HAL open science

Reversibility and composition of rewriting in hierarchies

Russ Harmer, Eugenia Oshurko

To cite this version:

Russ Harmer, Eugenia Oshurko. Reversibility and composition of rewriting in hierarchies. EPTCS, 2020. hal-02869865v2

HAL Id: hal-02869865

https://hal.science/hal-02869865v2

Submitted on 29 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reversibility and Composition of Rewriting in Hierarchies

Russ Harmer
Univ Lyon, EnsL, UCBL, CNRS, LIP
F-69342 LYON Cedex 07, France
russell.harmer@ens-lyon.fr

Eugenia Oshurko
Univ Lyon, EnsL, UCBL, CNRS, LIP
F-69342 LYON Cedex 07, France
ievgeniia.oshurko@ens-lyon.fr

In this paper, we study how graph transformations based on sesqui-pushout rewriting can be reversed and how the composition of rewrites can be constructed. We illustrate how such reversibility and composition can be used to design an audit trail system for individual graphs and graph hierarchies. This provides us with a compact way to maintain the history of updates of an object, including its multiple versions. The main application of the designed framework is an audit trail of updates to knowledge represented by hierarchies of graphs. Therefore, we introduce the notion of rule hierarchy that represents a transformation of the entire hierarchy, study how rule hierarchies can be applied to hierarchies and analyse the conditions under which this application is reversible. We then present a theory for constructing the composition of consecutive hierarchy rewrites. The prototype audit trail system for transformations in hierarchies of simple graphs with attributes is implemented as part of the ReGraph Python library.

1 Introduction

The main goal of this work is to design a system for maintaining an *audit trail* for the knowledge representation (KR) framework based on hierarchies of graphs presented in [16]. We would like to use this system to record the history of predominantly small localized updates of a large knowledge corpus represented with graphs and graph hierarchies, where storing the corpus at each point in the history is not feasible. The aforementioned requirement for the transformations to be reversible in the context of our practical application does not restrict the set of operations that can be performed for updating a corpus, but rather introduces a reasonably small overhead on storing additional information that allows us to revert the applied transformations. Furthermore, we design means for maintaining multiple versions of the corpus, crucial when accommodating different versions of knowledge that, for example, correspond to different view-points of knowledge curators or some intrinsic knowledge conflicts (e.g. contradicting experimental results, alternative hypotheses). The main use cases of audit trails for KRs based on hierarchies of graphs include version control for updates in schema-aware graph databases presented in [4] and updates to the knowledge corpora provided by the bio-curation framework KAMI [15].

An audit trail system with the desired capabilities heavily relies on: (1) the existence of an *efficient semantic representation of object transformations* (or deltas), which frees us from the necessity to store the state of the object at each point of its transformation history; (2) the *reversibility of transformations*, which guarantees that any sequence of transformations can be 'undone'; and (3) the existence of sound means for *composing a pair of successive transformations*, which allows us to efficiently store and switch between different versions of the same object. In this work we formulate the three above-mentioned ingredients with respect to both individual objects (Section 2), for instance graphs, and hierarchies of objects (Section 3) and discuss how these ingredients can be used to construct an efficient audit trail (Section 4).

B. Hoffmann and M. Minas (Eds.): Eleventh International Workshop on Graph Computation Models (GCM 2020)
EPTCS 330, 2020, pp. 145–162, doi:10.4204/EPTCS.330.9

© R. Harmer and E. Oshurko
This work is licensed under the Creative Commons Attribution License.

We present a mathematical framework for building a traceable history of transformations based on the sesqui-pushout (SqPO) approach [7] operating on individual objects and hierarchies of objects from appropriately structured categories. A transparent audit trail provides insight into the history of object transformations and allows us to revert to an arbitrary point in this history. The latter feature is extremely useful, when, for example, trying to fix an erroneous transformation. Moreover, such an audit trail provides means for efficient accommodation of multiple versions of the same object arising, for example, as the result of inconsistent transformations.

Related work

SqPO rewriting is a generalization of the double (DPO) and single (SPO) pushout approaches to graph transformation that allows for the cloning of graph elements in addition to the familiar operations of adding, deleting and merging of graph elements. While the reversibility of SqPO rewriting has been previously studied [9], the composition of consecutive (and not necessarily sequentially independent) SqPO rewrites has been less fully developed. The construction of such compositions has been given for the DPO approach (see D-concurrent productions in [12]) and for special cases of the SqPO approach (where rules are linear [3] and where the right-hand side of the first applied rule is exactly the left-hand side of the second rule [21]). In this work, we present such a construction for two consecutive applications of general SqPO rules, where the first is required to be reversible. This was inspired by [12] and adapted to SqPO rewriting. For the composition to be well-defined, we require the underlying category to have structure allowing for well-behaved generalized unions of objects (more formally, we require for our category to be adhesive [19]). The notion of rule hierarchy and the question of the reversibility and composition of hierarchy rewrites represent a novel direction that generalizes SqPO rewriting of individual objects to hierarchies of objects [16].

The main application of interest to us, a transformation audit trail, is closely related to the version control systems (VCSs) used in software development. While such systems typically provide control over different versions of software source code (represented by text files), our audit trail provides such control for different versions of a graph or a graph hierarchy. Similarly to VCSs, the transformation audit trail avoids maintaining the state of an object at the time of every transformation by keeping only its current state together with a compact representation of a history of transformations. Moreover, by using a mechanism similar to *delta compression* in VCSs, such audit trails allow for the maintenance of multiple versions of the same object and for switching between these versions.

The versioning of diagrams expressing object-oriented data modes in the domain of model-driven engineering is closely related to audit trails for graph transformations because such model diagrams can be seen as graphs with attributes. However, work in this domain [24, 2] typically considers the operations of addition and deletion of graph elements (including attributes), to express changes to the data model, but largely omits those of cloning and merging, as these operations are hard to interpret in the context of data modelling, whereas these operations are highly interesting in the context of KR.

There exists an extensive body of work on versioning for database systems [23], including graph databases[1] [5, 17]. Such systems are usually divided into snapshot- and delta-based classes (also called copy and log), where the first store snapshots of the entire database at different points of the revision history, while the second preserve the transformation log. In this context, our system falls into a rather hybrid category: while the audit trail stores a representations of transformations (logs), these logs themselves consists of subgraphs, i.e. graph patterns that are affected as the result of transformation.

[1] Versioning for the Neo4j database `https://github.com/h-omer/neo4j-versioner-core`

A different classification of versioning systems for graph databases divides them into those that provide the revision history from the point of view of individual nodes and edges (or relationships), and those that use the graph point of view [5]. While existing tools, such as `neo4j-versioner-core` or the one presented in [5], mostly tackle versioning from the point of view of individual elements, our framework allows us to version the entire graph and even the hierarchy of graphs. Moreover, as is the case with model versioning, existing database versioning techniques do not allow to record the operations of cloning and merging of graph elements.

2 Preliminaries

In this section we briefly present some useful notions that serve as preliminaries for the rest of the paper and allow us to construct the desired audit trail for SqPO rewriting of individual objects. We introduce SqPO rewriting, its reversible version and add the audit trail ingredient by presenting in Section 2.2 how the composition of two consecutive SqPO rewrites can be constructed when the first rewrite is reversible. Finally, we conclude this section by presenting the principal KR model of interest, a hierarchy of objects. The preliminaries, as well as the other technical parts of the paper, are presented using the rather formal notions of pullback, pushout and pullback complement originating from category theory, which can be thought of as generalized intersection, union and difference of objects (e.g. graphs) respectively.

2.1 SqPO Rewriting

SqPO rewriting is an approach to abstract deterministic rewriting in any category with POs and (final) pullback complements (PBCs) over monos [7]. In typical concrete settings, it enables us to apply operations of addition, deletion, cloning and merging of elements where, by element, we mean any concrete constituent of an object in a category of interest (such as nodes and edges in categories of graphs). The rewrite of an object G is defined by a *rule* $r : L \leftarrow r^- - P - r^+ \rightarrow R$ and its *instance* given by a mono $m : L \rightarrowtail G$. The application of r is performed in two phases as in Diagram 1: (a) an object G^- is constructed as the PBC of r^- and m and (b) the final result of rewriting G^+ is constructed as the PO of m^- and r^+. An arbitrary rewrite of an object in a category of interest can be decoupled into two phases: the *restrictive* rewrite (Diagram 1a) performing deletion and cloning of elements and the *expansive* rewrite (Diagram 1b) performing merging and addition of elements.

$$\begin{tikzcd} L \arrow[d, tail, "m"] \arrow[dr, phantom, "(a)"] & P \arrow[l, "r^-"'] \arrow[r, "r^+"] \arrow[d, tail, "m^-"] \arrow[dr, phantom, "(b)"] & R \arrow[d, tail, "m^+"] \\ G & G^- \arrow[l, "g^-"] \arrow[r, "g^+"'] & G^+ \end{tikzcd} \quad (1)$$

Transformations of individual objects through SqPO rewriting can be efficiently represented with corresponding rewriting rules and their instances. However, such rewriting may introduce *side-effects*, i.e. graph transformations not explicitly specified by the underlying rules and instances. The nature of these side-effects depends on the category in which we are working. For example, in both simple and non-simple graphs, edges not matched by the left-hand side of the rule can be removed as a side-effect of a node removal. Due to such side-effects, having applied a rewriting rule to an object, we can no longer restore this object by simply looking at the applied rule and its instance. The reversible variant of SqPO rewriting that does not introduce side-effects was presented in [9]. It corresponds to the scenario where the SqPO rewriting diagram can be read both forwards and backwards. More formally:

