Constructing Constraint-Preserving Interaction Schemes in Adhesive Categories

Extended Version

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

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

Abstract. When using graph transformations to formalize model transformations, it is often desirable to design transformations that preserve consistency with respect to a given set of (model) integrity constraints. The standard approach is to equip transformations with suitable application conditions such that the introduction of constraint violations is prevented. This may lead to rules that are applicable seldom or even inapplicable at all, though. To supplement this approach, we present a new and systematic procedure to develop correct-by-construction transformations with respect to a special kind of constraints. Instead of controlling the applicability of a rule we complement its action in such a way that a given constraint holds after application: For every way in which the rule could introduce a violation of the constraint, we derive a supplementary action for the rule that remedies that violation. We formalize this construction in the setting of adhesive categories for monotonic rules and positive atomic constraints and present sufficient conditions for its correctness.

Keywords: Algebraic Graph Transformation · Multi-Amalgamation · Nested Graph Constraints · Correctness-by-Construction

1 Introduction

Algebraic graph transformation [4] has proved to be a suitable formal framework to reason about model transformations [3]. In application scenarios like model generation and editing or refactoring of models, it is desirable that transformations preserve consistency with respect to a set of (model) integrity constraints. Nested constraints [7] allow to express (first-order) properties of graphs and (a large subset of) constraints formulated in OCL [16] – a widespread constraint language in modeling – may be (automatically) translated into those [17,13]. In the context of algebraic graph transformation, the standard approach to ensure the validity of results of transformations with respect to a constraint is to equip transformation rules with suitable application conditions. This approach is elaborated for arbitrarily nested constraints in $\mathcal{M}$-adhesive categories [7] and has tool support for

EMF model transformations [13]. It comes in two variants: Given a constraint c and a rule, one can construct a *c-guaranteeing* and a *c-preserving* application condition for the rule. The c-guaranteeing application condition ensures that the rules' application is possible if and only if the constraint c is fulfilled afterwards. The c-preserving one is logically weaker: The constraint c is only ensured to be fulfilled after a rule application if it was so before. Though sound from the formal point of view for every constraint, from the practical point of view the results are especially satisfactory in the case of *negative* constraints, which forbid a certain structure to exist. Then the new application conditions prohibit applications of the rule that would introduce this structure. However, for *positive* constraints requiring structures to exist, an application of a rule may be prohibited, e.g., because it creates a structure that necessitates another structure to exist that is not created likewise. In this way, frequently the application conditions stemming from positive constraints lead to rules which are applicable only rarely or are even inapplicable at all (see Sect. 2 for a concrete example).

To supplement this just described approach, we develop an alternative construction. Given a positive constraint c, instead of using application conditions, our idea is to *complement the action of a rule* in such a way that c holds after its application. Our construction works for *monotonic rules*, i.e., rules which only create structure, and *positive atomic constraints*. Some instance G satisfies a positive atomic constraint – which may be compactly notated as $\forall\,(P, \exists\, C)$ where P is a subobject of C – if for all subobjects of G that are isomorphic to P there exists a subobject isomorphic to C which includes the image of P. Positive atomic constraints are highly relevant in practice, e.g., they occur frequently when translating OCL into graph constraints and the application conditions arising from them in the standard approach are often way to restrictive.

The crucial idea of our approach is to calculate all possible ways in which the application of a rule r may lead to a new match for the *premise* P of a constraint $c = \forall\,(P, \exists\, C)$. These are the different ways in which an application of r may introduce a new violation of c. They can significantly differ from each other and thus require diverging actions to resolve the would-be introduced violation. Hence, for each such situation we derive a rule that includes r as subrule but additionally creates structure that complements the new match for P to a new match for C. All these rules are collected into an *interaction scheme* [6] that is *constraint-preserving*: Applying an interaction scheme means to apply a common subrule – here r – once and every other rule from the interaction scheme as often as possible but with fixed partial match given by the match of the common subrule (the common actions are only performed once). In this way, every image of P that gets newly created by an application of r is complemented to an image of C by application of one of the rules of the interaction scheme and the validity of the constraint is preserved. Slightly extending this construction also gives a *constraint-guaranteeing interaction scheme*, i.e., an interaction scheme that additionally includes rules that "repair" already existing violations.

The original motivation for this research is to continue work from [12]. There, multi-rules for triple graph grammars (TGGs) [19] – a formalism for the declarative description of consistency relationships between two modeling languages with graph-like representations – have been developed. The exemplary multi-rules in [12] serve to preserve consistency with (informally described) constraints but were developed "by hand". This has the disadvantage that preservation of the constraint is not ensured by construction and has to be checked on a case-by-case basis. Our work paves the way to automate the design of those multi-rules in a way that guarantees correctness by construction. Our restriction to monotonic rules corresponds with that motivation since TGGs are composed of those.

The main contribution of this paper is the introduction and formalization of constraint-preserving and -guaranteeing interaction schemes in the setting of adhesive categories [11]. It is organized as follows. In Sect. 2 we illustrate our results with an example. Section 3 recalls relevant background. In Sect. 4 we present our construction of interaction schemes while Sect. 5 points out further possibilities to refine that construction. Section 6 compares to related work before we conclude in Sect. 7. This extended version differs from the original paper in the following respects: A short introduction to adhesive categories and those of their properties important for our proofs is given in Appendix A. These proofs are given in Appendix B. Moreover, the main text includes additional definitions and an additional lemma, clearly marked as such. Example 1 is extended and Appendix C illustrates a concept, merely stated in Sect. 5, in more detail.

2 Introductory Example

We adapt the example from [12] since, in a way, we continue the work of constructing multi-amalgamated rules for triple-graph grammars (TGGs). There, a TGG for co-evolution of a class diagram and a documentation structure is given. We use a simplified version of this example, namely a plain graph grammar.

The meta-model in Fig. 1 is a blueprint for simple documentation structures. They consist of Docs owning Entries. Moreover, Docs reference Docs and Entries. Figure 2 presents a grammar allowing to create instances of the meta-model. Black elements have to exist for a rule to be applicable and green elements (additionally marked with ++) are newly created upon application. The rules allow for creation of a new Doc, insertion of a reference between existing Docs, and creation of an Entry to an existing Doc, respectively.

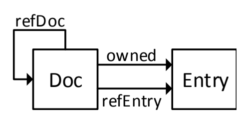


Fig. 1. Meta-model for documentation structures

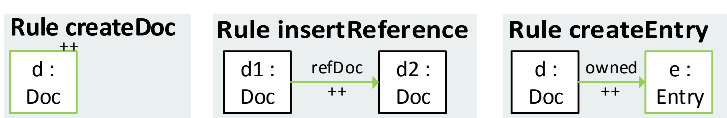


Fig. 2. Monotonic rules to create documentation structures

If an instance of that meta-model is to be understood as documentation structure for a class-diagram, it is reasonable, e.g., to expect owned Entries of referenced Docs to be referenced, too. This constraint is expressible as positive atomic constraint. Figure 3 depicts it in an intuitive graphical representation.

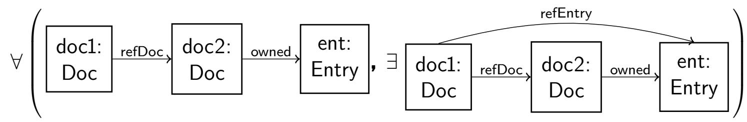


Fig. 3. Constraint *PropagationOfEntries*

The (simplified) constraint-preserving version of *createEntry* with respect to *PropagationOfEntries* is depicted in Fig. 4. Its application is prohibited at every Doc that is already referenced by another one. Thus, the creatable instances and the order of possible rule applications are severely restricted. As an alternative, we will automatically derive so-called multi-rules like *createEntry-multi* depicted in Fig. 5. Applying it as so-called *interaction scheme* with kernel rule *createEntry* at a Doc still creates an Entry but additionally inserts a reference to this Entry from every Doc that references the chosen Doc. This is equivalent to applying its complement rule (Fig. 6) at every possible match with fixed partial match induced by a precedent application of *createEntry*. Applying, e.g., the rule *createEntry* at

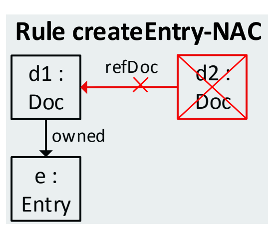


Fig. 4. Rule *createEntry* with *negative application condition* (the crossed-out red elements are forbidden to exist)

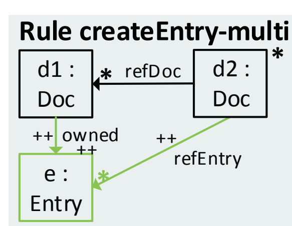


Fig. 5. Multi-rule of *createEntry* (when seen as interaction scheme, the parts decorated with a Kleene star are applied as often as possible)

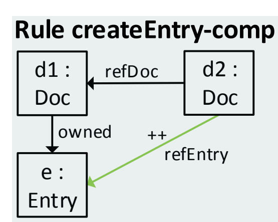


Fig. 6. Complement rule of *createEntry-multi* for *createEntry* as subrule

node d1 to the graph depicted in Fig. 7 leads to the graph in Fig. 8. This clearly violates the constraint *PropagationOfEntries*. While the rule *createEntry-NAC* is not applicable at that match, applying the rule *createEntry-multi* at it as interaction scheme leads to the valid instance depicted in Fig. 9.

