EasyChair Preprint

№ 8741

A PBPO+ Graph Rewriting Tutorial

Roy Overbeek and Joerg Endrullis

August 29, 2022

A PBPO$^+$ Graph Rewriting Tutorial

Roy Overbeek Jörg Endrullis
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
r.overbeek@vu.nl j.endrullis@vu.nl

We provide a tutorial introduction to the algebraic graph rewriting formalism PBPO$^+$. We show how PBPO$^+$ can be obtained by composing a few simple building blocks. Along the way, we comment on how alternative design decisions lead to related formalisms in the literature, such as DPO.

1 Introduction

Graphs are useful data structures for virtually every field of computer science. And they come in many varieties: they can be directed or undirected, may carry labels or attributes, the edges can be hyperedges, and so on. Computation on these graphs is however similar, and can be roughly described as the stepwise replacement of subgraphs, induced by a set of replacement rules. For this reason, it is helpful if the replacement mechanism can be defined in a general way, without having to commit to a particular notion of a graph. Ehrig et al. [9] first managed to do so in the 1970s, by specifying a replacement mechanism (namely, Double Pushout (DPO) rewriting) in the language of category theory. Their seminal work laid the foundation for the field of algebraic graph rewriting [7], in which a variety of categorical graph replacement mechanisms have been proposed and studied.

PBPO$^+$ [14] (short for *Pullback-Pushout with strong matching*) is a graph rewriting formalism in this algebraic tradition, obtained by adding a strong matching requirement to PBPO [4]. One feature of PBPO$^+$ is that it is expressive: in the abstract setting of quasitoposes, PBPO$^+$ has been shown [15] to subsume other well known formalisms such as DPO [9], SqPO [5], AGREE [3] and PBPO. Quasitoposes include various graph-like categories such as the categories of labeled multigraphs, hypergraphs, safely marked Petri nets and simple graphs; as well as various categories that are not graph-like.

In this tutorial, we introduce PBPO$^+$ in a stepwise manner, starting from minimal preliminaries. In particular, we do not assume an understanding of category theory or related formalisms. Instead, we introduce two toy formalisms ToyPO (Section 2) and ToyPB (Section 3), and show how PBPO$^+$ can be understood as a combination of the two (Section 4). Our approach complements existing tutorials for related formalisms (Section 5).

Preliminaries. We write $f \circ g$ to denote function composition ($(f \circ g)\,x = f(g(x))$). By a *graph* G we mean an *unlabeled directed multigraph*, i.e., $G = (V, E, s, t)$, where V is a set of *vertices* (or *nodes*), E a set of *edges*, $s : V \to E$ a *source* function and $t : E \to V$ a *target* function. A *graph homomorphism* $\phi : G \to H$ from graph G to graph H is a pair of functions $\phi_V : V_G \to V_H$ and $\phi_E : E_G \to E_H$ satisfying $\phi_V \circ s_G = s_H \circ \phi_E$ and $\phi_V \circ t_G = t_H \circ \phi_E$. We use $\rightarrowtail$ to denote injective homomorphisms, and $A \cong B$ to denote that A and B are isomorphic.

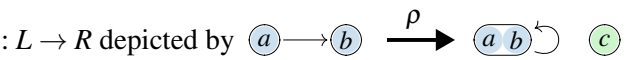
A note on vocabulary. A category consists of *objects* and *morphisms* between them. In our graph setting, these instantiate to graphs, and graph homomorphisms between graphs, respectively. We state some concepts (such as pushouts and pullbacks) categorically in this paper, but readers unacquainted with category theory may safely read "graph" for "object" and "graph homomorphism" for "morphism".

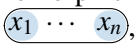
Submitted to:
TERMGRAPH 2022

2 ToyPO

We start by defining a toy graph rewrite formalism called ToyPO (short for ToyPushout). This formalism computes rewrite steps using a single pushout construction. It allows the specification of identification and addition of elements.

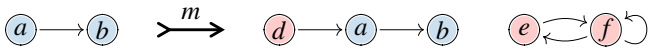
Definition 1 (ToyPO Rule)**.** A *ToyPO* rule is a morphism $\rho : L \to R$. L and R are called *patterns*.

Example 1. The rule $\rho : L \to R$ depicted by  can be described as specifying (i) the identification of two connected nodes a and b, and (ii) the addition of a node c.

