Short-Cut Rules

Sequential Composition of Rules Avoiding Unnecessary Deletions Extended Version

Lars Fritsche[1], Jens Kosiol[2], Andy Schürr[1], and Gabriele Taentzer[2]

[1] TU Darmstadt
{lars.fritsche,andy.schuerr}@es.tu-darmstadt.de
[2] Philipps-Universität Marburg
{kosiolje,taentzer}@mathematik.uni-marburg.de

Abstract. Sequences of rule applications in high-level replacement systems are difficult to adapt. Often, replacing a rule application at the beginning of a sequence, i.e., reverting a rule and applying another one instead, is prevented by structure created via rule applications later on in the sequence. A trivial solution would be to roll back all applications and reapply them in a proper way. This, however, has the disadvantage of being computationally expensive and, furthermore, may cause the loss of information in the process. Moreover, using existing constructions to compose the reversal of a rule with the application of another one, in particular the concurrent and amalgamated rule constructions, does not prevent the loss of information in case that the first rule deletes elements being recreated by the second one. To cope with both problems, we introduce a new kind of rule composition through 'short-cut rules'. We present our new kind of rule composition for monotonic rules in adhesive HLR systems, as they provide a well-established generalization of graph-based transformation systems, and motivate it on the example of Triple Graph Grammars, a declarative and rule-based bidirectional transformation approach.

Keywords: Rule Composition · Amalgamated Rule · E-Concurrent Rule · Triple Graph Grammars

1 Introduction

High-level replacement (HLR) systems [2,3] are a useful generalization for transforming various kinds of high-level structures, such as graphs, in a rule-based manner. Transformation processes consist of sequences of rule applications. These sequences effectively de-/construct and modify structures, yet, they also implicitly create dependency relationships: an earlier rule application may be the precondition for a later one. Often, these relationships prevent rule applications at the beginning of a sequence to be replaced by another one, as reverting the former would destruct preconditions used for transformations later in the sequence. A trivial solution would be to roll back all applications that depend on each other, until reaching the one that is to be replaced, and reapply them in a proper way. However, rolling back and recreating these sequences has the disadvantage of being computationally expensive and, furthermore, may cause the loss of information in the process. Thus, it would be highly beneficial to replace rule applications in a – preferably also rule-based – way that preserves the remaining sequence. Existing approaches to rule composition, namely the parallel, concurrent, and amalgamated rule constructions [1,2,3], are not apt to deal with that kind of dependency.

Hence, we introduce a novel kind of rule composition through *short-cut rules* whose applications serve as an alternative to possibly long chains of replacement actions. A short-cut rule composes the reversal of a monotonic rule, i.e., of a rule which only creates structure, with the application of a second one. Yet, doing this, the short-cut rule identifies elements, deleted by reverting the first rule, with elements, created by the second one, hereby preserving them. This preservation allows for applications of short-cut rules even in situations where the reversal of the first rule itself is impossible. We accomplish this by pair-wisely comparing the rules of a given HLR system searching for common substructures. Consequently, we exploit this information for creating short-cut rules that preserve those common substructures. While the approach is formalized for monotonic rules in HLR systems in general, we use Triple Graph Grammars (TGGs) [9] as example for demonstration purposes. TGGs are an established formalism for the declarative description of complex consistency relationships between two modelling languages with graph-like representations. They are especially useful for efficiently checking and restoring the consistency of a given pair of models [8] or for generating possible combinations of consistent pairs of models; unfortunately, they do not offer adequate means for the specification of arbitrarily complex editing operations that directly transform one consistent pair of models into another consistent pair of models. With our contribution we are able to solve a common problem of TGGs by using our novel rule composition scheme to take a set of TGG rules as input and produce a set of short-cut rules as output. The rule composition scheme guarantees that any combination of inverse and normal applications of TGG rules can be replaced by short-cut rules and may even be executed in several situations where the inverse application is impossible. They have the additional advantage of preserving some graph elements which otherwise would be deleted by the corresponding inverse application of a TGG rule and be recreated by the corresponding normal application of a TGG rule.

The main contributions of this paper are as follows: We illustrate the use of short-cut rules in the context of TGGs (Sect. 2). We formalize the construction of short-cut rules and prove the Short-Cut Theorem (Theorem 7), settling the synthesizability of applications of monotonic rules into an application of a short-cut rule and the analysability of applications of a short-cut rule into applications of monotonic rules (Sect. 4). We formally compare our new kind of rule composition with existing ones (Sect. 5). Furthermore, in Sect. 3 we recall transformation rules and HLR systems. Section 6 concludes the paper and points to some future work. Most of the proofs are presented in Appendices A to C.

2 Introductory Example

The construction and use of short-cut rules is motivated at the example of consistency between a simplified class diagram and a custom documentation structure. It is an excerpt of, and based on the example provided by Leblebici et al. [7], yet, in a simplistic form to show the basic idea of our approach. Thus, it contains no (propagation of) attributes, which will be covered in future work. Our example is an excerpt from a consistency specification between a class diagram and a documentation structure using Triple Graph Grammars (TGGs). It thus consists a *Package* structure containing *Classes* on the one side and a *Folder* structure containing *Doc-Files* on the other.

TGGs [9] are a declarative, rule-based bidirectional transformation approach proposed by Schürr. Given two input meta-models, a TGG specification defines consistency between instances of both. To this end, it consists of a finite set of graph grammar rules that define how consistent pairs of both models co-evolve. In order to relate elements from both sides, TGGs introduce a third meta-model, which is referred to as the correspondence meta-model. It is used to connect elements of both sides such that they become correlated and thus traceable.

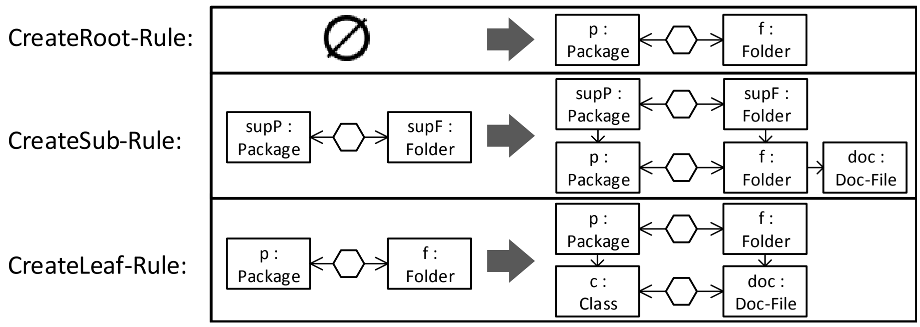


Fig. 1. A TGG to co-evolve class diagram and documentation structure

Figure 1 shows the rule set for our example consisting of three TGG rules. The first rule depicts the base TGG rule of the given rule set. Since its left-hand side (LHS) L is empty, and thus no precondition exists, it can always and arbitrarily often create a root *Package* together with a root *Folder* and a correspondence link between both. Given the context from the LHS, the second rule creates a *Package* and *Folder* hierarchy where every sub-folder has a *Doc-File* that may contain the documentation of the corresponding *Package*. Finally, the third rule creates a *Class* together with a corresponding *Doc-File* analogously to the *Package* and *Folder* of the previous rule.

Given these rules, one can create consistent graph triples, such as those shown in Fig. 2. The exemplary triple on the left consists of a hierarchy of three *Packages* on the left side which are correlated to a similar hierarchy of *Folders* via

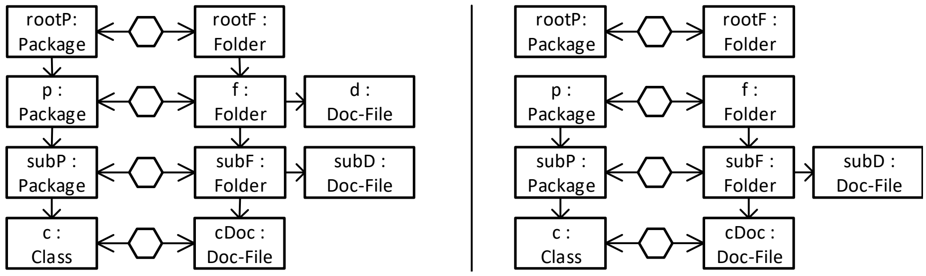


Fig. 2. Two examples for consistent triples

correspondence links. However, the *Folders* `f` and `subF` additionally contain their own *Doc-File*. Thus, the triple was created via four consecutive applications of TGG rules by applying first *CreateRoot-Rule*, followed by *CreateSub-Rule* twice and finally *CreateLeaf-Rule*.

An important point about this transformation sequence is that it creates entities for both the class diagram and the documentation structure simultaneously, but the resulting model does not contain any information about the contents of the created elements. This means that, in practical applications, the user may add data manually which is not correlated to the other side, like layout information for the class diagram or textual descriptions as the contents of *Doc-Files*. Due to this lack of correlation, one has to be careful on how to change models in order to avoid unnecessary data loss. Given the model on the left side of Fig. 2, a reasonable example for such a change would be the separation of the first two hierarchy levels making the former sub-elements `p` and `f` to be root elements by effectively deleting the connection to their former root elements (and the superfluous *Doc-File*) as is depicted on the right side of Fig. 2. However, no rule of the current grammar is able to perform such a change and to modify the triple by hand is a tedious and error-prone task that can create triples which do not longer comply with the TGG language. To solve this issue and to create a triple graph which contains *Package* `p` and *Folder* `f` as additional roots (and is unmodified otherwise) we have to proceed as follows: We have to roll back all rule applications except the first one (*CreateRoot-Rule*) and recreate the deleted parts of the graph triple from scratch again – despite the fact that the intended modification affects only a small portion of the graph triple. Executing this strategy with large hierarchies has two major disadvantages. First, it is tedious and might be computationally expensive for complex models. Second, one may loose a large amount of manually added data.

