

Sesqui-Pushout Rewriting with Type Refinements

Michael Löwe^(✉)

FHDW Hannover, Freundallee 15, 30173 Hannover, Germany
`michael.loewe@fhdw.de`

Abstract. Sesqui-pushout rewriting is an algebraic graph transformation approach that provides mechanisms for vertex cloning. If a vertex gets cloned, the original and the copy obtain the same context, i.e. all incoming and outgoing edges of the original are copied as well. This behaviour is not satisfactory in practical examples which require more control over the context cloning process. In this paper, we provide such a control mechanism by allowing each transformation rule to refine the underlying type graph. We discuss the relation to the existing approaches to controlled sesqui-pushout vertex cloning, elaborate a basic theoretical framework, and demonstrate its applicability by a practical example.

1 Introduction

Sesqui-pushout graph transformation (SqPO) [2] is a relatively new variant in the family of algebraic graph rewriting frameworks. It extends the double-pushout (DPO) [4, 5] and the single-pushout approach (SPO) [6, 8] by mechanisms for object cloning including the complete context of the object, which are all incoming and outgoing edges in the case of graphs. Many practical examples, however, require more control over the cloning process. In many cases, only edges of specific types shall be cloned. In this paper, we propose a new mechanism for controlled object cloning by allowing each rule to refine the underlying type graph in a way that is suitable for the cloning performed by the rule.

The paper is organised as follows: Sect. 2 recapitulates sesqui-pushout rewriting in a categorical set-up and presents major results that shall be valid in any extension. The example in Sect. 3 motivates the mechanisms that are introduced in Sect. 4. Section 5 formulates a categorical framework for the new approach and shows that many results known from the standard approach carry over. Finally, Sect. 6 discusses related work and future research.

2 Standard SqPO-Rewriting

In this section, we present sesqui-pushout rewriting in a categorical set-up. We require that the underlying category \mathcal{C} satisfies the following conditions:

C1 \mathcal{C} has all finite limits and co-limits.

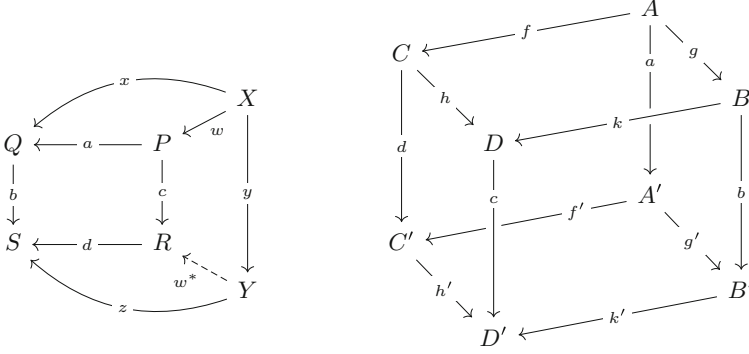


Fig. 1. Final pullback complement and commutative cube property

C2 Every pair $(a : P \rightarrow Q, b : Q \rightarrow S)$ of morphisms with monic b has a final pullback complement.

C3 Pushouts along monomorphisms are van-Kampen: In a commutative cube as in Fig. 1, where g' is monic, (h', k') is pushout of (f', g') , and (g, a) and (f, a) are pullbacks of (g', b) and (f', d) resp., we have: (h, d) and (k, b) are pullbacks of (h', c) and (k', c) resp., if and only if (h, k) is pushout of (f, g) .

A pair $(c : P \rightarrow R, d : R \rightarrow S)$ as in the left part of Fig. 1 is *final pullback complement (FPC)* of the pair $(a : P \rightarrow Q, b : Q \rightarrow S)$, if (a, c) is pullback of (b, d) and for each collection of morphisms (x, y, z, w) , where (x, y) is pullback of (b, z) and $a \circ w = x$, there is a unique w^* with $d \circ w^* = z$ and $c \circ w = w^* \circ y$. Note the following special cases of final pullback complement situations:

F1 For every morphism $f : P \rightarrow Q$, (id_P, f) is FPC of (f, id_Q) and vice versa.

F2 In a commutative cube as in the right part Fig. 1, where (b, k') is FPC of (k, c) , (f, g) is pullback of (k, h) , and (d, h) is pullback of (c, h') the following compatibility condition between pullbacks and final pullback complements holds: (a, f') is FPC of (f, d) , if and only if (f', g') is pullback of (k', h') .¹

Final pullback complements possess the following composition and decomposition properties (for proofs compare [11]):

F3 Horizontal composition and decomposition: Let $c \circ k = k' \circ b$ and $b \circ g = g' \circ a$ in the right part of Fig. 1 and let (b, k') be FPC of (k, c) : (a, g') is FPC of (g, b) , if and only if $(a, k' \circ g')$ is FPC of $(k \circ g, c)$.

F4 Vertical composition: If (g, b) and (k, c) are FPCs of (a, g') and (b, k') respectively in the right part of Fig. 1, then $(k \circ g, c)$ is FPC of $(a, k' \circ g')$.

Condition C3 guarantees:²

F5 Pushouts in \mathcal{C} preserve monomorphisms.

¹ For a proof of the if-part see [9].

² Compare [4, 7].

F6 Pushouts along monomorphisms are pullbacks.

For a compact notion of sesqui-pushout rewriting, we pass over from \mathcal{C} to the span category $\mathcal{C}^{\leftarrow\rightarrow}$ of \mathcal{C} . A *concrete span* is a pair of \mathcal{C} -morphisms $(p : K \rightarrow P, q : K \rightarrow Q)$. Two spans $(p_1, q_1), (p_2, q_2)$ are equivalent, if there is isomorphism i with $p_1 \circ i = p_2$ and $q_1 \circ i = q_2$; $[(p, q)]_{\equiv}$ denotes the class of spans equivalent to (p, q) . The *category of abstract spans* $\mathcal{C}^{\leftarrow\rightarrow}$ has the same objects as \mathcal{C} and equivalence classes of spans as morphisms. The identity for an object $A \in \mathcal{C}^{\leftarrow\rightarrow}$ is defined by $\text{id}_A^{\mathcal{C}^{\leftarrow\rightarrow}} = [(\text{id}_A, \text{id}_A)]_{\equiv}$. And composition of $[(p, q)]_{\equiv}$ and $[(r, s)]_{\equiv}$ is such that $\text{codomain}(q) = \text{codomain}(r)$ is given by $[(r, s)]_{\equiv} \circ_{\mathcal{C}^{\leftarrow\rightarrow}} [(p, q)]_{\equiv} = [(p \circ_C r', s \circ_C q')]_{\equiv}$ where (r', q') is a pullback of (q, r) . A span composition is *strong*, written $[(r, s)]_{\equiv} \bullet [(p, q)]_{\equiv}$, if (q', r) is final pullback complement of (r', q) .