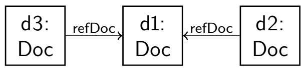


Fig. 7. Example for an instance graph

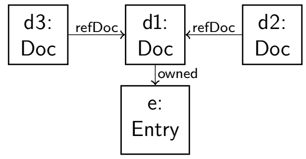


Fig. 8. After application of rule *createEntry* at `d1`

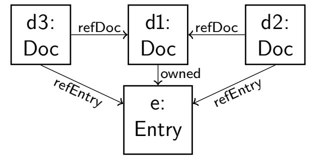


Fig. 9. After application of *createEntry-multi* at `d1`

3 Preliminaries

In this section, we present some technical preliminaries. We conduct our work in the framework of adhesive categories [11]. They have been introduced as a general setting for double pushout rewriting and can be understood as categories where pushouts along monomorphisms behave like pushouts along injective functions in the category of sets and functions. Here, we only recall rules and transformations in adhesive categories with a focus on multi-amalgamation after introducing positive atomic constraints.

To express properties of graphs in a way fitting to the algebraic approach to graph transformation, first graph predicates or graph conditions have been developed (being expressively equivalent to a first-order logic on graphs) and later been generalized to the setting of so-called $\mathcal{M}$-adhesive categories [18,7]. Our approach deals with a small but nonetheless in practice highly important fragment of that logic, namely *positive atomic constraints* [4].

Definition 1 (Positive atomic constraint). *In an adhesive category $\mathcal{C}$, a* positive atomic constraint *c is a monomorphism $p : P \hookrightarrow C$ between two objects. We will write $c = \forall\,(P, \exists\, p : P \hookrightarrow C)$ or $\forall\,(P, \exists\, C)$ for short for such a constraint. We call P the* premise *and C the* conclusion *of the constraint.*

An object G satisfies *a positive atomic constraint c, denoted by $G \models c$, if for every monomorphism $g : P \hookrightarrow G$ there exists a monomorphism $q : C \hookrightarrow G$ such that $g = q \circ p$.*

Positive atomic constraints are the only kind of constraints we consider, so we will just call them constraints when this is not apt to introduce misunderstanding.

Rules are a declarative way to define transformations of objects. They consist of a left-hand side L, a right-hand side R, and an interface K. Informally, in the category of graphs, the application of a rule to a graph G means to delete the elements of L and create those of R while preserving the elements stemming from the interface K. A match identifies the "location" in G where this is done. Formally and more generally in adhesive categories, a transformation can be defined using two pushout diagrams.

Definition 2 ((Monotonic) Rule. Transformation). *Given an adhesive category $\boldsymbol{C}$, a* rule *p consists of three objects L, K, and R, called* left-hand side *(LHS),* interface*, and* right-hand side *(RHS), and two monomorphisms*

$l : K \hookrightarrow L, r : K \hookrightarrow R$. *Its* inverse rule *is the rule* $p^{-1} = (R \hookleftarrow K \hookrightarrow L)$. *A rule* $p = (L \hookleftarrow K \hookrightarrow R)$ *is called* monotonic *(or* non-deleting*) if* $l : K \hookrightarrow L$ *is an isomorphism. In that case we just write* $r : L \hookrightarrow R$.

Given a rule $p = (L \hookleftarrow K \hookrightarrow R)$, *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 defined by the diagram to the right where both squares are pushouts.*

$$\begin{array}{ccccc} L & \xleftarrow{\;l\;} & K & \xrightarrow{\;r\;} & R \\ {\scriptstyle m}\downarrow & & \downarrow & & \downarrow{\scriptstyle n} \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

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.*

Since we exclusively consider monotonic rules in this paper, we will just call them rules and give the following definition for the monotonic case only. *Subrules* and their *multi-rules* have a twofold purpose. First, the application of a rule, then called a multi-rule, may be equivalently split into the application of a subrule followed by one of a *complement rule* with respect to that subrule. An example is displayed in Fig. 6; we give a definition after the following one. This decomposition of a rule application allows for important extensions in two directions: coordinated parallelism and a for-each like syntax. A subrule may capture the common behavior of several multi-rules and thus serve as a kernel to amalgamate their respective actions into application of a single *multi-amalgamated* rule. Secondly, often a subrule is intended to be applied once followed by as many applications of the complement rule as possible. *Interaction schemes* unify both ideas into one concept: An interaction scheme consists of a bundle of multi-rules for the same *kernel rule*. Its application is defined by applying the kernel rule once and each of the complement rules of the multi-rules as often as possible with the fixed partial match given by application of the kernel rule.

Definition 3 (Subrule. Multi-rule. Interaction scheme. Application). *A* subrule *or* kernel rule r_0 *of a rule* $r_1 : L_1 \hookrightarrow R_1$ *is a rule* $r_0 : L_0 \hookrightarrow R_0$ *with* kernel morphism $s_1 : r_0 \hookrightarrow r_1$ *consisting of the monic components* $s_{1,L} : L_0 \hookrightarrow L_1$ *and* $s_{1,R} : R_0 \hookrightarrow R_1$ *such that the arising square in the diagram to the right is a pullback. The rule* r_1 *is then called a* multi-rule *for* r_0.

$$\begin{array}{ccc} L_0 & \xrightarrow{\;r_0\;} & R_0 \\ \downarrow{\scriptstyle s_{1,L}} & & {\scriptstyle s_{1,R}}\downarrow \\ L_1 & \xrightarrow{\;r_1\;} & R_1 \end{array}$$

An interaction scheme *is a finite set* $is = \{s_1, \ldots, s_n\}$ *of kernel morphism from a kernel rule* r_0 *to different multi-rules* $r_1 : L_1 \hookrightarrow R_1, \ldots, r_n : L_n \hookrightarrow R_n$ *of* r_0. *The application of the interaction scheme is to an object* G *with kernel match* $m_0 : L_0 \hookrightarrow G$ *is defined as follows: A* maximal matching J *for is is computed, i.e., a family of matches* $(m_j : L_j \hookrightarrow G)_{j \in J}$ *that is (i)* consistent, *i.e.,*

$$m_0 = m_j \circ s_{j,L} \text{ for all } j \in J$$

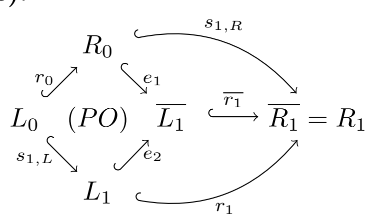
and (ii) maximal, *i.e., no further match for one of the LHSs* $L_1, \ldots, L_n$ *can be added to the family of matches such that it still is consistent.*

Then, the (multi-)amalgamated rule $r_s : L_s \hookrightarrow R_s$, *that is the rule arising by computing the colimits* L_s *of the family of morphisms* $(s_{j,L})_{j \in J}$ *and* R_s *of the family of morphisms* $(s_{j,R})_{j \in J}$ *with* r_s *being induced by the universal property of* L_s, *is applied.*

Remark 1. In adhesive categories, for every finite family J of matches the colimits exist (as iterated pushouts along monomorphisms), the morphism r_s is a monomorphism, and all rules r_j are subrules of the resulting multi-amalgamated rule r_s [6]. In practical applications, e.g., when working with finite graph-like structures, this property is automatically fulfilled.

The following definition is just one way to introduce complement rules; the construction of a rule with the desired properties is not unique. Our choice, also used in [12], is a simple possibility for monotonic rules. A more general, but quite involved one can be found in [6]. The relevant theorems derived there using the more general construction are still valid in our setting. The most important one of these is the Multi-Amalgamation Theorem stating how bundles of transformations may be composed into application of one multi-amalgamated rule and application of a multi-amalgamted rule may be decomposed into a bundle of transformations.

Additional Definition 1 (Complement rule). *In an adhesive category $\mathcal{C}$, given a kernel rule $r_0 : L_0 \hookrightarrow R_0$ with multi-rule $r_1 : L_1 \hookrightarrow R_1$ and kernel morphism $s_1 : r_0 \hookrightarrow r_1$, the* complement rule *$\overline{r_1}$ of r_1 with respect to r_0 is defined as $\overline{r_1} : \overline{L_1} \hookrightarrow \overline{R_1}$, where $\overline{L_1}$ is computed as pushout of r_0 and $s_{1,L}$, $\overline{R_1} = R_1$ and $\overline{r_1}$ is induced by the universal property of the pushout (as depicted to the right).*

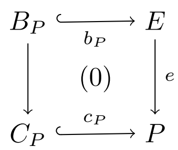


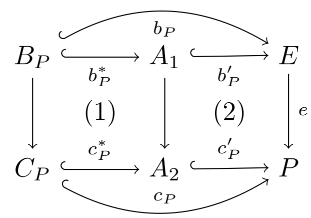
The next Lemma states the well-definedness of the above construction and, more importantly, that an application of a multi-rule may equivalently be split into an application of the kernel rule followed by an application of the complement rule where the matches agree on certain parts. Formally it expresses the multi-rule as a concurrent rule (see [4, Def. 5.21]). The Concurrency Theorem [4, Thm. 5.23] then gives the analysis of the application of the multi-rule into two applications of first the kernel and then the complement rule.