Notation 1 (Visual Notation). Our notation for graphs and graph homomorphisms can be described formally. A vertex is a non-empty set $\{x_1,\ldots,x_n\}$ represented by a box  , and each morphism $f = (\phi_V, \phi_E) : G \to G'$ between depicted graphs G and G' is the unique morphism satisfying $S \subseteq f(S)$ for all $S \in V_G$. For instance, for nodes $\{a\}$ and $\{b\}$ in the left hand side of ρ (Example 1), $\rho(\{a\}) = \rho(\{b\}) = \{a,b\}$ of the right hand side. We will use examples that ensure uniqueness of each f (in particular, we ensure that the mapping of edges ϕ_E is uniquely determined by the mapping of nodes ϕ_V). Colors are purely supplementary.

Definition 2 (Match)**.** A *match* for a ToyPO rule $\rho : L \to R$ in a *host graph* G is an injective morphism $m : L \rightarrowtail G$. The image $m(L) \subseteq G$ is an *occurrence* of L in G.

Example 2. The injective morphism $m : L \rightarrowtail G$ depicted by



is a match for rule ρ of Example 1 in the host graph G. Three other matches are possible: $d \to a$, $e \to f$ and $f \to e$.

Remark 1. We require injectivity of m for three reasons. First, it is easier to understand. Second, it yields a strictly more expressive formalism (cf. Habel et al. [10]). Third, it better prepares for PBPO$^+$, where matches are injective as well.

Given rule $\rho : L \to R$ and match $m : L \rightarrowtail G$ of Examples 1 and 2, respectively, arguably the most reasonable result of the rewrite step is the graph

$H =$ 

because this graph can be understood as the *minimal solution* in which $m(L)$ is replaced by R in G. More specifically, no elements of $G - m(L)$ are deleted, duplicated or identified; and no elements other than those in $R - \rho(L)$ are added. Astonishingly, this informal notion of a minimal solution can be defined formally and abstractly using the language of category theory; that is, without making any reference whatsoever to graphs or graph homomorphisms.

Definition 3 (Pushout [16, 2])**.** The *pushout* of a *span* $G \overset{m}{\leftarrow} L \overset{\rho}{\rightarrow} R$ is a *cospan* $\sigma = G \overset{i_G}{\rightarrow} H \overset{i_R}{\leftarrow} R$ such that

1. σ *is a candidate solution*: $i_G \circ m = i_R \circ \rho$; and
2. σ *is the minimal solution*: for any cospan $G \overset{i'_G}{\rightarrow} H' \overset{i'_R}{\leftarrow} R$ that satisfies ${i_G}' \circ m = {i_R}' \circ \rho$, there exists a *unique* morphism $x : H \to H'$ such that ${i_G}' = x \circ i_G$ and ${i_R}' = x \circ i_R$.

Both requirements are conveniently visualized in the commuting diagram depicted in Figure 1.

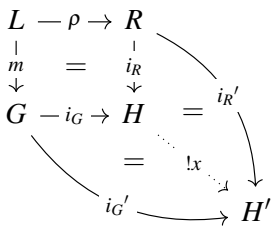
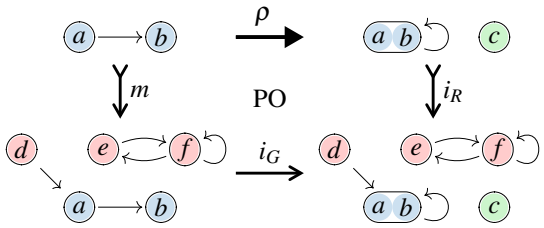


Figure 1: Pushout.

Pushouts are unique up to isomorphism if they exist (and they always do in the category of graphs). A useful intuitive description of a pushout is that of a *gluing construction*, because H can be thought of as the result of gluing G and R along shared interface L. This view is especially intuitive if m is injective, which is always the case in our setting.

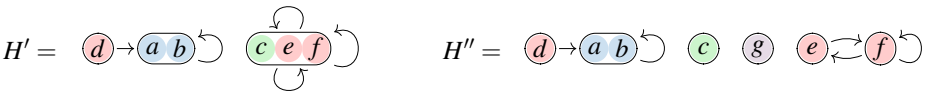
Exercise 1. For span $G \overset{m}{\leftarrow} L \overset{\rho}{\rightarrow} R$ given by Examples 1 and 2, verify that

1. the following pushout is indeed a candidate solution (i.e., satisfies condition 1 of Definition 3):



(1)

2. the following graphs (with the obvious choices for the cospan morphisms i_G, i_R) constitute two other candidate solutions:



3. H satisfies condition 2 of Definition 3 specifically for competing candidates H' and H''; and
4. H' and H'' both fail condition 2 of Definition 3 because
   (a) no suitable witness $x : H' \to H$ exists (the identification of elements cannot be undone); and
   (b) a suitable witness $x : H'' \to H$ does not exist uniquely (node g can be mapped freely).