Note that there is the natural and faithful embedding functor $\iota : \mathcal{C} \rightarrow \mathcal{C}^{\leftarrow\rightarrow}$ defined by identity on objects and $(f : A \rightarrow B) \mapsto [\text{id}_A : A \rightarrow A, f : A \rightarrow B]$ on morphisms. In the following, the composition of a span $(p, q) \in \mathcal{C}^{\leftarrow\rightarrow}$ with a morphism $m \in \text{mathcal{C}}$, i.e. $(p, q) \circ m$ (or $m \circ (p, q)$), is the span defined by $(p, q) \circ \iota(m)$ (resp. $\iota(m) \circ (p, q)$). By a slight abuse of notation, we write $[d : A' \rightarrow A, f : A' \rightarrow B] \in \mathcal{C}$ if d is an isomorphism. Direct derivations in sesqui-pushout rewriting are special strong compositions of spans.

Definition 1 (Standard Rule and Derivation). A rewrite rule p is a morphism in $\mathcal{C}^{\leftarrow\rightarrow}$, i.e. $p = (l : K \rightarrow L, r : K \rightarrow R)$. A match for p is a monic \mathcal{C} -morphism $m : L \rightarrow G$. The direct derivation with p at match m is constructed in two steps, compare Fig. 2:

1. $(m \langle l \rangle, l \langle m \rangle)$ is final pullback complement of (l, m) .
2. $(m \langle p \rangle, r \langle m \rangle)$ is pushout of $(m \langle l \rangle, r)$.

In a direct derivation, G is the source, $p@m$ is the target, the span $p \langle m \rangle = (l \langle m \rangle, r \langle m \rangle)$ is called the trace, and $m \langle p \rangle$ is also referred to as co-match.

Remarks. A derivation is determined up to isomorphism by the match. This is due to the fact that FPCs and pushouts are unique up to isomorphism. Since pullbacks preserve monomorphisms, $m \langle l \rangle$ is a monomorphism and Fact F5 provides monic $m \langle p \rangle$. Due to Condition C2, rules are applicable at every match. The direct derivation in Fig. 2 constitutes a special commutative diagram in $\mathcal{C}^{\leftarrow\rightarrow}$, i.e. $m \langle p \rangle \bullet p = p \langle m \rangle \bullet m$, with pullback $(l, m \langle l \rangle)$ of $(l \langle m \rangle, m)$.

Since traces are morphisms in $\mathcal{C}^{\leftarrow\rightarrow}$, they can be used as rules and, by Fact F4 and the composition property of pushouts, we immediately obtain:

$$\begin{array}{ccccc}
 L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
 m \downarrow & & \downarrow m \langle l \rangle & & \downarrow m \langle p \rangle \\
 G & \xleftarrow{l \langle m \rangle} & K \langle m \rangle & \xrightarrow{r \langle m \rangle} & p@m
 \end{array}
 \quad
 \begin{array}{ccc}
 & (1) & (2)
 \end{array}$$

Fig. 2. Direct transformation

Proposition 2 (Standard Derived Rule). *If $p \langle m \rangle$ is the trace of a derivation with rule p at match m and n is match for $p \langle m \rangle$, then $(p \langle m \rangle) \langle n \rangle = p \langle n \circ m \rangle$ and $(n \circ m) \langle p \rangle = n \langle p \langle m \rangle \rangle \circ m \langle p \rangle$.*

Since rules are spans, they can be composed and decomposed. Condition C3 guarantees that rule composition and decomposition carries over to derivations.

Proposition 3 (Composition of Standard Derivations). *If m is a match for $p' \circ p$, $(p' \circ p) \langle m \rangle = p' \langle m \langle p \rangle \rangle \circ p \langle m \rangle$ and $m \langle p' \circ p \rangle = m \langle p \rangle \langle p' \rangle$.*

The proposition is a direct consequence of Theorem 7 in [11]. Together with Proposition 2, it provides the fundamentals for a rich theory.³

3 Example: Version Management

As an example that demonstrates the power of SqPO-rewriting, we present a model for version management of decomposed components. It uses the category \mathcal{G} of graphs and graph morphisms or, more precisely, the slice category $\mathcal{G} \downarrow T$ of all graphs wrt. a type graph $T \in \mathcal{G}$. A *graph* $G = (V; E; s, t : E \rightarrow V)$ has a set V of vertices, a set E of edges, and source and target mappings s and t . A *graph morphism* $h : G \rightarrow H$ is a pair $(h_V : G_V \rightarrow H_V, h_E : G_E \rightarrow H_E)$ of mappings with $s_H \circ h_E = h_V \circ s_G$ and $t_H \circ h_E = h_V \circ t_G$. \mathcal{G} satisfies Conditions C1–C3.⁴

The left part of Fig. 3 depicts the type graph for the version management system. The right part shows a sample instance graph.⁵ The sample contains

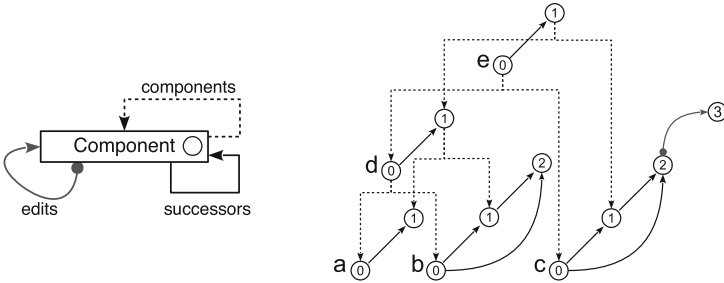


Fig. 3. Version management: model and instance

³ Compare for example [11].