Definition 2.1. *An SqPO rewriting as in Diagram 1 is* reversible*, if the square (a) is also a PO and the square (b) is also a PBC, i.e.* $P \rightarrowtail^{m^-} G^- \rightarrow^{g^+} G^+$ *is the final PBC of* r^+ *and* m^+*. We call* $r^{-1} : R \leftarrow r^+ - P - r^- \rightarrow L$ *the* reverse *of* r*.*

Remark 2.2. *When restricting SqPO rewriting to its reversible variant, we lose the capability to perform deletions in an unknown context [7]. However, this does not reduce our rewriting to the DPO approach [8] due to the operation of cloning, allowed by the PBC constituting the first phase of SqPO but not realizable by DPO (or SPO). In the context of KR, the operation of cloning represents an important update capability which allows us to perform concept refinement, as discussed in [4] and [16].*

2.2 Composition of SqPO Rewriting

Let $r_1 : L_1 \leftarrow P_1 \rightarrow R_1$ be a rewriting rule applied to an object G_1 through an instance $m_1 : L_1 \rightarrowtail G_1$ and let G_2 be the result of application of this rule (corresponding to Diagram 2). Let $r_2 : L_2 \leftarrow P_2 \rightarrow R_2$ be a rule applied to the resulting object G_2 through an instance $m_2 : L_2 \rightarrowtail G_2$ (as in Diagram 3).

$$\begin{tikzcd} L_1 \arrow[d, tail, "m_1"] & P_1 \arrow[l, "r_1^-"'] \arrow[r, "r_1^+"] \arrow[d, tail, "m_1^-"] & R_1 \arrow[d, tail, "m_1^+"'] \\ G_1 & G_1^- \arrow[l, "g_1^-"] \arrow[r, "g_1^+"'] & G_2 \end{tikzcd} \quad (2) \qquad \begin{tikzcd} L_2 \arrow[d, tail, "m_2"] & P_2 \arrow[l, "r_2^-"'] \arrow[r, "r_2^+"] \arrow[d, tail, "m_2^-"] & R_2 \arrow[d, tail, "m_2^+"'] \\ G_2 & G_2^- \arrow[l, "g_2^-"] \arrow[r, "g_2^+"'] & G_3 \end{tikzcd} \quad (3)$$

Given these two consecutive rule applications, we would like to find a rule $L \leftarrow r^- - P - r^+ \rightarrow R$ and an instance $m : L \rightarrowtail G_1$ that, when applied to G_1, directly produces the object G_3, i.e. such that Diagram 4 is an SqPO diagram. Apart from being well-structured for SqPO rewriting, the construction of such a rule will require the category in which we are working to be *adhesive* [19].

$$\begin{tikzcd} L \arrow[d, tail, "m"] & P \arrow[l, "r^+"'] \arrow[r, "r^-"] \arrow[d, tail, "m^-"] & R \arrow[d, tail, "m^+"'] \\ G_1 & G_1^{\ominus} \arrow[l, "g^-"] \arrow[r, "g^+"'] & G_3 \end{tikzcd} \quad (4)$$

Let us first proceed by constructing the pullback (PB) $R_1 \leftarrow x \prec D \succ y \rightarrow L_2$ from m_1^+ and m_2. Note that, because PBs preserve monos, arrows x and y are monos. We will call the span given by this PB the *overlap* of R_1 and L_2 given their matching inside G_2, and we will denote it with o. The PB of x and r_1^+ indicates whether the two rule applications are *sequentially independent*: if this PB is isomorphic to D then the overlap is included in the preserved region P_1 of r_1 and the application of r_2 did not depend on the prior application of r_1; otherwise, the application of r_1 creates something that is tested by r_2 so that its application is dependent on the prior application of r_1 (see [9, 14] for more formal details). The PO $R_1 \succ l_1^H \rightarrow H \leftarrow l_2^H \prec L_2$ from x and y as in Diagram 5 constructs the object H that can be seen as the union of two patterns R_1 and L_2 given their overlap. By the universal property (UP) of POs, there exists a unique arrow $m^H : H \rightarrow G_2$ that renders the diagram commutative. This arrow gives us the PO factorization of the PB of m_1^+ and m_2. Because m_1^+ and m_2 are monos, by adhesivity, m^H is also a mono (see *Theorem 5.1.* in [19]).

$$\begin{tikzcd} & D \arrow[dl, tail, "x"'] \arrow[dr, tail, "y"] & \\ R_1 \arrow[dr, tail, "r_1^H"] \arrow[ddr, tail, bend right, "m_1^+"'] & & L_2 \arrow[dl, tail, "l_2^H"'] \arrow[ddl, tail, bend left, "m_2"] \\ & H \arrow[d, tail, "m^H"] & \\ & G_2 & \end{tikzcd} \quad (5)$$

Using the object H we now construct two objects P_1^H and P_2^H given by the final PBC $P_1 \succ p_1^H \rightarrow P_1^H - h_1^+ \rightarrow H$ to r_1^+ and r_1^H and $P_2 \succ p_2^H \rightarrow P_2^H - h_2^- \rightarrow H$ to r_2^- and l_2^H as in Diagrams 6 and 7.

$$\begin{tikzcd} P_1 \arrow[r, "r_1^+"] \arrow[d, tail, "p_1^H"'] & R_1 \arrow[d, tail, "r_1^H"] \\ P_1^H \arrow[r, "h_1^+"'] & H \end{tikzcd} \quad (6) \qquad \begin{tikzcd} L_2 \arrow[d, tail, "l_2^H"'] & P_2 \arrow[l, "r_2^-"'] \arrow[d, tail, "p_2^H"] \\ H & P_2^H \arrow[l, "h_2^-"] \end{tikzcd} \quad (7)$$

For the first PBC to be 'meaningful', we need to make the assumption that the application of r_1 is *reversible*. Having made this assumption, the object P_1^H can be interpreted as the result of reverting the rewrite specified by r_1^+ on H. On the other hand, the second PBC simply applies the rewrite specified by the arrow r_2^- to H. It is easy to demonstrate that, by the UP of final PBCs, there exist unique arrows $m_1^H : P_1^H \to G_1^-$ and $m_2^H : P_2^H \to G_2^-$ that render Diagrams 8 and 9 commutative. Moreover, m_1^H and m_2^H are monos.

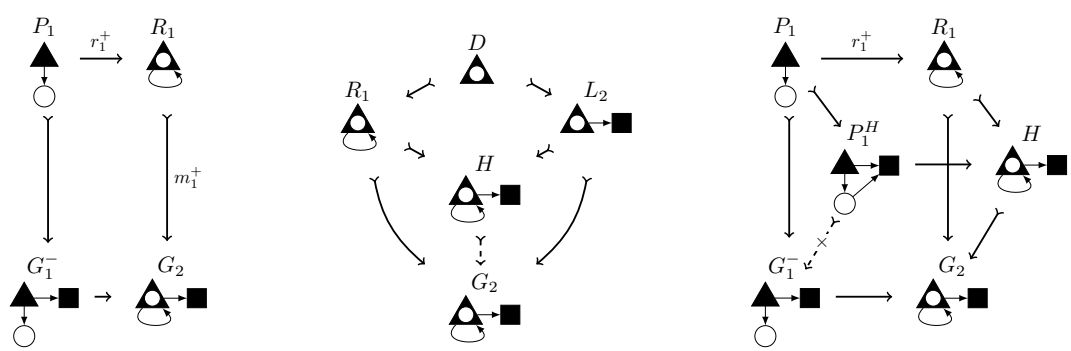
$$\begin{tikzcd}
P_1 \arrow[d, tail, "p_1^H"'] \arrow[dr, "Id_{P_1}"] \arrow[drr, tail, bend left, "r_1^+"] & & \\
P_1^H \arrow[dr, dashed, "m_1^H"'] \arrow[drr, tail, bend right] & P_1 \arrow[r, "r_1^+"] \arrow[d, tail, "m_1^-"'] & R_1 \arrow[d, tail, "m_1^+"] \\
& G_1^- \arrow[r, "g_1^+"'] & G_2
\end{tikzcd} \qquad (8)$$

$$\begin{tikzcd}
& & P_2 \arrow[dl, tail, bend right, "r_2^-"'] \arrow[dl, "Id_{P_2}"] \arrow[d, tail, "p_2^H"] \\
L_2 \arrow[d, tail, "m_2"'] & P_2 \arrow[l, "r_2^-"'] \arrow[d, tail, "m_2^-"] & P_2^H \arrow[dl, dashed, "m_2^H"] \arrow[dl, tail, bend left] \\
G_2 & G_2^- \arrow[l, "g_2^-"] &
\end{tikzcd} \qquad (9)$$

To understand how the non-reversibility of the first rule prevents us from finding a 'meaningful' P_1^H consider the following example.

Example 2.2.1. *In Figure 1a below, we give an example expansive phase of the first rewrite applied to G_1^-, as in the right-most square in Diagram 2 (different shapes are used to encode node identities and maps.) It selects the circle and the triangle node from G_1^- and merges them. This rewrite is not reversible, i.e. we cannot restore G_1^- by applying r_1^+ through m_1^+ because the depicted square does not form a PBC: the PBC of r_1^+ and m_1^+ would need an additional edge, from the circle node to the square node, in G_1^-.*

Given the left-hand side L_2 of a second rule and a matching into G_2, we obtain the union H of R_1 and L_2, given their overlap, as in Figure 1b. Reverting the rewrite specified by r_1^+ on H gives us the object P_1^H depicted in Figure 1c, which splits the merged circle and triangle. This splitting reconnects the black square to both circle and triangle which prevents us from constructing a match $P_1^H \rightarrowtail G_1^-$ necessary to obtain the desired composition—precisely because G_1^- is not the PBC of r_1^+ and m_1^+.