For a set theoretic description of the pushout, one may consult König et al. [12]. For this paper, however, it suffices if the reader understands the pushout intuitively as a gluing construction, and that it can be used for the specification of identification and addition of elements.

We may now define the notion of a ToyPO rewrite step.

Definition 4 (ToyPO Rewrite Step)**.** A ToyPO rule $\rho : L \to R$ and match $m : L \rightarrowtail G$ induce a *ToyPO rewrite step* $G \Rightarrow^{\rho,m}_{\text{ToyPO}} H$ if there exists a pushout of the form shown on the right.

$$\begin{array}{ccc} L & -\rho\to & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle i_R} \\ G & -i_G\to & H \end{array}$$

3 ToyPB

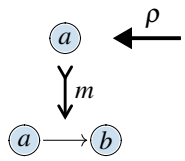
Next, we would like to specify the duplication and deletion of elements. This is slightly more challenging, and allows for a larger variety of solutions.

For some historical context, and to illustrate the solution space, consider the approach where we invert the procedure described in Section 2. For example, we read rule $\rho : L \to R$ of Example 1 right-to-left, suggestively writing the type signature as $\rho : R \leftarrow L$. Read in this way, ρ intuitively specifies the duplication of node *ab* and the deletion of node *c*. When we find a match $m : R \rightarrowtail G$, we try to show that G can be understood as the result of a suitable gluing. That is, we try to find morphisms $l_1 : L \to H$ and $l_2 : H \to G$ such that the diagram shown on the right is a pushout square. Morphisms l_1 and l_2 together constitute a *pushout complement* for ρ and m.

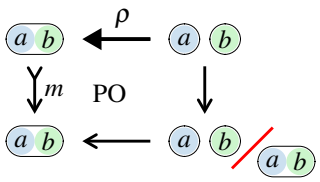
$$\begin{array}{ccc} R & \leftarrow\rho- & L \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle l_1} \\ G & \leftarrow l_2- & H \end{array}$$

Example 3. The pushout square of Diagram (1) in Exercise 1 can be read as an inverted application of ρ, by reading morphism i_R as the match, the bottom right graph as the host graph, and the bottom left graph as the result graph.

The inverse approach has two particular caveats:



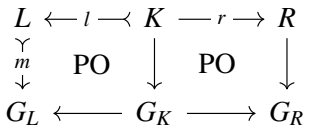
1. *Existence of pushout complements*: Consider a rule ρ that deletes a node a, and a match m as depicted on the right (the domain of ρ is the empty graph). Even though all pushouts exist in the category of graphs, and even though we have found a match m, a pushout complement does not exist for this corner: the depicted edge can never be obtained through a gluing around an empty interface. The more general implication of this observation is that we are only able to delete nodes if they do not leave any edges "dangling" (called the *gluing condition*). This can be considered a pleasant safety feature, but also a limitation: perhaps we would prefer to simply delete any incident edges (as a general principle), or even be able to specify more fine-grained control on the level of the rule itself.



2 *Uniqueness of pushout complements*: More importantly, unlike pushouts, pushout complements need not be unique: the square on the right shows that two pushout complements exist, one of which does not model duplication (similarly, there exists another pushout complement for the pushout square of Diagram (1)). This is problematic if it is desired that rewrite step results are uniquely determined by the rule and match. In the category of graphs, uniqueness of pushout complements can at least be ensured by restricting rules ρ to injective morphisms (thus allowing deletion, but not duplication of elements). However, in other categories, such as the category of simple graphs, this is not generally sufficient.

These two caveats notwithstanding, the most dominant graph rewriting method, called the Double Pushout (DPO) approach, combines the pushout complement approach (with the injectivity requirement) with the ToyPO approach. In doing so, it enables the specification of rewrite steps with deletion, identification and addition features, for matches m that satisfy the gluing condition.

