely nested), the actual complexity is never discussed. In particular, the resulting conditions showed as examples in [67] or in [55] seem to suggest that the complexity of the transformation is equivalent to the power set of the initial condition. Indeed, the shift over morphism considers all epimorphisms with the same domain. By duality, these epimorphisms essentially represent the category of sub-objects. Once the weakest precondition is obtained, one still has to show that the precondition on the graph implies the weakest precondition of the rule relative to the postcondition. Since nested conditions are equivalent to first-order formulas on graphs, this verification can be delegated to a solver. Following the work of [70, 71, 72] (handling a subset of nested constraints), the solver can be used as a tool to build counterexamples. In these works, invariants are checked during the evolution of a system based on a set of rules, while we are interested in generic rules that are sure to preserve the consistency of the model.

Another well-known method to check the correctness of a program with respect to a precondition and a postcondition is to provide a proof system and show its soundness with respect to the operational semantics [73]. In [74], the authors provide partial correctness of graph programs in the graph programming language GP [75] and show soundness with respect to the operational semantics of the language.

In [76], the authors developed an approach to bypass the loss of applicability of rules with application conditions. The authors guarantee that a constraint holds by construction after transformation. Their idea is to transform the rule into a multi-rule to fix every possible way it could introduce a violation. The formalization is done in adhesive categories for rules that only create structures and constraints[1] of the form $\forall(P, \exists C)$ where P is a sub-object of C. The authors provide sufficient conditions for correctness. Note that their approach is not directly transposable for our purposes since our rules may delete structures, and we need to express uniqueness as a constraint.

However, a rule (without condition) that can sometimes result in a well-formed graph and sometimes result in an ill-formed graph would be considered invalid in geometric modeling. In other words, a rule is valid if whenever there is a match from the left-hand side of the rule to a well-formed graph, the rule is both applicable and result in an equally well-formed graph. Let us consider the example of the gluing condition. It states that a rule is applicable whenever the match and the left morphism of the rule admits a pushout complement. When rewriting graphs, the conditions can be reduced to the dangling condition that essentially ensures that no unmatched arc loses its source or target. In the framework of nested application conditions, the gluing condition can be defined as a condition on the left-hand side of the rule. Thus, when the condition is not fulfilled, the rule is inapplicable. In our framework, the gluing condition can be checked on the rule (Fact 1). If L satisfies $\mathscr{I}_{(V_L \setminus V_R),L}(\mathbb{D})$, we are guaranteed that any match to a $\mathbb{D}$-combinatorial admits a pushout complement. Since we enriched DPO rewriting with a functorial approach and product construction, we needed to lift the consistency preservation from DPO-based rules to product-based rules. However, to our knowledge, there is no shift construction of nested application conditions for functors. As seen in Section 10, our very specific class of graphs ($\mathbb{D}$-combinatorial graphs) support verification by static analysis, which essentially resides in a graph traversal. Thus, every possible instantiation of a given rule scheme creates a consistency preserving production rule.

Our approach is similar to the one developed in [77] about consistent model transformations. In this article, the authors tackle consistency preservation in the ECLIPSE Modeling Framework (EMF). They provide a formalization of EMF model transformations that preserve EMF consistency (containment relations) with typed attributed graph transformations. This formalization revolves around six ad hoc conditions that define allowed transformations.

[1]In the sense of [76], the notation $\forall(P, \exists C)$ means that each occurrence of P lies within an occurrence of C

11.4. *Graph transformations applied to topology-based geometric modeling*

Numerous generic tools exploit the sound formalism of graphs transformations, such as PROGRES [78], AGG [79], Groove [80], and GrGen.NET [81]. Graph transformations modify structures at a low level, making rules self-contained and applicable with a single rule application engine. A single rule application engine minimizes development and debugging needs and enables the development of dedicated tools, such as Fujaba [82] to refactor code, DiaGen [83] to edit diagrams, GReTL [84] to manipulate metamodels, or Gremlin [85] to query graph databases.