(a) (b) (c)

Figure 1: Composition with an irreversible rule.

Next, let us construct two POs: $L_1 \rightarrowtail^{l_1^H} L \leftarrow^{h_1^-} P_1^H$ from r_1^- and p_1^H and $R_2 \rightarrowtail^{r_2^H} R \leftarrow^{h_2^+} P_2^H$ from r_2^+ and p_2^H. The first PO reverts the rewrite of P_1^H specified by r_1^-, and the second performs the rewrite of P_2^H using r_2^+ and p_2^H. The constructed object L represents the result of reversing the transformation of the pattern P_1^H specified by r_1^- and the instance p_1^H, precisely because the application of r_1 is reversible. By the UP of these POs we can construct unique matches $m : L \rightarrowtail G_1$ and $m^+ : R \rightarrowtail G_3$.

Finally, to construct the rule composition, we find the PB $P_1^H \leftarrow_{p'} - P -_{p''} \rightarrow P_2^H$ from h_1^+ and h_2^-. The resulting rule corresponds to the span $r : L \leftarrow_{h_1^- \circ p'} - P -_{h_2^+ \circ p''} \rightarrow R$. We will refer to it as the *composed rule* given the overlap o and write $r = \otimes(r_1, o, r_2)$.

Theorem 2.3 (Synthesis)**.** *In adhesive categories, if rewriting given by r_1 is reversible, application of the rule r given by $\otimes(r_1, o, r_2)$ with the instance $m : L \rightarrowtail G_1$ produces the object G_3, i.e. Diagram 4 with $r^- = h_1^- \circ p'$ and $r^+ = h_2^+ \circ p''$ is an SqPO diagram.*

Proposition 2.4. *In adhesive categories, the composition of two reversible rewrites is reversible.*

2.3 Hierarchies and SqPO Rewriting in Hierarchies

A *hierarchy* of objects in a category **C** is a directed acyclic graph (DAG) whose nodes are objects and whose edges are arrows from **C** such that all paths between each pair of objects are equal [16]. We refer to the latter condition as the *commutativity* condition. In the rest of this paper we assume that we are working in a fixed category **C** that has an appropriate structure for SqPO rewriting. For the commutativity of a hierarchy to be maintained, an SqPO rewrite of an object situated inside the hierarchy may require updates to other objects and arrows called *propagation*.

The main model of interest to us operates on hierarchies of (simple) graphs and uses both hierarchy objects and arrows to represent knowledge. Hierarchies provide a powerful formalism for representation and update of fragmented knowledge on different interrelated abstraction levels. In this model, an edge of a hierarchy associated to an arrow $h : G \rightarrow T$ is often interpreted as *typing*, i.e. the graph T defines the kinds of nodes and edges that can exist in G. We can further interpret moving in a hierarchy along the direction of its edges as moving away from more concrete to more abstract representation. In this context, the commutativity condition guarantees that the representation of knowledge on different abstraction levels is consistent, i.e. the representation of knowledge from some concrete graph obtained by moving along alternative paths leading to the same abstract graph is consistent. The propagation framework presented in [16] allows us to transform individual graphs inside a hierarchy and perform the *co-evolution* of its different layers, which guarantees the consistency of knowledge at all times.

When the knowledge represented in a hierarchy may be frequently updated by potentially different curators, it is often desirable to maintain the history of updates and be able to store multiple versions of knowledge at the same time. The design of a mathematical system providing such features, thus, constitutes the principal motivation for this work, and this system relies on the existence of a compact representation of object transformations. SqPO rewriting enables an efficient representation of object transformations using rules. To be able to build and study such a representation for transformations in hierarchies, let us briefly formulate the notions of rewriting and propagation in hierarchies (formal details can be found in [16]).

Rewriting an individual object situated at a node of a hierarchy affects the objects associated at its ancestor and the descendant nodes, while the rest of the objects stay unchanged. The restrictive phase is propagated *backwards* to all the objects typed by the target of rewriting. For instance, let G be an object corresponding to an ancestor of an object T in a hierarchy with an associated arrow $h : G \rightarrow T$. A restrictive rewrite performing deletion and cloning of elements in T induces propagation to instances of these elements in G. The expansive phase is propagated *forward* to all the objects typing the target. For instance, for $h : G \rightarrow T$ as before, an expansive rewrite performing addition and merging of elements in G induces propagation to T affecting the types of these elements. Interestingly, merging of elements in T induced by such forward propagation affects all the hierarchy objects typed by T (corresponding to the hierarchy side effects described in Subsection 3.2). The commutativity of the updated hierarchy is guaranteed by the *composability conditions* imposed on the performed propagations.

3 Rule Hierarchies

In this section we formulate the notion of a *rule hierarchy* that serves us as a compact representation of coupled transformations of objects in a hierarchy. Intuitively, a rule hierarchy is a description of how every object situated in the hierarchy should be transformed and how the arrows between these objects should be reconstructed. In the practical application of interest, we would like to use such descriptions for recording the history of hierarchy updates.

We then define formally how a rule hierarchy can be applied to the corresponding hierarchy of objects through specified instances and study the side-effects introduced by the application of a rule hierarchy. Such side-effects, apart from the side-effects introduced by SqPO rewriting on objects, may include some implicit changes to the arrows in the hierarchy. Thus, we formulate the conditions under which a given application of a rule hierarchy is reversible. Finally, we present how consecutive rewrites of a hierarchy can be composed. In this section we consider SqPO rewriting rules operating on objects from **C**. Such rules are spans formed by objects and arrows from **C**.

Definition 3.1. *A* rule homomorphism *f from $r_1 : L_1 \xleftarrow{r_1^-} P_1 \xrightarrow{r_1^+} R_1$ to $r_2 : L_2 \xleftarrow{r_2^-} P_2 \xrightarrow{r_2^+} R_2$ is given by three arrows $\lambda : L_1 \to L_2$, $\pi : P_1 \to P_2$ and $\rho : R_1 \to R_2$ from* **C** *such that $\lambda \circ r_1^- = r_2^- \circ \pi$ and $r_2^+ \circ \pi = \rho \circ r_1^+$.*

Using rules as objects and rule homomorphisms as arrows, we obtain the category of rules **Rule**[**C**] over the category **C**.

Definition 3.2. *A* rule hierarchy *is a hierarchy of objects in the category of rules.*

Let $\mathscr{H}$ be a hierarchy of objects in **C** and $\mathscr{R}$ be a hierarchy of rules operating on objects in **C** both defined over the same DAG $\mathscr{G} = (V, E \subseteq V \times V)$. We refer to such $\mathscr{G}$ as the *skeleton* of $\mathscr{H}$ and $\mathscr{R}$. For the sake of simplicity, in the rest of this section we will assume that we are working on a fixed pair $(\mathscr{H}, \mathscr{R})$ defined over the same skeleton. As a short-hand, for every node $v \in V$ we will denote the object associated to v in $\mathscr{H}$ with G_v and the rule associated to v in $\mathscr{R}$ with $r_v : L_v \xleftarrow{r_v^-} P_v \xrightarrow{r_v^+} R_v$. For every edge $(s,t) \in E$ we will denote the associated arrows in $\mathscr{H}$ as $h_{(s,t)}$ and the arrows constituting the rule homomorphism in $\mathscr{R}$ as $\lambda_{(s,t)}$, $\pi_{(s,t)}$ and $\rho_{(s,t)}$.

Definition 3.3. *An* instance *of $\mathscr{R}$ in $\mathscr{H}$ is given by a function $\mathscr{I} : V \to Monos(\mathbf{C})$ that associates every node of the skeleton to an instance of the corresponding rule from $\mathscr{R}$ in the corresponding object from $\mathscr{H}$, i.e. $\mathscr{I}(v) : L_v \rightarrowtail G_v$ for all $v \in V$. For every node $v \in V$ we will denote the instance $\mathscr{I}(v)$ as m_v.*

Definition 3.4. *$\mathscr{R}$ is* applicable *to $\mathscr{H}$ through an instance $\mathscr{I}$, if for any pair of nodes $s,t \in V$ such that $(s,t) \in E$:*

- *$h_{(s,t)} \circ m_s = m_t \circ \lambda_{(s,t)}$, i.e. their instances commute;*
- *if G_s^- and G_t^- are the results of the restrictive phase of rewriting given by the final PBC of r_s^- and m_s, and the final PBC of r_t^- and m_t respectively, then there exists a unique $h_{(s,t)}^- : G_s^- \to G_t^-$ that renders Diagram 10 commutative.*

$$
\begin{tikzcd}
L_s \arrow[d, tail, "m_s"'] \arrow[dr, "\lambda_{(s,t)}"] & P_s \arrow[l, "r_s^-"'] \arrow[d, tail, "m_s^-"'] \arrow[dr, "\pi_{(s,t)}"] & \\
G_s \arrow[dr, "h_{(s,t)}"'] & G_s^- \arrow[l, "s^-"'] \arrow[dr, "h_{(s,t)}^-"] & \\
 & L_t \arrow[d, tail, "m_t"] & P_t \arrow[l, "r_t^-"'] \arrow[d, tail, "m_t^-"] \\
 & G_t & G_t^- \arrow[l, "t^-"']
\end{tikzcd}
\tag{10}
$$

Remark 3.5. *Observe that, if the left face in Diagram 10 is a PB, $\mathscr{R}$ is always applicable to $\mathscr{H}$ as the unique arrow $h_{(s,t)}^-$ exists by the UP of final PBCs.*