However, when studying the TGG rules of Fig. 1 in detail, we see that *CreateRoot-Rule* and *CreateSub-Rule* have common substructures, i.e., we can find nodes and edges of the same type arranged in the same way in left- and right-hand sides of both rules. In our example, such a common substructure of their right-hand sides (RHS) R stems from the fact that both rules create a *Package* and a *Folder* together with a correspondence link between those two

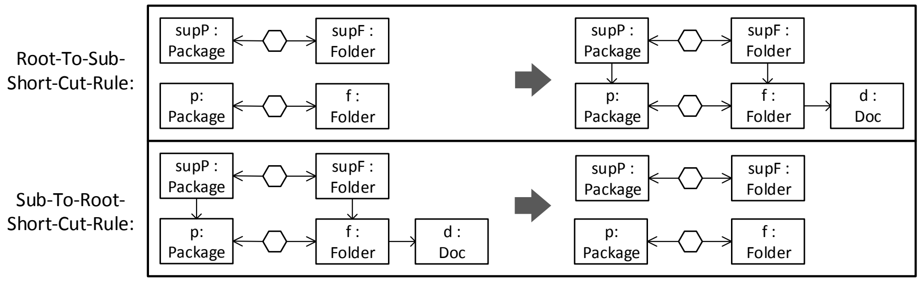


Fig. 3. Two examples for short-cut rules (interface K of rules given implicitly as $L \cap R$)

elements. It consists of the *Folders* `f` and *Packages* `p` but does not include the *Doc-File* only contained by *CreateSub-Rule*.

Taking a closer look at our example in Fig. 2, one can see how this insight propagates to the model level and that the only difference between a root-*Folder* and a sub-*Folder* is that the latter one possesses an additional *Doc-File* and has an incoming hierarchy edge. Hence, one might want to exploit this knowledge by replacing a TGG rule application somewhere in a sequence of rule applications by another similar rule application such that formerly created elements are possibly preserved and the need to roll back sub-sequences does not arise. In the current case this would mean to preserve all elements that are contained in the `root` elements by changing the *CreateSub-Rule*-application to become a *CreateRoot-Rule*-application. Therefore, we have to use the common parts of both rules to create a new rule which directly transforms the left to the right graph triple depicted in Fig. 2, which again is an element of the language of the TGG of Fig. 1. Thus, the result of the application of such a 'short-cut rule' looks like the composition of the effects of the reverse application of *CreateSub-Rule* followed by the application of *CreateRoot-Rule*. Implicitly, the application of the short-cut rule operates as a kind of meta-rule on sequences of TGG rule applications as it replaces an occurrence of a rule with the occurrence of another rule in an arbitrarily long sequence of rule applications. Figure 3 depicts two short-cut rules that enable to replace *CreateRoot-Rule* with *CreateSub-Rule* and vice versa. In our example, *Sub-To-Root-Short-Cut-Rule* replaces an occurrence of *CreateSub-Rule* with an occurrence of *CreateRoot-Rule* as shown in Fig. 4. Note, however, that short-cut rules extend the set of rules rather than replace it.

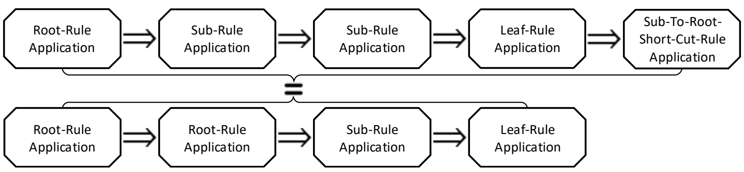


Fig. 4. Example: Application of Short-Cut Rule

It, thus, preserves the consistency of the graph triple of Fig. 2 by selecting the elements `p` and `f` as new root elements and by deleting the now superfluous `d` element associated with `f` as well as the edges connecting `rootP` and `rootF` to `p` and `f`, respectively. This singular application of one short-cut rule stands in contrast to the deletion and recreation of the affected triple graph from scratch.

3 Preliminaries

Since adhesive categories [5] provide a suitable formal framework generalizing many instances of rule-based rewriting of graph-like structures (including triple graphs), we present our work in that setting. This section shortly recalls the definition of rule-based transformation systems. Adhesive categories and their properties are recalled in Appendix A before the presentation of most of the proofs in Appendices B and C.

Rules are a declarative way to define transformations of objects. They consist of a left-hand side (LHS) L, a right-hand side (RHS) R, and a common subobject K, the interface of the rule. In case of (typed) triple graphs, application of a rule p to a graph G amounts to choosing an image of the rule's LHS L in G, deleting the image of $L \setminus K$ and adding a copy of $R \setminus K$. This procedure can be formalized, also in the more general setting of adhesive categories, by two pushouts. Rules and their application semantics are defined as follows.

Definition 1 (Rules and adhesive HLR systems). *Given an adhesive category* $\boldsymbol{C}$*, a* rule *(or* production*)* p *consists of three objects* L, K*, and* R*, called* left-hand side*,* interface *(or* gluing object*), and* right-hand side*, and two monomorphisms* $l : K \hookrightarrow L, r : K \hookrightarrow R$*. Given a rule* $p = (L \stackrel{l}{\hookleftarrow} K \stackrel{r}{\hookrightarrow} R)$*, the* inverse rule p^{-1} *is defined as* $p^{-1} = (R \stackrel{r}{\hookleftarrow} K \stackrel{l}{\hookrightarrow} L)$*. A rule* $p = (L \stackrel{l}{\hookleftarrow} K \stackrel{r}{\hookrightarrow} R)$ *is called* monotonic *(or* non-deleting*) if* $l : K \hookrightarrow L$ *is an isomorphism. In that case we just write* $r : L \hookrightarrow R$*.*

A subrule p' *of a rule* $p = (L \stackrel{l}{\hookleftarrow} K \stackrel{r}{\hookrightarrow} R)$ *is a rule* $p' = (L' \stackrel{l'}{\hookleftarrow} K' \stackrel{r'}{\hookrightarrow} R')$ *with monomorphisms* $u : L' \hookrightarrow L$*,* $w : K' \hookrightarrow K$*,* $v : R' \hookrightarrow R$ *such that both squares in the diagram to the right are pullbacks and a pushout complement for* $u \circ l'$ *exists.*

$$\begin{array}{ccccc} L' & \stackrel{l'}{\hookleftarrow} & K' & \stackrel{r'}{\hookrightarrow} & R' \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle w} & & \downarrow{\scriptstyle v} \\ L & \stackrel{l}{\hookleftarrow} & K & \stackrel{r}{\hookrightarrow} & R \end{array}$$

A common kernel rule p *for rules* p_1 *and* p_2 *is a common subrule of both.*

An adhesive high-level replacement system *(or* HLR system *for short) consists of an adhesive category* $\boldsymbol{C}$ *and a set of rules* P *in that category.*

Figures 1 and 3 depict rules in the category of triple graphs. The first are monotonic, the second set includes a general rule. Together they form an HLR system.

For the construction of short-cut rules, we are mainly interested in common kernel rules of monotonic rules, which we will denote by $k : L_\cap \hookrightarrow R_\cap$. They are necessarily monotonic themselves. Note that, in adhesive categories with strict initial object, i.e., with initial object $\emptyset$ where each morphism into $\emptyset$ is an isomorphism, the *trivial common kernel rule* $id_\emptyset : \emptyset \hookrightarrow \emptyset$ is a common kernel rule for any two monotonic rules r_1 and r_2. Such strict initial objects exist, e.g., in the categories of sets, graphs, and triple graphs.

The next definition determines the semantics of the application of a rule.

Definition 2 (Transformation). *In an adhesive category* $\boldsymbol{C}$*, given a rule* $p = (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$*, an object* G*, and a monomorphism* $m : L \hookrightarrow G$*, called* match*, a* (direct) transformation $G \Rightarrow_{p,m} H$ *from* G *to* H *via* p *at match* m *is given by the diagram to the right where both squares are pushouts.*

$$\begin{array}{ccccc} L & \overset{l}{\hookleftarrow} & K & \overset{r}{\hookrightarrow} & R \\ {\scriptstyle m}\downarrow & & \downarrow & & \downarrow{\scriptstyle n} \\ G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

A rule p *is called* applicable *at match* m *if the first pushout square above exists, i.e., if* $m \circ l$ *has a pushout complement. When applying a rule* p *to an object* G*, the arising object* D *is called the* context object *of the transformation.*

4 Construction Process

In this section, we formalize the construction of short-cut rules. As explained in Sect. 2, a short-cut rule is a composition of a monotonic rule r_2 with the inverse rule r_1^{-1} of a monotonic rule r_1. The composition is done in such a way that the short-cut rule may preserve certain elements which an inverse application of r_1 would delete and an application of r_2 would recreate. The extent to which preservation of elements takes place is flexible, depending on a chosen common kernel rule of the two rules. In the following, we first present the construction of a short-cut rule given a common kernel rule. Afterwards, we prove the correctness of the construction and discuss its merits.

We use common kernel rules to construct short-cut rules. Given a common kernel rule k of monotonic rules r_1 and r_2, their short-cut rule $r_1^{-1} \ltimes_k r_2$ arises by gluing r_1^{-1} and r_2 along k. The LHS of k contains the information how to glue r_1^{-1} and r_2 to receive the LHS L and the RHS R of the short-cut rule $r_1^{-1} \ltimes_k r_2$. I.e., $r_1^{-1} \ltimes_k r_2$ is constructed in such a way, that a match for it consists of matches for r_1^{-1} and r_2 which intersect in the LHS of k. The RHS of k contains the information how to construct the interface K of the short-cut rule $r_1^{-1} \ltimes_k r_2$. In case of (triple) graphs, elements of $R_\cap \setminus L_\cap$ are included in K, i.e., $R_\cap \setminus L_\cap$ specifies exactly those elements that would have been deleted by r_1^{-1} and recreated by r_2. Hence, they are to be preserved when applying the short-cut rule.