In computer graphics as well, formalizations have been accomplished with graph transformations. For instance, graph grammars were used for 2D shapes classification [86], plants multiscale modeling [87], indoor scenes reconstruction [88], tunnels modeling process [89]. Previous works introduced a graph-based representation of G-maps for geometric modeling [15] and used DPO graph transformations to design modeling operations [16]. In this article, we dealt with the topological part of geometric modeling. The representation of objects also relies on other data, such as vertex positions, edge curvature, or volume density. In topology-based geometric modeling, these data are referred to as embedding. In [33], embeddings were defined as a family of node labels with appropriate consistency constraints and conditions. Since rules can modify both the topology and the embeddings of an object, conditions to preserve the embedding consistency in meta-rules were studied in [34]. In [15], DPO rewriting was enriched with variables to construct meta-rules instantiated with a set-based operation similar to a pullback. In [16], sufficient consistency preservation conditions were introduced for DPO transformations, and only the incident arcs condition was lifted to meta-rules.

Jerboa [17] has been developed, based on the work from [15] and [16]. This platform allows for the prototyping of dedicated modelers for G-maps. To design a modeler with Jerboa, one has to specify the dimensions of the objects (2D, 3D, 4D), the embeddings (position, curvature, density), and the rule-based transformations. Jerboa has been successively used for several modelers: one dedicated to architecture [90], one for the simulation of plant growth [91], and another one for the analysis of soils in geology [92]. In its current iteration, Jerboa supports the definition of modeler kernels using G-maps in any dimension. In this paper, we replaced the topological variables of [32] with a product construction. The new approach has a more substantial relationship with categorical concepts thanks to graph products, terminal graphs, and functors. Our scheme rules subsume previous works and allow us to deal with the more popular model of O-maps. Therefore, an evolution of Jerboa seems conceivable to support both G-maps and O-maps, as well as the product construction of rule schemes.

12. Conclusion

Towards defining topological operations on combinatorial maps, we studied three constraints on arc-labeled graphs: the incident arcs constraint, the non-orientation constraint, and the cycle constraint. We defined constraint-preserving DPO rules thanks to necessary and sufficient conditions allowing for rules to be statically checked. Thus, the study englobes both the G-map and the O-map models. To our knowledge, this is the first time that geometric modeling operations for O-maps are described using graph transformations.

Any framework dedicated to geometric modeling should be as unified as possible. Similar to certain implementations that allow several combinatorial models to coexist, this framework should encompass several combinatorial models. Hence, the unification of the G-maps and the O-maps in a single framework is already an exciting result to promote formal methods and, more specifically, graph transformations in geometric modeling.

We also enriched DPO rewriting for arc-labeled graphs with categorical constructions so that the designed rules are generic with respect to the underlying topology of the object. The extrapolation is based on functors to embed the transformation in a different category of labeled graphs. Then, we encode functions to describe how the operation should be carried out. Applying these functions is simulated with the categorical product to instantiate the more abstract rule schemes. We discussed how to unambiguously instantiate such rule schemes for a given combinatorial graph (and a fortiori for a given G-map or O-map). We also lifted the consistency conditions from DPO rules to rule schemes, although we lost the necessity of the cycle condition. We provided algorithms describing a static analyzer for consistency preservation. Finally, we gave a representative example of a pipeline for the modeling of a geometric object.

As a result of our new approach, closer to algebraic transformation formalism, we plan to extend the current version of the Jerboa platform. In order to be able to develop such a new platform, we will first need to provide conditions to preserve the embedding consistency, i.e., ensure that the transformation of a geometrically correct object yields an equally well-formed object.

References

[1] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Graphs, Typed Graphs, and the Gluing Construction, in: Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series), Springer, 2006, pp. 21–35. doi:10.1007/3-540-31188-2_2.

[2] S. Marschner, P. Shirley, Fundamentals of Computer Graphics, CRC Press, 2015.

[3] I. Baran, J. Popović, Automatic rigging and animation of 3D characters, ACM Transactions on Graphics 26 (3) (2007) 72–es. doi:10.1145/1276377.1276467.

[4] M. Perrin, J.-F. Rainaud, Shared earth modeling: knowledge driven solutions for building and managing subsurface 3D geological models, Editions Technip, 2013.

[5] S. Horna, D. Meneveaux, G. Damiand, Y. Bertrand, Consistency constraints and 3D building reconstruction, Computer-Aided Design (CAD) 41 (1) (2009) 13–27. doi:10.1016/j.cad.2008.11.006.

[6] J. Wu, R. Westermann, C. Dick, Physically-based Simulation of Cuts in Deformable Bodies: A Survey, in: S. Lefebvre, M. Spagnuolo (Eds.), Eurographics 2014 - State of the Art Reports, The Eurographics Association, 2014, pp. 1–19. doi:10.2312/egst.20141033.