Definition 5 (DPO Rewriting [9])**.** A *DPO rewrite rule* ρ is a span $L \overset{l}{\hookleftarrow} K \overset{r}{\to} R$. A diagram depicted on the right defines a *DPO rewrite step* $G_L \Rightarrow^{\rho,m}_{\text{DPO}} G_R$, i.e., a step from G_L to G_R using rule ρ and match $m : L \rightarrowtail G_L$.



Alternatives to the DPO approach avoid the construction of pushout complements. For instance, the Single Pushout (SPO) approach [13] relies on a single pushout construction, but uses partial graph homomorphisms instead of total morphisms, in order to specify deletion. In this approach, the gluing condition no longer needs to be checked either: all edges incident to a removed vertex are simply deleted. As another example, the Sesqui-Pushout (SqPO) approach [5] replaces the first PO square of DPO by what is called a *final pullback complement* square. This square allows duplication with deterministic behavior, and like SPO, deletes any edges that would be left dangling.

For our proposal, called ToyPB (short for ToyPullback), we consider a perspective that is *dual* to the ToyPO approach, rather than its inverse. First, instead of trying to find an occurrence of some pattern L in a graph G (through a match $m : L \rightarrowtail G$), we try to find a graph homomorphism $\alpha : G \to T$ *into* some graph T. When we adopt such a viewpoint, we will call α an *adherence morphism*, and we can think of T as a kind of type graph. For instance, if T is the graph from Figure 2 then any adherence $\alpha : G \to T$ assigns one of two

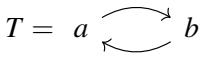


Figure 2: Node colors.

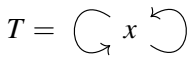
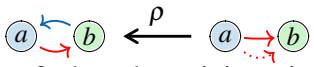


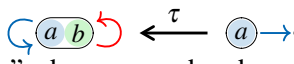
Figure 3: Edge colors.

"colors" a and b to nodes of G such that no two G-neighbors have the same color (α is effectively a proof that G is 2-colorable or bipartite); and if T is as in Figure 3 then we can regard any α as assigning every edge of G one of two "colors", depending on which loop is being assigned. We now consider a ToyPB rule to specify a manipulation of such type graphs.

Definition 6 (ToyPB Rule)**.** A *ToyPB rule* is a morphism $\rho : L' \leftarrow R'$. L' and R' are called *type graphs*.

Observe that ToyPO and ToyPB rules are formally identical. But it helps to emphasize the difference in perspective by presenting the definitions differently.

Example 4. Rule  specifies the deletion of edges that originate in b-nodes, and the duplication of edges that originate in a-nodes. An intended example application is given by the commutative square in Figure 4. In this and the next example, the rule lies at the bottom of the square, and the node identities of the rule have been adjusted to indicate how the morphisms are defined (see Notation 1).

Example 5. Let us again think of an adherence $\alpha : G \to T$ on graphs G and T as assigning colors to elements of G. Rule  specifies the deletion of all "red" edges, and splits all nodes so that the "blue" edges are rendered as edges of a (one-way) bipartite graph. An intended example application is given by the commutative square in Figure 5.

The desired outcomes of Examples 4 and 5 (which we hope the reader agrees are the most reasonable ones given the adopted intuitive perspective) can be obtained through a pullback construction, a notion that is the dual to the pushout (Definition 3). Like pushouts, pullbacks are unique up to isomorphism.

Definition 7 (Pullback [16, 2])**.** The *pullback* of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$ is a span $\sigma = G \xleftarrow{i_G} H \xrightarrow{i_R} R$ such that

1. σ *is a candidate solution*: $\alpha \circ i_G = \rho \circ i_R$; and
2. σ *is the minimal solution*: for any span $G \xleftarrow{i_G'} H' \xrightarrow{i_R'} R'$ that satisfies $\alpha \circ i_G' = \rho \circ i_R'$, there exists a *unique* morphism $x : H' \to H$ such that $i_G' = i_G \circ x$ and $i_R' = i_R \circ x$.

Both requirements are conveniently visualized in the commuting diagram in Figure 6.

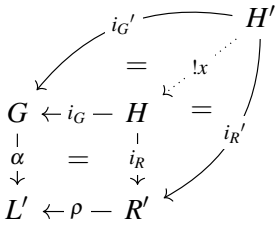


Figure 6: Pullback.