To rewrite $\mathscr{H}$ using the rule hierarchy $\mathscr{R}$, applicable through an instance $\mathscr{I}$, for every node of the skeleton we simply apply the associated rule to the associated object through the instance specified by $\mathscr{I}$. To restore the arrows of $\mathscr{H}$, for every edge $(s,t) \in E$, we use the applicability condition and the UP of POs as follows. Let the back and the front faces of the cube in Diagram 11 be two SqPO diagrams corresponding to the above-mentioned rewriting of the objects G_s and G_t respectively. First of all, by the applicability of $\mathscr{R}$ given $\mathscr{I}$, there exists a unique arrow $h^-_{(s,t)}$ such that $h_{(s,t)} \circ s^- = t^- \circ h^-_{(s,t)}$ and $h^-_{(s,t)} \circ m^-_s = m^-_t \circ \pi_{(s,t)}$. This enables us to use the UP of the PO G^+_s and show that there exists a unique arrow $h^+_{(s,t)}$ that renders Diagram 11 commutative.

$$\begin{tikzcd}
L_s \arrow[d, tail, "m_s"'] \arrow[dr, "\lambda_{(s,t)}"] & P_s \arrow[l, "r^-_s"'] \arrow[r, "r^+_s"] \arrow[d, tail, "m^-_s"'] \arrow[dr, "\pi_{(s,t)}"] & R_s \arrow[d, tail, "m^+_s"'] \arrow[dr, "\rho_{(s,t)}"] & \\
G_s \arrow[dr, "h_{(s,t)}"'] & G^-_s \arrow[l, "s^-"'] \arrow[r, "s^+"] \arrow[dr, dashed, "h^-_{(s,t)}"] & G^+_s \arrow[dr, dashed, "h^+_{(s,t)}"] & \\
& L_t \arrow[d, tail, "m_t"] & P_t \arrow[l, "r^-_t"'] \arrow[r, "r^+_t"] \arrow[d, tail, "m^-_t"] \arrow[dr, dashed] & R_t \arrow[d, tail, "m^+_t"] \\
& G_t & G^-_t \arrow[l, "t^-"'] \arrow[r, "t^+"] & G^+_t
\end{tikzcd} \tag{11}$$

Therefore, the notion of a rule hierarchy can be used as a compact representation of coupled updates in hierarchies of objects. Rules describe tranformations of hierarchy objects and rule homomorphisms allow us to restore arrows between them. The commutativity condition imposed on the rule homomorphisms in a hierarchy guarantees that their application results in a valid hierarchy of objects.

3.1 Expressing Rewriting and Propagation in Hierarchies

In this subsection we will briefly discuss how the transformations of objects and arrows in a hierarchy $\mathscr{H}$ induced by a rewrite of an object G with a rule $r : L \leftarrow^{r^-} - P - r^+ \rightarrow R$ through $m : L \rightarrowtail G$ can be represented as a rule hierarchy and its instance. Recall that, upon rewriting of an object in a hierarchy, the objects associated to ancestors and descendants of the origin of rewriting are updated according to the framework of backward and forward propagation [16], while the rest of the objects stay unaffected. We would like to construct a rule hierarchy that is defined over the skeleton of $\mathscr{H}$ and, therefore, contains rules for both affected and unaffected objects. For the sake of conciseness, here we will focus only on non-trivial updates to objects, i.e. on the construction of a rule *sub-hierarchy* corresponding to the objects updated as the result of backward or forward propagation. In this paper, we give only the high-level idea behind these constructions; full technical details can be found in [22].

Backward propagation rules. Let H be an object corresponding to an ancestor of an object G in a hierarchy with an associated arrow $h : H \rightarrow G$. Backward propagation of r^- to H can be expressed as a rule $\hat{r} : L_H \leftarrow^{\hat{r}^-} - P_H - {}^{Id_{P_H}}\!\rightarrow P_H$ with an instance $\hat{m} : L_H \rightarrowtail H$ that, when applied to H, results in an object H^- that is homomorphic to the result of the original rewriting of G, in a way that makes Diagram 12 commute.

$$\begin{tikzcd}
L_H \arrow[d, tail, "\hat{m}"'] \arrow[dr, "\hat{h}"] & P_H \arrow[l, "\hat{r}^-"'] \arrow[r, "Id_{P_H}"] \arrow[d, tail, "\hat{m}^-"'] \arrow[dr, "\hat{h}^-"] & P_H \arrow[ddr, "r^+ \circ \hat{h}^-"] & \\
H \arrow[ddr, "h"'] & H^- \arrow[l, "-"'] \arrow[dr, "h^-"] & & \\
& L \arrow[d, tail, "m"] & P \arrow[l, "r^-"'] \arrow[r, "r^+"] \arrow[d, tail, "m^-"] \arrow[dr] & R \arrow[d, tail, "m^+"] \\
& G & G^- \arrow[l, "g^-"'] \arrow[r, "g^+"] & G^+
\end{tikzcd} \tag{12}$$

Such a rule, called the *lifting* of r^-, is constructed given a specification for backward propagation (i.e. a rule factorization and a clean-up arrow [16]), and contains only restrictive updates (given by deletion and cloning). Informally, this specification indicates which changes to G should be propagated to H and how the updated H can be 'retyped' by the updated G. This typing is obtained as a composition $g^+ \circ h^-$ in the diagram. Together with the rule lifting, we obtain the arrows $\hat{h} : L_H \to L$ and $\hat{h}^- : P_H \to P$ defining the rule homomorphism given by $(\hat{h}, \hat{h}^-, r^+ \circ \hat{h}^-)$ as in the diagram. Therefore, the backward propagation framework allows us to extract a rule representing the specified propagation for every ancestor of the rewritten object together with homomorphisms to the original rule r. Moreover, under appropriate conditions (see *backward composability* in [16]), we can construct the homomorphisms between backward propagation rules required by the skeleton of the hierarchy.

Forward propagation rules. Let T be an object corresponding to a descendant of an object G in a given hierarchy with an associated arrow $h : G \to T$. Forward propagation of r^+ to T can be expressed as a rule $\hat{r} : P_T \leftarrow P_T \to R_T$ with an instance $\hat{m}^- : P_T \rightarrowtail T$ that, when applied to T, results in an object T^+ to which the result of the original rewriting of G is homomorphic, in a way renders Diagram 13 commutative. Such a rule, called the *projection* of r^+, is constructed given a specification for forward propagation (i.e. a rule factorization and a clean-up arrow [16]), and contains only expansive updates (addition and merging).

$$\begin{tikzcd}
 & L \arrow[d, tail, "m"] & P \arrow[l, "r^-"'] \arrow[r, "r^+"] \arrow[d, tail, "m^-"] & R \arrow[d, tail, "m^+"] \\
 & G & G^- \arrow[l, "g^-"] \arrow[r, "g^+"] & G^+ \\
P_T \arrow[uur, "\hat{h}"] & P_T \arrow[l, "Id_{P_T}"'] \arrow[r, "\hat{r}^+"] \arrow[uur, "\hat{h}^-"] \arrow[d, tail, "\hat{m}^-"'] & R_T \arrow[uur, "\hat{h}^+"] \arrow[d, tail, "\hat{m}^+"] & \\
 & T \arrow[r, "t^+"'] \arrow[uur, "h"] & T^+ \arrow[uuur, "h^+"] &
\end{tikzcd} \tag{13}$$

Informally, this specifies which changes to G should be propagated to T and how the updated G can be 'retyped' by the updated T. Together with the rule projection, we obtain three arrows $\hat{h} : L \to P_T$, $\hat{h}^- : P \to P_T$ and $\hat{h}^+ : R \to R_T$ that define the rule homomorphism given by $(\hat{h}, \hat{h}^-, \hat{h}^+)$. Forward propagation thus allows us to extract a rule representing the specified propagation for every descendant of the rewritten object together with homomorphisms from the original rule r and, under appropriate conditions (see *forward composability* in [16]), we can also construct the required rule homomorphisms.

Example 3.5.1. *Consider the hierarchy $\mathscr{H}$ depicted in Figure 2. Let G be the target of rewriting with the rule highlighted in the grey area: this rule clones the circle into two semi-circles and merges one of these circles with the square node. Let the rules delimited with dashed arrows be the objects of the rule hierarchy representing respective rewriting and propagation. The rule $L_H \leftarrow P_H \to R_H$ is a backward propagation rule that specifies how the cloning in G is propagated to different instances of the circle: in this case, the white circle in H is to be cloned but the black circle can still be typed by the right semi-circle. The rule $L_T \leftarrow P_T \to R_T$ is a forward propagation rule that describes the merging of the nodes typing the semi-circle and the square in G. Finally, the hierarchy $\mathscr{H}'$ on the right represents the result of the rule hierarchy application.*

3.2 Reversible Rewriting in Hierarchies

In this subsection, we study the side-effects introduced by the application of a rule hierarchy. These side-effects may induce some implicit changes to the arrows representing hierarchy edges, which may prevent us from restoring the original hierarchy simply by looking at the applied rule hierarchy. In general, such side-effects make the rewriting produced by reversing the original rule hierarchy not *applicable*. Let us first consider the following example.

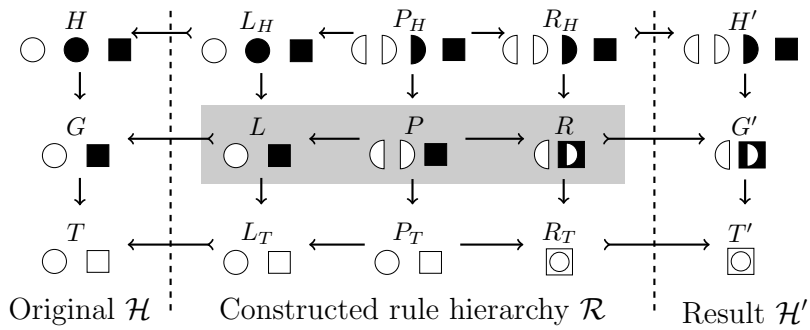


Figure 2: Example of rewriting in a hierarchy represented with a rule hierarchy.