⁴ Given arbitrary morphism $a : P \rightarrow Q$ and monomorphism $b : Q \rightarrow S$, the final pullback complement $(c : P \rightarrow R, d : R \rightarrow S)$ is constructed as follows, compare [2]: **Vertices:** $R_V = P_V \uplus (S_V - b_V(Q_V))$, $c_V = \text{id}_{P_V}$, $d_V = b_V(a_V(v))$ if $v \in P_V$ and $d_V = \text{id}_{S_V}$ otherwise. **Edges:** R_E contains P_E and an edge “copy” (v, e, v') for every edge $e \in S_E - b_E(Q_E)$ and pair of vertices $v, v' \in R_V$ with $s_S(e) = d_V(v)$ and $t_S(e) = d_V(v')$ with the following structure: $s_R(v, e, v') = v$ and $s_R(e) = s_P(e)$ if $e \in P_E$, $t_R(v, e, v') = v'$ and $t_R(e) = t_P(e)$ if $e \in P_E$, $c_E = \text{id}_{P_E}$, and $d_E(v, e, v') = e$ and $d_E(e) = b_E(a_E(e))$ if $e \in P_E$.

⁵ The typing is indicated by the graphical symbols.

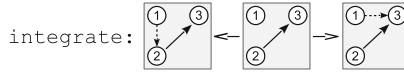


Fig. 4. Evolution of decomposed components

5 components, the components **a**, **b** and **c** are elementary and the components **d** and **e** are decomposed into **a** and **b** respectively **d** and **c**. In the example, only component **c** has an editable version, namely **c3**. Decomposed components evolve by integrating successor versions of their components. The corresponding rule is depicted in Fig. 4.

If a new editable version \mathbf{x}' of a component \mathbf{x} is created, it shall integrate the same components as \mathbf{x} . Here the copy mechanism of sesqui-pushout rewriting shall be applicable. But it is not, since \mathbf{x}' shall only have copies of the outgoing **components**-edges of \mathbf{x} but neither of the incoming **components**-edges nor of outgoing or incoming **successor**-edges. A similar problem arises, when an editable version \mathbf{x}' gets ready to be published. Then it shall become successor of the version \mathbf{x} it has been spun off and of all predecessor versions of that \mathbf{x} . This means that the **successor**-relation shall be transitive, for example to skip some versions in component integration, compare Fig. 4. Here, we do not need a complete copy of \mathbf{x} , we only need a copy of all **successor**-edges pointing to \mathbf{x} .

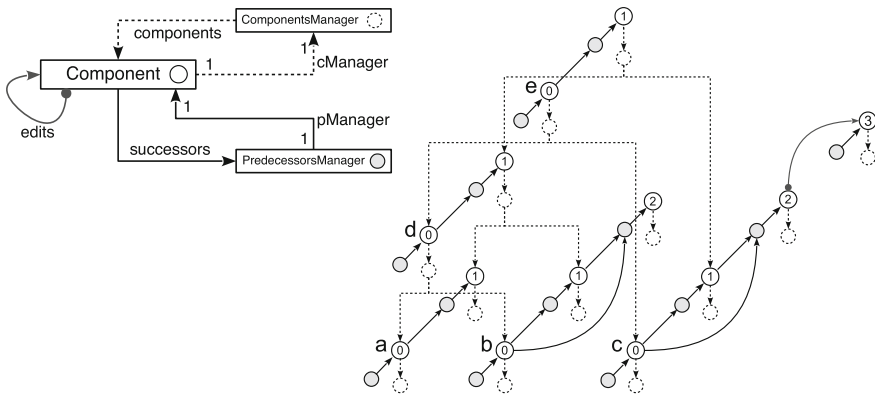


Fig. 5. Refined version management: model and instance

Problems of this type can be tackled by using a refined type graph. The left part of Fig. 5 shows a suitable refinement for the problems described above. The right part of the figure shows the instance of Fig. 3 in the refined version. The trick is that we provide internal structure to each component by two one-to-one relations. Now, we have a port-object for every **Component** that handles incoming **successors**-edges, namely a node the type of which is **PredecessorsManager**, and a port that handles outgoing **components**-edges, namely a node of type

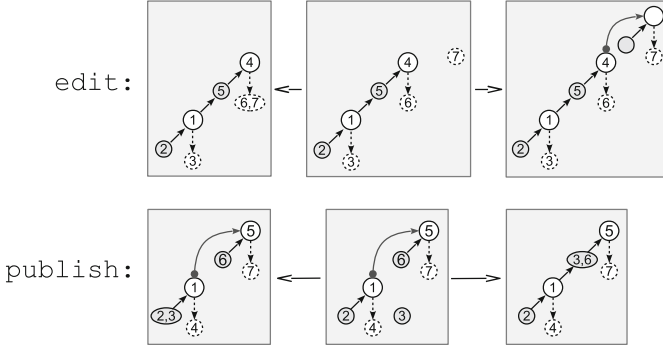


Fig. 6. Rules for editing and publishing

ComponentsManager. Having these structures at hand, we can formulate the rules for the creation of a new and for the publication of an existing editable version, compare Fig. 6.⁶

Although this approach works quite satisfactory, there are a lot of drawbacks of such a *global* type graph refinement. First of all, the model gets more complex and many additional consistence conditions come into play, like the one-to-one relations in Fig. 5, that have to be preserved by all rules. Thus, rules get more complex as well, even those rules that had no need for a refinement. E.g. the rule in Fig. 4 must be reengineered in order to conform to the refined model.

In order to tackle the problem of *partial copies* more adequately, we propose to stick to a simple global type graph and allow each rule to perform the refinements which it needs *locally*. In such a framework, that is elaborated in the following, a rule like **integrate** in Fig. 4 is perfect while the **edit**- and the **publish**-rules in Fig. 6 can use smaller more dedicated refinements. We will come back to the example later.

4 SqPO-Rewriting with Local Type Refinement

In this section, we introduce the sesqui-pushout rewriting framework that allows individual type refinements for every rule. Again, the set-up is purely categorical. Besides Conditions C1–C3, we need:

C4 The underlying category \mathcal{C} has epi-mono-factorisations.

C5 Pullbacks in \mathcal{C} preserve epimorphisms.

By \mathcal{C}^\rightarrow , we denote the arrow category over \mathcal{C} .⁷ And, for any given (type) object $T \in \mathcal{C}$, $\mathcal{C} \downarrow T$ denotes the slice category of all objects under T .⁸

⁶ Note the **edit**-rule copies the node labelled “6, 7” from the left- to the right-hand side, and the **publish**-rule copies the node labelled “2, 3” from left to right.