Definition 3 (Short-cut rule). *In an adhesive category* $\boldsymbol{C}$*, given two monotonic rules* $r_i : L_i \hookrightarrow R_i$*,* $i = 1, 2$*, and a common kernel rule* $k : L_\cap \hookrightarrow R_\cap$ *for them, the* short-cut rule $r_1^{-1} \ltimes_k r_2 := (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$ *is computed by executing the following steps:*

1. *The union* $L_\cup$ *of* L_1 *and* L_2 *along* $L_\cap$ *is computed as pushout* (2) *in Fig. 5.*

2. *The LHS L of the short-cut rule $r_1^{-1} \ltimes_k r_2$ is constructed as pushout* $(3a)$ *in* Fig. 5.
3. *The RHS R of the short-cut rule $r_1^{-1} \ltimes_k r_2$ is constructed as pushout* $(3b)$ *in* Fig. 5.
4. *The interface K of the short-cut rule $r_1^{-1} \ltimes_k r_2$ is constructed as pushout* (4) *in* Fig. 6.
5. *Morphisms $l : K \to L$ and $r : K \to R$ are obtained by the universal property of K.*

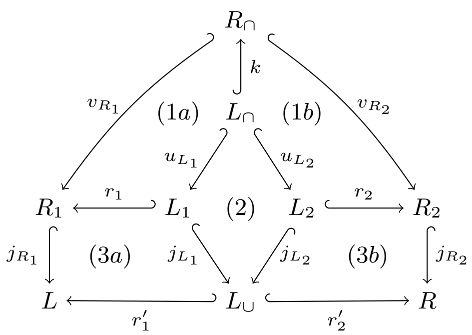


Fig. 5. Construction of LHS and RHS of short-cut rule $r_1^{-1} \ltimes_k r_2$

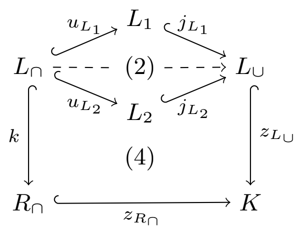


Fig. 6. Construction of interface K of $r_1^{-1} \ltimes_k r_2$

Example 4. We illustrate the construction of short-cut rules with a detailed example. First, *CreateRoot-Rule* is a (non-trivial) common kernel rule for *CreateSub-Rule* and itself, as depicted in Fig. 7. Here, and in the following figures, morphisms are indicated by the names of the nodes; the mapping of edges follows unambiguously. Hence, *CreateRoot-Rule* is embedded into itself via the identity morphism and its RHS is mapped to nodes `p` of type *Package* and `f` of type *Folder* in the RHS of *CreateSub-Rule*; the morphism between the LHSs is the unique empty map.

Next, computation of $L_\cup$ and the LHS and RHS of the short-cut rule is done by computing the three pushouts as depicted in Fig. 8. It is a concrete instantiation of the lower part of the diagram depicted in Fig. 5. The two pushouts to the left and in the middle are pushouts along the empty triple graph, i.e., the respective objects are just copied next to each other. The pushout to the right is a pushout along an isomorphism, hence the resulting morphism to the very right is an isomorphism as well.

Lastly, the interface of the short-cut rule is calculated as pushout as depicted in Fig. 9. It is a concrete instantiation of the diagram depicted in Fig. 6. As pushout along the empty triple graph, again, the resulting triple graph consists of copies of the two triples at the lower left and the upper right. The monomorphisms

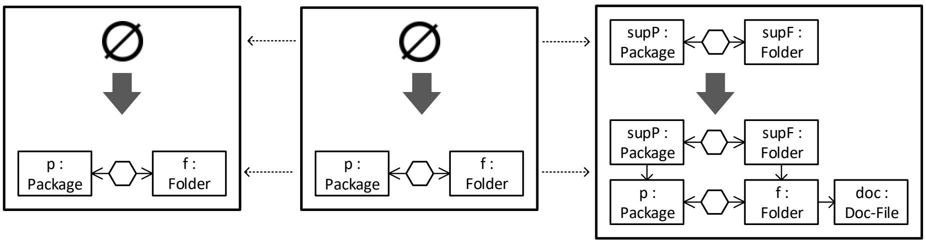


Fig. 7. *CreateRoot-Rule* as common kernel rule for *CreateSub-Rule* and itself

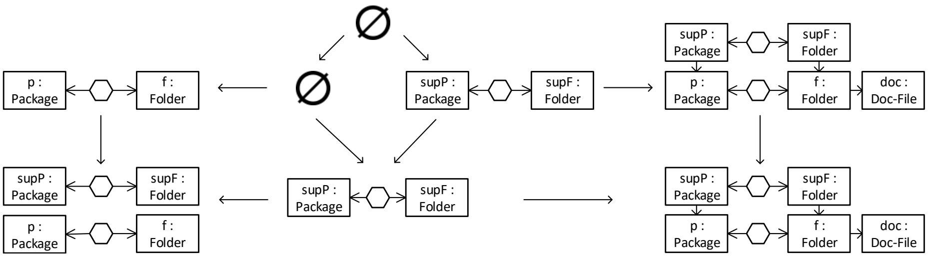


Fig. 8. Construction of LHS and RHS of a short-cut rule for *CreateRoot-Rule* and *CreateSub-Rule*

from the interface into the LHS and RHS computed above, are, again, indicated by the names of the nodes. Thus, the resulting short-cut rule is *Root-To-Sub-Short-Cut-Rule* as displayed in Fig. 3 or in the upper part of Fig. 12.

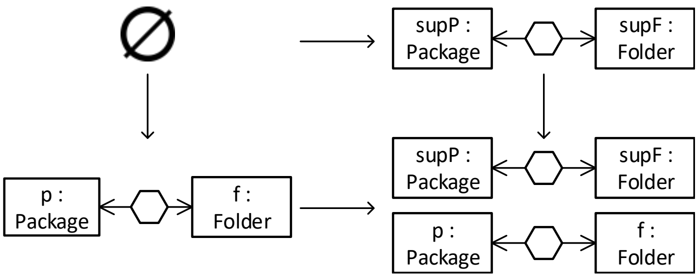


Fig. 9. Construction of the interface of a short-cut rule

The following lemma ensures that short-cut rules are rules in the sense of Definition 1, i.e., that the morphisms from the interface to the LHS and RHS are monomorphisms. (Such rules are also called *linear* rules.)

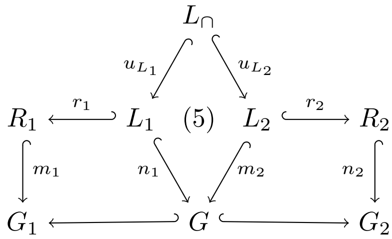
Lemma 5 (Linearity of short-cut rule). *In an adhesive category* $\mathbf{C}$*, given two monotonic rules* $r_i : L_i \hookrightarrow R_i$*,* $i = 1, 2$*, and a common kernel rule* $k : L_\cap \hookrightarrow R_\cap$

for them, the induced morphisms $l : K \to L$ and $r : K \to R$ in the short-cut rule $r_1^{-1} \ltimes_k r_2$ are monomorphisms.

The next definition relates common kernel rules for rules r_1, r_2 with sequences of applications of r_1^{-1} and r_2.

Definition 6 (Compatibility).

Given a sequence $G_1 \Rightarrow_{r_1^{-1},m_1} G \Rightarrow_{r_2,m_2} G_2$ of rule applications, where rules r_1 and r_2 are monotonic, and a common kernel rule $k : L_\cap \hookrightarrow R_\cap$ for these rules, then k is called compatible *with the application sequence if the resulting square (5) in the diagram to the right is a pullback.*



Compatibility as defined above ensures the existence of a unique morphism $h : L_\cup \hookrightarrow G$ such that $n_1 = h \circ j_{L_1}$ and $m_2 = h \circ j_{L_2}$ (compare pushout square (2) in Fig. 5). Moreover, in adhesive categories h is a monomorphism. Note that, given a sequence of rule applications, a compatible common kernel rule can always be obtained by computing $L_\cap$ and the corresponding embeddings into L_1, L_2 as pullback and setting $R_\cap = L_\cap$ (with the embedding being the identity).

The following Short-cut Theorem is our main result. Its synthesis part states that an inverse application of a monotonic rule followed by an application of a monotonic rule may indeed be replaced by an application of a short-cut rule. Its analysis part states that the application of a short-cut rule may be split into the reverse application of a monotonic rule followed by the application of a second one if the reverse application of the first rule is possible at all. Its proof makes use of a technical lemma, stating the equivalence of the existence of certain pushout complements, whose statement we postpone towards the end of this section. If analysis is possible then synthesis and analysis are inverse to each other.

Theorem 7 (Short-cut Theorem). *In an adhesive category $\mathbf{C}$, let $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, be two monotonic rules, $k : L_\cap \hookrightarrow R_\cap$ a common kernel rule for them, and $r_1^{-1} \ltimes_k r_2$ the corresponding short-cut rule. Then the following holds:*

1. ***Synthesis:*** *For each transformation sequence $G_1 \Rightarrow_{r_1^{-1},m_1} G \Rightarrow_{r_2,m_2} G_2$ compatible with k there exists a direct transformation $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$ with context object G' and a monomorphism $g : G \hookrightarrow G'$, s. t. $m_1' \circ j_{R_1} = m_1$.*
2. ***Conditional Analysis:*** *Given a direct transformation $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$ with context object G' such that a pushout complement for $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ exists, where $m_1 = m_1' \circ j_{R_1}$, then there exists a transformation sequence $G_1 \Rightarrow_{r_1^{-1},m_1} G \Rightarrow_{r_2,m_2} G_2$ compatible with k. Moreover, a monomorphism $g : G \hookrightarrow G'$ exists.*
3. ***Correspondence:*** *In those cases, where the pushout complement necessary for the analysis construction exists, the synthesis and analysis constructions are inverse to each other (up to isomorphism).*

Proof. 1. Let a transformation $G_1 \Rightarrow_{r_1^{-1},m_1} G \Rightarrow_{r_2,m_2} G_2$ be given. The outer square in Fig. 10 is the pushout given by the application of r_1^{-1} with match m_1 and (3a) is the pushout used to define L. Since the transformation sequence is compatible with k, a unique monomorphism $h : L_\cup \hookrightarrow G$ with $n_1 = h \circ j_{L_1}$ exists. Since (3a) is a pushout, $m_1' : L \hookrightarrow G_1$ exists. It is a monomorphism since $G \hookrightarrow G_1$ and $m_1 : R_1 \hookrightarrow G_1$ are monomorphisms (Fact 17, Item 4). By pushout decomposition, the resulting square (6) + (7a) is a pushout. Define (6) again by taking the pushout. Like above, the resulting map $G' \hookrightarrow G_1$ is a monomorphism and square (7a) is a pushout by pushout decomposition. Thus, rule $r_1^{-1} \ltimes_k r_2$ is applicable at G_1 with match m_1' and G' is the context object of the resulting transformation. Moreover, G embeds into G' by $g : G \hookrightarrow G'$. Comparing Fig. 11, an analogous argument shows that G_2 is the pushout of $r : K \hookrightarrow R$ and $n_1' : K \hookrightarrow G'$. Altogether, the resulting transformation, applying $r_1^{-1} \ltimes_k r_2$ at match m_1', consists of (7a) and (7b).