Example 3.5.2. *Let $G \to T$ in Figure 3a be two homomorphic objects and let $P_G \to R_G$ and $P_T \to R_T$ specify expansive phases of rules applied to these objects. The rule $P_G \to R_G$ merges one of the white circles of G with the black circle; the rule $P_T \to R_T$ merges the white and black circles. As a side-effect, the other white circle is also now typed by the merged node in T^+: we 'forget' that it was an instance of the white circle in T. In Figure 3b, we reverse the rules and apply them to G^+ and T^+: the merged node in T^+ is cloned into two circle nodes and one instance of the merged node in G^+ is cloned. We recover G but cannot type it by T, precisely because we 'forgot' how the circle coloured grey was typed in T.*

(a) Application of the original rules

(b) Application of the reversed rules

Figure 3: Example of side-effects affecting hierarchy arrows

Definition 3.6. *The* reverse *$\mathscr{R}^{-1}$ of $\mathscr{R}$ is the rule hierarchy whose nodes correspond to the rules r_v^{-1} for all $v \in V$, and whose edges correspond to the rule homomorphisms $(\rho_{(s,t)}, \pi_{(s,t)}, \lambda_{(s,t)})$ for all edges $(s,t) \in E$.*

Definition 3.7. *Rewriting of $\mathscr{H}$ with $\mathscr{R}$, applicable through an instance $\mathscr{I}$, is* reversible*, if rewriting of every individual object is reversible and the reverse $\mathscr{R}^{-1}$ is applicable, i.e. for any pair of nodes $s,t \in V$ such that $(s,t) \in E$ corresponding to objects and rules as in Diagram 11, if G_s^- is given as the final PBC of r_s^+ and m_s^+ and G_t^- as the final PBC of r_t^+ and m_t^+, there exists a unique arrow $h_{(s,t)}^- : G_s^- \to G_t^-$ that makes the right cube in Diagram 11 commute.*

Remark 3.8. *Even though the latter definition imposes rather abstract requirements, some intuitive sufficient conditions can be formulated. For example, if for every hierarchy edge the left-most face in Diagram 11 is a PB, the rewriting is reversible. Informally, this condition guarantees that all the instances of the elements selected by m_t are also selected by m_s. This means that the scenario from Example 3.5.2 cannot be realized, because all the instances of the merged types are selected by the rule.*

3.3 Composition of Rewriting in Hierarchies

To study composition of rewriting in a hierarchy, we will focus on a simple hierarchy with two nodes and one edge, corresponding to objects G_1, T_1 and an arrow $h_1 : G_1 \to T_1$. Composition of rewriting in general hierarchies can be trivially obtained by applying this technique to every pair of hierarchy nodes connected with an edge.

Let $\mathscr{H}$ be a hierarchy corresponding to an arrow $G_1 \xrightarrow{h_1} T_1$ and let $\mathscr{R}_1$ be a rule hierarchy corresponding to rules p_G and p_T, whose arrow $f^p : p_G \to p_T$ is given by arrows λ_1, π_1 and ρ_1 as in Diagram 14. Let $G_2 \xrightarrow{h_2} T_2$ correspond to the result of the application of $\mathscr{R}_1$ through the instances m_G and m_T (we assume that $\mathscr{R}_1$ is applicable given m_G and m_T). Let $\mathscr{R}_2$ be another rule hierarchy given by a homomorphic pair of rules q_G and q_T as in Diagram 15. Their homomorphism $f^q : q_G \to q_T$ is given by arrows λ_2, π_2 and ρ_2. Let $G_3 \xrightarrow{h_3} T_3$ be the result of the application of $\mathscr{R}_2$ through the instances n_G and n_T as in the diagram (similarly, we assume that $\mathscr{R}_2$ is applicable given the instances).

$$
\begin{tikzcd}
L_1^G \arrow[d, tail, "m_G"'] \arrow[dr, "\lambda_1"] & P_1^G \arrow[l, "p_G^-"'] \arrow[r, "p_G^+"] \arrow[d, tail, "m_G^-"'] \arrow[dr, "\pi_1"] & R_1^G \arrow[d, tail, "m_G^+"'] \arrow[dr, "\rho_1"] & \\
G_1 \arrow[dr, "h_1"'] & G_1^- \arrow[l, "g_1^-"'] \arrow[r, "g_1^+"] \arrow[d, "h_1^-"] & G_2 \arrow[d, "h_2"] & \\
& L_1^T \arrow[d, tail, "m_T"] & P_1^T \arrow[l, "p_T^-"'] \arrow[r, "p_T^+"'] \arrow[d, tail, "m_T^-"] & R_1^T \arrow[d, tail, "m_T^+"] \\
& T_1 & T_1^- \arrow[l, "t_1^-"] \arrow[r, "t_1^+"'] & T_2
\end{tikzcd}
\qquad (14)
$$

$$
\begin{tikzcd}
L_2^G \arrow[d, tail, "n_G"'] \arrow[dr, "\lambda_2"] & P_2^G \arrow[l, "q_G^-"'] \arrow[r, "q_G^+"] \arrow[d, tail, "n_G^-"'] \arrow[dr, "\pi_2"] & R_2^G \arrow[d, tail, "n_G^+"'] \arrow[dr, "\rho_2"] & \\
G_2 \arrow[dr, "h_2"'] & G_2^- \arrow[l, "g_2^-"'] \arrow[r, "g_2^+"] \arrow[d, "h_2^-"] & G_3 \arrow[d, "h_3"] & \\
& L_2^T \arrow[d, tail, "n_T"] & P_2^T \arrow[l, "q_T^-"'] \arrow[r, "q_T^+"'] \arrow[d, tail, "n_T^-"] & R_2^T \arrow[d, tail, "n_T^+"] \\
& T_2 & T_2^- \arrow[l, "t_2^-"] \arrow[r, "t_2^+"'] & T_3
\end{tikzcd}
\qquad (15)
$$

We can compose these pairs of rewrites using the constructions presented in Subsection 2.2. Namely, if the rules p_G and p_T are reversible, we can find a pair of rules, $r_G : L_G \xleftarrow{r_G^-} P_G \xrightarrow{r_G^+} R_G$ and $r_T : L_T \xleftarrow{r_T^-} P_T \xrightarrow{r_T^+} R_T$, and a pair of instances, $l_G : L_G \rightarrowtail G_1$ and $l_T : L_T \rightarrowtail T_1$, such that applying r_G to G_1 and r_T to T_1 through l_G and l_T respectively (as in Diagram 16, where X stands for G or T), we directly obtain G_3 and T_3 from Diagram 15.

$$
\begin{tikzcd}
L_X \arrow[d, tail, "l_X"] & P_X \arrow[l, "r_X^-"'] \arrow[r, "r_X^+"] \arrow[d, tail, "l_X^-"] & R_X \arrow[d, tail, "l_X^+"] \\
X_1 & X_1^{\ominus} \arrow[l, "h_X^-"] \arrow[r, "h_X^+"'] & X_3
\end{tikzcd}
\qquad (16)
$$

To be able to construct a rule homomorphism $f : r_G \to r_T$, we need to make the assumption that the rewriting specified by $\mathscr{R}_1$ given m_G and m_T is reversible, i.e. for G_1^- being the PBC of p_G^+ and m_G^+ and T_1^- being the PBC of p_T^+ and m_T^+, there always exists a unique arrow h_1^- that renders the right-most cube in Diagram 14 commutative. Let D^G, x^G, y^G, D^T, x^T and y^T from Diagram 17 be the two overlaps of respectively R_1^G with L_2^G and R_1^T with L_2^T, constructed as described in Subsection 2.2 and denoted with o^G and o^T. By the UP of PBs, there exists a unique arrow $d : D^G \to D^T$ using which we can construct a hierarchy of such overlaps defined over the same skeleton as $\mathscr{H}$, and together with arrows x^G, y^G, x^T and y^T, this gives us the *hierarchy overlap* $\mathscr{O}$.

$$
\begin{tikzcd}
& D^G \arrow[dl, tail, "x^G"'] \arrow[d, dashed, "d"] \arrow[dr, tail, "y^G"] & \\
R_1^G \arrow[d, "\rho_1"'] & D^T \arrow[dl, tail, "x^T"] \arrow[dr, tail, "y^T"'] & L_2^G \arrow[d, "\lambda_2"] \\
R_1^T \arrow[dr, tail, "m_T^+"'] & & L_2^T \arrow[dl, tail, "n_T"] \\
& T_2 &
\end{tikzcd}
\qquad (17)
$$

Lemma 3.9. *If the rewriting of $\mathscr{H}$ given by $\mathscr{R}_1$ through m_G and m_T is reversible then, given the hierarchy overlap $\mathscr{O}$, there exists a unique rule homomorphism $f : r_G \to r_T$.*

Therefore, we can construct the rule hierarchy $\mathscr{R}$ corresponding to $r_G - f \to r_T$. We will refer to it as the *composed rule hierarchy* given the hierarchy overlap $\mathscr{O}$ and write $\mathscr{R} = \otimes(\mathscr{R}_1, \mathscr{O}, \mathscr{R}_2)$.

Theorem 3.10. *In an adhesive category, if the rewriting given by $\mathscr{R}_1$ is reversible, $\mathscr{R}$ is applicable given l_G and l_T, and its application results in $G_3 - h_3 \to T_3$.*

Proposition 3.11. *In an adhesive category, the composition of two reversible hierarchy rewrites is reversible.*

4 Transformation Audit Trails

In this section we describe how reversibility and composition of rewriting can be used to construct the audit trail for transformations of individual objects and hierarchies of objects. The proposed framework is implemented as a part of the `ReGraph`[2] Python library for building hierarchical knowledge representations based on simple graphs with attributes.