⁷ The objects of \mathcal{C}^\rightarrow are all morphisms of \mathcal{C} and a morphism i from $f : A \rightarrow B$ to $g : C \rightarrow D$ is a pair $(i_A : A \rightarrow C, i_B : B \rightarrow D)$ of \mathcal{C} -morphisms such that $i_B \circ f = g \circ i_A$.

⁸ $\mathcal{C} \downarrow T$ is the restriction of \mathcal{C}^\rightarrow to morphisms with co-domain T . Note that every category \mathcal{C} in our set-up is equivalent to $\mathcal{C} \downarrow F$, where F is the final object in \mathcal{C} .

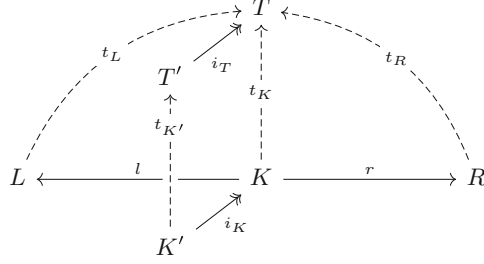


Fig. 7. Rule

Definition 4 (Rule). Given an object T in \mathcal{C} , a T -typed rewrite rule $p = ((l : t_K \rightarrow t_L, r : t_K \rightarrow t_R), i : t_{K'} \twoheadrightarrow t_K)$ consists of a $\mathcal{C} \downarrow T$ -span (l, r) and an epic⁹ type refinement $i \in \mathcal{C}^{\twoheadrightarrow}$, compare Fig. 7.

A rule is applicable at a monic match morphism $m : t_L \rightarrow t_G$. The derivation shall comprise 3 phases, namely (1) the refinement of the rule and the match to the type T' , (2) the SqPO-rewriting with the refined rule at the refined match, and (3) the abstraction of the derivation back to type T .

Definition 5 (Derivation). The direct derivation with a rewrite rule $p = (l : t_K \rightarrow t_L, r : t_K \rightarrow t_R, i : t_{K'} \twoheadrightarrow t_K)$ at a monic match morphism $m : t_L \rightarrow t_G$ consists of the trace $p(m) = (l \langle m \rangle : t_{K \langle m \rangle} \rightarrow t_G, r \langle m \rangle : t_{K \langle m \rangle} \rightarrow t_{p@m})$ and the co-match $m \langle p \rangle : t_R \rightarrow t_{p@m}$ in $\mathcal{C} \downarrow T$ together with an epic type refinement $i \langle m \rangle : t'_{K' \langle m' \rangle} \twoheadrightarrow t_{K \langle m \rangle}$ which are constructed as follows, compare Fig. 8:

1. Refinement: Let $(i_L, t'_{L'})$ and $(i_R, t'_{R'})$ be the pullbacks of (i_T, t_L) and (i_T, t_R) resp. and l' and r' the morphisms making the diagram commute. Call $p' = (l', r')$ the i -refined rule of $p = (l, r)$. Let $(i_G, t'_{G'})$ be the pullback (i_T, t_G) and m' the unique morphisms such that (i_L, m') is pullback of (i_G, m) .
2. Derivation: Let $(l \langle m \rangle_A, r \langle m \rangle_A) : G \rightarrow p@m_A$ be the trace and $m \langle p \rangle_A : R \rightarrow p@m_A$ the co-match of the sesqui-pushout derivation with p at m . Let $(l' \langle m' \rangle, r' \langle m' \rangle) : G' \rightarrow p'@m'$ be the trace and $m' \langle p' \rangle : R' \rightarrow p'@m'$ the co-match of the sesqui-pushout derivation with p' at m' . And let $i \langle m \rangle_A$ and $i_{p@m_A}$ be the morphisms into the final pullback complement of (l, m) and from the pushout of $(r', m' \langle l' \rangle)$ making the resulting diagram commute.
3. Abstraction: Construct $(i \langle m \rangle, d)$ and $(i_{p@m}, d_A)$ as epi-mono-factorisations of $i \langle m \rangle_A$ and $i_{p@m_A}$. Set $l \langle m \rangle = l \langle m \rangle_A \circ d$. And, finally, let $m \langle l \rangle$, $r \langle m \rangle$, and $m \langle p \rangle$ be the diagonal morphisms making the diagram commute.

Although the construction for a direct derivation is rather complex, it possesses good properties that can lead to a rich theory and are investigated in the following. The first result is obvious, namely that the new rewriting mechanism subsumes simple sesqui-pushout rewriting.

⁹ I.e. both components are epimorphisms.

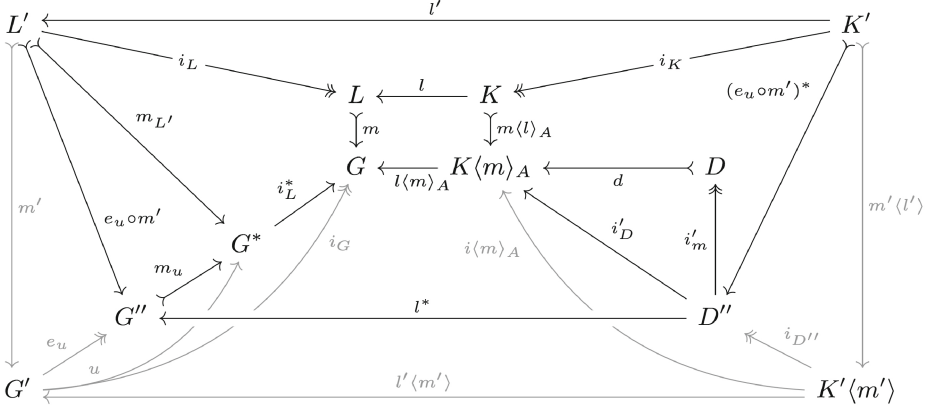


Fig. 9. Local derivation

Since $d_X \circ e_X \circ m' \langle p' \rangle = d_X \circ m \langle l \rangle^* \circ i_R = m \langle p \rangle_A \circ i_R = i_{p@m_A} \circ m' \langle p' \rangle$ and $d_X \circ e_X \circ r' \langle m' \rangle = d_X \circ r^* \circ i \langle m \rangle = r \langle m \rangle_A \circ d \circ i \langle m \rangle = r \langle m \rangle_A \circ i \langle m \rangle_A = i_{p@m_A} \circ r' \langle m' \rangle$, we can conclude that $d_X \circ e_X = i_{p@m_A}$.