Additional Lemma 1 (Well-definedness and property of complement rule). *In an adhesive category $\mathcal{C}$, given a kernel rule $r_0 : L_0 \hookrightarrow R_0$ with multi-rule $r_1 : L_1 \hookrightarrow R_1$ and kernel morphism $s_1 : r_0 \hookrightarrow r_1$ and the complement rule $\overline{r_1} : \overline{L_1} \hookrightarrow \overline{R_1}$ as defined above, then $\overline{r_1}$ is a monomorphism, i.e., the rule $\overline{r_1} : \overline{L_1} \hookrightarrow \overline{R_1}$ is well-defined. Moreover, $r_1 = r_0 *_{\overline{L_1}} \overline{r_1}$, where $*_{\overline{L_1}}$ denotes the concurrent rule construction with E-dependency object $\overline{L_1}$.*

For formalizing our intended construction, we need a way to express the difference between objects in the setting of adhesive categories. The concept of initial pushouts [4] allows for this.

Definition 4 (Initial pushout). *Given a morphism $e : E \to P$, an* initial pushout *over e is a pushout* (0) *as in the left square below such that b_P is a monomorphism and the pushout* (0) *factors as a pushout* (1) *uniquely through* *every pushout* (2) *over* e *where* b'_P *is a monomorphism as in the right diagram below. Given an initial pushout* (0) *over* e*,* B_P *is called the* boundary object *and* C_P *the* context object with respect to e.





In the category of sets, if the morphism $e : E \to P$ is injective, the set B_P is empty and C_P is isomorphic to $P \setminus e(E)$. In the category of graphs, if e is injective, C_P is the graph arising by adding to $P \setminus e(E)$ those nodes from E necessary to complete it to a graph and B_P consists exactly of those *boundary nodes*. Thus, B_P consists of those nodes of E which get mapped to a node with an adjacent edge in P that has no preimage in E. Thus, we will often use $P \setminus E$ instead of C_P to denote the context object in an initial pushout over a monomorphism.

4 Constraint-Preserving Interaction Schemes

In this section, we develop our construction of constraint-preserving and -guaranteeing interaction schemes. We prove the construction to be well-defined, i.e., it results in a family of multi-rules, and present sufficient conditions for the desired preservation property of the arising interaction schemes.

The central idea of the construction of a constraint-preserving interaction scheme is to identify each way in which the application of a rule may introduce a violation of a given positive atomic constraint $c = \forall\,(P, \exists\,C)$. These are the different ways in which the rule can create a new occurrence of P. For each such situation we derive a multi-rule that additionally amends this new P to C. Applying the such arising interaction scheme instead of the original rule preserves the validity of the constraint (if the multi-rules themselves do not introduce new violations again). Slightly extending this strategy additionally provides rules that repair already existing violations of the constraint, i.e., enable to not only preserve but guarantee consistency.

We first introduce *compatible rule-constraint intersections* to classify the ways in which a rule $r : L \hookrightarrow R$ may introduce an image of the premise P of a constraint $\forall\,(P, \exists\,C)$. Assume a transformation $G \Rightarrow_{r,m} H$ to be given; the co-match is an embedding of R in H. If there is an image of P in H, it intersects with that image of R in H (maybe emptily). This intersection restricts to an intersection between the images of L and P in H as well. Thus, we introduce compatible intersections as monomorphisms $\iota : D \hookrightarrow E$ where E is a subobject of R and P and D is a subobject of L and P in a compatible way. The intuition is that D is that part of P already matched by the rule and $E \setminus D$ is the part the rule's application created anew. This suggests that if ι is an isomorphism, a situation is captured where the rule only matches some part of an image of P

without creating a part of it (compare Lemma 2). We only need to consider those intersections that actually stem from applications of rules. Thus, calculating all such intersections gives all conceivable different ways in which an application of the rule might introduce a new subobject (isomorphic to) P.

Definition 5 (Compatible rule-constraint intersection). *Let $\mathcal{C}$ be an adhesive category, $r : L \hookrightarrow R$ a monotonic rule, and $c = \forall\,(P, \exists\, C)$ a positive atomic constraint in $\mathcal{C}$. A* compatible rule-constraint intersection *for r and c is a pair of spans of monomorphisms $d : L \overset{d_L}{\hookleftarrow} D \overset{d_P}{\hookrightarrow} P$ and $e : R \overset{e_R}{\hookleftarrow} E \overset{e_P}{\hookrightarrow} P$ with monomorphism $\iota : D \hookrightarrow E$ such that (compare Fig. 10)*

1. *$d_P = e_P \circ \iota$,*
2. *the square (1) is a pullback, i.e., $\iota : D \hookrightarrow E$ is a subrule of r, and*
3. *there exists an object H and monomorphisms $n_1 : R \hookrightarrow H$ and $m_2 : P \hookrightarrow H$ such that the arising square (2) is a pullback and a pushout complement G for $n_1 \circ r$ exists (i.e., the rule r^{-1} is applicable at H with match n_1).*

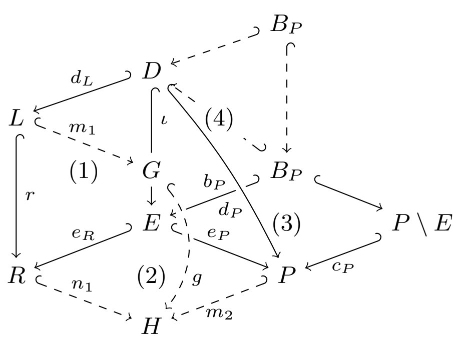


Fig. 10. Compatible intersections

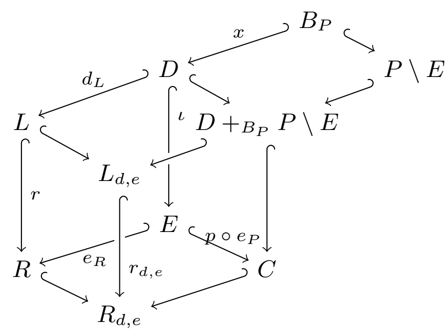


Fig. 11. Construction of the multi-rule $r_{d,e}$

Example 1. The uppermost diagram in Fig. 12 depicts a compatible rule-constraint intersection for rule *createEntry* and the constraint *PropagationOfEntries*. It is the only compatible intersection for them where ι is not an isomorphism.

Only intersecting the node of type Entry from the rule's RHS with the one from the constraint's premise (inducing an empty intersection between the LHS and the premise as depicted in the second diagram in Fig. 12) leads to a pair of intersections that satisfy the first two properties of Definition 5 but not the third one: It is not possible that an application of *createEntry* only creates the node of type Entry of an occurrence of the premise but not its corresponding incoming edge of type ownedEntry as this would imply that the edge existed without a target node before the application of the rule *createEntry* which is not allowed in a graph. This situation is depicted as last diagram in Fig. 12.

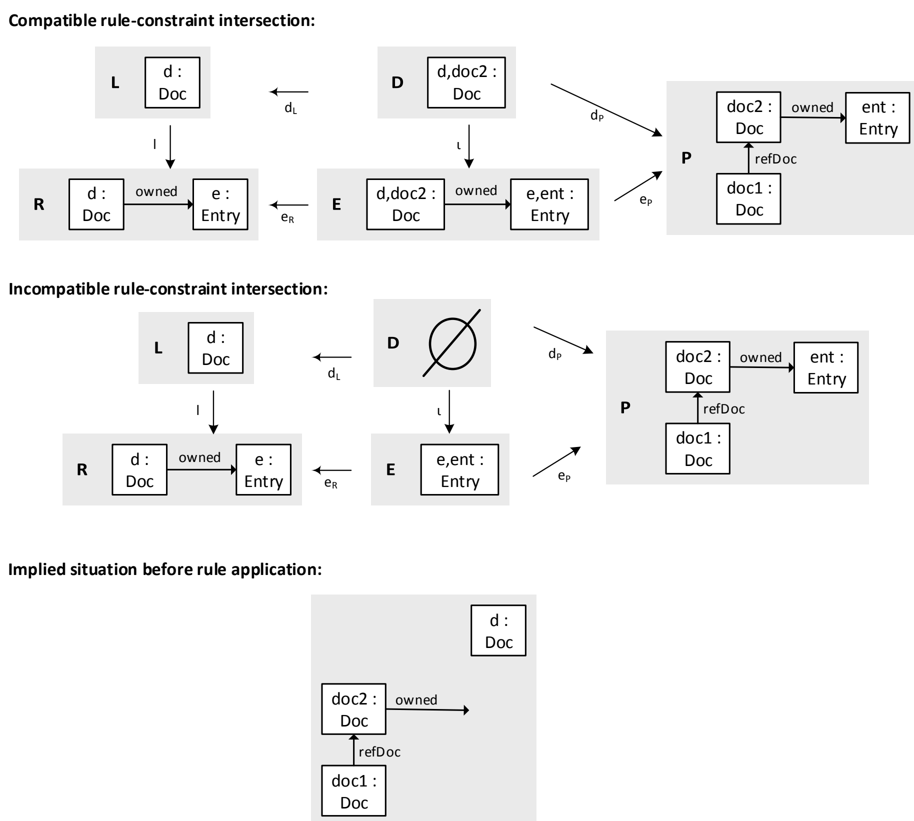


Fig. 12. A compatible and an incompatible rule-constraint intersection for rule *createEntry* and constraint *PropagationOfEntries* (node names indicate the morphisms)