4.1 Audit Trails for Object Transformations

Let G^0 be the starting object whose history of transformations we would like to maintain and let $\langle r_i : L^i \leftarrow r_i^- - P^i - r_i^+ \to R^i \mid i \in [1 \dots n] \rangle$ be a sequence of rules consecutively applied to G^0 through the instances $m_i : L^i \rightarrowtail G^{i-1}$, resulting in a sequence of objects $\langle G^i \mid i \in [1 \dots n] \rangle$ with $m_i^+ : R^i \rightarrowtail G^i$ for $1 \leq i \leq n$, i.e. such that for every $1 \leq i \leq n$, Diagram 18 is a SqPO diagram. To be able to build a sound audit trail, we additionally require such a sequence of rewrites to be reversible.

$$\begin{tikzcd} L^i \arrow[d, tail, "m_i"] & P^i \arrow[l, "r_i^-"'] \arrow[r, "r_i^+"] \arrow[d, tail, "m_i^-"] & R^i \arrow[d, tail, "m_i^+"] \\ G^{i-1} & \bar{G}^{i-1} \arrow[l, "\bar{g}_i^-"] \arrow[r, "\bar{g}_i^+"'] & G^i \end{tikzcd} \quad (18)$$

Definition 4.1. *The* audit trail *for the object G^n consists of the sequence of rules $\langle r_i \mid i \in [1 \dots n] \rangle$ and the right-hand side instances $m_i^+ : R^i \rightarrowtail G^i$ for $1 \leq i \leq n$.*

Example 4.1.1. *Consider Figure 4 depicting an audit trail. Gray circles represent the states of a potentially large graph whose history of transformations we record. The small graphs inside gray circles represent the localized patterns affected by the transformations. The audit trail contains the SqPO rules representing transformations and their instances. Only the elements highlighted with solid lines are stored by the system at any time, i.e. the current state of the object, the rules and the instances representing the history of updates.*

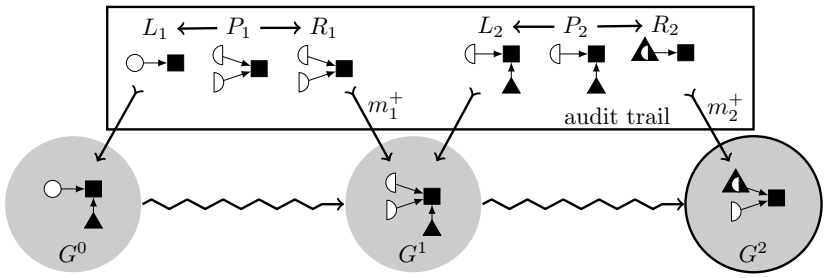


Figure 4: Example of an audit trail

[2] `https://github.com/Kappa-Dev/ReGraph`

Remark 4.2. *The reader may wonder how rule instances (matches) are stored in the audit trail without storing the domain objects of these matches (for example, how do we store $m_1^+ : R_1 \rightarrowtail G^1$ from Example 4.1.1 without storing G^1 itself). In practice, we assume that graph elements (nodes and edges) are uniquely identified (e.g. with shapes and colors as in our example), and instead of storing the domain of our instances (e.g. G^1), it is enough to store the mapping of unique element identifiers of the co-domain and domain object constituting the rule instance (identifiers of nodes and edges of R_1 and G^1 in our example).*

Rollback. Using such an audit trail we can *rollback* to any point in the history of transformations corresponding to some intermediate object G^i for $0 \leq i \leq n-1$ by applying the sequence rules $\langle r_j^{-1} \mid j \in [n \dots i+1] \rangle$ with the corresponding instances $m_j^+ : R^j \rightarrowtail G^j$ for $j \in [n \dots i+1]$.

Maintain diverged versions. To maintain multiple versions of an object in the audit trail, we use a technique known from VCSs as *delta compression*, i.e. only the current version of the object is stored, while the other versions are encoded in a *delta*, a representation of the 'difference' between the versions. Let v_1 and v_2 be two versions of the starting object G^0 with v_1 being the current version. Initial delta Δ from v_1 to v_2 is set to the identity rule (the rule that does not perform any transformations) $\emptyset \xleftarrow{Id_\emptyset} \emptyset \xrightarrow{Id_\emptyset} \emptyset$ and the instance $u : \emptyset \rightarrowtail G^0$, where $\emptyset$ stands for the initial object in $\mathbf{C}$ and u is the unique arrow from the initial object to G^0. Every rewrite of the current version of the object induces an update of the delta that consists in the composition of the previous delta and the reverse of the applied rule (recall that we assume that every rewriting in the audit trail is reversible).

As before, let v_1 be the current version corresponding to some object G (e.g. obtained by transforming of G^0) and let $r_\Delta : L^\Delta \xleftarrow{r_\Delta^-} P^\Delta \xrightarrow{r_\Delta^+} R_\Delta$ and $m_\Delta : L^\Delta \rightarrowtail G$ be respectively the rule and the instance given by Δ. Let $r : L \xleftarrow{r^-} P \xrightarrow{r^+} R$ be a rule applied to G through the instance $m : L \rightarrowtail G$ and G' be the result of application of r given m. To update the delta, we compute the composition $\otimes(r^{-1}, o, r^\Delta)$ with o being a span $L \xleftarrow{x} \prec D \xrightarrow{y} L^\Delta$ obtained as a PB from m and m_Δ. The new delta is, thus, set to the rule and the instance given by the composition $\otimes(r^{-1}, o, r_\Delta)$.

Example 4.2.1. *Figure 5 illustrates how multiple version of the same object are maintained within the proposed audit trail system. Figure 1a illustrates two versions, v_1 and v_2, of the same object. As in the previous example, only the elements highlighted with solid lines are stored by the system. In this example we store the current version v_1 of the object and the difference between the two versions expressed with the delta. The right-most rule in Figure 1a represents a transformation of the current version v_1. In Figure 1b, using the overlap between the delta and the transformation rule, we compute the new delta representing the difference between the updated v_1 and v_2.*

Switch versions. Switching between different versions of the object can be done by simply applying the rule through the instance given by the delta. Namely, if v_1 is the current version corresponding to an object G with the delta to v_2 given by $\Delta = (r_\Delta, m_\Delta)$, switching to v_2 is performed by applying r_Δ to G through the instance m_Δ. If G' is the result of the above-mentioned rewriting and $m_\Delta^+ : R^\Delta \rightarrowtail G'$ is its right-hand side instance, then v_2 becomes the current version of the object and the new delta Δ is set to $(r_\Delta^{-1}, m_\Delta^+)$.

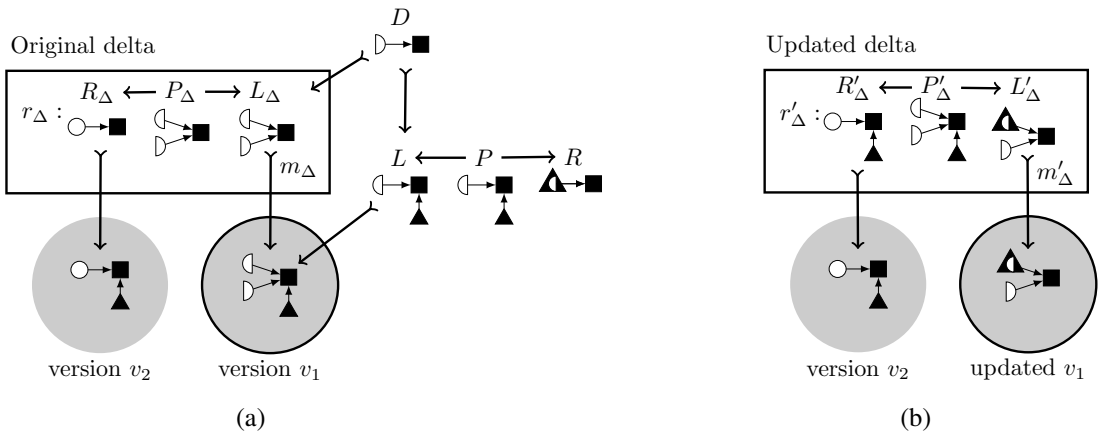


Figure 5: Update of a delta representing different versions of an object.

Merge versions. Let v_1 be the current version corresponding to an object G, v_2 be another version corresponding to an object G' and the delta between v_1 and v_2 be given by $\Delta = (r_\Delta, m_\Delta)$. The left and top faces of the cube in Diagram 19 correspond to the two phases of the application of r_Δ through m_Δ.

$$\begin{tikzcd}
P_\Delta \arrow[rr, "r^+_\Delta"] \arrow[d, "r^-_\Delta"'] \arrow[dr, tail, "m^-_\Delta"] & & R_\Delta \arrow[d, "\hat{r}^-"'] \arrow[dr, tail, "m^+_\Delta"] & \\
L_\Delta \arrow[rr, "\hat{r}^+"'] \arrow[dr, tail, "m_\Delta"'] \arrow[drr] & & \hat{M} \arrow[dr, tail, "\hat{m}"] \arrow[drr] & \\
& \bar{G} \arrow[rr, "g^+"'] \arrow[d, "g^-"] & & G' \arrow[d, dashed, "\hat{g}^-"] \\
& G \arrow[rr, "\hat{g}^+"] & & \hat{G}
\end{tikzcd} \tag{19}$$

The canonical merging rules for v_1 and v_2 are given by two arrows $\hat{r}^+$ and $\hat{r}^-$ constructed by the pushout $L_\Delta - \hat{r}^+ \rightarrow \hat{M} \leftarrow \hat{r}^- - R_\Delta$ from r^-_Δ and r^+_Δ (see the back face of the cube in the diagram). The merging rule for the current object G is then applied by finding the pushout $G - \hat{g}^+ \rightarrow \hat{G} \leftarrow \hat{m} \prec \hat{M}$ from m_Δ and $\hat{r}^+$ as in the bottom face of Diagram 19. By the universal property of pushouts there exists a unique arrow $\hat{g}^- : G' \rightarrow \hat{G}$ that renders the cube commutative. The object $\hat{G}$ is the result of the canonical merging of G and G'. Note that, because the application of r_Δ is reversible, the left face is also a pushout, which implies that the right face of the cube is also a pushout. We thus obtain the merged object $\hat{G}$ by applying the merging rule $\hat{r}^-$ to G' through the instance m^+_Δ.