Therefore, (e_X, d_X) is an epi-mono-factorisation of $i_{p@m_A}$ which provides $X \cong p@m$. Uniqueness of diagonals leads to $r^* = r \langle m \rangle$ and $m \langle l \rangle^* = m \langle p \rangle$. \square

We conclude this section by an important observation, namely that a derivation in sesqui-pushout rewriting with local type refinements has local effects only. For a rewrite, we do not have to refine the complete source object, it is sufficient to refine the part that is in the image of the match.

Consider Fig. 9. It depicts the left-hand side of a rule, namely $l : K \rightarrow L$, its refinement $l' : K' \rightarrow L'$, the match $m : L \rightarrow G$, and the refined match $m' : L' \rightarrow G'$. While Definition 5 constructed the final pullback complement $(m' \langle l' \rangle, l' \langle m' \rangle)$ of l' and m' (grey in Fig. 9), we now construct the “local” FPC $(m_{L'} : L' \rightarrow G^*, i_L^* : G^* \rightarrow G)$ of the match m and the refinement of the rule’s left-hand side, namely i_L . Since (m', i_L) is pullback of (i_G, m) , we obtain morphism u which makes the diagram commute. By factorisation of u into epic $e_u : G' \rightarrow G''$ and monic $m_u : G'' \rightarrow G^*$, we construct a local refinement of G , i.e. $i_L^* \circ m_u : G'' \rightarrow G$, and a locally refined match $e_u \circ m'$.¹¹ Note that $(id_{L'}, e_u \circ m')$ is pullback of $(m_u, m_{L'})$, since m_u is monic, and $(id_{L'}, m')$ is pullback of $(e_u, e_u \circ m')$ by decomposition of pullbacks. Now, construct the FPC $((e_u \circ m')^*, l^*)$ of $(l', e_u \circ m')$ which provides the morphism i_D' making the diagram commute.

We show that the epi-mono-factorisation $(i_m' : D'' \twoheadrightarrow D, d : D \rightarrow K \langle m \rangle_A)$ of this morphism provides the same sub-object of $K \langle m \rangle_A$ as the epi-mono-factorisation of $i \langle m \rangle_A$. The argument is straightforward, since there is the morphism $i_{D''} : K' \langle m' \rangle \rightarrow D''$ mediating between the “global” and the “local” FPC and making the diagram commute. Since $(id_{L'}, m')$ is pullback of $(e_u, e_u \circ m')$, $(id_{K'}, l')$ is

¹¹ Note that the object G'' cannot be typed in the refined type of the rule, since it contains unrefined parts, namely the parts outside $m(L)$.

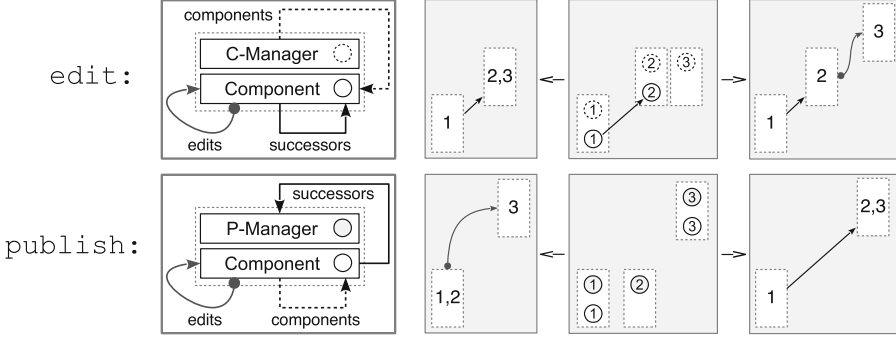


Fig. 10. Rules for editing and publishing – refined version

pullback of $(l', \text{id}_{L'})$, and $(m' \langle l' \rangle, l' \langle m' \rangle)$ as well as $((e_u \circ m')^*, l^*)$ are FPCs, Fact F2 makes sure that $(i_{D''}, l' \langle m' \rangle)$ is pullback of (e_u, l^*) and Condition C5 guarantees that i_D'' is epimorphism. Therefore the pair $(i_m' \circ i_{D''}, d)$ is epi-mono-factorisation of $i \langle m \rangle_A$, q.e.d.

Example 8 (Version Management – Refined Version). Using the mechanisms introduced in this section, we can specify the rules for the version management system described in Sect. 3: We use the standard type graph given in Fig. 3. The rule for component integration is given in Fig. 4. It does not specify any type-refinement and works as it is. The rules for editing and publishing are depicted in Fig. 10. The **edit**-rule refines the type graph such that outgoing **components**-relations of a component can be handled separately, compare middle object of the rule span for **edit** in Fig. 10. The **publish**-rule uses a different type refinement such that incoming **successors**-relations can be handled and copied separately.

5 The Category of Type-Refined Spans

In this section, we define a composition operator on rewrite rules which leads to the category of type-refined spans. We show that rule composition and decomposition carries over to composition and decomposition of direct derivations, i.e. that theorems like Propositions 2 and 3 are also valid in the new framework.

Let $T \in \mathcal{C}$ be a fixed type object. Two rules

$$p_1 = ((l_1 : t_{K_1} \rightarrow t_L, r_1 : t_{K_1} \rightarrow t_R), (i_{T'}, i_1) : (t_{K'_1} : K'_1 \rightarrow T') \rightarrow t_{K_1}) \text{ and} \\ p_2 = ((l_2 : t_{K_2} \rightarrow t_L, r_2 : t_{K_2} \rightarrow t_R), (i_{T''}, i_2) : (t_{K'_2} : K'_2 \rightarrow T'') \rightarrow t_{K_2})$$

with common domain and co-domain are equivalent if there is a triple

$$(j_K : K_2 \rightarrow K_1, j_{K'} : K'_2 \rightarrow K'_1, j_T : T'' \rightarrow T')$$

of isomorphisms, such that the resulting diagram commutes, i.e. $l_1 \circ j_K = l_2$, $r_1 \circ j_K = r_2$, $t_{K_1} \circ j_K = t_{K_2}$, $i_{T'} \circ j_T = i_{T''}$, $j_T \circ t_{K'_2} = t_{K'_1} \circ j_{K'}$, and $j_K \circ i_2 =$