In the example above, the node doc2 is a (in fact the only) boundary element: When deleting E from P, this node needs to be added to the result again for it to become a graph (so that the edge of type refDoc is not dangling). This signifies that this node must have been existent before application of the rule and is one of the places (here even *the place*) at which E and $P \setminus E$ are glued to receive a complete copy of P. Hence, this node needs to be matched by the rule application to create P and, thus, is already part of D, the intersection of L and P. The next lemma states that this generalizes, i.e., the boundary object B_P is always a subobject of D and the existence of this inclusion morphism is vital for our construction.

Lemma 1 (Covering of boundary elements). *Let $\mathcal{C}$ be an adhesive category with initial pushouts, $r : L \hookrightarrow R$ a rule, $c = \forall\,(P, \exists\, p : P \hookrightarrow C)$ a constraint in $\mathcal{C}$, and $(d : L \hookleftarrow D \hookrightarrow P, e : R \hookleftarrow E \hookrightarrow P)$ with monomorphism $\iota : D \hookrightarrow E$ a compatible rule-constraint intersection. Let B_P be the boundary object of the initial pushout over $e_P : E \hookrightarrow P$. Then there exists a monomorphism $x : B_P \hookrightarrow D$ such that $\iota \circ x = b_P$. In particular, B_P is a pullback object for the monomorphisms $\iota : D \hookrightarrow E$ and $b_P : B_P \hookrightarrow E$ (compare Fig. 10).*

Given a compatible rule-constraint intersection (d, e), we use it to compute a multi-rule $r_{d,e}$. Its RHS is just the join of R and C along E, thus completing a newly created P to C. The multi-rule $r_{d,e}$ should match exactly in those cases where r creates a new P by adding $E \setminus D$ to some already existing structure. Thus, P has to exist except for the part $E \setminus D$ before rule application and the multi-rule needs to match that structure. This is achieved by joining L with the join of $P \setminus E$ and D along the boundary object B_P.

Construction 1 (Constraint-guaranteeing and constraint-preserving interaction scheme). *Let $\mathcal{C}$ be an adhesive category with initial pushouts, $r : L \hookrightarrow R$ a monotonic rule, and $c = \forall\,(P, \exists\, p : P \hookrightarrow C)$ a positive atomic constraint in $\mathcal{C}$. For each (up to isomorphism) compatible rule-constraint intersection $(d : L \xhookleftarrow{d_L} D \xhookrightarrow{d_P} P, e : R \xhookleftarrow{e_R} E \xhookrightarrow{e_P} P)$ with monomorphism $\iota : D \rightarrow E$ for r and c, compute the multi-rule $r_{d,e} : L_{d,e} \hookrightarrow R_{d,e}$ in the following way (compare Fig. 11):*

1. *Compute the LHS $L_{d,e}$ of $r_{d,e}$ as pushout of the morphisms $d_L \circ x : B_P \hookrightarrow L$ and $B_P \hookrightarrow P \setminus E$.*
2. *Compute the RHS $R_{d,e}$ of $r_{d,e}$ as pushout of the morphisms $e_R : E \hookrightarrow R$ and $p \circ e_P : E \hookrightarrow C$.*
3. *The morphism $r_{d,e} : L_{d,e} \hookrightarrow R_{d,e}$ is induced by the universal property of the pushout computing $L_{d,e}$.*

The constraint-guaranteeing interaction scheme for r with respect to c *consists of the arising multi-rules and is denoted with $is_{r,c}$. Its restriction $is^{p}_{r,c}$ consists of those multi-rules stemming from pairs of intersections (d, e) where $\iota : D \hookrightarrow E$ is* not *an isomorphism and is called* constraint-preserving interaction scheme*.*

Example 2. Given the rule *createEntry* and the constraint *PropagationOfEntries*, the constraint-preserving interaction scheme consists only of the kernel morphism embedding *createEntry* into the multi-rule *createEntry-multi*, depicted in Fig. 5, as the only possible choice for a compatible rule-constraint intersection, which is not an isomorphism, is the one given in Example 1.

The constraint-guaranteeing interaction scheme additionally contains the multi-rules depicted in Fig. 13. The intersections from which they are arising are given by the two isomorphisms from the empty graph to itself and from one node of type Doc to itself, respectively. The node of type Doc then might by identified with each of the two nodes of type Doc occurring in the constraint.

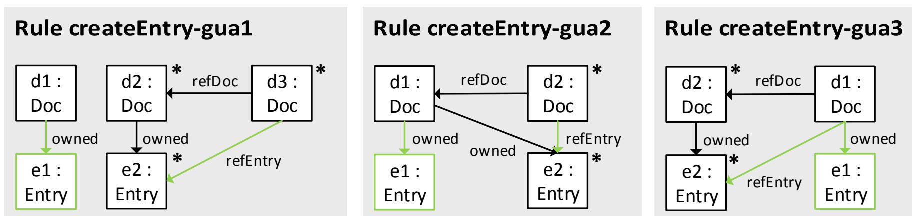


Fig. 13. Additional multi-rules contained in the constraint-guaranteeing interaction scheme of rule *createEntry* and constraint *PropagationOfEntries*

Remark 2. We introduced interaction schemes, as usual, as *finite sets* in Definition 3. For the construction of the amalgamated rule (Definition 3 and Remark 1) it is important that the set J is finite – otherwise the colimits do not need to exist. The most natural setting to guarantee that both the interaction schemes arising by the above construction and the number of matches in their maximal matchings are finite is that of finitary categories, i.e., categories where every object has only finitely many subobjects. Moreover, in finitary adhesive categories initial pushouts always exist [5, Fact 3.12] – thus they do not need to be required as additional precondition. But since, for the above construction, the number of compatible rule-constraint intersections is not relevant and the number of matches in a maximal matching might be finite even if the considered interaction scheme is not, we did not restrict ourselves to finitary categories.

Showing that this construction actually leads to multi-rules of the original rule exploits the van Kampen property of the cube in Fig. 11; in this way the right front face is a pullback.

Theorem 1 (Well-definedness of construction). *Let $\mathcal{C}$ be an adhesive category with initial pushouts, $r : L \hookrightarrow R$ a monotonic rule, and $c = \forall(P, \exists p : P \hookrightarrow C)$ a positive atomic constraint in $\mathcal{C}$. Then for every compatible rule-constraint intersection (d, e) the rule $r_{d,e} : L_{d,e} \hookrightarrow R_{d,e}$ as introduced in Construction 1 is a multi-rule of r and the morphism $r_{d,e}$ is a monomorphism.*

The next lemma states that compatible rule-constraint intersections are capable of distinguishing if an application of a rule r first enabled a matching of P in H or if this match already restricts to a match in G where $G \Rightarrow_r H$.

Lemma 2 (Classification of compatible rule-constraint intersections). *Let $\mathcal{C}$ be an adhesive category with initial pushouts, $r : L \hookrightarrow R$ a monotonic rule, $c = \forall(P, \exists p : P \hookrightarrow C)$ a positive atomic constraint, and $(d : L \overset{d_L}{\hookleftarrow} D \overset{d_P}{\hookrightarrow} P, e : R \overset{e_R}{\hookleftarrow} E \overset{e_P}{\hookrightarrow} P)$ with morphism $\iota : D \hookrightarrow E$ a compatible rule-constraint intersection for the rule r and the premise P of the constraint c. Let H be an object with monomorphisms $n_1 : R \hookrightarrow H$ and $m_2 : P \hookrightarrow H$ such that the induced square is a pullback and a pushout complement G for $n_1 \circ r$ exists. Then P already embeds into G in a way compatible to its embedding in H if and only if $\iota : D \hookrightarrow E$ is an isomorphism.*

Intuitively, applying the original rule r and the complement rules of the constructed interactions scheme afterwards, these complement rules supplement each at that moment existing image of the premise P of the constraint with an image of its conclusion C. But it may happen that the application of such complement rules introduces new occurrences of P which, by nature of our construction, are not supplemented. Consequently, it may happen that after applying the interaction scheme the constraint is violated, nonetheless. The next definition serves to be able to exclude situations like that rigorously and assumes familiarity with the concept of parallel independence [4]. The subsequent theorem states that this condition of independence is a sufficient (not necessary) condition for the constructed interaction schemes to preserve (guarantee) consistency with respect to a constraint. Its proof uses this independence to show that occurrences of P must stem from application of r or have already existed before. Than one needs to check that each such situation is covered by a multi-rule which is done by showing the arising intersections to be compatible (see Definition 5).