[7] F. Ben Salah, H. Belhaouari, A. Arnould, P. Meseure, A Modular Approach Based on Graph Transformation to Simulate Tearing and Fractures on Various Mechanical Models, Journal of WSCG 25 (1) (2017) 39–48.

[8] S. Campagna, L. Kobbelt, H.-P. Seidel, Directed Edges—A Scalable Representation for Triangle Meshes, Journal of Graphics Tools 3 (4) (1998) 1–11. doi:10.1080/10867651.1998.10487494.

[9] K. Weiler, Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments, IEEE Computer Graphics and Applications 5 (1) (1985) 21–40. doi:10.1109/MCG.1985.276271.

[10] P. Lienhardt, Topological models for boundary representation: a comparison with n-dimensional generalized maps, Computer-Aided Design 23 (11) (1991) 59–82. doi:10.1016/0010-4485(91)90100-B.

[11] G. Damiand, P. Lienhardt, Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing, CRC Press, 2014.

[12] A. Vince, Combinatorial maps, Journal of Combinatorial Theory, Series B 34 (1) (1983) 1–21. doi:10.1016/0095-8956(83)90002-3.

[13] S. K. Lando, A. K. Zvonkin, Constellations, coverings, and maps, in: Graphs on Surfaces and Their Applications, Encyclopaedia of Mathematical Sciences, Springer, 2004, pp. 7–77. doi:10.1007/978-3-540-38361-1_2.

[14] P. Lienhardt, Subdivisions of N-dimensional Spaces and N-dimensional Generalized Maps, in: Proceedings of the Fifth Annual Symposium on Computational Geometry, SCG '89, Association for Computing Machinery, New York, NY, USA, 1989, pp. 228–236. doi:10.1145/73833.73859.

[15] M. Poudret, J.-P. Comet, P. Le Gall, A. Arnould, P. Meseure, Topology-based geometric modelling for biological cellular processes, in: International Conference on Language and Automata Theory and Applications, 2007, pp. 497–508.

[16] M. Poudret, A. Arnould, J.-P. Comet, P. Le Gall, Graph transformation for topology modelling, in: H. Ehrig, R. Heckel, G. Rozenberg, G. Taentzer (Eds.), Graph Transformations (ICGT 2008), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 147–161. doi:10.1007/978-3-540-87405-8_11.

[17] H. Belhaouari, A. Arnould, P. Le Gall, T. Bellet, Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling, in: H. Giese, B. König (Eds.), Graph Transformation (ICGT 2014), Lecture Notes in Computer Science, Springer International Publishing, Cham, 2014, pp. 269–284. doi:10.1007/978-3-319-09108-2_18.

[18] C. Dehlinger, J.-F. Dufourd, Formal specification and proofs for the topology and classification of combinatorial surfaces, Computational Geometry 47 (9) (2014) 869–890. doi:10.1016/j.comgeo.2014.04.007.

[19] P. Kraemer, L. Untereiner, T. Jund, S. Thery, D. Cazier, CGoGN: n-dimensional Meshes with Combinatorial Maps, in: J. Sarrate, M. Staten (Eds.), Proceedings of the 22nd International Meshing Roundtable, Springer International Publishing, Cham, 2014, pp. 485–503. doi:10.1007/978-3-319-02335-9_27.

[20] B. König, D. Nolte, J. Padberg, A. Rensink, A Tutorial on Graph Transformation, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 83–104. doi:10.1007/978-3-319-75396-6_5.

[21] S. Vigna, A Guided Tour in the Topos of Graphs, Tech. rep. (Jun. 2003).
URL http://arxiv.org/abs/math/0306394

[22] S. Lack, P. Sobociński, Adhesive Categories, in: I. Walukiewicz (Ed.), Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, Springer, 2004, pp. 273–288. doi:10.1007/978-3-540-24727-2_20.

[23] A. Habel, J. Müller, D. Plump, Double-pushout graph transformation revisited, Mathematical Structures in Computer Science 11 (5) (2001) 637–688. doi:10.1017/S0960129501003425.

[24] H. Ehrig, M. Korff, M. Löwe, Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science, Vol. 532 of Lecture Notes in Computer Science, Springer, 1991, pp. 24–37. doi:10.1007/BFb0017375.

