Double-Pushout Rewriting in Context

Michael Löwe

FHDW Hannover, Freundallee 15, 30173

Abstract Double-pushout rewriting (DPO) is the most popular algebraic approach to graph transformation. Most of its theory has been developed for linear rules, which allow deletion, preservation, and addition of vertices and edges only. Deletion takes place in a careful and circumspect way: a double pushout derivation does never delete vertices or edges which are not in the image of the applied match. Due to these restrictions, every DPO-rewrite is invertible. In this paper, we extend the DPO-approach to non-linear and still invertible rules. Some model transformation examples show that the extension is worthwhile from the practical point of view. And there is a good chance for the extension of the existing theory. In this paper, we investigate parallel independence.

1 Introduction

Double-pushout rewriting (DPO) is the most popular algebraic approach to graph and model transformation [3,4]. It can be formulated on a purely categorical level.

Definition 1 (DPO-rewriting). *A* rule $\varrho = (l : K \rightarrowtail L, r : K \rightarrowtail R)$ *is a span of monomorphisms. A* match $m : L \to G$ *for rule* ϱ *in an object* G *is a morphism from* ϱ*'s left-hand side to* G*. Rule* ϱ *can be applied at match* m*, if there are two pushout diagrams as depicted in Figure 1, i. e.* (m, g) *and* (p, h) *are pushouts of* (l, n) *and* (r, n) *resp. The two pushouts constitute a* direct derivation.

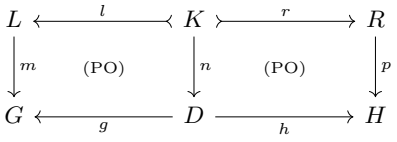

Figure 1. Double-pushout rewrite

By definition, every direct derivation is reversible: If $\varrho^{-1} = (r, l)$ denotes the inverse rule for a rule $\varrho = (r, l)$, we obtain for every direct derivation from G to H using rule ϱ, that there is a direct derivation using ϱ^{-1} from H to G.

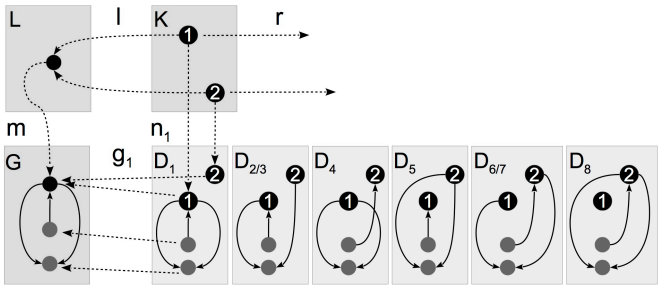

Figure 2. Indeterministic double-pushout rewrite

Furthermore, the pushout complement object D constructed as the intermediate object in a direct derivation is unique (up to isomorphism) in suitable categories.[1] This uniqueness property is lost, if the rule's left-hand sides are not restricted to monomorphisms, i. e. if we allow so-called *non-linear* left-hand sides. An example in the category of graphs is depicted in Figure 2. The morphism $l : K \to L$ is a rule's left-hand side which is not monic and splits a vertex into two particles. The depicted match $m : L \to G$ allows 8 different pushout complements and 3 pairwise non-isomorphic variants.[2] The concrete distribution of the adjacent edges of the split vertex is not specified by the rule and can be chosen arbitrarily by the direct derivation. Thus, the effect of the rule is underspecified.

For a deterministic effect of such rules, we have to specify how the context (adjacent objects) of split items shall be handled. We can do this, if we address the context explicitly in the rule itself and specify exactly where the context shall be attached to. Two of such specifications for the example of Figure 2 are depicted in Figure 3. In the left part of Figure 3, the context specification in K and L (thick grey arrows without source or target vertex) states that all incoming *and* all outgoing context edges shall be attached to the split-particle "1". Thus, the corresponding direct derivation picks D_1 (compare Figure 2) from the choice of possible pushout complements. In the right part of Figure 3, the context specification in K' and L (again thick grey arrows) states that all incoming context edges shall be attached to split-particle "2" whereas all outgoing context edges shall be attached to the split-particle "1". Thus, the corresponding direct derivation picks D_4 (compare Figure 2) from the choice of possible pushout complements.

The context handling we introduced on the intuitive level in Figure 3 cannot single out each derivation (all the possible pushout complements) of Figure 2, since all incoming context edges as well as all outgoing context edges of a split vertex are handled in an uniform way. For the sample situation of Figure 2, we

[1] Adhesive categories, details see below.

[2] The pushout complements D_2 and D_3 as well D_6 and D_7 produce isomorphic objects but differ in the assignments of edges to G, i. e. $g_2 \neq g_3$ and $g_6 \neq g_7$. The complement pairs D_1 and D_8, D_4 and D_5, as well as $D_{2/3}$ and $D_{6/7}$ are isomorphic and can only be distinguished if we fix the embedding of K.

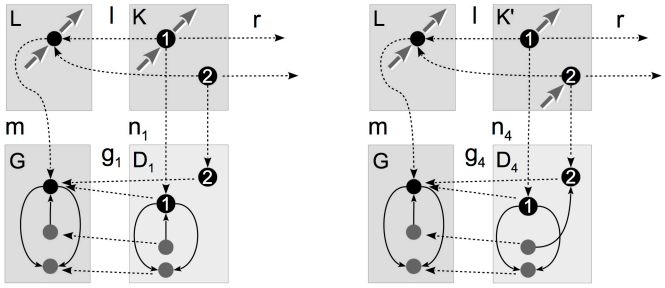

Figure 3. Handling of context

can only distinguish rewrites picking the complements D_1, D_4, D_5, and D_8. As we will see in the sample Section 4, this is not a major drawback wrt. applicability.

The context specification cannot copy context items or distribute them indeterministically. If we specified in the example that all outgoing edges shall be attached to particle "1" and all incoming edges shall be attached to both particles "1" *and* "2", there are two possible interpretations: (1) incoming edges shall be 'copied' to both particles or (2) incoming edges can be attached arbitrarily. In the first case, there is no suitable pushout complement (compare possibilities D_1 – D_8 in Figure 2); in the second case, the rewrite is underspecified again, since D_1 and D_4 are non-isomorphic pushout complements satisfying the specification.