Definition 6 (Complementable). *Given a monotonic rule $r : L \hookrightarrow R$ and a positive atomic constraint $c = \forall(P, \exists C)$, r is called* (strongly) complementable *with respect to c if for every multi-rule $r_{d,e} : L_{d,e} \hookrightarrow R_{d,e}$ from the interaction scheme $is^p_{r,c}$ ($is_{r,c}$) the inverse of its complement rule and the constant rule $P \hookrightarrow P$ are parallel independent.*

Theorem 2 (Guarantee and preservation). *Let an adhesive category $\mathcal{C}$ with initial pushouts be given. Let $r : L \hookrightarrow R$ be a monotonic rule, $c = \forall(P, \exists p : P \hookrightarrow C)$ a positive atomic constraint, and $is_{r,c}$ and $is^p_{r,c}$ the constraint-guaranteeing and constraint-preserving interaction schemes from Construction 1, respectively. Let m_1 be a match for r at an arbitrary object G with maximal matchings J and J^p for $is_{r,c}$ and $is^p_{r,c}$ extending m_1.*

1. *If r is strongly complementable with respect to c and J is finite, then the object H arising by application of $is_{r,c}$ with that matching satisfies the constraint c.*
2. *If r is complementable with respect to c and J^p is finite then the object H arising by application of $is_{r,c}$ with that matching satisfies the constraint c.*

5 Prospects and Future Work

This paper focuses on presenting the general idea behind the construction of constraint-preserving interaction schemes and proving their fundamental property. However, there are a lot of possible refinements and interesting future work:

In adhesive categories with strict initial object $\emptyset$, a constraint of the form $\exists\, C$, which requires C to be a subobject of G to be valid for an object G, is semantically equivalent to $\forall\,(\emptyset, \exists\, \emptyset \hookrightarrow C)$, i.e., preservation or guarantee of those constraints is dealt with in our framework as well.

The general idea presented in this paper may be used for deleting rules $r = (L \hookleftarrow K \hookrightarrow R)$ as long as it is not possible that an application of that rule deletes an occurrence C of the relevant constraint $\forall\,(P, \exists\, C)$. The construction stays the same, in principle, with K playing the role of L. Only every arising multi-rule gets equipped with left-hand side $L_{d,e}$ arising as pushout of $K \hookrightarrow L$ and $K \hookrightarrow K_{d,e}$.

The precondition of complementability that comes with Theorem 2 is a severe restriction. There are examples, where this precondition is not met, but where recursively repeating our construction for the computed multi-rules again terminates (with no new multi-rules arising anymore) and where these multi-rules of multi-rules can be equivalently expressed as simple multi-rules of the original rule and the constraint is preserved by that extended interaction scheme. We give an example for this, namely a constraint requiring transitive closure over edges of a certain type, in Appendix C. Investigating this possibility in more detail is future work.

We plan to further integrate application conditions into our approach. Another topic for future research is to support preservation or guarantee of more than one constraint simultaneously. It would be especially interesting to combine support for positive atomic constraints and negative ones of the form $\neg\exists\, C$, forbidding C to be an subobject of G to be valid for G. Our vision is to characterize those combinations of rules and constraints such that it is possible to preserve consistency with the whole set of constraints by incorporating the negative ones as application conditions into the rules and then constructing the preserving interaction schemes to support the positive atomic ones.

6 Related Work

As already discussed in Sect. 1 and 2, our work can be seen as a continuation of [12] and a supplement to [7]. To the best of our knowledge, we are the first to suggest an automatable construction to alter a rule into an interaction scheme to preserve consistency with respect to a given constraint.

We are aware of two works, both with tool support, allowing for automatic construction of interaction schemes in the context of EMF model transformation, but none of them explicitly aims at preservation of constraints. In [1], Alshanqiti et al. derive visual contracts from Java programs. They monitor the execution of programs and generate transformation rules describing the behavior of the program. The process supports the generation of interaction schemes. In contrast to our work aiming at preservation of constraints, their derived rules provide an abstract and visual representation of observed behavior.

Kehrer et al. allow a user to specify examples of how a transformation should (or should not) act and from that infer a rule subsuming the provided examples [9]. Their tool is also able to derive interaction schemes. As above, the aim is not to preserve correctness of an explicitly given constraint, and a generated interaction scheme may or may not happen to do so (depending on the quality of the input of the user) or serve a completely different purpose.

There are two works, [14,15] and [8], that derive repair programs from (graph) constraints. In both works, given a set of constraints, a set of rules and some control structure are derived such that applying the resulting program to an instance results in an instance satisfying those constraints. In neither case interaction schemes are derived, but by passing of parameters or marking and applying rules as often as possible, the same effects are achieved. The second work is done in the context of (labeled) graphs and supports quite a large class of graph constraints, though support for sets of constraints is limited. The first one is done in the context of EMF models and repairs violations of multiplicities while respecting the in-built constraints of EMF. Moreover, it is implemented [15].

In [2], Becker et al. propose a method to design consistency-preserving rule-based refactorings. Given such a refactoring and a constraint, an invariant checker provides the user with minimal counterexamples for situations in which the refactoring does not preserve consistency with respect to the constraint. The user may use this information to redesign the refactoring and iterate over this process. The class of supported constraints is slightly larger than ours (supporting also negative constraint). Refactorings are specified via graph transformation rules and though there is no explicit support for interaction schemes, it is possible to use marking and iterate the application of rules as often as possible, which can have the same effect. If a constraint-preserving specification of a refactoring is achieved, of course, depends on the user.

In [10], Kehrer et al. automatically derive edit rules from meta-models. Starting with atomic operations, they revise rules such that elementary consistency constraints, necessary to be met for a model to be opened in a typical editor, are preserved on application of those roles. When viewing lower bounds of multiplicities of containment edges as positive atomic constraints and applying our construction to these constraints and their elementary rules for node creation, we receive exactly the rules they are using, too. In this case, however, the resulting rules actually not are multi-rules since the LHSs of the resulting multi-rules are the LHSs of the original rules again.

7 Conclusion

In the setting of adhesive categories, given a positive atomic constraint and a monotonic rule, we presented a construction of a constraint-preserving (or -guaranteeing) interaction scheme. We showed that our construction is well-defined and gave (sufficient) conditions under which the constraint is preserved (or guaranteed) when applying the resulting interaction schemes. With this approach, we are aiming at being able to replace the computation of constraint-preserving (or -guaranteeing) application conditions for rules in a situation where this has the often undesired effect of severely restricting the applicability of the rule. With our approach we are already able to automate the construction of multi-rules that had to be developed by hand before in [12]. We pointed at promising directions of research to be able to generalize our construction, to overcome the precondition of complementability that was necessary for our main theorem to hold, and to increase the effectiveness of our method in practice.

Acknowledgments This work was partially funded by the German Research Foundation (DFG), projects "Triple Graph Grammars (TGG) 2.0" and "Generating Development Environments for Modeling Languages".

References

1. Alshanqiti, A., Heckel, R., Kehrer, T.: Inferring visual contracts from Java programs. Automated Software Engineering (Jul 2018)
2. Becker, B., Lambers, L., Dyck, J., Birth, S., Giese, H.: Iterative Development of Consistency-Preserving Rule-Based Refactorings. In: Cabot, J., Visser, E. (eds.) Theory and Practice of Model Transformations. pp. 123–137. Springer, Berlin (2011)
3. Biermann, E., Ermel, C., Taentzer, G.: Formal foundation of consistent EMF model transformations by algebraic graph transformation. Software & Systems Modeling **11**(2), 227–250 (2012)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science, Springer (2006)
5. Gabriel, K., Braatz, B., Ehrig, H., Golas, U.: Finitary $\mathcal{M}$-adhesive categories. Mathematical Structures in Computer Science **24**(4), 240403 (2014)
6. Golas, U., Habel, A., Ehrig, H.: Multi-amalgamation of rules with application conditions in $\mathcal{M}$-adhesive categories. Math. Struct. in Comp. Science **24** (2014)
7. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. in Comp. Science **19**, 245–296 (2009)
8. Habel, A., Sandmann, C.: Graph repair by graph programs. In: Kreowski, H.J. (ed.) Graph Computation Models (GCM 2018), Electronic Pre-Proceedings (2018), https://www.gcm2018.uni-bremen.de/assets/gcm_2018_paper_5.pdf
9. Kehrer, T., Alshanqiti, A., Heckel, R.: Automatic Inference of Rule-Based Specifications of Complex In-place Model Transformations. In: Guerra, E., van den Brand, M. (eds.) Theory and Practice of Model Transformation. pp. 92–107. Springer, Cham (2017)
10. Kehrer, T., Taentzer, G., Rindt, M., Kelter, U.: Automatically deriving the specification of model editing operations from meta-models. In: Van Gorp, P., Engels, G. (eds.) Theory and Practice of Model Transformations. pp. 173–188. Springer, Cham (2016)
11. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. Theoretical Informatics and Applications **39**(3), 511–545 (2005)