[25] J. Engelfriet, G. Rozenberg, Node replacement graph grammars, in: Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations, World Scientific, 1997, pp. 1–94. doi:10.5555/278918.278925.

[26] M. Bauderon, A uniform approach to graph rewriting: The pullback approach, in: M. Nagl (Ed.), Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer, 1995, pp. 101–115. doi:10.1007/3-540-60618-1_69.

[27] P. Kraemer, D. Cazier, D. Bechmann, Extension of half-edges for the representation of multiresolution subdivision surfaces, The Visual Computer 25 (2) (2009) 149–163. doi:10.1007/s00371-008-0211-6.

[28] C. J. Paulus, L. Untereiner, H. Courtecuisse, S. Cotin, D. Cazier, Virtual cutting of deformable objects based on efficient topological operations, The Visual Computer 31 (6) (2015) 831–841. doi:10.1007/s00371-015-1123-x.

[29] L. Untereiner, P. Kraemer, D. Cazier, D. Bechmann, CPH: a compact representation for hierarchical meshes generated by primal refinement, Computer Graphics Forum 34 (8) (2015) 155–166. doi:10.1111/cgf.12667.

[30] G. Damiand, F. Zara, Merge-and-simplify operation for compact combinatorial pyramid definition, Pattern Recognition Letters 129 (2020) 48–55. doi:10.1016/j.patrec.2019.11.009.

[31] J.-F. Dufourd, An Intuitionistic Proof of a Discrete Form of the Jordan Curve Theorem Formalized in Coq with Combinatorial Hypermaps, Journal of Automated Reasoning 43 (1) (2009) 19–51. doi:10.1007/s10817-009-9117-x.

[32] T. Bellet, M. Poudret, A. Arnould, L. Fuchs, P. Le Gall, Designing a Topological Modeler Kernel: A Rule-Based Approach, in: 2010 Shape Modeling International Conference, 2010, pp. 100–112. doi:10.1109/SMI.2010.31.

[33] T. Bellet, A. Arnould, P. Le Gall, Rule-based transformations for geometric modeling, in: R. Echahed (Ed.), 6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), Vol. 48, Saarbrücken, Germany, 2011, p. 20–37. doi:10.4204/eptcs.48.5.

[34] T. Bellet, A. Arnould, H. Belhaouari, P. Le Gall, Geometric modeling: Consistency preservation using two-layered variable substitutions, in: J. de Lara, D. Plump (Eds.), Graph Transformation (ICGT 2017), Lecture Notes in Computer Science, Springer, Cham, 2017, pp. 36–53. doi:10.1007/978-3-319-61470-0_3.

[35] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, D. Zorin, Quad-Mesh Generation and Processing: A Survey, Computer Graphics Forum 32 (6) (2013) 51–76. doi:10.1111/cgf.12014.

[36] R. V. Book, F. Otto, String-Rewriting Systems, Text and Monographs in Computer Science, Springer, New York, NY, 1993.

[37] P. Bourquat, H. Belhaouari, P. Meseure, V. Gauthier, A. Arnould, Transparent parallelization of enrichment operations in geometric modeling, in: K. Bouatouch, A. A. de Sousa, J. Braz (Eds.), Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2020, Volume 1: GRAPP, Valletta, Malta, February 27-29, 2020, SCITEPRESS, 2020, pp. 125–136. doi:10.5220/0008965701250136.

[38] H. Ehrig, Introduction to the algebraic theory of graph grammars (a survey), in: V. Claus, H. Ehrig, G. Rozenberg (Eds.), Graph-Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science, Springer, 1979, pp. 1–69. doi:10.1007/BFb0025714.

[39] M. Gyssens, J. Paredaens, J. van den Bussche, D. van Gucht, A graph-oriented object database model, IEEE Transactions on Knowledge and Data Engineering 6 (4) (1994) 572–586. doi:10.1109/69.298174.

[40] H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation: Concurrency, Parallelism, and Distribution, Vol. 3 of Concurrency, Parallelism, and Distribution, World Scientific, 1999.

[41] L. Baresi, R. Heckel, Tutorial Introduction to Graph Transformation: A Software Engineering Perspective, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Graph Transformations (ICGT 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 431–433. doi:10.1007/978-3-540-30203-2_30.