But not only non-linear left-hand sides of rewrite rules cause problems in the double-pushout approach. We obtain some indeterminism as well, if we admit non-linear right-hand sides, i. e. if we do not restrict the rules' right-hand sides to monomorphisms. These problems are not concerned with the rewrite itself, since pushouts are uniquely determined (up to isomorphism) for arbitrary pairs of morphisms. The inverse rule (r, l) for a rule (l, r) with non-monic r, however, has a non-linear left-hand side and produces the sort of indeterminism which we observed above for example in Figure 2.

For a rewrite example with a non-linear right-hand side see Figure 4: the vertices "1" and "2" are mapped to the same vertex by the rule's right-hand side $r : K \to R$. A direct derivation with that rule merges the two matched vertices and connects *all* edges (incoming and outgoing) of these vertices ("1" and "2" in D_1) to the merged result vertex in H, compare morphism $h_1 : D_1 \to H$ in Figure 4.

If we apply the inverse rule (r, l) at the induced co-match $p : R \to H$, we again obtain several different pushout complements as in Figure 2. Among these pushout complements is the "original" one, namely D_1, that was used in the derivation that lead to the co-match p. But if we forgot the derivation structure and remembered the resulting co-match only, we are not able to choose the correct inverse derivation (among the 8 possible choices). This means that the information about the merging that took place in the derivation step is stored in the direct derivation only. Knowing the rule and the induced co-match is not sufficient to construct the compensating inverse derivation.

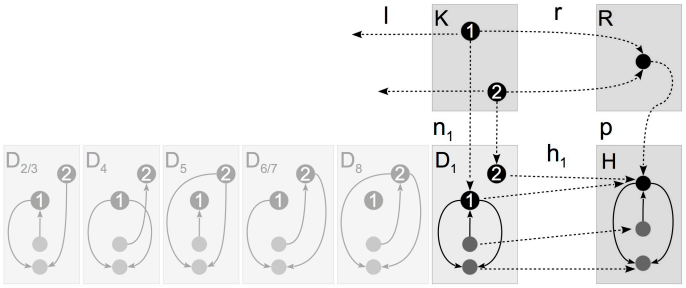

Figure 4. Indeterministic inverse rewrite

Again, an explicit specification of context handling can help making rules and their inverse rules deterministic. The information about the merging that took place in Figure 4 can be stored in the rule itself, if we use the context specification of K in the left part of Figure 3. With this 'context decoration', the rule and the induced co-match carry enough information to uniquely determine D_1 as the intermediate object for the compensating inverse derivation.

In this paper, we formalise the sort of context specification which we informally introduced above in Figure 3. For this purpose, we borrow and specialise constructions and mechanisms from AGREE-rewriting [1] and from rewriting in span categories [12] in Section 2. In Section 3, we show that the new rewrite construction is a conservative extension of double-pushout rewriting with left- and right-linear rules. Section 4 demonstrates the applicability of the introduced rewrite mechanism in the field of model transformation. Section 5 provides first theoretical results wrt. parallel independence which demonstrates that theoretical results for the DPO-approach are very likely to carry over to the extended rewrite mechanism. Finally, the conclusion provides a preview of future research.

2 DPO-Rewriting in Context

The theory for double-pushout rewriting has been formulated in adhesive categories [3,11]. We adopt this basic requirement for the constructions and results presented below for double-pushout rewriting in context, which we call DPO-C.

Definition 2 (Adhesive category). *A category is* adhesive *if*

1. *it has all pullbacks and*
2. *it has pushouts along monomorphisms which are all van-Kampen squares.*

A pushout $(f' : B \to D, g' : C \to D)$ *of a span* $(g : A \to B, f : A \to C)$ *is a van-Kampen square, if, for every commutative diagram as depicted in the left part of Figure 5 in which sub-diagrams* (2) *and* (3) *are pullbacks, the following*

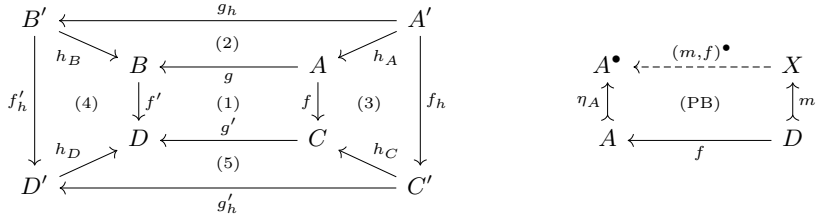

Figure 5. Adhesivity, hereditariness, and partial arrow classifier

compatibility of pushouts and pullbacks is satisfied: the pair (f'_h, g'_h) is pushout of the span (g_h, f_h), if and only if sub-diagrams (4) *and* (5) *are pullbacks.*

As we said in the introduction, DPO-C borrows major ingredients from AGREE-rewriting. A central issue is the existence of partial arrow classifiers.

Definition 3 (Partial arrow classifiers). *A category has* partial arrow classifiers*, if there is monic $\eta_A : A \rightarrowtail A^\bullet$ for every object A satisfying: For every pair $(m : D \rightarrowtail X, f : D \to A)$ of morphisms with monic m, there is a unique morphism $(m, f)^\bullet : X \to A^\bullet$ such that (m, f) is the pullback of $(\eta_A, (m, f)^\bullet)$, compare right part of Figure 5. In the following, the unique morphism $(m, f)^\bullet$ is also called* totalisation *of (m, f). We abbreviate $(m, \mathrm{id}_A)^\bullet$ by $m^\bullet$ and obtain for this special case where $m : A \rightarrowtail X$ is monic and $f = \mathrm{id}_A : A \to A$: $m^\bullet \circ m = \eta_A$.*

Fact 4 (Classifier). *There are well-known facts for partial arrow classifiers:*

1. *If $f : D \rightarrowtail A$ is monic, $(\eta_D, f)^\bullet$ is monic.*
2. *If $c : C \rightarrowtail B$, $b : B \to A$, and $a : C \rightarrowtail A$ are morphisms with monic c and a, then (c, id_C) is pullback of (a, b) and $b \circ c = a$, if and only if $c^\bullet = a^\bullet \circ b$.*
3. *All pushouts are hereditary:*[3] *Pushout (f', g') of (g, f) in sub-diagram (1) of Figure 5 is* hereditary*, if all commutative situations as in the left part of Figure 5 where sub-diagrams* (2) *and* (3) *are pullbacks and h_B and h_C are monic satisfy: (f'_h, g'_h) is pushout of (g_h, f_h), if and only if sub-diagrams* (4) *and* (5) *are pullbacks and h_D is monic.*

Almost all categories which are used in graph transformation are adhesive and possess partial arrow classifiers. Examples are graphs, i. e. algebras and homomorphisms wrt. the signature `G` depicted in Figure 6, and the simplified object-oriented class models, i. e. algebras and homomorphisms wrt. the signature `M` depicted in Figure 6, which we use for the sample transformations in Section 4.