2. Let a direct transformation $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$ with context object G' be given. Defining $m_1 = m_1' \circ j_{R_1}$ gives a match for r_1^{-1} in G_1. By assumption, the rule r_1^{-1} is applicable at that match, i.e., a pushout complement for $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ exists (compare again Fig. 10). Lemma 9 states that the existence of such a pushout complement is equivalent to the existence of a pushout complement for $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ (with arising objects being isomorphic). Therefore, application of r_1^{-1} at match m_1 results in an object G with morphism $g : G \to G'$ to the context object of the transformation $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$. The morphism g is a monomorphism, since pushout (6) is a pushout along the monomorphism $z_{L_\cup}$.
Define $m_2 := h \circ j_{L_2} : L_2 \hookrightarrow G$ as match for r_2 in G (compare again Fig. 11). Then, since (3b), (6), and (7b) are pushouts, the outer square is also a pushout, and hence G_2 is the result of applying r_2 with match m_2 at G. Moreover, by definition of m_2, the resulting transformation sequence is compatible to k.

3. If the analysis construction is possible, the synthesis and analysis constructions are inverse to each other because pushout complements along monomorphisms and pushouts are unique (up to isomorphism) in adhesive categories. □

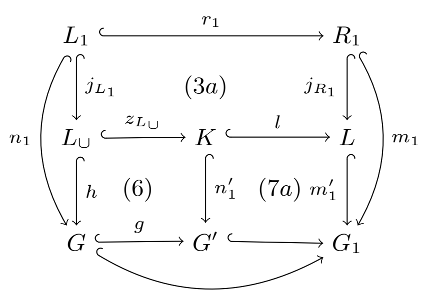


Fig. 10. Synthesis and Analysis: formation of context object G'

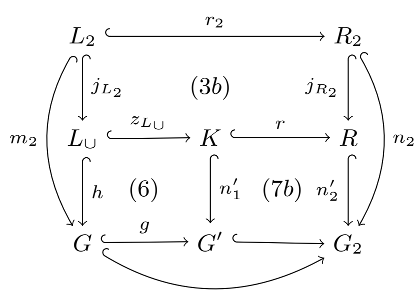


Fig. 11. Synthesis and Analysis: result of rule application

The following lemma states that, generally, the monomorphism $g : G \hookrightarrow G'$, arising in both the synthesis and the analysis construction above, is not an isomorphism. Thus, in case of (triple) graphs, applying a short-cut rule instead of the original rules actually preserves elements, namely the elements of $G' \setminus G$.

Lemma 8 (Preservation). *In an adhesive category* $\mathbf{C}$*, let* $r_i : L_i \hookrightarrow R_i$*,* $i = 1, 2$*, be two monotonic rules and* $k : L_\cap \hookrightarrow R_\cap$ *a common kernel rule for them. Let* $g : G \hookrightarrow G'$ *be a monomorphism arising by synthesis of a transformation sequence* $G_1 \Rightarrow_{r_1^{-1}, m_1} G \Rightarrow_{r_2, m_2} G_2$ *or by analysis of a transformation* $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$ *(compare Theorem 7, especially Figs. 10 and 11). Then* g *is an isomorphism iff* k *is.*

Before concluding this section with a discussion of the value of short-cut rules, we state the lemma used in the proof of Theorem 7. A proof for it is given in Appendix C where additionally the notion of initial pushouts [2], on which the proof depends, is recalled.

Lemma 9 (Characterization of PO-complements). *In any adhesive category with initial pushouts, given a commutative diagram like Fig. 10 where* (3a) *and* (7a) *are pushouts, a pushout complement object* G *for* $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ *is a pushout complement object for* $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ *and vice versa. Particularly, a pushout complement for* $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ *exists iff a pushout complement for* $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ *exists.*

Benefits and Limitations of Short-Cut Rules. We motivated the use of short-cut rules twofold. (1) That the application of short-cut rules generally preserves elements instead of deleting and recreating them, as stated in Lemma 8. (2) That the application of a short-cut rule may actually amount to a 'short-cut' which is due to the asymmetry of synthesis and analysis in the Short-Cut Theorem. Applications of the short-cut rules *Sub-To-Root-Short-Cut-Rule* and *Root-To-Sub-Short-Cut-Rule* (Fig. 3) with the obvious matches transform between the two consistent triples depicted in Fig. 2. But in either case, dangling edges prevent the analysis of the short-cut rule's application into a sequence of two rule applications. Thus, the subsequent applications of rules in the upper transformation chain in Fig. 4 would need to be revoked first, before a reverse application of the respective second rule application is possible in the first place.

However, not every application of a short-cut rule, that may not be analyzed, is a 'short-cut'. For example, applying the short-cut rule *Root-To-Sub-Short-Cut-Rule* to the left instance in Fig. 2, but with nodes `rootP` and `subP` of type *Package* and `rootF` and `subF` of type *Folder* as match instead, creates additional container edges for nodes `subP` and `subF` and a second *Doc-File* inside of node `subF`. This instance is not an element of the language defined by the original TGG (Fig. 1). This stems from the fact that the short-cut rule *Root-To-Sub-Short-Cut-Rule* revokes an application of the rule *CreateRoot*, while the elements chosen to be revoked by the match have actually been created using the rule *CreateSub*.

A first possible strategy to resolve that issue is the development of application conditions [4] for short-cut rules ensuring that a short-cut rule is only applicable at matches on which it revokes the proper rule. For example, the short-cut rule *Root-To-Sub-Short-Cut-Rule* could be equipped with an application condition forbidding the existence of incoming edges to nodes `p` and `f`, respectively. Another possible strategy is the use of marked TGGs and trace information [6] to the same end, i.e., to only allow those matches for a short-cut rule where the rule that was actually used to create the structure is revoked. We plan to further elaborate and compare between both strategies as future work. Our aim is to arrive at short-cut rules whose application does not divert from the language defined by the HLR system from which the short-cut rules were derived.

5 Related Work: Comparison to Other Formalisms of Rule Composition

In the literature, there exist several formalisms for composition of rules, most importantly *parallel*, *concurrent*, and *amalgamated rules* [1,2,3]. We relate our construction of short-cut rules to these other formalisms. A common difference to short-cut rules is that the parallel, concurrent, and amalgamated rule constructions are defined for general rules, whereas our construction of short-cut rules is restricted to the case of monotonic rules for now. Therefore, in this section, we first recall the relevant constructions generally and then relate these to our construction of short-cut rules in the special case of monotonic rules.

The parallel rule of two rules combines their respective actions into one rule. Two independent direct transformations arising by applications of these rules may alternatively be replaced by an application of their parallel rule [2].

Definition 10 (Parallel rule). *Given an adhesive category* $\boldsymbol{C}$ *with binary coproducts, the parallel rule* $p_1 + p_2$ *of two rules* $p_i = (L_i \xhookleftarrow{l_i} K_i \xhookrightarrow{r_i} R_i)$, $i = 1, 2$, *is defined by* $p_1 + p_2 = (L_1 + L_2 \xhookleftarrow{l_1 + l_2} K_1 + K_2 \xhookrightarrow{r_1 + r_2} R_1 + R_2)$, *where* $+$ *denotes the coproduct or the induced morphism, respectively.*

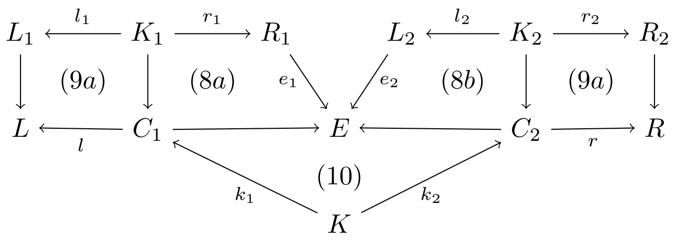
In categories with strict initial object (explained in Sect. 3) short-cut rules along the trivial common kernel rule are the same as parallel rules. This is, e.g., the case in the category of (triple) graphs, where the empty (triple) graph is the (only) strict initial object.

Proposition 11 (Relation to parallel rule). *Let two monotonic rules* $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, *in an adhesive category* $\boldsymbol{C}$ *with strict initial object* $\emptyset$ *be given. Then, for the trivial common kernel rule* $id_\emptyset : \emptyset \hookrightarrow \emptyset$, *the short-cut and the parallel rule coincide, i.e.,* $r_1^{-1} + r_2 = r_1^{-1} \ltimes_{id_\emptyset} r_2$.

Like the parallel rule, a so-called E-concurrent rule combines the action of two rule applications into the application of one rule. But here, the rule applications may be sequentially dependent [2]. An E-dependency relation encodes this possible dependency. The definition of E-dependency relations and E-concurrent rules assumes a given class $\mathcal{E}$ of pairs of morphisms with the same codomain.