Remark 2 (Fibered Product). The metaphor of a pushout as a gluing construction unfortunately does not dualize very well. However, in the category of sets, the pullback can be understood as *fibered product*, which is a generalization of the familiar Cartesian product: the pullback object H (Figure 6) contains all pairs (x,y) ($x \in G$ and $y \in R'$) for which $\alpha(x) = \rho(y)$. For the category of graphs, the vertex set V and edge set E of H can in fact be constructed by constructing the fibered products for V and E independently.

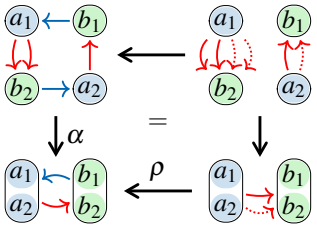


Figure 4: Example 4.

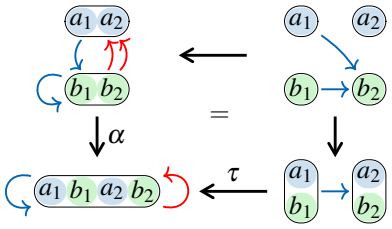


Figure 5: Example 5.

Exercise 2. Adapt Exercise 1 for pullbacks: construct some candidates that compete with the solutions given in Examples 4 and 5 (e.g., consider the empty graph), and assess why these candidates do not qualify as pullbacks.

We may now define the notion of a ToyPB rewrite step.

$$\begin{array}{ccc} G & \leftarrow i_G - & H \\ {\scriptstyle\alpha}\downarrow & \text{PB} & \downarrow{\scriptstyle i_R} \\ L' & \leftarrow \rho - & R' \end{array}$$

Definition 8 (ToyPB Rewrite Step)**.** A ToyPB rule $\rho : L' \leftarrow R'$ and adherence morphism $\alpha : G \to L'$ induce a *ToyPB rewrite step* $G \Rightarrow^{\rho,\alpha}_{\text{ToyPB}} H$ if there exists a pullback of the form depicted on the right.

Note that unlike ToyPO, ToyPB does not have any injectivity restrictions, because we would usually like arbitrarily large graphs G to be typeable by a relatively small type graph L'.

4 PBPO$^+$

The following preliminary exercise shows that we can use pullbacks to construct preimages. It is useful for understanding PBPO$^+$ rules and matches.

Exercise 3 (Computing Preimages). For cospans $L \overset{t_L}{\rightarrowtail} L' \overset{l'}{\leftarrow} K'$ with one injective leg t_L, define $l'^{-1}(t_L)$ as the subgraph of K' that is mapped onto the subgraph $t_L(L)$ of L'. Using intuitive reasoning, convince yourself that for a pullback, as depicted on the right, we have $K \cong l'^{-1}(t_L)$.

$$\begin{array}{ccc} L & \leftarrow l - & K \\ {\scriptstyle t_L}\downarrow & \text{PB} & \downarrow{\scriptstyle t_K} \\ L' & \leftarrow l' - & K' \end{array}$$

Definition 9 (PBPO$^+$ Rewrite Rule [4, 14])**.** A *PBPO$^+$ rewrite rule* ρ is a collection of objects and morphisms, arranged as follows around a pullback square:

$$\rho \;=\; \begin{array}{ccccc} L & \leftarrow l - & K & - r \rightarrow & R \\ {\scriptstyle t_L}\downarrow & \text{PB} & \downarrow{\scriptstyle t_K} & & \\ L' & \leftarrow l' - & K' & & \end{array}$$

L is the *lhs pattern* of the rule, L' its *type graph*, and t_L the *embedding* of L into L'. Similarly for the *interface* K. R is the *rhs pattern* or *replacement for* L.

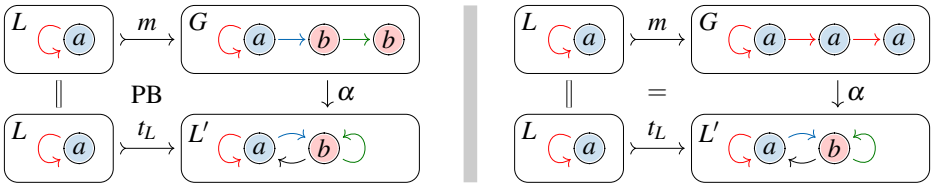
Thus, a PBPO$^+$ rule uses a type graph L' with a distinguished pattern graph L, specified by means of an injection $t_L : L \rightarrowtail L'$. A morphism $l' : L' \leftarrow K'$ is used to specify deletion and duplication on the type graph, similar to a ToyPB rule. A morphism $r : K \to R$ is then used to specify identification and addition specifically on $l'^{-1}(t_L)$, similar to a ToyPO rule.