Figure 7 depicts three sample partial arrow classifiers in `G`: (1) for a single vertex, (2) for a discrete graph with two vertices, and (3) for a graph with two vertices, a loop, and an edge between the vertices. The graph A that is classified is painted black, the grey parts are added by the classifier $A^\bullet$, and the classifying

[3] Compare [10].

```
G(raph) = sorts V(ertex) [painted as: •], E(dge) [painted as: →]
          opns  s(ource), t(arget): E ⟶ V

M(odel) = sorts T(ype) [painted as: □],
                I(nheritance) [painted as: ⊸▷],
                A(ssociation) [painted as: - - →]
          opns  c(hild), p(arent): I ⟶ T
                o(wner), t(arget): A ⟶ T
```

Figure 6. Graphs and simple object-oriented models

monomorphism is the inclusion. The classifier provides the additional structure that is needed to uniquely map the objects that are not in the image of m in arbitrarily given pair $(m : D \rightarrowtail X, f : D \to A)$. Note that the additional structure that the classifier adds to a classified graph does not differ, if we change the number of edges only, compare (2) and (3) in Figure 7.

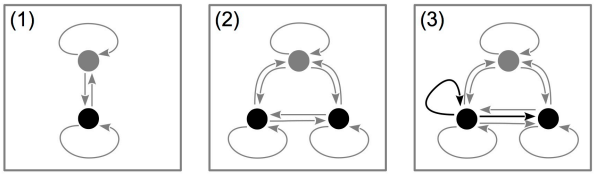

Figure 7. Sample partial arrow classifiers in the category G

Figure 8 shows two sample classifiers in M: (S) for a model with a single type C and (P) for a model with a pair of types C and C'.[4] Again, the classified models are painted black, the structure added by the classifier is painted grey, and the classifier is the inclusion homomorphism. (Arrows with two heads abbreviate two arrows, namely one in each direction.)

According to Definition 3, the classifying M-homomorphism $\eta_S : S \rightarrowtail S^\bullet$ in Figure 8 can be interpreted as follows: For any M-algebra A and an assignment f for a subset $\mathcal{T}$ of the types in A to the classified type C in S, i. e. $f : \mathcal{T} \to S$ and $\subseteq_{\mathcal{T}}: \mathcal{T} \hookrightarrow A$, there is a unique way to extend this assignment to a homomorphism $f^\bullet : A \to S^\bullet$, namely by mapping all types outside $\mathcal{T}$ to the 'grey' type and all inheritance relations and association to the uniquely available suitable 'grey' relations in S. Therefore, S provides the *sufficient* and *necessary* structure to map the *context* of $\mathcal{T}$ to S. And this extension of f to $f^\bullet$ [or more precisely to $(\subseteq_{\mathcal{T}}, f)^\bullet$] has the property that the pair $(\subseteq_{\mathcal{T}}, f)$ is pullback of $(\eta_S, f^\bullet)$. This additional property/requirement of partial arrow classifiers is essential. It prevents

[4] Inheritance relations and associations in the classified model do not change the structure that is added by the classifier. The classifier structure depends on the (number of) types only.

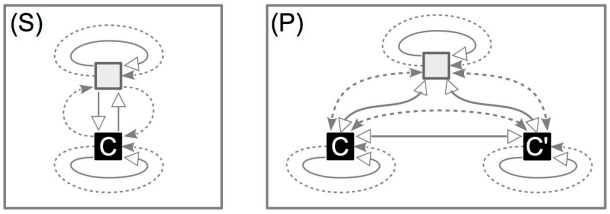

Figure 8. Sample partial arrow classifiers in the category M

that, for the special case that $\mathcal{T} = \emptyset$ and f is the empty mapping, some type in A is mapped to the type C in S by $(\emptyset, \emptyset)^\bullet$.

The partial arrow classifier for given M-algebra A is constructed as follows: (1) Add a Type-element $\bot$. (2) For every pair (t, t') of T-elements, add a I-element ${}^{\mathtt{I}}\bot^{t}_{t'}$ with $t' = \mathtt{c}({}^{\mathtt{I}}\bot^{t}_{t'})$ and $t = \mathtt{p}({}^{\mathtt{I}}\bot^{t}_{t'})$ and A-element ${}^{\mathtt{A}}\bot^{t}_{t'}$ with $t' = \mathtt{o}({}^{\mathtt{A}}\bot^{t}_{t'})$ and $t = \mathtt{t}({}^{\mathtt{A}}\bot^{t}_{t'})$. The added type $\bot$ is called the *undefined* type and the I- and A-loop added on this type are called *completely undefined* I- and A-edge respectively.

Assumption 5 (Basic category). *For the rest of the paper, we assume an adhesive category with partial arrow classifiers.*

For this sort of categories, we know the following facts [11]:

Fact 6 (Properties of the underlying category).

1. *Pushouts along monomorphisms are pullbacks.*
2. *Pushouts preserve monomorphisms.*
3. *Pushout of intersection is union: If* $(x : X \rightarrowtail Z, y : Y \rightarrowtail Z)$ *is a cospan of monomorphisms,* $(x' : I \rightarrowtail Y, y' : I \rightarrowtail X)$ *its pullback span, and* $(x^* : X \rightarrowtail U, y^* : Y \rightarrowtail U)$ *the pushout of* (x', y')*, then the unique morphism* $u : U \rightarrowtail Z$ *with* $u \circ x^* = x$ *and* $u \circ y^* = y$ *is* monic*, compare Figure 9.*

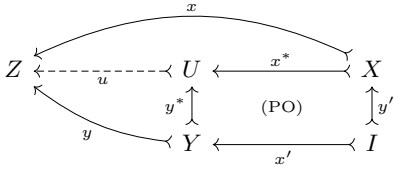

Figure 9. Union of intersection

Partial arrow classifier constructions have been successfully applied in AGREE-rewriting [1] in order to control the deletion and copy process of context items in a rewrite which can be stipulated by non-linear left-hand sides of rules. For a

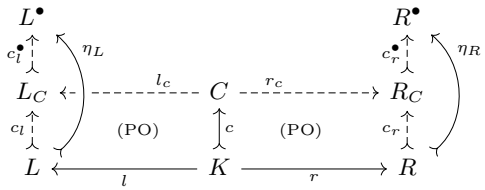

Figure 10. DPO-C rule

double-pushout semantics, we need to restrict the AGREE-rewriting mechanism: context items must not be copied nor deleted, they can only be distributed to split particles, compare motivating examples in the introduction.