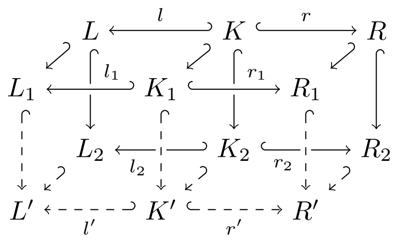
Definition 12 (E-dependency relation and E-concurrent rule). *Given two rules $p_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$, $i = 1, 2$, an object E with morphisms $e_1 : R_1 \to E$ and $e_2 : L_2 \to E$ is an E-*dependency relation *for p_1 and p_2 if $(e_1, e_2) \in \mathcal{E}$ and the pushout complements $(8a)$ and $(8a)$ over $K_1 \xrightarrow{r_1} R_1 \xrightarrow{e_1} E$ and $K_2 \xrightarrow{l_2} L_2 \xrightarrow{e_2} E$ as depicted below exist.*

*Given an E-dependency relation $(e_1, e_2) \in \mathcal{E}$ for rules p_1, p_2, the E-*concurrent rule *$p_1 *_E p_2$ is defined by $p_1 *_E p_2 := (L \xleftarrow{l \circ k_1} K \xrightarrow{r \circ k_2} R)$ as shown below, where $(9a)$ and $(9b)$ are pushouts and (10) is a pullback.*



The amalgamated rule combines the actions of two, maybe parallel dependent, rule applications into one rule [1,3].

Definition 13 (Amalgamated rule). *Given a common subrule $p = (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$ of rules $p_i = (L_i \overset{l_i}{\hookleftarrow} K_i \overset{r_i}{\hookrightarrow} R_i)$, $i = 1, 2$, the amalgamated rule $p_1 \oplus_p p_2 = (L' \overset{l'}{\hookleftarrow} K' \overset{r'}{\hookrightarrow} R')$ is constructed by taking the three pushouts depicted to the right, where morphisms l', r' are given by the universal property of pushout object K'.*



We now relate short-cut rules to E-concurrent and amalgamated rules of rules, where the first rule only deletes and the second rule only creates. Further, we take $\mathcal{E}$ to be the class of pairs of morphisms which are jointly epimorphic and where both morphisms are monomorphisms, i.e., the following statements for concurrent rules hold under that assumption. To begin, both concurrent and amalgamated rules "degenerate" in that setting. They are merely constructed as sums over constant rules.

Lemma 14 (Degeneration). *Let two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, in an adhesive category $\boldsymbol{C}$ be given. Then the classes of E-concurrent rules and amalgamated rules for r_1^{-1} and r_2 coincide. In particular, they both coincide with $C := \{r_1^{-1} \oplus_p r_2 \,|\, p = (X_1 \overset{x_1}{\hookleftarrow} X \overset{x_2}{\hookrightarrow} X_2),\ x_1, x_2$ isomorphisms, and p common subrule of $r_1, r_2\}$, i.e., the class of rules amalgamated along a common constant subrule of r_1^{-1} and r_2.*

As a consequence of the above lemma, in our context every E-concurrent or amalgamated rule can be constructed as a short-cut rule. On the contrary, concrete examples show that short-cut rules exist which cannot be constructed as E-concurrent or amalgamated rule (and hence neither as parallel rule).

Proposition 15 (Subsumption). *Let two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, in an adhesive category $\boldsymbol{C}$ be given. Then every E-concurrent or amalgamated rule for r_1^{-1} and r_2 coincides with a short-cut rule for them, but generally not the other way around, i.e., generally the class C of E-concurrent and amalgamated rules for r_1^{-1} and r_2 (Lemma 14) is properly contained in the class $C' := \{r_1^{-1} \ltimes_k r_2 \,|\, k : L_\cap \hookrightarrow R_\cap$ is a common kernel rule for $r_1, r_2\}$ of short-cut rules for r_1^{-1} and r_2.*

Idea of Proof. To show the containment relationship, it suffices to check that $r_1^{-1} \oplus_p r_2 = r_1^{-1} \ltimes_p r_2$ for a common constant subrule p of r_1^{-1} and r_2 (in particular, p is a common kernel rule for r_1 and r_2).

As stated in Example 4, *Root-To-Sub-Short-Cut-Rule* is the short-cut rule for the inverse rule of *CreateRoot-Rule* and *CreateSub-Rule* along *CreateRoot-Rule* as common kernel rule. Their parallel rule and the only possibility for an amalgamated or E-concurrent rule is the second rule depicted in Fig. 12, which differs from the short-cut rule in its interface graph. □

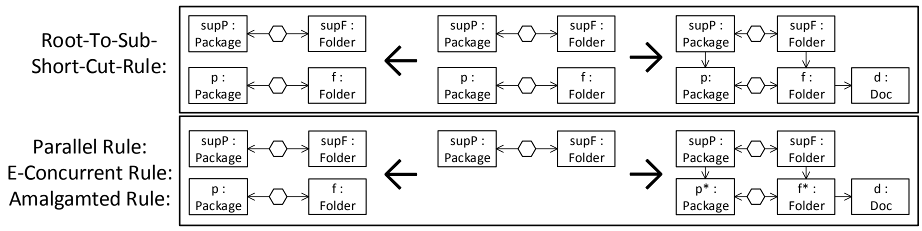


Fig. 12. Relating short-cut rule to other formalisms of rule composition

6 Conclusion

In this paper, we formally introduced short-cut rules for monotonic rules in adhesive HLR systems, a novel kind of rule composition. We proved that short-cut rules preserve information instead of deleting elements and recreating them again, when revoking a transformation and applying another one instead. Additionally, we gave examples using a TGG where applying short-cut rules spares us rolling back whole chains of transformations, thus providing ‘short-cuts’ when revising those. Moreover, we proved short-cut rules to differ from the already established formalizations for composition of rules, i.e., the parallel, concurrent, and amalgamated rules.

Besides developing language-preserving short-cut rules (as already discussed at the end of Sect. 4), we plan to develop a construction of short-cut rules for general rules, also, and advance the theory of short-cut rules by respecting possible application conditions of the involved rules. On the practical side, we plan to operationalize short-cut rules stemming from TGGs to enhance model synchronization.

Acknowledgments This work was partially funded by the German Research Foundation (DFG), project “Triple Graph Grammars (TGG) 2.0”.

References

1. Boehm, P., Fonio, H.R., Habel, A.: Amalgamation of graph transformations. A synchronization mechanism. Journal of Computer and System Sciences **34**(2), 377–408 (1987). https://doi.org/10.1016/0022-0000(87)90030-4
2. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science, Springer (2006). https://doi.org/10.1007/3-540-31188-2
3. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: $\mathcal{M}$-adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation. Mathematical Structures in Computer Science **24**(4), 240406 (2014). https://doi.org/10.1017/S0960129512000357
4. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science **19**(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202
5. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. Theoretical Informatics and Applications **39**(3), 511–545 (2005). https://doi.org/10.1051/ita:2005028
6. Leblebici, E., Anjorin, A., Fritsche, L., Varró, G., Schürr, A.: Leveraging Incremental Pattern Matching Techniques for Model Synchronisation. In: de Lara, J., Plump, D. (eds.) 10th International Conference on Graph Transformation. Lecture Notes in Computer Science, vol. 10373, pp. 179–195. Springer (2017). https://doi.org/10.1007/978-3-319-61470-0_11
7. Leblebici, E., Anjorin, A., Schürr, A., Taentzer, G.: Multi-amalgamated triple graph grammars: Formal foundation and application to visual language translation. Journal of Visual Languages & Computing **42**, 99–121 (2017). https://doi.org/10.1016/j.jvlc.2016.03.001
8. Leblebici, E., Anjorin, A., Schürr, A.: Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In: Huisman, M., Rubin, J. (eds.) Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, vol. 10202, pp. 191–207. Springer (2017). https://doi.org/10.1007/978-3-662-54494-5_11
9. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, vol. 903, pp. 151–163. Springer (1995). https://doi.org/10.1007/3-540-59071-4_45

A Adhesive Categories

In this section, we recall adhesive categories and those of their properties that we make use of in the proofs presented in Appendices B and C. Adhesive categories can be understood as categories where pushouts along monomorphisms behave like pushouts along injective maps in the category of sets. The definition of an adhesive category uses the notion of van Kampen squares.

Definition 16 (Van Kampen square and adhesive category). *A pushout diagram as depicted in Fig. 13 is a* van Kampen square *if for every commutative cube over it (like depicted in Fig. 14) where the backfaces are pullbacks, the front faces are pullbacks iff the top face is a pushout.*

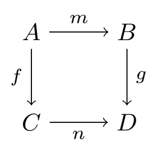


Fig. 13. A pushout square

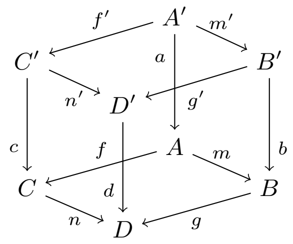


Fig. 14. Commutative cube over pushout square

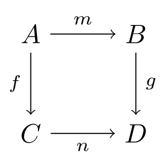
A category ***C*** *is called* adhesive *if*

1. ***C*** *has pushouts along monomorphisms (i.e., pushouts whenever at least one of the two morphisms f or m in Fig. 13 is a monomorphism),*
2. ***C*** *has pullbacks, and*
3. *pushouts along monomorphisms are van Kampen squares.*

Important examples of adhesive categories include the categories of sets, of (typed) graphs, and of (typed) triple graphs [5,2]. We will use the following properties of adhesive categories frequently:

Fact 17 (Properties of adhesive categories). *If* ***C*** *is an adhesive category, the following properties hold [5]:*

1. *Monomorphisms are stable under pushout, i.e., whenever m (or f) is a monomorphism in the pushout diagram to the right, n (or g) is a monomorphism. Moreover, pushouts along monomorphisms are pullbacks.*



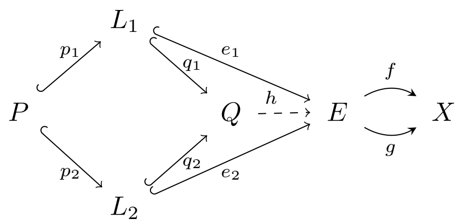
2. *If f is a monomorphism (compare the diagram above), pushout complements for $n \circ f$ are unique (up to isomorphism).*

3. *The category $\boldsymbol{C}$ is balanced, i.e., each morphism which is a mono- and an epimorphism is already an isomorphism.*
4. *The subobjects of an object in an adhesive category form a distributive lattice. Given two subobjects $a, b : A, B \hookrightarrow C$ of an object C in category $\boldsymbol{C}$, their meet $A \cap B$ is given by taking the pullback of a and b and their join $A \cup B$ by taking the pushout of their meet. Particularly, the join $A \cup B$ is a subobject of C.*

While every pushout along a monomorphism is a pullback in an adhesive category, not every pullback along a monomorphism is a pushout again. The following Corollary characterizes a special situation where a pullback already is a pushout.