12. 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)
13. Nassar, N., Kosiol, J., Arendt, T., Taentzer, G.: OCL2AC. Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules. In: Lambers, L., Weber, J. (eds.) Proc. of ICGT 2018. pp. 171–177. Springer, Cham (2018)
14. Nassar, N., Kosiol, J., Radke, H.: Rule-based repair of emf models: Formalization and correctness proof. In: Corradini, A. (ed.) Graph Computation Models (GCM 2017), Electronic Pre-Proceedings (2017), pages.di.unipi.it/corradini/Workshops/GCM2017/papers/Nassar-Kosiol-Radke-GCM2017.pdf
15. Nassar, N., Radke, H., Arendt, T.: Rule-based repair of emf models: An automated interactive approach. In: Guerra, E., van den Brand, M. (eds.) Theory and Practice of Model Transformation. pp. 171–181. Springer, Cham (2017)
16. OMG: Object Constraint Language, http://www.omg.org/spec/OCL/
17. Radke, H., Arendt, T., Becker, J.S., Habel, A., Taentzer, G.: Translating essential OCL invariants to nested graph constraints for generating instances of meta-models. Science of Computer Programming **152**, 38–62 (2018)
18. Rensink, A.: Representing First-Order Logic Using Graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) Graph Transformations. pp. 319–335. Springer, Berlin (2004)
19. 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. pp. 151–163. Springer (1995)

A Adhesive Categories

In this section, we recall adhesive categories and some of their properties. 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 7 (Van Kampen square and adhesive category). *A pushout diagram as depicted in Fig. 14 is a* van Kampen square *if for every commutative cube over it (like depicted in Fig. 15) where the backfaces are pullbacks, the front faces are pullbacks iff the top face is a pushout.*

A category $\boldsymbol{C}$ *is called* adhesive *if*

1. $\boldsymbol{C}$ *has pushouts along monomorphisms (i.e., pushouts whenever at least one of the two morphisms* f *or* m *in Fig. 14 is a monomorphism),*
2. $\boldsymbol{C}$ *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 [11,4]. We will use (or assume implicitly) the following properties of adhesive categories:

Fact 1 (Properties of adhesive categories). *If* $\boldsymbol{C}$ *is an adhesive category, the following properties hold [11]:*

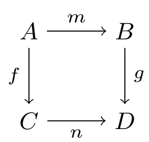


Fig. 14. A pushout square

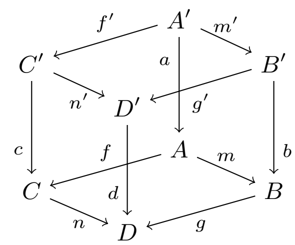
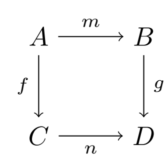


Fig. 15. Commutative cube over pushout square

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.*

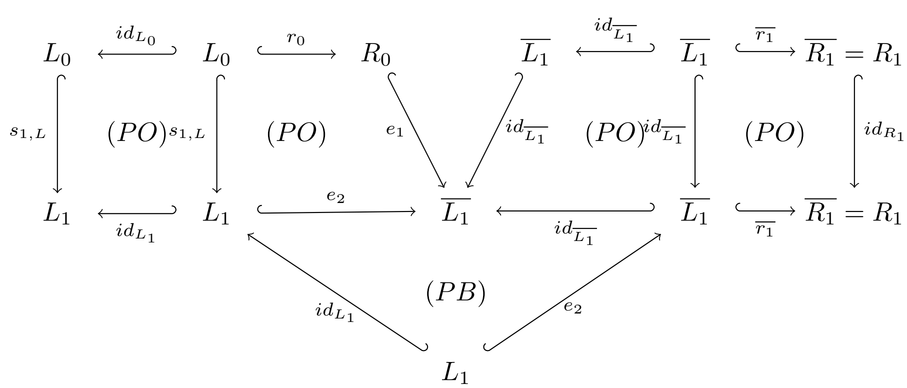


2. *If f is a monomorphism (compare the diagram above), pushout complements for $n \circ f$ are unique (up to isomorphism).*
3. *The subobjects of an object in an adhesive category form a distributive lattice. Given two subobjects $a, b : A, B \hookrightarrow C$ of an object C in category $\boldsymbol{C}$, 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.*

B Proofs

Proof (of Additional Lemma 1). First, in an adhesive category, the mediating morphism of the pushout of a pullback of two monomorphisms is a monomorphism again.

Secondly, the following diagram exhibits r_1 as E-concurrent rule of r_0 and $\overline{r_1}$ with E-dependency relation $\overline{L_1}$ (observe that the pair of morphisms $(e_1, id_{\overline{L_1}})$ is jointly epi since $id_{\overline{L_1}}$ is epi).



□

Proof (of Lemma 1). To show the existence of a monomorphism $x : B_P \hookrightarrow D$ with $\iota \circ x = b_P$, first compute the pushout H' of $e_R : E \hookrightarrow R$ and $e_P : E \hookrightarrow P$ (square (1) in Fig. 16). Since $\mathcal{C}$ is adhesive, the arising mediating map $h : H' \hookrightarrow H$, where H is given by definition of compatible intersections, is a monomorphism. Hence, by the Restriction Theorem [4, Theorem 6.18], the rule r^{-1} is applicable at the match $n_1' : R \hookrightarrow H''$ yielding an object G' (square (2)). By the closure property of initial pushouts [4, Lemma 6.5], (1)+(3) is an initial pushout over n_1'. Since (2) is another pushout over n_1', there exists a (unique) monomorphism $x' : B_P \hookrightarrow L$ such that $r \circ x' = e_R \circ b_P$ (and a (unique) monomorphism $y' : P \setminus E \hookrightarrow G'$ with the corresponding property). The universal property of the pullback (4) implies the existence of a morphism $x : B_P \hookrightarrow D$ with $\iota \circ x = b_P$ (and $d_L \circ x = x'$). Moreover, the morphism x is monic since x' is. □

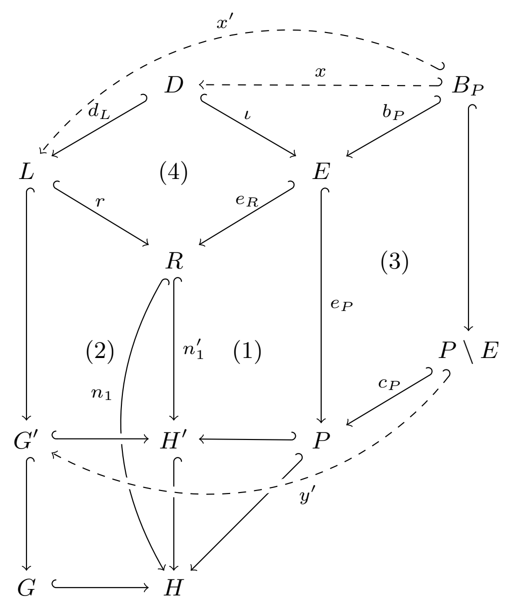


Fig. 16. Existence of embedding $x : B_P \hookrightarrow D$

Proof (of Theorem 1). First, the pushout of the morphisms $B_P \hookrightarrow P \setminus E$ and $d_L \circ x : B_P \hookrightarrow L$ computing the LHS $L_{d,e}$ of the multi-rule can be split into two pushouts (compare Fig. 17): The pushout of the morphisms $x : B_P \hookrightarrow D$ and $B_P \hookrightarrow P \setminus E$ gives rise to a morphism $D +_{B_P} P \setminus E \hookrightarrow L_{d,e}$ by its universal property. Since the outer and the right square in Fig. 17 are pushouts, by pushout decomposition, the arising square to the left is a pushout as well. Hence, $L_{d,e}$

can equivalently be computed as pushout of the morphisms $d_L : D \hookrightarrow L$ and $D \hookrightarrow D +_{B_P} P \setminus E$ which we will exploit in the following: We construct the cube depicted in Fig. 11. Since both the top and the bottom squares are pushouts, if both back faces are pullbacks, both front faces are pullbacks by the van Kampen property of pushouts in adhesive categories. The left face in the back is a pullback by assumption.