Definition 7 (DPO-C-rule). *A* rule $(l : K \to L, c : K \rightarrowtail C, r : K \to R)$ *is a triple of morphisms such that the context specification* c *is monic and, given the pushouts* $(c_l : L \rightarrowtail L_C, l_c : C \to L_C)$ *and* $(c_r : R \rightarrowtail R_C, r_c : C \to R_C)$ *of* (l, c) *and* (r, c) *respectively, the morphisms* $c_l^\bullet$ *and* $c_r^\bullet$ *are monic, compare Figure 10.*[5]

DPO-C rules are special AGREE-rules. The special rule format makes sure that items in L_C which are not in L, have a 'unique preimage' under l_c. In the category `G` of graphs for example, we cannot choose $C = K^\bullet$ and $c = \eta_K$, if there are vertices $v_1 \neq v_2$ with $l(v_1) = l(v_2)$. In this case, the pushout of l and η_K results in a graph with at least 4 context loops on $l(v_1)$ and this graph is not a sub-graph of $L^\bullet$, which has a one loop only, compare (1) and (2) in Fig. 7. The symmetric restriction of the right side will ensure reversibility of rewrites.

Definition 8 (DPO-C-match and -derivation). *Given rule* $\sigma = (l : K \to L, c : K \rightarrowtail C, r : K \to R)$*, a monomorphism* $m : L \rightarrowtail G$ *is a* match*, if the following* match condition *is satisfied: The morphism* $m^\bullet : G \to L^\bullet$ *factors through* L_C*, i. e. there is* $m' : G \to L_C$ *such that* $c_l^\bullet \circ m' = m^\bullet$*.*

A derivation *with rule* σ *at match* m *is constructed as follows, compare Figure 11:*

1. *Construct pullback* $(g : D \to G, n' : D \to C)$ *of* $(m' : G \to L_C, l_c : C \to L_C)$*.*
2. *Let* $n : K \rightarrowtail D$ *be the unique mediating morphism for this pullback for* $(m \circ l, c)$*. By pullback decomposition*[6] *and Fact 6(1) for pushout* (l_c, c_l)*,* (l, n) *is pullback of* (g, m)*. Since pullbacks preserve monomorphisms,* n *is monic.*
3. *Construct pushout* $(h : D \to H, p : R \rightarrowtail H)$ *of* $(n : K \rightarrowtail D, r : K \to R)$*. The morphism* p *is monic by Fact 6(2).*

Remarks. Note that we restrict matches to monomorphisms.[7] The morphism m' which satisfies the matching condition is unique, if it exists, since $c_l^\bullet$ is monic. The morphism n can be constructed in Step 2 of Definition 8, since $c_l^\bullet \circ m' \circ m = m^\bullet \circ m = \eta_L = c_l^\bullet \circ c_l$ implies $m' \circ m = c_l$ due to $c_l^\bullet$ being monic.

[5] The pushout morphisms c_l and c_r are monic by Fact6 (2).

[6] Compare Appendix A.

[7] Therefore, the *identification condition* for rule applicability [9] does not matter here.

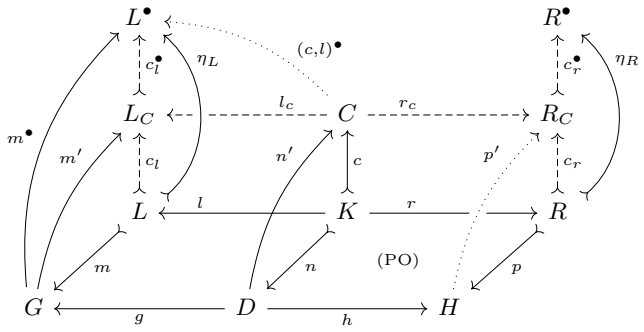

Figure 11. DPO-C match and derivation

The match condition in Definition 8 formulates a negative application condition as in [8]. Especially the *dangling condition* of double pushout rewriting in the category `G` of graphs [9] is reformulated this way: if the rule's left-hand side $l : K \to L$ is not epic on vertices, there is vertex v without pre-image under l. Since L_C is pushout, this means that $c_l(v)$ can only have adjacent edges that have pre-images under c_l. If there is an edge adjacent to $m(v)$ without pre-image under m, this edge 'is' not in L and not in L_C, it is *dangling*, and it cannot be mapped by m' to any edge in L_C in order to satisfy the match condition.

The rewrite mechanism in Def. 8 is a special case of AGREE-rewriting: If (g, h) is DPO-C trace of DPO-C rule (l, c, r) at match m, then it is also AGREE trace of AGREE rule (l, c, r) at m.[8] The rule restriction of Def. 7 and the match condition of Def. 8, however, tame the 'AGREE-tiger' such that (1) items outside the match cannot be deleted nor copied and (2) irreversible merging is avoided.

3 Analysis of DPO-C derivations

In this section, we analyse the properties of DPO-C-derivations. Especially, we investigate reversibility and show that the DPO-C-approach is a conservative extension of the DPO-approach with left- and right-linear rules at monic matches.

Proposition 9 (Determinism). *DPO-C-rewrites are deterministic.*[9]

This result justifies the following notation:

Notation 10 (Deterministic rewrite). *In a derivation with rule σ at match m as in Def. 8, the* result *H is denoted by $\sigma@m$, the span (g, h) is called the* trace*, written $\sigma\langle m\rangle$, and morphism p constitutes the* co-match*, written $m\langle\sigma\rangle$.*

[8] Since (id_L, c_l) and (c, l) are pullbacks of $(\eta_L, c_l^\bullet)$ resp. (l_c, c_l) by Fact 6 (1) and, therefore, (l, c) is pullback of $(c_l^\bullet \circ l_c, \eta_L)$, we have that $c_l^\bullet \circ l_c = (c, l)^\bullet$. Since $c_l^\bullet$ is monic, (id_G, m') is pullback of $(c_l^\bullet, m^\bullet)$ and (n', g) is pullback of $(m^\bullet, c_l^\bullet \circ l_c)$.

[9] For a proof, see Appendix B.1.