[42] M. Haeusler, T. Trojer, J. Kessler, M. Farwick, E. Nowakowski, R. Breu, ChronoSphere: a graph-based EMF model repository for IT landscape models, Software and Systems Modeling 18 (6) (2019) 3487–3526. doi:10.1007/s10270-019-00725-0.

[43] H. Ehrig, A. Habel, Graph Grammars with Application Conditions, in: G. Rozenberg, A. Salomaa (Eds.), The Book of L, Springer, 1986, pp. 87–100. doi:10.1007/978-3-642-95486-3_7.

[44] A. Corradini, U. Montanari, F. Rossi, Graph processes, Fundamenta Informaticae 26 (3-4) (1996) 241–265. doi:10.3233/FI-1996-263402.

[45] M. Löwe, M. Korff, A. Wagner, An algebraic framework for the transformation of attributed graphs, in: R. Sleep, R. Plasmeijer, M. van Eekelen (Eds.), Term Graph Rewriting: Theory and Practice, John Wiley and Sons Ltd., 1993, pp. 185–199.

[46] H. Ehrig, A. Habel, H.-J. Kreowski, F. Parisi-Presicce, From graph grammars to high level replacement systems, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science, Vol. 532 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1991, pp. 269–291. `doi:10.1007/BFb0017395`.

[47] H. Ehrig, A. Habel, J. Padberg, U. Prange, Adhesive High-Level Replacement Categories and Systems, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Graph Transformations (ICGT 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 144–160. `doi:10.1007/978-3-540-30203-2_12`.

[48] A. Corradini, T. Heindel, B. König, D. Nolte, A. Rensink, Rewriting Abstract Structures: Materialization Explained Categorically, in: M. Bojańczyk, A. Simpson (Eds.), Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2019, pp. 169–188. `doi:10.1007/978-3-030-17127-8_10`.

[49] J. Bauer, R. Wilhelm, Static Analysis of Dynamic Communication Systems by Partner Abstraction, in: H. R. Nielson, G. Filé (Eds.), Static Analysis, Vol. 4634 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2007, pp. 249–264. `doi:10.1007/978-3-540-74061-2_16`.

[50] A. Rensink, E. Zambon, Neighbourhood Abstraction in GROOVE - Tool Paper, in: J. de Lara, D. Varro (Eds.), Proceedings of the Fourth International Workshop on Graph-Based Tools (GraBaTs 2010), CTIT Workshop Proceedings, Centre for Telematics and Information Technology (CTIT), 2010, pp. 55–61.

[51] D. Steenken, H. Wehrheim, D. Wonisch, Sound and Complete Abstract Graph Transformation, in: A. Simao, C. Morgan (Eds.), Formal Methods, Foundations and Applications, Vol. 7021 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 92–107. `doi:10.1007/978-3-642-25032-3_7`.

[52] P. Backes, J. Reineke, Analysis of Infinite-State Graph Transformation Systems by Cluster Abstraction, in: D. D'Souza, A. Lal, K. G. Larsen (Eds.), Verification, Model Checking, and Abstract Interpretation, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2015, pp. 135–152. `doi:10.1007/978-3-662-46081-8_8`.

[53] R. Heckel, J. M. Küster, G. Taentzer, Confluence of Typed Attributed Graph Transformation Systems, in: A. Corradini, H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Transformation (ICGT 2002), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002, pp. 161–176. `doi:10.1007/3-540-45832-8_14`.

[54] A. Habel, D. Plump, Relabelling in Graph Transformation, in: A. Corradini, H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Transformation (ICGT 2002), Vol. 2505 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002, pp. 135–147. `doi:10.1007/3-540-45832-8_12`.

[55] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas, M-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation, Mathematical Structures in Computer Science 24 (4) (Aug. 2014). `doi:10.1017/S0960129512000357`.

[56] P. Boehm, H.-R. Fonio, A. Habel, Amalgamation of graph transformations: A synchronization mechanism, Journal of Computer and System Sciences 34 (2) (1987) 377–408. `doi:10.1016/0022-0000(87)90030-4`.

[57] G. Taentzer, M. Beyer, Amalgamated graph transformations and their use for specifying AGG — an algebraic graph grammar system, in: H. J. Schneider, H. Ehrig (Eds.), Graph Transformations in Computer Science, Lecture Notes in Computer Science, Springer, 1994, pp. 380–394. `doi:10.1007/3-540-57787-4_24`.