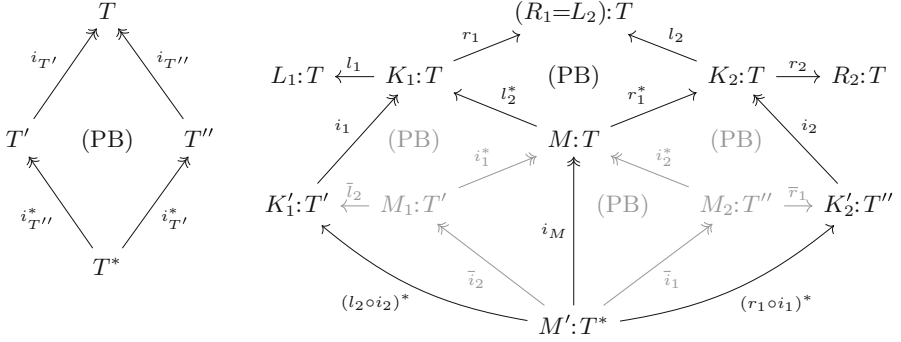


Fig. 11. Rule composition

$i_1 \circ j_{K'}$. In this case, we write $p_1 \equiv p_2$ and denote the class of rules that are equivalent to p by $[p]_{\equiv}$. Such a class is called *abstract type-refined span*. These spans can be composed.

Definition 9 (Composition). Given two abstract typed-refined spans $p_1 = [((l_1, r_1), (i_{T'}, i_1))]$ and $p_2 = [((l_2, r_2), (i_{T''}, i_2))]$ such that the co-domain of r_1 coincides with the domain of l_2 , the composition $p_2 \circ p_1$ of p_1 and p_2 is defined by $[(((l_1 \circ l_2^*, r_2 \circ r_1^*)(i_{T'} \circ i_{T''}^*, i_M)))]$, where $(i_{T''}^* : T^* \rightarrow T', i_{T'}^* : T^* \rightarrow T'')$, $(r_1^* : M \rightarrow K_2, l_2^* : M \rightarrow K_1)$, and $((r_1 \circ i_1)^* : M' \rightarrow K_2', (l_2 \circ i_2)^* : M' \rightarrow K_1')$ are the pullbacks of the pairs $(i_{T'}, i_{T''})$, (l_2, r_1) , and $(l_2 \circ i_2, r_1 \circ i_1)$ resp. and $t_M : M \rightarrow T$, $t_{M'} : M' \rightarrow T^*$, and $i_M : M' \rightarrow M$ are the unique morphisms making the diagram commute, compare Fig. 11.

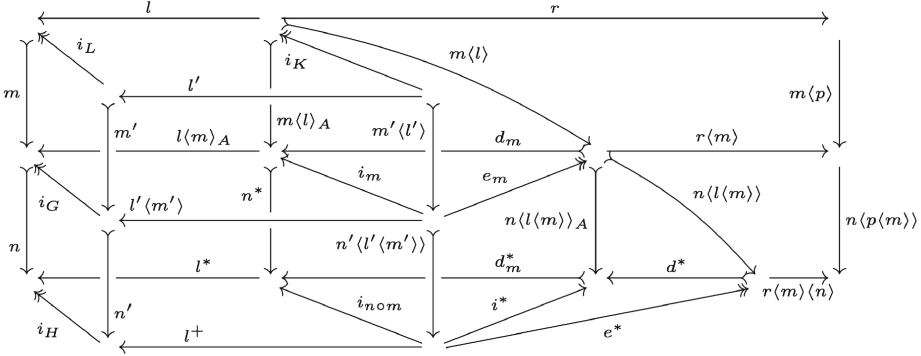
Note that the composition operator is defined independent of the choice of representatives, since pullbacks of isomorphic diagrams are isomorphic. The composition is associative due to composition/decomposition properties of pullbacks.

The composition operator of Definition 9 gives rise to the category $\mathcal{C}^{\leftarrow \uparrow \rightarrow T}$ of abstract type-refined spans under T , which has the same objects as $\mathcal{C} \downarrow T$ and abstract type-refined spans of type T as morphisms. The identity on $A \in \mathcal{C} \downarrow T$ is given by $\text{id}_A = [((\text{id}_A, \text{id}_A), (\text{id}_T, \text{id}_A))]$. We call a type-refined span $p = [((l, r), i)]$ *total*, if l and i are isomorphisms and *co-total* if r is isomorphism. Note that any morphism in $\mathcal{C} \downarrow T$ one to one corresponds to a total morphism in the category of abstract type-refined spans. With this categorical background, a rewrite rule p is just a morphism, a match is a monic and total morphism, and a direct derivation is a special commutative diagram.

Definition 10 (Refined Trace). Let $p(m)$ be the trace, $m \langle p \rangle$ the co-match, and $i \langle m \rangle$ the type refinement in a derivation with rule p at match m as given in Definition 5. Then the $\mathcal{C}^{\leftarrow \uparrow \rightarrow T}$ -morphism $p \langle m \rangle = (p(m), i \langle m \rangle)$ is called *refined trace*.

Proposition 11 (Direct Derivation). If $p \langle m \rangle$ and $m \langle p \rangle$ are trace and co-match in a derivation with rule p at match m , then $p \langle m \rangle \circ m = m \langle p \rangle \circ p$.

Theorem 12 (Derived Rule). *If $p \langle m \rangle$ and $m \langle p \rangle$ are trace and co-match in a derivation and n is match for $p \langle m \rangle$, then $(p \langle m \rangle) \langle n \rangle = p \langle n \circ m \rangle$ and $(n \circ m) \langle p \rangle = n \langle p \langle m \rangle \rangle \circ m \langle p \rangle$.*



Proof. The situation is depicted in Fig. 12: The rule $p = (l, r, i_K)$ has been applied at match m resulting in trace $(l\langle m \rangle_A \circ d_m, r\langle m \rangle, e_m)$ and the co-match $m\langle p \rangle$, where (e_m, d_m) is the factorisation of the morphism i_m the mediator between the two FPCs of the derivation $p @ m$. The application of $p\langle m \rangle$ at match n results in the trace $(l^* \circ d_m^* \circ d^*, r\langle m \rangle\langle n \rangle, e^*)$ and co-match $n\langle p\langle m \rangle \rangle$, where $(n\langle l\langle m \rangle \rangle_A, l^* \circ d_m^*)$ is FPC of $(l\langle m \rangle = l\langle m \rangle_A \circ d_m, n)$, $(n'\langle l'\langle m' \rangle \rangle, l^+)$ is FPC of $(l'\langle m' \rangle, n')$, and (e^*, d^*) is the factorisation of i^* which is the mediator between these two pullback complements. Due to Condition C2 and Fact F3, we can decompose the pullback complement $(n\langle l\langle m \rangle \rangle_A, l^* \circ d_m^*)$ into two FPCs, i.e. (n^*, l^*) and $(n\langle l\langle m \rangle \rangle_A, d_m^*)$. Since FPCs preserve monomorphisms¹², d_m^* is monic. By Fact F4, $(n^* \circ m\langle l \rangle_A, l^*)$ and $(n'\langle l'\langle m' \rangle \rangle \circ m'\langle l' \rangle, l^+)$ are FPCs. Thus, we obtain the mediator i_{nom} which is subject to an epi-mono-factorisation

¹² Compare [10].