For matching, we would like to view L' as providing a type environment *around* the designated pattern $t_L(L) \subseteq L'$. For rewrite steps, this means that *precisely one* occurrence of L in G should be mapped onto $t_L(L)$ by an adherence morphism $\alpha : G \to L'$. Stated in terms of preimages, we require that $\alpha^{-1}(t_L) \cong L$. We call such a match a *strong match*.

Definition 10 (Strong Match [14])**.** An *adherence morphism* $\alpha : G \to L'$ establishes a *strong match* for an embedding $t_L : L \rightarrowtail L'$ if the square depicted on the right is a pullback square. The induced morphism $m : L \rightarrowtail G$ is called the *match morphism*.

$$\begin{array}{ccc} L & \succ m \rightarrow & G \\ \| & \text{PB} & \downarrow{\scriptstyle\alpha} \\ L & \succ t_L \rightarrow & L' \end{array}$$

Example 6. The left of



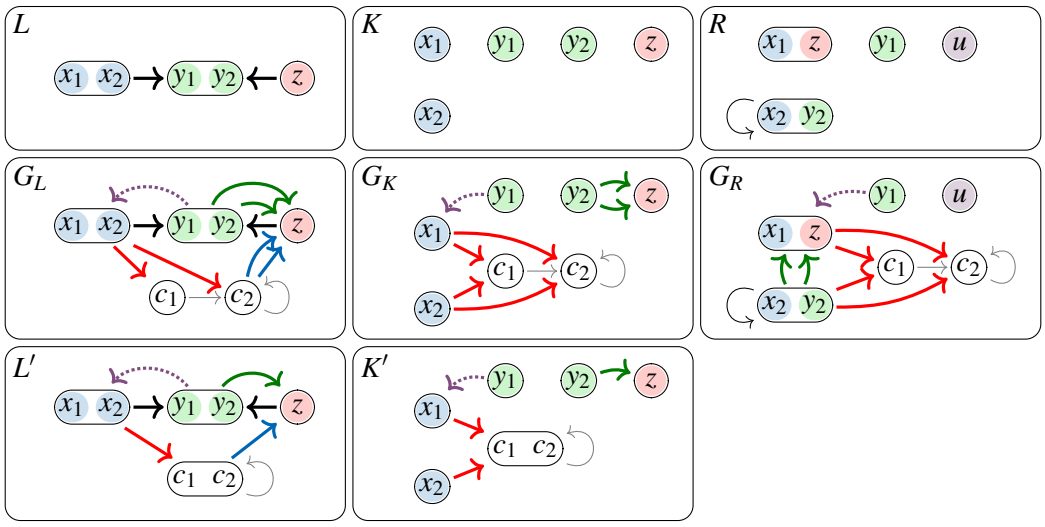
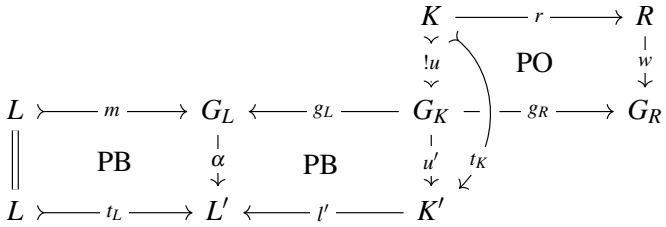


Figure 7: Example illustrating duplication, deletion and redirection and constraining application.

visualizes a strong match: L' has exactly one occurrence of L (namely $m(L)$) mapped onto $t_L(L)$. The square is a pullback square. The right is merely a commutative square, not a pullback square: because all of G is collapsed onto $t_L(L)$, the pullback object $\alpha^{-1}(t_L)$ is in fact G itself.

The definition of a PBPO$^+$ rewrite step consists of a strong match square, a ToyPB-like pullback square (using "rule" $l' : L' \leftarrow K'$), and a ToyPO-like pushout square (using "rule" $r : K \to R$). For this, the pullback and pushout squares need to be appropriately connected. As stated in the definition below, this can always be done, because the existence of a unique morphism $u : K \rightarrowtail G_K$ satisfying $t_K = u' \circ u$ is guaranteed.