[58] A. Rensink, J.-H. Kuperus, Repotting the Geraniums: On Nested Graph Transformation Rules, Electronic Communications of the EASST 18 (Sep. 2009). `doi:10.14279/tuj.eceasst.18.260`.

[59] H. Ehrig, B. K. Rosen, Parallelism and concurrency of graph manipulations, Theoretical Computer Science 11 (3) (1980) 247–275. doi:10.1016/0304-3975(80)90016-X.

[60] A. Corradini, T. Heindel, F. Hermann, B. König, Sesqui-pushout rewriting, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.), Graph Transformations (ICGT 2006), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 30–45. doi:10.1007/11841883_4.

[61] A. Corradini, D. Duval, R. Echahed, F. Prost, L. Ribeiro, Algebraic graph rewriting with controlled embedding, Theoretical Computer Science 802 (2020) 19–37. doi:10.1016/j.tcs.2019.06.004.

[62] A. Corradini, D. Duval, R. Echahed, F. Prost, L. Ribeiro, The PBPO graph transformation approach, Journal of Logical and Algebraic Methods in Programming 103 (2019) 213–231. doi:10.1016/j.jlamp.2018.12.003.

[63] H. J. Schneider, Changing Labels in the Double-Pushout Approach Can Be Treated Categorically, in: H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, G. Taentzer (Eds.), Formal Methods in Software and Systems Modeling: Essays Dedicated to Hartmut Ehrig on the Occasion of His 60th Birthday, Vol. 3393 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 134–149. doi:10.1007/978-3-540-31847-7_8.

[64] L. Lambers, M. Navarro, F. Orejas, E. Pino, Towards a Navigational Logic for Graphical Structures, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 124–141. doi:10.1007/978-3-319-75396-6_7.

[65] M. Navarro, F. Orejas, E. Pino, L. Lambers, A navigational logic for reasoning about graph properties, Journal of Logical and Algebraic Methods in Programming 118 (Jan. 2021). doi:10.1016/j.jlamp.2020.100616.

[66] A. Habel, R. Heckel, G. Taentzer, Graph Grammars with Negative Application Conditions, Fundamenta Informaticae 26 (3) (1996) 287–313. doi:10.3233/FI-1996-263404.

[67] A. Habel, K.-H. Pennemann, Correctness of high-level transformation systems relative to nested conditions, Mathematical Structures in Computer Science 19 (2) (2009) 245–296. doi:10.1017/S0960129508007202.

[68] A. Rensink, Representing First-Order Logic Using Graphs, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Graph Transformations (ICGT 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 319–335. doi:10.1007/978-3-540-30203-2_23.

[69] E. W. Dijkstra, A Discipline of Programming, Prentice-Hall, 1976.

[70] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling, Symbolic invariant verification for systems with dynamic structural adaptation, in: Proceedings of the 28th international conference on Software engineering, ICSE '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 72–81. doi:10.1145/1134285.1134297.

[71] B. Becker, L. Lambers, J. Dyck, S. Birth, H. Giese, Iterative Development of Consistency-Preserving Rule-Based Refactorings, in: J. Cabot, E. Visser (Eds.), Theory and Practice of Model Transformations, Vol. 6707 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 123–137. doi:10.1007/978-3-642-21732-6_9.

[72] J. Dyck, H. Giese, Inductive Invariant Checking with Partial Negative Application Conditions, in: F. Parisi-Presicce, B. Westfechtel (Eds.), Graph Transformation (ICGT 2015), Lecture Notes in Computer Science, Springer International Publishing, Cham, 2015, pp. 237–253. doi:10.1007/978-3-319-21145-9_15.

[73] C. A. R. Hoare, An axiomatic basis for computer programming, Communications of the ACM 12 (10) (1969) 576–580, 583. doi:10.1145/363235.363259.

[74] C. M. Poskitt, D. Plump, Hoare-Style Verification of Graph Programs, Fundamenta Informaticae 118 (1-2) (2012) 135–175. doi:10.3233/FI-2012-708.

[75] D. Plump, The Graph Programming Language GP, in: S. Bozapalidis, G. Rahonis (Eds.), Algebraic Informatics, Vol. 5725 of Lecture Notes in Computer Science, Springer, 2009, pp. 99–122. doi:10.1007/978-3-642-03564-7_6.