Proposition 11 (Rewrite properties). *Consider a derivation with rule $\sigma = (l, c, r)$ at match $m : L \rightarrowtail G$ as depicted in Figure 11. The participating subdiagrams have the following properties:*[10]

1. *(m, id_L) and (n, id_K) are pullback of (m', c_l) and (n', c) respectively.*
2. *(m, g) and (m', l_c) are pushouts of (l, n) and (g, n') respectively.*
3. *If $p' : H \to R_C$ is the unique morphism for pushout (p, h) providing $p' \circ p = c_r$ and $p' \circ h = r_c \circ n'$, then*
   (a) (h, n') and (p, id_R) are pullbacks of (r_c, p') and (p', c_r) respectively and
   (b) (r_c, p') is pushout of (n', h).

Thus, every square in Figure 11 is pushout *and* pullback. Therefore, DPO-C-rewriting could also be called *triple* double-pushout transformation.

Corollary 12 (Reversibility). *Every DPO-C-rewrite is* reversible*: if (g, h) is trace and p co-match of the application of rule (l, c, r) at match m, then (h, g) is the trace and m the co-match of applying the inverse rule (r, c, l) at match p.*[11]

We close this section by showing that standard DPO-rewriting with left- and right-linear rules is a special case of DPO-C-derivations.

Definition 13 (DPO-simulation). *The DPO-C-simulation of a left- and right-linear DPO-rule $\varrho = (l : K \rightarrowtail L, r : K \rightarrowtail R)$ is the triple $\sigma_\varrho = (l, \eta_K, r)$.*

Proposition 14 (DPO-simulation). *DPO-C-simulations are DPO-C-rules.*

Proof. We have to show the conditions of Definition 7. For this purpose, consider Figure 12, where u_l and u_r are the unique morphisms providing (i) $u_l \circ c_l = \eta_L$, (ii) $u_l \circ l_c = (\eta_k, l)^\bullet$, (iii) $u_r \circ c_r = \eta_R$, and (iv) $u_r \circ r_c = (\eta_k, r)^\bullet$. By Fact 4(1), $(\eta_k, l)^\bullet$ and $(\eta_k, r)^\bullet$ are monic and, by Fact 6(3), u_l and u_r are monic. Equations (i) and (iii) and u_l and u_r being monic implies that (c_l, id_L) and (c_r, id_R) are pullbacks of (u_l, η_L) and (u_r, η_R) respectively. Thus, $u_l = c_l^\bullet$ and $u_r = c_r^\bullet$.

Theorem 15 (DPO-extension). *If $\varrho \langle m \rangle$ and $m \langle \varrho \rangle$ are trace and co-match of a DPO-derivation with linear rule ϱ, then $\sigma_\varrho \langle m \rangle = \varrho \langle m \rangle$ and $m \langle \sigma_\varrho \rangle = m \langle \varrho \rangle$.*

Proof. Consider Figure 12 where the two bottom pushouts constitute a DPO-derivation with left- and right-linear rule (l, r). Then there is $n^\bullet : D \to K^\bullet$ such that (n, id_K) is pullback of $(n^\bullet, \eta_K)$ and especially $n^\bullet \circ n = \eta_K$. Since (m, g) is pushout, we obtain morphism $m' : G \to L_C$ making the diagram commutative. Given pullbacks (n, id_K) of $(n^\bullet, \eta_K)$ and (l, id_K) of (id_L, l) and the given pushouts (m, g) and (l_c, c_l) together with the van-Kampen property of Definition 2, guarantee that $(g, n^\bullet)$ is pullback of (m', l_c) and (m, id_L) is pullback of (m', c_l). This last pullback property and u_l being monic such that (m', id_G) is pullback of $(u_l, u_l \circ m')$ implies that $u_l \circ m' = m^\bullet$ by pullback composition and uniqueness of totalisations. Since $(n^\bullet, g)$ is pullback of (m', l_c) and (h, p) is pushout of (r, n), $\sigma_\varrho \langle m \rangle = \varrho \langle m \rangle$ and $m \langle \sigma_\varrho \rangle = m \langle \varrho \rangle$.

[10] For proofs, see Appendix B.2.

[11] For the proof, see Appendix B.3.

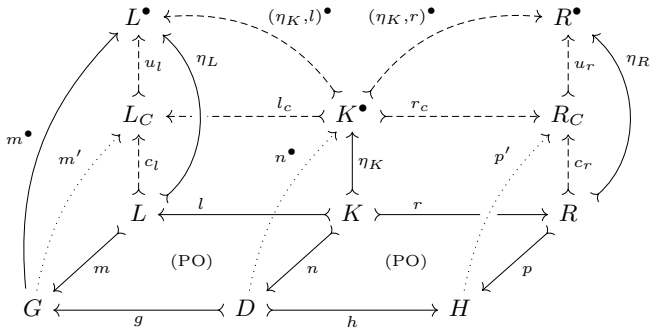

Figure 12. DPO-rule simulation

4 Model Refactorisation - Some Sample Rules

In this section, we demonstrate the applicability of double-pushout rewriting in context by some sample rules for object-oriented system refactoring [7]. To keep the examples simple, we use the simplified meta-model `M` for object-oriented models defined in Figure 6.

Figure 13 depicts two first sample rules. The mapping of the morphisms is indicated by number correspondence. For easy notation, we identify the undefined type of each partial arrow classifier (grey boxes in Figure 8) with the framing box which surrounds the respective graphical visualisation of the algebra. We also implicitly assume that the two (completely undefined) loops on the undefined type are contained in and preserved by the context specification of all rules. An edge which connects a type inside a picture with the frame is some context inheritance relation or association, i. e. belongs to L_C, C, or R_C.

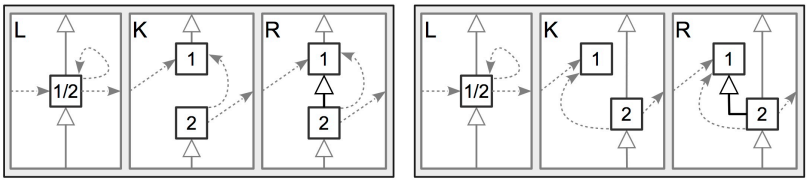

Figure 13. Extracting abstract type

Figure 13 depicts variants for extracting an abstract type out of a given type. The first variant on the left refines the inheritance hierarchy by splitting the matched type "1/2" into an abstract particle "1" and a concrete particle "2" in the intermediate structure K. The context relations of the split type are distributed as follows: All `target`-roles of associations and `child`-roles of inheritance relations are attached to the abstract particle, all other roles are connected with

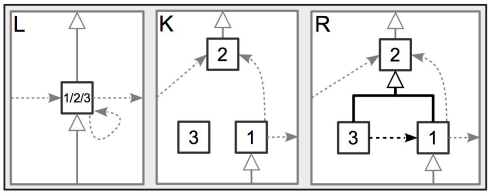

Figure 14. Introducing proxy

the concrete particle. Note the explicit handling of association loops which also follows this rule wrt. `owner`- and `target`-roles. Finally, the rule's right-hand side R adds the needed inheritance relation between the two particles. The difference of the second variant is that the new abstract type is not integrated into the existent inheritance hierarchy.