Constructing the four FPCs of (m, l_1) , (m', l'_1) , (m^+, \bar{l}_1) , and (m'', l''_1) results in $(\bar{r}'_2, \bar{r}_1^*)$ being pullback of (r'_2, r_1^*) and r'_2 being a refinement wrt. the refined type of p_2 . The refinement r_2 used in the derivation with p_2 can be constructed as the pullback of d_1 and r'_2 . This provides e'_1 making the diagram commute and (e'_1, \bar{r}'_2) the pullback of (r_2, e_1) . By Condition C5, e'_1 is epic.

Now the FPCs $(d''_1, l_2^{*'})$ of (l_2^*, d_1) , (d_1^*, l_2^+) of $(l_2^{*'}, d'_1)$, and (\bar{m}'', l''_2^*) of $(l''_2, m'')^*$ lead to the morphism $r_2^{*'} \circ d_1^* = d''_1 \circ r_2^*$ and $r_2^{*'} \circ r'_1$ is the morphism which has to be epi-mono-factored in the derivation with $p_2 \circ p_1$ at match m . Since $(m'')^*, \bar{l}_1^*)$ is pullback of (\bar{m}', e'_1) , we get e_1^* with $l_2^{*'} \circ e_1^* = e'_1 \circ l_2^{*'} \circ e_1^*$ and $e_1^* \circ \bar{m}'' = \bar{m}' \circ \bar{l}_1^*$. We also get $d_1^* \circ e_1^* = r'_1$ since both morphisms are into a final pullback complement. And Fact F2 implies that $(e_1^*, l_2^{*'})$ is pullback of $(e'_1, l_2^{*'})$ such that e_1^* is epic due to Condition C5. But now $(e_2 \circ e_1^*, d_1^* \circ d_2)$ is epi-mono-factorisation of $r_2^{*'} \circ r'_1$. \square

Lemma 14. *If $p_2 \circ p_1$ is the composition of a total morphism p_1 and a co-total morphism p_2 , then $(p_2 \circ p_1) \langle m \rangle = p_2 \langle m \langle p_1 \rangle \rangle \circ p_1 \langle m \rangle$ and $m \langle p_2 \circ p_1 \rangle = m \langle p_1 \rangle \langle p_2 \rangle$, for every match m for p_1 .*

Proof. The situation of this lemma is depicted in Fig. 14: $(n, r_1 \langle m \rangle)$ is pushout of (r_1, m) , i.e. constitutes the direct derivation with rule p_1 at match m . The co-total rule p_2 is represented by (l_2, i_2) . The pullbacks (l_2^*, r_1^*) of (r_1, l_2) and (\bar{r}_1, \bar{i}_2) of (i_2, r_1^*) define the rule $p_2 \circ p_1 = (l_2^*, r_1^*, \bar{i}_2)$. The application of this composition at match m is defined by the trace $(l_2^* \langle m \rangle \circ d^*, r_1^* \langle m \rangle, i \langle m \rangle)$ and the co-match $n \langle p_2 \circ p_1 \rangle$, i.e. \bar{i}_2^* is the FPC-mediator of the derivation, $(i \langle m \rangle, d^*)$ its epi-mono-factorisation, and $(r_1^* \langle m \rangle, n \langle p_2 \circ p_1 \rangle)$ is pushout of $(m \langle l_2^* \rangle, r_1^*)$. Let, finally, i_2^* be the FPC-mediator of the derivation with p_2 at $n = m \langle p_1 \rangle$.

Consider the cube defined by refinements i_L , i_G , i'_G , and i'_L . Its top, back, front, and bottom faces are pullbacks and the right face is pushout and pullback by Fact F6. Condition C3 implies that the left face is pushout and pullback. Now consider the inner and outer cube in Fig. 14 defined by $(l_2, l_2^*, l_2 \langle n \rangle_A, l_2^* \langle m \rangle)$ and $(l'_2, l_2^{*'}, l_2 \langle n' \rangle, l_2^{*'} \langle m' \rangle)$. In both cubes, the left face is pushout and pullback, the top face is pullback, and the front and back faces are FPCs. By Condition C3 and Fact F2, their bottom faces are pullbacks and their right faces are pushouts and pullbacks. Therefore, we obtain monic d , pushout (d, g') of $(d^*, r_1^* \langle m \rangle)$, and $i \langle n \rangle$ with $i \langle n \rangle \circ \bar{r}'_1 = r_1^* \langle m \rangle \circ i \langle m \rangle$ and $i \langle n \rangle \circ n' \langle l'_2 \rangle = n \langle p_2 \circ p_1 \rangle \circ i_2$. We know $d \circ i \langle n \rangle = i_2^*$, since both morphisms coincide under prefix composition with \bar{r}'_1 and $n' \langle l'_2 \rangle$. A similar argument as in Proposition 7 shows that $i \langle n \rangle$ is epimorphism and $(i \langle n \rangle, d)$ is epi-mono-factorisation of i_2^* . \square

Theorem 15 (Composition). *If m is a match for $p' \circ p$, then $(p' \circ p) \langle m \rangle = p' \langle m \langle p \rangle \rangle \circ p \langle m \rangle$ and $m \langle p' \circ p \rangle = m \langle p \rangle \langle p' \rangle$.*

Proof. Consequence of Lemmata 13 and 14 and the fact that pushouts compose.

Theorems 12 and 15 demonstrate that the extension of sesqui-pushout rewriting presented in this paper is as well-behaved as the standard approach as far as rule composition and decomposition is concerned. This provides a good fundament for future research wrt. subrules, remainders and amalgamation.