Definition 11 (PBPO$^+$ Rewrite Step [14])**.** A PBPO$^+$ rewrite rule ρ (Definition 9), match morphism $m : L \rightarrowtail G_L$ and adherence morphism $\alpha : G_L \to L'$ induce a rewrite step $G_L \Rightarrow^{\rho,(m,\alpha)}_{\text{PBPO}^+} G_R$ if the properties indicated by the commuting diagram



hold, where $u : K \to G_K$ is the unique (and necessarily monic) morphism satisfying $t_K = u' \circ u$ [14, Lemma 11].[1]

Example 7 (Rewrite Step). The rewrite step $G_L \Rightarrow^{\rho,(m,\alpha)}_{\text{PBPO}^+} G_R$ depicted in Figure 7 illustrates some important features of PBPO$^+$. The rule ρ consists of the objects L, K, R, L' and K' together with the obvious morphisms. The pattern L requires any host graph G_L to contain three nodes, and two of these nodes have an edge targeting the third node. The graph L' describes the permitted shapes of the host graph

[1] Morphism u can be obtained by pulling back m along g_L. However, as shown in the cited lemma, the solution for u in $t_K = u' \circ u$ exists and is uniquely determined for PBPO$^+$ (it is not for PBPO [14, Remark 17]). We thus need not specify how u is constructed, allowing us to simplify the diagram.

around the pattern L. Due to the strong match condition, any $\alpha : G_L \to L'$ has to map all nodes and edges in the context $G_L - m(L)$ of the host graph nodes onto $L' - t_L(L)$. So, in particular, c_1c_2 in L' captures all the context nodes of G_L. Each edge in $L' - t_L(L)$ is a placeholder for an arbitrary number (0 or more) of edges in the host graph. This example illustrates the following features:

(i) *Application conditions:*

The graph L' allows for (an arbitrary number of) edges from x_1x_2 to the context, from the context to z, from y_1y_2 to x_1x_2 and from y_1y_2 to z (and edges among context nodes). Besides the edges in the pattern, any other edges are forbidden. For instance there cannot be edges from z to the context, and no additional edges from x_1x_2 to y_1y_2.

(ii) *Duplicating and deleting elements:*

The morphism $l' : K' \to L'$ enables the duplication and deletion of nodes and edges. For instance, from L' to K', the node x_1x_2 is duplicated along with its edges to the context (indicated in red). The edges from c_2 to x (indicated in blue) and the thick edges are deleted, because they do not lie in the image of l'.

(iii) *Identifying and adding elements:*

The morphism $r : K \to R$ enables the identification of elements of K, and the addition of new elements. Here, K is the result of restricting the duplication and deletion effects of l' to $t_L(L) \cong L$. In the example, r identifies (or merges) x_1 with z and x_2 with y_2, adds a fresh node u and a fresh edge from x_2y_2 to itself. In the middle row, observe that the sources and targets of edges are updated accordingly from G_K to G_R. For instance, the edges $y_1 \to x_1$ and $y_2 \to z$ in G_K are redirected to target the merged node x_1z in G_R.

(iv) *Edge redirection:*

The combination of duplication along $l' : K' \to L'$ and merging along $r : K \to R$ enables the arbitrary redirection of all those endpoints (source and target) of edges that lie in the pattern $m(L)$. Importantly, the edge itself does not need to be part of the pattern, only the endpoint to be redirected. For instance, the edges from y_1y_2 to z (indicated in green) are redirected to go from x_2 to x_1. This is achieved by first duplicating one endpoint of the edge, namely the source y_1y_2, and then merging the fresh source y_2 with x_2, and the target z with x_1.

For another example, see [15, Example 15].

5 Related Work

One of the first tutorials by Ehrig et al. [8] present intuitive approaches to DPO and SPO, also covering some metatheoretic properties. Baresi et al. [1] take a different approach and provide a broad and applied introduction to the graph transformation research field, introducing DPO set theoretically. The tutorial by Heckel [11] discusses a notion of typed graph transformation informally. The problem of dangling edges is highlighted, and some solutions that have been proposed to deal with them are discussed. The very recent tutorial by König et al. [12] explains the "essence" of DPO, and gives a gentle build-up towards pushouts. In their case, this involves giving a set-theoretic definition of pushouts. Only SPO is mentioned when the subject of deletion in unknown contexts is discussed. In addition, the tutorial discusses attributed graph rewriting and tools.