Note that both rules formulate a negative application condition, namely that there are no `Inheritance`-loops on the refactored type. This shall be true in all reasonable object-oriented models where the inheritance relation is hierarchical.

The rules in Figure 13 read from right to left specify the elimination of superfluous types. For these elimination rules to work correctly, the type "1" shall be abstract. This is a feature that must be added to the model signature in Figure 6. We do not describe the details here due to space limitations.

Figure 14 depicts a rule that puts indirection into an object-oriented model by introducing some proxy-objects [6]. Again the context is distributed as in Figure 13, the refactored type, however, is split into *three* particles, namely "1" which further manages the resources of the type, "2" which provides abstract access to 'objects' of type "1", and "3" which can be interpreted as an approximation of the original type. The rule's right-hand side adds the needed inheritance relations and the association which allows 'proxy' objects to delegate to 'real' objects.

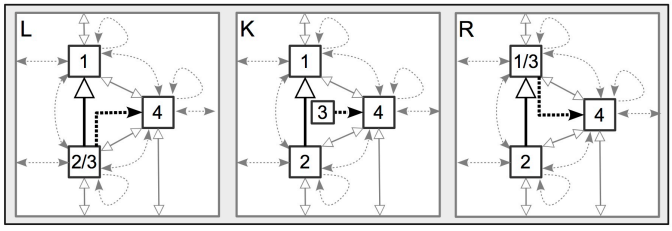

Figure 15. Pulling-up association

Figure 15 describes the shift of the `owner`-role of an association to a more abstract type.[12] This rule is neither left- nor right-linear. The standard DPO-solution for

[12] Double-headed context arrows represent a pair of arrows one in each direction.

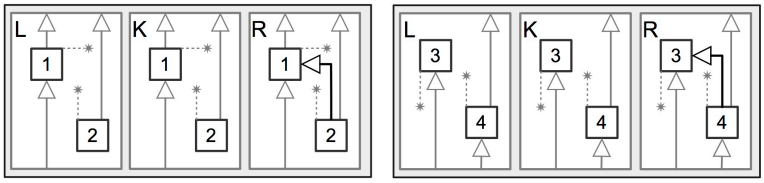

Figure 16. Preserving inheritance hierarchy

this purpose is a linear rule that deletes the association on the left-hand side and adds a *new* association on the right-hand side. This rule has the same effect on the model level. But if there were `instanceOf`-relations from the object level to the model level that point to this association,[13] the rule in Figure 15 preserves all these links while the linear rule deletes them and introduces a new 'empty' association without any links. For details compare also [13].

Figure 16 demonstrates a useful application of the application conditions which are built-in in DPO-C-rewriting. These two rules add inheritance relations carefully, i. e. they keep the inheritance hierarchy cycle-free: A type "2" can only become new sub-type of type "1" if it has no sub-types itself and a type "3" can only become new super-type of type "4" if it has no super-types itself. Since the association context is not important here and the rules are linear, the asterisk-notation we used in Figure 16 indicates *complete association contexts.*[14]

These examples demonstrate the DPO-C-rewriting can be useful in practical applications and that it is worthwhile to elaborate more complex case studies.

5 Parallel Independence

Parallel independence analysis investigates the conditions under which two rewrites of the same object can be performed in either order and produce the same result. Essential for the theory is the notion of *residual match*: Under which conditions are two matches $m_G : L \rightarrowtail G$ and $m_H : L \rightarrowtail H$ for a rule's left-hand side L *the same match*, if there is a trace $(g : D \to G, h : D \to H)$? The DPO-answer is: m_G and m_H are the same, if there is $m_D : L \rightarrowtail D$ with $g \circ m_D = m_G$ and $h \circ m_D = m_H$, compare [9]. This answer is not sufficient for DPO-C, since we need to take the context matches into account as well, i. e. $m_G^\bullet$, $m_D^\bullet$, and $m_H^\bullet$ shall classify the 'same objects' the same way. This means that we must require $m_G^\bullet \circ g = m_D^\bullet$ and $m_H^\bullet \circ h = m_D^\bullet$ which, by Fact 4 (2), is equivalent to requiring that (id_L, m_D) is pullback of (m_G, g) and (m_H, h).[15]

[13] This feature needs to be added in the model signature in Figure 6.

[14] *Complete association contexts* means e. g. for type "1" in Figure 16 that there 'are' 2 adjacent association pairs from and to type "2" and from and to the undefined type.

[15] This condition is identical to the one in [2].

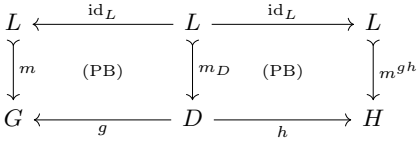

Figure 17. Residual

Definition 16 (Residual). *Let $(g : D \to G, h : D \to H)$ be a trace of a direct derivation and m match for rule σ in G. A match m^{gh} for σ in H is* the residual *of m* for trace (g, h)*, if there is morphism m_D from the left-hand side L of σ to D such that (id_L, m_D) is pullback of (m, g) and (m^{gh}, h), compare Figure 17.*

The pullback properties uniquely determine *the* residual, if it exists. Two derivations of the same object are independent, if they have mutual residuals.

Definition 17 (Parallel independence). *Two direct derivations with rules σ_1 and σ_2 at matches m_1 and m_2 resp. rewriting the same object are parallel independent, if m_1 has a residual for $\sigma_2 \langle m_2 \rangle$ and m_2 has a residual for $\sigma_1 \langle m_1 \rangle$.*

Theorem 18 (Confluence). *If derivations with rules σ_1 and σ_2 at matches m_1 and m_2 are parallel independent, then the derivations with the mutual residuals $m_1^{\sigma_2\langle m_2\rangle}$ and $m_2^{\sigma_1\langle m_1\rangle}$ produce the same result, i. e. $\sigma_1 @ m_1^{\sigma_2\langle m_2\rangle} \approx \sigma_2 @ m_2^{\sigma_1\langle m_1\rangle}$.*[16]

6 Conclusion

We introduced a conservative extension of linear DPO-rewriting which we call DPO-C. The "C" indicates that the extension allows explicit handling of the context of a match. The context specification allows non-linear rules with deterministic and reversible rewrites: Given a match $m : L \to G$ for rule σ with left-hand side L and right-hand side R, the rewrite with σ at m produces a uniquely determined result H and provides a co-match $p : R \to H$ such that the rewrite with the inverse rule σ^{-1} at p results in an object isomorphic to G.