Non-canonical merging rules are given by two arrows $r^+_M : L_\Delta \rightarrow M$ and $r^-_M : R_\Delta \rightarrow M$ such that $r^+_M \circ r^-_\Delta = r^-_M \circ r^+_\Delta$. The merged object G_M is, thus, obtained by applying the rule r^+_M to G through the instance m_Δ or the rule r^-_M to G' through the instance m^+_Δ.

Example 4.2.2. *Figure 6 illustrates how the operation of merge can be performed within the audit trail. The delta depicted in the figure corresponds to the delta from Figure 5. Using this delta, we compute the canonical merging rules for the two versions given by $L_\Delta \rightarrow \hat{M}$ and $R_\Delta \rightarrow \hat{M}$ respectively. In this example, the merging rule merges the two semi-circles produced by the cloning in v_1 (see the first rule applied tot G^0 in Figure 4), which results into the object represented with gray solid circle.*

4.2 Audit Trails for Hierarchy Transformations

Let $\mathscr{H}^0$ be the starting hierarchy of objects, defined over a skeleton $\mathscr{G} = (V, E)$, whose history of transformations we would like to maintain. Let $\langle \mathscr{R}_i \mid i \in [1 \ldots n] \rangle$ be a sequence of rule hierarchies consecutively applied to $\mathscr{H}^0$ through the instances $\mathscr{I}_i$, resulting in a sequence of hierarchies $\langle \mathscr{H}^i \mid i \in [1 \ldots n] \rangle$ with the right-hand side instances given by $\mathscr{I}_i^+$ for $1 \leq i \leq n$, i.e. for every $v \in V$, $\mathscr{I}_i^+(v) : R^i_v \rightarrowtail G^i_v$. As in the case of individual objects, to be able to build an audit trail, we require all the rewrites to be reversible.

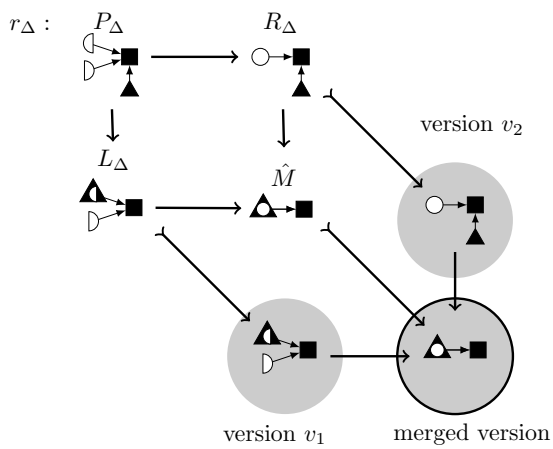


Figure 6: Merging different versions of an object.

Definition 4.3. *The* audit trail *for $\mathscr{H}^n$ consists of the sequence of rule hierarchies $\langle \mathscr{R}_i \mid i \in [1 \dots n] \rangle$ and the right-hand side instances $\mathscr{I}_i^+$ for $1 \leq i \leq n$.*

Rollback. Using the audit trail we can *rollback* to any point in the history of transformations corresponding to some intermediate hierarchy $\mathscr{H}^i$ for $0 \leq i \leq n-1$. This can be done by applying the rule hierarchies $\langle \mathscr{R}_j^{-1} \mid j \in [n \dots i+1] \rangle$ with the corresponding instances $\mathscr{I}_j^+$, where $\mathscr{I}_j^+(v) : R_v^j \rightarrowtail G_v^j$ for every $v \in V$ and $j \in [n \dots i+1]$.

Maintain diverged versions. To accommodate multiple versions of a hierarchy, we use delta compression. Let v_1 and v_2 be two versions of the starting hierarchy $\mathscr{H}^0$ with v_1 being the current version. The initial delta Δ from v_1 to v_2 is set to the identity rule hierarchy with the rule $\emptyset \xleftarrow{Id_\emptyset} \emptyset \xrightarrow{Id_\emptyset} \emptyset$ at every node $v \in V$. We set the instance $\mathscr{I}(v)$ for every $v \in V$ to be the unique arrow $u_v : \emptyset \rightarrowtail G_v^0$. Every rewrite of the current version of the hierarchy induces an update of the delta that consists in the composition of the previous delta and the reverse of the applied rule hierarchy. Let v_1 be the current version corresponding to some hierarchy $\mathscr{H}$.

Let $\mathscr{R}_\Delta$ and $\mathscr{I}_\Delta$ be the rule hierarchy and instance given by Δ, where $r_v^\Delta : L_v^\Delta \xleftarrow{r_{v,\Delta}^-} P_v^\Delta \xrightarrow{r_{v,\Delta}^+} R_v^\Delta$ and $m_v^\Delta : L_v^\Delta \rightarrowtail G_v$ are the rule and the instance corresponding to a node $v \in V$. Let $\mathscr{R}$ be a rule hierarchy applied to $\mathscr{H}$ through the instance $\mathscr{I}$ and $\mathscr{H}'$ be the result of the corresponding rewriting. The new delta is given by the rule hierarchy and the instance obtained as the composition $\otimes(\mathscr{R}^{-1}, \mathscr{O}, \mathscr{R}^\Delta)$ with $\mathscr{O}$ being the hierarchy overlap computed by finding the overlaps between L_v and L_v^Δ for every node $v \in V$ and the arrow $D_s \to D_t$ between overlaps given by the UP of PBs as in Diagram 20 for every edge $(s,t) \in E$.

$$\begin{tikzcd}
 & D_s \arrow[dl, tail, "x_s"'] \arrow[d, dashed] \arrow[dr, tail, "y_s"] & \\
L_s \arrow[d, "\lambda_{(s,t)}"'] & D_t \arrow[dl, tail, "x_t"] \arrow[dr, tail, "y_t"'] & L_s^\Delta \arrow[d, "\lambda^\Delta_{(s,t)}"] \\
L_t \arrow[dr, tail, "m_t"'] & & L_t^\Delta \arrow[dl, tail, "m_t^\Delta"] \\
 & G_t &
\end{tikzcd} \tag{20}$$

Switch versions. Switching between versions of the hierarchy is performed by applying the rule hierarchy though the instance given by the delta. If v_1 is the current version corresponding to a hierarchy $\mathscr{H}$ with the delta given by $\Delta = (\mathscr{R}_\Delta, \mathscr{I}_\Delta)$, switching to v_2 is performed by applying $\mathscr{R}_\Delta$ to $\mathscr{H}$ through $\mathscr{I}_\Delta$. For $\mathscr{H}'$ being the result of rewriting and $\mathscr{I}_\Delta^+$ being its right-hand side instance (where for every $v \in V$, $\mathscr{I}_\Delta^+(v) : R_v^\Delta \rightarrowtail G_v'$), v_2 becomes the current version of the object and the new delta is $(\mathscr{R}_\Delta^{-1}, \mathscr{I}_\Delta^+)$.

Merge versions. Let v_1 be the current version corresponding to a hierarchy $\mathscr{H}$, v_2 be another version corresponding to a hierarchy $\mathscr{H}'$ and the delta between v_1 and v_2 be given by $\Delta = (\mathscr{R}_\Delta, \mathscr{I}_\Delta)$.

The canonical merging rule hierarchy can be constructed in the following way. For every individual hierarchy node we construct the canonical merging rule according to the framework described in Subsection 4.1. Let the back and front faces of the cube in Diagram 21 correspond to the POs defining pairs of merging rules corresponding to nodes $s,t \in V$ such that $(s,t) \in E$. We can apply the universal property of the POs and show that there exists a unique arrow $m_{(s,t)} : \hat{M}_s \to \hat{M}_t$ that makes the diagram commute. The merging rule hierarchy $\hat{\mathscr{R}}^+$ for $\mathscr{H}$ is, thus, given by rules $L_v \xleftarrow{Id_{L_v}} L_v \xrightarrow{\hat{r}_v^+} \hat{M}_v$, for all $v \in V$, and rule homomorphisms defined by arrows $(\lambda_{(s,t)}, \lambda_{(s,t)}, m_{(s,t)})$, for all $(s,t) \in E$. On the other hand, the merging rule hierarchy for $\mathscr{H}'$ is given by rules $R_v \xleftarrow{Id_{R_v}} R_v \xrightarrow{\hat{r}_v^-} \hat{M}_v$, for all $v \in V$, and rule homomorphisms defined by arrows $(\rho_{(s,t)}, \rho_{(s,t)}, m_{(s,t)})$, for all $(s,t) \in E$. Let $\hat{G}_s$ and $\hat{G}_t$ be the result of merging corresponding to the nodes s and t. By the universal property of POs there exists a unique arrow $\hat{h}_{(s,t)}$ that renders Diagram 22 commutative. Therefore, using such objects $\hat{G}_v$ for every $v \in V$ and arrows $h_{(s,t)}$ for every $(s,t) \in E$, we can construct the hierarchy $\hat{\mathscr{H}}$ corresponding to the result of canonical merging of $\mathscr{H}$ and $\mathscr{H}'$. *Non-canonical merging* can be specified using a hierarchy of objects $\mathscr{M}$ defined over the same skeleton as $\mathscr{H}$, and a pair of arrows $\bar{r}_v^+ : L_v \to M_v$ and $\bar{r}_v^- : R_v \to M_v$ such that $\bar{r}_v^+ \circ r_v^- = \bar{r}_v^- \circ r_v^+$ for every $v \in V$.