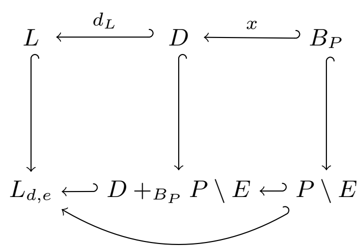


Fig. 17. Splitting the pushout that computes $L_{d,e}$

To show the right face in the back to be a pullback square, first compare Fig. 18. The morphism $D +_{B_P} P \setminus E \hookrightarrow P$ is induced by the universal property of the just constructed pushout at the top. The bottom square is the initial pushout over e_P. Since B_P is a subobject of D, the vertical square to the left in the back is a pullback. The vertical square to the right in the back is a pullback in every category. By the van Kampen property, both vertical squares in the front are pullbacks. In particular, D is a pullback object of $E \hookrightarrow P$ and $D +_{B_P} P \setminus E$.

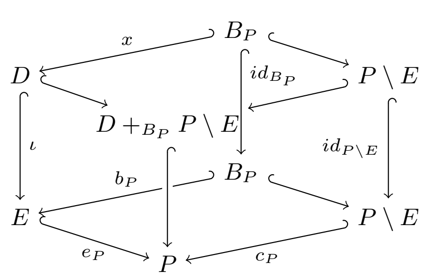


Fig. 18. Van Kampen square showing D to be a pullback object of E and $D +_{B_P} P \setminus E$ in P

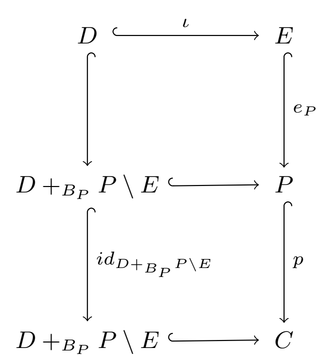


Fig. 19. D as pullback object of E and $D +_{B_P} P \setminus E$ in C

This pullback appears again as top square in Fig. 19. The bottom square is a pullback since $D +_{B_P} P \setminus E$ is a subobject of P. Hence, the outer square in

Fig. 19 – i.e., the square we need to show to be a pullback – is one by pullback composition.

As pushout of the pullback of two monomorphisms $L \hookrightarrow R_{d,e}$ and $D +_{B_P} P \setminus E \hookrightarrow R_{d,e}$, the intermediate morphism $r_{d,e} : L_{d,e} \hookrightarrow R_{d,e}$ is a monomorphism in an adhesive category. □

Proof (of Lemma 2). By definition, there exists (at least) one object H with monomorphisms $n_1 : R \hookrightarrow H$ and $m_2 : P \hookrightarrow H$ such that the induced square is a pullback and a pushout complement G for $n_1 \circ r$ exists. Without loss of generality, the pullback square is also a pushout. Otherwise one builds the pushout of the pullback. In an adhesive category, this is a pullback as well, and, by the Restriction Theorem [4, Theorem 6.18], a pushout complement still exist.

Let $\iota : D \hookrightarrow E$ be an isomorphism. Building the pullback of g and m_2 leads to an object X. Pulling back the pullback diagram along the morphism $n_1 : R \hookrightarrow H$ leads to the cube displayed in Fig. 20: Since pushouts along monomorphisms are pullbacks in adhesive categories, the pullbacks of n_1 and g or n_1 and m_2, respectively, are given by the pushout diagrams existing by assumption. Then, D is the pullback of r and e_R by assumption. The resulting cube has pullbacks as bottom and top faces and pushouts as front faces. Hence, by the Cube pushout-pullback lemma [4, Theorem 4.26], the backfaces are pushouts, too. Since ι is an isomorphism and pushouts along isomorphisms result in isomorphisms, the morphism $x_P : X \hookrightarrow P$ is an isomorphism. Hence, the morphism $m'_2 := x_G \circ x_P^{-1} : P \hookrightarrow G$ is a monomorphism with $m_2 = g \circ m'_2 \circ x_P^{-1}$.

If, on the other hand, there is a monomorphism $m'_2 : P \hookrightarrow G$ with $m_2 = g \circ m'_2$, the diagram at the bottom of the cube in Fig. 21 is a pullback. Pulling this pullback square again back along $n_1 : R \hookrightarrow H$ results in the cube displayed in Fig. 21 with the square in the right back being a pushout square, analogously to the argument above. Since a pushout along a monomorphism is a pullback in an adhesive category and pullbacks of isomorphisms are isomorphisms, the morphism $\iota : D \hookrightarrow E$ is one. □

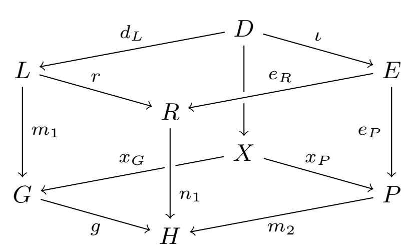


Fig. 20. Cube showing P to be subobject of G

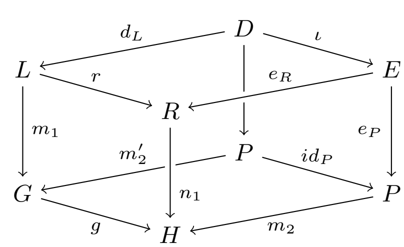
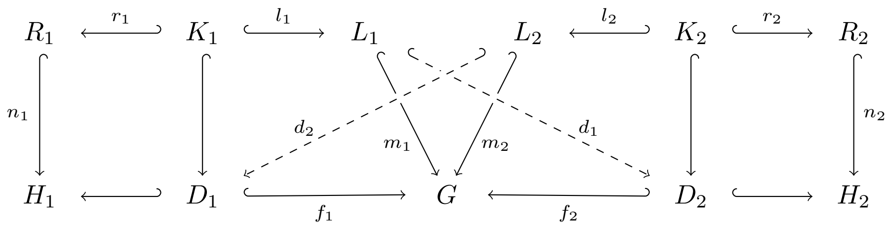


Fig. 21. Cube showing $\iota : D \hookrightarrow E$ to be an isomorphism

Before the next proof, we shortly recall the notion of parallel independence.

Additional Definition 2 (Parallel independence). *In an adhesive category $\mathcal{C}$, given two rules $p_i = (L_i \overset{l_i}{\hookleftarrow} K_i \overset{r_i}{\hookrightarrow} R_i)$ with $i = 1, 2$ two direct transformations $G \Rightarrow_{p_1,m_1} H_1$ and $G \Rightarrow_{p_2,m_2} H_2$ via those rules are* parallel independent *if there exist two morphisms $d_1 : L_1 \to D_2$ and $d_2 : L_2 \to D_1$ as depicted below such that $m_1 = f_2 \circ d_1$ and $m_2 = f_1 \circ d_2$. The rules p_1 and p_2 are* parallel independent *if every pair of transformations $H_1 \underset{p_1}{\Leftarrow} G \underset{p_2}{\Rightarrow} H_2$ is.*



Proof (of Theorem 2). Let an object G and a match m_1 for r be given such that the maximal matching J for $is_{r,c}$ (or $is^p_{r,c}$) extending this match is finite. Let H be the object resulting from application of $is_{r,c}$ (or $is^p_{r,c}$) at J and let $m_2 : P \hookrightarrow H$ be an arbitrary match for P in H. We need to show the existence of a monomorphism $q : C \hookrightarrow H$ such that $q \circ p = m_2$.

Applying $is_{r,c}$ (or $is^p_{r,c}$) at J, by [6, Fact 5.8 and Theorem 5.9] (the Multi-Amalgamation Theorem), inductively splits into a finite sequence $G \Rightarrow_{r,m_1} H' \Rightarrow \cdots \Rightarrow H$ of direct transformations, where the second part consists of the applications of the relevant complement rules of the multi-rules. Because of the (strong) complementability, these complement rules are in parallel independence with the constant rule $id_P : P \hookrightarrow P$. Thus, by repeated application of the Local Church-Rosser Theorem [4, Theorem 5.12], the match m_2 of P in H restricts to a match m_2' of P in H'. We show that this P was supplemented to an image of C by application of a suitable multi-rule (or such an image of C already existed). We do this in two steps, namely in showing a suitable multi-rule to exist by presenting the according compatible rule-constraint intersection and showing a match for its complement rule to exist in H' (i.e., it was actually applied to supplement P to C).

First, given such a monomorphism $m_2' : P \hookrightarrow H'$, we compute the pullback (2) of m_2' and the comatch n_1 of R in H' (compare Fig. 22) and subsequently the pullback (1) of r and e_R. By construction, the resulting pair of spans $(d : L \overset{d_L}{\hookleftarrow} D \overset{e_P \circ \iota}{\hookrightarrow} P, e : R \overset{e_R}{\hookleftarrow} E \overset{e_P}{\hookrightarrow} P)$ is a compatible rule-constraint intersection for r and c.