The deterministic and reversible behaviour of DPO-C allows to extend well-known theoretical results. We started the analysis of parallel independence in this paper. And the explicit handling of context improves the applicability of the rewrite approach in situations where 'unknown context' must be checked (by some negative application conditions), distributed, or merged. We demonstrated this mechanism by some examples from system refactoring. Thus, a further development of DPO-C seems worthwhile from the practical *and* theoretical point of view. Future research can address the following issues:

– Characterising conditions for parallel independence.

[16] For the proof, see Appendix B.4.

– Extension of the theory for example with respect to sequential independence, concurrency, critical pair analysis, parallelism, and amalgamation.
– Comparison of the DPO-C-built-in negative application conditions to the well-known negative application conditions from the literature, e. g. [8].
– Comparison of DPO-C to other reversible approaches e. g. [2].
– Development of a clear and handy visual notation for the rules especially for the context specification.
– Elaboration of bigger case studies e. g. in the field of model transformation.

References

1. Andrea Corradini, Dominique Duval, Rachid Echahed, Frédéric Prost, and Leila Ribeiro. AGREE - algebraic graph rewriting with controlled embedding. In Francesco Parisi-Presicce and Bernhard Westfechtel, editors, *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, volume 9151 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2015.
2. Vincent Danos, Tobias Heindel, Ricardo Honorato-Zimmer, and Sandro Stucki. Reversible sesqui-pushout rewriting. In *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*, pages 161–176, 2014.
3. Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
4. Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation - General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2015.
5. Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors. *Graph Transformations - 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 - - October 2, 2010. Proceedings*, volume 6372 of *Lecture Notes in Computer Science*. Springer, 2010.
6. Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
7. Martin Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999.
8. Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Inform.*, 26(3/4):287–313, 1996.
9. Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
10. Tobias Heindel. Hereditary pushouts reconsidered. In Ehrig et al. [5], pages 250–265.
11. Stephen Lack and Pawel Sobocinski. Adhesive and quasiadhesive categories. *ITA*, 39(3):511–545, 2005.
12. Michael Löwe. Graph rewriting in span-categories. In Ehrig et al. [5], pages 218–233.
13. Michael Löwe. Refactoring information systems: association folding and unfolding. *ACM SIGSOFT Software Engineering Notes*, 36(4):1–7, 2011.

A Pushout and Pullback Composition and Decomposition

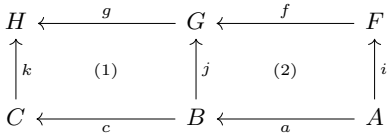

Figure 18. Pushout and pullback composition and decomposition

Proposition 19 (Pullback composition and decomposition). *Let a commutative diagram as in Figure 18 be given such that sub-diagram* (1) *is pullback. Then* $(i, c \circ a)$ *is pullback of* $(g \circ f, k)$*, if and only if sub-diagram* (2) *is pullback.*

Proposition 20 (Pushout composition and decomposition). *Let a commutative diagram as in Figure 18 be given such that sub-diagram* (2) *is pushout. Then* $(g \circ f, k)$ *is pushout of* $(i, c \circ a)$*, if and only if sub-diagram* (1) *is pushout.*

B Proofs

B.1 Proof for Proposition 9

The pullback and pushout constructed in Step 1 resp. 3 of Definition 8 are unique up to isomorphism. So given two trace and co-match pairs $((g_1, h_1), p_1)$ and $((g_2, h_2), p_2)$ for two derivations with rule σ at the same match m, there are isomorphisms i_g and i_h such that $i_g \circ g_1 = g_2$, $i_h \circ h_1 = h_2 \circ i_g$, and $i_h \circ p_1 = p_2$.

B.2 Proof for Proposition 11

(1) We know that (m, id_L) is pullback of $(m^\bullet, \eta_L)$ and, since $c_l^\bullet$ is monic, that (m', id_G) is pullback of $(m^\bullet, c_l^\bullet)$. Since $c_l^\bullet \circ c_l = \eta_L$ and $m' \circ m = c_l$, pullback decomposition[17] provides (m, id_L) as pullback of (m', c_l). Now (m, id_L) is pullback of (m', c_l) and we always have that (id_K, l) is pullback of (id_L, l). Thus, pullback composition provides $(\mathrm{id}_K, m \circ l)$ as pullback of $(m', c_l \circ l)$. Since $c_l \circ l = l_c \circ c$ and $m \circ l = g \circ n$, $(\mathrm{id}_K, g \circ n)$ is pullback of $(m', l_c \circ c)$. Since (g, n') has been constructed as pullback of (m', l_c), pullback decomposition guarantees that (n, id_K) is pullback of (n', c).

(2) Adhesivity (compare Definition 2 van-Kampen property if-part) guarantees that (g, m) is pushout of (l, n), since the pushout (l_c, c_l) of (l, c) is a pushout along monomorphism c and surrounded by 4 pullbacks, namely (i) (id_K, n) of

[17] See Appendix A.

(n', c), (ii) (id_K, l) of (id_L, l), (iii) (id_L, m) of (m', c_l), and (iv) (n', g) of (m', l_c). Now, pushout decomposition provides (m', l_c) as pushout of (g, n').

(3a) Adhesivity (compare Definition 2 van-Kampen property only-if-part) guarantees the desired pullback properties, since (r_c, c_r) is pushout of (r, c) along monic c, (p, h) is pushout of (r, n) by the construction of the derivation, (id_K, n) is pullback of (n', c) by (1) above, and (id_K, r) is trivially pullback of (id_R, r).

(3b) Pushout decomposition provides that (r_c, p') is pushout of (n', h).