$$
\begin{tikzcd}
P_s \arrow[rr, "r_s^+"] \arrow[d, "r_s^-"'] \arrow[dr, "\pi_{(s,t)}"] & & R_s \arrow[d, "\hat{r}_s^-"'] \arrow[dr, "\rho_{(s,t)}"] & \\
L_s \arrow[rr, "\hat{r}_s^+"] \arrow[dr, "\lambda_{(s,t)}"'] & & \hat{M}_s \arrow[dr, dashed, "m_{(s,t)}"] & \\
& P_t \arrow[rr, "r_t^+"'] \arrow[d, "r_t^-"] & & R_t \arrow[d, "\hat{r}_t^-"] \\
& L_t \arrow[rr, "\hat{r}_t^+"] & & \hat{M}_t
\end{tikzcd}
\qquad (21)
$$

$$
\begin{tikzcd}
L_s \arrow[rr, "\hat{r}_s^+"] \arrow[d, tail, "m_s"'] \arrow[dr, "\lambda_{(s,t)}"] & & \hat{M}_s \arrow[d, tail, "\hat{m}_s"'] \arrow[dr, "m_{(s,t)}"] & \\
G_s \arrow[rr, "\hat{g}_s^+"] \arrow[dr, "h_{(s,t)}"'] & & \hat{G}_s \arrow[dr, dashed, "\hat{h}_{(s,t)}"] & \\
& L_t \arrow[rr, "r_t^+"'] \arrow[d, tail, "m_t"] & & \hat{M}_t \arrow[d, tail, "\hat{m}_t"] \\
& G_t \arrow[rr, "\hat{g}_t^+"] & & \hat{G}_t
\end{tikzcd}
\qquad (22)
$$

5 Conclusions

In this paper we have described how the reversibility and composition of SqPO rewriting can be used to design an audit trail framework for individual objects and hierarchies of objects. In particular, we have presented the construction that composes consecutive SqPO rewrites, provided the first rewrite is reversible. We have also presented the notion of rule hierarchy, that generalizes SqPO rewriting to hierarchies of objects, and allows for an efficient representation of rewriting and its propagation, as previously presented in [16]. We have studied the conditions under which an arbitrary rule hierarchy can be applied to the corresponding hierarchy of objects and described the conditions for such an application to be reversible. We then briefly discussed the construction that can be used to compose consecutive applications of two rule hierarchies. Finally, we have described how an audit trail for individual objects and hierarchies of objects can be defined. Such an audit trail allows for the maintenance of the history of transformations and provides the means for reverting sequences of such transformations. Moreover, it enables the accommodation of multiple versions of the same object that have diverged as the result of conflicting rewrites.

As future work, we would like to study how an arbitrary transformation from a sequence of rewrites can be undone for individual objects and hierarchies of objects. This question is directly related to the theory of *causality* for SqPO rewriting and requires a generalization for such rewriting in hierarchies.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider & Daniele Nardi (2003): *The Description Logic handbook: theory, implementation and applications*. Cambridge University Press.

[2] Konstantinos Barmpis, Seyyed Shah & Dimitrios S. Kolovos (2015): *Towards incremental updates in large-scale model indexes*. In: *Modelling Foundations and Applications (ECMFA)*, Springer, Cham, pp. 137–153, doi:10.1007/978-3-319-21151-0_10.

[3] Nicolas Behr (2019): *Sesqui-pushout rewriting: concurrency, associativity and rule algebra framework*. *Electronic Proceedings in Theoretical Computer Science* 309, p. 23–52, doi:10.4204/eptcs.309.2.

[4] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko & Hannes Voigt (2019): *Schema validation and evolution for graph databases*. In: *Conceptual Modeling (ER)*, Springer, Cham, pp. 448–456, doi:10.1007/978-3-030-33223-5_37.

[5] Arnaud Castelltort & Anne Laurent (2013): *Representing history in graph-oriented NoSQL databases: a versioning system*. In: *Digital Information Management (ICDIM)*, IEEE, pp. 228–234, doi:10.1109/ICDIM.2013.6694022.

[6] Peter Pin-Shan Chen (1976): *The entity-relationship model—toward a unified view of data*. *ACM Transactions on Database Systems (TODS)* 1(1), pp. 9–36, doi:10.1145/320434.320440.

[7] Andrea Corradini, Tobias Heindel, Frank Hermann & Barbara König (2006): *Sesqui-pushout rewriting*. In: *Graph Transformations (ICGT)*, Springer, Berlin, Heidelberg, pp. 30–45, doi:10.1007/11841883_4.

[8] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel & Michael Löwe (1997): *Algebraic approaches to graph transformation–part I: basic concepts and double pushout approach*. In: *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, World Scientific, pp. 163–245, doi:10.1142/9789812384720_0003.

[9] Vincent Danos, Tobias Heindel, Ricardo Honorato-Zimmer & Sandro Stucki (2014): *Reversible sesqui-pushout rewriting*. In: *Graph Transformation (ICGT)*, Springer, Cham, pp. 161–176, doi:10.1007/978-3-319-09108-2_11.

[10] Roy Dyckhoff & Walter Tholen (1987): *Exponentiable morphisms, partial products and pullback complements*. *Journal of Pure and Applied Algebra* 49(1-2), pp. 103–116, doi:10.1016/0022-4049(87)90124-1.

[11] Hartmut Ehrig, Claudia Ermel & Gabriele Taentzer (2011): *A formal resolution strategy for operation-based conflicts in model versioning using graph modifications*. In: *Fundamental Approaches to Software Engineering (FASE)*, Springer, Berlin, Heidelberg, pp. 202–216, doi:10.1007/978-3-642-19811-3_15.

[12] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski & Francesco Parisi-Presicce (1991): *Parallelism and concurrency in high-level replacement systems*. *Mathematical Structures in Computer Science* 1(3), pp. 361–404, doi:10.1017/S0960129500001353.

[13] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer & Andrés Taylor (2018): *Cypher: An evolving query language for property graphs*. In: *Management of Data (SIGMOD)*, ACM, pp. 1433–1445, doi:10.1145/3183713.3190657.

[14] Russ Harmer (2017): *Rule-based meta-modelling for bio-curation*. Habilitation à Diriger des Recherches, ENS Lyon, France. Available at `https://tel.archives-ouvertes.fr/tel-01534703/`.

[15] Russ Harmer, Yves-Stan Le Cornec, Sébastien Légaré & Eugenia Oshurko (2019): *Bio-curation for cellular signalling: the KAMI project*. *IEEE/ACM transactions on computational biology and bioinformatics* 16(5), pp. 1562–1573, doi:10.1109/TCBB.2019.2906164.

[16] Russ Harmer & Eugenia Oshurko (2020): *Knowledge representation and update in hierarchies of graphs*. *Journal of Logical and Algebraic Methods in Programming* 114, 100559, doi:10.1016/j.jlamp.2020.100559.

[17] Udayan Khurana & Amol Deshpande (2013): *Efficient snapshot retrieval over historical graph data*. In: *Data Engineering (ICDE)*, IEEE, pp. 997–1008, doi:10.1109/ICDE.2013.6544892.

[18] Stephen Lack & Pawel Sobocinski (2004): *Adhesive categories*. In: *Foundations of Software Science and Computation Structures (FoSSaCS)*, Springer, Berlin, Heidelberg, pp. 273–288, doi:10.1007/978-3-540-24727-2_20.

[19] Stephen Lack & Paweł Sobociński (2005): *Adhesive and quasiadhesive categories*. *RAIRO-Theoretical Informatics and Applications* 39(3), pp. 511–545, doi:10.1051/ita:2005028.

[20] James J. Leifer & Robin Milner (2000): *Deriving bisimulation congruences for reactive systems*. In: *Concurrency Theory (Concur)*, Springer, Berlin, Heidelberg, pp. 243–258, doi:10.1007/3-540-44618-4_19.

[21] Michael Löwe (2015): *Polymorphic sesqui-pushout graph rewriting*. In: *Graph Transformation (ICGT)*, Springer, Cham, pp. 3–18, doi:10.1007/978-3-319-21145-9_1.

[22] Eugenia Oshurko (2020): *Knowledge representation and curation in hierarchies of graphs*. PhD thesis, Université de Lyon. Available at `https://tel.archives-ouvertes.fr/tel-02917559`.

[23] Betty Salzberg & Vassilis J. Tsotras (1999): *Comparison of access methods for time-evolving data*. *ACM Computing Surveys (CSUR)* 31(2), pp. 158–221, doi:10.1145/319806.319816.

[24] Christian Schneider, Albert Zündorf & Jörg Niere (2004): *CoObRA–a small step for development tools to collaborative environments*. In: *Workshop on Directions in Software Engineering Environments (WoDiSEE)*, IET, doi:10.1049/ic:20040206.

[25] Gabriele Taentzer, Claudia Ermel, Philip Langer & Manuel Wimmer (2010): *Conflict detection for model versioning based on graph modifications*. In: *Graph Transformation (ICGT)*, Springer, Berlin, Heidelberg, pp. 171–186, doi:10.1007/978-3-642-15928-2_12.