Secondly, if (d, e) is a compatible rule-constraint intersection, an according multi-rule $r_{d,e}$ for r exists. Let $\overline{L_{d,e}}$ be the LHS of its complement rule (with respect to r). The proof of Lemma 1 showed also a monomorphism $y' : P \setminus E \hookrightarrow G$ to exist in such a situation, since (3) is an initial pushout; hence, $g \circ y' : P \setminus E \hookrightarrow H$ is a monomorphism to H. Then, iteratively using the universal property of the pushouts defining $\underline{D +_{B_P} P \setminus E}$, $L_{d,e}$, and $\overline{L_{d,e}}$, respectively, results in a monomorphism $m_1^* : \overline{L_{d,e}} \hookrightarrow H'$ such that $n_1 \circ r = m_1^* \circ u_{d,e} = g \circ m_1$, i.e., a

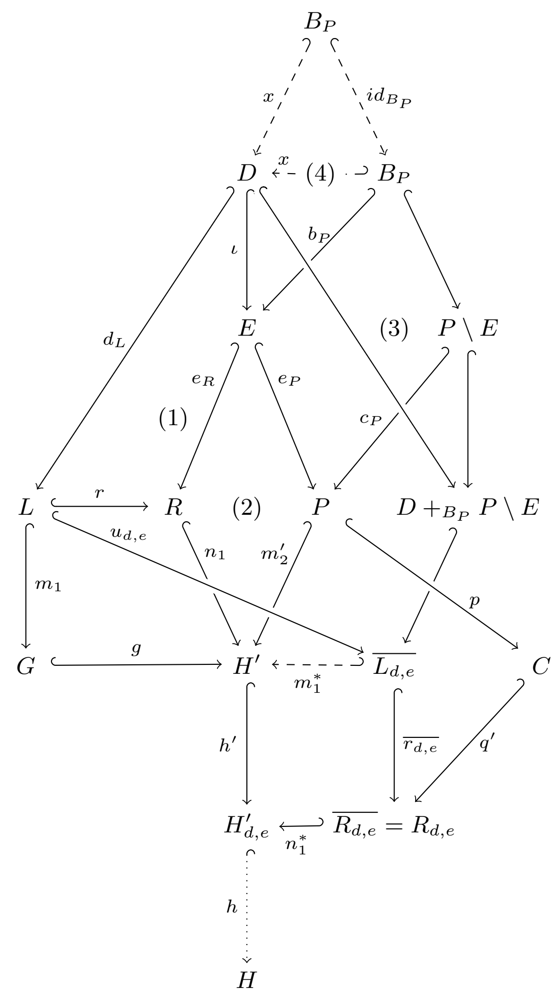


Fig. 22. Proving guarantee or preservation

match for the complement rule in H' (and, simultaneously, in a match for $L_{d,e}$ in G). Hence, the complement rule $\overline{r_{d,e}}$ with respect to r is applicable to H' and the corresponding match for $L_{d,e}$ in G is part of the maximal matching J. Since all the applications of complement rules constituting the sequence of direct transformations $H' \Rightarrow \cdots \Rightarrow H$ are pairwise parallelly independent [6, Fact 5.8], this sequence is of the form $H' \Rightarrow_{\overline{r_{d,e}},m_1^*} H'_{d,e} \Rightarrow \cdots \Rightarrow H$ without loss of generality and the original match m_2 of P in H decomposes as $m_2 = h \circ h' \circ m_2'$. Finally,

$$\begin{aligned} m_2 &= h \circ h' \circ m_2' \\ &= h \circ n_1^* \circ q' \circ p \end{aligned}$$

which means that $q \coloneqq h \circ n_1^* \circ q'$ is the desired monomorphism, i.e., the image of C created by application of $r_{d,e}$ supplemented the image of P correctly.

The only difference between the two statements concering the interaction schemes $is_{r,c}$ and $is^p_{r,c}$ is the existence of multi-rules $r_{d,e}$ where $\iota : D \hookrightarrow E$ is an isomorphism. These are not part of $is^p_{r,c}$ and are, consequently, not applied when applying that interaction scheme. But, by Lemma 2, if ι is an isomorphism, the image of P in H' given by m_2' restricts to an image in G. And, by assumption, $G \models c$ in the case for preservation, thus an according monomorphism $C \hookrightarrow G$ exists and obviously extends to the required monomorphism $q : C \hookrightarrow H$. $\square$

C Additional Example

The following example shows how repeating our construction can result in constraint-preserving interaction schemes, even if the considered rule is not strongly complementable with respect to the given constraint, i.e., in situations where the derived multi-rules might create new images for P themselves.

Additional Example 1. *As a second constraint over the meta-model from Fig. 1 consider the constraint displayed in Fig. 23. It requires the edges of type* refDoc *to be transitively closed.*

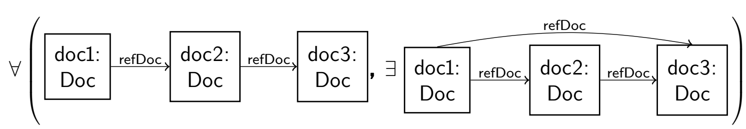


Fig. 23. Constraint *TransitiveClosure*

Calculating the constraint-preserving interaction scheme for the constraint TransitiveClosure *and the rule* insertReference *(see Fig. 2) results in the two multi-rules displayed in Figs. 24 and 25. They insert additional references from the source-*Doc *of the inserted reference the to all* Docs *referenced by the target-*Doc *or from all* Docs *referencing the source-*Doc *of the inserted reference to the*

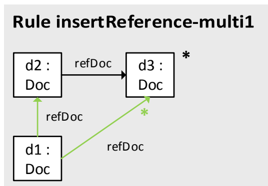


Fig. 24. Multi-rule inserting references to referenced Docs

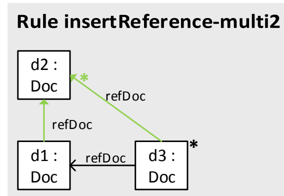


Fig. 25. Multi-rule inserting references from referencing Docs

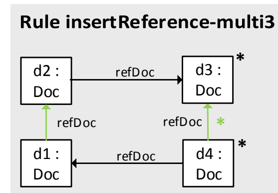


Fig. 26. Additional multi-rule inserting references from referencing to referenced Docs

target-Doc, respectively. This interaction scheme is not yet consistency-preserving, as an application of the multi-rules may create new situations where for a chain of two edges of type refDoc the required third one does not exist.

The situation displayed in Fig. 27 serves as an example. Applying the interaction scheme consisting of the multi-rules insertReference-multi1 *and* insertReference-multi2 *with the kernel rule* insertReference *matching the nodes d1 and d2 results in the instance displayed in Fig. 28. The multi-rules additionally connect d3 and d2 and d1 and d4. This results in the need to also connect the nodes d3 and d4 for which no multi-rule is available.*

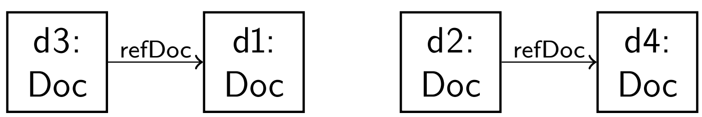


Fig. 27. Instance showing the derived interaction scheme to not be constraint-preserving

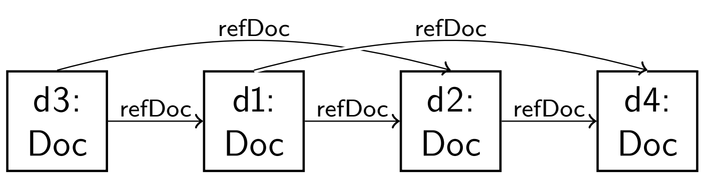


Fig. 28. Instance after application violating the constraint *TransitiveClosure*

Repeating our construction of constraint-preserving interaction schemes with the complement rules of the multi-rules insertReference-multi1 *and* insertReference-multi2 *results in a multi-rule of a multi-rule (only one, since both complement rules coincide). In that case this multi-rule of a multi-rule can be equivalently restricted to be a simple multi-rule of the original rule* insertReference*. It is the* *multi-rule displayed in Fig. 26. Further continuing the process does not result in structurally new multi-rules so it terminates. Applying the interaction scheme consisting of all three multi-rules instead results in the valid instance depicted in Fig. 29.*

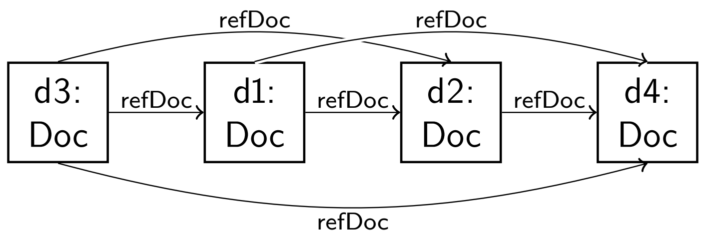


Fig. 29. Instance after application of the extended interaction scheme satisfying the constraint *TransitiveClosure*

It is not difficult to prove that interaction scheme to be constraint-preserving. However, more generally identifying rules and constraints such that iterating our proposed construction terminates and yields a constraint-preserving interaction scheme is future work.