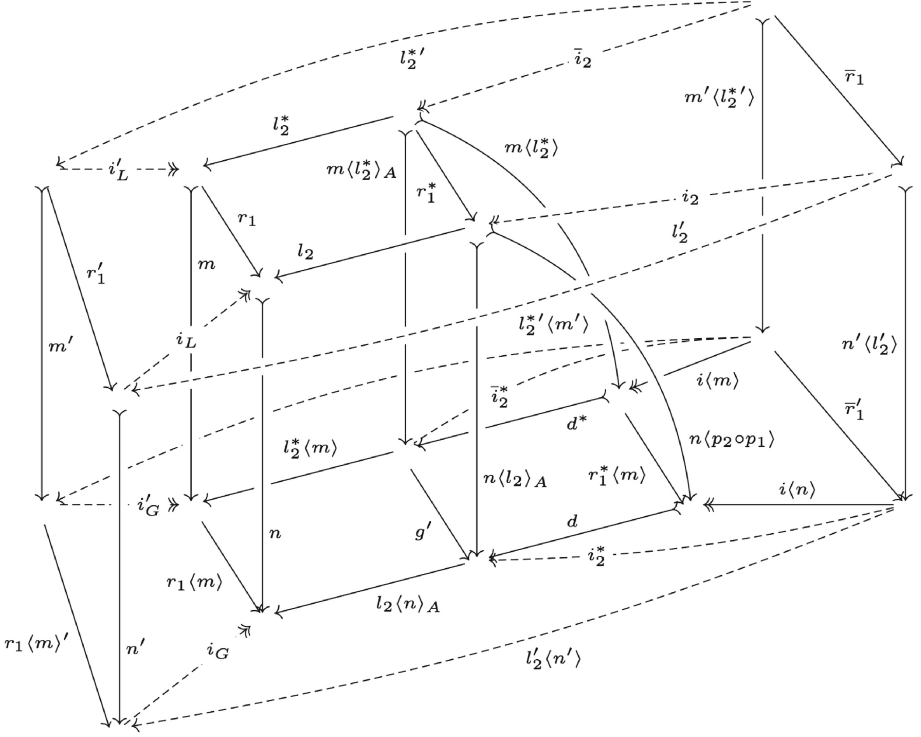


Fig. 14. Derivation composition of total and co-total morphism

6 Related Work and Future Research

There are two major other approaches to controlled sesqui-pushout cloning, namely rewriting on polarised graphs [3] and the AGREE framework [1]. Sesqui-pushout rewriting of polarised graphs allows to specify for every vertex in the middle graph K of a rule $(l : K \rightarrow L, r : K \rightarrow R)$ if it allows incoming edges only, outgoing edges only, or incoming and outgoing edges. This specifies, whether a copy of a vertex obtains the full context, the incoming context only, or the outgoing context only. Thus, SqPO-rewriting of polarised graphs is a special case of the mechanism presented here. Polarisation of graphs can be represented by passing from $\mathcal{G} \downarrow F$, i.e. the comma category of all graphs under the final graph (node with a singleton loop) to $\mathcal{G} \downarrow E$ where E is the graph with two vertices connected by a singleton edge. E is a refinement of F in our sense.

AGREE controls the cloning by a monomorphism $t : K \hookrightarrow T_K$ from middle object K of a rule $(l : K \rightarrow L, r : K \rightarrow R)$ typically into a subobject of the partial arrow classifier K^* of K . By interpreting a monic match $m : L \hookrightarrow G$ as a partial arrow $m' = (m, \text{id}_L)$ from the source object G and $l' = (t, l)$ as a partial arrow from T_K into the rules left-hand side, the cloning in AGREE is performed by the pullback of $\bar{m} : G \rightarrow L^*$ and $\bar{l} : T_K \rightarrow L^*$ which are the totalisation

of m' and l' into the partial arrow classifier L^* of L . If $t : K \rightarrow T_K$ is the complete partial arrow classifier, a complete copy is performed. By choosing T_K as a proper subobject of K^* , the copy process can be controlled very precisely. In graphs for example, it can be specified that two cloned vertices possess the same context wrt. third vertices but do not have cloned edges between themselves. This is not possible in our approach, and has to be substituted by some sort of polymorphism, compare [11]. Nevertheless, the introduced rewriting approach is an interesting alternative to AGREE. On the one hand, it can simulate the AGREE-effects at least wrt. unknown context (outside the image of the match). And on the other hand, while there are very few theoretical results available for AGREE (at least for the time being), the results wrt. vertical and horizontal composition in this paper provide solid hints that the theory of standard SqPO-rewriting can be generalised to SqPO-rewriting with local type refinement. Work in this direction will be a major topic of future research.

References

1. Corradini, A., Duval, D., Echahed, R., Prost, F., Ribeiro, L.: AGREE – algebraic graph rewriting with controlled embedding. In: Parisi-Presicce, F., Westfechtel, B. (eds.) ICGT 2015. LNCS, vol. 9151, pp. 35–51. Springer, Heidelberg (2015)
2. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer, Heidelberg (2006)
3. Duval, D., Echahed, R., Prost, F.: Graph transformation with focus on incident edges. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 156–171. Springer, Heidelberg (2012)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
5. Ehrig, H., Pfender, M., Schneider, H.J., Graph-grammars: an algebraic approach. In: FOCS, pp. 167–180. IEEE (1973)
6. Kennaway, R.: Graph rewriting in some categories of partial morphisms. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph-Grammars and Their Application to Computer Science. LNCS, vol. 532, pp. 490–504. Springer, Heidelberg (1990)
7. Lack, S., Sobocinski, P.: Adhesive and quasiadhesive categories. ITA **39**(3), 511–545 (2005)
8. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci. **109**(1&2), 181–224 (1993)
9. Löwe, M.: Graph rewriting in span-categories. Technical report 2010/02, FHDW-Hannover (2010)
10. Löwe, M.: A unifying framework for algebraic graph transformation. Technical report 2012/03, FHDW-Hannover (2012)
11. Löwe, M.: Polymorphic sesqui-pushout graph rewriting. Technical report 2014/02, FHDW-Hannover (2014)

Graph Transformation

9th International Conference, ICGT 2016, in Memory of

Hartmut Ehrig, Held as Part of STAF 2016, Vienna,

Austria, July 5-6, 2016, Proceedings

Echahed, R.; Minas, M. (Eds.)

2016, XVIII, 253 p. 109 illus., Softcover

ISBN: 978-3-319-40529-2