Corollary 18 (Pullbacks as pushouts). *In any adhesive category $\boldsymbol{C}$, let $(e_1, e_2) : L_1, L_2 \hookrightarrow E$ be a pair of jointly epimorphic morphisms such that both of them are monomorphisms. Then E is (isomorphic to) the pushout of the pullback of (e_1, e_2).*

Proof. Given the diagram below, where P arises as pullback of e_1, e_2, Q as pushout of p_1, p_2, and the morphism h from the universal property of Q, we show that h is an isomorphism. Since adhesive categories are balanced (Fact 17, Item 3), it suffices to show that h is a monomorphism and an epimorphism.



Since e_1, e_2 are monomorphisms, the morphism h is a monomorphism (Fact 17, Item 4). Given two morphisms $f, g : E \to X$ with $f \circ h = g \circ h$, it follows that

$$\begin{aligned} f \circ h = g \circ h &\Rightarrow f \circ h \circ q_1 = g \circ h \circ q_1 \\ &\Rightarrow f \circ e_1 = g \circ e_1 \end{aligned}$$

and analogously $f \circ e_2 = g \circ e_2$. Since e_1, e_2 are jointly epimorphic, it follows that $f = g$. Thus, h is an epimorphism. □

B Detailed Proofs

This section contains the proofs that where omitted in the presentation of the paper except for the proof of Lemma 9 which is presented separately in Appendix C. We mainly just repeat the statements to be proven and give the proofs subsequently. We follow the order in which the statements occur in the paper.

Lemma 5 (Linearity of short-cut rule). *In an adhesive category $\boldsymbol{C}$, given two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, and a common kernel rule $k : L_\cap \hookrightarrow R_\cap$ for them, the induced morphisms $l : K \to L$ and $r : K \to R$ in the short-cut rule $r_1^{-1} \ltimes_k r_2$ are monomorphisms.*

Proof. For the following proof compare Fig. 15 where the outer square corresponds to pushout $(1b)+(3b)$ in Fig. 5 and (4) is the pushout from Fig. 6. Since $(1b)+(3b)$

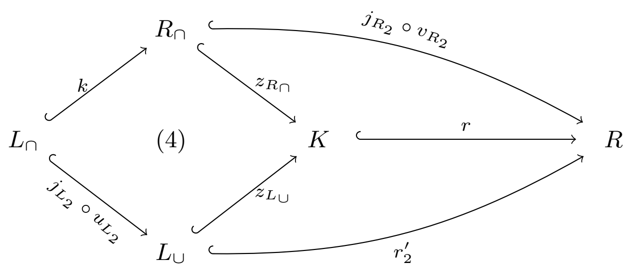


Fig. 15. Linearity of short-cut rules

is a pushout along a monomorphism, it is also a pullback (Fact 17, Item 1), namely of the two monomorphisms $r_2' : L_\cup \hookrightarrow R$ and $j_{R_2} \circ v_{R_2} : R_\cap \hookrightarrow R$. Thus, pushout (4) results as pushout of the pullback (1) + (3). Hence, the unique morphism $r : K \hookrightarrow R$ obtained by the universal property of the pushout K is a monomorphism (see Fact 17, Item 4). Analogously, the morphism $l : K \hookrightarrow L$ is a monomorphism. ☐

Lemma 8 (Preservation). *In an adhesive category $\boldsymbol{C}$, let $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, be two monotonic rules and $k : L_\cap \hookrightarrow R_\cap$ a common kernel rule for them. Let $g : G \hookrightarrow G'$ be a monomorphism arising by synthesis of a transformation sequence $G_1 \Rightarrow_{r_1^{-1}, m_1} G \Rightarrow_{r_2, m_2} G_2$ or by analysis of a transformation $G_1 \Rightarrow_{r_1^{-1} \ltimes_k r_2, m_1'} G_2$ (compare Theorem 7, especially Figs. 10 and 11). Then g is an isomorphism iff k is.*

Proof. The morphism g stems from first computing pushout (4) along $k : L_\cap \hookrightarrow R_\cap$ (compare Fig. 16) and pushout (6) along $z_{L_\cup} : L_\cup \hookrightarrow K$, subsequently. Since pushouts along isomorphisms result in isomorphisms, g is an isomorphism if k is one. The other way around, pushouts (4) and (6) are pullbacks in an adhesive category and pullbacks along isomorphisms result in isomorphisms again. Thus, if g is an isomorphism, first $z_{L_\cup}$ and subsequently k are both isomorphisms, too. ☐

Proposition 11 (Relation to parallel rule). *Let two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, in an adhesive category $\boldsymbol{C}$ with strict initial object $\emptyset$ be given. Then, for the trivial common kernel rule $id_\emptyset : \emptyset \hookrightarrow \emptyset$, the short-cut and the parallel rule coincide, i.e., $r_1^{-1} + r_2 = r_1^{-1} \ltimes_{id_\emptyset} r_2$.*

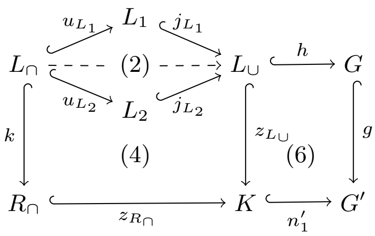


Fig. 16. Relation between morphisms $k : L_\cap \hookrightarrow R_\cap$ and $g : G \hookrightarrow G'$

Proof. In case of monotonic rules $r_1^{-1} + r_2 = (R_1 + L_2 \xleftarrow{r_1+l_2} L_1 + L_2 \xrightarrow{l_1+r_2} L_1 + R_2)$, where l_1, l_2 are isomorphisms. Calculating $r_1^{-1} \ltimes_k r_2$, first the pushout (2) in Definition 3 leads to $L_\cup = L_1 + L_2$ and j_{L_i} being the coprojection of L_i into the coproduct, $i = 1, 2$. It is easy to check that pushouts $(3a)$ and $(3b)$ are given by $R_1 + L_2$ and $L_1 + R_2$, respectively, with morphisms $r_1 + l_2 : L_\cup = L_1 + L_2 \hookrightarrow R_1 + L_2$ and $l_1 + r_2 : L_\cup = L_1 + L_2 \hookrightarrow L_1 + R_2$, respectively. Furthermore, computing pushout (4) leads to $K = L_\cup = L_1 + L_2$ since $id_\emptyset : \emptyset \hookrightarrow \emptyset$ is the identity morphism. Thus, $r_1^{-1} \ltimes_{id_\emptyset} r_2 = r_1^{-1} + r_2$. □

As was already stated in the main text, the next Lemma and Proposition assume $\mathcal{E}$ to be the class of pairs of morphisms which are jointly epimorphic and where both morphisms are monomorphisms.

Lemma 14 (Degeneration). *Let two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, in an adhesive category $\boldsymbol{C}$ be given. Then the classes of E-concurrent rules and amalgamated rules for r_1^{-1} and r_2 coincide. In particular, they both coincide with $C := \{r_1^{-1} \oplus_p r_2 \mid p = (X_1 \xleftarrow{x_1} X \xrightarrow{x_2} X_2),\ x_1, x_2$ isomorphisms, and p common subrule of $r_1, r_2\}$, i.e., the class of rules amalgamated along a common constant subrule of r_1^{-1} and r_2.*

Proof. We show that both the classes of amalgamated and E-concurrent rules coincide with the class C.

First, by definition every element of C is an amalgamated rule for r_1^{-1} and r_2. For the converse, compare Fig. 17 which depicts an amalgamation along a subrule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ of r_1^{-1} and r_2. Since r_1, r_2 are monotonic, both l_1 and l_2 are isomorphisms. Hence, l and r arise as pullbacks of isomorphisms and, consequently, they both are isomorphisms. Thus, any common subrule p of p_1^{-1} and p_2 is of the form $p = (X_1 \xleftarrow{x_1} X \xrightarrow{x_2} X_2)$ with x_1, x_2 being isomorphisms. Consequently, $p' = p_1^{-1} \oplus_p p_2$.

Secondly, for a constant subrule $p = (X_1 \xleftarrow{x_1} X \xrightarrow{x_2} X_2)$ of r_1^{-1} and r_2 by definition there exist monomorphisms $w_{L_1} : X \hookrightarrow L_1$ and $w_{L_2} : X \hookrightarrow L_2$. Computing their pushout (depicted as square (11) in Fig. 18), leads to $(e_1, e_2) \in \mathcal{E}$: as pushouts along a monomorphism they both are monomorphisms and they are jointly epimorphic, since they are coprojections of a pushout by construction.