[76] J. Kosiol, L. Fritsche, N. Nassar, A. Schürr, G. Taentzer, Constructing Constraint-Preserving Interaction Schemes in Adhesive Categories, in: J. L. Fiadeiro, I. Ţuţu (Eds.), Recent Trends in Algebraic Development Techniques, Vol. 11563 of Lecture Notes in Computer Science, 2019, pp. 139–153. doi:10.1007/978-3-030-23220-7_8.

[77] E. Biermann, C. Ermel, G. Taentzer, Formal foundation of consistent EMF model transformations by algebraic graph transformation, Software & Systems Modeling 11 (2) (2012) 227–250. doi:10.1007/s10270-011-0199-7.

[78] A. Schürr, A. J. Winter, A. Zündorf, Graph grammar engineering with PROGRES, in: W. Schäfer, P. Botella (Eds.), Software Engineering — ESEC '95, Lecture Notes in Computer Science, 1995, pp. 219–234. doi:10.1007/3-540-60406-5_17.

[79] G. Taentzer, AGG: A Graph Transformation Environment for Modeling and Validation of Software, in: J. L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science, 2004, pp. 446–453. doi:10.1007/978-3-540-25959-6_35.

[80] A. Rensink, The GROOVE Simulator: A Tool for State Space Generation, in: J. L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science, 2004, pp. 479–485. doi:10.1007/978-3-540-25959-6_40.

[81] R. Geiß, G. V. Batz, D. Grund, S. Hack, A. Szalkowski, GrGen: A Fast SPO-Based Graph Rewriting Tool, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.), Graph Transformations (ICGT 2006), Lecture Notes in Computer Science, 2006, pp. 383–397. doi:10.1007/11841883_27.

[82] U. Nickel, J. Niere, A. Zündorf, The FUJABA environment, in: Proceedings of the 22nd international conference on Software engineering, ICSE '00, Association for Computing Machinery, 2000, pp. 742–745. doi:10.1145/337180.337620.

[83] M. Minas, G. Viehstaedt, DiaGen: a generator for diagram editors providing direct manipulation and execution of diagrams, in: Proceedings of Symposium on Visual Languages, 1995, pp. 203–210. doi:10.1109/VL.1995.520810.

[84] J. Ebert, T. Horn, GReTL: an extensible, operational, graph-based transformation language, Software and Systems Modeling 13 (1) (2014) 301–321. doi:10.1007/s10270-012-0250-3.

[85] M. A. Rodriguez, The Gremlin graph traversal machine and language (invited talk), in: Proceedings of the 15th Symposium on Database Programming Languages, DBPL 2015, 2015, pp. 1–10. doi:10.1145/2815072.2815073.

[86] K. C. You, K.-S. Fu, A Syntactic Approach to Shape Recognition Using Attributed Grammars, IEEE Transactions on Systems, Man, and Cybernetics 9 (6) (1979) 334–345. doi:10.1109/TSMC.1979.4310222.

[87] Y. Ong, W. Kurth, A graph model and grammar for multi-scale modelling using XL, in: 2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops, IEEE, 2012, pp. 1–8. doi:10.1109/BIBMW.2012.6470293.

[88] S. Ikehata, H. Yang, Y. Furukawa, Structured Indoor Modeling, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), IEEE, 2015, pp. 1323–1331. doi:10.1109/ICCV.2015.156.

[89] S. Vilgertshofer, A. Borrmann, Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models, Advanced Engineering Informatics 33 (2017) 502–515. doi:10.1016/j.aei.2017.07.003.

[90] A. Cardot, D. Marcheix, X. Skapin, A. Arnould, H. Belhaouari, Persistent naming based on graph transformation rules to reevaluate parametric specification, Computer-Aided Design and Applications 16 (5) (2019) 985–1002. doi:10.14733/cadaps.2019.985-1002.

[91] E. Bohl, O. Terraz, D. Ghazanfarpour, Modeling fruits and their internal structure using parametric 3gmap l-systems, The Visual Computer 31 (6) (2015) 819–829. doi:10.1007/s00371-015-1108-9.

[92] V. Gauthier, Développement d'un langage de programmation dédié à la modélisation géométrique à base topologique, application à la reconstruction de modèles géologiques 3D, These de doctorat, Poitiers, https://www.theses.fr/2019POIT2252 (Jan. 2019).