B.3 Proof for Corollary 12

Consider the derivation in Figure 11. We are done, if the pair (p, p') is match for the inverse rule, since, by Proposition 11, (n', h) is pullback of (r_c, p') and (g, m) is pushout of (l, n). Thus, it remains to show that $c_r^\bullet \circ p' = p^\bullet$, i. e. that (id_R, p) is pullback of $(\eta_R, c_r^\bullet \circ p')$. We know by Proposition 11 (3a) that (p, id_R) is pullback of (p', c_r) and, since $c_r^\bullet$ is monic, that (p', id_H) is pullback of $(c_r^\bullet \circ p', c_r^\bullet)$. Since $c_r^\bullet \circ c_r = \eta_R$, pullback composition provides the desired result.

B.4 Proof for Theorem 18

The complete situation with two derivations using rule $\sigma_1 = (l_1, c_1, r_1)$ at match m_1 and rule $\sigma_2 = (l_2, c_2, r_2)$ at match m_2 is depicted in Figure 19. The two morphisms to the intermediate objects D_1 and D_2 provided by Definition 16 for residuals are m_{21} and m_{12} resp. Thus, the residual matches are $h_1 \circ m_{21}$ and $h_2 \circ m_{12}$. Since they are matches, their totalisations $(h_1 \circ m_{21})^\bullet$ and $(h_2 \circ m_{12})^\bullet$ factor through L_C^2 and L_C^1 resp. The morphism $m_2^* : H_1 \to L_C^2$ denotes one of these (factor) morphisms, i. e. $\left(c_l^2\right)^\bullet \circ m_2^* = (h_1 \circ m_{21})^\bullet$. By Fact 4 (2) $m_{21}^\bullet = m_2^\bullet \circ g_1$. Since $m_2^\bullet$ factors through L_C^2 by m_2', the same must be true for $m_{21}^\bullet$ such that there is m_{21}' with $m_{21}' = m_2' \circ g_1$. Again by Fact 4 (2) $m_2^* \circ h_1 = m_{21}'$.

Construct pullback (g_1', g_2') of (g_1, g_2) which provides morphisms n_{12} and n_{21} which make the diagram commutative. Pullback composition and decomposition guarantees that (l_1, n_{12}) and (l_2, n_{21}) are pullbacks of (g_1', m_{12}) resp. (g_2', m_{21}) as well as that $(\mathrm{id}_{K_1}, n_{12})$ and $(\mathrm{id}_{K_2}, n_{21})$ are pullbacks of (g_2', n_1) resp. (g_1', n_2). The two pullbacks (n_2', g_2) and (g_1', g_2') compose to the pullback (n_{21}', g_2') of (l_c^2, m_{21}').

Now construct the left-hand side (g_{12}, n_2^*) of the derivation at residual $h_1 \circ m_{21}$ as pullback of (m_2^*, l_c^2). Then we obtain h_1' such that (h_1', g_2') is pullback of (h_1, g_{12}). Using the van-Kampen property of pushouts along monomorphisms, we obtain pushouts (g_2', m_{21}) of (n_{21}, l_2) and $(g_{12}, h_1 \circ m_{21})$ of $(h_1' \circ n_{21}, l_2)$ as in the proof of Proposition 11. By pushout decomposition, (g_{12}, h_1) is pushout of (g_2', h_1').

Construct pullback (p_{12}, x) of (p_1, g_{12}).[18] This provides monic p_{12} and morphism y (not in Fig. 19) with $x \circ y = r_1$ and (y, n_{12}) as pullback of (h_1', p_{12}). Fact 4 (3) implies that (x, r_1) is pushout of (id_{K_1}, y) such that $x = \mathrm{id}_{R_1}$ and $y = r_1$. Now, pushout (p_1, h_1) of (r_1, n_1) is surrounded by 4 pullbacks, namely $(p_{12}, \mathrm{id}_{R_1})$,

[18] The morphism x is not depicted in Figure 19. It will turn out to be the identity.

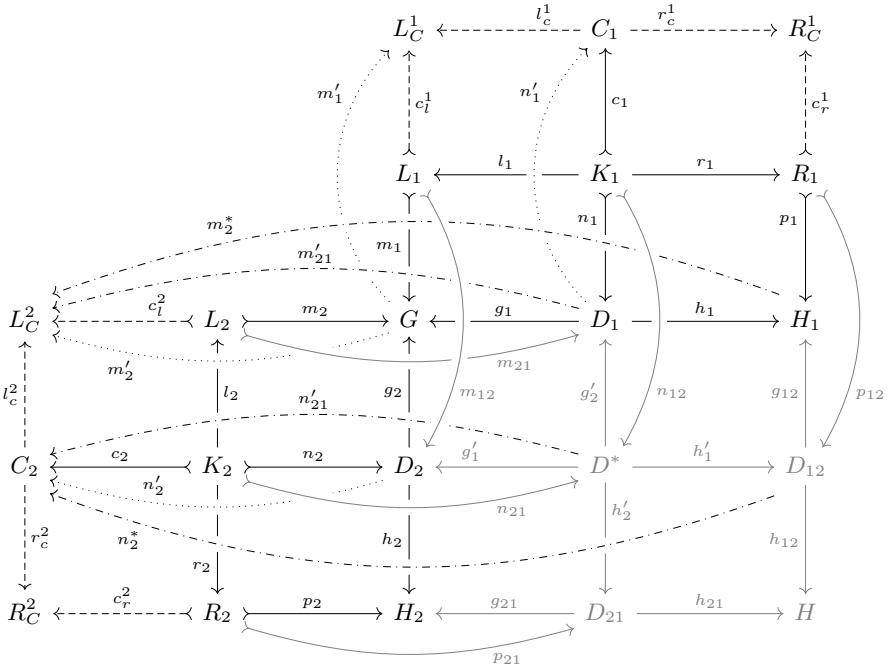

Figure 19. Parallel independence

(r_1, id_{K_1}), $(n_{12}, \mathrm{id}_{K_1})$, and (h_1', g_2'), such that (h_1', p_{12}) is pushout of (r_1, n_{12}) by adhesivity. By symmetry (h_2', p_{21}) is pushout of (r_2, n_{21}).

The right-hand side pushout of the derivation with σ_1 at the residual match $m_1^{\sigma_2\langle m_2\rangle}$, i. e. the pushout of r_1 and $h_2' \circ n_{12}$, can now be decomposed providing pushout (h_{12}, h_{21}) of (h_1', h_2'). Pushout composition shows that h_{12} is the right part of the trace for the derivation with σ_2 at residual $m_2^{\sigma_1\langle m_1\rangle}$. Thus, (g_{12}, h_{12}) and (g_{21}, h_{21}) are the traces of the rewrites at the residuals.