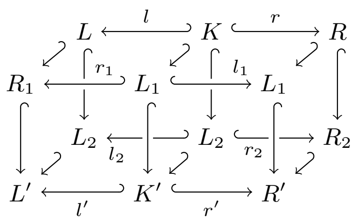


Fig. 17. Amalgamation along a subrule of p_1^{-1} and p_2

The pushout (11) coincides with the pushout in the middle of Fig. 17 for $p = (X_1 \overset{x_1}{\hookleftarrow} X \overset{x_2}{\hookrightarrow} X_2) = (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$. Since l_1 and l_2 are isomorphisms, squares $(8a)$ and $(8b)$ in Fig. 18 can be computed as pushouts (instead of pushout complements) and K' is (up to isomorphism) the resulting pushout object. Pushout $(9a)$ may equivalently be computed as pushout of $w_{L_2} : X \hookrightarrow L_2$ and $r_1 \circ l_1^{-1} \circ w_{L_1} : X \hookrightarrow R_1$, i.e., as pushout $(9a) + (8a) + (11)$. Because $X \cong X_1$ (or $K \cong L$), this is the same as the left pushout square in Fig. 17, i.e., L' with morphism $l' : K' \hookrightarrow L'$ is a pushout object for $(9a)$. Completely analogous, R' is the result of computing pushout $(9b)$. Thus, $r_1^{-1} \oplus_p r_2 = r_1^{-1} *_{K'} r_2$.

Conversely, let an E-dependency relation (e_1, e_2) for r_1^{-1} and r_2 be given, i.e., jointly epimorphic monomorphisms $e_1, e_2 : L_1, L_2 \hookrightarrow K'$ for some object K'. Corollary 18 states that the pushout of the pullback of (e_1, e_2) is K' again. In particular, (e_1, e_2) are coprojections of a pushout. We denote the pullback of (e_1, e_2) with X and the resulting morphisms by $w_{L_i} : X \hookrightarrow L_i$, $i = 1, 2$ (Fig. 18). Since l_1 and l_2 are isomorphisms, squares $(8a)$ and $(8b)$ can be computed as pushouts (instead of pushout complements). Now, computing the missing squares $(9a), (9b)$ and (10) equals computing $r_1^{-1} \oplus_p r_2$ where $p = (X \overset{id_X}{\hookleftarrow} X \overset{id_X}{\hookrightarrow} X)$. Moreover, p is a constant common subrule of r_1^{-1} and r_2. Thus, $r_1^{-1} *_{K'} r_2 = r_1^{-1} \oplus_p r_2$ for a constant common subrule p of r_1^{-1} and r_2 and hence $r_1^{-1} *_{K'} r_2 \in C$. □

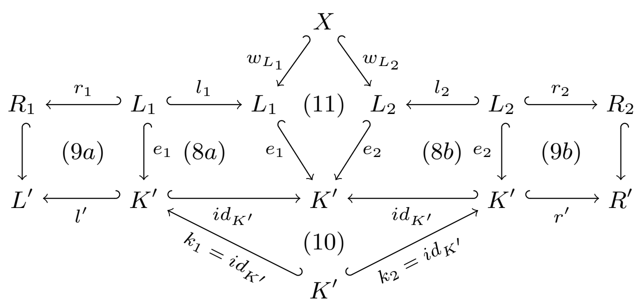


Fig. 18. E-concurrent rule for p_1^{-1} and p_2

Proposition 15 (Subsumption). *Let two monotonic rules $r_i : L_i \hookrightarrow R_i$, $i = 1, 2$, in an adhesive category* $\boldsymbol{C}$ *be given. Then every* E*-concurrent or amalgamated rule for* r_1^{-1} *and* r_2 *coincides with a short-cut rule for them, but generally not the other way around, i.e., generally the class* C *of* E*-concurrent and amalgamated rules for* r_1^{-1} *and* r_2 *(Lemma 14) is properly contained in the class* $C' := \{r_1^{-1} \ltimes_k r_2 \,|\, k : L_\cap \hookrightarrow R_\cap$ *is a common kernel rule for* $r_1, r_2\}$ *of short-cut rules for* r_1^{-1} *and* r_2.

Proof. Lemma 14 states that both amalgamated and E-concurrent rules are of the form $r_1^{-1} \oplus_p r_2$ for a constant common subrule $p = (X_1 \overset{x_1}{\hookleftarrow} X \overset{x_2}{\hookrightarrow} X_2)$ of r_1^{-1} and r_2. By definition of subrule, $x_1 : X \hookrightarrow X_1$ (and analogously $x_2 : X \hookrightarrow X_2$) is a common kernel rule for r_1 and r_2. It is easy to calculate that $r_1^{-1} \ltimes_{x_1} r_2 = r_1^{-1} \oplus_p r_2 (= r_1^{-1} \ltimes_{x_2} r_2)$. Thus, every E-concurrent or amalgamated rule coincides with a short-cut rule.

By presenting an example, we prove that not every short-cut rule can be constructed as parallel, E-concurrent, or amalgamated rule. As stated in Example 4, *Root-To-Sub-Short-Cut-Rule* arises as short-cut rule for the inverse rule of *CreateRoot-Rule* (r_1^{-1}) and *CreateSub-Rule* (r_2) along *CreateRoot-Rule* as common kernel rule. It is depicted again, also showing the interface graph K, as first rule in Fig. 12. It is immediate that the differing second rule in Fig. 12 is the parallel rule of r_1^{-1} and r_2. Since the RHS of r_1^{-1} is empty, there is (up to isomorphism) only one possible choice for an E-dependency relation for r_1^{-1} and r_2: the interface graph K in the second rule in Fig. 12. Computing the corresponding E-concurrent rule results in the second rule of Fig. 12 again. Since the LHS and the RHS of *Root-To-Sub-Short-Cut-Rule* are given as sums of the respective LHSs and RHSs of r_1^{-1} and r_2, the only possibility for a common subrule p such that the LHS and RHS of $r_1^{-1} \oplus_p r_2$ equal the LHS and RHS of *Root-To-Sub-Short-Cut-Rule* is $p = (\emptyset \hookleftarrow X \hookrightarrow \emptyset)$ for some triple graph X. But that implies $X = \emptyset$, which results in the parallel rule again. □

C Characterization of Pushout Complements

The aim of this section is to prove the following Lemma:

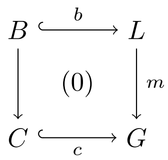
Lemma 9 (Characterization of PO-complements). *In any adhesive category with initial pushouts, given a commutative diagram like Fig. 10 where* (3a) *and* (7a) *are pushouts, a pushout complement object* G *for* $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ *is a pushout complement object for* $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ *and vice versa. Particularly, a pushout complement for* $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ *exists iff a pushout complement for* $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ *exists.*

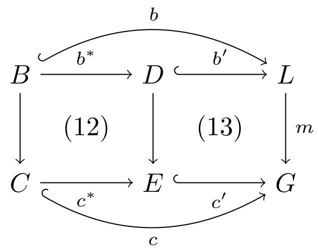
Proving that a pushout complement G for $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ is also a pushout complement for $m_1 \circ r_1 : L_1 \hookrightarrow G_1$ is easy. We first sketch the set-theoretic idea behind the proof of the other direction: Given a pushout complement G for the outer square in Fig. 10, since n_1' is injective, a pushout complement for $n_1' \circ z_{L_\cup} : L_\cup \hookrightarrow G'$ does not exist only if n_1' maps a node x of K without adjacent edge and without preimage in $L_\cup$ to a node with adjacent edge in G'. But since $(3a)$ and $(7a)$ are pushouts, node $l(x)$ has a preimage in R_1 which has no preimage in L_1 and is mapped to a node with an adjacent edge in G_1 by m_1 such that the adjacent edge has no preimage in R_1. This is a contradiction to the existence of a pushout complement for $m_1 \circ r_1$.

To prove the above Lemma in the more general setting of adhesive categories, we need to be able to characterize the situations in which a pushout complement exists in more general terms. This is possible using the notion of *initial pushouts* [2]. Therefore, we first introduce initial pushouts and some of their properties and subsequently use those to lift the above sketched idea to prove Lemma 9 to the setting of adhesive categories.

In the category of graphs, an initial pushout over a morphism $m_1 : R_1 \to G_1$ consists of a *boundary graph* B and a *context graph* C where B embeds into R_1 and C embeds into G_1. The graph B is the "smallest" graph such that a context graph C can be found so that the gluing of C and R_1 along B results in G_1. Therefore, if m_1 is injective, B consists exactly of those nodes of R_1 which get mapped to a node with an adjacent edge in G_1 that has no preimage under m_1.

Definition 25 (Boundary and initial pushout [2]). *Given a morphism $m : L \to G$ in an adhesive category $\boldsymbol{C}$, an* initial pushout *over m is a pushout* (0) *such that b is a monomorphism and* (0) *factors uniquely through every pushout* (13) *over m where b' is a monomorphism. I.e., for every pushout* (13) *over m where b' is a monomorphism, there exist unique morphisms b^*, c^* with $b = b' \circ b^*$ and $c = c' \circ c^*$. If* (0) *is an initial pushout, b is called a* boundary *over m_1, B the* boundary object *and C the* context object *with respect to m.*





Fact 26 (Properties initial pushout). *Given an initial pushout over a morphism $m : L \to G$ like depicted above, the following holds:*

- *By decomposition of monomorphisms, b^* and c^* are also monomorphisms.*
- *The square* (12) *is a pushout ([5, Lemma 4.7]).*
- *If m is a monomorphism, every morphism in diagram* (12) + (13) *is a monomorphism.*

Initial pushouts can be used to characterize matches for which a rule is applicable.

Fact 27 (Existence and uniqueness of contexts [2]). *In an adhesive category with initial pushouts, given a rule $p = (L \overset{l}{\hookleftarrow} K \overset{r}{\hookrightarrow} R)$ and a match*

$m : L \hookrightarrow G$ for that rule, the rule p is applicable at match m (i.e., the context object D for application of the rule exists) iff there exists a morphism $b^* : B \to K$ with $l \circ b^* = b$ where B is the boundary object with respect to m and b is the boundary over m (compare Fig. 19).

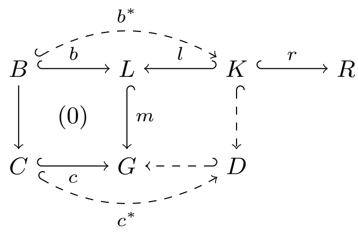


Fig. 19. Initial pushout and context object

Remark 28. Since b is a monomorphism, by decomposition of monomorphisms the morphism b^* in the above Fact is a monomorphism, too.