For a more extensive overview of tutorials, see [12, Section 7.1].

Acknowledgments

We thank Jasmin Blanchette, Wouter Brozius and Femke van Raamsdonk for discussions and feedback. We also thank the anonymous reviewers for their helpful suggestions. Both authors received funding from the Netherlands Organization for Scientific Research (NWO) under the Innovational Research Incentives Scheme Vidi (project. No. VI.Vidi.192.004).

References

[1] L. Baresi & R. Heckel (2002): *Tutorial Introduction to Graph Transformation: A Software Engineering Perspective*. In: *Proc. Conf. on Graph Transformation (ICGT)*, *LNCS* 2505, Springer, pp. 402–429, doi:10.1007/3-540-45832-8_30.

[2] M. Barr & C. Wells (1990): *Category theory for computing science*. Prentice Hall.

[3] A. Corradini, D. Duval, R. Echahed, F. Prost & L. Ribeiro (2015): *AGREE – Algebraic Graph Rewriting with Controlled Embedding*. In: *Proc. Conf. on Graph Transformation (ICGT)*, *LNCS* 9151, Springer, pp. 35–51, doi:10.1007/978-3-319-21145-9_3.

[4] A. Corradini, D. Duval, R. Echahed, F. Prost & L. Ribeiro (2019): *The PBPO Graph Transformation Approach*. *J. Log. Algebraic Methods Program.* 103, pp. 213–231.

[5] A. Corradini, T. Heindel, F. Hermann & B. König (2006): *Sesqui-Pushout Rewriting*. In: *Proc. Conf. on Graph Transformation (ICGT)*, *LNCS* 4178, Springer, pp. 30–45, doi:10.1007/11841883_4.

[6] H. Ehrig (1986): *Tutorial introduction to the algebraic approach of graph grammars*. In: *Proc. Workshop on Graph-Grammars and Their Application to Computer Science*, *LNCS* 291, Springer, pp. 3–14, doi:10.1007/3-540-18771-5_40.

[7] H. Ehrig, K. Ehrig, U. Prange & G. Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/3-540-31188-2.

[8] H. Ehrig, M. Korff & M. Löwe (1990): *Tutorial Introduction to the Algebraic Approach of Graph Grammars Based on Double and Single Pushouts*. In: *Proc. Workshop on Graph-Grammars and Their Application to Computer Science*, *LNCS* 532, Springer, pp. 24–37, doi:10.1007/BFb0017375.

[9] H. Ehrig, M. Pfender & H. J. Schneider (1973): *Graph-Grammars: An Algebraic Approach*. In: *Proc. Symp. on on Switching and Automata Theory (SWAT)*, IEEE Computer Society, p. 167–180, doi:10.1109/SWAT.1973.11.

[10] A. Habel, J. Müller & D. Plump (2001): *Double-pushout graph transformation revisited*. *Math. Struct. Comput. Sci.* 11(5), pp. 637–688, doi:10.1017/S0960129501003425.

[11] R. Heckel (2006): *Graph Transformation in a Nutshell*. *Electron. Notes Theor. Comput. Sci.* 148(1), pp. 187–198, doi:10.1016/j.entcs.2005.12.018.

[12] B. König, D. Nolte, J. Padberg & A. Rensink (2018): *A Tutorial on Graph Transformation*. In: *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig*, *LNCS* 10800, Springer, pp. 83–104, doi:10.1007/978-3-319-75396-6_5.

[13] M. Löwe (1993): *Algebraic Approach to Single-Pushout Graph Transformation*. *Theor. Comput. Sci.* 109(1&2), pp. 181–224, doi:10.1016/0304-3975(93)90068-5.

[14] R. Overbeek, J. Endrullis & A. Rosset (2021): *Graph Rewriting and Relabeling with PBPO$^+$*. In: *Proc. Conf. on Graph Transformation (ICGT)*, *LNCS* 12741, Springer, pp. 60–80, doi:10.1007/978-3-030-78946-6_4.

[15] R. Overbeek, J. Endrullis & A. Rosset (2022): *Graph Rewriting and Relabeling with PBPO$^+$: A Unifying Theory for Quasitoposes*. *CoRR* abs/2203.01032, doi:10.48550/arXiv.2203.01032. arXiv:2203.01032.

[16] B. C. Pierce (1991): *Basic category theory for computer scientists*. MIT press.