Initial pushouts over monomorphisms have the following closure property:

Fact 29 (Closure property of initial pushouts [2]). *In an adhesive category, given an initial pushout* (0) *over a monomorphism $m : L \hookrightarrow G$ and a pushout diagram* (14) *where $a : A \hookrightarrow L$ is a monomorphism, also $d : A \to A'$ is a monomorphism and B and C are also boundary (resp. context) objects with respect to d and b' the boundary over d, where b' is the monomorphism induced by initiality of* (0) *(compare Fig. 20).*

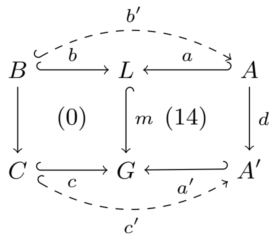


Fig. 20. Closure of initial pushouts

Using the just introduced notions, we are able to prove Lemma 9.

Proof (Proof of Lemma 9). A pushout complement G for square (6) in Fig. 21 is a pushout complement for the outer square $(m_1 \circ r_1)$ as in the proof of Theorem 7 and pushout complements are unique (up to isomorphism) in adhesive categories.

Now, assume a pushout complement G for the larger right square $((3a)+(7a)+(6))$ in Fig. 21 to exist. According to Fact 27, this is equivalent to the existence of monomorphism $b^*: B \hookrightarrow L_1$ such that $r_1 \circ b^* = b$, where $(0a)$ is the initial pushout over m_1. Let $(0b)$ denote the initial pushout over monomorphism m_1' (with boundary b'). It is also the initial pushout over n_1' (with boundary b'') because n_1' is a monomorphism and Fact 29. To show that a pushout complement $\bar{G}$ for $n_1' \circ z_{L_\cup}: L_\cup \hookrightarrow G'$ exists, it suffices to show the existence of a morphism $b''^*: B' \to L_\cup$ such that $z_{L_\cup} \circ b''^* = b''$ (again, according to Fact 27). If such an object $\bar{G}$ exists, it is also a pushout complement for $m_1 \circ r_1$ (compare the proof of Theorem 7) such that $\bar{G} \cong G$ since pushout complements along monomorphisms are unique in adhesive categories. We show the existence of the morphism b''^* in several steps.

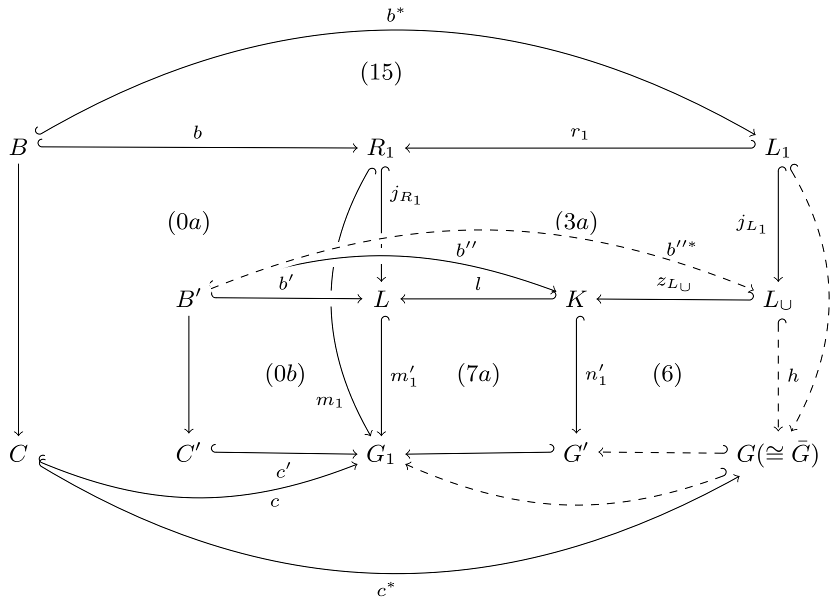


Fig. 21. Overview for proof of equivalence of existence of pushout complements

First, compare Fig. 22. Let $B' \cap R_1$ denote the pullback of b' and j_{R_1}. We show the existence of a monomorphism $s_B: B' \cap R_1 \hookrightarrow B$ such that the triangle (16) commutes. Take the pullback (18) of m_1' and c. Both are monomorphisms and since c and $m_1' \circ j_{R_1}$ are jointly epimorphic, also c and m_1' are jointly epimorphic. Thus, Corollary 18 ensures that (18) is also a pushout. Hence, the initial pushout $(17)+(18)$ over m_1' factors uniquely through (18) and a monomorphism $C' \hookrightarrow C$ exists. Now, pushout $(0a)$ is also a pullback and its universal property as pullback guarantees the existence of s_B such that (16) commutes. Moreover, s_B is a monomorphism because of the decomposition property of monomorphisms: As pullback of monomorphism b', s_{R_1} is a monomorphism.

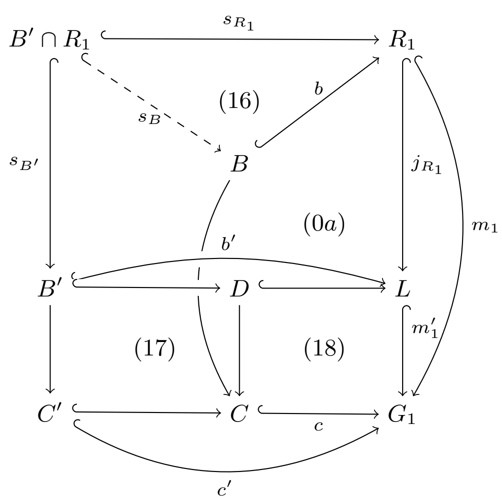


Fig. 22. Existence of monomorphism $s_B : B' \cap R_1 \hookrightarrow B$

Monomorphism s_B can now be used to prove the existence of b''^* as depicted in Fig. 23. First take pullbacks (19a) and (19b) of j_{R_1} and b' and b' and $l \circ z_{L_\cup}$, respectively. The monomorphisms $s_{B'}$ and $t_{B'}$ are the coprojections of pushout (20), since $R_1 \cup L_\cup = L$. Furthermore,

$$\begin{aligned} l \circ z_{L_\cup} \circ j_{L_1} \circ b^* \circ s_B \circ o_{B' \cap R_1} &= j_{R_1} \circ r_1 \circ b^* \circ s_B \circ o_{B' \cap R_1} \\ &= j_{R_1} \circ b \circ s_B \circ o_{B' \cap R_1} \\ &= j_{R_1} \circ s_{R_1} \circ o_{B' \cap R_1} \\ &= b' \circ s_{B'} \circ o_{B' \cap R_1} \\ &= l \circ b'' \circ s_{B'} \circ o_{B' \cap R_1} \\ &= l \circ b'' \circ t_{B'} \circ o_{B' \cap L_\cup} \\ &= l \circ z_{L_\cup} \circ t_{L_\cup} \circ o_{B' \cap L_\cup} \end{aligned}$$

and since $l \circ z_{L_\cup}$ is a monomorphism $j_{L_1} \circ b^* \circ s_B \circ o_{B' \cap R_1} = t_{L_\cup} \circ o_{B' \cap L_\cup}$. Hence, a unique morphism $b''^* : B' \to L_\cup$ with $b''^* \circ t_{B'} = t_{L_\cup}$ and $b''^* \circ s_{B'} = j_{L_1} \circ b^* \circ s_B$ exists.

With those equation, it is finally possible to compute

$$\begin{aligned} l \circ z_{L_\cup} \circ b''^* \circ t_{B'} &= l \circ z_{L_\cup} \circ t_{L_\cup} \\ &= b' \circ t_{B'} \\ &= l \circ b'' \circ t_{B'} \quad , \end{aligned}$$

thus $z_{L_\cup} \circ b''^* \circ t_{B'} = b'' \circ t_{B'}$ since l is mono and similarly

$$\begin{aligned} l \circ z_{L_\cup} \circ b''^* \circ s_{B'} &= l \circ z_{L_\cup} \circ j_{L_1} \circ b^* \circ s_B \\ &= j_{R_1} \circ r_1 \circ b^* \circ s_B \\ &= j_{R_1} \circ b \circ s_B \\ &= j_{R_1} \circ s_{R_1} \\ &= b' \circ s_{B'} \\ &= l \circ b'' \circ s_{B'} \ , \end{aligned}$$

thus $z_{L_\cup} \circ b''^* \circ s_{B'} = b'' \circ s_{B'}$ since l is mono. Since $s_{B'}$ and $t_{B'}$ are jointly epimorphic, it follows that $z_{L_\cup} \circ b''^* = b''$. □

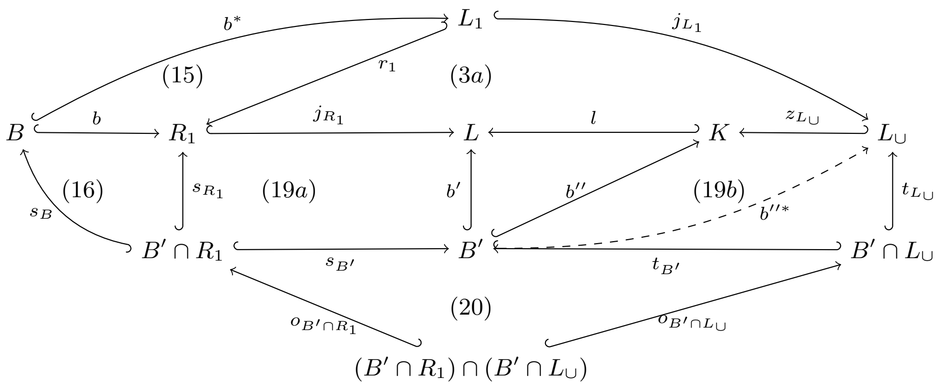


Fig. 23. Existence and properties of b''^*